**Computer Organization &Architecture-UNIT-II**

**Syllabus:**

**Peripheral devices and their characteristics:** Input-output subsystems, I/O device interface, I/O transfers

– program controlled, interrupt driven and DMA, privileged and non-privileged instructions, software interrupts and exceptions. Programs and processes – role of interrupts in process state transitions, I/O device interfaces – SCSI, USB

*Introduction to x86 architecture.*

**CPU control unit design:** Hardwired and micro-programmed design approaches, design of a simple hypothetical CPU.

***Input-output subsystems:***

The input-output subsystem of a computer, referred to as I/O which provides an efficient mode of communication between the central system and the outside environment.

* Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user.
* A computer serves **no useful purpose** without the ability to receive information from an outside source and to transmit results in a meaningful form.
* Input or output devices attached to the computer are also called **peripherals**.
* The most common peripherals are keyboards, display units, and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes. Peripherals are electromechanical and electromagnetic devices of some complexity.
* The most familiar means of entering information into a computer is a **keyboard** that allows a person to enter alphanumeric information directly. Every time a key is depressed, the terminal sends a binary coded character to the computer. The fastest possible speed for entering Information this way depends on the person's typing speed.
* There are different types of **video monitors**, but the most popular use a cathode ray tube (CRT). The CRT contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube.
* Printers provide a permanent record on paper of computer output data or text. There are three basic types of character printers, 1. Daisywheel 2. Dot matrix 3. Laser printers
* **Magnetic tapes** are used mostly for storing files of data: for example, a company's payroll record. Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read-write mechanism. It is one of the cheapest and slowest methods for storage and has the advantage that tapes can be removed when not in use.
* **Magnetic disks** have high-speed rotational surfaces coated with magnetic material. Access is achieved by moving a read-write mechanism to a track in the magnetized surface. Disks are used mostly for bulk storage of programs and data

Other input and output devices encountered in computer systems are **digital plotters**, **optical and magnetic character readers**, and various **data acquisition equipment** like **USB** devices.

Not all input comes from people, and not all output is intended for people. Computers are used to control various processes in real time, such as machine tooling, assembly line procedures, and chemical and industrial processes. For such applications, a method must be provided for sensing status conditions in the process and sending control signals to the process being controlled.

***Input-Output Device Interface:***

Input-output device interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to a computer need special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The **major differences** are:

1. Peripherals are electromechanical and electromagnetic devices and their manner of operation is different from the operation of the CPU and memory, which are electronic devices. Therefore, a conversion of signal values may be required.
2. The data transfer rate of peripherals is usually slower than the transfer rate of the CPU, and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output transfers. These components are called ***interface*** units because they interface between the processor bus and the peripheral device.

**l/O Bus and Interface Modules**

A typical communication link between the processor and several peripherals is shown in Figure(2.1).



**Figure (2.1):** Connection of I/O bus to input-output devices

Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device. **For example**, the printer controller controls the paper motion, the print timing, and the selection of printing characters.

* To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.
* At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit. The interpretation of the command depends on the peripheral that the processor is addressing.

There are **four types** of commands that an interface may receive. They are classified as

1. Control command

2. Status command

3. Data output command

4. Data input command

A **control command** is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral.

A **status command** is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

A **data output command** causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit. The computer starts the tape moving by issuing a control command. The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address command and transfers the information from the data lines in the bus to its buffer register. The interface then communicates with the tape controller and sends the data to be stored on tape.

The **data input command** is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

**I/O vs Memory Bus**

In addition to communicating with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address, and read/write control lines. There are three ways that computer buses can be used to communicate with memory and I/O:

1. Use two separate buses, one for memory and the other for I/O.

2. Use one common bus for both memory and I/O but have separate control lines for each.

3. Use one common bus for memory and I/O with common control lines.

* In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate **I/O processor (IOP)** in addition to the central processing unit (CPU).

The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/O bus with its own address, data and control lines. The I/O processor is sometimes called a **data channel**.

**Isolated vs Memory-Mapped I/O**

**Isolated I/O (**one common bus for both memory and I/O but have separate control lines for each**):** Many computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines.

* The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines.
* The I/O read and I/O write control lines are enabled during an I/O transfer.
* The memory read and memory write control lines are enabled during a memory transfer.

This configuration isolates (separates) all I/O interface addresses from the addresses assigned to memory and is referred to as the **isolated I/O** method for assigning addresses in a common bus.

**Memory-mapped I/O (**one common bus for both memory and I/O with common control lines**):**

The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space.

The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and I/O addresses. This configuration is referred to as **memory-mapped I/O**.

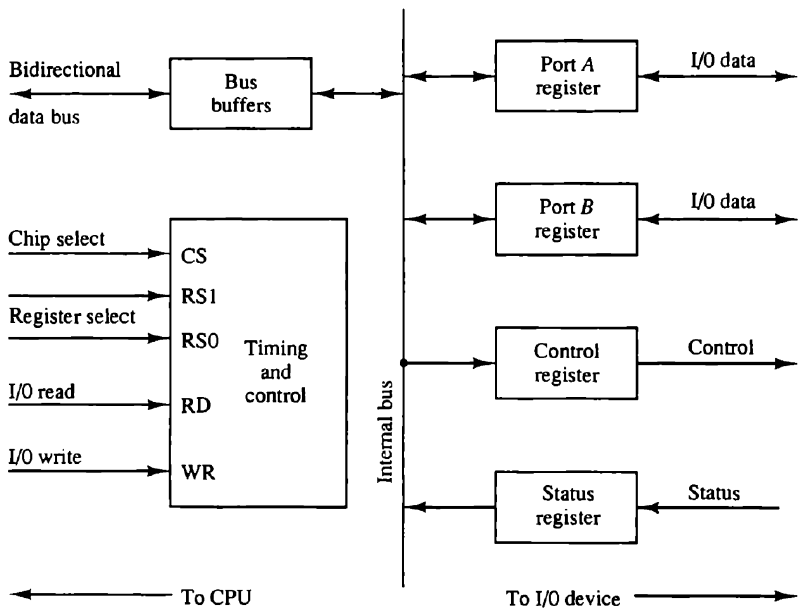
* In a memory-mapped I/O organization there are no specific input or output instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words. Each interface is organized as a set of registers that respond to read and write requests in the normal address space. Typically, a segment of the total address space is reserved for interface registers.
* Computers with memory-mapped I/O can use memory-type instructions to access I/O data. It allows the computer to use the same instructions for either input-output transfers or for memory transfers.

**Example of I/O Interface**

An example of an I/O interface unit is shown in block diagram form in Figure (2.2). It consists of two data registers called **ports**, a **control register**, a **status register**, **bus buffers**, and **timing and control circuits**.

The interface communicates with the CPU through the data bus. The **chip select** and **register select** inputs determine the address assigned to the interface. The I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or port B. The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data.





**Figure (2.2):** Example of an I/O Interface

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it can be placed in a variety of operating modes.

The interface registers communicate with the CPU through the bidirectional data bus. The address bus selects the interface unit through the chip select and the two register select inputs. A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (CS) input when the interface is selected by the address bus. The two register select inputs RS1and RS0 are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram.

***Modes of Transfer or I/O Transfers:***

Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit. The CPU executes the I/O instructions and may accept the data temporarily, but the **ultimate source or destination is the memory unit.**

Data transfer between the central computer and I/O devices may be handled in different modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit.

Data transfer to and from peripherals may be handled in one of three possible modes:

**1. Programmed I/O**

**2. Interrupt-initiated I/O**

**3. Direct memory access (DMA)**

**Programmed I/O**

* Programmed I/O operations are the result of I/O instructions written in the computer program. Each data item transfer is initiated by an instruction in the program.
* Transferring data under program control requires constant monitoring of the peripheral by the CPU.
* Once a data transfer is initiated, the CPU is required to monitor the interface to see when a transfer can again be made. It is up to the programmed instructions executed in the CPU to keep close tabs on everything that is taking place in the interface unit and the I/O device.

**Interrupt-Initiated I/O**

* In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device.

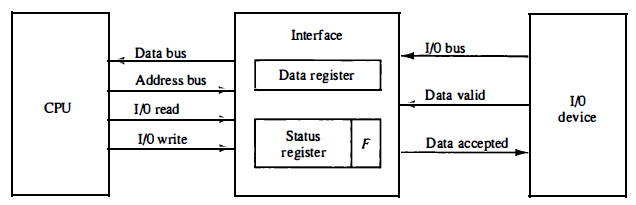
In the meantime the CPU can proceed to execute another program. The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

**DMA**

* Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks.
* When the transfer initiation is made by the CPU is done, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer.

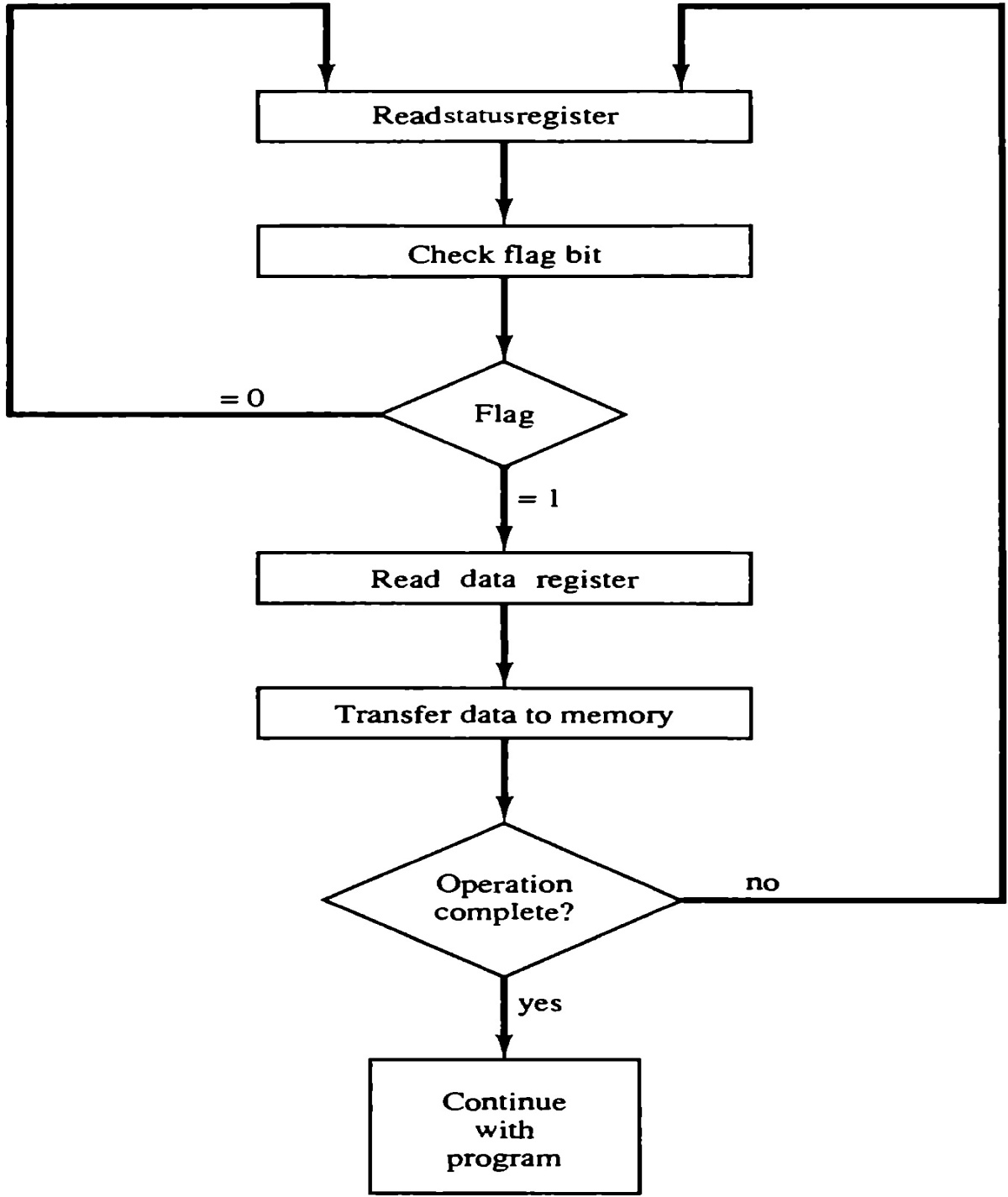
**Example of Programmed I/O:**

In the programmed I/O method, the I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory. An example of data transfer from an I/O device through an interface into the CPU is shown in Figure (2.3). The device transfers bytes of data one at a time as they are available.



**Figure (2.3):** Data transfer from an I/O device to CPU

A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the I/O device. This is done by reading the status register into a CPU register and checking the value of the flag bit. If the flag is equal to 1, the CPU reads the data from the data register. The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed. Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte.



**Figure (2.4):** Flowchart for CPU program to input data

🡪 Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer.

🡪 The main **drawback** of this method is, lot of valuable CPU time wastes while checking the status register continuously. (CPU will not do any other useful work in this case)

🡪 The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. The difference in information transfer rate between the CPU and the I/O device makes this type of transfer inefficient.

**Interrupt-Initiated I/O:**

An alternative to the CPU constantly monitoring the flag is to let the interface inform the CPU when it is ready to transfer data. This mode of transfer uses the interrupt facility.

* While the CPU is running a program, it does not check the flag. However, when the flag is set, the CPU is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set.
* The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the CPU returns to the previous program to continue what it was doing before the interrupt.
* The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer. The way that the processor chooses the branch address of the service routine varies from one unit to another.

In principle, there are two methods for accomplishing this. One is called **vectored interrupt** and the other, **non-vectored interrupt**.

In a **non-vectored interrupt**, the branch address is assigned to a fixed location in memory.

In a **vectored interrupt**, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.

* In some computers the interrupt vector is the first address of the I/O service routine.
* In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

**Priority Interrupts:**

Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer.

In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first.

A priority interrupt is a system that establishes a priority over the various sources to determine which request is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Devices with high speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority. When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

* Establishing the priority of simultaneous interrupts can be done by software or hardware.

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 program that takes care of interrupts 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.

The **disadvantage** of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.

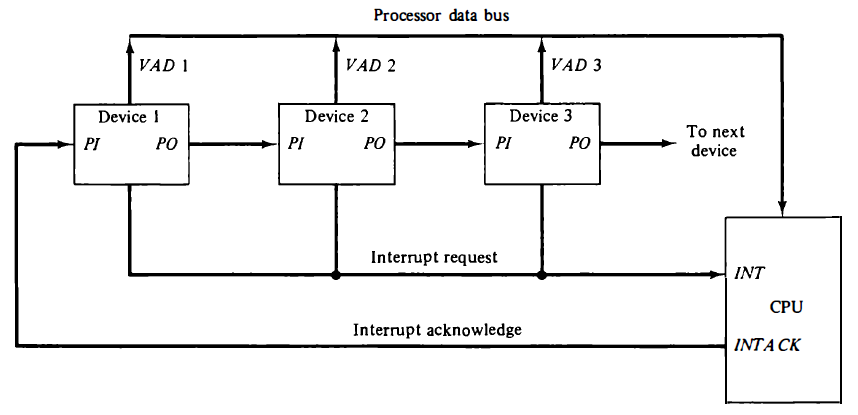
A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination. To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit. 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**.

**Daisy-Chaining Priority:**

The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt.

* The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in Figure (2.5).
* The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU.
* When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input.

The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.



**Figure (2.5):** Daisy-chain priority interrupt.

* A device with a 1 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
* If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus.
* The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.

**Parallel Priority Interrupt:** The parallel priority interrupt method uses a register (inside the arbitration circuit) whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.

Device

Device

circuit

Priority arbitration

Processor

Device

Device

I

N

T

R

1

I

N

T

R

*p*

INTA1

INTA

*p*

**Figure (2.6):** Priority Interrupt Hardware

* When I/O devices were organized into a priority structure, each device had its own interrupt-request and interrupt-acknowledge line.
* When I/O devices were organized in a daisy chain fashion, the devices shared an interrupt-request line, and the interrupt-acknowledge propagated through the devices.
* A combination of priority structure and daisy chain scheme can also be used.
* Devices are organized into groups. Each group is assigned a different priority level.
* All the devices within a single group share an interrupt-request line, and are connected to form a daisy chain.
* Only those devices that are being used in a program should be allowed to generate interrupt requests.
* To control which devices are allowed to generate interrupt requests, the interface circuit of each I/O device has an interrupt-enable bit.
  + If the interrupt-enable bit in the device interface is set to 1, then the device is allowed to generate an interrupt-request.
* Interrupt-enable bit in the processor status register or the priority structure of the interrupts determines whether a given interrupt will be accepted or not.

***Programs and processes – role of interrupts in process state transitions:***

The operating system makes extensive use of interrupts to perform I/O operations, as well as to communicate with and control the execution of programs. The interrupt mechanism enables the OS

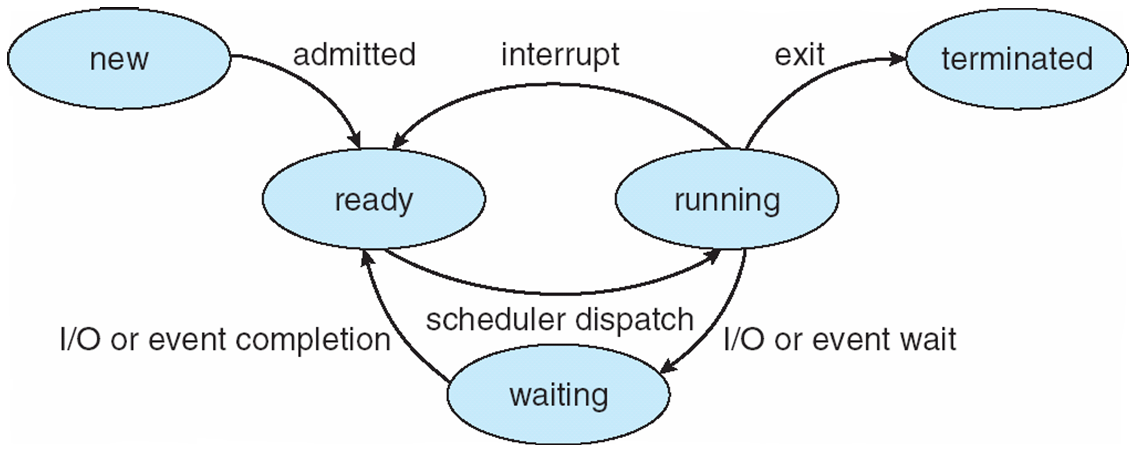
* To assign priorities,
* Switch from one program to another,
* Terminate programs,
* Implement security and protection features, and
* Coordinate I/O activities.

Some of these aspects will be discussed briefly to illustrate how interrupts are used.

A program which is under execution, together with any information that describes its current state of execution, is called a ***process***. A process can be in one of five states: **New**, **Ready**, **Running**, **Waiting** and **Terminated**. As a process executes, it changes state. The state of a process is defined in part by the current activity of that process.

* **New**: The process is being created.
* **Running**: Instructions are currently being executed.
* **Waiting**: The process is waiting for some event to occur (For example, it may be waiting for completion of an I/O operation that it requested earlier).
* **Ready**: The process is waiting to be assigned to a processor.
* **Terminated:** The process has finished execution

The state diagram corresponding to these states is presented in Figure 2.7.



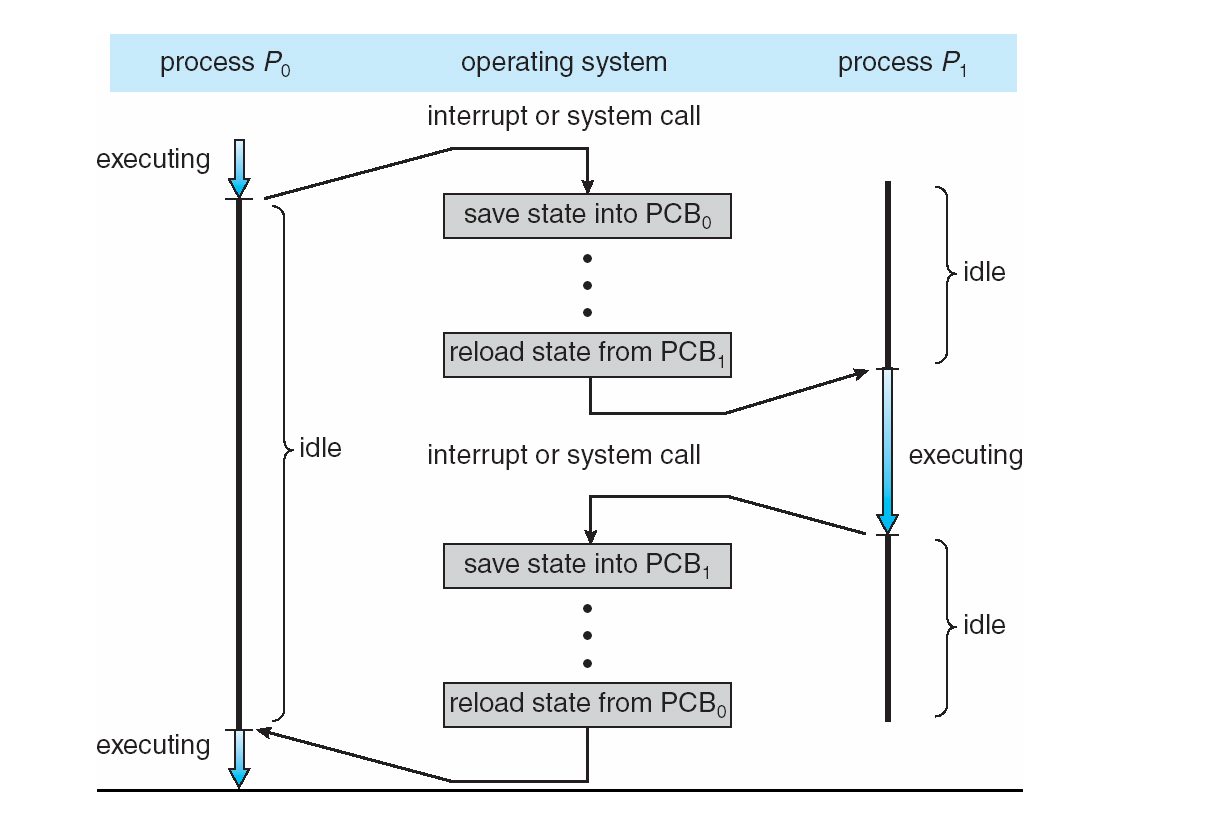
**Figure (2.7):** Process State diagram.

Each process is represented in the operating system by a process control block (PCB)-also called a task control block. A PCB is shown in Figure 4.2. It contains many pieces of information associated with a specific process, including these:

* Process state
* Program counter
* CPU registers
* CPU scheduling information
* Memory-management information
* Accounting information
* I/O status information

The PCB simply serves as the repository for any information that may vary from process to process.

The OS incorporates the interrupt-service routines for all devices connected to a computer that are capable of raising interrupts. In a general-purpose computer with an operating system, application programs do not directly perform I/O operations themselves. When an application program needs an I/O operation, it points to the data to be transferred and asks the OS to perform the operation. The request from the application program is often made through a hardware interrupt or software interrupt (system call). The OS temporarily suspends the execution of the requesting program, then initiates the requested I/O operation by saving suspended process details in a PCB for future use and also reload the details of interrupt handler routine into another PCB. When the I/O operation is completed, the OS is normally informed of this condition through a hardware interrupt. The OS then allows the suspended program to resume execution. The OS and the application program pass control back and forth using software interrupts. The process of changing the control between various programs is called as ***context switching***. This context switch is illustrated in the below figure 2.8.



**Figure (2.8):** Diagram showing CPU switch from process to process.

***Small Computer System Interface (SCSI)***

* SCSI is a common standard for connecting peripheral devices (disks, modems, printers, etc.) to small and medium-sized computers. Although SCSI has evolved to higher data rates, it has lost its popularity due to competitors like USB and FireWire in smaller systems.
* However, high-speed versions of SCSI remain popular for mass memory support on enterprise systems. For example, the IBM zEnterprise EC12 and other IBM mainframes offer support for SCSI and a number of Seagate hard drive systems use SCSI.
* The physical organization of SCSI is a shared bus, which can support up to 16 or 32 devices, depending on the generation of the standard. The bus provides for parallel transmission rather than serial, with a bus width of 16 bits on earlier generations and 32 bits on later generations. Speeds range from 5 Mbps on the original SCSI-1 specification to 160 Mbps on SCSI-U3.

**Data Transfer**

Data and commands are transferred in the form of multi-byte messages called ***packets***. To send commands or data to a device, the processor assembles the information in the memory then instructs the SCSI controller to transfer it to the device. Similarly, when data are read from a device, the controller transfers the data to the memory and then informs the processor by raising an interrupt.

To illustrate the operation of the SCSI bus, let us consider how it may be used with a disk drive. Communication with a disk drive is completely different from communication with the main memory. Data are stored on a disk in blocks called **sectors**, where each sector may contain several hundred bytes. When a data file is written on a disk, it is not always stored in contiguous sectors. Some sectors may already contain previously stored information; others may be defective and must be skipped. Hence, a Read or Write request may result in accessing several disk sectors that are not necessarily contiguous. Because of the constraints of the mechanical motion of the disk, there is a **long delay** of several milliseconds, before reaching the first sector to or from which data are to be transferred. Then, a burst of data are transferred at high speed. Another delay may be happened to reach the next sector, followed by a burst of data. A single Read or Write request may involve several such bursts.

Let us examine a complete Read operation as an example. Assume that the processor wishes to read a block of data from a disk drive and that these data are stored in two disk sectors that are not contiguous. The processor sends a command to the SCSI controller, which causes the following sequence of events to take place:

1. The SCSI controller contends for control of the SCSI bus.
2. When it wins the arbitration process, the SCSI controller sends a command to the disk controller, specifying the required Read operation.
3. The disk controller cannot start to transfer data immediately. It must first move the read head of the disk to the required sector. Hence, it sends a message to the SCSI controller indicating that it will temporarily suspend the connection between them. The SCSI bus is now free to be used by other devices.
4. The disk controller sends a command to the disk drive to move the read head to the first sector involved in the requested Read operation. It reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data, it requests control of the bus. After it wins arbitration, it re-establishes the connection with the SCSI controller, sends the contents of the data buffer, then suspends the connection again.
5. The process is repeated to read and transfer the contents of the second disk sector.
6. The SCSI controller transfers the requested data to the main memory and sends an interrupt to the processor indicating that the data are now available.

***Universal Serial Bus (USB):***

The Universal Serial Bus (USB) is the most widely used interconnection standard. A large variety of devices are available with a USB connector including mice, memory keys, disk drives, printers, cameras, and many more. The commercial success of the USB is due to its **simplicity** and **low cost**.

1. The original USB 1.0 specification supports two speeds of operation, called low-speed (1.5 Mbps) and full-speed (12 Mbps).
2. Later, USB 2.0, called High-Speed USB was introduced. It enables data transfers up to 480 Mbps.
3. As I/O devices continued to evolve with even higher speed requirements, USB 3.0 (Superspeed) was developed. It supports data transfer rates up to 5 Gbps.
4. The most recent specification is USB 3.1, which includes a faster transfer mode called Superspeed+. This transfers at a rate of 10 Gbps.

The USB has been designed to meet several key objectives:

* Provide a simple, low-cost, and easy to use interconnection system.
* Accommodate a wide range of I/O devices and bit rates, including Internet connections and audio and video applications.
* Enhance user convenience through a “plug-and-play” mode of operation.

**Plug-and-Play feature**

When an I/O device is connected to a computer, the operating system needs some information about it. It needs to know what type of device it is so that it can use the appropriate device driver. It also needs to know the addresses of the registers in the device’s interface to be able to communicate with it.

The USB standard defines both the USB hardware and the software that communicates with it. Its ***plug-and-play*** feature means that when a new device is connected, the system detects its existence automatically. The software determines the kind of device and how to communicate with it, as well as any special requirements it might have. As a result, the user simply plugs in a USB device and begins to use it, without having to get involved in any of these details. The USB is also ***hot-pluggable***, which means a device can be plugged into or removed from a USB port while power is turned on.

**Electrical Characteristics**

USB connections consist of **four** wires, of which two carry power, +5 V and Ground, and two carry data. Thus, I/O devices that do not have large power requirements can be powered directly from the USB. This avoids the need for a separate power supply for simple devices such as a memory key or a mouse.

**USB Architecture**

* The USB uses point-to-point connections and a serial transmission format. When multiple devices are connected, they are arranged in a **tree structure** as shown in the below Figure 2.9. Each node of the tree has a device called a **hub**, which acts as an intermediate transfer point between the host computer and the I/O devices. At the root of the tree, a **root hub** connects the entire tree to the host computer. The leaves of the tree are the I/O devices: a mouse, a keyboard, a printer, an Internet connection, a camera, or a speaker. The tree structure makes it possible to connect many devices using simple point-to-point serial links.
* When a device is first connected to a hub, or when it is powered on, it has the address 0 (7-btt address). Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected. When the host is informed that a new device has been connected, it reads the information in a special memory in the device’s USB interface to learn about the device’s capabilities. It then assigns the device a unique USB address and writes that address in one of the device’s interface registers. It is this initial connection procedure that gives the USB its plug-and-play capability.



**Figure (2.9):** Universal Serial Bus tree structure

***Privileged and non-privileged instructions:***

An inexperienced programmer might execute some sensitive code instructions and that might lead to system failure and erroneous functioning of the [operating system](https://www.codingninjas.com/codestudio/library/introduction-to-operating-system). **For example**, in any computer system in which independent user programs coexist in the main memory, Thus to ensure the proper functioning of the operating system, processor and memory, the need of protection must be addressed. No program should be allowed to destroy the function of processor or OS or the data or instructions of other programs in the memory. The needed protection can be provided in several ways. Let us first consider the most basic form of protection.

Most processors can operate in one of two modes, the ***supervisor mode (****System mode or kernel mode****)*** and the ***user mode***.

🡪The processor is usually placed in the supervisor mode when operating system routines are being executed.

🡪When it is the user mode, it will execute user programs.

🡪Some machine instructions cannot be executed in the user mode. These are called ***privileged instructions*** which can only be executed while the processor is in the supervisor mode. These instructions are like,

1. Modifying the page table base register
2. Executing interrupt-service routines (Interrupt management)
3. Manipulation of the status bits register
4. Executing some I/O instructions and to control I/O devices
5. Specialized instructions deals to access some regions in the memory
6. Performing some privileged operating system tasks (for example, Context switching)
7. Shut downing the system (or Halting the processor)
8. Change the contents of a control register.

If a privileged instruction is encountered by the processor while it is executing a user program, the processor hardware considers this as run-time error and an interrupt or an exception will be occurred and then transfers control to the OS.

Most application programs execute in ***user mode***. While the processor is in user mode, the program

being executed is unable to access protected system resources or to change mode, other than by causing an exception to occur. **Non-Privileged instructions** are the instructions that are only executed in user mode.

Example Non-Privileged instructions are:

1. Executing any normal instructions (load, store , add, subtract etc)
2. Generating the interrupts through user programs
3. Reading status of processor
4. Reading system time
5. Calling the OS for some user operations (printing, getting input from the user, displaying output etc) by using system calls or library functions of OS.

***Software interrupts:***

An Interrupt refers to the transfer of program control from a currently running program to another service program as a result of an external or internal generated request. Control returns to the original program after the service program is executed.

There are three major types of interrupts that cause a break in the normal execution of a program. They can be classified as:

1. External interrupts 2. Internal interrupts 3. Software interrupts

**External interrupts** come from input-output (I/O) devices, from a timing device, from a circuit monitoring the power supply, or from any other external source. Examples that cause external interrupts are I/O device requesting transfer of data, I/O device finished transfer of data, elapsed time of an event, or power failure.

**Internal interrupts** arise from illegal or erroneous use of an instruction or data. Internal interrupts are also called traps . Examples of interrupts caused by internal error conditions are register overflow, attempt to divide by zero, an invalid operation code, stack overflow, and protection violation. These error conditions usually occur as a result of a premature termination of the instruction execution.

* External and internal interrupts are initiated from signals that occur in the hardware of the CPU.

A **software interrupt** is initiated by executing an instruction. Software interrupt is a special call instruction that behaves like an interrupt rather than a subroutine (function) call. It can be used by the programmer to initiate an interrupt procedure at any desired point in the program. The most common use of software interrupt is associated with a supervisor call instruction. This instruction provides means for switching from a CPU user mode to the supervisor mode.

A program written by a user must run in the user mode. When an input or output transfer is required, the supervisor mode is requested by means of a supervisor call instruction. This instruction causes a software interrupt that stores the old CPU state and brings in a new PSW that belongs to the supervisor mode. The calling program must pass information to the operating system in order to specify the particular task requested.

***Exceptions:***

An interrupt is an event that causes the execution of one program to be suspended and the execution of another program to begin.

The term *exception* is often used to refer to any event that causes an interruption. In fact, the term exception is used generally to describe any hardware-initiated or software-initiated deviation from normal execution. Hence, I/O interrupts are one example of an exception.

***Types of exceptions***.

1. **Recovery from Errors**

The processor may also interrupt a program if it detects an error or an unusual condition while executing the instructions of this program. For example, the OP-code field of an instruction may not correspond to any legal instruction, or an arithmetic instruction may attempt a division by zero.

When exception processing is initiated as a result of such errors, the processor proceeds in exactly the same manner as in the case of an I/O interrupt request. It suspends the program being executed and starts an exception-service routine, which takes appropriate action to recover from the error if possible, or to inform the user about it.

1. **Debugging**

System software usually includes a program called a debugger, which helps the programmer find errors in a program. The debugger uses exceptions to provide two important facilities: trace mode and breakpoints.

***Trace Mode****:* When a processor is operating in the trace mode, an interrupt occurs after the execution of every instruction. An interrupt-service routine in the debugger program is invoked each time this interrupt occurs.

***Breakpoints:*** Breakpoints provide a similar interrupt-based debugging facility, except that the object program being debugged is interrupted only at specific points indicated by the programmer. For example, the programmer may set a breakpoint to determine whether a particular subroutine in the object program is ever reached. If it is, the debugger is activated through an interrupt. A special instruction called Trap or Software-interrupt is usually used to implement breakpoints.

1. **Privilege mode exceptions**

There are some exceptions which occur when the processor in the privileged mode. For example, the ARM processor has seven operating modes. One is the **user mode**, five are exception modes and seventh mode is **system mode (**Exception modes and system mode are called as privileged modes**)**.

🡪 Application or user programs run in **User mode**.

🡪 The **System mode** is used for running certain privileged operating system tasks. System mode tasks may be interrupted by any of the five exception categories.

The exception modes have full access to system resources and can change modes freely. These are entered when specific exceptions occur. Various exceptions occur in the exception modes are as follows:

■ **Supervisor mode:** Usually what the OS runs in. It is entered when the processor encounters a software interrupt instruction. Software interrupts are a standard way to invoke operating system services on ARM.

■ **Abort mode:** Entered in response to memory faults.

■ **Undefined mode:** Entered when the processor attempts to execute an instruction that is supported neither by the main integer core nor by one of the coprocessors.

■ **Fast interrupt mode:** Entered whenever the processor receives an interrupt signal from the designated fast interrupt source. A fast interrupt cannot be interrupted, but a fast interrupt may interrupt a normal interrupt.

■ **Interrupt mode:** Entered whenever the processor receives an interrupt signal from any other interrupt source (other than fast interrupt). An interrupt may only be interrupted by a fast interrupt.

When an exception is occurred, then the exception should be handled by the processor. For example, the Nios II processor performs the following actions when an exception is occurred:

1. Saves the existing processor status information by copying the contents of the ***status*** register (ctl0) into the ***estatus*** register (ctl1).
2. Clears the ***U***bit in the status register, to ensure that the processor is in the Supervisor mode.
3. Clears the ***PIE***bit in the status register, which prevents further external processor interrupts.
4. Writes the return address, which is the address of the instruction after the exception into the **ea** register (r29).
5. Transfers execution to the address of the ***exception handler***, which determines the cause of the exception and calls the required ***exception-service routine***to respond to the exception.