UNIT-3
INPUT-OUTPUT ORGANIZATION

Accessing of I/O Devices: (Input Output Interface)

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

Peripherals connected to a computer need <u>special communication links</u> for interfacing them with the central processing unit.

The purpose of communication link is to <u>resolve the differences that exist between the central computer and each peripheral.</u>

The Major Differences are:-

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

To resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervises and synchronizes all input and out transfersThese components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals.

The I/O Bus consists of data lines, address lines and control lines.

The I/O bus from the processor is attached to all peripherals interface.

To communicate with a particular device, the processor places a device address on address lines.

Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller. It is also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller.

For example, the printer controller controls the paper motion, the print timing

The control lines are referred as I/O command. The commands are as following:

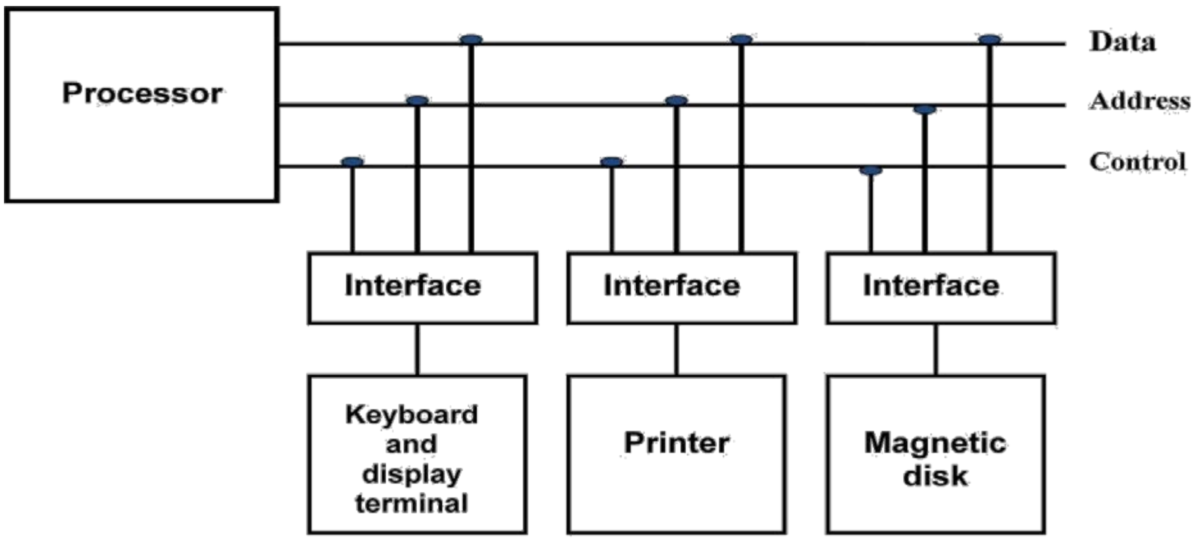
Control command- A control command is issued to activate the peripheral and to inform it what to do.

Status command- A status command is used to test various status conditions in the interface and the peripheral.

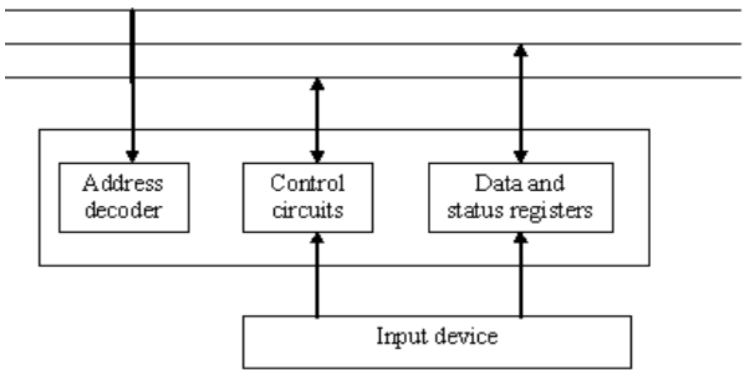
Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives on item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus



Connection of I/O bus to input-output devices



I/O interface for an input device

Asynchronous Data Transfer:

This Scheme is used when speed of I/O devices does not match with microprocessor, and timing characteristics of I/O devices is not predictable. In this method, process initiates the device and checks its status. As a result, CPU has to wait till I/O device is ready to transfer data. When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.

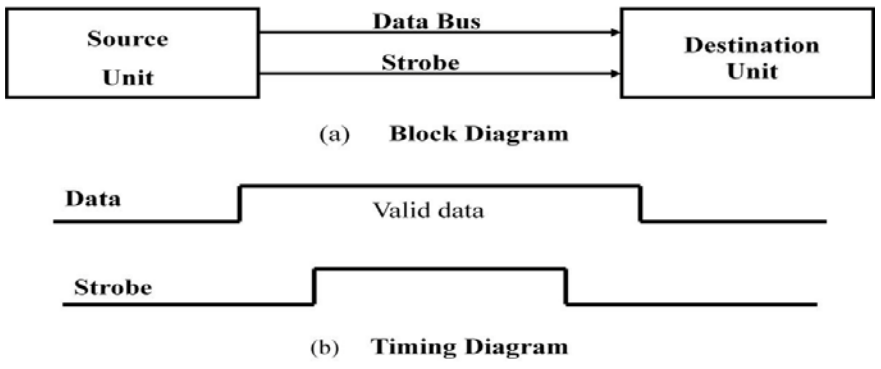
i. Strobe Control

ii. Handshaking

Strobe Signal:

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

Source Initiated Strobe method:



Source-Initiated strobe for Data Transfer

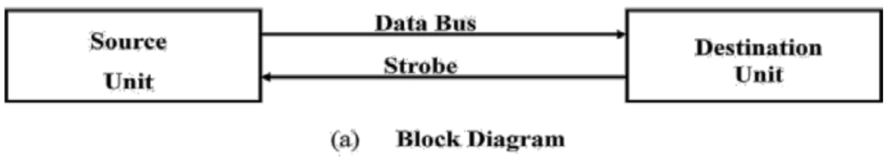
In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.

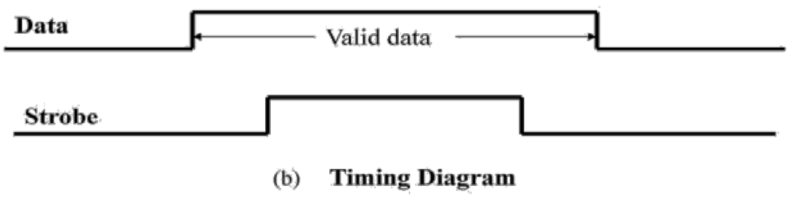
The timing diagram fig. (b) The source unit first places the data on the data bus. The information on the data bus and strobe signal remains in the active state to allow the destination unit to receive the data.

Destination Initiated Strobe method:

In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond 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. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.





Destination-Initiated strobe for Data Transfer

Disadvantage of Strobe Method:

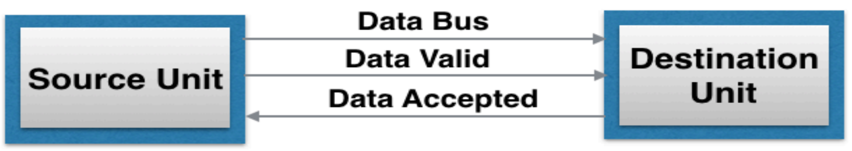
The disadvantage of the strobe method is that, the source unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was places in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus. The Handshaking method solves this problem.

Handshaking:

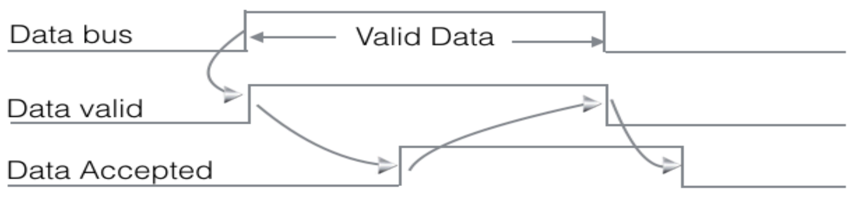
The handshaking method solves the problem of strobe method by introducing asecond control signal that provides a reply to the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

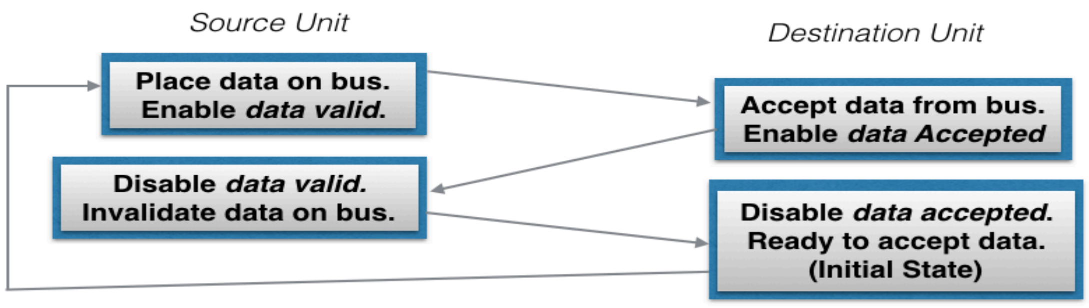
The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its *data valid* signal. The *data accepted* signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its *data accepted* signal and the system goes into its initial state.



a) Block Diagram



b) Timing Diagram

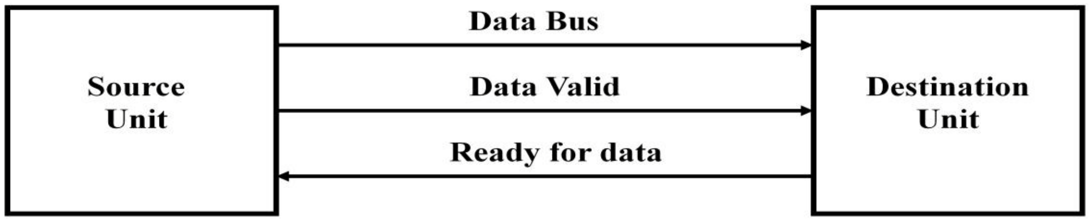

Source Unit
Destination Unit
Place data on bus. Enable data valid.
Accept data from bus. Enable data Accepted
Disable data valid. Invalidate data on bus.
Disable data accepted. Ready to accept data. (Initial State)

c) Sequence Diagram(Sequence of events)

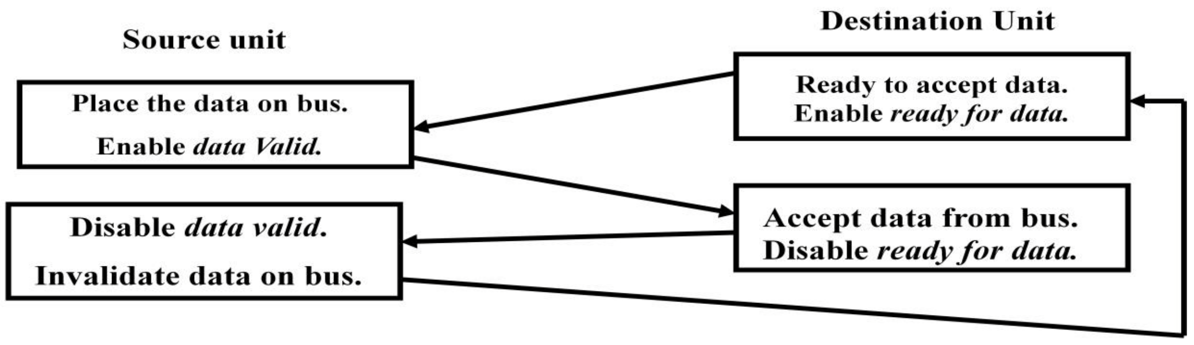
Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflect its new meaning. The source unit in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial sate.



(a) Block Diagram


Source unit
Destination Unit
Ready to accept data. Enable *ready for data.*
Place the data on bus. Enable *data Valid.*
Accept data from bus. Disable *ready for data.*
Disable *data valid.* Invalidate data on bus.

(b) Sequence of events

Destination-Initiated transfer using Handshaking

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *Timeout mechanism* which provides an alarm if the data is not completed within time.

Asynchronous Serial Transmission:

The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle.

In this technique each character consists of three points :

i. Start bit

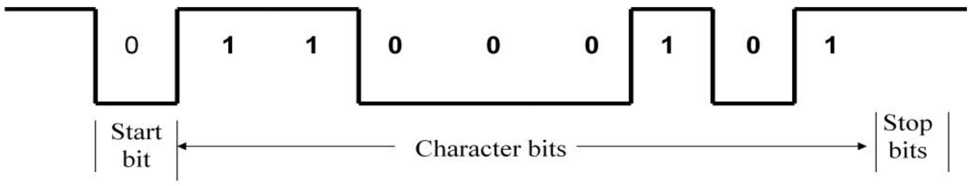
ii. Character bit

iii. Stop bit

i. Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.

ii. Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.

iii. Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by Asynchronous Communication Interface

Asynchronous Communication Interface:

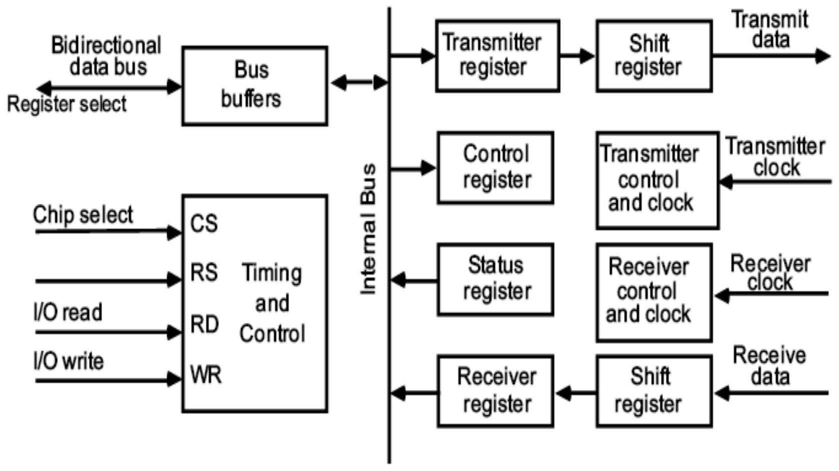
It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register.

The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission.

The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register.

CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

A typical asynchronous communication interface available as an IC



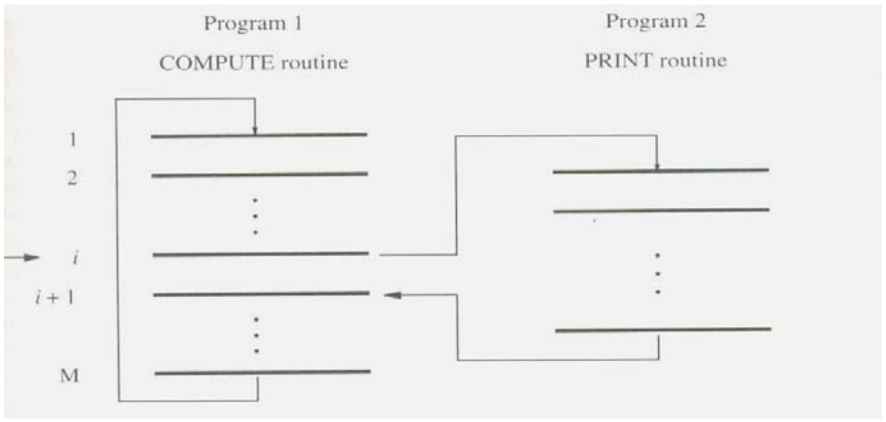
CS	RS	Oper.	Register selected
0	x	x	None
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the Interrupt service routine.
- There are two ways of choosing the branch address:
  - Vectored Interrupt
  - Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

INTERRUPTS

- When a program enters a wait loop, it will repeatedly check the device status. During this period, the processor will not perform any function.
- Whenever any device wants the attention, it sends the interrupt signal to the CPU. The Interrupt request line will send a hardware signal called the interrupt signal to the processor.
- On receiving this signal, the processor will perform the useful function during the waiting period
- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the Interrupt service routine.

Fig: Transfer of control through the use of interrupts



- The processor first completes the execution of instruction i Then it loads the PC (Program Counter) with the address of the first instruction of the ISR.
- After the execution of ISR, the processor has to come back to instruction i+1.Therefore,

  When an interrupt occurs, the current contents of PC which point to i+1 is put in temporary

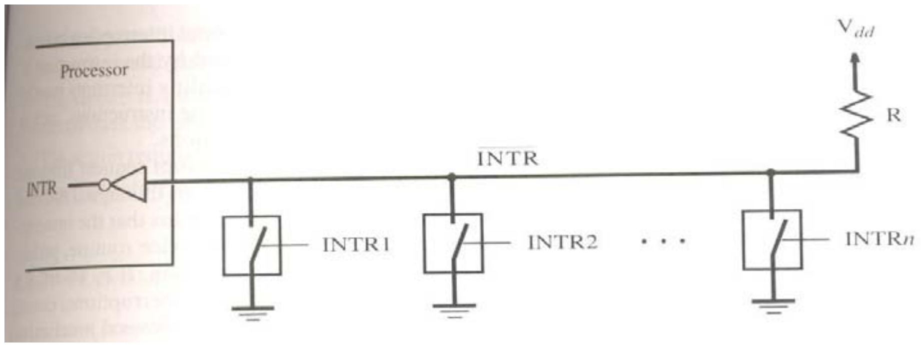
  storage in a known location.
- A return from interrupt instruction at the end of ISR reloads the PC from that temporary storage location, causing the execution to resume at instructioni+1.
- When the processor is handling the interrupts, it must inform the device that its request has been recognized so that it removes its interrupt requests signal.
- This may be accomplished by a special control signal called the **interrupt acknowledge signal.**
- The task of saving and restoring the information can be done automatically by the processor.
- The processor saves only the contents of **program counter & status register** (ie) it saves only the minimal amount of information to maintain the integrity of the program execution.

- Saving registers also increases the delay between the time an interrupt request is received and the start of the execution of the ISR. This delay is called the **Interrupt Latency.** Generally, the long interrupt latency in unacceptable.

- The concept of interrupts is used in Operating System and in Control Applications, where processing of certain routines must be accurately timed relative to external events. This application is also called as **real-time processing**.

Interrupt Hardware:

Fig: An equivalent circuit for an open drain bus used to implement a common interrupt request line



- A single interrupt request line may be used to serve "n‟devices. All devices are connected to the line via switches to ground.
- To request an interrupt, a device closes its associated switch, the voltage on INTR line drops to 0(zero).
- If all the interrupt request signals (INTR1 to INTR n) are inactive, all switches are open and the voltage on INTR line is equal to V_{dd}.
- When a device requests an interrupts, the value of INTR is the logical OR of the requests from individual devices.

(ie) **INTR = INTR$_1$+…………+INTR n**

$\overline{\text{INTR}}$ →It is used to name the INTR signal on common line it is active in the low Voltage state.

- **Open collector** (bipolar ckt) or **Open drain** (MOS circuits) is used to drive $\overline{\text{INTR}}$ line.
- The Output of the Open collector (or) Open drain control is equal to a switch to thegroundthatisopenwhengatesinputisin„0‟stateandclosedwhenthegates input is in „1‟state.
- Resistor „R‟ is called a **pull-up resistor** because it pulls the line voltage up to the high voltage state when the switches are open.

Enabling and Disabling Interrupts:

- The arrival of an interrupt request from an external device causes the processor to suspend the execution of one program & start the execution of another because the interrupt may alter the sequence of events to be executed.INTR is active during the execution of **Interrupt Service Routine**.
- There are 3 mechanisms to solve the problem of infinite loop which occurs due to successive interruptions of active INTR signals.
- The following are the typical scenario.

➔The device raises an interrupt request.
➔The processor interrupts the program currently being executed.
➔ Interrupts are disabled by changing the control bits is PS (Processor Status register)
➔The device is informed that its request has been recognized & in response, it deactivates the INTR signal.
➔The actions are enabled & execution of the interrupted program is resumed.

Handling Multiple Devices (Priority Interrupt):

- There are number of IO devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
  - Using Software
  - Using Hardware
- A pooling procedure is used to identify highest priority in software means.

Polling Procedure :

- There is one common branch address for all interrupts.
- Branch address contain the code that polls the interrupt sources in sequence. The highest priority is tested first.
- The particular service routine of the highest priority device is served.
- The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

Using Hardware:

- Hardware priority system functions as an overall manager.
- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.
- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU through common interrupt request line .
- CPU then sends the INTACK signal which is applied to PI(priority in) of the first device.
- If it had requested the attention, it place its VAD(vector address) on the bus. And it block the signal by placing 0 in PO(priority out)
- If not it pass the signal to next device through PO(priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

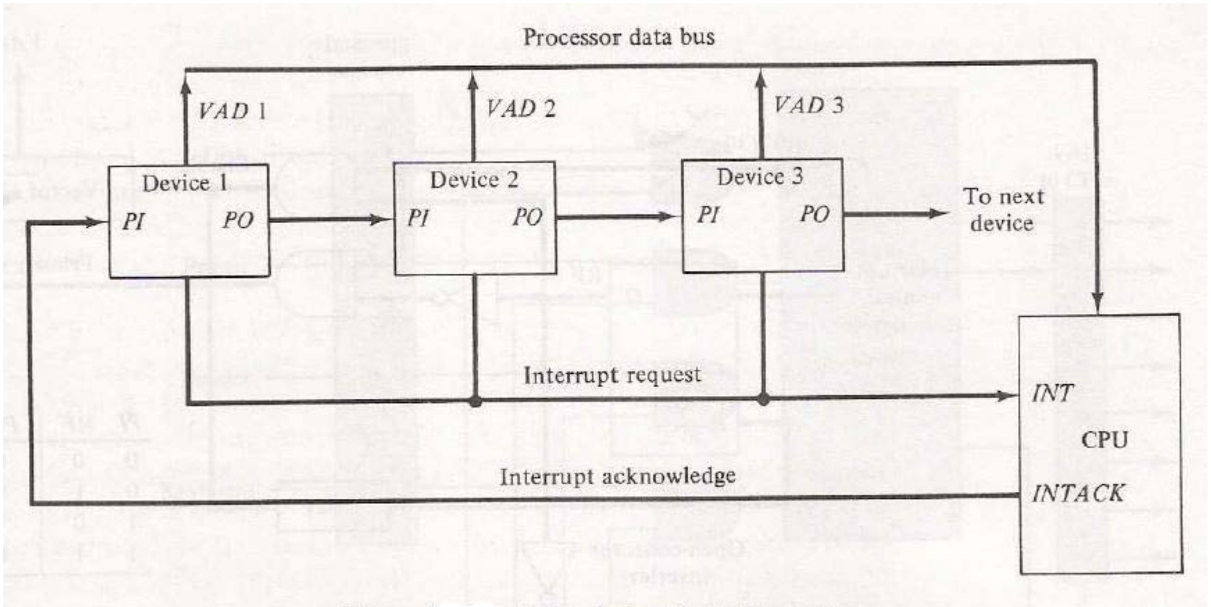
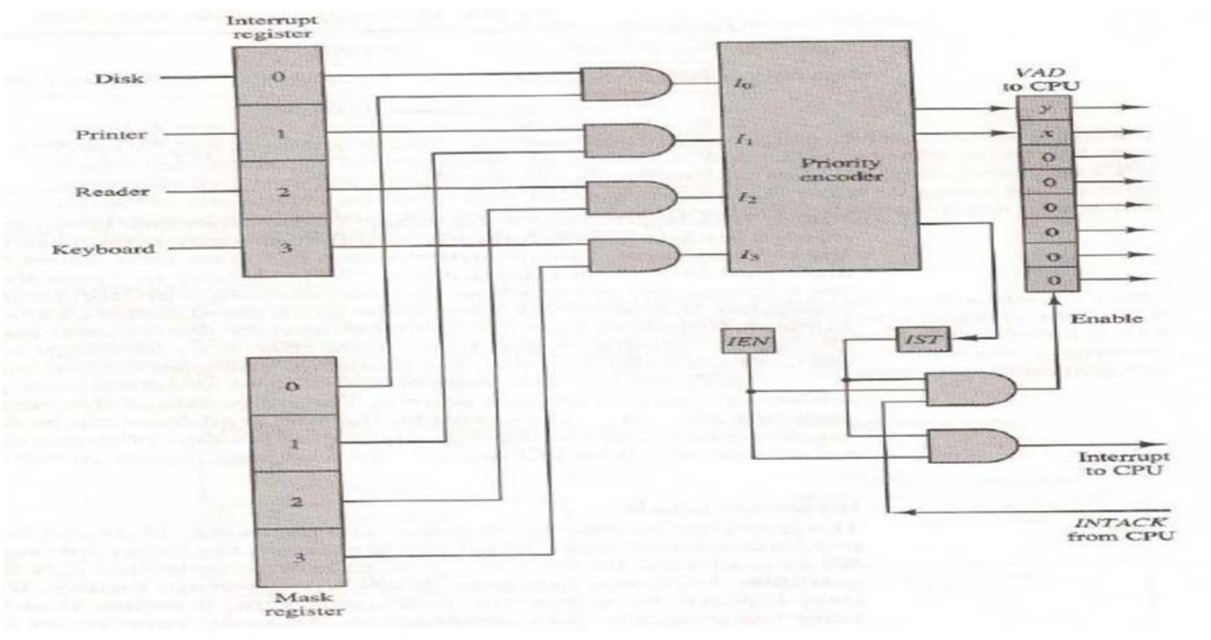
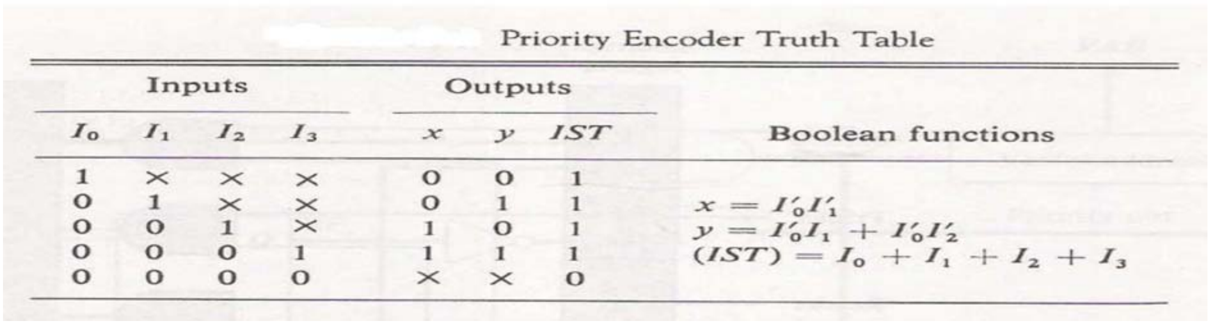


Figure Daisy-chain priority interrupt.

Parallel Priority Interrupt:

- It consists of interrupt register whose bits are set separately by the interrupting devices.
- Priority is established according to the position of the bits in the register.
- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being serviced.
- Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.
- Priority encoder generates two bits of vector address.
- Another output from it sets IST(interrupt status flip flop).





Priority Encoder Truth Table

Inputs				Outputs			
I_0	I_1	I_2	I_3	x	y	IST	Boolean functions
1	×	×	×	0	0	1	
0	1	×	×	0	1	1	$x = I'_0 I'_1$
0	0	1	×	1	0	1	$y = I'_0 I_1 + I'_0 I'_2$
0	0	0	1	1	1	1	$(IST) = I_0 + I_1 + I_2 + I_3$
0	0	0	0	×	×	0	

The Execution process of Interrupt–Initiated I/O is represented in the flowchart:

Direct Memory Access (DMA):

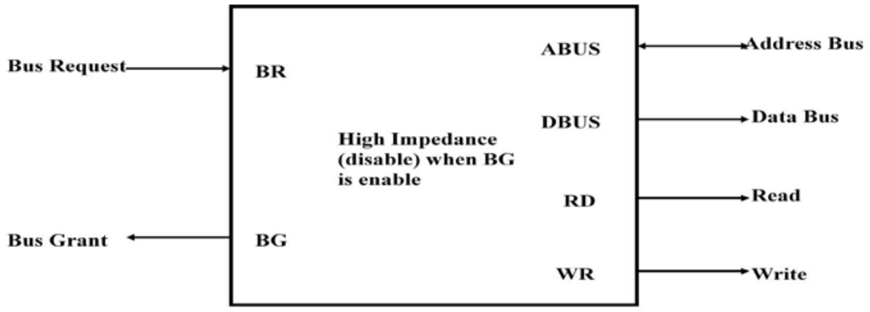
In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

During the 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 placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

- Bus Request (BR)
- Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request (BR)* input is used by the *DMA controller* to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a *high Impedance state.* High Impedance state means that the output is disconnected.



CPU bus Signals for DMA Transfer

The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the Bus Request (BR) can now take control of the buses to conduct memory transfer without processor.

When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPU disables the *Bus Grant (BG),* takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

i. DMA Burst

ii. Cycle Stealing

i) DMA Burst transfer :- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.

ii) Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must returns control of the buses to the CPU.

DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

i. Address Register
ii. Word Count Register
iii. Control Register

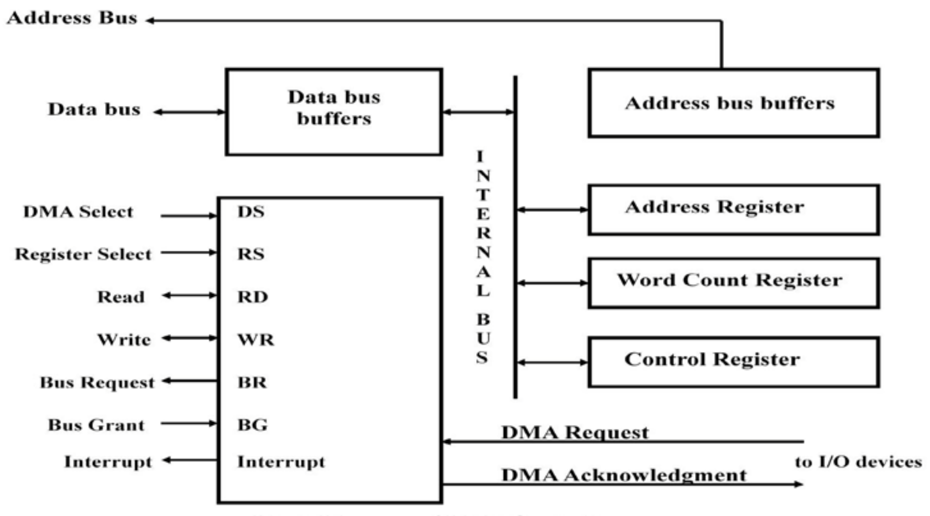
i. Address Register: - Address Register contains an address to specify the desired location in memory.

ii. Word Count Register:- it holds the number of words to be transferred. The register is increment/decrement by one after each word transfer and internally tested for zero.

iii. Control Register :- Control Register specifies the mode of transfer

The unit communicates 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 RD (read) and WR (write) inputs are bidirectional.

When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.



Block Diagram of DMA Controller

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as 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 transfer between the peripheral and the memory.

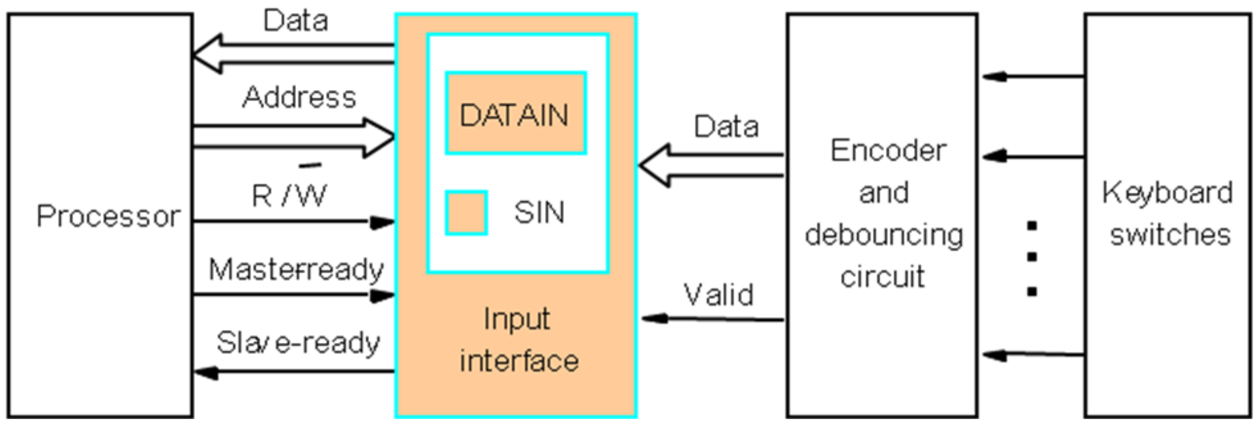
When BG = 0 the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG=1, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.

INTERFACE CIRCUITS:

The interface circuits are of two types. They are

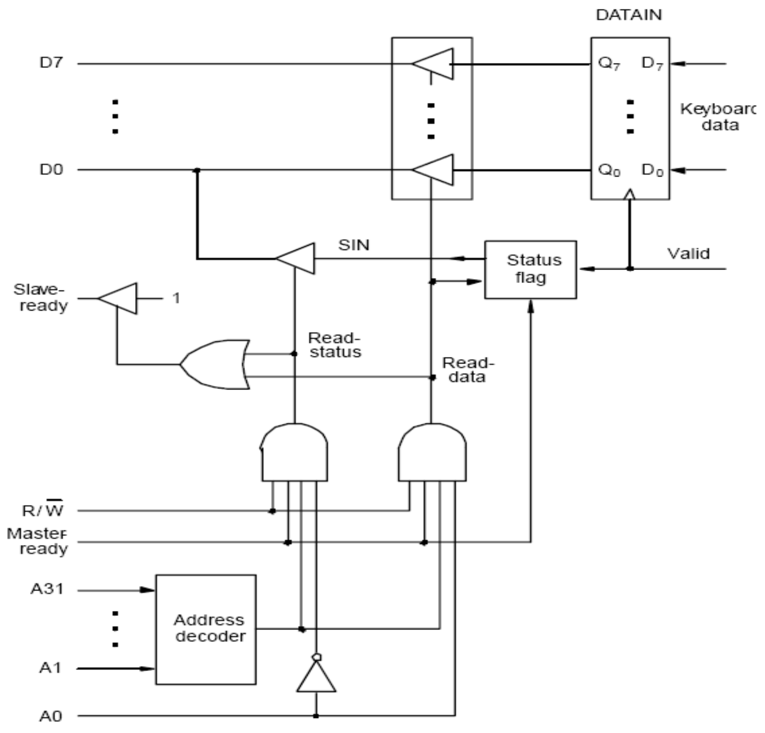
1) Parallel Port 2)Serial Port

Parallel Port(input):



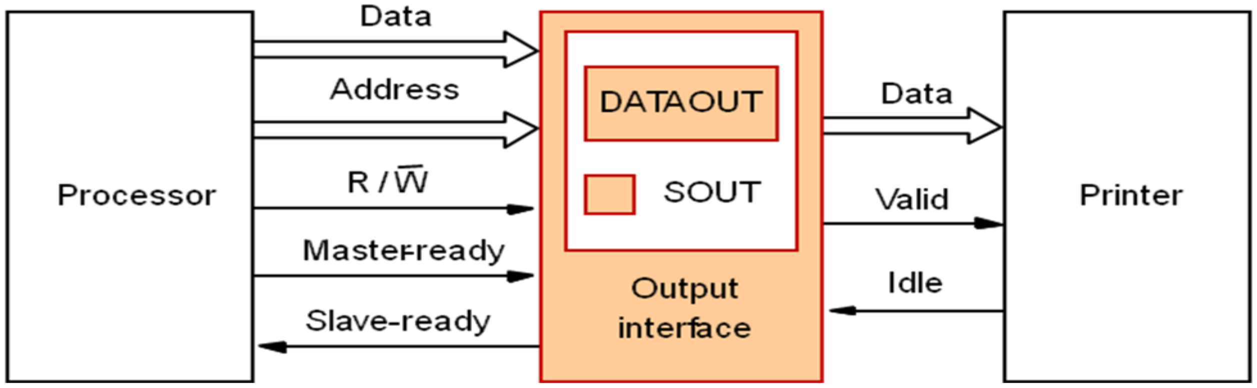
1) The output of the encoder consists of the bits that represent the encoded character and one signal called **vali**d, whi**c**h indicates the key is pressed.
2) The information is sent to the interface circuits , which contains a data register, DATAIN and a status flag SIN.
3) When a key is pressed, the Valid signal changes from 0 to1,causing the ASCII code to be loaded into DATAIN and SIN set to1.
4) The status flag SIN set to 0 when the processor reads the contents of the DATAIN register.
5) The interface circuit is connected to the asynchronous bus on which transfers are controlled using the Handshake signals Master ready and Slave-ready

Parallel input Interface Circuit:



- Output lines of DATAIN are connected to the data lines of the bus by means of 3 state drivers
- Drivers are turned on when the processor issues a read signal and the address selects this register
- SIN signal is generated using a status flag circuit.
- It is connected to line D_0 of the processor bus using a three-state driver.
- Address decoder selects the input interface based on bits A_1 through A_{31}.
- Bit A0 determines whether the status or data register is to be read, when Master-ready is active.
- In response, the processor activates the Slave-ready signal, when either the Read-status or Read-data is equal to 1, which depends on line A_0.

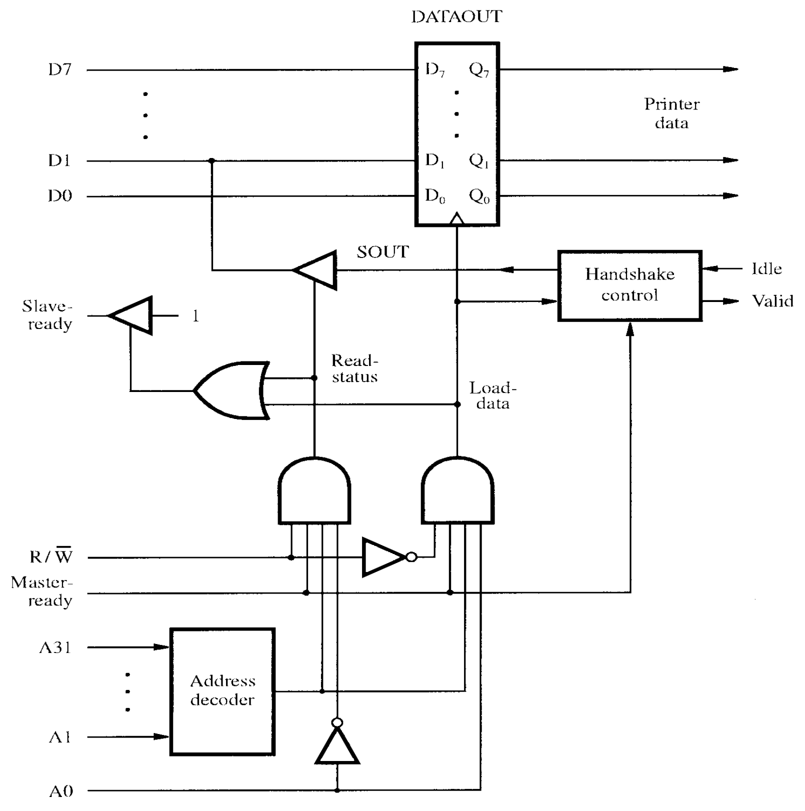
Parallel Port (output):



- Printer is connected to a processor using a parallel port.
- Processor is 32 bits, uses memory-mapped I/O and asynchronous bus protocol.
- On the processor side:
  - Data lines.
  - Address lines
  - Control or R/W line.
  - Master-ready signal and
  - Slave-ready signal.
- On the printer side:

- Idle signal line which the printer asserts when it is ready to accept a character. This causes the SOUT flag to be set to 1.
- Processor places a new character into a DATAOUT register.
- Valid signal, asserted by the interface circuit when it places a new character on the data lines.

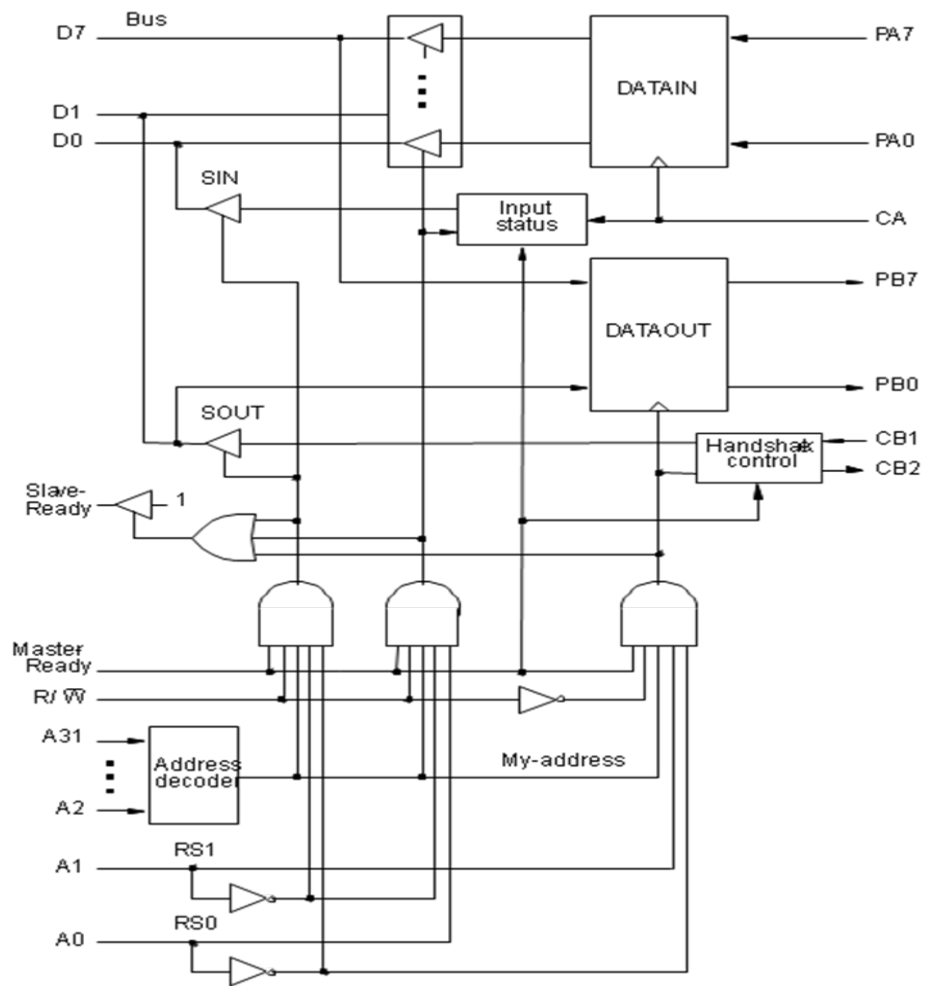
Parallel output Interface Circuit:



- Data lines of the processor bus are connected to the DATAOUT register of the interface.
- The status flag SOUT is connected to the data line D1 using a three-state driver.
- The three-state driver is turned on, when the control Read-status line is 1.
- Address decoder selects the output interface using address lines A1 through A31.
- Address line A0 determines whether the data is to be loaded into the DATAOUT register or status flag is to be read.
- If the Load-data line is 1, then the Valid line is set to 1.
- If the Idle line is 1, then the status flag SOUT is set to 1.

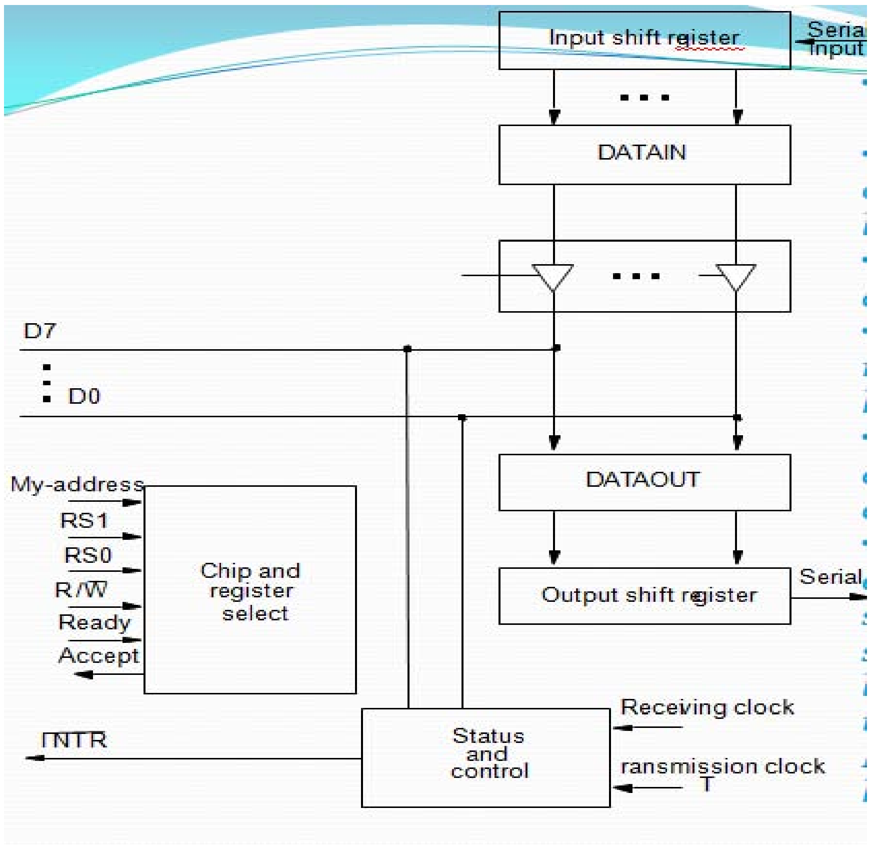
Combined Input Output interface circuit.

- Address bits A2 through A31, that is 30 bits are used to select the overall interface.
- Address bits A1 through A0, that is, 2 bits select one of the three registers, namely, DATAIN, DATAOUT, and the status register.
- Status register contains the flags SIN and SOUT in bits 0 and 1.
- Data lines PA0 through PA7 connect the input device to the DATAIN register.
- DATAOUT register connects the data lines on the processor bus to lines PB0 through PB7 which connect to the output device.
- Separate input and output data lines for connection to an I/O device.



Serial port:

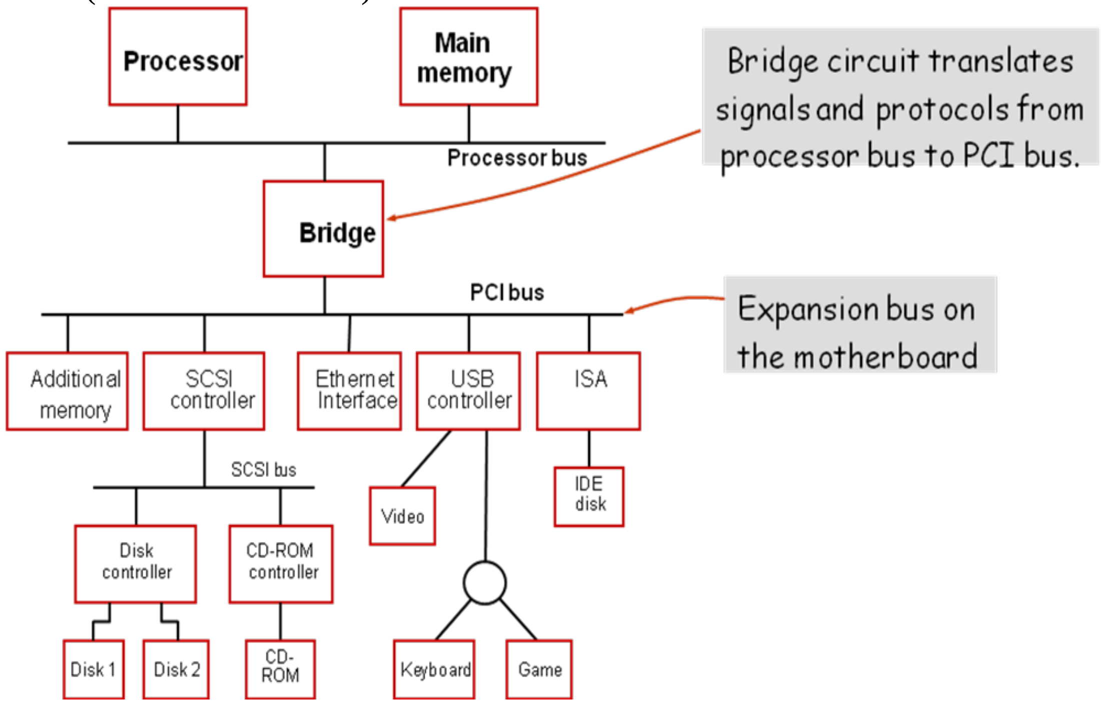
- Serial port is used to connect the processor to I/O devices that require transmission of data one bit at a time.
- Serial port communicates in a bit-serial fashion on the device side and bit parallel fashion on the bus side.
    - Transformation between the parallel and serial formats is achieved with shift registers that have parallel access capability.



- *Input shift register accepts input one bit at a time from the I/O device.*
- *Once all the 8 bits are received, the contents of the input shift register are loaded in parallel into DATAIN register.*
- *Output data in the DATAOUT register are loaded into the output shift register.*
- *Bits are shifted out of the output shift register and sent out to the I/O device one bit at a time.*
- *As soon as data from the input shift reg. are loaded into DATAIN, it can start accepting another 8 bits of data.*
- *Input shift register and DATAIN registers are both used at input so that the input shift register can start receiving another set of 8 bits from the input device after loading the contents to DATAIN, before the processor reads the contents of DATAIN. This is called as double- buffering.*
- Serial interfaces require fewer wires, and hence serial transmission is convenient for connecting devices that are physically distant from the computer.
- Speed of transmission of the data over a serial interface is known as the “bit rate”.
- Bit rate depends on the nature of the devices connected.
- In order to accommodate devices with a range of speeds, a serial interface must be able to use a range of clock speeds.

Standard I/O interfaces

- I/O device is connected to a computer using an interface circuit.
- A practical approach is to develop standard interfaces and protocols instead of design a different interface for every combination of an I/O device.
- A personal computer has:
  - A motherboard which houses the processor chip, main memory and some I/O interfaces.
  - A few connectors into which additional interfaces can be plugged.
- Processor bus is defined by the signals on the processor chip.
  - Devices which require high-speed connection to the processor are connected directly to this bus.
- Because of electrical reasons only a few devices can be connected directly to the processor bus.
- Motherboard usually provides another bus that can support more devices.
  - Processor bus and the other bus (called as expansion bus) are interconnected by a circuit called "bridge".
  - Devices connected to the expansion bus experience a small delay in data transfers.
- Design of a processor has no uniform standard can be defined.
- Expansion bus however can have uniform standard defined.
- Three widely used bus standards:
  - PCI (Peripheral Component Interconnect)
  - SCSI (Small Computer System Interface)
  - USB (Universal Serial Bus)



PCI(Peripheral component interconnect bus):

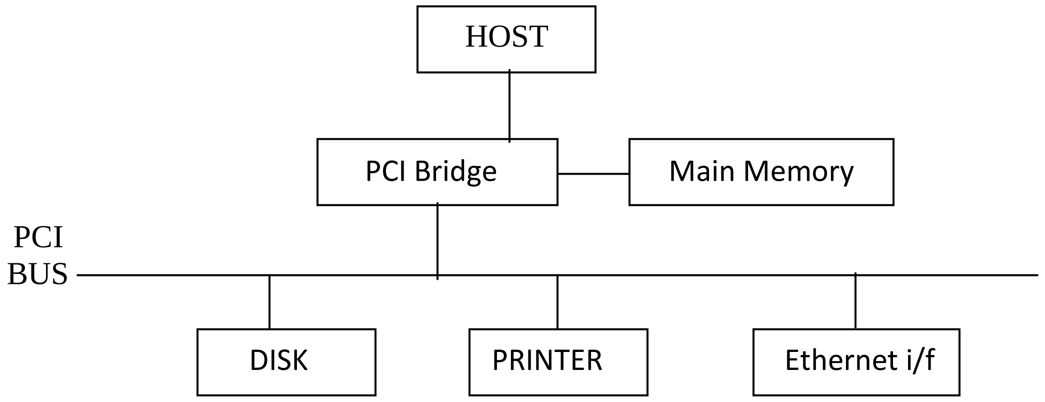
- PCI is developed as a low cost bus that is truly processor independent.
- It supports high speed disk, graphics and video devices.
- PCI has plug and play capability for connecting I/O devices.
- To connect new devices, the user simply connects the device interface board to the bus.

Data Transfer:

- The data are transferred between cache and main memory is the bursts of several words and they are stored in successive memory locations.
- When the processor specifies an address and request a „read" operation from memory, the memory responds by sending a sequence of data words starting at that address.
- During write operation, the processor sends the address followed by sequence of data words to be written in successive memory locations.
- PCI supports read and write operation.
- A read / write operation involving a single word is treated as a burst of lengthone.
- PCI has three address spaces. They are
    - ➢ Memory address space
    - ➢ I/O address space
    - ➢ Configuration address space

- I/O address space → It is intended for use with processor
- Configuration space →It is intended to give PCI, its plug and play Capability.
- PCI Bridge provides a separate physical connection to main memory.
- The master maintains the address information on the bus until data transfer is Completed.
- At any time, only one device acts as **bus master**.
- A master is called „initiator" in PCI which is either processor or **DMA.**
- The addressed device that responds to read and write commands is called a **target.**
- A complete transfer operation on the bus, involving an address and bust of data is called a **transaction'.**

Fig: Use of a PCI bus in a Computer system

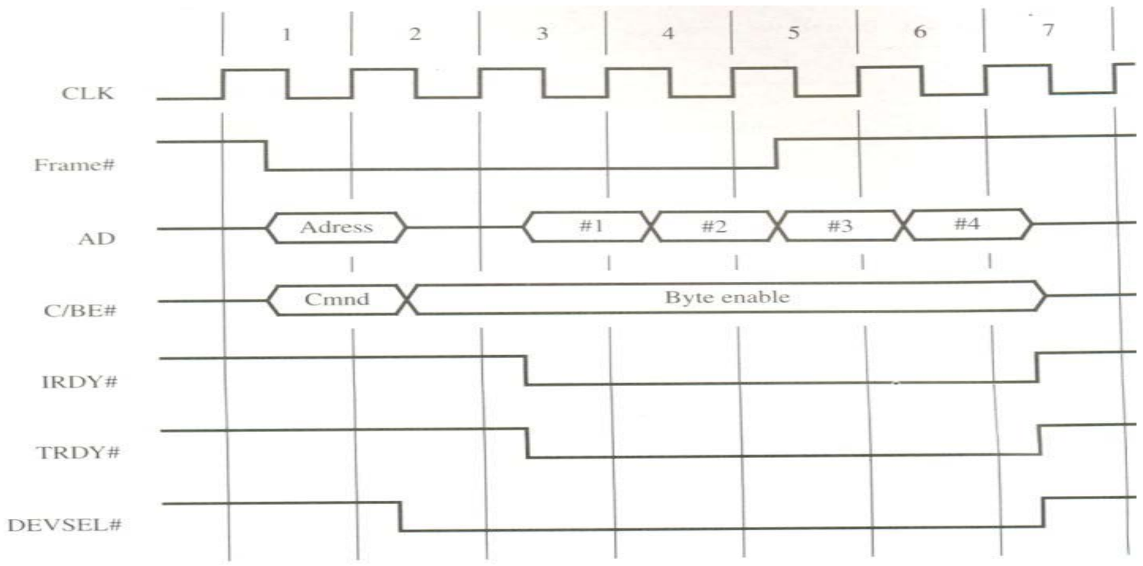


Data Transfer Signals on PCI Bus:

Name		**Function**
CLK	→	33 MHZ / 66 MHZ clock
FRAME #	→	Sent by the indicator to indicate the duration of transaction
AD	→	32 address / data line
C/BE #	→	4 command / byte Enable Lines
IRDY, TRDYA	→	Initiator Ready, Target Ready Signals
DEVSEL#	→	A response from the device indicating that it has recognized its address and is ready for data transfer transaction.
IDSEL #	→	Initialization Device Select

Individual word transfers are called ’**phases’**.

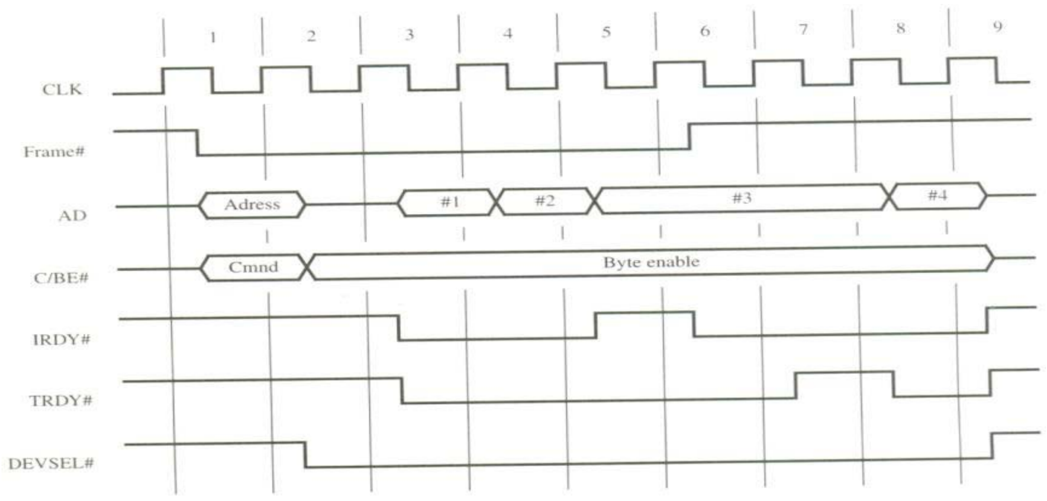
Fig : Read operation an PCI Bus



- In Clock cycle1, the processor asserts FRAME # to indicate the beginning of a transaction ; it sends the address on AD lines and command on C/BE #Lines.
- Clock cycle2 is used to turn the AD Bus lines around ; the processor ; The processor removes the address and disconnects its drives from AD lines.
- The selected target enable its drivers on AD lines and fetches the requested datato be placed on thebus.
- It asserts DEVSEL # and maintains it in asserted state until the end of the transaction.
- C/BE # is used to send a bus command in clock cycle and it is used for different purpose during the rest of thetransaction.

- During clock cycle 3, the initiator asserts IRDY #, to indicate that it is ready to receivedata.
- If the target has data ready to send then it asserts TRDY #. In our eg, the target sends 3 more words of data in clock cycle 4 to6.
- The indicator uses FRAME # to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME # during clock cycle5.
- After sending the 4th word, the target disconnects its drivers and negates DEVSEL # during clockcycle7.

Fig: A read operation showing the role of IRDY# /TRY#



- It indicates the pause in the middle of thetransaction.
- The first and words are transferred and the target sends the 3rd word in cycle 5.
- But the indicator is not able to receive it. Hence it negates IRDY#.
- In response the target maintains 3rd data on AD line until IRDY is asserted again.
- In cycle 6, the indicator asserts IRDY. But the target is not ready to transfer the fourth word immediately, hence it negates TRDY in cycle 7. Hence it sends the 4th word and asserts TRDY# at cycle8.

Device Configuration:

- The PCI has a configuration ROM memory that stores information about that device.
- The configuration ROM‟s of all devices are accessible in the configuration address space.
- Theinitializations/wreadtheseROM‟swhenevertheS/Mispowereduporreset
- In each case ,it determines whether the device is a printer, keyboard, Ethernet interface or disk controller.
- Devices are assigned address during initialization process and each device has an w/p signal called IDSEL # (Initialization device select) which has 21 address lines (AD) (AD toAD_{31}).
- During configuration operation, the address is applied to AD i/p of the device and the corresponding AD line is set to and all other lines are set to0.

- AD11 - AD31 →**Upper address line**

A00-A10→**Lower address line** → Specify the type of the operation and to access the content of device configuration ROM.

- The configuration software scans
- all 21 locations. PCI bus has
- interrupt request lines.

Each device may requests an address in the I/O space or memory space

Electrical Characteristics:

- The connectors can be plugged only in compatible motherboards PCI bus can operate with either 5 – 33V power supply.
- The motherboard can operate with signaling system.

Universal Serial Bus(USB):

Universal Serial Bus (USB) is an industry standard developed through a collaborative effort of several computer and communication companies, including Compaq, Hewlett-Packard, Intel, Lucent, Microsoft, Nortel Networks, and Philips.

- USB operate at different Speed
  - Low-speed(1.5 Mb/s)
  - Full-speed(12 Mb/s)
  - High-speed(480 Mb/s)
- USB follow Plug-and-play implementation.

Universal Serial Bus 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. The leaves of the tree are the I/O devices being served (for example, keyboard, Internet connection, speaker, or digital TV)
- 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.

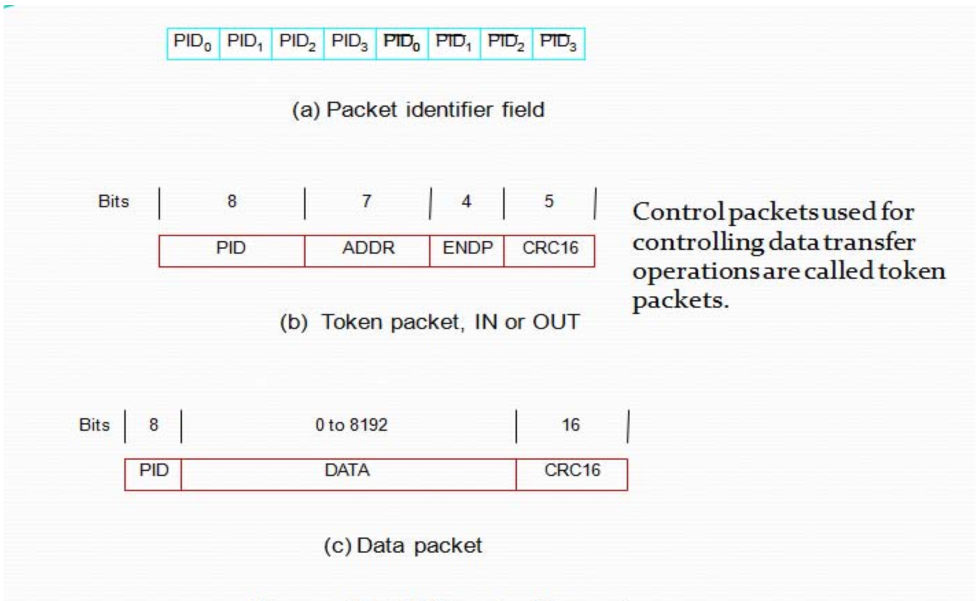
Addressing:

- When a USB is connected to a host computer, its root hub is attached to the processor bus, where it appears as a single device. The host software communicates with individual devices attached to the USB by sending packets of information, which the root hub forwards to the appropriate device in the USB tree.

- 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. When a device is first connected to a hub, or when it is powered on, it has the address 0. The hardware of the hub to which this device is connected is capable of detecting that the device has been connected, and it records this fact as part of its own status information. Periodically, the host polls each hub to collect status information and learn about new devices that may have been added or disconnected.
- When the host is informed that a new device has been connected, it uses a sequence of commands to send a reset signal on the corresponding hub port, read information from the device about its capabilities, send configuration information to the device, and assign the device a unique USB address. Once this sequence is completed the device begins normal operation and responds only to the new address.

USB Protocols

- All information transferred over the USB is organized in packets, where a packet consists of one or more bytes of information. There are many types of packets that perform a variety of control functions.
- The information transferred on the USB can be divided into two broad categories: control and data.
    - Control packets perform such tasks as addressing a device to initiate data transfer, acknowledging that data have been received correctly, or indicating an error.
    - Data packets carry information that is delivered to a device.
- A packet consists of one or more fields containing different kinds of information. The first field of any packet is called the packet identifier, PID, which identifies the type of that packet.
- They are transmitted twice. The first time they are sent with their true values, and the second time with each bit complemented
- The four PID bits identify one of 16 different packet types. Some control packets, such as ACK (Acknowledge), consist only of the PID byte.



PID_0 PID_1 PID_2 PID_3 $\overline{PID_0}$ $\overline{PID_1}$ $\overline{PID_2}$ $\overline{PID_3}$
(a) Packet identifier field
Bits
8
7
4
5
PID
ADDR
ENDP
CRC16
Control packets used for controlling data transfer operations are called token packets.
(b) Token packet, IN or OUT
Bits
8
0 to 8192
16
PID
DATA
CRC16
(c) Data packet

Figure 45. USB packet format.

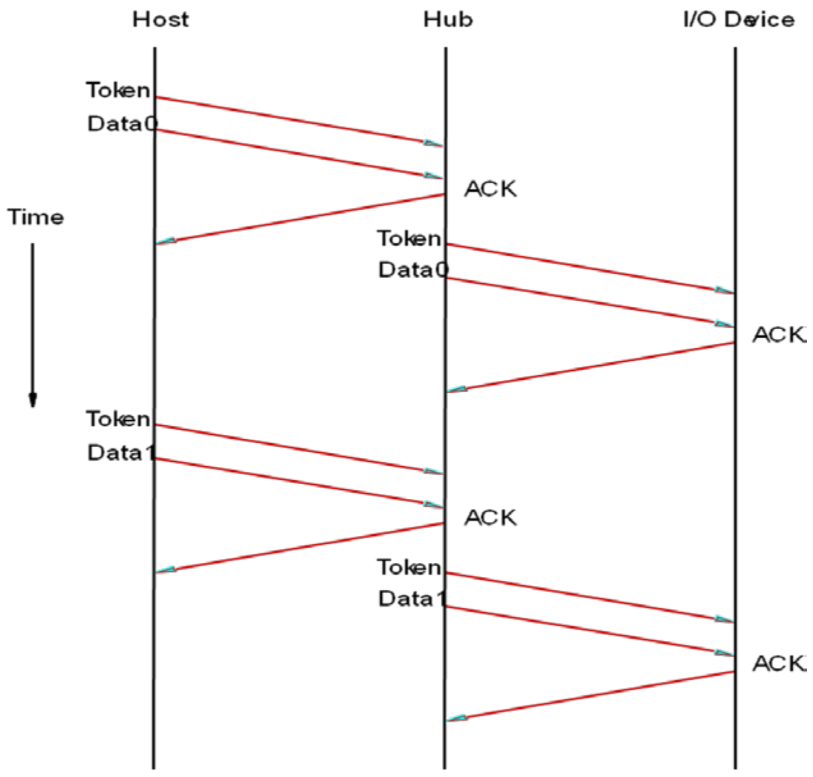


Figure: An output transfer

Isochronous traffic on USB

- One of the key objectives of the USB is to support the transfer of isochronous data.
- Devices that generates or receives isochronous data require a time reference to control the sampling process.
- To provide this reference. Transmission over the USB is divided into frames of equal length.
- A frame is 1ms long for low-and full-speed data.

- The root hub generates a Start of Frame control packet (SOF) precisely once every 1 ms to mark the beginning of a new frame.
- The arrival of an SOF packet at any device constitutes a regular clock signal that the device can use for its own purposes.
- To assist devices that may need longer periods of time, the SOF packet carries an 11-bit frame number.
- Following each SOF packet, the host carries out input and output transfers for isochronous devices.
- This means that each device will have an opportunity for an input or output transfer once every 1 ms.

Electrical Characteristics:

- The cables used for USB connections consist of four wires.
- Two are used to carry power, +5V and Ground.
    - Thus, a hub or an I/O device may be powered directly from the bus, or it may have its own external power connection.
- The other two wires are used to carry data.
- Different signaling schemes are used for different speeds of transmission.
    - At low speed, 1s and 0s are transmitted by sending a high voltage state (5V) on one or the other o the two signal wires. For high-speed links, differential transmission is used.

Input-Output Processor:

PIPELINE AND VECTOR PROCESSING

Parallel processing:

- Parallel processing is a term used for a large class of techniques that

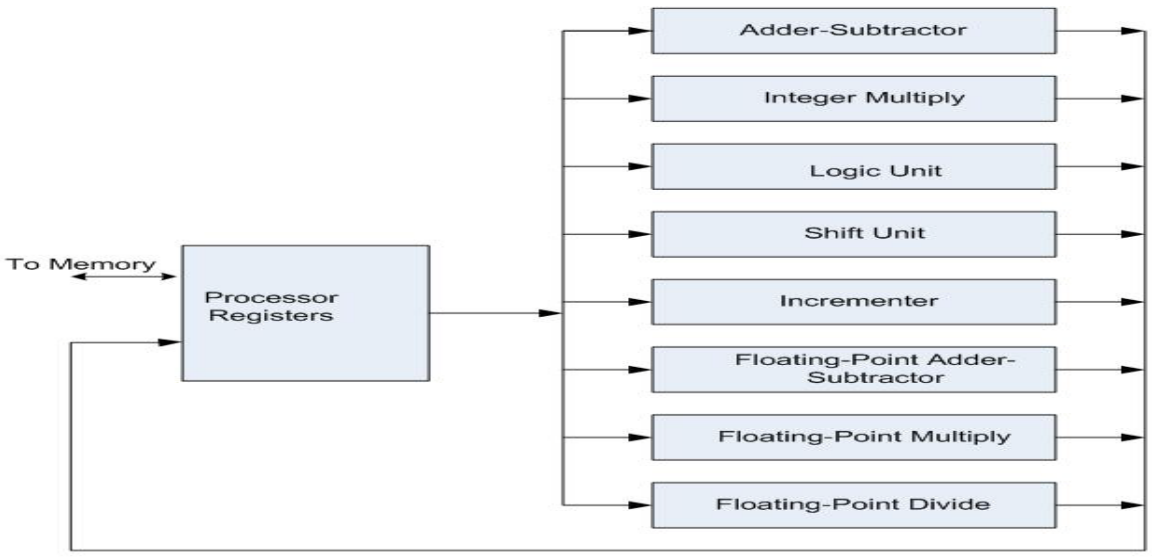
are used to provide simultaneous data-processing tasks for the purpose of increasing the computational speed of a computer system.

- It refers to techniques that are used to provide simultaneous data processing.
- The system may have two or more ALUs to be able to execute two or more instruction at the same time.
- The system may have two or more processors operating concurrently.
- It can be achieved by having multiple functional units that perform same or different operation simultaneously.

- Example of parallel Processing:
  - Multiple Functional Unit:

Separate the execution unit into eight functional units operating in parallel.

- There are variety of ways in which the parallel processing can be classified
  - Internal Organization of Processor
  - Interconnection structure between processors
  - Flow of information through system



Architectural Classification:

- Flynn's classification
  - » Based on the multiplicity of *Instruction Streams* and *Data Streams*
  - » Instruction Stream
    - Sequence of Instructions read from memory
  - » Data Stream
    - Operations performed on the data in the processor

		Number of *Data Streams*	
		Single	Multiple
Number of *Instruction Streams*	Single	SISD	SIMD
	Multiple	MISD	MIMD

- SISD represents the organization containing single control unit, a processor unit and a memory unit. Instruction are executed sequentially and system may or may not have internal parallel processing capabilities.
- SIMD represents an organization that includes many processing units under the supervision of a common control unit.
- MISD structure is of only theoretical interest since no practical system has been constructed using this organization.
- MIMD organization refers to a computer system capable of processing several programs at the same time.

The main difference between multicomputer system and multiprocessor system is that the multiprocessor system is controlled by one operating system that provides interaction between processors and all the component of the system cooperate in the solution of a problem.

- Parallel Processing can be discussed under following topics:
  - Pipeline Processing
  - Vector Processing
  - Array Processors

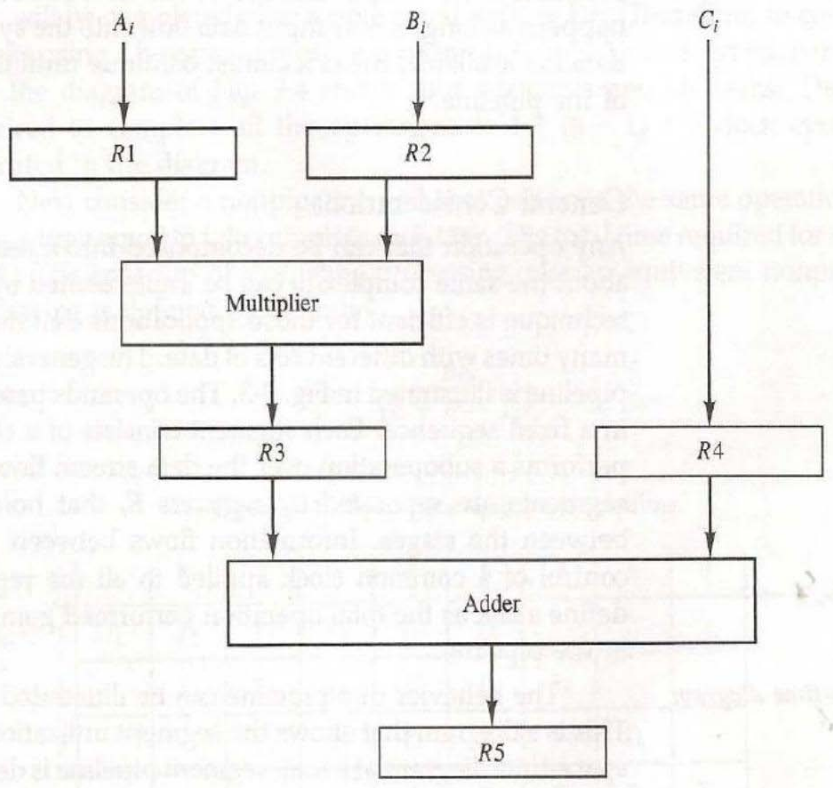
PIPELINING:

- A technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments.
- It is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segments that operates concurrently with all other segments.
- Each segment performs partial processing dictated by the way task is partitioned.
- The result obtained from each segment is transferred to next segment.
- The final result is obtained when data have passed through all segments.

- Suppose we have to perform the following task:
- Each sub operation is to be performed in a segment within a pipeline. Each segment has one or two registers and a combinational circuit.

$$A_i * B_i + C_i \qquad \text{for } i = 1, 2, 3, \ldots, 7$$

Figure Example of pipeline processing.

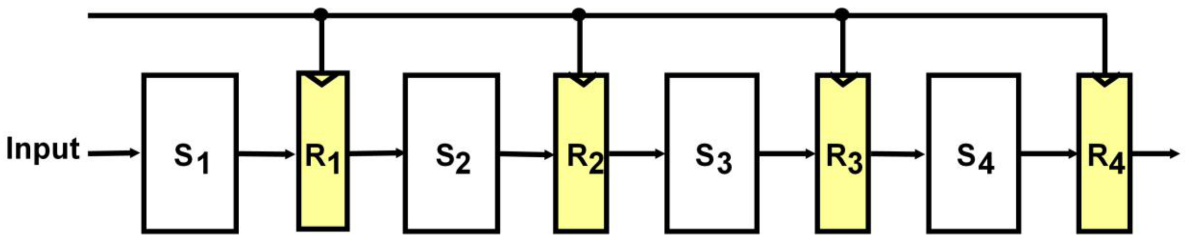


Suboperations in each segment:	$R1 \leftarrow A_i$, $R2 \leftarrow B_i$	Load A_i and B_i
	$R3 \leftarrow R1 * R2$, $R4 \leftarrow C_i$	Multiply and load C_i
	$R5 \leftarrow R3 + R4$	Add

OPERATIONS IN EACH PIPELINE STAGE:

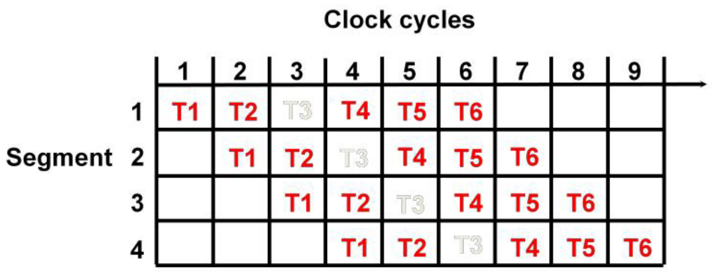
Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A1	B1	---	---	-------
2	A2	B2	A1 * B1	C1	-------
3	A3	B3	A2 * B2	C2	A1 * B1 + C1
4	A4	B4	A3 * B3	C3	A2 * B2 + C2
5	A5	B5	A4 * B4	C4	A3 * B3 + C3
6	A6	B6	A5 * B5	C5	A4 * B4 + C4
7	A7	B7	A6 * B6	C6	A5 * B5 + C5
8			A7 * B7	C7	A6 * B6 + C6
9					A7 * B7 + C7

- General Structure of a 4-Segment Pipeline



- Space-Time Diagram

The following diagram shows 6 tasks T1 through T6 executed in 4segments.



Segment \ Clock cycles	1	2	3	4	5	6	7	8	9
1	T1	T2	T3	T4	T5	T6			
2		T1	T2	T3	T4	T5	T6		
3			T1	T2	T3	T4	T5	T6	
4				T1	T2	T3	T4	T5	T6

No matter how many segments, once the pipeline is full, it takes only one clock period to obtain an output.

PIPELINE SPEEDUP:

Consider the case where a k-segment pipeline used to execute n tasks.

- n = 6 in previous example

- ➢ k = 4 in previous example

- Pipelined Machine (k stages, n tasks)
  - ➢ The first task t1 requires k clock cycles to complete its operation since there are k segments
  - ➢ The remaining n-1 tasks require n-1 clock cycles
  - ➢ The n tasks clock cycles = k+(n-1) (9 in previous example)
- Conventional Machine (Non-Pipelined)
  - ➢ Cycles to complete each task in nonpipeline = k
  - ➢ For n tasks, n cycles required is
- Speedup (S)
  - ➢ S = Nonpipeline time /Pipeline time
- ➢ For n tasks: S = nk/(k+n-1)
- ➢ As n becomes much larger than k-1; Therefore, S = nk/n = k

PIPELINE AND MULTIPLE FUNCTION UNITS:

Example:

- 4-stage pipeline

- 100 tasks to be executed

- 1 task in non-pipelined system; 4 clock cycles

Pipelined System : k + n - 1 = 4 + 99 = 103 clock cycles

Non-Pipelined System : n*k = 100 * 4 = 400 clock cycles

Speedup :S_k = 400 / 103 = 3.88

- Arithmetic Pipeline
- Instruction Pipeline

ARITHMETIC PIPELINE:

- Pipeline arithmetic units are usually found in very high speed computers.
- They are used to implement floating point operations.

- We will now discuss the pipeline unit for the floating point addition and subtraction.
- The inputs to floating point adder pipeline are two normalized floating point numbers.
- A and B are mantissas and a and b are the exponents.
- The floating point addition and subtraction can be performed in four segments. Floating-point adder:

[1] Compare the exponents

[2] Align the mantissa

[3] Add/sub the mantissa

[4] Normalize the result

1) Compare exponents :

3 - 2 = 1

2) Align mantissas

$X = 0.9504 \times 10^3$

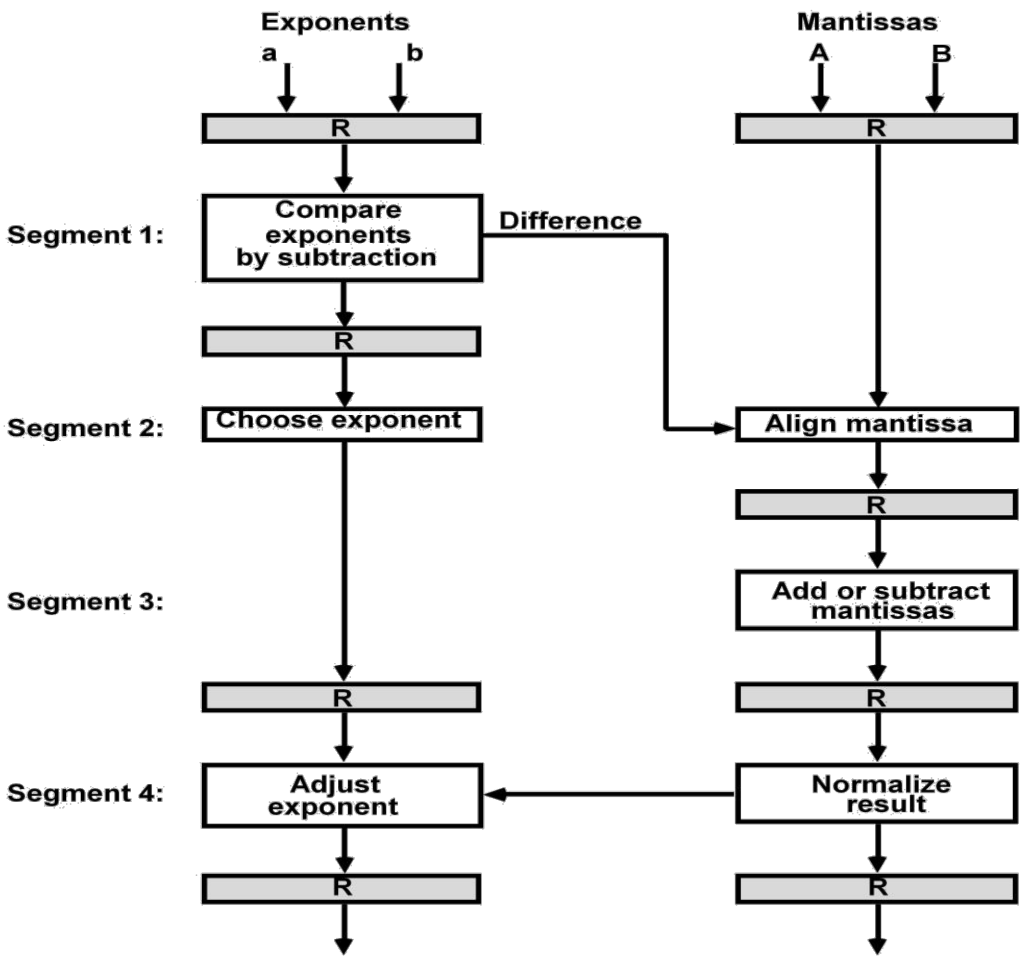
$Y = 0.08200 \times 10^3$

3) Add mantissas

$Z = 1.0324 \times 10^3$

4) Normalize result

$Z = 0.10324 \times 10^4$



Instruction Pipeline:

- Pipeline processing can occur not only in the data stream but in the instruction stream as well.
- An instruction pipeline reads consecutive instruction from memory while previous instruction are being executed in other segments.
- This caused the instruction fetch and execute segments to overlap and perform simultaneous operation.

Four Segment CPU Pipeline:

- FI segment fetches the instruction.
- DA segment decodes the instruction and calculate the effective address.
- FO segment fetches the operand.
- EX segment executes the instruction.

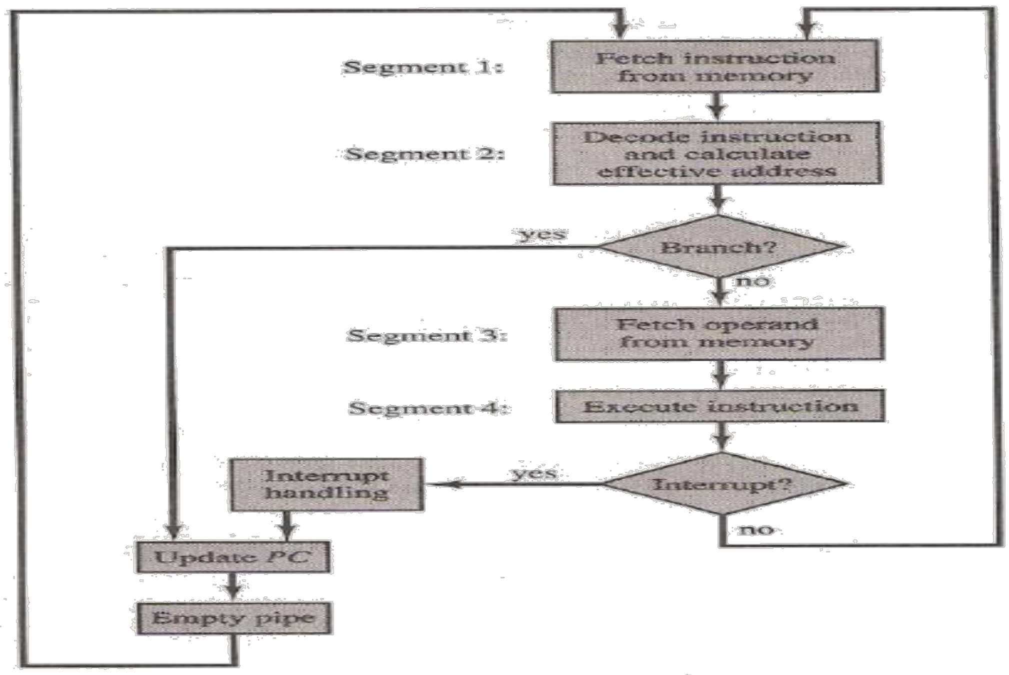
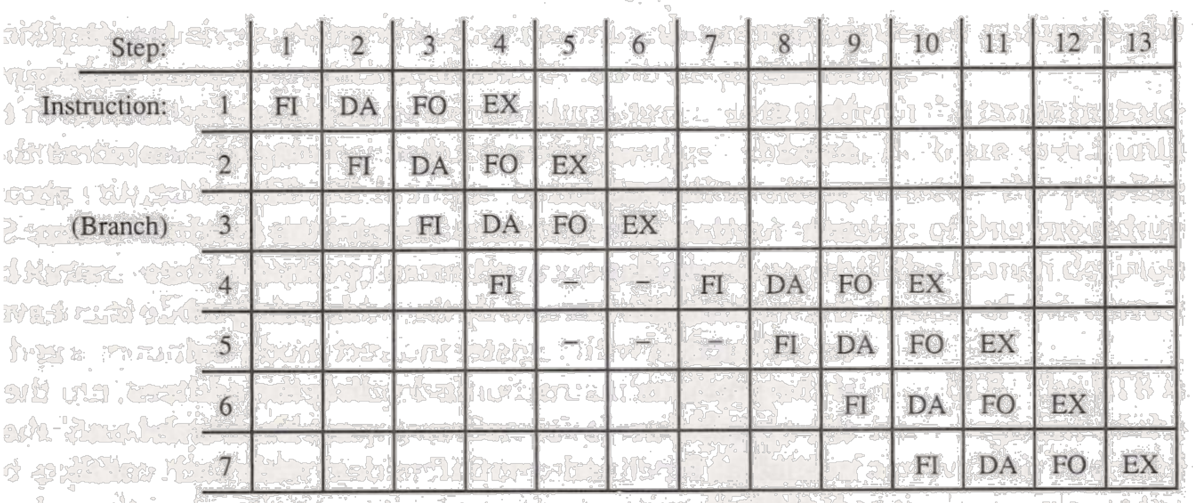


Figure Four-segment CPU pipeline.



Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	—	—	FI	DA	FO	EX			
	5					—	—	—	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

Figure Timing of instruction pipeline.

INSTRUCTION CYCLE:

Pipeline processing can occur also in the instruction stream. An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. Six Phases* in an Instruction Cycle

[1] Fetch an instruction from memory

[2] Decode the instruction

[3] Calculate the effective address of the operand

[4] Fetch the operands from memory

[5] Execute the operation

[6] Store the result in the proper place

* Some instructions skip some phases

* Effective address calculation can be done in the part of the decoding phase

* Storage of the operation result into a register is done automatically in the execution phase

==> 4-Stage Pipeline

[1] FI: Fetch an instruction from memory

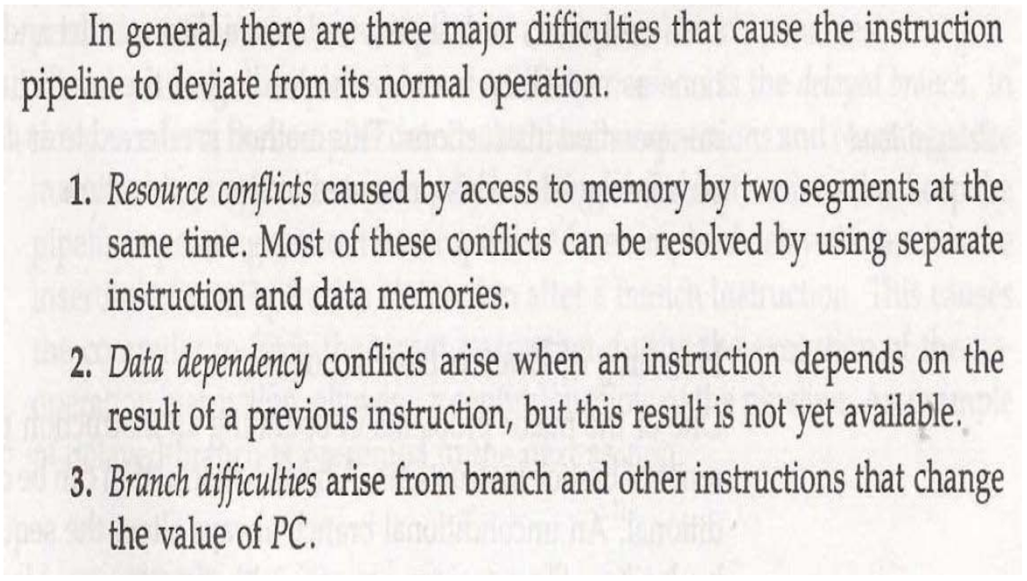
[2] DA: Decode the instruction and calculate the effective address of the operand

[3] FO: Fetch the operand

[4] EX: Execute the operation

Pipeline Conflicts :

– Pipeline Conflicts : 3 major difficulties


In general, there are three major difficulties that cause the instruction pipeline to deviate from its normal operation.

1. *Resource conflicts* caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.
2. *Data dependency* conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.
3. *Branch difficulties* arise from branch and other instructions that change the value of PC.

–

1) Resource conflicts: memory access by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2) Data dependency: when an instruction depend on the result of a previous instruction, but this result is not yet available.

Example: an instruction with register indirect mode cannot proceed to fetch the operand if the previous instruction is loading the address into the register.

3) Branch difficulties: branch and other instruction (interrupt, ret, ..) that change the value of PC.

Handling Data Dependency:

- This problem can be solved in the following ways:
  - Hardware interlocks: It is the circuit that detects the conflict situation and delayed the instruction by sufficient cycles to resolve the conflict.
  - Operand Forwarding: It uses the special hardware to detect the conflict and avoid it by routing the data through the special path between pipeline segments.
  - Delayed Loads: The compiler detects the data conflict and reorder the instruction as necessary to delay the loading of the conflicting data by inserting no operation instruction.

Handling of Branch Instruction:

- Pre fetch the target instruction.
- Branch target buffer(BTB) included in the fetch segment of the pipeline
- Branch Prediction
- Delayed Branch

RISC Pipeline:

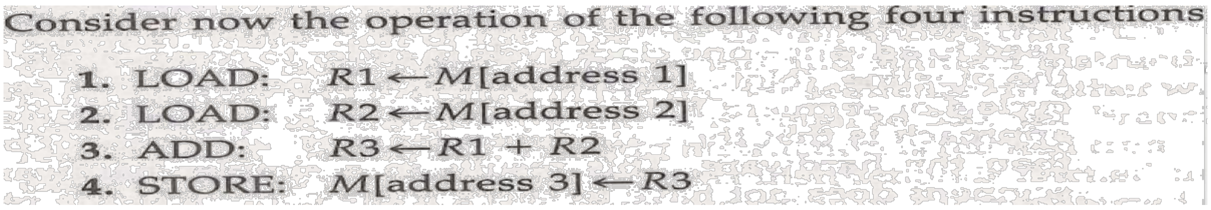
- Simplicity of instruction set is utilized to implement an instruction pipeline using small number of sub-operation, with each being executed in single clock cycle.

Since all operation are performed in the register, there is no need of effective address calculation.

Three Segment Instruction Pipeline:

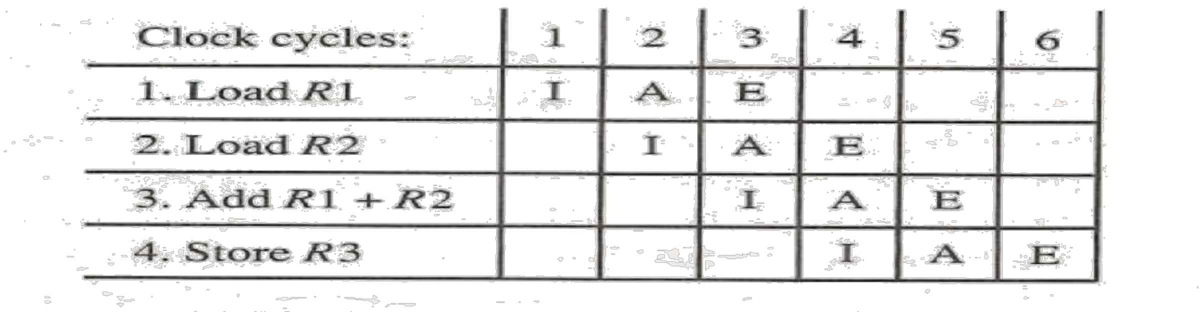
- I: Instruction Fetch
- A: ALU Operation
- E: Execute

Instruction Delayed Load:



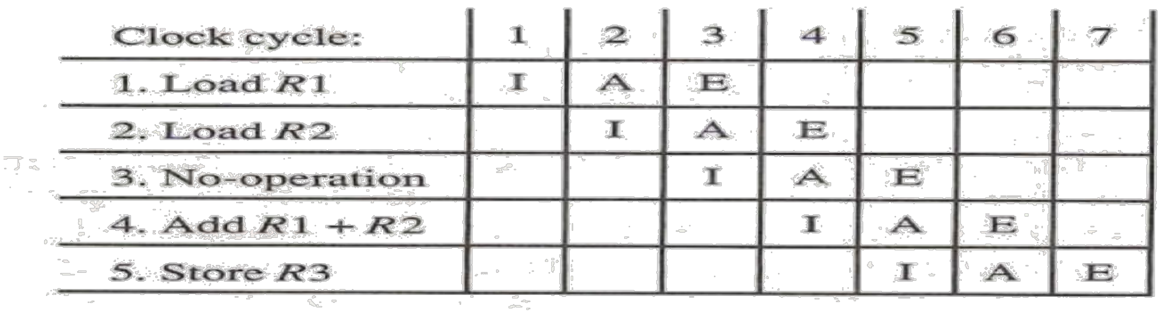
Consider now the operation of the following four instructions

1. LOAD: $R1 \leftarrow M[\text{address } 1]$
2. LOAD: $R2 \leftarrow M[\text{address } 2]$
3. ADD: $R3 \leftarrow R1 + R2$
4. STORE: $M[\text{address } 3] \leftarrow R3$



Clock cycles:	1	2	3	4	5	6
1. Load $R1$	I	A	E			
2. Load $R2$		I	A	E		
3. Add $R1 + R2$			I	A	E	
4. Store $R3$				I	A	E

Pipeline timing with data conflict

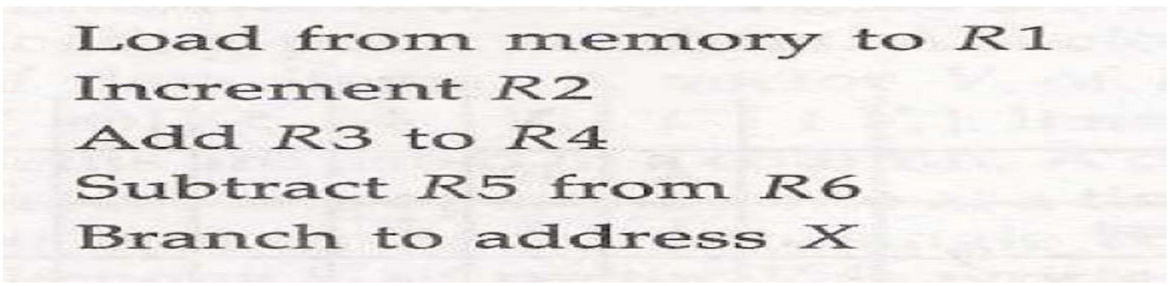


Clock cycle:	1	2	3	4	5	6	7
1. Load $R1$	I	A	E				
2. Load $R2$		I	A	E			
3. No-operation			I	A	E		
4. Add $R1 + R2$				I	A	E	
5. Store $R3$					I	A	E

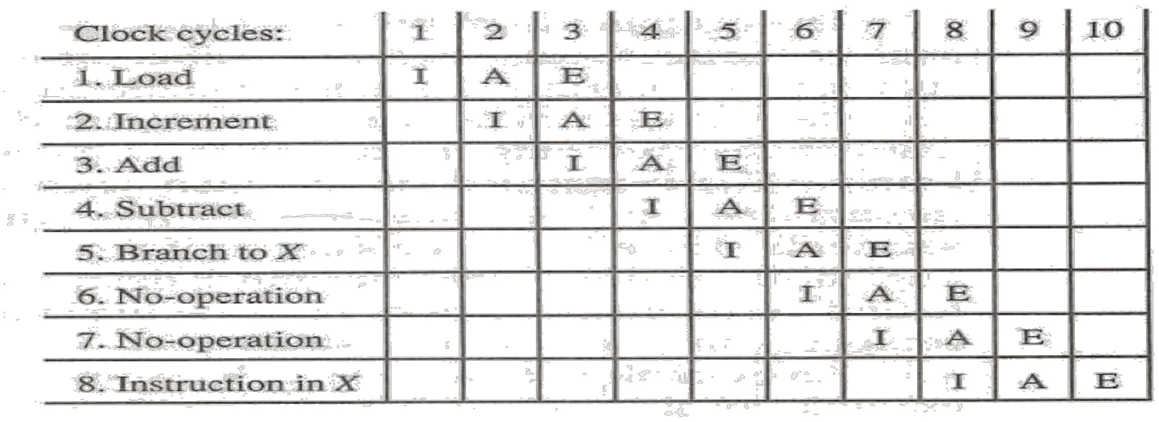
Pipeline timing with delayed load

Delayed Branch:

Let us consider the program having the following 5 instructions



Load from memory to $R1$
Increment $R2$
Add $R3$ to $R4$
Subtract $R5$ from $R6$
Branch to address X



Clock cycles:	1	2	3	4	5	6	7	8	9	10
1. Load	I	A	E							
2. Increment		I	A	E						
3. Add			I	A	E					
4. Subtract				I	A	E				
5. Branch to *X*					I	A	E			
6. No-operation						I	A	E		
7. No-operation							I	A	E	
8. Instruction in *X*								I	A	E

Using no-operation instructions

Organization of Intel 8085 Micro-Processor:

The microprocessors that are available today came with a wide variety of capabilities and architectural features. All of them, regardless of their diversity, are provided with at least the following functional components, which form the central processing unit (CPU) of a classical computer.

1. Register Section : A set of registers for temporary storage of instructions, data and address of data .
2. Arithmetic and Logic Unit : Hardware for performing primitive arithmetic and logical operations .
3. Interface Section : Input and output lines through which the microprocessor communicates with the outside world .
4. Timing and Control Section : Hardware for coordinating and controlling the activities of the various sections within the microprocessor and other devices connected to the interface section .

The block diagram of the microprocessor along with the memory and Input/Output (I/O) devices is shown in the Figure 11.1.

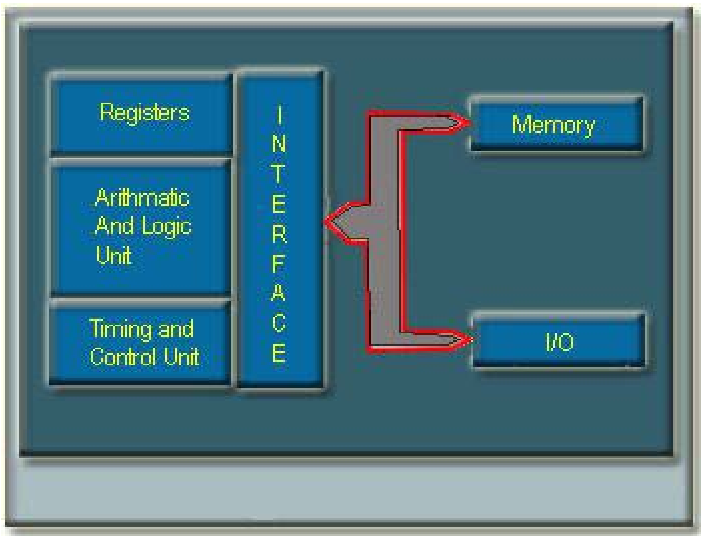


Figure 11.1: Block diagram of Micorprocessor with memory and I/O.

Intel Microprocessors:

Intel 4004 is the first 4-bit microprocessor introduced by Intel in 1971. After that Intel introduced its first 8-bit microprocessor 8088 in 1972.

These microprocessors could not last long as general-purpose microprocessors due to their design and performance limitations.

In 1974, Intel introduced the first general purpose 8-bit microprocessor 8080 and this is the first step of Intel towards the development of advanced microprocessor.

After 8080, Intel launched microprocessor 8085 with a few more features added to its architecture, and it is considered to be the first functionally complete microprocessor.

The main limitations of the 8-bit microprocessors were their low speed, low memory capacity, limited number of general purpose registers and a less powerful instruction set .

To overcome these limitations Intel moves from 8-bit microprocessor to 16-bit microprocessor.

In the family of 16-bit microprocessors, Intel's 8086 was the first one introduced in 1978 .

8086 microprocessor has a much powerful instruction set along with the architectural developments, which imparted substantial programming flexibility and improvement over the 8-bit microprocessor.

Microprocessor Intel 8085 :

Intel 8085 is the first popular microprocessor used by many vendors. Due to its simple architecture and organization, it is easy to understand the working principle of a microprocessor.

Register in the Intel 8085:

The programmable registers of 8085 are as follows -

- One 8-bit accumulator A.
- Six 8-bit general purpose register (GPR's)
  B, C, D , E , H and L.
- The GPR's are also accessible as three 16-bit register pairs BC, DE and HL.
    - There is a 16-bit program counter(PC), one 16-bit stack pointer(SP) and 8-bit flag register . Out of 8 bits of the flag register , only 5 bits are in use.

The programmable registers of the 8085 are shown in the Figure 11.2-

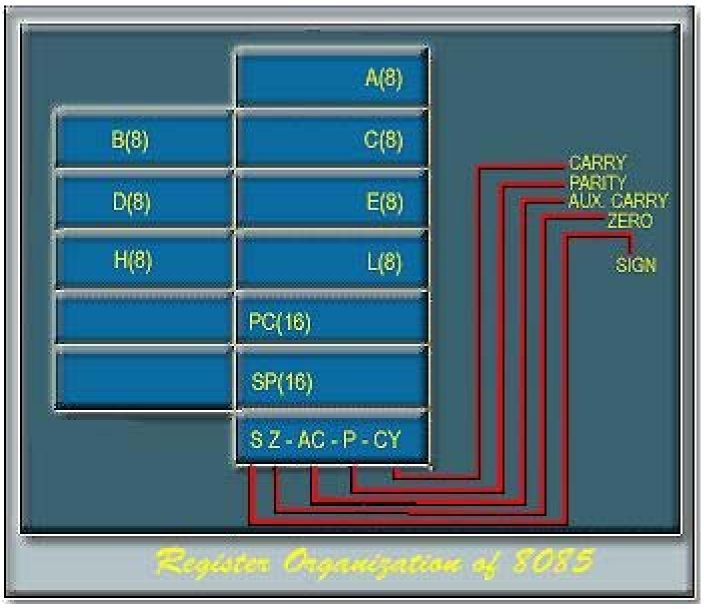


Figure 11.2: Register Organisation of 8085

Apart from these programmable registers , some other registers are also available which are not accessible to the programmer . These registers include -

- Instruction Register(IR).
- Memory address and data buffers(MAR & MDR).
  - MAR: Memory Address Register.
  - MDR: Memory Data Register.
- Temporary register for ALU use.

ALU of 8085 :

The 8-bit parallel ALU of 8085 is capable of performing the following operations –

Arithmetic : Addition, Subtraction, Increment, Decrement, Compare.

Logical : AND, OR, EXOR, NOT, SHIFT / ROTATE, CLEAR.

Because of limited chip area , complex operations like multiplication, division, etc are not available, in earlier processors like 8085.

The operations performed on binary 2's complement data.

The five flag bits give the status of the microprocessor after an ALU operation.

The carry (C) flag bit indicates whether there is any overflow from the MSB.

The parity (P) flag bit is set if the parity of the accumulater is even.

The Auxiliary Carry (AC) flag bit indicates overflow out of bit –3 (lower nibble) in the same manner, as the C-flag indicates the overflow out of the bit-7.

The Zero (Z) flag bit is set if the content of the accumulator after any ALU operations is zero.

The Sign(S) flag bit is set to the condition of bit-7 of the accumulator as per the sign of the contents of the accumulator(positive or negative).

The Interface Section:

Microprocessor chips are equipped with a number of pins for communication with the outside world. This is known as the system bus.
The interface lines of the Intel 8085 microprocessor are shown in the Figure 11.3 –

Address and Data Bus

The AD0 - AD7 lines are used as lower order 8-bit address bus and data bus , in time division multiplexed manner .

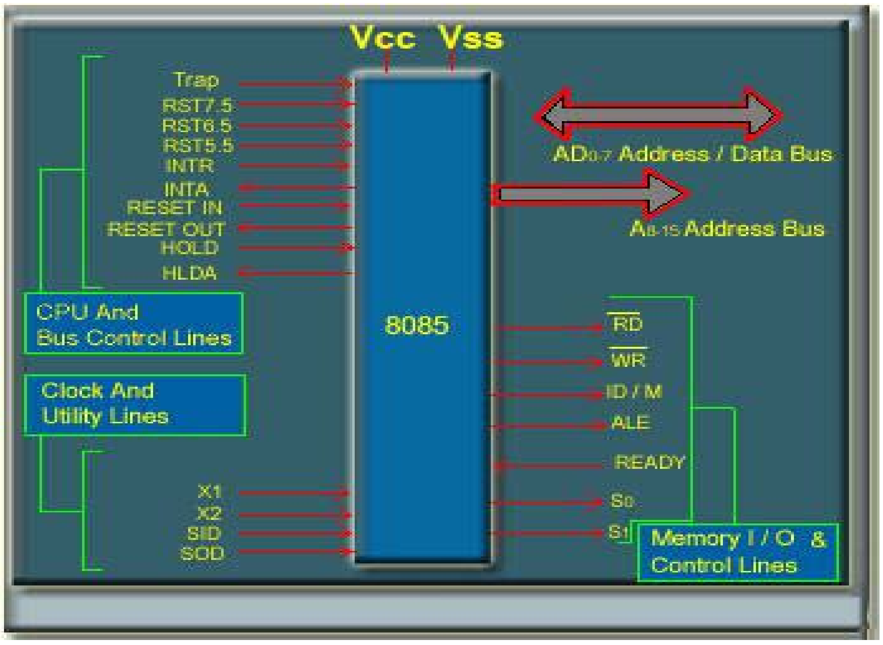
The A8 - A15 lines are used for higher order 8 bit of address bus.

There are seven memory and I/O control lines -

RD : indicates a READ operation when the signal is LOW .

WR : indicates a WRITE operation when the signal is LOW .

IO/M : indicates memory access for LOW and I/O access for HIGH .



ALE : ALE is an address latch enable signal , this signal is HIGH when address information is present in AD0-AD7 . The falling edge of ALU can be used to latch the address into an external buffer to de-multiples the address bus .

READY : READY line is used for communication with slow memory and I/O devices .

S0 and S1 : The status of the system bus is difined by the S0 and S1 lines as follows -

S1	S0	Operation Specified
0	0	Halt
0	1	Memory or I/O WRITE
1	0	Memory or I/O READ
1	1	Instruction Fetch

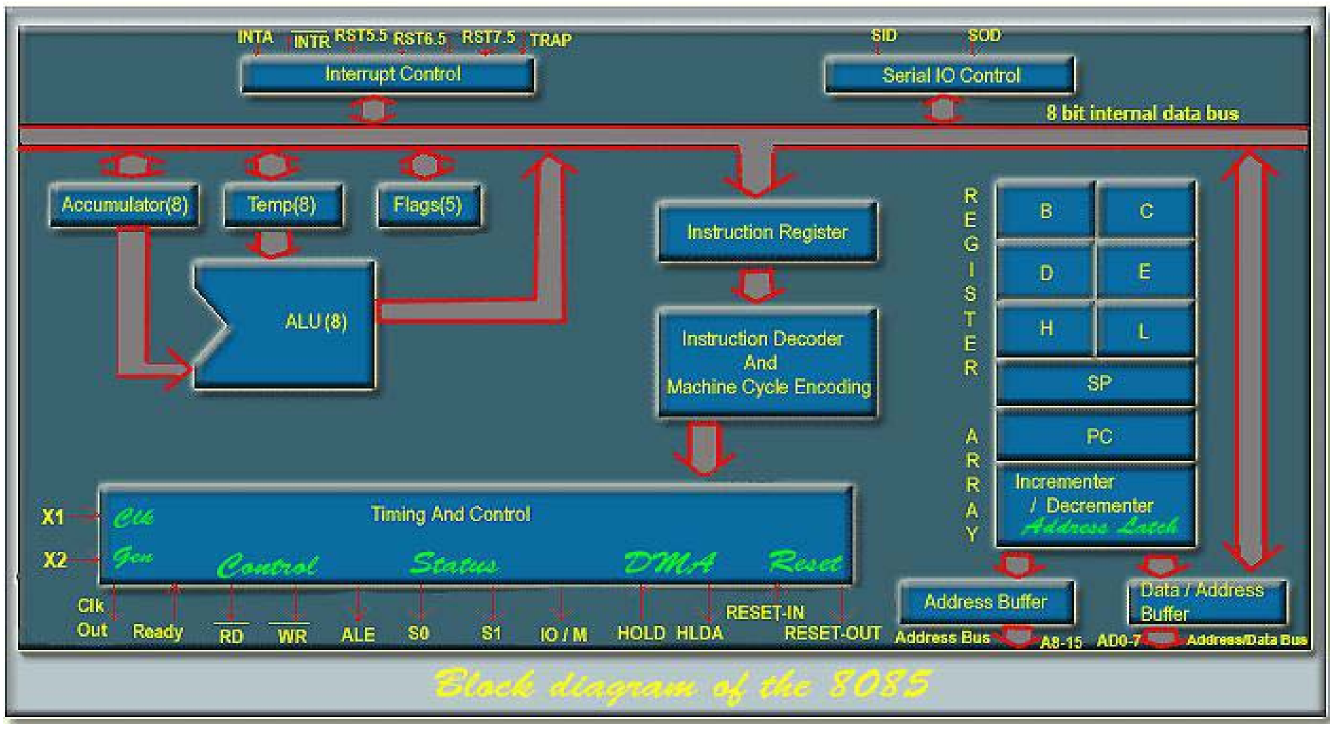
There are ten lines associated with CPU and bus control-

- TRAP , RST7.5 , RST6.5 , RST5.5 and INTR are the Interrupt lines.
- INTA: Interrupt acknowledge line.
- RESET IN : This is the reset input signal to the 8085.
- RESET OUT : The 8085 generates the RESET-OUT signal in response to RESET-IN signal , which can be used as a system reset signal .
- HOLD : HOLD signal is used for DMA request.
- HLDA : HLDA signal is used for DMA grant .

- Clock and Utility Lines :

X1 and X2: X1 and X2 are provided to connect a crystal or a RC network for generating theclockinternaltothe chip.
Sid: input line for serial data communication.
Sod: output line for serial data communication.
V_{CC} and v_{SS}: power supply.

- The block diagram of the Intel 8085 is shown in the Figure 11.4 -



Addressing Modes :

The 8085 has four different modes for addressing data stored in memory or in registers -

Direct: Bytes 2 and 3 of the instruction contains the exact memory address of the data item(the low-order bits of the address are in byte 2 , the high-order bits in byte 3).

Register: The instruction specifies the register or register pair in which the data are located.

Register Indirect: The instruction specifies a register pair which contains the memory address where the data are located .(the high-order bits of the address are in the first register of the pair and the low order bits in the second).

Immediate: The instruction contains the data itself . This is either and 8-bit quantity or a 16-bit quantity (least significant byte first , most significant byte second).

Unless directed by an interrupt or branch instruction the execution of instructions proceeds through consecutively increasing memory locations.

A branch instruction can specify the address of the next instruction to be executed in one of two ways -

Direct: The branch instruction contains the address of the next instruction to be executed .

REFERENCE :

1. COMPUTER SYSTEM ARCHITECTURE, MORRIS M. MANO, 3RD EDITION, PRENTICE HALL INDIA.
2. HTTP://NPTEL.AC.IN/COURSES