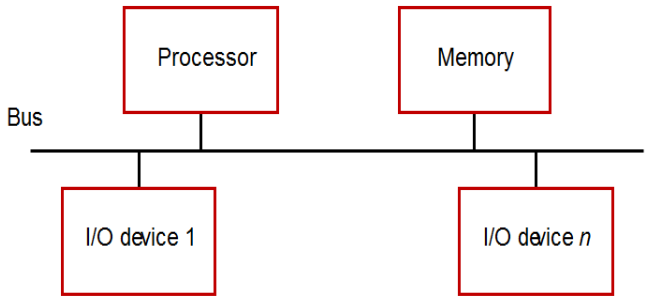
4. INPUT / OUTPUT ORGANIZATION

4.1 Accessing I/O Devices

Under this section we are going to know how exactly a processor identifies various I/O devices connected to it and how it actually communicates with those devices. To get the better understanding on this concept, we will consider a typical single bus arrangement shown below.



A common bus called as processor bus connects processor, memory and various I/O devices across it. A bus consists of three set of lines:

i. Address lines: carries address information to be accessed

ii. Data lines: carries the data that need to be transferred to/ from the processor.

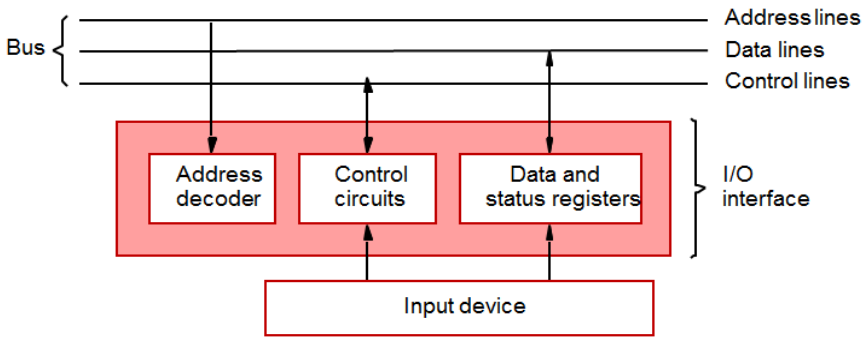
iii. Control lines: carries the commands specifying the type of operation (read/write) to be performed. Apart from this, it also carries the timing signals which decide when exactly the operation has to be initiated.

Each I/O device has a unique address to identify it and to access it; processor places address of the device to be accessed on the address line, device with that address recognizes its address and responds to the command issued over control lines and then the requested data is transferred over data lines.

For the addressing purpose, if I/O devices and memory share same address space, then the system is called memory mapped I/O. Some system even has a different spaces for I/O devices and memory unit, then they are called I/O mapped I/O.

4.1.1 Hardware for connecting I/O device

A hardware that is used to connect I/O device on to the processor bus is called as I/O interface. Basically, it is the one that acts as middleware for communication between the processor and an I/O device. Typical interface hardware for connecting an I/O device is shown:



Following are the components of the interface hardware and their functionalities:

i. Address decoder: This component is used to recognize the device address sent over a address lines and is connected to address lines of the processor bus.

ii. Control circuits: This component is used to control and co-ordinate all the I/O transfer activity that takes place between a processor and a device. These components send and receive the control signals for coordinating the activities via control lines.

iii. Data and status registers: The data register holds the data that needs to be transferred to/ from the device and the status register holds some important information that is required for the operation that needs to be carried on the device.

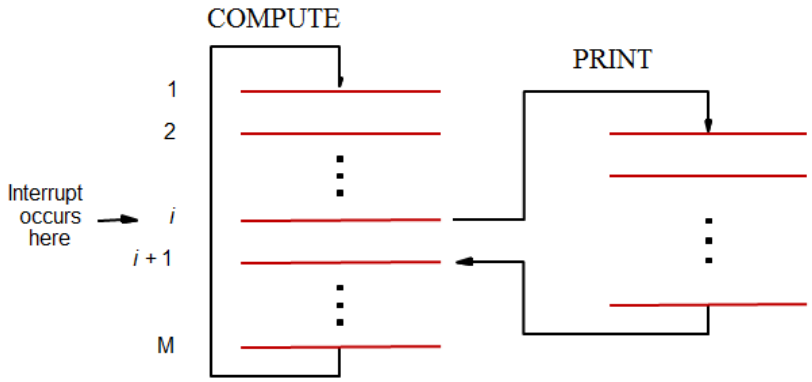
4.2 Interrupts

Interrupts are the hardware signal sent by a device informing the processor that it is ready to perform the required operation. One of the control line of the bus is dedicated for sending the interrupt signal and is called as **interrupt request line.**

Basically when a processor request some I/O operation from the I/O device, a convectional approach is that it has to wait till the device completes its operation and to do so, processor has to monitor the status of the device continuously to know whether it has become free or not. According to the concept of interrupts, processor instead of wasting the processor cycles in waiting for the device to complete the operation, it can continue doing some other useful task and when the device becomes free, device itself informs the processor that it is free.

To get a very clear picture on the concept of interrupts, lets us consider some example.

Example: Consider a task of performing some computation and results generated from the computation must be printed using a line printer. To implement this we will consider using some program that consists of two routines, COMPUTE and PRINT. Routine COMPUTE produces a results and sends it to PRINT routine, PRINT routine sends the received results one by one into the line printer for printing. Problem can be schematically represented as shown:



Here to accomplish the required task, the COMPUTE routine and PRINT routine is executed repeatedly. The COMPUTE routine performs some computations on the data and produces the results. As and when the results are produced, they are sent to the PRINT routine for printing the result through line printer. Since the line printer accepts only one line at a time, a single line of text is sent to the line printer by PRINT routine and then wait for it to complete the action, then send the next line and so on. The main disadvantage of this approach is that, processor has to waste lot of clock cycles in waiting for the device to get ready. Instead of doing that, with the concept of interrupts, processor can overlap the execution of COMPUTE routine and PRINT routine. This can be achieved as follows:

First the processor executes COMPUTE routine and produces n number of results. Then the produced results are sent to the PRINT routine for performing printing of the results through line printer. PRINT routine sends a single line of text to the line printer and instead of waiting for line printer to complete the operation, PRINT routine is temporarily suspended and execution of the COMPUTE routine is continued. Whenever the printer becomes ready, it sends the interrupt request signal the processor via interrupt request line and in response, processor interrupts the execution of COMPUTE routine and transfers the control to PRINT routine to carry out the printing of next line.

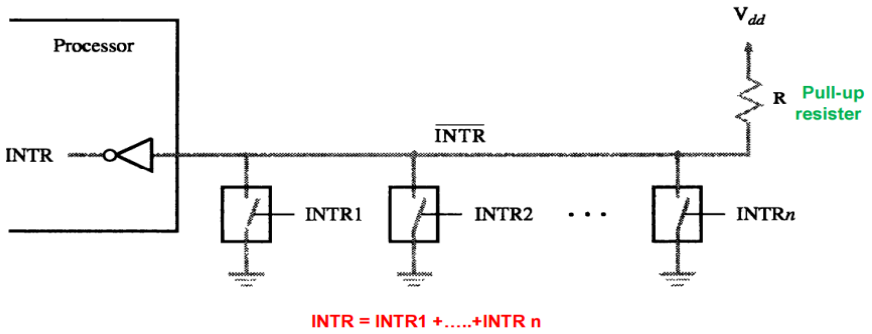
This example illustrates all the concepts of the interrupts. When there is a interrupt occurring, in response to the interrupt, processor suspends the current execution and transfers the control of execution to a routine and this routine is called as **interrupt service routine.** In the example we discussed, PRINT routine is the interrupt service routine.

When any device requests interrupt, processor has to respond back to the device informing that it has accepted the interrupt request and this is done by a special signal called **interrupt acknowledge signal.**

The only difference between subroutine call and interrupt service routine is that subroutines are called by a program currently executing it where as the Interrupt Service Routines are called by itself.

4.2.1 Interrupt Hardware

I/O device requests an interrupt by activating a bus control line called **interrupt request line.** Typical hardware arrangement for interrupt concept is shown below:



A very simple approach is used in the basic hardware for interrupt were a s single interrupt request line is used to serve n devices connected to the processor.

In this approach n number of devices connected to the system is connected to a common interrupt request line INTR (Active low line) via switch. Any device wanting request interrupt closes its corresponding switch connected to the interrupt request line. On closing the switch the line gets activated and signals the processor that a device has requested for the interrupt.

4.2.2 Enabling and disabling the interrupts

According to the convectional approach discussed in the beginning states that when there is a interrupt request, processor suspends the current execution and starts executing the interrupt service routine of the corresponding device that requested interrupt.

But it does not hold well in all cases. In some cases, the interrupt request must be ignored. This can be done either by the processor or by some other procedure were there should be a method for enabling and disabling the interrupts when required.

Different methods for handling interrupt for single device

Method 1:

First possibility is to have processor ignore any interrupt request till it starts the execution of the first instruction of the interrupt service routine and first instruction of the ISR will interrupt disable instruction and the last instruction before the return instruction is interrupt enable instruction

Method 2:

In this method, rather than controlling the interrupt through instructions, the controlling is done by the processor hardware. A processor hardware in this case maintains a status bit called IEN for enabling or disabling the interrupt. When IRQ is received, the processor sets the IEN bit to 0 which disables the interrupt request line and when return is executed, processor automatically sets IEN to 1 so that the further requests are activated.

Method 3:
In the last method, processor has a special kind of interrupt request line, which responds only to the leading edge which is called **edge triggered interrupt request line**. In this case, interrupts are accepted only on the leading edge and once after accepting, the line remains in the negative edge till the ISR is serviced and goes to the positive edge once completing the execution of ISR.

4.3 Handling Interrupts in Multiple devices

We know that, a system has n number devices connected across it and any device can request interrupt at any instant of time. So when it comes to handling of interrupts in multiple devices, there few problems that need to be handled and following are the problems:

1. How can a processor recognize a device requesting interrupt?
2. How will processor come to know the starting address of the ISR of the device requesting interrupt?
3. Should a device be allowed to interrupt while another interrupt is getting serviced?
4. How to handle two or more simultaneous interrupt request from the devices?

In the following sections we will discuss these problems in detail and will know the method to handle these problems effectively.

4.3.1 How can a process recognize the device requesting interrupt and how it obtains the starting address of the ISR

When there are many devices connected over a common interrupt request line, any device can raise an interrupt request and it becomes essential for the device to identify a particular device that has requested for interrupt and also to execute its corresponding ISR. There are two different methods to overcome this problem:

1. Polling scheme
2. Vectored interrupts

1. Polling scheme:

In this method, when any device requests for an interrupt over a IRQ line, it sets its corresponding IRQ bit value to 1. Processor when receives an interrupt request over a common line, it executes a common ISR which polls each and every device and checks the IRQ bit of

each and every device in sequence. Any device polled first with IRQ bit set to 1 is serviced by executing its corresponding ISR.

Disadvantage: The only disadvantage in this method is that the processor has to spend lot of time interrogating the status of device.

2. Vectored Interrupts:

This scheme came up with the intention to overcome the disadvantage of polling scheme. Here according to the principle of this method, device requesting an interrupt identifies itself to the processor and processor immediately starts executing the ISR of the corresponding device that requested interrupt.

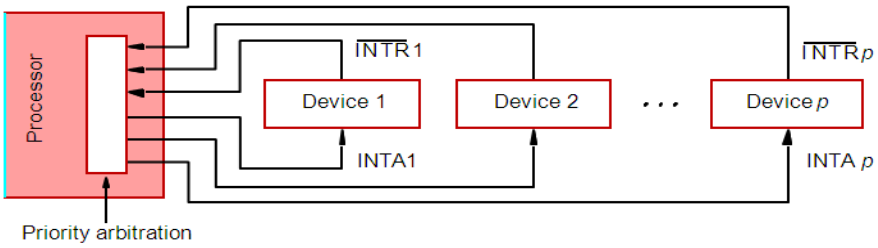
In this technique device that needs to interrupt sends an IRQ signal over a common line. Processor on receiving this sends interrupt acknowledge INTA signal to all the devices. Device that raised an interrupt responds to the INTA signal by sending a special code to the processor. This special code sent by the device is the code representing the starting address of the ISR of the corresponding device. Hence the address is also called as **interrupt vector.**

4.3.2 Should a device be allowed to interrupt the processor while another interrupt is being serviced

Answer to this problem is yes but with an exception. The exception is that another device is allowed to interrupt only if the priority of the device requesting interrupt is greater than the one that is being serviced currently.

For implementing this, processor requires a different type of interrupt hardware with a separate IRQ line and INTA line for each and every device. Implementing such scheme is called **interrupt nesting.**

Interrupt Nesting:



Here in this scheme, I/O devices are organized according to the priority structure and any interrupt from high priority is serviced. Device that is closer to the processer has a high priority and the one that is away from the processor has the lower priority.

Each and every device in a priority structure has a separate INTR line and INTA line and the processor has a special circuit called priority arbitration circuit which is meant for evaluating the priority of a device requested interrupt.

Here the device requests an interrupt by enabling its corresponding IRQ line. Once when the interrupt request is received by the processor, the priority arbitration circuit compares the priority of the requested device with the priority of the device that is getting serviced and if the priority of requested device is high then the interrupt request is accepted and INTA signal is sent to the corresponding device.

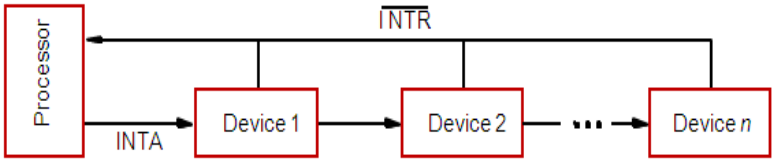
4.3.3 How to handle two or more simultaneous request

When more than one device raises the interrupt request at the same time, then it is called as **simultaneous request**.

When there is a simultaneous request, the problem is that the processor has to select only one device that needs to be serviced. There are two methods to implement the solution for this problem:

1. Daisy chain scheme
2. Priority Group

1. Daisy chain Scheme

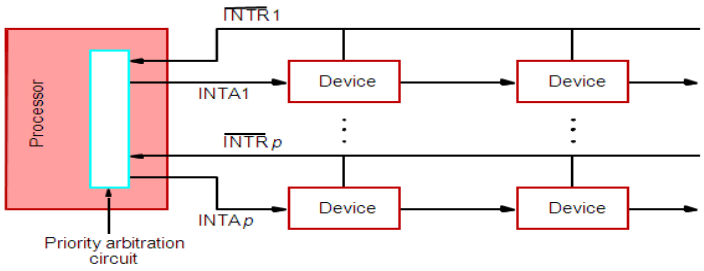


In the daisy chain scheme, I/O devices are organized according to the priority over a common interrupt request line INTR. Device that is closer to the processer has a high priority and the one that is away from the processor has the lower priority. On receiving the interrupt request, INTA signal from the processor enters each and every device in a daisy chain fashion were in the signal enters the device in a priority structure. Device that hasn't requested interrupt will allow the INTA signal to cross through and the device that requested for interrupt holds the INTA signal without allowing them to propagate further.

Disadvantage: Only disadvantage of this scheme is that, interrupt nesting cannot be implemented.

2. Priority Group

The scheme overcomes the disadvantage of the daisy chain scheme. This is basically a combination of daisy chain scheme and interrupts nesting.



In this scheme, collection of devices are grouped together under different priority and within the group, they are connected in a daisy chain fashion. Here interrupts can be nested only if the device belonging to a different group requests for the interrupt. Then the interrupt acknowledge is sent in a daisy chain fashion only if the device requested for an interrupt belongs to the higher priority group or else the interrupt is discarded. Again to evaluate the priority, processor has a priority arbitration circuit.

4.4 Exceptions

Interrupts that are caused by the software or a program are called as **exceptions.** Basically in this, the executing program is terminated by some instruction within the executing program.

4.4.1 Different types of exceptions

1. **Recovery from errors:** This type of exception is invoked when a particular instruction is executed if the executing instruction creates some unusual results, then the program will be terminated.
Example: divide by zero error

2. Debugging: Debugging is a process of finding errors in a program and the program that is used to perform this process is called as **Debugger.**
Debugger is basically an exception that uses two other exceptions:
i. **Trace mode:** In this type, the exception occurs after the execution of each and every instruction.
ii. **Break Points:** This does the similar activity but the exceptions are caused only at the required points of the program. To create an exception in the require points of the program an instruction called trap is added.

3. Privilege exceptions: These exceptions are the one that are meant for protecting the operating system of a computer. These exceptions are invoked when a user tries to change some operating system programs.

4.5 Direct Memory Access (DMA)

Direct Memory Access (DMA) is a technique that is meant for speeding up the data transfer rate between the memory unit and the external I/O device. In this technique, data transfer between main memory and external device is done without the continuous involvement by the processor.

DMA is implemented by a special controller circuit called DMA controller which is part of the I/O interface. This circuit performs the operation similar to the processor when doing data transfer between main memory and processor.

Although DMA performs the operation independently without the involvement of the processor, DMA operation has to be initiated by the processor by sending the following information:
1. Starting address
2. Number of words in a block
3. Direction of the transfer (Read/write)

When the entire block of data is transferred, DMA controller then informs the processor by raising an interrupt.

4.5.1 Cycle stealing & Block Mode

Cycle Stealing: Requests from DMA devices for using the processor bus is always given a higher priority. Even if most of the memory access is done by the processor, DMA overrides the processor and uses the bus. Hence the concept is termed as cycle stealing.

Block Mode (Burst Mode): When doing the block transfer, DMA controller is given exclusive access to the main memory top transfer the data block without any interruption from any devices and this process is called **Block mode** or **Burst mode.** When doing the DMA transfer it enters into this mode.

4.5.2 Bus Arbitration

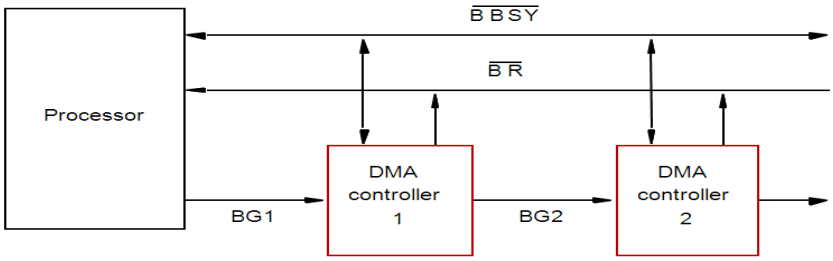
Any device that is allowed to initiate data transfer on a processor bus is termed as **bus master.** Bus arbitration is a process by which next device to become the bus master is selected.

Bus arbitration process is broadly classified into two methods:
1. Centralized arbitration
2. Distributed arbitration.

4.5.2.1 Centralized Arbitration

In this technique processor takes the responsibility of selecting the device to become a bus master. Processor is the bus master unless it grants mastership to nay DMA controllers.

Hardware arrangement to implement this technique is shown below:



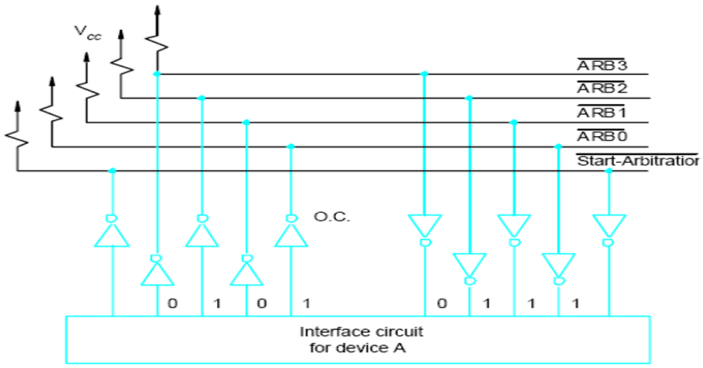
Any device that wants to initialize DMA indicates that it needs to become bus master by activating a common bus request line BR. When bus request line is activated, processor sends bus grant signal BG in a daisy chain fashion from one device to another device. Device that requested to become bus master will hold BG signal and doesn't allow propagating further. Device that captured BG signal checks if the bus is busy through BBSY line and initiates DMA transfer when it becomes free and activates BBSY line informing other devices that it is implementing DMA transfer.

4.5.2.2 Distributed Arbitration

In this technique, devices that wants to become bus master involves in the process of bus arbitration were in the arbitration process is carried out by competing method and the winning device becomes the bus master.

Here in this scheme, each device is given a unique identification number and the device that wants to become a bus master activates start arbitration line and sends its identification number on the arbitrary lines and then the interface circuit reads the identification number of different devices and selects the winner by replying with its identification number.

Implementation of distributed arbitration system circuit is shown below for the device bearing a 4 bit identification number

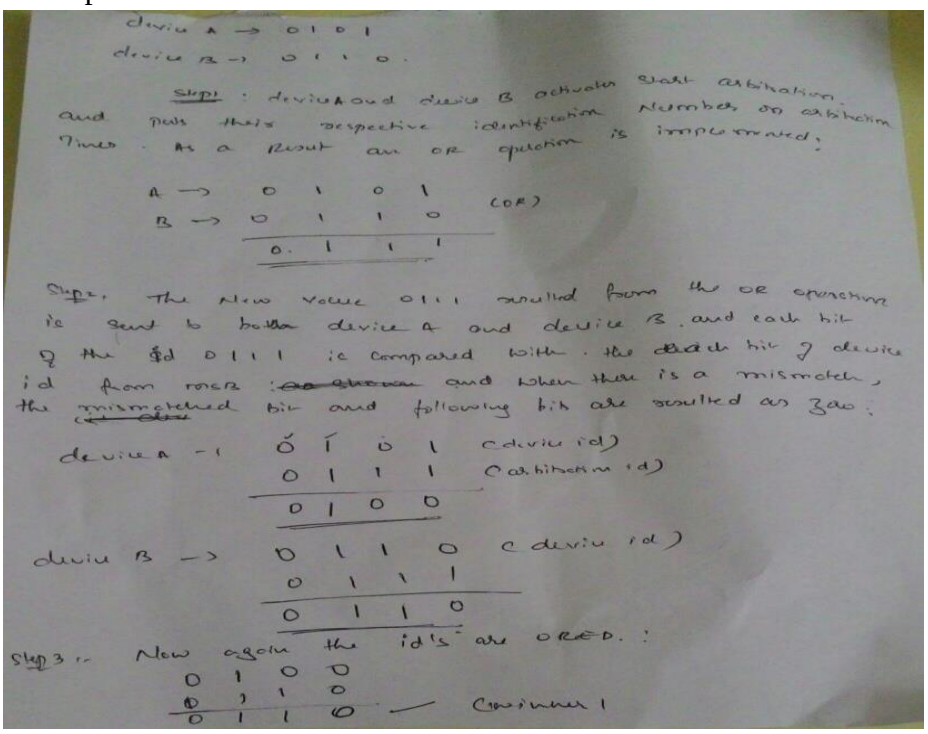


Here in this case the circuit consists of four lines from ARB0 to ARB3 called as arbitration line and these lines are active low lines has one output and one input connection going into each

device. Apart from that the circuit has one more active low line called start arbitration and even to this line one input and one output line is connected from each and every device.

We will understand the working of this circuit through an example:

Consider each device in a system has a 4 bit identification number and let's assume that device A has identification number 0101 and device B has identification number 0110. Following are the steps of arbitration process:



(Refer notes for the problem solved in the class)

4.6 Buses

A processor, main memory and I/O devices are interconnected by means of a medium called bus which is meant for the transfer of data to/from the processor. Bus even includes lines to support interrupts and arbitration process as discussed earlier. The behavior of the bus is governed by the set of rules which are called as **bus protocols.** Bus protocols are the one that will decide when to place information or assert control signal and so on.

As we discussed earlier, bus lines for transferring data is grouped into three types:

1. Address lines
2. Data lines
3. Control lines

Control lines are the one that is meant for specifying the type of operation and the size of data that needs to be transferred. Apart from this, they also carry very important information called

timing signals which specify the time entity for I/O device and processor for placing the data over the bus. Based on the timing entity, there are two types of buses:

i. Synchronous bus

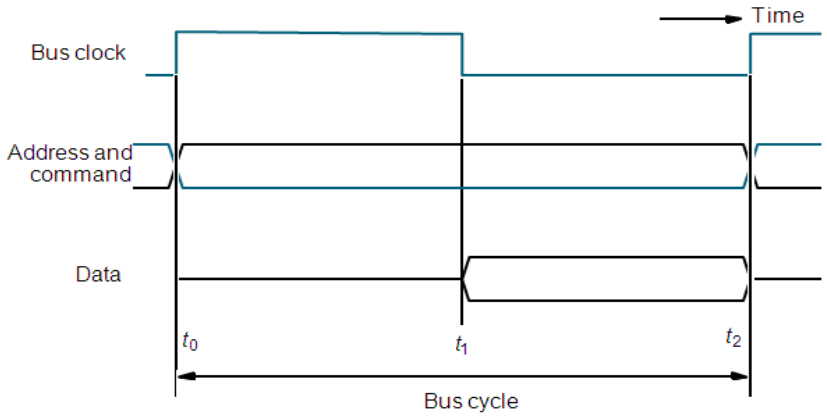
ii. Asynchronous bus

Any device that initiates data transfer on the bus is called as **master** and the device that responds for the transfer is called as **slave.**

4.6.1 Synchronous Bus

In synchronous bus, all the devices connected across the process derive timing information from a common clock line called **bus clock.** In this type of bus, each of the time interval constitute to one cycle during which one data transfer takes place in the bus.

In this type of bus, sequence of events is fixed and takes place in a fixed time entities. Below represented timing diagram gives a very clear picture on the sequence of events taking place for the input operation.



Following are the sequence of actions taking place at different time instances:

t_0: At this time, Master places the device address and command on the bus, and indicates that it is a Read operation.

t_1: Addressed slave places data on the data lines

t_2: Master "strobes" the data on the data lines into its input buffer, for a Read operation.

Main drawback of this system is that the common clock signal must be designed such that, all the devices connected across the system must complete the operation in one clock cycle. This means that all devices are not compatible with this system.

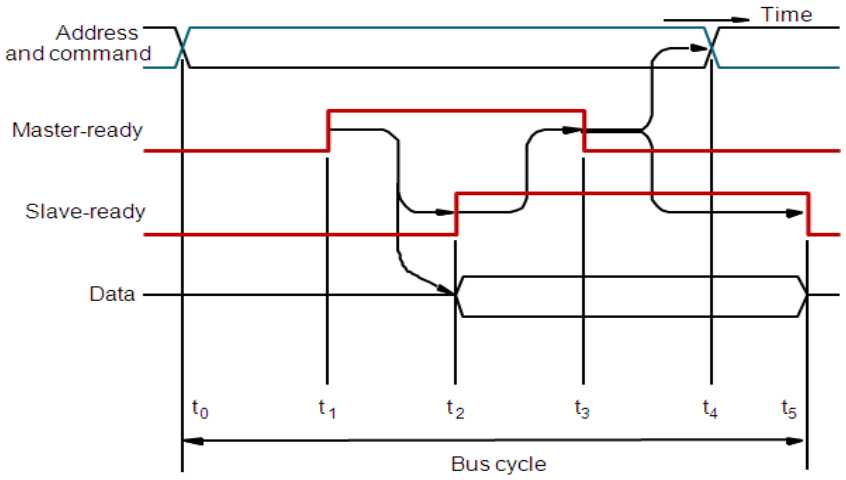
4.6.2 Asynchronous Bus

This system is being replaced to overcome the drawback of synchronous bus system where there is no common clock to control the timing signals of the device rather than they have their own independent timing signals to perform the operation.

In Asynchronous bus, there is no central timing signal to control and co-ordinate the activities of the I/O devices. Asynchronous bus use the concept of handshake protocol between the master device and the slave device were the operations are performed by response to every request.

In this case, the common clock signal of synchronous bus is replaced by two signals, master ready and slave ready signal and operations are carried out according to the arrival of these signals.

To get a better understanding on this we will look into an example of read operation carried out in the asynchronous bus system.



Following are the series of actions performed during the data transfer using handshake protocol:

t_0 - Master places the address and command information on the bus.

t_1 - Master asserts the Master-ready signal. Master-ready signal is asserted at t_1 instead of t_0

t_2 - Addressed slave places the data on the bus and asserts the Slave-ready signal.

t_3 - Slave-ready signal arrives at the master.

t_4 - Master removes the address and command information.

t_5 - Slave receives the transition of the Master-ready signal from 1 to 0. It removes the data and the Slave-ready signal from the bus.

4.7 Interface Circuit

Circuit connecting I/O device to the processor bus is called as **Interface circuit.** On one side it is connected to the processor bus and on the other side, it is connected to the device. Basically acts as a bridge between I/O device and processor bus.

Interface circuit part connecting to the processor bus as three set of wires:

1. Address lines
2. Data lines
3. Control lines

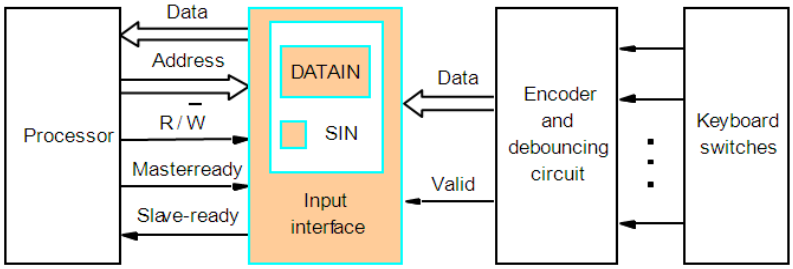
Another side of the interface circuit connecting to the I/O device has a data line and control lines associated with the device and this side is collectively called as **port** of the device. Ports are classified into two types:

1. Parallel port: transfers data as a collection of bits at a time, normally 8 to 16 bits.
2. Serial Port: transfers data bit by bit

4.7.1 Parallel Port

4.7.1.1 Parallel Port for input interface circuit

To know the working of parallel port for input interface circuit, let us consider the example of keyboard to processor connection shown below:



Construction:

As shown in the diagram, input interface circuit is connected to processor on one side and to the keyboard on another side. On the processor side, it has data lines, Address lines, control lines which sends the signals by asynchronous system. Then on the device side, it is connected to data line (input) and a valid line.

The interface circuit has one register DATAIN and a single bit element SIN.

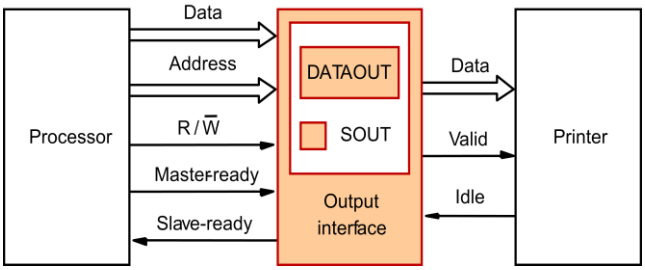
Working:

On the keyboard side of the interface, encoder circuit which generates a code for the key pressed and the debouncing circuit eliminates the effect of a key bounce. Data lines contain the code for the key. Valid line changes from 0 to 1 when the key is pressed. This causes the code to be loaded into DATAIN and SIN to be set to 1. Then the processor sends the master ready signal indicating that it is ready to transfer the data and places data on the data lines. Then processor

starts strobing the data by keeping the slave ready signal active. Once the operation is completed the SIN is again set to 0 indicating that the operation is performed and there are no more data available on the keyboard.

4.7.1.2 Parallel Port for output interface circuit

To know the working of parallel port for output interface circuit, let us consider the example of printer to processor connection shown below:



Construction:

As shown in the diagram, output interface circuit is connected to processor on one side and to the keyboard on another side. On the processor side, it has data lines, Address lines, control lines which sends the signals by asynchronous system. Then on the device side, it is connected to data line (input) and a valid line.

The interface circuit has one register DATOUT and a single bit element SOUT

Working:

On the printer side, Idle signal line is activated when it is ready to accept printing job. This causes the SOUT flag to be set to 1 indicating the processor that printer is ready to take the task. Processor then places a new character into a DATAOUT register. Valid signal is activated by the interface circuit informing the printer that the printing character is ready. Printer then access the data over the data line and asserts SOUT to 0 till it completes the printing operation, indicating the processor that it is busy.

4.8 Standard I/O Interfaces

What are standard I/O interfaces?

We know that I/O device is connected to a computer using an interface circuit and there are several alternatives to design the processor bus. This means that interface circuit designed for one computer system may not be compatible with another computer system. So to rule out this problem, the practical solution is to develop an interface circuit with standard signals and protocols that is compatible with all the computer system. This is called as standard I/O interfaces.

Basically a personal computer has a motherboard which houses the processor chip, main memory and some I/O interfaces and a few connectors into which additional interfaces can be plugged.

Due to some electrical reasons, all the deices cannot be connected to the processor bus directly. So devices which require high-speed connection to the processor are connected directly to the processor bus and the remaining devices are connected to the processor bus via other bus called expansion bus. Expansion bus is connected to a processor bus via circuit called **bridge**. Bridge basically transfers signals and protocols of processor bus into the expansion bus.

There are three major standards defined for expansion bus:

1. PCI (Peripheral Component Interconnect)
2. SCSI (Small Computer System Interface)
3. USB (Universal Serial Bus)

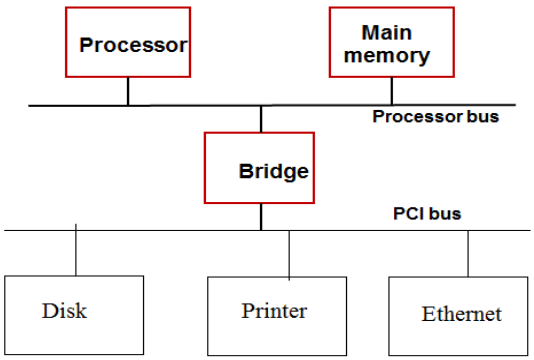
We will be discussing each one of them in detail

4.8.1 Peripheral Component Interconnect (PCI) Bus

PCI bus standard was introduced in 1992 and following are the very important features of this bus:

1. Low cost bus
2. Processor independent
3. Plug and play capability : devices can be connected directly without installing their drivers manually
4. Supports burst mode
5. Supports three independent address spaces:
   i. Memory address
   ii. I/O address
   iii. Configuration address

Schematic shown below gives a model configuration of PCI bus and the devices connected across it



Data transfer signals: Following are the data transfer signals

Name	Function
CLK	A 33-MHz or 66-MHz clock.
FRAME#	Sent by the initiator to indicate the duration of a transaction.
AD	32 address/data lines, which may be optionally increased to 64.
C/BE#	4 command/byte-enable lines (8 for a 64-bit bus).
IRD Y#, TRD Y#	Initiator-ready and Target-ready signals.
DEVSEL#	A response from the device indicating that it has recognized its address and is ready for a data transfer transaction.
IDSEL#	Initialization Device Select.

Device Configuration

It is a process carried out by PCI bus to assign address to the new devices connected across the PCI bus. When the system is powered on or reset, PCI runs a program called **initialization process.** During this process PCI scans all the lines of configuration space and assigns the address for the device and stores it inside its ROM called as **configuration ROM.**

4.8.2 SCSI Bus

The acronym SCSI stands for Small Computer System Interface and very often pronounced as skuzzy. It is basically a expansion bus meant for connecting the secondary storage devices such as hard disk, optical disk etc. The SCSI bus standard has undergone many revisions, and its data transfer capability has increased very rapidly, almost doubling every two years. SCSI-2 and SCSI-3 have been defined, and each has several options.

Devices connected to the SCSI bus are not part of the address space of the processor

The SCSI bus is connected to the processor bus through a SCSI controller. This controller uses DMA to transfer data packets from the main memory to the device, or vice versa. A packet may contain a block of data, commands from the processor to the device, or status information about the device. A controller connected to a SCSI bus is one of two types – an initiator or a target. An initiator has the ability to select a particular target and to send commands specifying the operations to be performed. The disk controller operates as a target. It carries out the commands it receives from the initiator.

Following are the sequence of actions carried out by SCSI bus for transferring data from disk into processor:

1. The SCSI controller, acting as an initiator, contends for control of the bus.
2. When the initiator wins the arbitration process, it selects the target controller and hands over control of the bus to it.
3. The target starts an output operation (from initiator to target); in response to this, the initiator sends a command specifying the required read operation.
4. The target, realizing that it first needs to perform a disk seek operation, sends a message to the initiator indicating that it will temporarily suspend the connection between them. Then it releases the bus.
5. The target controller sends a command to the disk drive to move the read head to the first sector involved in the requested read operation. Then, it reads the data stored in that sector and stores them in a data buffer. When it is ready to begin transferring data to the initiator, the target requests control of the bus. After it wins arbitration, it reselects the initiator controller, thus restoring the suspended connection.
6. The target transfers the contents of the data buffer to the initiator and then suspends the connection again
7. The target controller sends a command to the disk drive to perform another seek operation. Then, it transfers the contents of the second disk sector to the initiator as before. At the end of this transfers, the logical connection between the two controllers is terminated.
8. As the initiator controller receives the data, it stores them into the main memory using the DMA approach.
9. The SCSI controller sends as interrupt to the processor to inform it that the requested operation has been completed

Main Phases involved in SCSI

1. Arbitration
2. Selection
3. Information transfer
4. Reselection

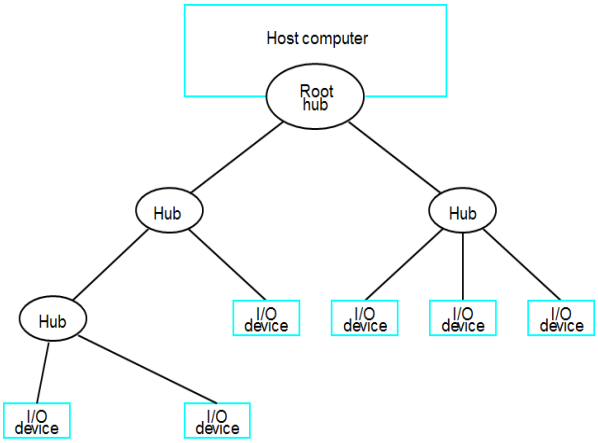
4.8.3 Universal Serial Bus (USB)

USB is another expansion bus meant for connecting several I/O devices of different types. They are called as universal bus because this type of bus has compatibility of all kind of I/O devices. It even includes different audio, video, internet devices etc. One of the very important feature of this bus is that it can be extended to any level using tree structure method.

It is being designed to meet the following key objectives:

1. To provide simple, low cost and easy to interconnect system that even overcomes the limitations of I/O ports available
2. Can accommodate wide variety of data transfer characteristics like audio, video , internet etc.
3. Plug and Play facility.

USB tree structure



➔ To accommodate a large number of devices that can be added or removed at any time, the USB has the tree structure as shown in the figure.

➔ Each node of the tree has a device called a hub, which acts as an intermediate control point between the host and the I/O devices. At the root of the tree, a root hub connects the entire tree to the host computer.

➔ In normal operation, a hub copies a message that it receives from its upstream connection to all its downstream ports. As a result, a message sent by the host computer is broadcast to all I/O devices, but only the addressed device will respond to that message. However, a message from an I/O device is sent only upstream towards the root of the tree and is not seen by other devices. Hence, the USB enables the host to communicate with the I/O devices, but it does not enable these devices to communicate with each other.

Addressing in USB

➔ Each device on the USB, whether it is a hub or an I/O device, is assigned a 7-bit address. This address is local to the USB tree and is not related in any way to the addresses used on the processor bus.

➔ A hub may have any number of devices or other hubs connected to it, and addresses are assigned arbitrarily.