



Microprocessor & Computer Architecture (μ pCA)

UE19CS252

V R BADRI PRASAD

Department of
Computer Science and Engineering



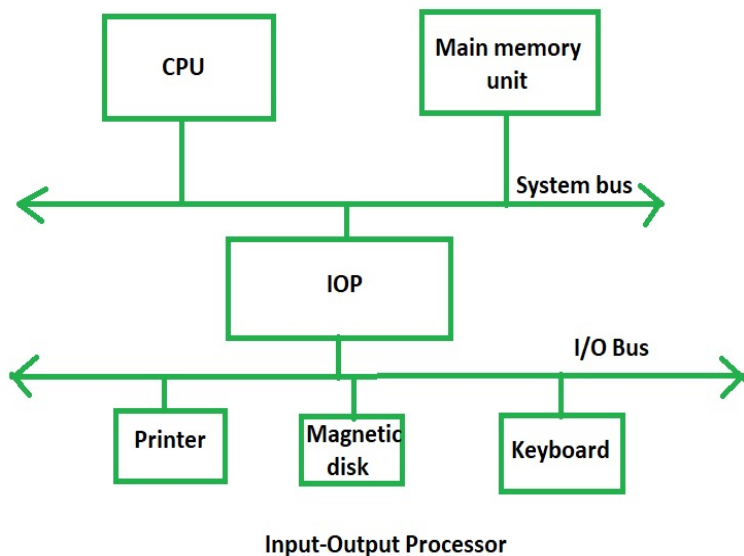
I/O and Bus Architecture

V R BADRI PRASAD

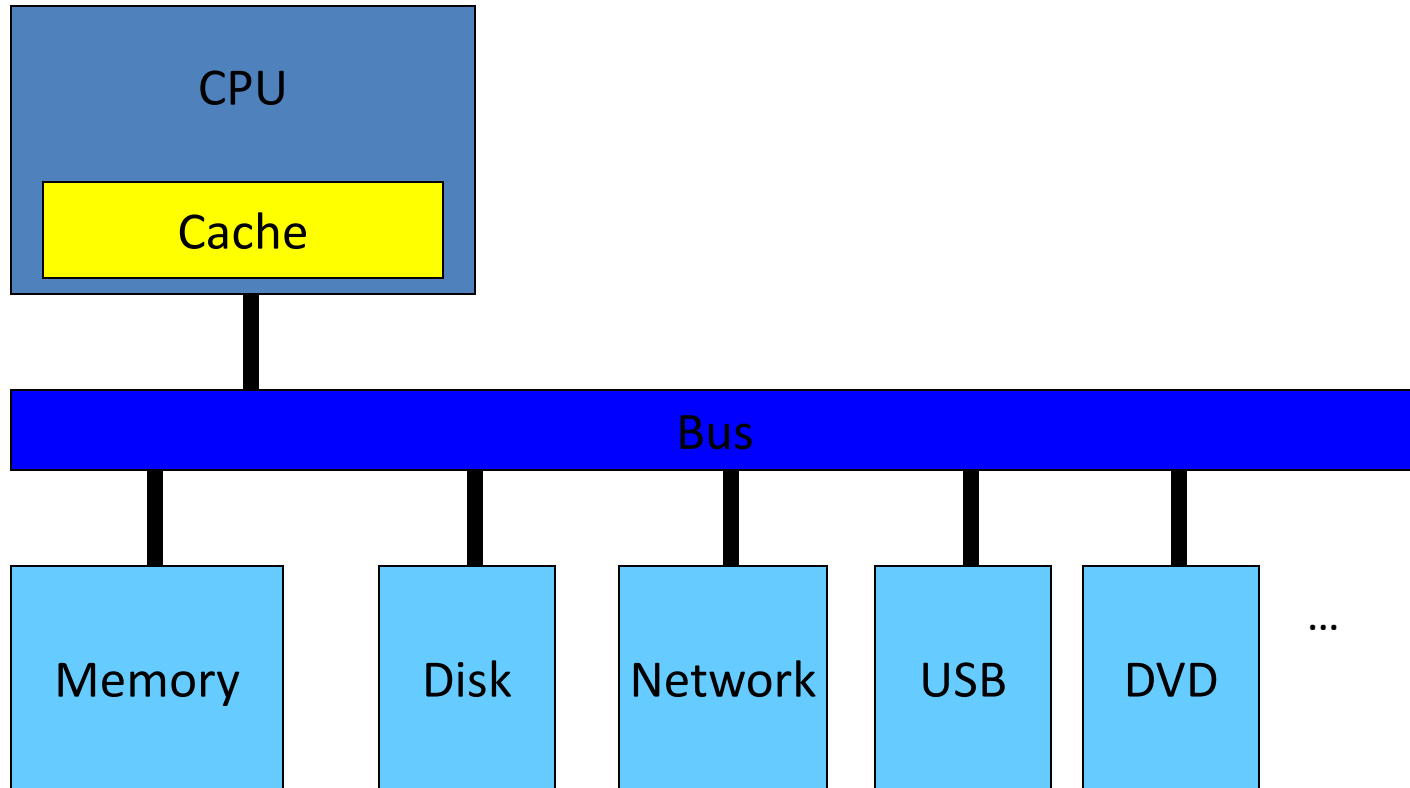
Department of
Computer Science and Engineering

Accessing I/O Devices - Introduction

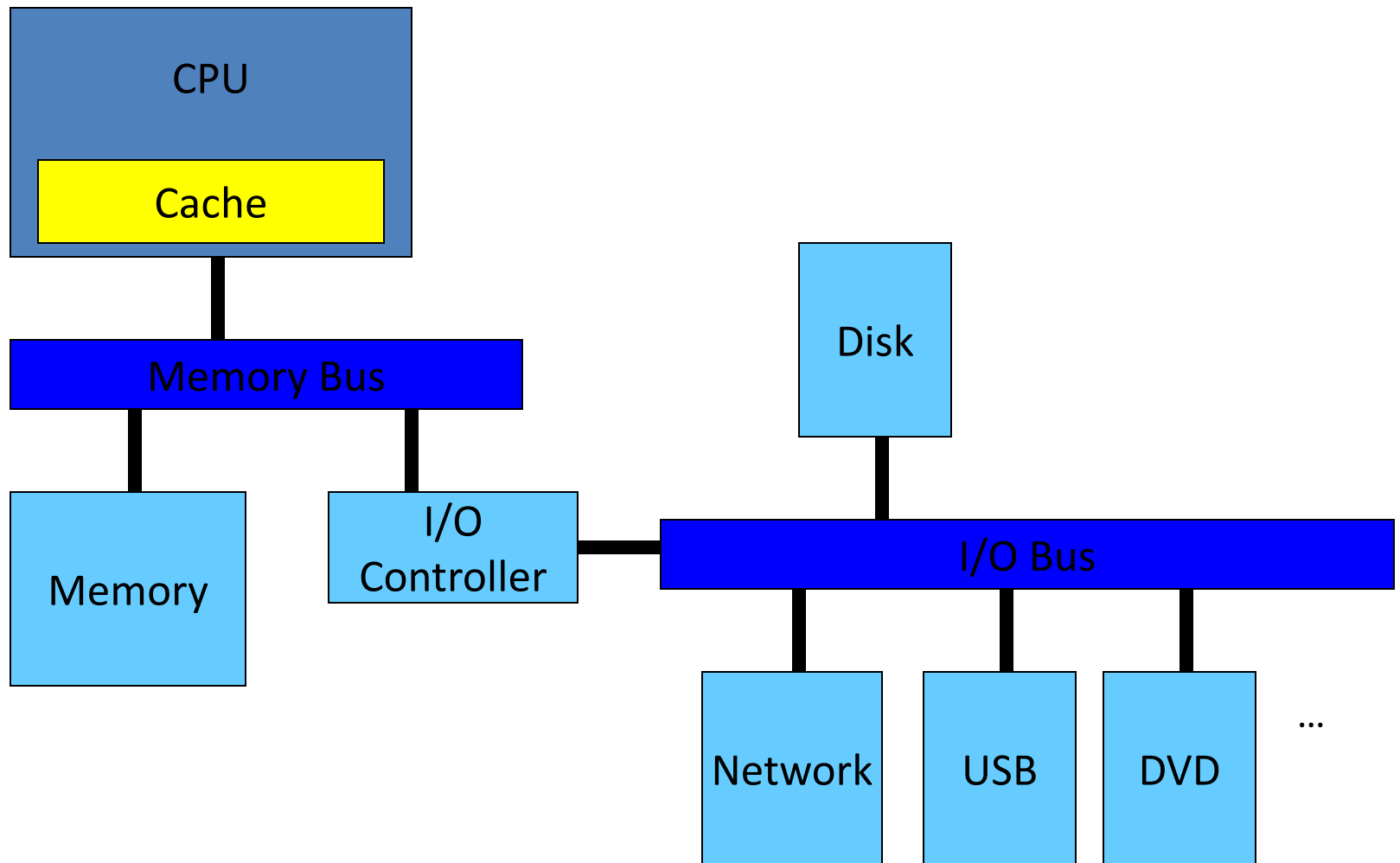
- The method that is used to transfer information between internal storage and external I/O devices is known as I/O interface.
- A typical link between the processor , memory and several peripherals.
- These devices work at varying speeds.
 - Ex: Monitors, Mouse, Keyboards, Video cameras, etc.,
 - Data transfer rate can either be regular or irregular.



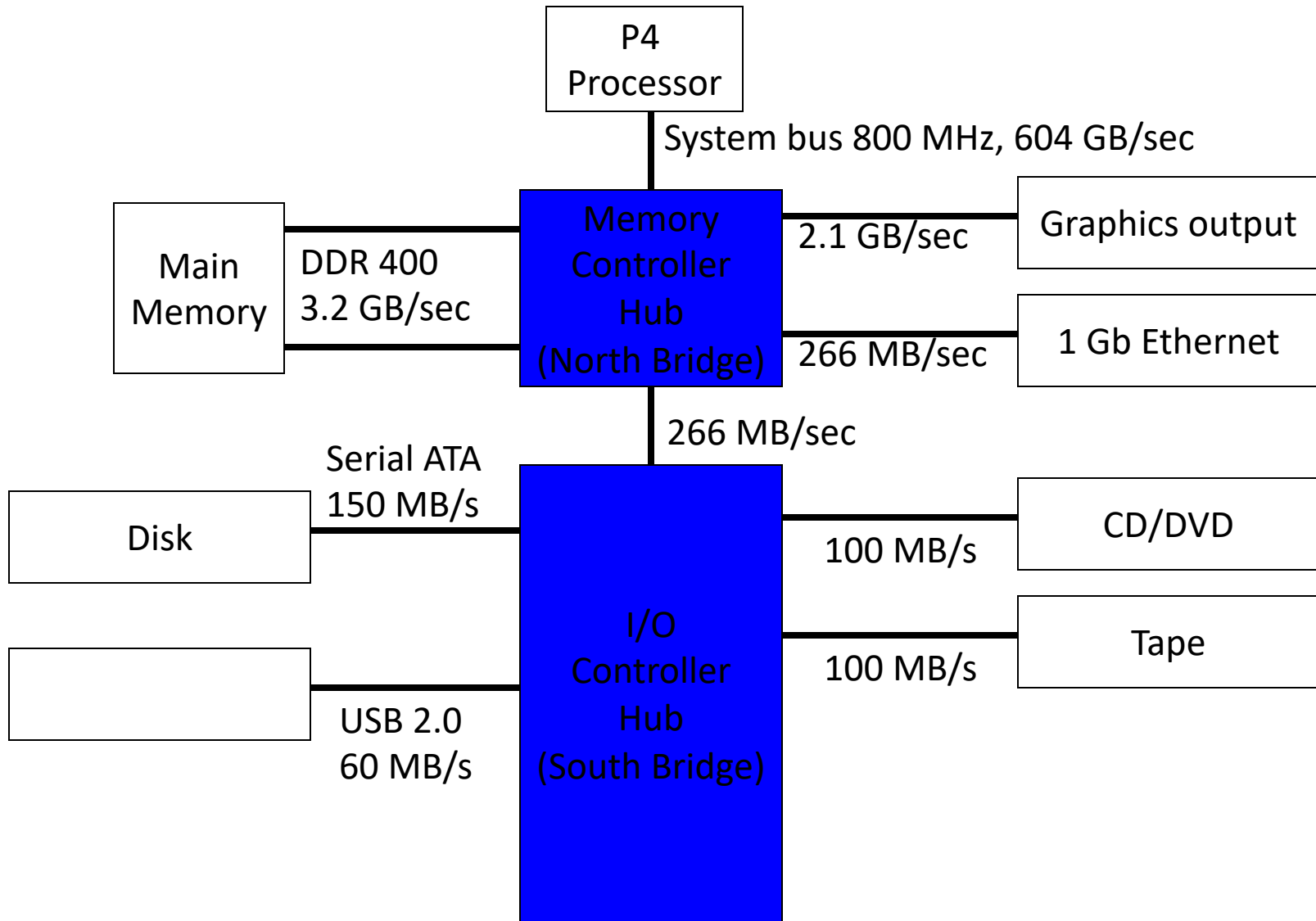
Input/Output



I/O Hierarchy

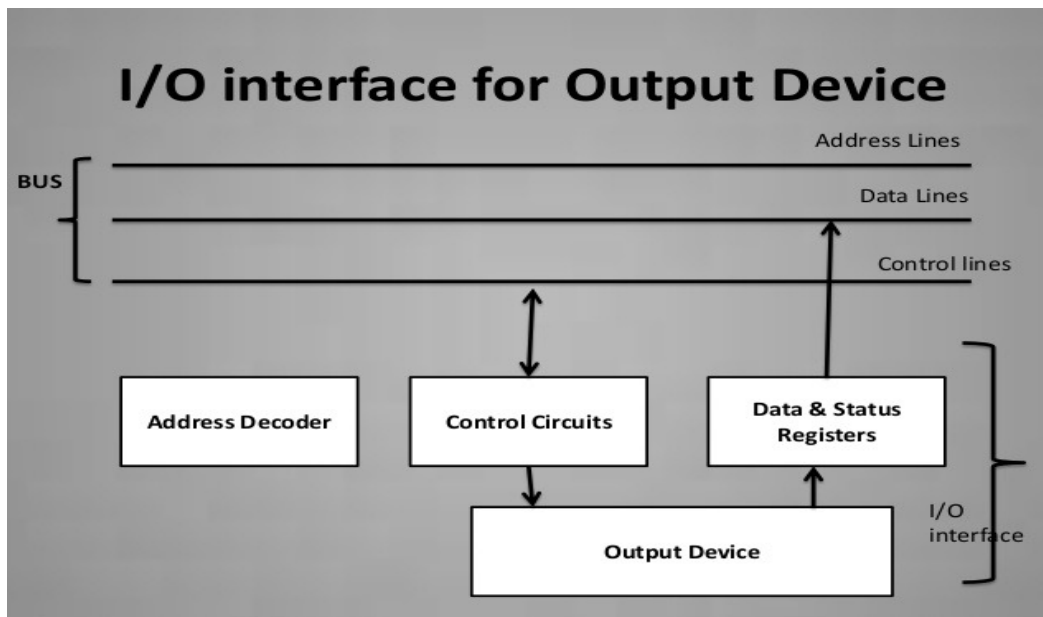


Intel Example



Accessing I/O Devices - Introduction

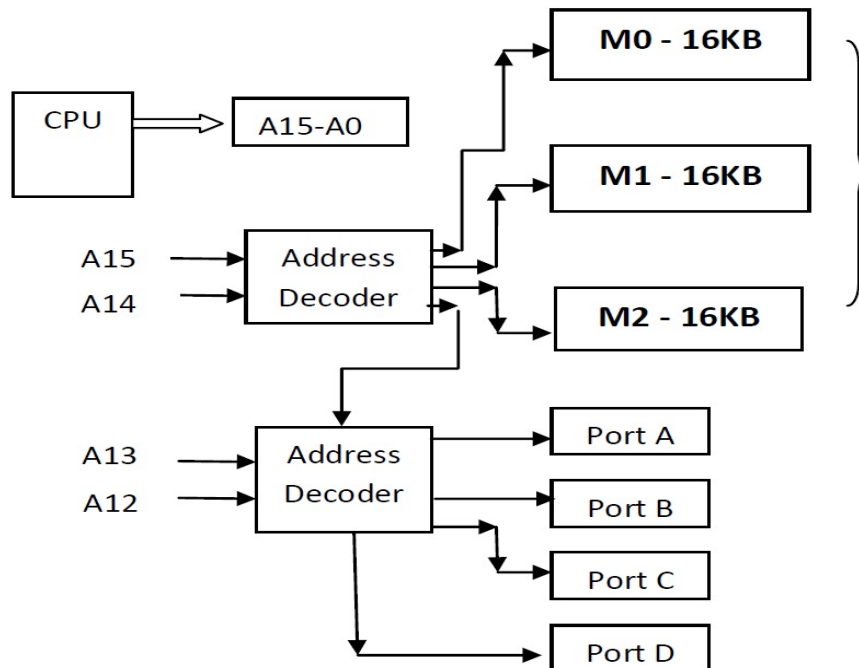
- There exists special hardware components between CPU and peripherals to supervise and synchronize all the input and output transfers that are called interface units.
- The I/O **Bus** consists of data lines, address lines and control lines.
- The I/O **bus** from the processor is attached to all peripherals interface.
- To communicate with a particular device, the processor places a device address on address lines.



Accessing I/O Devices :

Memory Mapped device interface

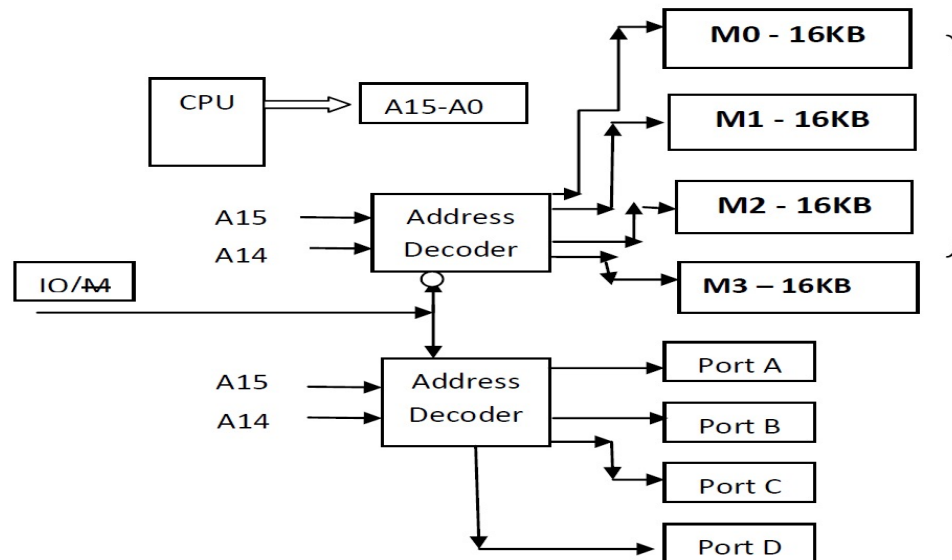
- **Memory Mapped I/O device interface:**
 - Same address decoder selects memory and I/O ports.
 - Some memory address space is occupied by the I/O devices.
 - All data transfer instructions to / from memory can be used to transfer to/from I/O devices.
 - Processor need not have separate instructions for I/O.



Accessing I/O Devices :

I/O Mapped device interface

- I/O Mapped I/O device interface:
 - Separate instructions for I/O data transfer [IN / OUT].
 - Processor signal identifies whether a generated address refers to memory or I/O device.
 - Separate address decoder for selecting memory and I/O ports.
 - Complete memory address space can be utilized.



Accessing I/O Devices – Data Transfer Techniques

- Data transfer to and from the peripherals may be done in any of these ways:

1. Programmed I/O: CPU executes a program and that transfers data between I/O device and Memory

- a. **Synchronous** : Fixed rate of transfer
- b. **Asynchronous** : Handshaking – polling for sending / receiving the data
- c. **Interrupt- Driven** : (next slide)

2. Direct memory access(DMA): An external controller directly transfers data between I/O device and Memory without CPU intervention.

WHAT IS INTERRUPT/EXCEPTION?

- Main ()

- {

- :

Can happen anytime
Depends on types of interrupts

- Doing something

- (e.g.

- browsing)

- :


- } ring



Phone rings

Phone rings



```
_isr() // Interrupt service routine
{
    
    some tasks (e.g. answer
        telephone)

} //when finished,
  //goes back to main
```

Accessing I/O Devices – Data Transfer Techniques

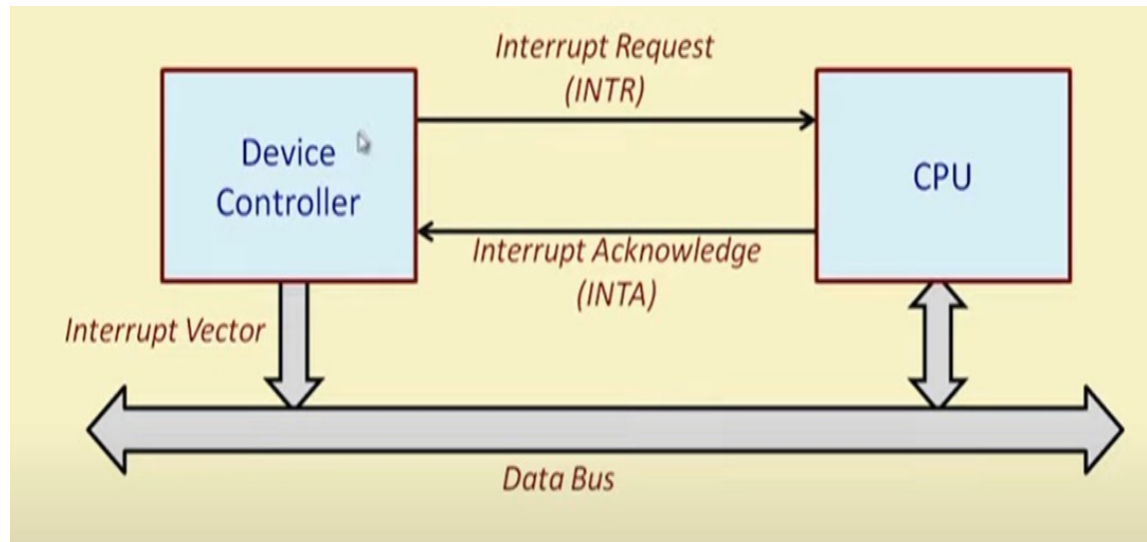
c. Interrupt- Driven:

- CPU initiates the data transfer and proceeds to perform some other task.
- When I/O module is ready for data transfer, it informs the CPU by activating a signal (**Interrupt request**).
- The CPU suspends the task it was doing, services the request from the device and return back to the task it was doing.
- **Advantages:**
 - CPU time is not wasted by polling the device
 - CPU time is required only during the data transfer plus some overheads for transferring and returning the control.

Accessing I/O Devices – Data Transfer Techniques

c. What happens when an interrupt request arrives?

- At the end of the execution of the current instruction, PC and the status register contents are saved in the stack automatically.
- Interrupt is acknowledged, interrupt vector is obtained based which the control transfers to the appropriate ISR.
- After handling the interrupt, the ISR executes a special instruction *return from interrupt(RTI)*.



Accessing I/O Devices – Data Transfer Techniques

c. What happens when an interrupt request arrives?

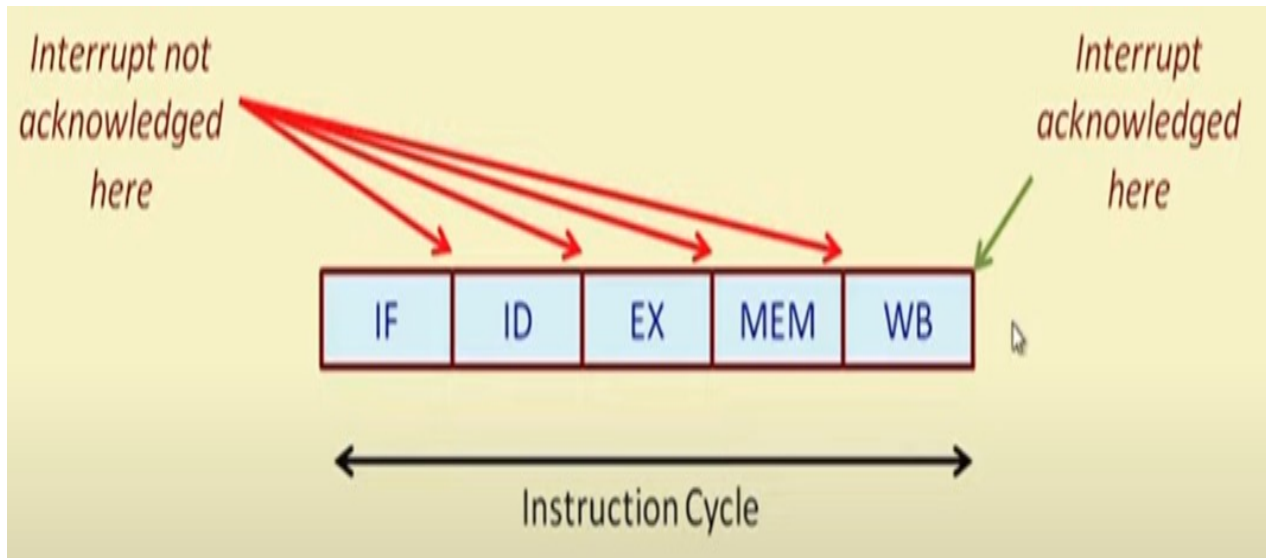
- At the end of the execution of the current instruction, PC and the status register contents are saved in the stack automatically.
- Interrupt is acknowledged, interrupt vector is obtained based which the control transfers to the appropriate ISR.
- After handling the interrupt, the ISR executes a special instruction *return from interrupt(RTI)*.

ARM Interrupt Vector Table

| | Address | Exception | Mode in Entry |
|---|------------|--------------------------|---------------|
| 1 | 0x00000000 | Reset | Supervisor |
| 2 | 0x00000004 | Undefined instruction | Undefined |
| 3 | 0x00000008 | Software Interrupt | Supervisor |
| 4 | 0x0000000C | Abort (prefetch) | Abort |
| 5 | 0x00000010 | Abort (data) | Abort |
| x | 0x00000014 | Reserved | Reserved |
| 6 | 0x00000018 | IRQ (external interrupt) | IRQ |
| 7 | 0x0000001C | FIQ (fast interrupt) | FIQ |

Accessing I/O Devices :

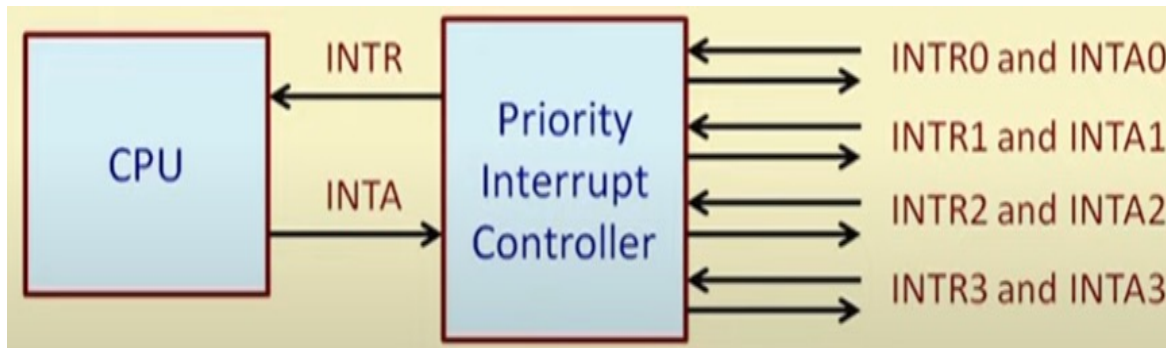
Interrupt request during the execution of an instruction.



Accessing I/O Devices :

Multiple devices interrupting CPU

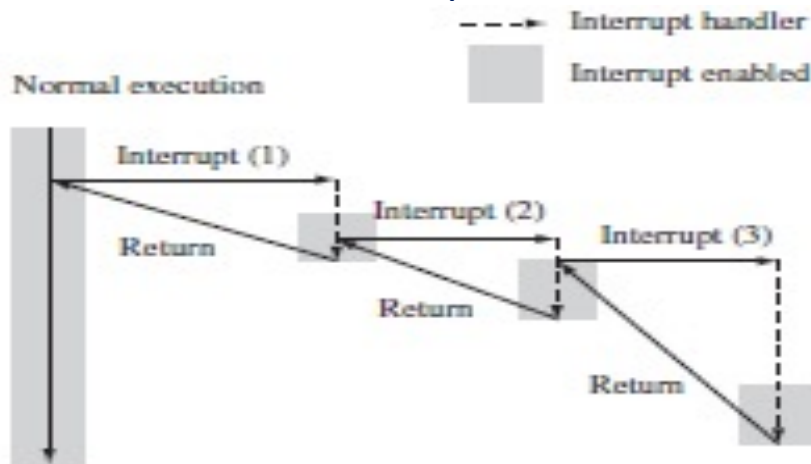
- Common solution is to use a **Priority Interrupt Controller**.
 - Interrupt controller interacts with CPU on one side and multiple devices on the other side.
 - For simultaneous interrupt requests, interrupt priority is defined.
 - Interrupt controller is responsible for sending the interrupt vector to the CPU.



Accessing I/O Devices :

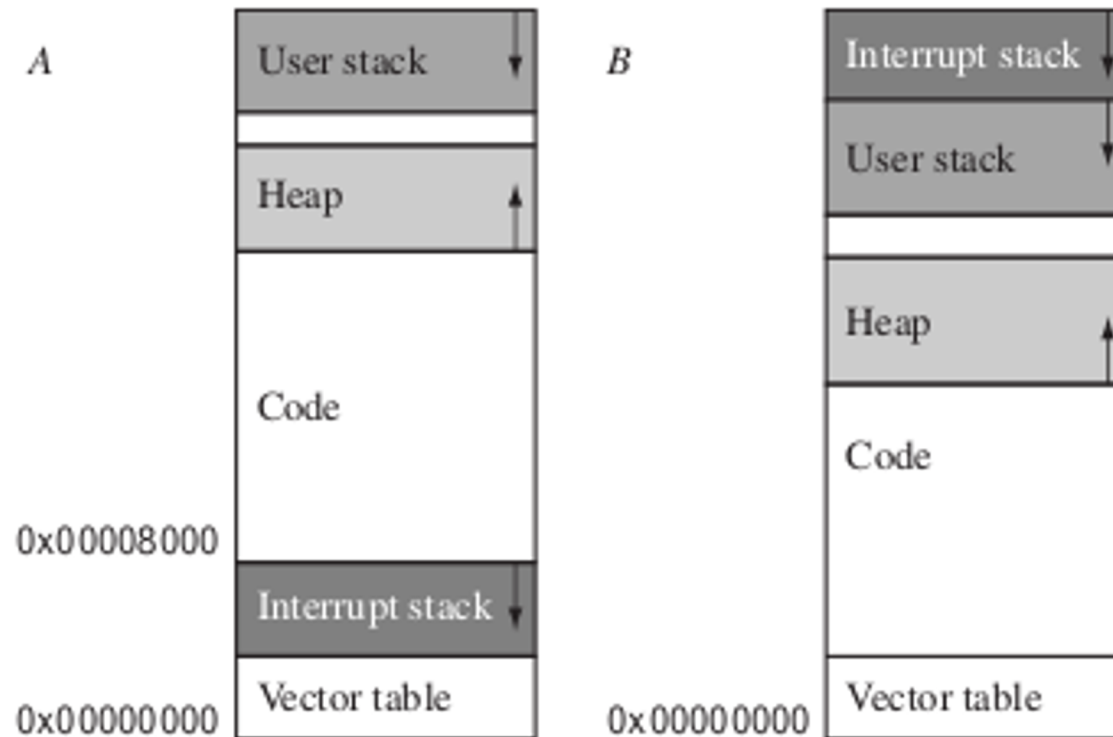
How is interrupt nesting handled?

- Consider a scenario.
- A device D0 has interrupted and the CPU is servicing the (executing the ISR) device D0.
- In the meantime, device D1 has interrupted.
- Two possible scenarios are here:
 1. D1 will interrupt the ISR for D0, get processed first, and then the ISR for device D0 will be resumed.
This creates problem for multi nesting.
 2. Disable the interrupt system automatically whenever an interrupt is acknowledged.
This will not require nested interrupts to be handled.



Basic Interrupt Stack Design and Implementation

Stack Layouts



Basic Interrupt Stack Design and Implementation

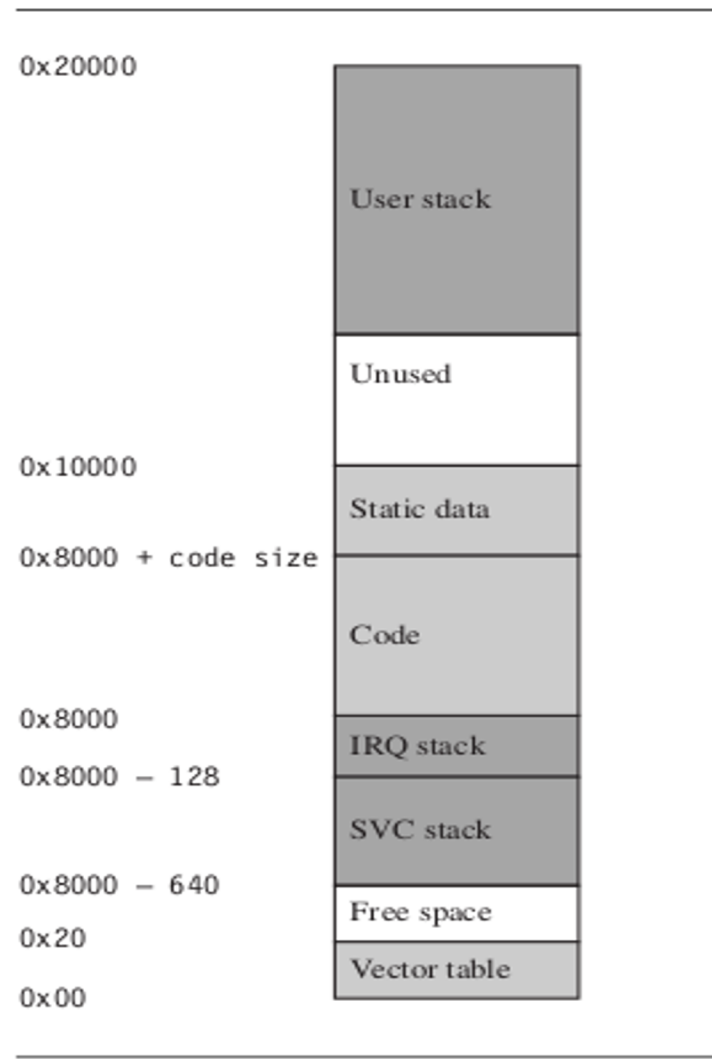
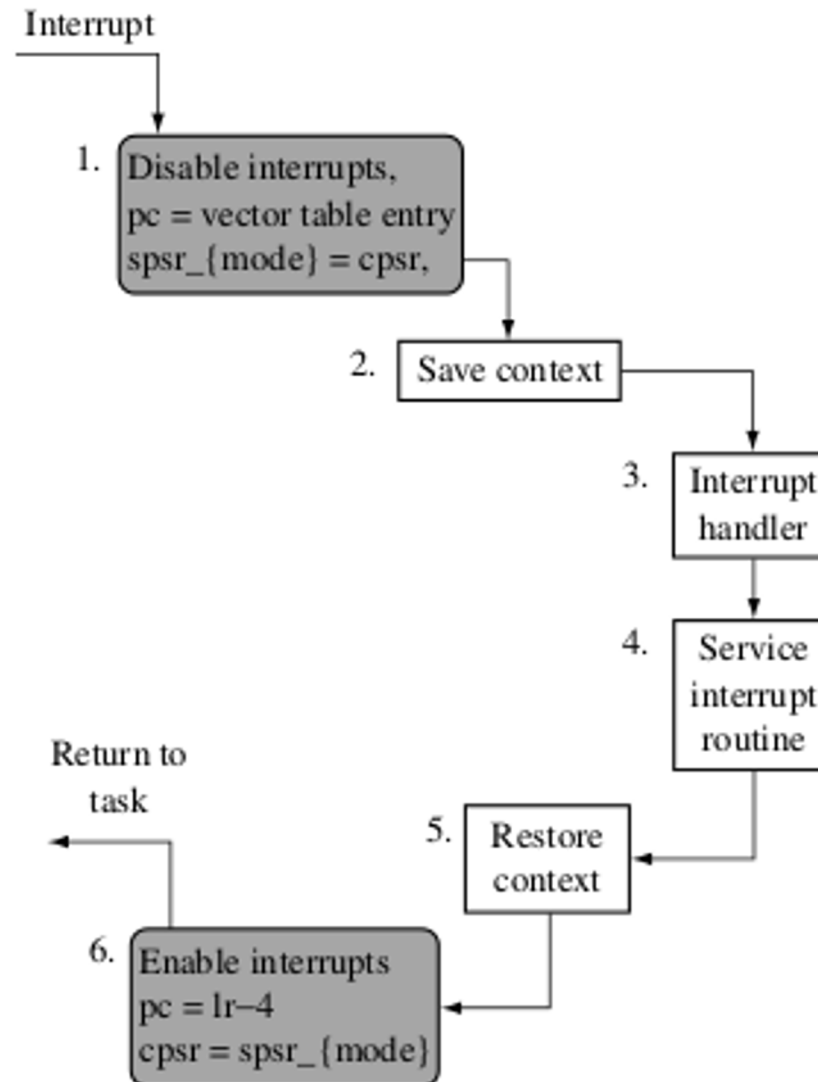


Figure 9.7 Example implementation using layout A.

- Various stages in a NNIH –
 - Enable Interrupts
 - $\text{spsr} \leftarrow \text{spsr}_{\{\text{interrupt request mode}\}}$
 - pc is restored

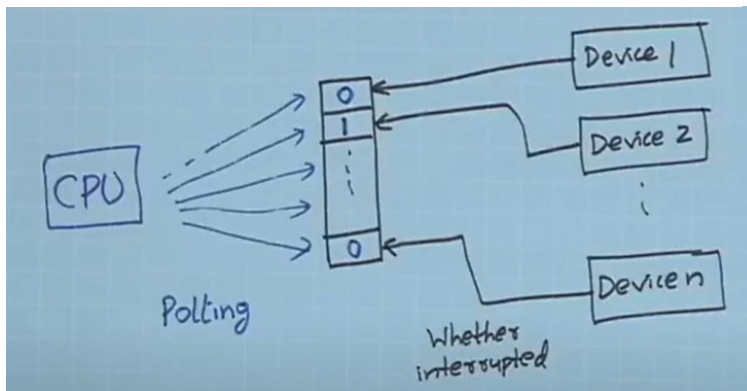
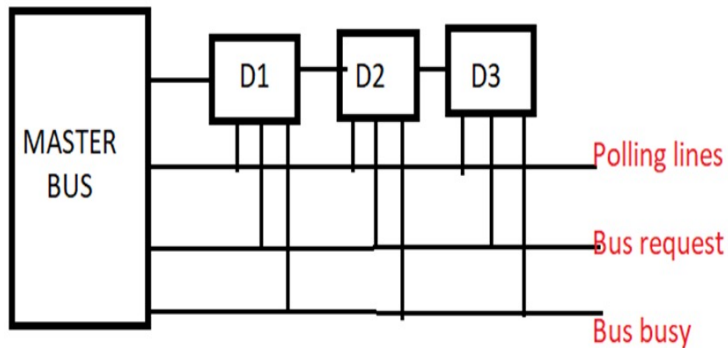
```
interrupt_handler
    SUB    r14,r14,#4                ; adjust lr
    STMFD  r13!,{r0-r3,r12,r14}     ; save context
    <interrupt service routine>
    LDMFD  r13!,{r0-r3,r12,pc}^     ; return
```

Interrupt Handling Schemes Non-Nested Interrupt Handler



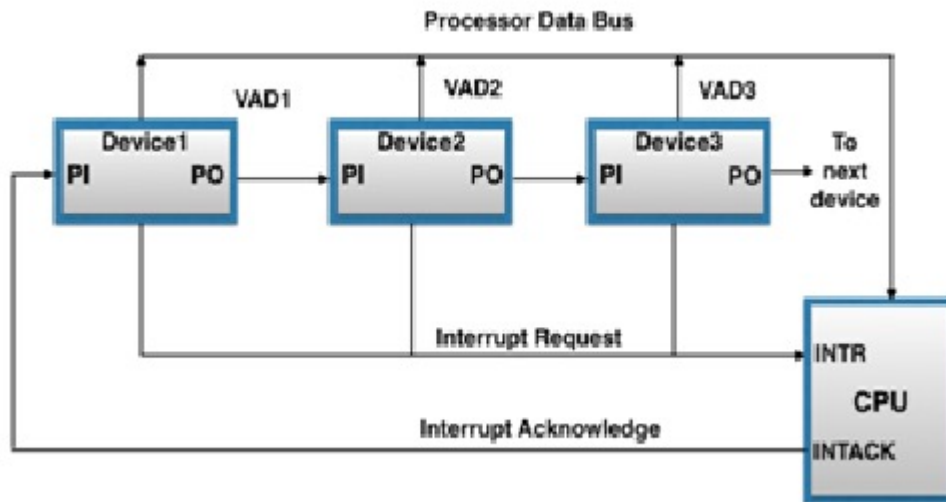
Accessing I/O Devices – Polling Technique

- Each device can have a status bit whether the device has interrupted?
- CPU can poll the status bit to check for the device which has interrupted.



Accessing I/O Devices – Daisy Chain Technique

- In Daisy chain technique, INTR line is common to all the devices.
- INTA line is connected in a daisy chain fashion [as shown].
- This allows to propagate serially through the devices.
- A device when it receives INTA, passes the signal to the next device only if it has not interrupted.
- Else, it stops the propagation of INTA and puts the identifying code on the data bus.
- Thus the device that is electrically closest to the CPU will have the highest priority.



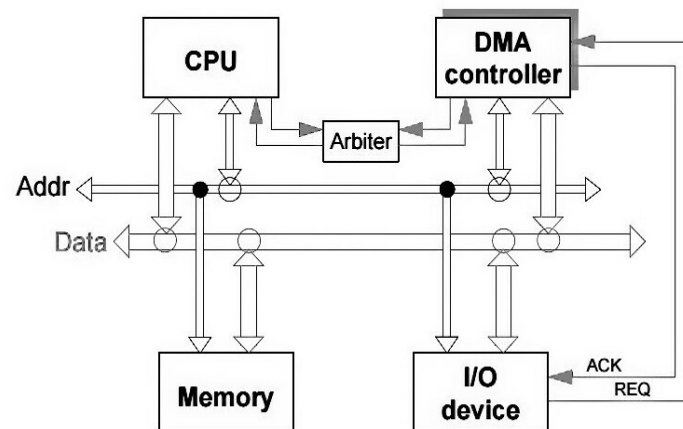
2. Direct Memory Access(DMA) Controller

What is actually DMA?

- The direct memory access is a system where the samples are saved in the memory of the system while the processor does something else.

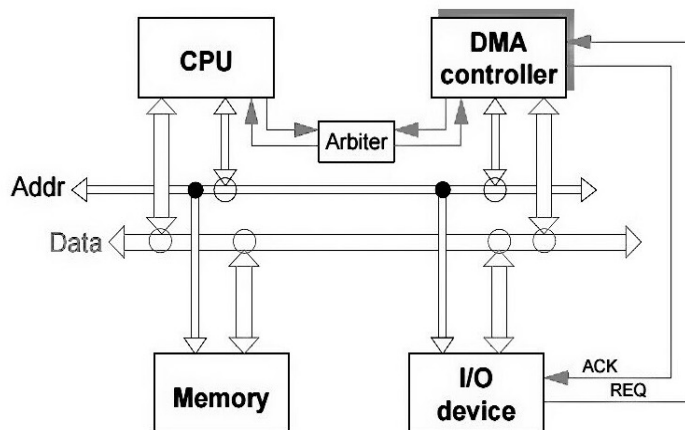
Why we need DMA?

- The assumption about the I/O machines like keyboards, mouse, and printer etc. are genuinely very slow when compared with the central processing unit (CPU).
- To overcome this issue an interrupt handler was used and the I/O machine produces all the signals that the CPU produce, then the I/O machine can bypass the information alienates to central processing unit and hence increases the speed.



2. Direct Memory Access(DMA) Controller

- **Cycle Stealing Process:** Traditionally it is a method of accessing RAM or bus without interfering with the CPU.
- DMA allows I/O controllers to read or write RAM without CPU intervention.
- Clever exploitation of specific CPU or bus timings can permit the CPU to run at full speed without any delay if external devices access memory not actively participating in the CPU's current activity and complete the operations before any possible CPU conflict.
- Such systems are nearly dual-port RAM without the expense of high speed RAM.
- Most systems halt the CPU during the **steal**, essentially making it a form of DMA by another name



- **Universal Serial Bus**
- **Peripheral Component Interconnect Bus**
- **Small Computer System Interface Bus**



Microprocessor & Computer Architecture (μ pCA)

Thank You

V R BADRI PRASAD

Department of
Computer Science and Engineering



USB, PCI, SCSI Bus Architecture

V R BADRI PRASAD

Department of
Computer Science and Engineering

Universal Serial Bus Architecture

- **Universal Serial Bus is a data interface used with computers enabling the computer to send and receive data as well as providing power to some peripherals like disc drives, Flash memory sticks and the like so that separate power sources are not needed for each item**



Universal Serial Bus Architecture

- USB Peripherals are slaves responding to commands from hosts
- When a peripheral is attached to the USB network, the host communicates with the device.
 - To learn its identity
 - To discover which device driver is required
 - This is called Enumeration
 - It is supported as the **device driver for the USB** port on the host.
- **Power :**
 - USB devices can pull limited amount power from the bus.
- **Speed:**
 - Low : 1.5Mbps
 - Full : 12Mbps
 - High : 480 Mbps

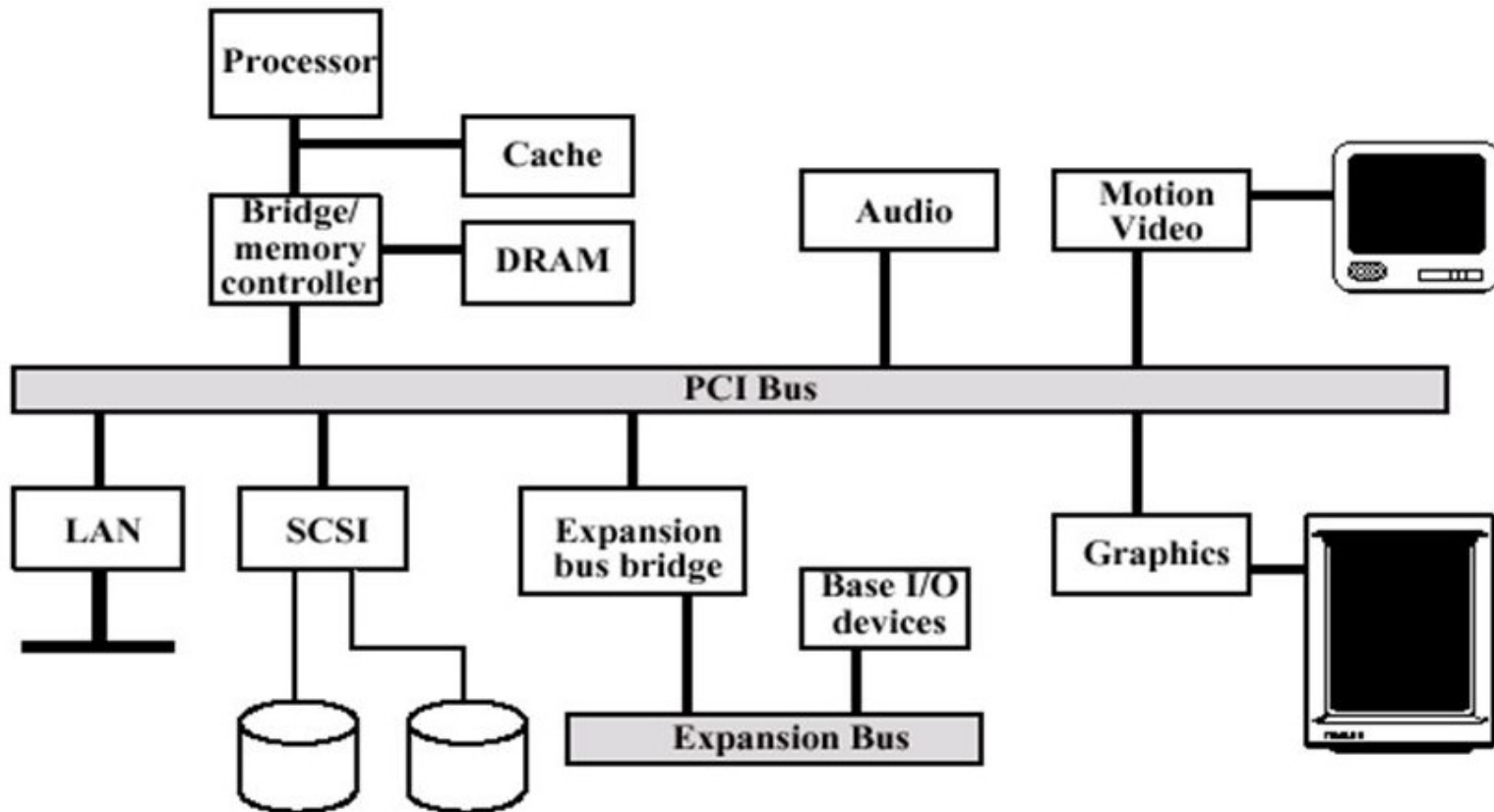
Universal Serial Bus Architecture

- **USB Devices are of two types.**
 - **Stand-alone:**
 - **Single Function Units like Mouse,etc.,**
 - **Compound Devices:**
 - **More than one peripheral sharing a USB port.**
Ex: Video Camera (both audio and video).
- **USB Hubs:**
 - **Hubs are bridge**
 - **Themselves are USB devices**
 - **Hubs detect topology changes due to insertion and deletion of devices.**



Peripheral Component Interconnect : PCI-Bus Architecture

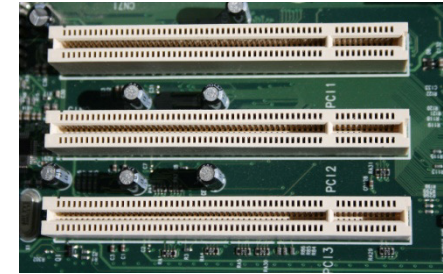
- PCI is a local computer bus for attaching hardware devices in a computer.



Peripheral Component Interconnect : PCI-Bus Architecture

Features:

- **32 bit bus**
- **Transfer rate : 133MB/s**
- Any PCI device may initiate a transaction.
- First, it must request permission from a PCI bus arbiter on the motherboard.
- The arbiter grants permission to one of the requesting devices.
- The initiator begins the address phase by broadcasting a 32- bit address plus a 4-bit command code, then waits for a target to respond.
- All other devices examine this address and one of them responds a few cycles later.



SCSI -Bus Architecture

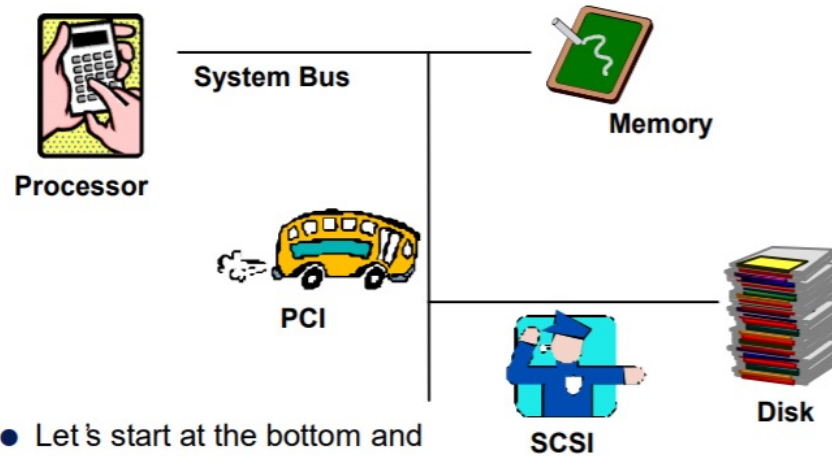
Small Computer System Interface – SCSI :

- Derived from **SASI** – Shugart Associates System Interface
- Provided as a bridge between hard disk low-level interface and a host computer
- Standard Interface and communication protocol for connecting computer peripherals.
- SCSI is used to increase performance and deliver faster
- Bus can address upto 8 devices (0 to 7).
- Provide larger expansion for devices such as CDs, scanners, etc.,
- Based on the Client Server Architecture
- Clients are Initiators – creates(begins)and sends SCSI commands to the target.
- Servers are Targets – Collection of logical units – carries out the requested task.
- Early SCSI assumes a bus based architecture

SCSI -Bus Architecture

Small Computer System Interface – SCSI :

Parallel SCSI: SCSI-1 : 8 bit wide bus – speed 3.5MHz
SCSI-2 : 16 /32 bit wide – speed 10MHz to 20MHz
SCSI-3 : Also called as ultra SCSI.
Cable length- 3 mts.
Speed – 20MHz



- Let's start at the bottom and work our way up...

SCSI -Bus Architecture

SCSI Bus Operation in phases:



- Bus Free
- Arbitration
- Selection / Reselection : target in control
 - Target decides when to recv msg, cmd, data from initiator.
 - OR send status to initiator
- Message
 - Used by target to send / recv protocol information.
Eg : Identify, cmd complete, msg parity error, etc.,
- Data, command, message, status



Microprocessor & Computer Architecture (μ pCA)

Thank You

V R BADRI PRASAD

Department of
Computer Science and Engineering



AMBA and ASB Bus Architecture

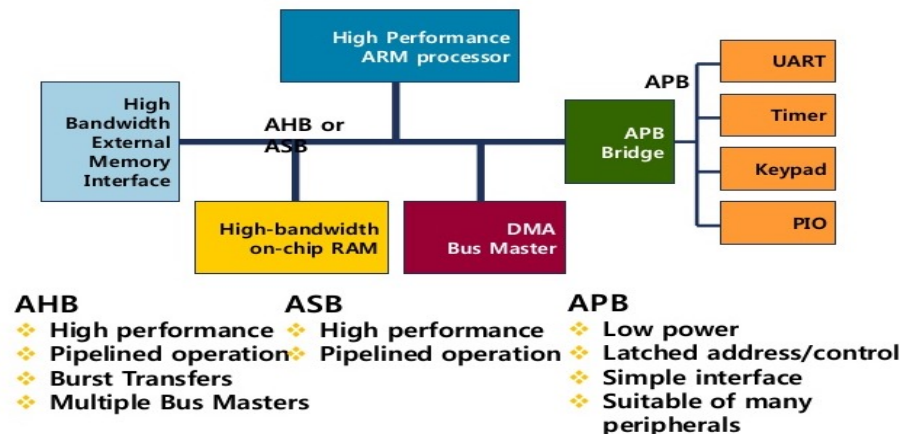
V R BADRI PRASAD

Department of
Computer Science and Engineering

AMBA (ADVANCED MICROPROCESSOR BUS ARCHITECTURE) PROTOCOL

- The Arm AMBA protocols are an open standard, on-chip interconnect specification for the connection and management of functional blocks in a System-on-Chip (SoC).
- It facilitates right-first-time development of multi-processor designs with large numbers of controllers and peripherals.
- Features of AMBA Interfaces:
 - IP reuse is essential in reducing SoC development costs and timescales. AMBA specifications provide interface standards that enables IP reuse if the following essential requirements are met:

AMBA



APB (ADVANCED PERIPHERAL BUS) PROTOCOL

- The Advanced Peripheral Bus (APB) is part of the Advanced Microcontroller Bus Architecture (AMBA) protocol family.
- It defines a low-cost interface that is optimized for minimal power consumption and reduced interface complexity.
- The APB protocol is not pipelined, use it to connect to low-bandwidth peripherals that do not require the high performance of the AXI protocol.
- The APB protocol relates a signal transition to the rising edge of the clock, to simplify the integration of APB peripherals into any design flow.
- Every transfer takes at least two cycles.
- The APB can interface with:
 - AMBA Advanced High-performance Bus (AHB)
 - AMBA Advanced High-performance Bus Lite (AHB-Lite)
 - AMBA Advanced Extensible Interface (AXI)
 - AMBA Advanced Extensible Interface Lite (AXI4-Lite)



Microprocessor & Computer Architecture (μ pCA)

Thank You

V R BADRI PRASAD

Department of
Computer Science and Engineering