UNIT-VI

Input - Output Organization

Peripheral Devices :-

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

Devices that are under the direct control of the computer are said to be connected on-line. These devices are designed to read information into or out of the memory unit upon command from the CPU and are considered to be part of the total computer system.

Input or output devices attached to a pc computer are also called 'peripherals'. There are three types of peripherals 1) input 2) output 3) Input - output. These peripherals may be analog or digital and serial or parallel.

The common peripherals are keyboards, display units and printers.

peripherals that provide auxiliary storage for the system are magnetic disks & tapes.

Monitor And Keyboard :-

The most commonly used peripheral is 'Video monitors'. They consist of a keyboard as the input device and a display unit as the O/p device.

There are different types of video monitors, the most popular use is Cathode Ray Tube (CRT). The CRT contains

an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube. The beam can be deflected horizontally and vertically. To produce a pattern on the screen, a grid inside the CRT receives a variable voltage that causes the beam to hit the screen and make it glow at selected spots.

A characteristic feature of display devices is a cursor that marks the position in the screen where the next character will be inserted. The cursor can be moved to any position in the screen, to a single character, beginning of a word, ending of a coord etc.

Edit keys add or delete information based on the cursor position. The display terminal can operate in a single character mode where all characters entered on the screen through the keyboard.

In the block mode, the edited text is first stored in a local memory inside the terminal. The text is transferred to the computer as a block of data.

Printer :-

printers provide a permanent record on paper of computer output data or text.

There are three different types of character printers.

1) Daisy wheel printer 2) Dot matrix 3) Laser printer.

The daisy wheel printer contains a wheel with the characters placed along the circumference. To print a character, the wheel rotates to the proper position and a magnet then presses the letter against the ribbon.

The dot matrix printer contains a set of dots along the printing mechanism. Each dot can be printed or not, is depending on the specific characters that are printed on the line.

The laser printer uses a rotating photographic drum that is used to imprint the character images. The pattern is then transferred onto paper in the same manner as a copying machine.

Magnetic tape :-

Magnetic tapes are used mostly for storing files of data.

Magnetic disk :-

Magnetic disks have high speed rotational surfaces coated with magnetic material.

Input - output Interface :-

I/O interface provides a method for transferring information between internal storage and external I/O devices.

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. so, a conversion of signal values may be required.

2) The data transfer of peripherals is usually slower than the

transfer rate of the CPU and 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 h/w components between the CPU and peripherals to supervise and synchronize all i/p & o/p transfers. These components are called 'interface' units.

In general, the word "interface" is used to the point of contact between two parts of a system.

Two main types of interface are 1) CPU interface that corresponds to the system bus.

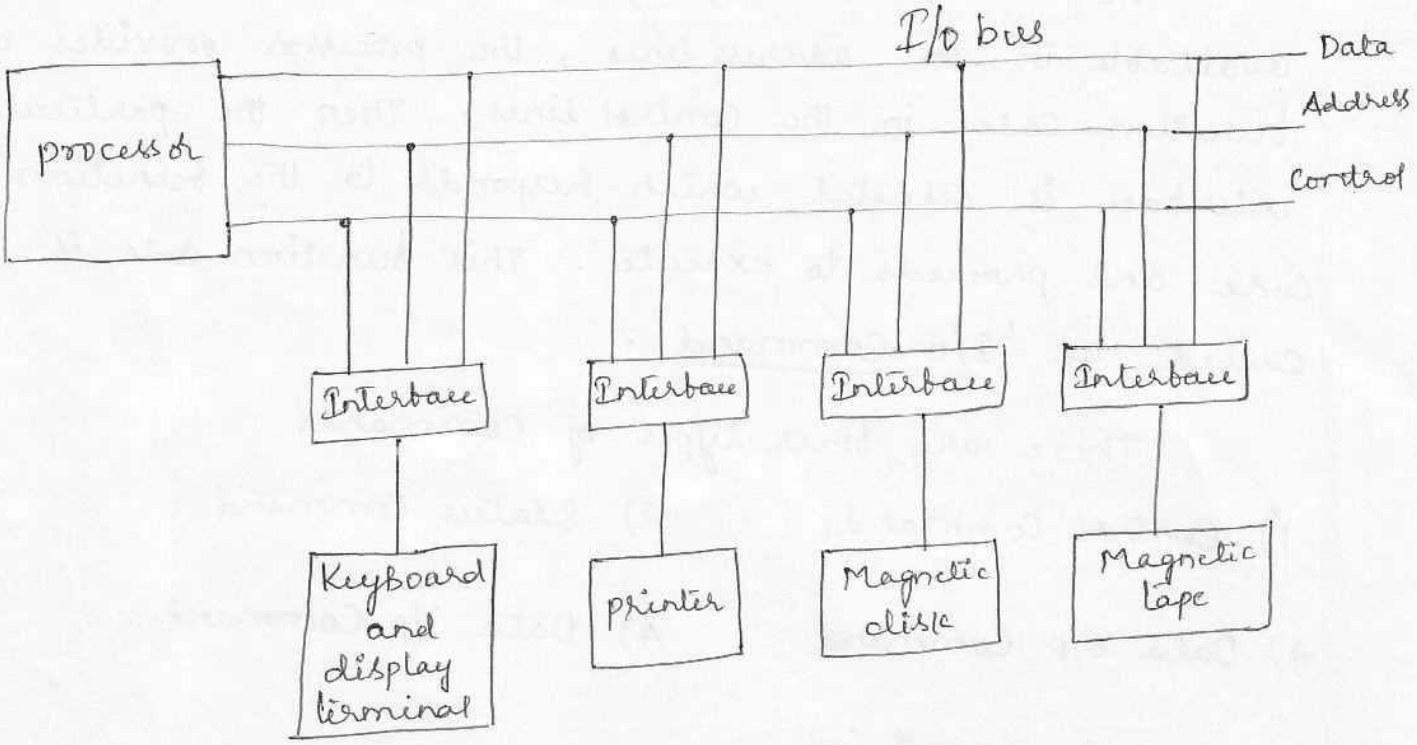
2) input-output interface that corresponds to the input-output device.

I/O Bus and interface Modules :-

The communication link between the processor and several peripherals is shown in fig.

The I/O bus consists of data lines, address lines and control lines. The magnetic disk, printer and terminal are employed in any general-purpose computer.

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 electro-mechanical device.

The I/O bus from the processor is attached to all peripheral interfaces.

To communicate with a device, the processor places a device address on the address lines. Each interface is 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, remaining others are disabled by their interface.

At that particular instance, the address is made available in the address lines, the processor provides a function code in the control lines, Then the particular interface is selected which responds to the function code and proceeds to execute. This function code is called as 'I/O Command'.

There are four types of commands

1) Control Command
2) Status Command
3) Data o/p Command
4) Data i/p Command.

Control Command :-

It is issued to activate the peripheral and to inform it what to do.

Status Command :-

It is used to test various status conditions in the interface and the peripheral.

Data o/p Command :-

It causes the interface to respond by transferring data from the bus into one of its registers.

Data i/p Command :-

The interface which 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 i/p Command. Then the interface places the data on the data lines, where they are accepted by the processor.

I/O Versus Memory Bus :-

The memory bus contains data lines, address lines 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 other for I/O.
2) use one common bus for memory and I/O with common control lines.
3) use one common bus for both memory and I/O but have separate control lines.

In method 1, the computer has independent sets of data, address and control buses i.e., one for accessing memory and the other for I/O. This method can be implemented, where we have separate I/O processor in addition to CPU.

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

The purpose of the IOP is to provide an independent pathway for transfer of information between external devices and internal devices.

The IOP is sometimes called 'data channel'.

Isolated Versus Memory-Mapped I/O :-

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 an memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is refered to as the 'isolated I/O' for assigning addresses in a common bus.

In the isolated I/O configuration, the CPU has distinct i/p and o/p instructions and each of these instructions is associated with address of an interface register.

When the CPU fetches and decodes the operation code of an i/p or o/p instruction, it places the address associated with the instruction into the common address lines. At mean time, it enables the I/O read or I/O write control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word.

When the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface register.

The second alternative, has same address space for both memory and I/O. i.e., it employs only one set of read and write signals and not distinguish between memory and I/O addresses. This configuration is referred to as memory-mapped I/O.

The assigned addresses for interface registers cannot be used for memory words, which reduce the memory address range available.

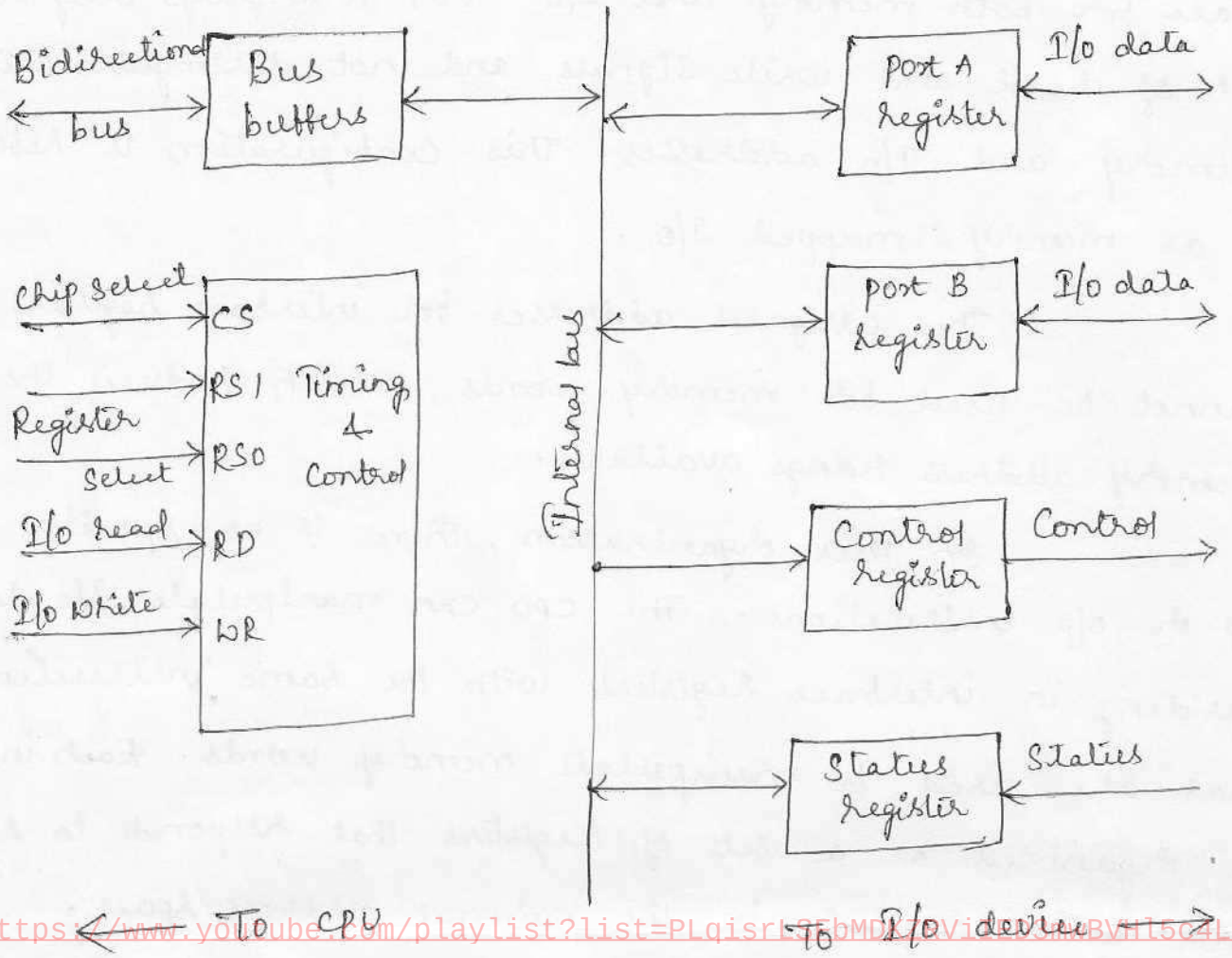
In this organization, there is no specific i/p or o/p instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words. Each interface is organized as a set of registers that respond to read and write requests in the normal address space.

The computers who implement this organization can use memory-type instructions to access I/O data. It allows the computer to use the same instructions for either i-o transfers or for memory transfers.

The advantage is that the load and store instructions used for reading and writing from memory, can be used as input and output data from I/O registers.

Example of I/O interface :-

An example of an I/O interface unit is shown in the fig.



CS	RS1	RS0	Register Selected
0	X	X	None: data bus in high impedance
1	0	0	port A register
1	0	1	port B register
1	1	0	Control register
1	1	1	Status register

The example 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 i/p or o/p 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 o/p device or with an i/p device or with a device that requires both i/p and o/p.

If the interface is connected to a printer, it will only o/p data. But a magnetic disk unit transfers data in both directions, so the interface can use bidirectional lines.

A command is passed to the I/O device by sending a word to the appropriate interface register. In a system, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register and data are transferred to and from ports A and B registers. Thus the transfer of data, control and status information is always through the common data bus.

The difference between data, control or

Status information is determined from the particular interface register with which the CPU communicates.

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.

for eg, port A may be defined as i/p port and port B may be defined as o/p port. A magnetic tape unit may be instructed to rewind the tape or to start the tape moving in the forward direction.

The bits in the Status register are used for Status Conditions and for recording errors that may occur during the data transfer.

for eg, a Status bit may indicate that port A has received a new data item from the I/O device. Another bit indicates that, there is an error occured during data transfer.

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, to detect the address assigned to the interface registers. This circuit enables the chip Select (CS) i/p when the interface is selected by the address bus.

Asynchronous Data Transfer :-

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted.

There are two different methods to achieve this.

1) Strobe 2) Handshaking

Strobe :-

The strobe pulse supplied, one of the units to indicate to the other units when the transfer has to occur.

Handshaking :-

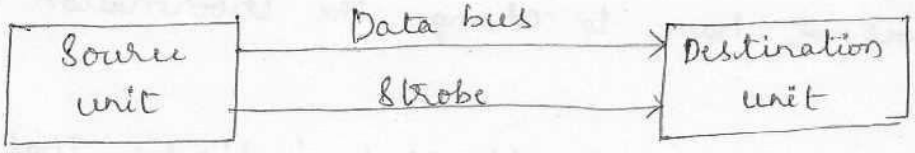
Each data item is being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units called 'Handshaking'.

Strobe Control :-

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either source or destination.

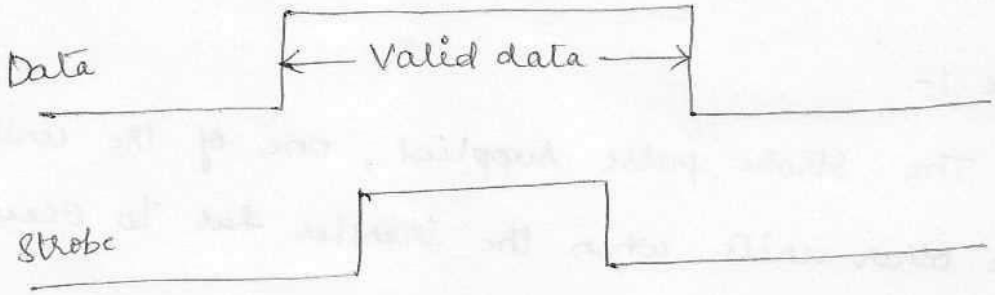
(i) Source - initiated transfer :-

The below fig. shows the source - initiated transfer.



The data bus carries the binary information from source to destination unit. The bus has multiple lines to transfer an entire byte or word. The Strobe is a single line that informs the destination when a valid data word is available in the bus.

The below shows the timing diagram.



In this,

The source unit first places data on the data bus.

The data settle to a steady value, the source activates the strobe pulse.

The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data item.

The destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.

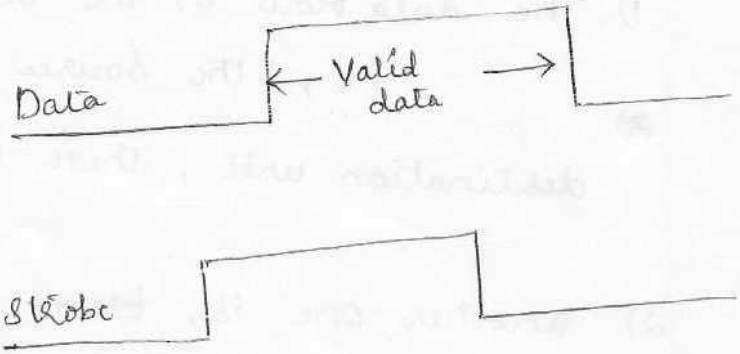
The source removes the data from the bus for a period after it disables its strobe pulse. Actually, the source does not have to change the information in the data bus.

The strobe signal is disabled indicates that the data bus

does not contain valid data.

New valid data will be available only after the strobe is enabled again.

2) Destination - initiated transfer :-



In this, the destination unit activates the strobe pulse, informing the source to provide the data.

The source unit responds by placing the requested binary information on the data bus.

The data must be valid and remain in the bus long enough for the destination unit to accept it.

The falling edge of the strobe pulse can be used again to trigger a destination register.

The destination unit then disables the strobe. The source removes the data from the bus after a predetermined time interval.

Hand shaking :-

In the strobe method, there is no reply from destination unit to source unit, whether it receives a particular data item or not.

The same happens to the destination unit, whether the source unit has actually placed the data on the bus.

The handshake method solves this problem, by introducing a second control signal that provides a reply to the unit that initials the transfer.

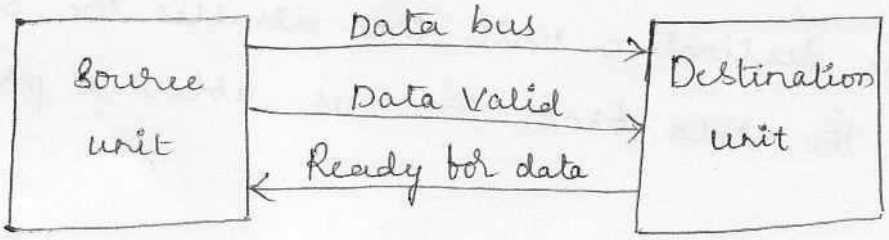
The basic principle is 'two - wire'

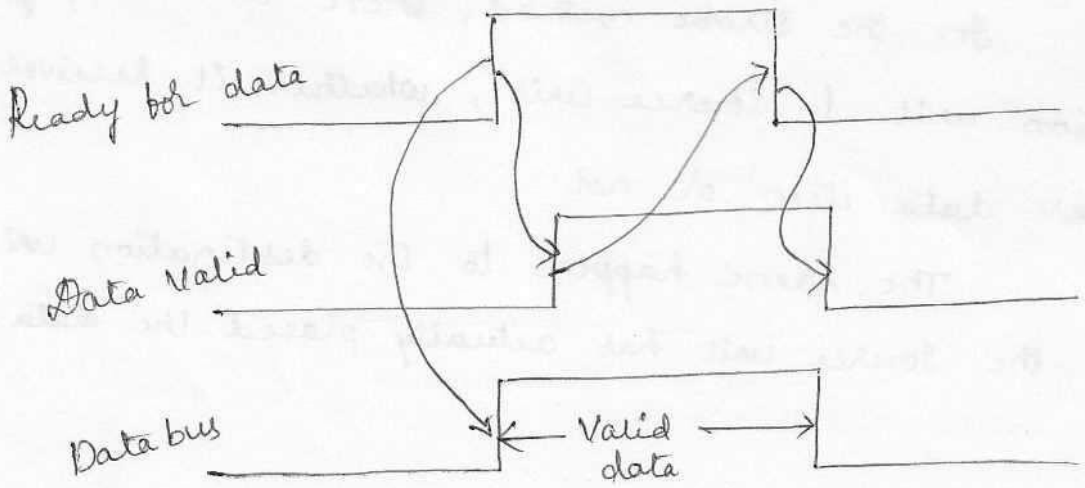
1) The data flow in the bus from source to destination.
   i.e., the source unit makes a information to destination unit, there is a valid data on the bus.

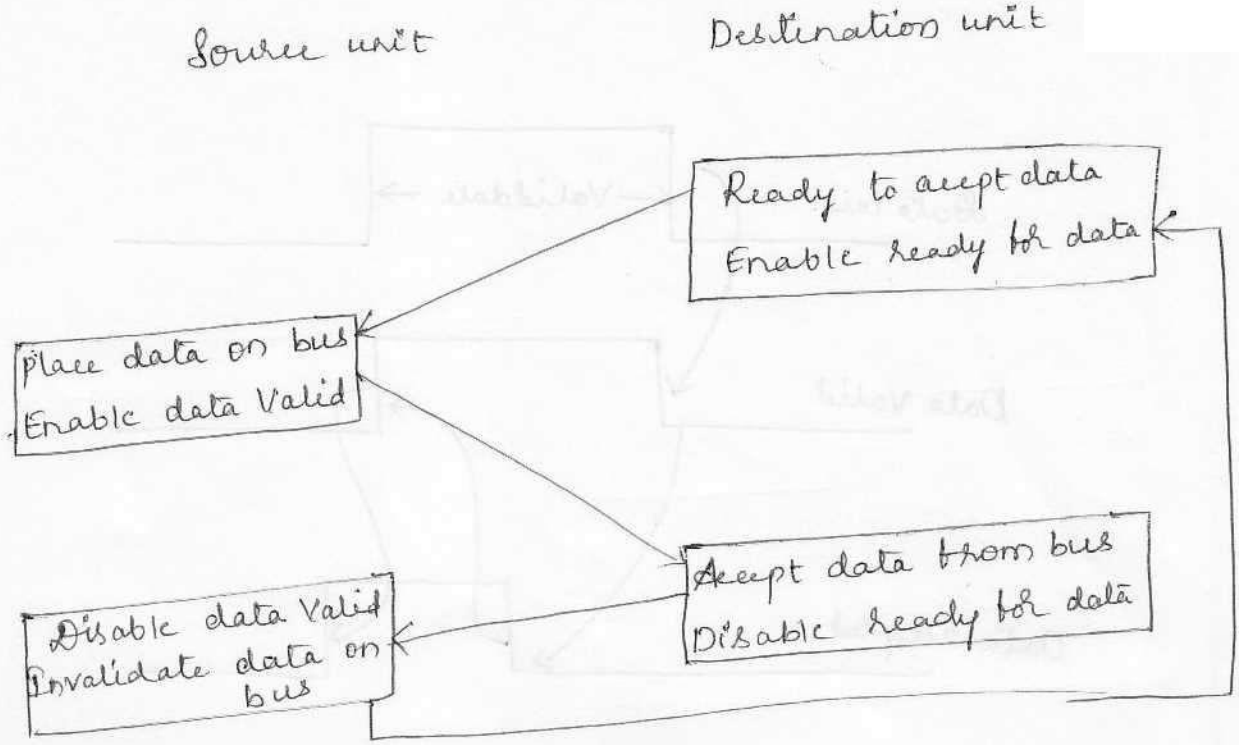
2) another one is, from destination to source.
   i.e., destination unit informs that, the data is received from the source unit.

Destination
~~Source~~ - initialed transfer using handshaking :-

The below fig shows different procedures how the data transfer exists.







Here, we are introducing two hand-shaking lines are

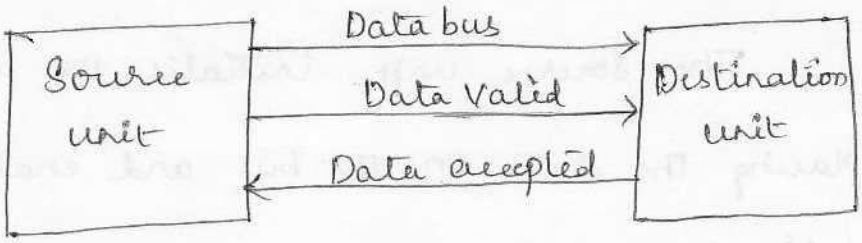
1) data valid, generated by source unit.

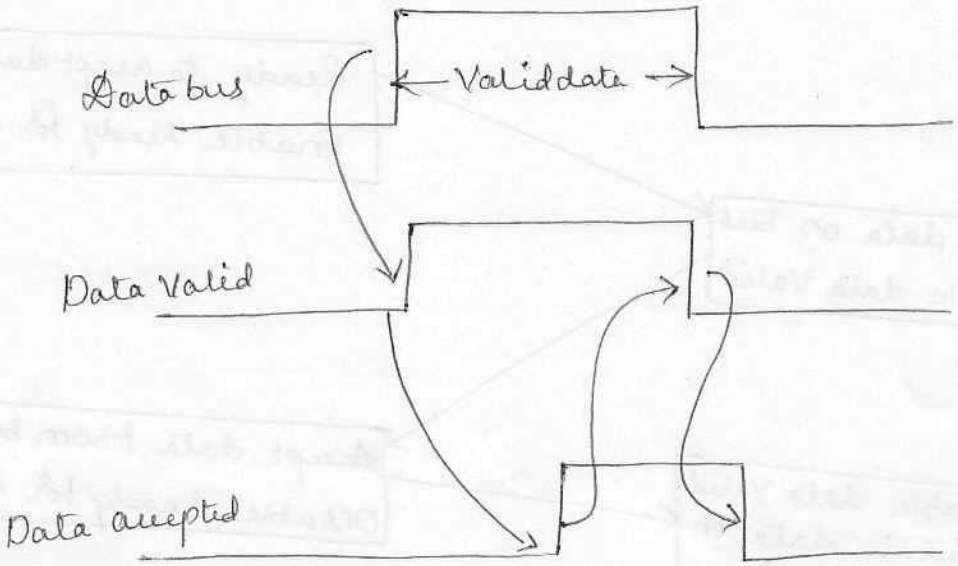
2) data accepted, generated by destination unit.

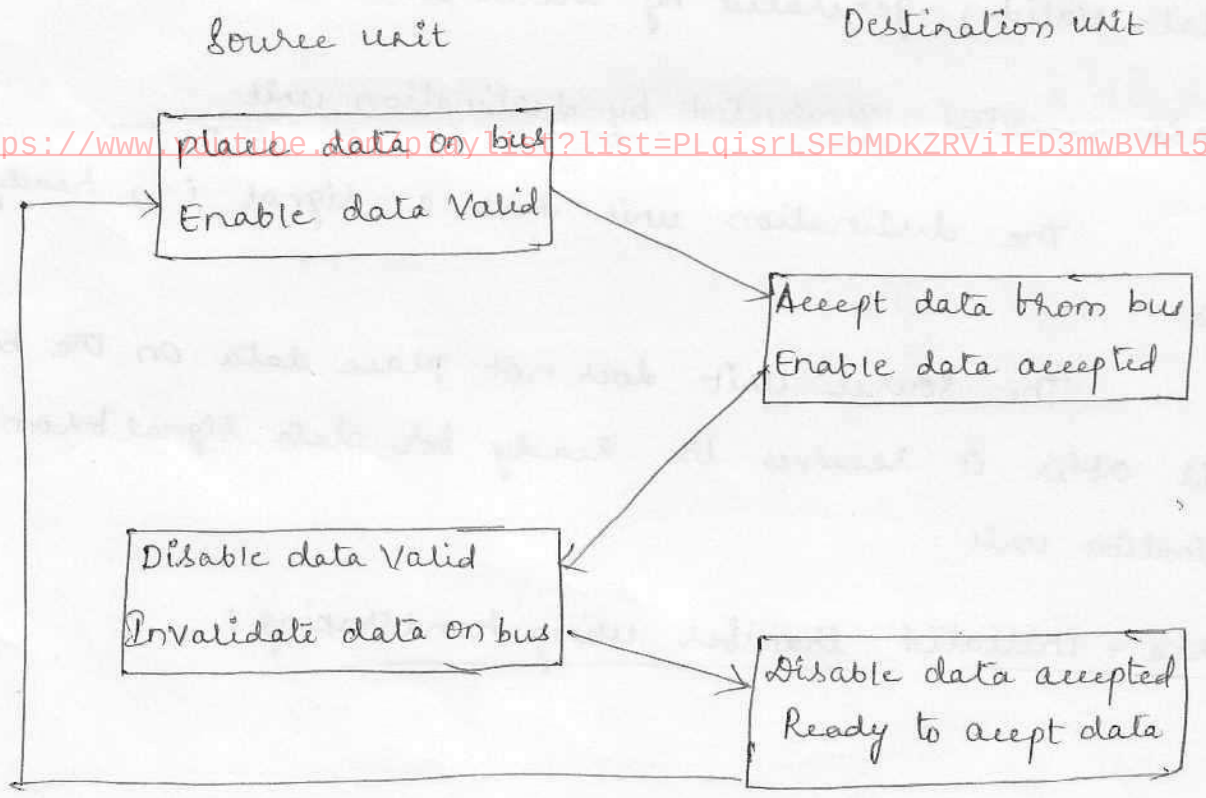
The destination unit has a signal i.e, ready for data.

The source unit does not place data on the bus until after it receives the ready for data signal from the destination unit.

Source-initiated transfer using handshaking :-







The Source unit initiates the transfer of data by placing the data on the bus and enabling its 'data Valid' signal.

The data accepted signal is activated by

The destination unit after it accepts the data from the bus. Then source unit disables its data valid signal, which invalidates the data on the bus.

The destination unit then disables its data accepted signal and the system goes into its initial state.

The source does not send the next data item until after the destination unit shows its readiness to accept new data by disabling its 'data accepted signal'.

The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units.

If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed with in a predetermined time. The time out is implemented by means of an internal clock that starts counting time when the unit enables one of its handshaking control signals.

If the return handshake signal does not respond with in a given time period, the unit assumes that an error has occured. The time out signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

Asynchronous Serial Transfer :-

The transfer of data between two units may be done in parallel or serial.

In parallel data transmission, each bit of the message has its own path and the total message is transmitted at the same time. In this, n-bit message must be transmitted through 'n' separate conductor paths.

In serial data transmission, each bit in the message is sent in sequence one at a time.

parallel data transmission is faster than the serial data transmission.

Serial transmission can be either synchronous or asynchronous.

In synchronous, the bits are transmitted continuously at the rate of clock pulses. In this, each bit is driven a separate clock of the same frequency.

In asynchronous, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted.

In a asynchronous serial Transfer, a special bits are inserted at both ends of the character code. Each character consists of three parts

1) Start bit 2) character bits 3) Stop bits.

The first bit called the start bit always starts with '0' and it indicate the beginning of a character

The last bit called the stop bit is always a '1'.

A transmitted character can be detected by the receiver from knowledge of the transmission rules.

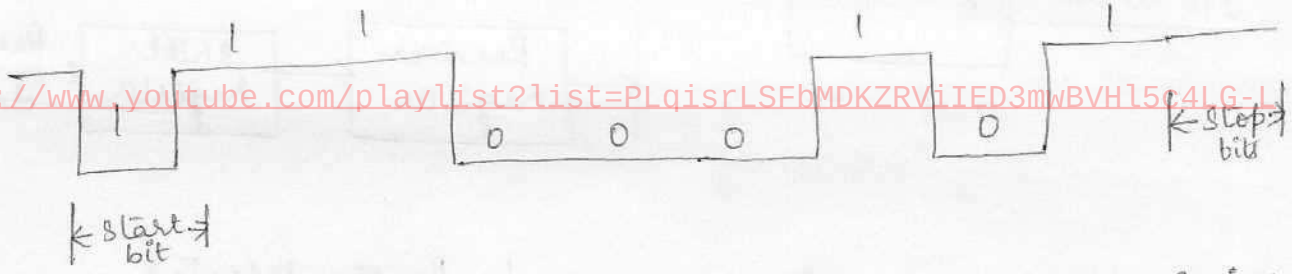
1) When a character is not being sent, the line is kept in the 1-state.

2) The initiation of a character transmission is detected from the start bit, which is always '0'.

3) The character bits always follow the start bit.

4) After the last bit of the character is transmitted, a stop bit is detected when the line returns to the 1-state for atleast one bit time.

Eg: 1 1 0 0 0 1 0 1



The 'baud rate' is defined as the rate at which serial information is transmitted and the data transfer in bits per second.

Integrated circuits are available which are specifically designed to provide the interface between computer and similar interactive terminals. Such a circuit is called an asynchronous communication interface or a universal asynchronous receiver transmitter (UART).

This byte is transferred to a shift register for serial transmission. The receiver portion receives serial information into another shift register, and when a complete data byte is accumulated, it is transferred to the receiver register. The CPU can select the receiver register to read the byte through the data bus.

The bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission.

The chip select is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls. Two registers are write-only and two are read-only.

Operation :-

The operation is initialized by the CPU by sending a byte to the control register. The initialization procedure places the interface in a specific mode of operation as it defines certain parameters such as the baud rate to use, how many bits are in each character, whether to generate and check parity and how many stop bits are appended to each character.

In status register, the two bits are used as flags. One bit is used to indicate whether the transmitter register is empty and another bit is used to indicate whether the receiver register is full.

Transmitter :-

The operation of transmitter is as follows :

- The CPU reads the status register and checks the flag to

see if the transmitter register is empty. If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.

- The first bit in the transmitter shift register is set to '0' to generate a start bit. This character is transferred in parallel from the transmitter register to shift register and appropriate stop bits are appended in the shift register. The transmitter register is empty.

- The character is transmitted one bit at a time by shifting the data in the shift register at the specified baud rate.

- The CPU can transfer another character to the transmitter register after checking the flag in the status register.

- The interface is double buffered because a new character can be loaded as soon as the previous one starts transmission

Receiver :-

- The receiver data input is in the 1-state when the line is idle.

- The receiver control monitors the receive-data line for a '0' signal to detect the occurrence of a start bit. Once a start bit has been detected, the incoming bits of the character are shifted into the shift register at the prescribed baudrate.

- After receiving the data bits, the interface checks for the parity and stop bits.

- The character with out the start and stop bits is then

transferred in parallel from the shift register to the receiver register.

- The flag in the status register is set to indicate that the receiver register is 'full'.

- The CPU reads the status register and checks the flag, and if set, it reads the data from the receiver register.

Errors :-

There are three possible errors that the interface checks during transmission are

1) parity error 2) framing error 3) Overrun error.

A parity error occurs if the number of 1's in received data is not correct parity.

A framing error occurs if the right number of stop bits is not detected at the end of the received character.

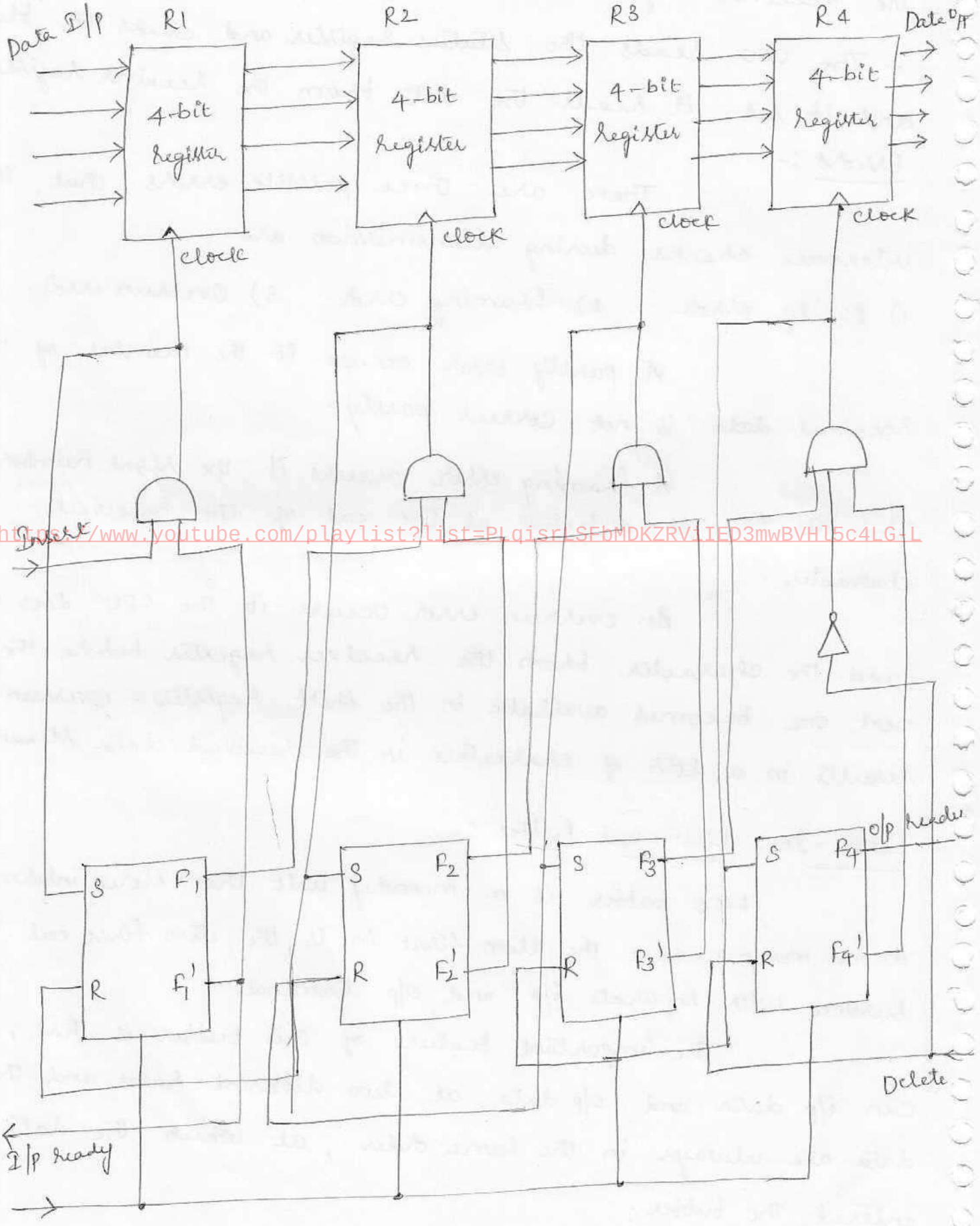
An overrun error occurs if the CPU does not read the character from the receiver register before the next one becomes available in the shift register. Overrun error results in a loss of characters in the received data stream.

First - In, First - Out Buffer :-

FIFO buffer is a memory unit that stores information in a manner that the item first in is the item first out. It survive with separate i/p and o/p terminals.

The important feature of this buffer is that, it can i/p data and o/p data at two different rates and the o/p data are always in the same order, at which the data entered the buffer.

The logic diagram of a typical 4x4 FIFO buffer is shown in fig.



It consists of four 4-bit registers R_I, $I = 1,2,3,4$ and a control register with flip-flops F_i, $i = 1,2,3,4$ one for each register.

The FIFO can store four words of four bits each. The no. of bits per word can be increased by increasing the no. of bits in each register and the no. of words can be increased by increasing the number of registers

A flip-flop F_i in the control register that is set to '1' indicates that a 4-bit data word is stored in the corresponding register R_I.

A '0' in F_i indicates that the corresponding register does not contain valid data.

The control register directs the movement of data through the registers.

Whenever the F_i bit of the control register is set ($F_i = 1$) and the F_{i+1} bit is reset ($F'_{i+1} = 1$), a clock is generated causing register $R(I+1)$ to accept the data from register R_I.

The clock transition sets F_{i+1} to '1' and resets F_i to '0'. This causes the control flag to move one position to the right together with the data

Data are inserted into the buffer provided that the input ready signal is enabled. This occurs when the first control flip-flop F_1 is reset, indicating that the register R_1 is empty.

Data are loaded from the I/p lines by enabling the clock in R_1 through the 'insert' control line. The same clock

Sets F_1, which disables the i/p ready control, indicating that the FIFO is now busy and unable to accept more data.

The ripple-through process begins provided that R2 is empty. The data in R1 are transferred into R2 and f_1 is cleared. This enables the i/p ready line, indicating that the I/p's are now available for another data word.

If the FIFO is full, F_1 remains set and the i/p ready line stays in the '0' state.

The data falling through the registers stack up at the o/p end. The o/p ready control line is enabled when the last control flip-flop F_4 is set, indicating that there are valid data in the o/p register R4.

The o/p data from R4 are accepted by a destination unit, which then enables the 'delete' control signal. This resets F_4 causing o/p ready to disable, indicating that the data on the o/p are no longer valid.

MODES OF TRANSFER

Data transfer between the central computer and I/O devices may be handled in a variety of modes. 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

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 . The transfer is to and from a CPU register and peripheral.Other instructions are needed to transfer the data to and from CPU and memory . Transferring data under program control requires constant monitoring of the peripheral by the CPU.

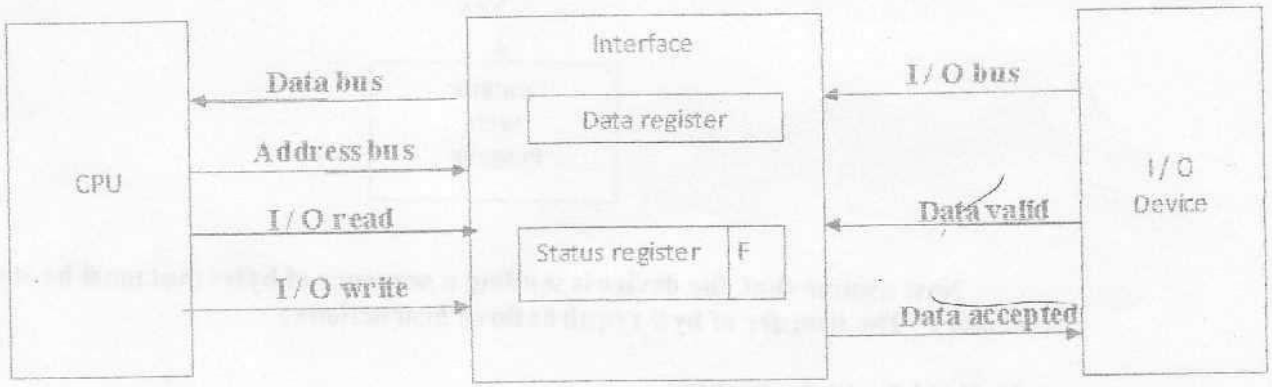
In the Programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer.

This is a time-consuming process since it keeps the processor busy needlessly

Example:

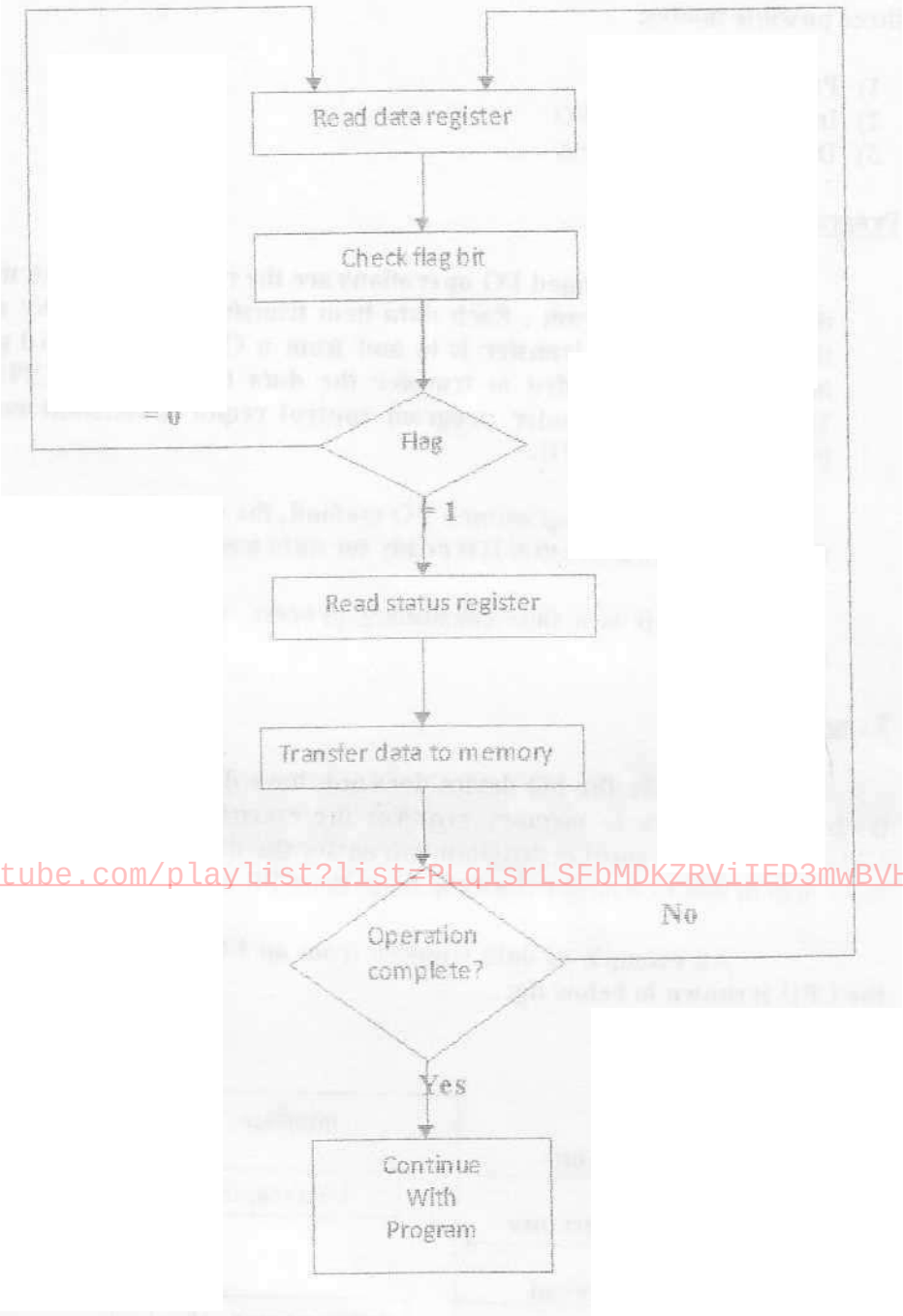
In this, the I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and store instruction to transfer the data from the CPU and memory.

An example of data transfer from an I/O device through an interface into the CPU is shown in below fig:



The device transfers bytes of information one at a time .When a byte of data is available , the device places it in the I/O bus and enables the data valid line. The interface accepts the byte into its data register and enables the data accepted line . The interface sets a bit in the status register as F bit. The device can disable the data valid line ,but it will not transfer another byte until the data accepted line is disabled by the interface .

The flow chart of the program that must be written for the CPU is shown below:



Now assume that the device is sending a sequence of bytes that must be stored in memory . The transfer of byte requires three instructions :

1) Read the Status register .
2) Check the status of the flag bit and branch to step 1 if not set or to step3 if set.
3) Read the data register.

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 .

This method is used in small low-speed computers.
Here the CPU is wasting time while checking the flag instead of doing some other useful processing task.

Interrupt-Initiated I/O :

Programmed I/O is an time-consuming process ,so in order to keep CPU in a busy state introduce an interrupt and special commands to inform the interface to issue an interrupt request signal when data are available from the device. In the mean time ,the CPU can proceed to execute another program

When the device is ready for data transfer , it generates an interrupt request to the computer.

When there is an external interrupt signal is generated ,the CPU will stops the execution of the original program and branches to service program to process the I/O transfer and then returns back to the original program.

The CPU responds to the interrupt signal by storing the return addresses from program counter into a memory stack and then control branches to a service routine that processes the required I/O transfer.

The way the processor chooses the branch address of the service routine varies from one unit to another. There are two methods for using this:

1) Vectored interrupt
2) Non Vectored interrupt

In a nonvectored interrupt , the branch address is assigned to fixed location in memory.

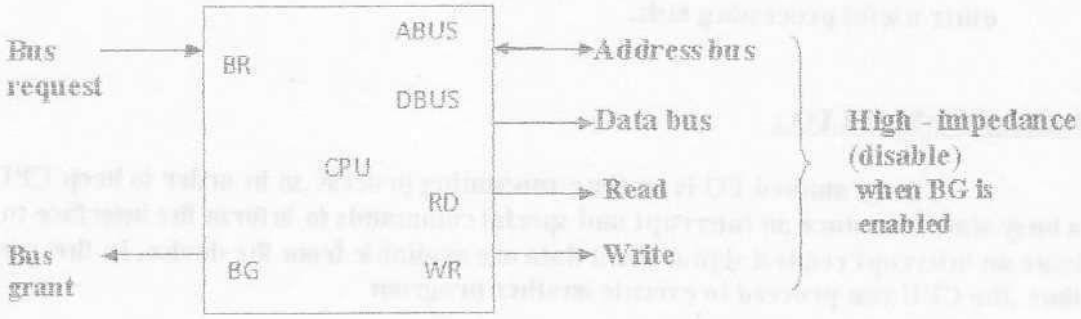
In vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector.

Direct Memory Access (DMA) :

The transfer of data between a fast storage device such as magnetic disk is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called DMA. During DMA transfer,the cpu is idle and has no control of the memory buses . A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be in idle state in many ways. One method is used in microprocessors is to disable the buses through special control signals.

The below fig. shows the control signals of the CPU to facilitate the DMA transfer.



The bus request (BR) input is used by the DMA controller to request the CPU to get control of buses.When this is input is active ,the CPU terminates the execution of the current instruction and places the address bus, data bus and read & write lines into the high-impedance state.

The CPU activates the bus grant(BG) output to inform the external DMA that the buses are in a high-impedance state. Then the DMA will take over the control of the buses with out any processor intervention. When the DMA terminates the transfer,it disables the bus request line.the CPU disables the bus grant,takes control of the buses and returns to its normal operation.

When the DMA takes control of the bus system,it communicates directly with the memory. The transfer can be done in different ways:

1) Burst Transfer: In this , a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of all the buses. This Mode of transfer can be used by fast device such as magnetic disks etc.

2) Cycle stealing : In this , the DMA controller to transfer one data word at a time ,after which it must return control of the buses to the CPU .The CPU delays its operation for one memory cycle to allow the direct memory I/O transfer to steal one memory cycle.

DMA Controller:

The below figure shows the DMA controller. The communication with the CPU via the data bus and control lines.

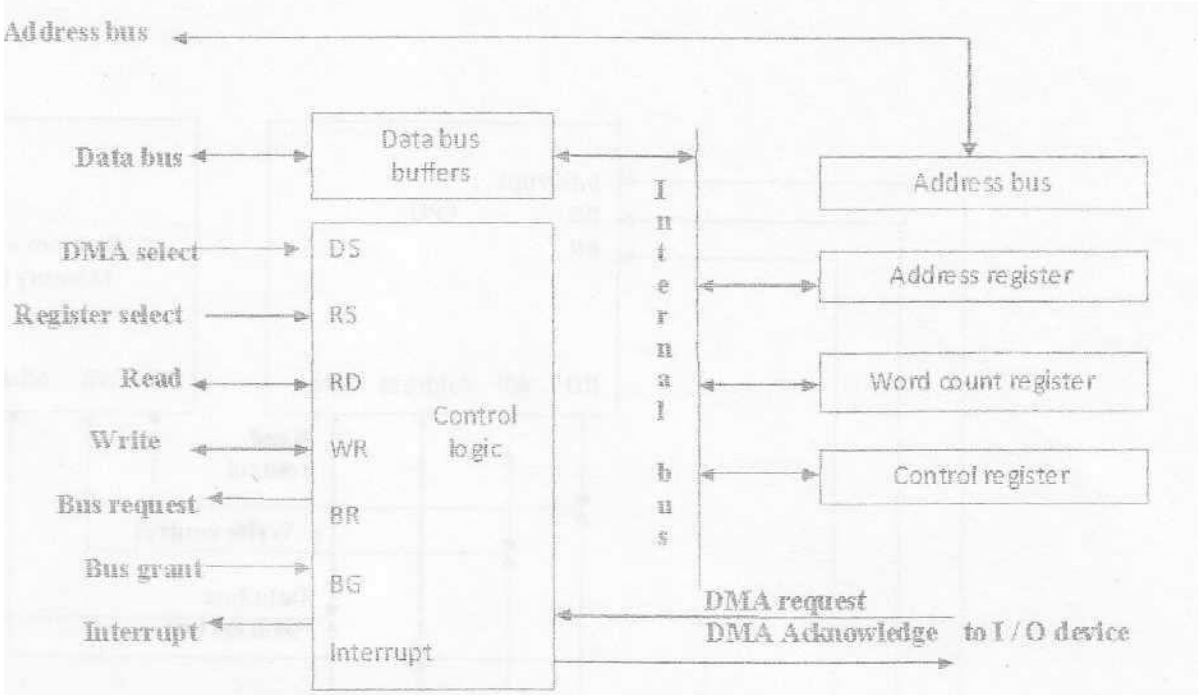
The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs . The read and write inputs are bidirectional . When the BG(bus grant) input is 0 , the CPU communicate with the DMA registers through the data bus to read from or write to the DMA registers

When BG=1 , the CPU has the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.

The DMA communicates with the external peripheral through the request and acknowledge lines.

The DMA controller has three registers:

1) address register
2) word count register
3) control register.



The address register contains an address to specify the desired location in memory. The address bits go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.

The word count register holds the number of words to be transferred. This register is decremented by one after each word is transferred.

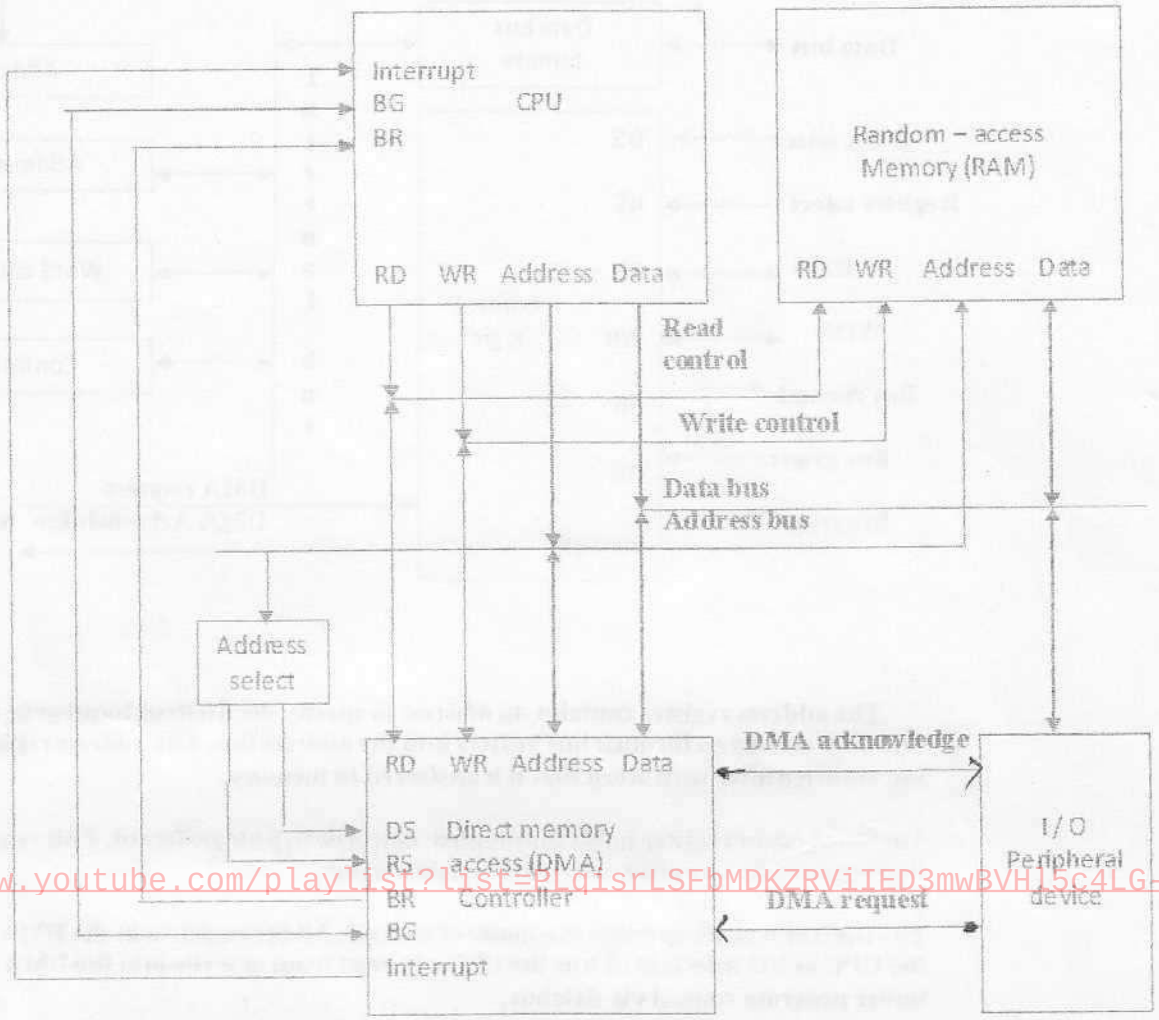
The control register specifies the mode of transfer. All the registers in the DMA appear to the CPU as I/O interface. Thus the CPU can read from or write into the DMA registers under program control via databus.

The CPU initializes the DMA by sending the following information through the databus:

1) The starting address of the memory block where data are available(for read) or where data are to be stored(for write)
2) The word count, which is the number of words in the memory block.
3) Control to specify the mode of transfer such as read or write.
4) A conrol to start the DMA transfer.

DMA transfer:

The position of the DMA controller among the other components in a system is illustrated is shown in below fig::



The CPU communicates with the DMA through the data and address buses with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus.

Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.

When the peripheral devices sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses.

The CPU responds with its BG line, informing the DMA that its buses are disabled. The DMA then puts the current value of its address register into the address bus, initiates the RD & WR signal, and sends a DMA acknowledge to the peripheral device.

Both RD and WR lines in the DMA Controller are bidirectional. The direction of transfer depends on the status of the BG line. When BG=0, the RD and WR are i/p lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are o/p lines from the DMA Controller to the random-access memory to specify the read or write operation for the data.

When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read). Thus the DMA controls the read & write operations and supplies the address for the memory.

The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.

For each word that is transferred, the DMA increments its address register and decrements its word-count register. If the word count does n't reach zero, the DMA checks the request line coming from the peripheral.

For a high speed device, the line will be active as soon as the previous transfer is completed. A second transfer is initiated, and the process continues until the entire block is transferred.

If the peripheral device's speed is slow, then the DMA request line will become late. In this case, the DMA disables the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses again.

If the word count register reaches zero, the DMA stops the transaction and removes its bus request.

It also informs the CPU of the termination by means of an interrupt. Now, the CPU responds to the interrupt, it reads the content of the word count register. The zero of this register indicates that all the words were transferred successfully.

If the DMA controller has more than channel. Each channel has a request and acknowledge pair of control signals which are connected to separate peripheral devices.

Each channel also has its own address register and word count register with in the DMA controller. A priority among the channels may be established so that the channels with high priority are serviced before channels with lower priority.

It is used for fast transfer of information between magnetic disks and memory.

Input - Output Processor (IOP) :—

A computer may incorporate one (or) more external processors and assign them the task of communicating directly with all I/O devices, instead of having each interface communicate with the CPU.

An Input-Output Processor (IOP), is a processor with direct memory access capability that communicates with I/O devices. Each IOP takes care of input and output tasks involved in I/O transfers.

A processor that communicates with remote terminals over telephone and other communication media in a serial fashion is called a data Communication Processor.

I/O processing:—

We know that, DMA controller must be entirely set up by the CPU. where as IOP can fetch and execute its own instructions.

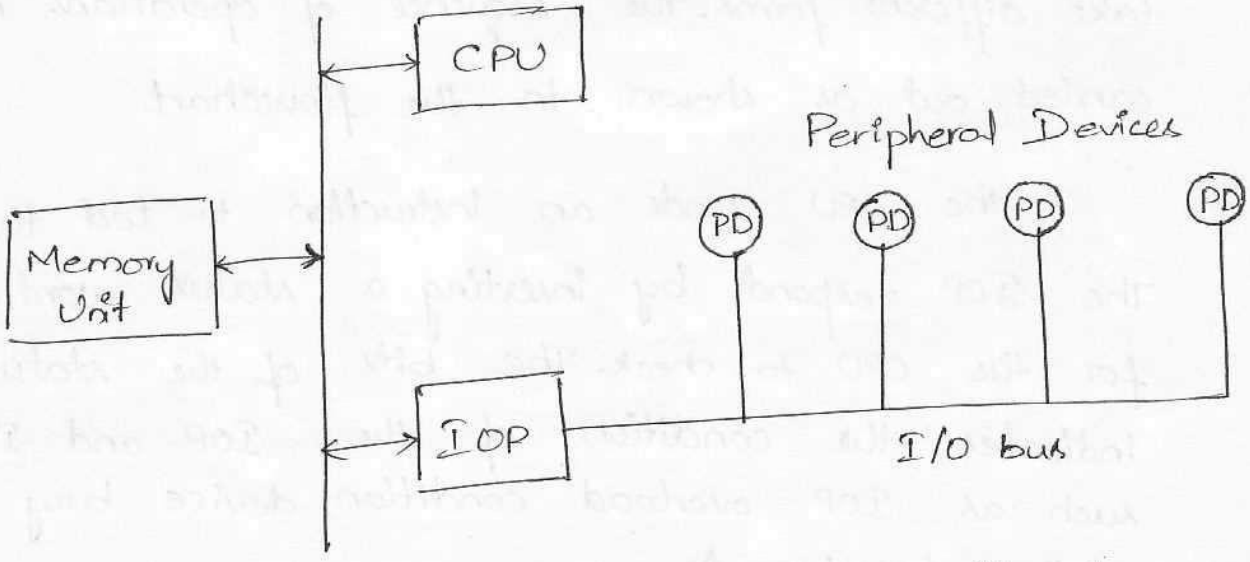


fig:- Block Diagram of a computer with I/O processor

In the block diagram, the memory unit occupies a central position and can communicate with each processor by means of direct memory access. The CPU is responsible for processing data needed in the solution of computational tasks. The IOP provides a path for transfer of data between various peripheral devices and the memory unit. The CPU is usually assigned the task of initiating the I/O program.

The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources. The communication between the IOP and the devices attached to it is similar to the program control method of transfer. Communication with the CPU and IOP depends on the level of sophistication in system & communication with memory is similar to the direct access method.

CPU-IOP Communication:-

The communication between CPU and IOP may take different forms. The sequence of operations may be carried out as shown in the flowchart.

The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicates the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer etc.,

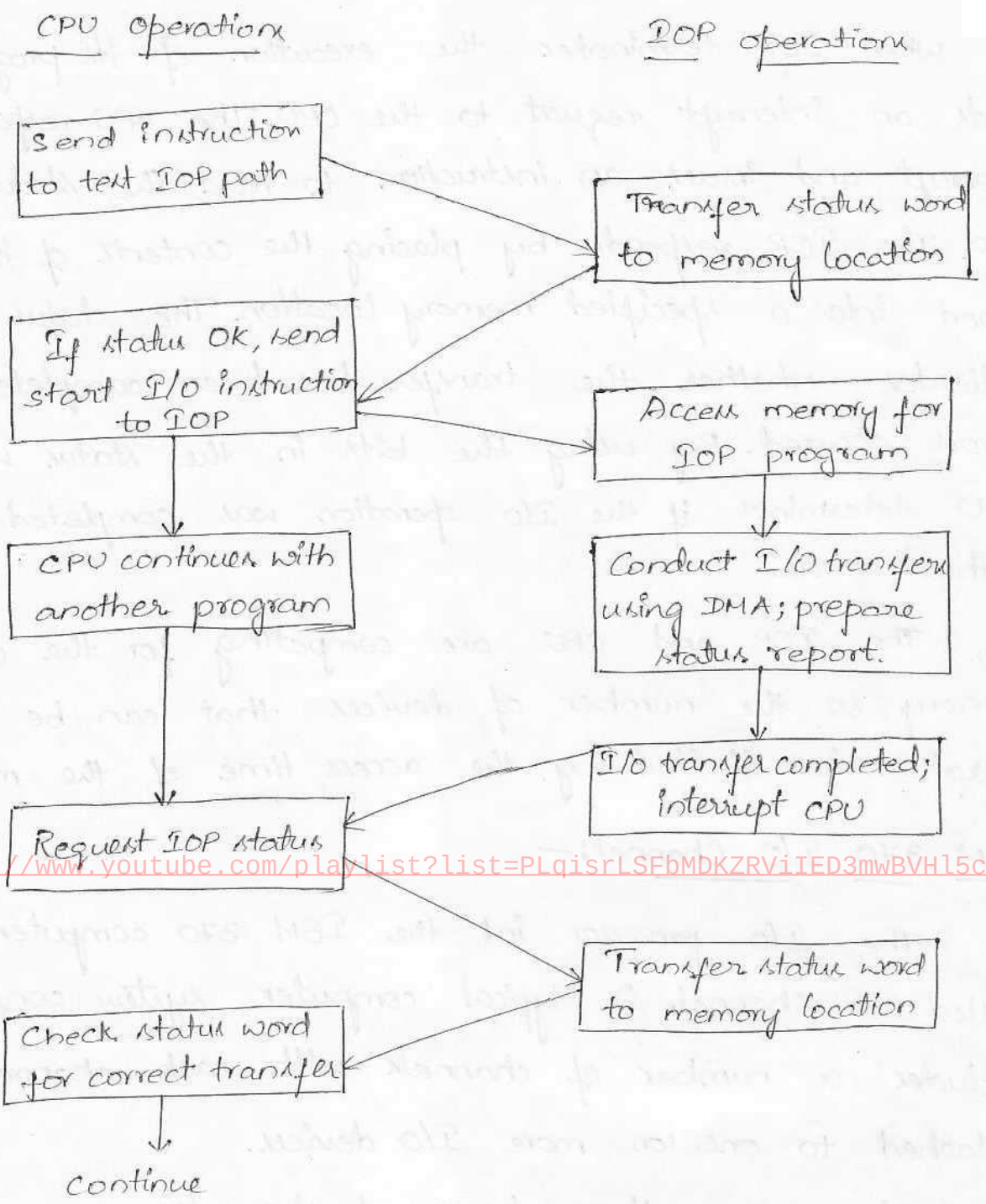


fig: CPU - IOP communication

The CPU refers to the status word in memory to decide the work to be done. Then CPU sends the I instruction to start I/O transfer. The memory address received with this instruction refers to the program. & CPU can continue with another program while IOP is busy with the I/O program.

When IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt and issues an instruction to read the status from IOP. The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed (or) any errors occured. By using the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

The IOP and CPU are competing for the use of memory, so the number of devices that can be in operation is limited by the access time of the memory.

IBM 370 I/O Channel:—

The I/O procesor in the IBM 370 computer is called a channel. A typical computer system configuration includes a number of channels with each channel attached to one (or) more I/O devices.

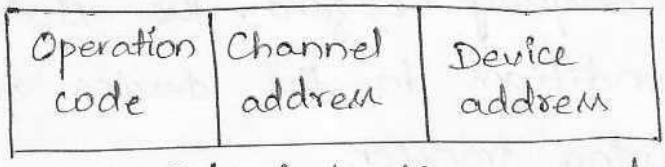
There are three types of channels:

i. Multiplexer
ii. Selector
iii. Block Multiplexer.

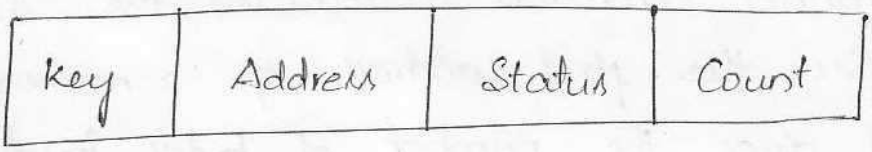
The Multiplexer channel can be connected to a number of slow and medium-speed devices. It is capable of operating with a number of I/O devices simultaneously.

The selector channel is designed to handle one I/O operation at a time. It is normally used to control one high-speed device.

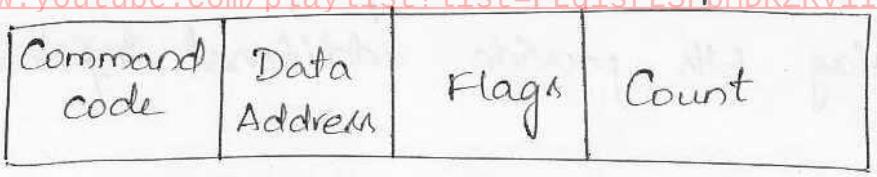
The block-Multiplexer channel combines the features of both the multiplexer and selector channels.



(a) I/O instruction format



(b) Channel status word format



(c) Channel command word format

fig:- IBM 370 I/O related word formats

The I/O instruction format has three fields: operation code, channel address, and device address, which contains the address of channels of computer system. Each channel may be connected to several devices and its address is specified by device address. The operation code specifies one of eight I/O instructions: start I/O, start I/O fast release, test I/O, clear I/O, halt I/O, halt device, test channel & store channel. The CPU can check the condition code in the PSW (program Status Word) to determine the result of the I/O operation

The channel status word is always stored in location 64 in memory. The key field is a protection mechanism used to prevent un-authorized access by one user information to another user. Address field gives the address of last command word used by the channel. Count field gives the residue count when the transfer was completed successfully i.e., zero. The status field identifies the conditions in the device and the channel & also errors during transfer.

In the channel Command Word (CCW), the data address field specifies the first address of a memory buffer. The count field gives the number of bytes involved in the transfer. The command field specifies an I/O operation and the flag bits provide additional information for the channel.

The command field corresponds to an operation code that specifies one of the six types of I/O operations:

1. Read: Transfer data from I/O device to memory
2. Write: Transfer data from memory to I/O device
3. Read backwards: Read magnetic tape with tape moving backward
4. Control: Used to initiate an operation not involving transfer of data.
5. Sense: Informs the channel to transfer its channel status word to memory location 64.
6. Transfer in channel:- Used instead of a jump instruction. Here the data address field specifies the address of next command word to be executed by the channel.

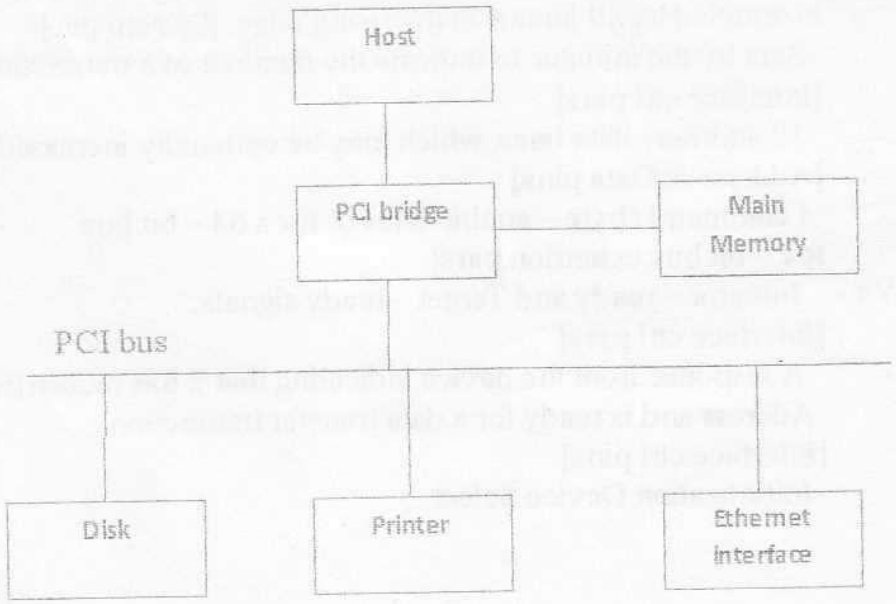
PCI: (Peripheral Component Interconnect). BUS:-

The PCI is a popular high – bandwidth, Processor – independent bus that can function as a mezzanine (or) peripheral bus. Compared with other common bus specifications, PCI delivers better system performance for high – speed I / O subsystems like graphics display adapters, network interface controllers, disk controllers and so on. PCI is specifically designed to meet economically the I / O requirements of modern systems. It requires very few chips to implement and supports other buses attached to the PCI bus.

PCI is designed to support a variety of microprocessor based configurations, including both single – and multiple – processor systems. It Provides a general – Purpose set of functions. It makes use of synchronous timing and a centralized arbitration scheme.

PCI was developed as a low – cost bus that is truly processor independent. An important feature that the PCI is a plug – and – play capability – for connecting I / O devices. To connect a new device, the user simply connects the device interface board to the bus.

Data transfer: - Most memory transfers involve a burst of data rather than one word. Data are transferred between the cache and the main memory in bursts of several words each. The words involved in such a transfer are stored at successive memory locations. When the processor specifies an address and requests a read operation from the main memory, the memory responds by sending a sequence of data words starting at that address. During a write operation the processor sends a memory address followed by a sequence of data words, to be written in successive memory locations starting at that address. The PCI is designed primarily to support this mode of operation.



The PCI bridge provides a separate physical connection for the main memory. For electrical reasons, the bus may be further divided into segments connected via bridges. The segment device address is mapped into the processor's memory address space.

At any given time, one device is the bus master. It has the right to initiate data transfer y issuing read and write commands. A 'master' is called an "initiator" in PCI terminology. This is either a processor (or) a DMA controller. The addressed device that responds to read & write commands is called a "target".

Consider a bus transaction in which the processor reads four 32 – bit words from the memory. In this case, the initiator is the processor and the target is the memory. A complete transfer operation on the bus, involving an address and a burst of data, is called a "transaction". Individual word transfers within a transaction are called "phases". A clock signal provides the timing reference used to coordinate different phases of transaction. All signal transactions are triggered by the rising edge of the clock.

BUS Structure:- PCI may be configured as a 32 – or 64 – bit bus. PCI contains 49 mandatory signal lines. These are divided into 5 groups.

1.System pins :- Include the clock and reset pins.
2.Address & data pins
3.Interface ctrl pins
4.Arbitration pins
5.Error reporting pins.

And PCI specification defines 51 optional signal lines divided into 4 groups.

1.Interrupt pins
2.cache support pins
3.64 – bit bus extension pins
4.JTAG / boundary scan pins

PCI Commands : -

Name	Function
1.CLK -	A 33 – MHZ clock. Provides timing to all transactions and is sampled by all inputs on the rising edge. [System pin]
2.FRAME# -	Sent by the initiator to indicate the duration of a transaction. [Interface ctrl pins]
3.AD -	32 address / data lines, which may be optionally increased to 64. [Address & Data pins]
4.C/BE# -	4 command / byte – enable lines (8 for a 64 – bit bus. [64 – bit bus extension pins]
5.IRDY#, TRDY# -	Initiator – ready and Target – ready signals. [Interface ctrl pins]
6.DEVSEL# - (Device Select	A response from the device indicating that it has recognized its Address and is ready for a data transfer transaction. [Interface ctrl pins]
7.IDSWL# -	Initialization Device Select

Serial communication standards

Serial communication is a popular means of transmitting data between a computer and a peripheral device such as a programmable instrument or even another computer. Serial communication uses a transmitter to send data, one bit at a time, over a single communication line to a receiver. We can use this method when data transfer rates are low or you must transfer data over long distances. Serial communication is popular because most computers have one or more serial ports, so no extra hardware is needed other than a cable to connect the instrument to the computer or two computers together.

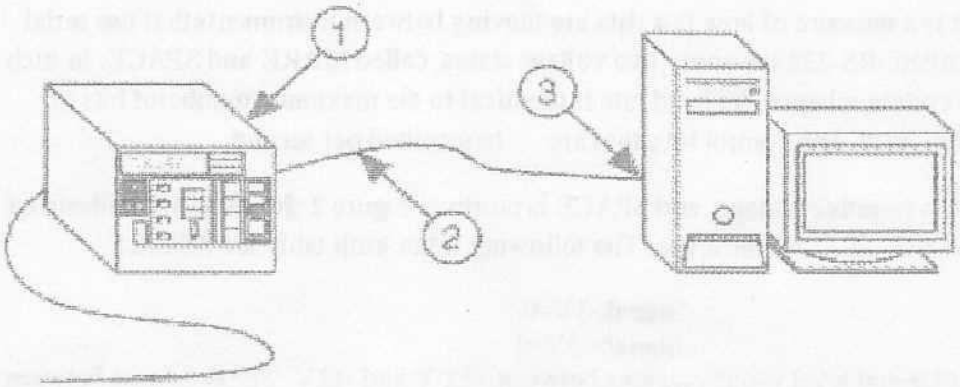


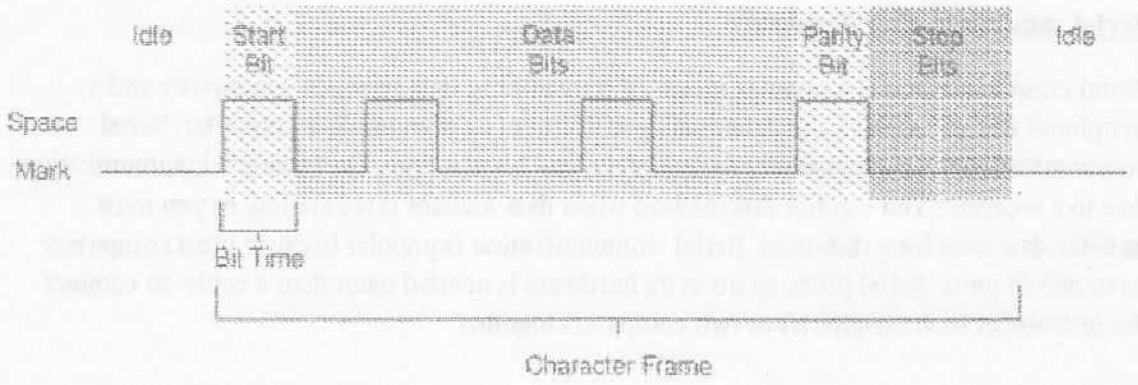
Figure 1:

1; RS-232 Instrument 2; RS-232 Cable 3; Serial Port

Serial communication requires that you specify the following four parameters:

1. The baud rate of the transmission
2. The number of data bits encoding a character
3. The sense of the optional parity bit
4. The number of stop bits

Each transmitted character is packaged in a character frame that consists of a single start bit followed by the data bits, the optional parity bit, and the stop bit or bits. Figure 2 shows a typical character frame encoding the letter m.



Baud rate is a measure of how fast data are moving between instruments that use serial ommunication. RS-232 uses only two voltage states, called MARK and SPACE. In such a two-state coding scheme, the baud rate is identical to the maximum numberof bits of information, including control bits, that are transmitted per second.

MARK is a negative voltage, and SPACE is positive. Figure 2 shows how the idealized signal looks on an oscilloscope. The following is the truth table for RS-232:

Signal>3V=0
Signal>-3V=1

The output signal level usually swings between +12 V and -12V. The dead area between +3 V and -3 V is designed to absorb line noise.

A start bit signals the beginning of each character frame. It is a transition from negative (MARK) to positive (SPACE) voltage. Its duration in seconds is the reciprocal of the baud rate. If the instrument is transmitting at 9,600 baud, the duration of the start bit and each subsequent bit is about 0.104 ms. The entire character frame of eleven bits would be transmitted in about 1.146 ms.

Data bits are transmitted upside down and backwards. That is, inverted logic is used, and the order of transmission is from least significant bit (LSB) to most significant bit (MSB). To interpret the data bits in a character frame, you must read from right to left and read 1 for negative voltage and 0 for positive voltage. This yields 1101101 (binary) or 6D (hex). An ASCII conversion table shows that this is the letter m.

An optional parity bit follows the data bits in the character frame. The parity bit, if present, also follows inverted logic,1 for negative voltage and 0 for positive voltage. This bit is included as a simple means of error handling. You specify ahead of time whether the parity of the transmission is to be even or odd. If the parity is chosen to be odd, the transmitter then sets the parity bit in such a way as to make an odd number of ones among the data bits and the parity bit. This transmission uses odd parity. There are five ones among the data bits, already an odd number, so the parity bit is set to 0.

The last part of a character frame consists of 1, 1.5, or 2 stop bits. These bits are always represented by a negative voltage. If no further characters are transmitted, the line stays in the negative (MARK) condition. The transmission of the next character frame, if any, is heralded by a start bit of positive (SPACE) voltage.

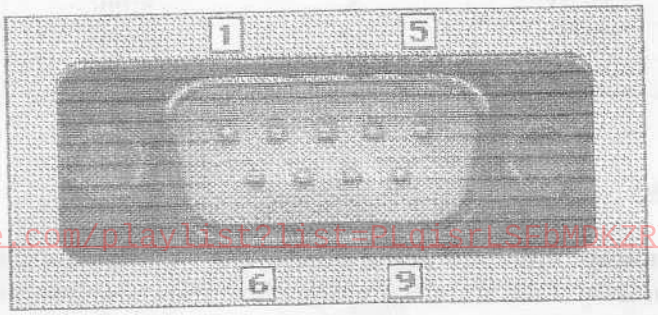
Hardware Overview

There are many different recommended standards of serial port communication, including the following most common types.

The RS-232 is a standard developed by the Electronic Industries Association (EIA) and other interested parties, specifying the serial interface between Data Terminal Equipment (DTE) and Data Communications Equipment (DCE). The RS-232 standard includes electrical signal characteristics (voltage levels), interface mechanical characteristics (connectors), functional description of interchange circuits (the function of each electrical signal), and some recipes for common kinds of terminal-to-modem connections. The most frequently encountered revision of this standard is called RS-232C. Parts of this standard have been adopted (with various degrees of fidelity) for use in serial communications between computers and printers, modems, and other equipment. The serial ports on standard IBM-compatible personal computers follow RS-232.

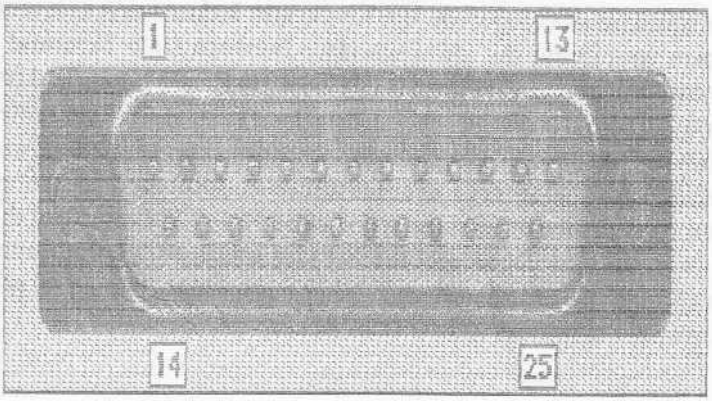
RS-232 Cabling

Devices that use serial cables for their communication are split into two categories. These are DCE and DTE. DCE are devices such as a modem, TA adapter, plotter, and so on, while DTE is a computer or terminal. RS-232 serial ports come in two sizes, the D-Type 25-pin connector and the D-Type 9-pin connector. Both of these connectors are male on the back of the PC. Thus, you require a female connector on the device. Table 1 shows the pin connections for the 9-pin and 25-pin D-Type connectors.



Function	Signal	PIN	DTE	DCE
Data	TxD	3	Output	Input
	RxD	2	Input	Output
Handshake	RTS	7	Output	Input
	CTS	8	Input	Output
	DSR	6	Input	Output
	DCD	1	Input	Output
	STR	4	Output	Input
Common	Com	5	--	--
Other	RI	9	Output	Input

The DB-9 connector is occasionally found on smaller RS-232 lab equipment. It is compact, yet has enough pins for the core set of serial pins (with one pin extra). The DB-25 connector is the standard RS-232 connector, with enough pins to cover all the signals specified in the standard. Table 2 shows only the core set of pins that are used for most RS-232 interfaces.



Function	Signal	PIN	DTE	DCE
Data	TxD	2	Output	Input
	RxD	3	Input	Output
Handshake	RTS	4	Output	Input
	CTS	5	Input	Output
	DSR	6	Input	Output
	DCD	8	Input	Output
	STR	20	Output	Input
Common	Com	7	--	--

Universal Serial Bus

In information technology, Universal Serial Bus (USB) is a serial bus standard to connect devices to a host computer. USB was designed to allow many peripherals to be connected using a single standardized interface socket and to improve plug and play capabilities by allowing hot swapping; that is, by allowing devices to be connected and disconnected without rebooting the computer or turning off the device. Other convenient features include providing power to low-consumption devices, eliminating the need for an external power supply; and allowing many devices to be used without requiring manufacturer-specific device drivers to be installed.

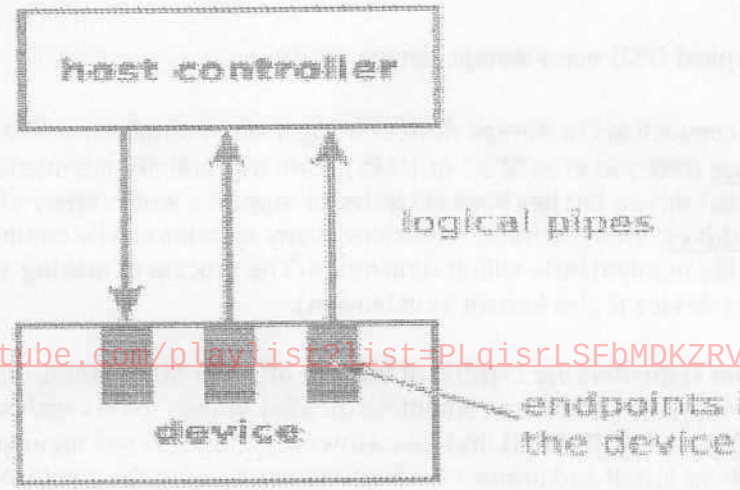
USB is intended to replace many varieties of serial and parallel ports. USB can connect computer peripherals such as mice, keyboards, PDAs, gamepads and joysticks, scanners, digital cameras, printers, personal media players, flash drives, and external hard drives. For many of those devices, USB has become the standard connection method. USB was designed for personal computers, but it has become commonplace on other devices such as PDAs and video game consoles, and as a power cord between a device and an AC adapter plugged into a wall plug for charging. As of 2008, there are about 2 billion USB devices sold per year, and about 6 billion total sold to date.[1]

The design of USB is standardized by the USB Implementers Fomm (USB-IF), an industry standards body incorporating leading companies from the computer and electronics industries. Notable members have included Agere (now merged with LSI Corporation), Apple Inc., Hewlett-Packard, Intel, Microsoft and NEC.

A USB system has an asymmetric design, consisting of a host, a multitude of downstream USB ports, and multiple peripheral devices connected in a tiered-star topology. Additional USB hubs may be included in the tiers, allowing branching into a tree structure with up to five tier levels. A USB host may have multiple host controllers and each host controller may provide one or more USB ports. Up to 127 devices, including the hub devices, may be connected to a single host controller.

USB devices are linked in series through *hubs*. There always exists one hub known as the root hub, which is built into the host controller. So-called "sharing hubs", which allow multiple computers to access the same peripheral device(s), also exist and work by switching access between PCs, either automatically or manually. They are popular in small-office environments. In network terms, they converge rather than diverge branches.

A physical USB device may consist of several logical sub-devices that are referred to as *device functions*. A single device may provide several functions, for example, a webcam (video device function) with a built-in microphone (audio device function).



USB device communication is based on *pipes* (logical channels). Pipes are connections from the host controller to a logical entity on the device named an endpoint. The term *endpoint* is occasionally used to incorrectly refer to the pipe. A USB device can have up to 32 active pipes, 16 into the host controller and 16 out of the controller.

Each endpoint can transfer data in one direction only, either into or out of the device, so each pipe is uni-directional. Endpoints are grouped into *interfaces* and each interface is associated with a single device function. An exception to this is endpoint zero, which is used for device configuration and which is not associated with any interface.

When a USB device is first connected to a USB host, the USB device enumeration process is started. The enumeration starts by sending a reset signal to the USB device. The speed of the USB device is determined during the reset signaling. After reset, the USB device's information is read by the host, then the device is assigned a unique 7-bit address. If the device is supported by the host, the device drivers needed for communicating with the device are loaded and the device is set to a configured state. If the USB host is restarted, the enumeration process is repeated for all connected devices.

implementation costs and a simplified, more adaptable cabling system. The 1394 standard also defines a backplane interface, though this is not as widely used.

IEEE 1394 has been adopted as the High-Definition Audio-Video Network Alliance (HANA) standard connection interface for A/V (audio/visual) component communication and control.[1] FireWire is also available in wireless, fiber optic, and coaxial versions using the isochronous protocols.

ireWire is Apple Inc.'s name for the IEEE 1394 High Speed Serial Bus. It was initiated by Apple (in 1986[2]) and developed by the IEEE P1394 Working Group, largely driven by contributions from Apple, although major contributions were also made by engineers from Texas Instruments, Sony, Digital Equipment Corporation, IBM, and INMOS/SGS Thomson (now STMicroelectronics).

Apple intended FireWire to be a serial replacement for the parallel SCSI (Small Computer System Interface) bus while also providing connectivity for digital audio and video equipment. Apple's development began in the late 1980s, later presented to the IEEE,[3] and was completed in 1995. As of 2007, IEEE 1394 is a composite of four documents: the original IEEE Std. 1394-1995, the IEEE Std. 1394a-2000 amendment, the IEEE Std. 1394b-2002 amendment, and the IEEE Std. 1394c-2006 amendment. On June 12, 2008, all these amendments as well as errata and some technical updates were incorporated into a superseding standard IEEE Std. 1394-2008.[4]

Apple's internal code-name for FireWire was "Greyhound" as of May 11, 1992.

Sony's implementation of the system, known as "i.LINK" used a smaller connector with only the four signal circuits, omitting the two circuits which provide power to the device in favor of a separate power connector. This style was later added into the 1394a amendment.[3] This port is sometimes labeled "S100" or "S400" to indicate speed in Mbit/s.

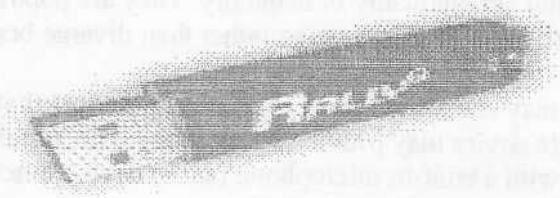
The system is commonly used for connection of data storage devices and DV (digital video) cameras, but is also popular in industrial systems for machine vision and professional audio systems. It is preferred over the more common USB for its greater effective speed and power distribution capabilities, and because it does not need a computer host. Perhaps more important, FireWire makes full use of all SCSI capabilities and has high sustained data transfer rates, a feature especially important for audio and video editors. Benchmarks show that the sustained data transfer rates are higher for FireWire than for USB 2.0, especially on Apple Mac OS X with more varied results on Microsoft Windows.[5][6]

However, the royalty which Apple Inc. and other patent holders initially demanded from users of FireWire (US$0.25 per end-user system) and the more expensive hardware needed to implement it (US$1–$2), both of which have since been dropped, have prevented FireWire from displacing USB in low-end mass-market computer peripherals, where product cost is a major constraint.[3]

The host controller directs traffic flow to devices, so no USB device can transfer any data on the bus without an explicit request from the host controller. In USB 2.0, host controller polls the bus for traffic, usually in a round-robin fashion. In SuperSpeed USB, connected devices can request service from host.

USB mass storage

Main article: USB mass storage device class



A flash drive, a typical USB mass-storage device.

USB implements connections to storage devices using a set of standards called the *USB mass storage device class* (referred to as MSC or UMS). This was initially intended for traditional magnetic and optical drives, but has been extended to support a wide variety of devices, particularly flash drives. This generality is because many systems can be controlled with the familiar idiom of file manipulation within directories (The process of making a novel device look like a familiar device is also known as extension).

Though most newer computers are capable of booting off USB Mass Storage devices, USB is not intended to be a primary bus for a computer's internal storage: buses such as ATA (IDE), Serial ATA (SATA), and SCSI fulfill that role. However, USB has one important advantage in that it is possible to install and remove devices without opening the computer case, making it useful for external drives. Originally conceived and still used today for optical storage devices (CD-RW drives, DVD drives, etc.), a number of manufacturers offer external portable USB hard drives, or empty enclosures for drives, that offer performance comparable to internal drives[citation needed]. These external drives usually contain a translating device that interfaces a drive of conventional technology (IDE, ATA, SATA, ATAPI, or even SCSI) to a USB port. Functionally, the drive appears to the user just like an internal drive. Other competing standards that allow for external connectivity are eSATA and FireWire.

Another use for USB Mass Storage devices is the portable execution of software applications without the need of installation on the host computer,[5] such as Web Browsers, VoIP clients,[6] etc.

IEEE 1394

The **IEEE 1394 interface** is a serial bus interface standard for high-speed communications and isochronous real-time data transfer, frequently used by personal computers, as well as in digital audio, digital video, automotive, and aeronautics applications. The interface is also known by the brand names of **FireWire** (Apple Inc.), **i.LINK** (Sony), and **Lynx** (Texas Instruments). IEEE 1394 replaced parallel SCSI in many applications, because of lower