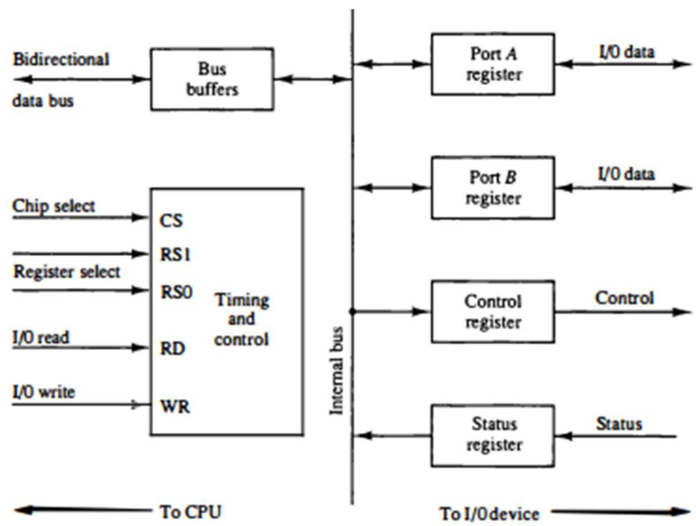
UNIT – V

41.	Give an Example of I/O interface unit.	8	Section-5	5

Ans.

- An example of an l/O interface unit is shown in block diagram form in Fig. 11-2. It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus. The chip select and register select inputs determine the address assigned to the interface. The l/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the l/O device attached to the interface.
- The l/O data to and from the device can be transferred into either port A or port B. The interface may operate with an output device or with an input device, or with a device that requires both input and output. If the interface is connected to a printer, it will only output data, and if it services a character reader, it will only input data. A magnetic disk unit transfers data in both directions but not at the same time, so the interface can use bidirectional lines. A command is passed to the l/O device by sending a word to the appropriate interface register. In a system like this, the function code in the l/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 via the common data bus. The distinction between data, control, or status information is determined from the particular interface register with which the CPU communicates.



CS	RS1	RS0	Register selected
0	×	×	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

Figure 11-2 Example of I/O interface unit.

- The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the l/O device attached to it can be placed in a variety of operating modes. For example, port A may be defined as an input port and port B as an output 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 example, a status bit may indicate that portA has received a new data item from the 110 device. Another bit in the status register may indicate that a parity error has occurred during the 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 (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (CS) input when the interface is selected by the address bus. The two register select inputs RSl and RSO are usually connected to the two least significant lines of the address bus. These two inputs select one of the four registers in the interface as specified in the table accompanying the diagram. The content of the selected register is transfer into the CPU via the data bus when the VO read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the VO write input is enabled.

42.	Explain Connection of I/O bus to input–output devices.	8	Section-5	5

Ans.

Input-Output:

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

Input-Output Bus to Input-Output Devices:

- A typical communication link between the processor and several peripherals is shown in Fig. 11·1. The 1/0 bus consists of data lines, address lines, and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. The magnetic tape is used in some computers for backup storage. Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the UO bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device. For example, the printer controller controls the paper motion, the print timing, and the selection of printing characters. A controller may be housed separately or may be physically integrated with the peripheral.
- The I/O bus from the processor is attached to all peripheral interfaces. To communicate with a particular device, the processor places a device address on the address lines. Each interface attached to the I/O bus contains an address decoder that monitors the address lines. When the interface detects its own address, it activates the path between the bus lines and the device that it controls. All peripherals whose address does not correspond to the address in the bus are disabled by their interface.
- At the same time that the address is made available in the address lines, the processor provides a function code in the control lines. The interface selected responds to the function code and proceeds to execute it. The function code is referred to as an I/O command and is in essence an instruction that is executed in the interface and its attached peripheral unit. The interpretation of the command depends on the peripheral that the processor is addressing. There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.

I/O Bus and Interface Modules

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

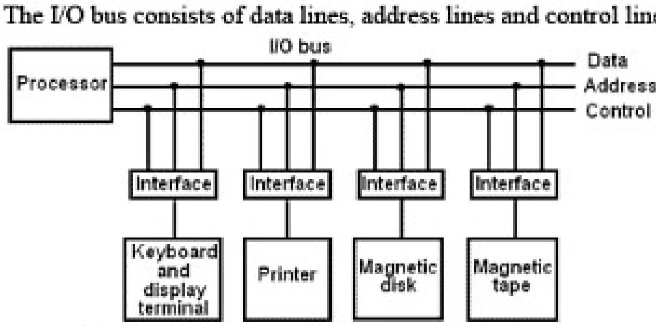


Fig: Connection of I/O bus to input-output devices

- A **control command** is issued to activate the peripheral and to inform it what to do. For example, a magnetic tape unit may be instructed to backspace the tape by one record, to rewind the tape, or to start the tape moving in the forward direction. The particular control command issued depends on the peripheral, and each peripheral receives its own distinguished sequence of control commands, depending on its mode of operation.
- A **status command** is used to test various status conditions in the interface and the peripheral. For example, the computer may wish to check the status of the peripheral before a transfer is initiated. During the transfer, one

or more errors may occur which are detected by the interface. These errors are designated by setting bits in a status register that the processor can read at certain intervals.

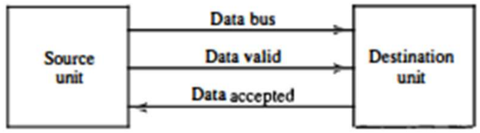
- A **data output command** causes the interface to respond by transferring data from the bus into one of its registers. Consider an example with a tape unit. The computer starts the tape moving by issuing a control command. The processor then monitors the status of the tape by means of a status command. When the tape is in the correct position, the processor issues a data output command. The interface responds to the address and command and transfers the information from the data lines in the bus to its buffer register. The interface then communicates with the tape controller and sends the data to be stored on tape.
- The **data input command** is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register. The processor checks if data are available by means of a status command and then issues a data input command. The interface places the data on the data lines, where they are accepted by the processor.

43.	What is meant by handshaking? Explain with neat diagram.	8	Section-5	5

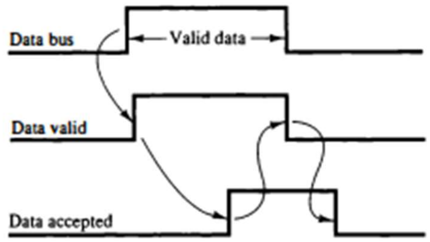
Ans.

Handshaking

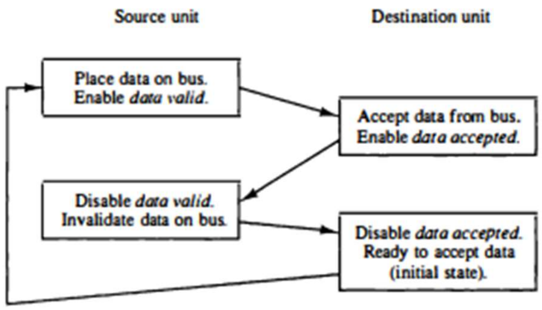
- The disadvantage of the strobe method is that the source unit that initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on the bus. The handshake method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer. The basic principle of the two-wire handshaking method of data transfer is as follows. One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept data. The sequence of control during the transfer depends on the unit that initiates the transfer.
- Figure 11-5 shows the data transfer procedure when initiated by the source. The two handshaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit. The timing diagram shows the exchange of signals between the two units. The sequence of events listed in part (c) shows the 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 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. This scheme allows arbitrary delays from one state to the next and permits each unit to respond at its own data transfer rate. The rate of transfer is determined by the slowest unit.



(a) Block diagram



(b) Timing diagram

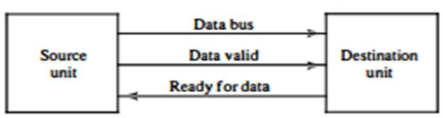


(c) Sequence of events

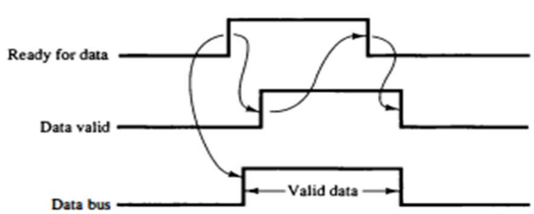
Figure 11-5 Source-initiated transfer using handshaking.

- The destination-initiated transfer using handshaking lines is shown in Fig. 11-6. Note that 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. Note that the sequence of events in both cases would be identical if we consider the ready for data signal as the complement of data accepted. In fact, the only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.

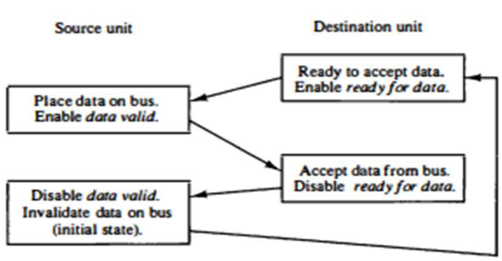
Figure 11-6 Destination-initiated transfer using handshaking.



(a) Block diagram



(b) Timing diagram



(c) Sequence of events

- 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 timeout completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed within a predetermined time. The timeout 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 within a given time period, the unit assumes that an error has occurred. The timeout signal can be used to interrupt the processor and hence execute a service routine that takes appropriate error recovery action.

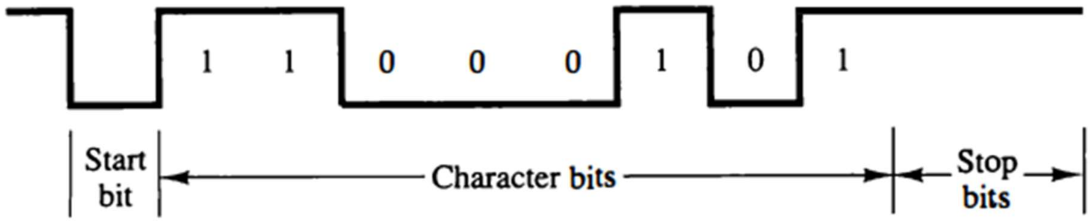
44.	What is Asynchronous Serial Transfer.	8	Section-5	5

Ans.

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. This means that an 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. This method reqttires the use of one pair of conductors or one conductor and a common ground. Parallel transmission is faster but requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive since it requires only one pair of conductors.
- Serial transmission can be synchronous or asynchronous. In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. In long distant serial transmission, each unit is driven by a separate clock of the same frequency. Synchronization signals are transmitted periodically between the two units to keep their clocks in step with each other. In asynchronous transmission, binary information is sent only when it is available and the line remains idle when there is no information to be transmitted. This is in contrast to synchronous transmission, where bits must be transmitted continuously to keep the clock frequency in both units synchronized with each other. Synchronous serial transmission is discussed further in Sec. 11-8.
- A serial asynchronous data transmission technique used in many interactive terminals employs special bits that are inserted at both ends of the character code. With this technique, each character consists of three parts: a start bit, the character bits, and stop bits. The convention is that the transmitter rests at the 1-state when no characters are transmitted. The first bit, called the start bit, is always a 0 and is used to indicate the beginning of a character. The last bit called the stop bit is always a 1. An example of this format is shown in Fig. 11-7.
- 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 at least one bit time.
- Using these rules, the receiver can detect the start bit when the line goes from 1 to 0. A clock in the receiver examines the line at proper bit times. The receiver knows the transfer rate of the bits and the number of character bits to accept. After the character bits are transmitted, one or two stop bits are sent. The stop bits are always in the 1-state and frame the end of the character to signify the idle or wait state.
- At the end of the character the line is held at the 1-state for a period of at least one or two bit times so that both the transmitter and receiver can resynchronize. The length of time that the line stays in this state depends on the amount of time required for the equipment to resynchronize. Some older electromechanical terminals use two stop bits, but newer terminals use one stop bit. The line remains in the 1-state until another character is transmitted. The stop time ensures that a new character will not follow for one or two bit times.
- As an illustration, consider the serial transmission of a terminal whose transfer rate is 10 characters per second. Each transmitted character consists of a start bit, eight information bits, and two stop bits, for a total of 11 bits. Ten characters per second means that each character takes 0.1 s for transfer. Since there are 11 bits to be transmitted, it follows that the bit time is 9.09 ms. The baud rate is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second. Ten characters per second with an 11-bit format has a transfer rate of 110 baud.

Figure 11-7 Asynchronous serial transmission.



- The terminal has a keyboard and a printer. Every time a key is depressed, the terminal sends 11 bits serially along a wire. To print a character in the printer, an 11-bit message must be received along another wire. The terminal interface consists of a transmitter and a receiver. The transmitter accepts an 8-bit character from the computer and proceeds to send a serial 11-bit message into the printer line. The receiver accepts a serial 11-bit message from the keyboard line and forwards the 8-bit character code into the computer. 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).

45.	Explain the strobe control method of asynchronous data transfer.	8	Section-5	5

Ans.

Strobe Control Method:

- 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. Figure 11-3(a) shows a source-initiated transfer.

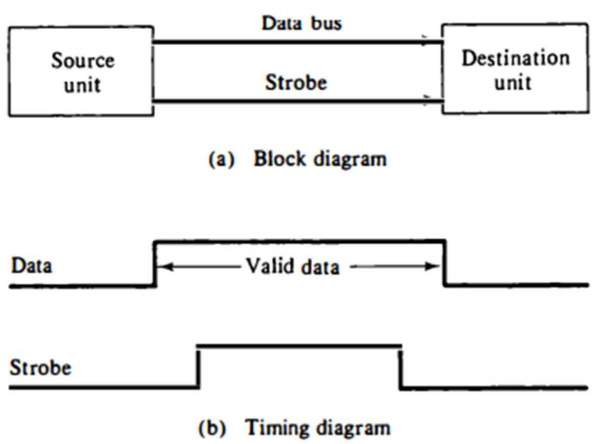


Figure 11-3 Source-initiated strobe for data transfer.

- The data bus carries the binary information from source unit to the 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 in the bus.
- As shown in the timing diagram of Fig. 11-3(b), the source unit first places the data on the data bus. After a brief delay to ensure that 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. Often, 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 a brief period after it disables its strobe pulse. Actually, the source does not have to change the information in the data bus. The fact that 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.
- Figure 11-4 shows a data transfer initiated by the destination unit. In this case 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.

- In many computers the strobe pulse is actually controlled by the clock pulses in the CPU. The CPU is always in control of the buses and informs the external units how to transfer data. For example, the strobe of Fig. 11-3 could be a memory-write control signal from the CPU to a memory unit. The source, being the CPU, places a word on the data bus and informs the memory unit, which is the destination, that this is a write operation. Similarly, the strobe of Fig. 11-4 could be a memory-read control signal from the CPU to a memory unit. The destination, the CPU, initiates the read operation to inform the memory, which is the source, to place a selected word into the data bus.

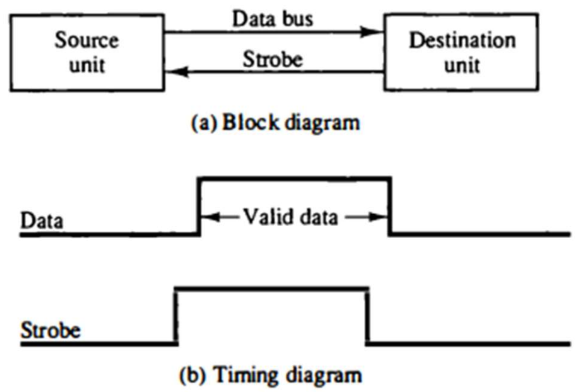


Figure 11-4 Destination-initiated strobe for data transfer.

- The transfer of data between the CPU and an interface unit is similar to the strobe transfer just described. Data transfer between an interface and an I/O device is commonly controlled by a set of handshaking lines.

46.	What are the Modes of Transfers? Explain Programmed I/O.	8	Section-5	5

Ans.

Modes of Transfer:

- Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit. The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit. Data transfer to and from peripherals may be handled in one of three possible modes:
  1. Programmed I/O
  2. Interrupt-initiated I/O
  3. Direct memory access (DMA)

Programmed I/O:

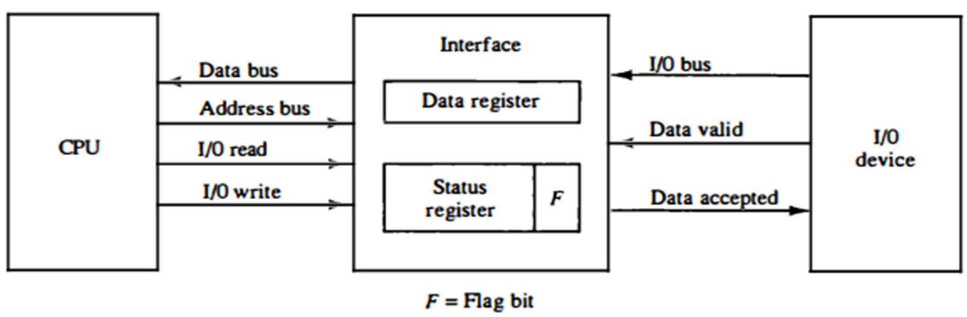
- Programmed I/O operations are the result of I/O instructions written in the computer program. Each data item transfer is initiated by an instruction in the program. Usually, the transfer is to and from a CPU register and peripheral. Other instructions are needed to transfer the data to and from CPU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

Example of Programmed I/O

- In the programmed I/O method, the I/O device does not have direct access to memory. A transfer from an I/O device to memory requires the execution of several instructions by the CPU, including an input instruction to transfer the data from the device to the CPU and a store instruction to transfer the data from the CPU to memory. Other instructions may be needed to verify that the data are available from the device and to count the numbers of words transferred.

- An example of data transfer from an VO device through an interface into the CPU is shown in Fig. 11-10. The device transfers bytes of data one at a time as they are available. When a byte of data is available, the device places it in the VO bus and enables its 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 that we will refer to as an F or "flag" bit. The device can now disable the data valid line, but it will not transfer another byte until the data accepted line is disabled by the interface. This is according to the handshaking procedure established in Fig. 11-5.
- A program is written for the computer to check the flag in the status register to determine if a byte has been placed in the data register by the VO device. This is done by reading the status register into a CPU register and checking the value of the flag bit. If the flag is equal to 1, the CPU reads the data from the data register. The flag bit is then cleared to 0 by either the CPU or the interface, depending on how the interface circuits are designed. Once the flag is cleared, the interface disables the data accepted line and the device can then transfer the next data byte.
- A flowchart of the program that must be written for the CPU is shown in Fig. 11-11. It is assumed that the device is sending a sequence of bytes that must be stored in memory. The transfer of each 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 step 3 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 VO programming task is to transfer a block of words from an VO device and store them in a memory buffer. A program that stores input characters in a memory buffer using the instructions defined in Chap. 6 is listed in Table 6-21.

Figure 11-10 Data transfer from I/O device to CPU.



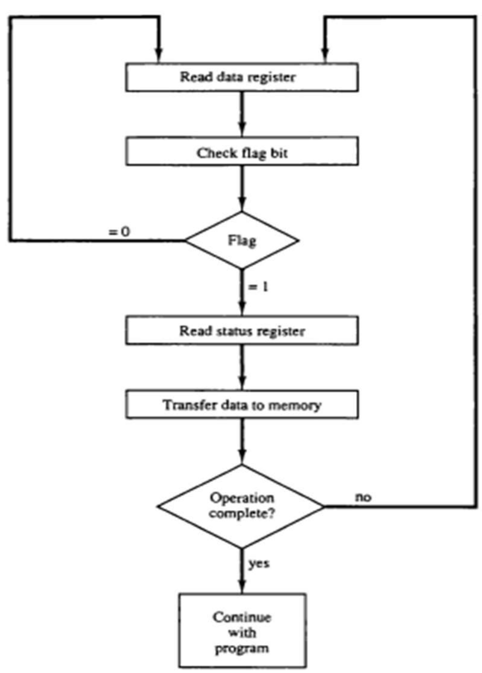


Figure 11-11 Flowchart for CPU program to input data.

- The programmed VO method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously. The difference in information transfer rate between the CPU and the VO device makes this type of transfer inefficient. To see why this is inefficient, consider a typical computer that can execute the two instructions that read the status register and check the flag in 1 µs. Assume that the input device transfers its data at an average rate of 100 bytes per second. This is equivalent to one byte every 10,000 IJ.S. This means that the CPU will check the flag 10,000 times between each transfer. The CPU is wasting time while checking the flag instead of doing some other useful processing task.

47.	Explain Priority Interrupt.	8	Section-5	5

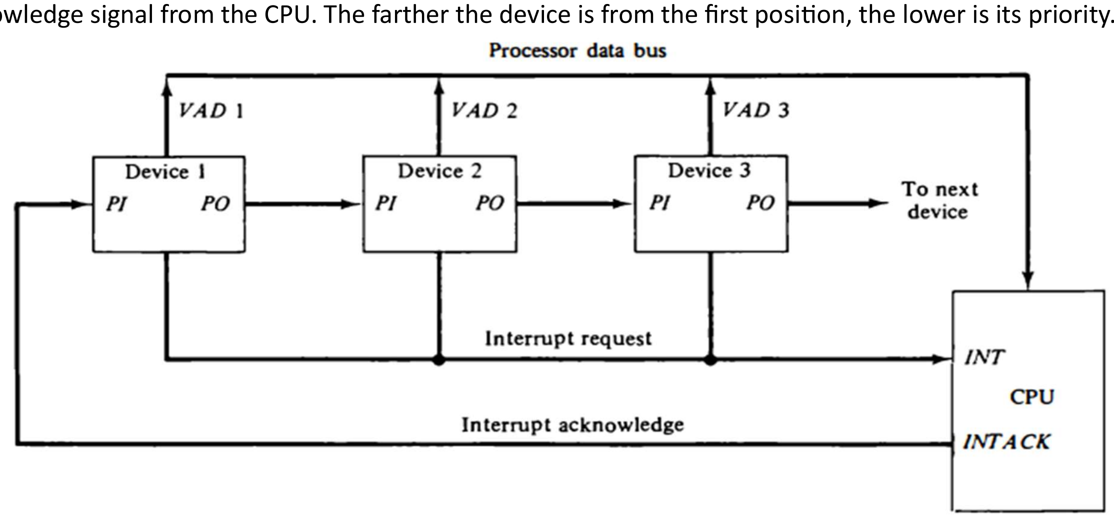
Ans.

<u>Priority Interrupt:</u>

- A priority interrupt is a system that establishes a priority over the various sources to determine which condition is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher-priority interrupt levels are assigned to requests which, if delayed or interrupted, could have serious consequences. Devices with highspeed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority. When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

Daisy-Chaining Priority

- The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in Fig. 11-12. The interrupt request line is common to all devices and forms a wired logic connection. If any device has its interrupt signal in the low-level state, the interrupt line goes to the low-level state and enables the interrupt input in the CPU. When no interrupts are pending, the interrupt line stays in the high-level state and no interrupts are recognized by the CPU. This is equivalent to a negative logic OR operation. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked. A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output. If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output. Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus. The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU. The farther the device is from the first position, the lower is its priority.



- Figure 11-13 shows the internal logic that must be included within each device when connected in the daisy-chaining scheme. The device sets its RF flip-flop when it wants to interrupt the CPU. The output of the RF flip-flop goes through an open-collector inverter, a circuit that provides the wired logic for the common interrupt line. If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF. If PI = 1 and RF=0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO. The device is active when PI = 1 and RF = 1. This condition places a 0 in PO and enables the vector address for the data bus. It is assumed that each device has its own distinct vector address. The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

Parallel Priority Interrupt

- The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a mask register whose purpose is to control the status of each interrupt request. The mask register can be programmed to disable lower-priority interrupts while a higher-priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.
- The priority logic for a system of four interrupt sources is shown in Fig. 11-14. It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions. The magnetic disk, being a high-speed device, is given the highest priority. The printer has the next priority, followed by a character reader and a keyboard. The mask register has the same number of bits as the interrupt register. By means of program instructions, it is possible to set or reset any bit in the mask register. Each interrupt bit and its corresponding mask bit are applied to an AND gate to produce the four inputs to a priority encoder. In this way an interrupt is recognized only if its corresponding mask bit is set to 1 by the program. The priority encoder generates two bits of the vector address, which is transferred to the CPU.
- Another output from the encoder sets an interrupt status flip-flop IST when an interrupt that is not masked occurs. The interrupt enable flip-flop IEN can be set or cleared by the program to provide an overall control over the interrupt system. The outputs of IST ANDed with IEN provide a common interrupt signal for the CPU. The interrupt acknowledge IN'TACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus. We will now explain the priority encoder circuit and then discuss the interaction between the priority interrupt controller and the CPU.

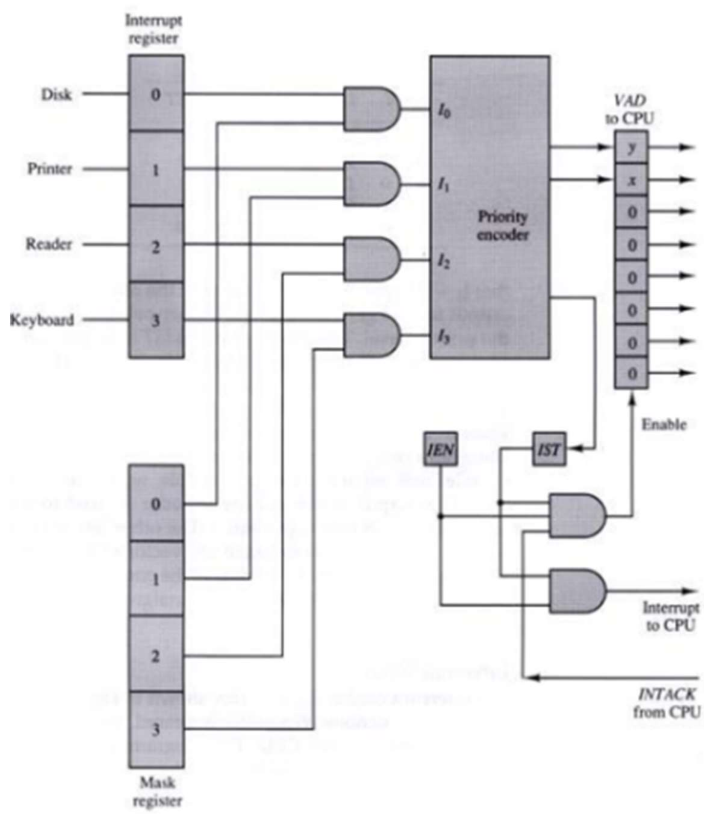


Figure 11-14 Priority interrupt hardware.

48.	What is direct memory transfer? Give an overview and the block diagram of a DMA controller.	8	Section-5	5

49.	What are the Modes of Transfers? Explain interrupt-initiated I/O.	8	Section-5	5

Ans.

Modes of Transfer:

- Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit. The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate source or destination is the memory unit. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes use the CPU as an intermediate path; others transfer the data directly to and from the memory unit. Data transfer to and from peripherals may be handled in one of three possible modes:
    1. Programmed I/O
    2. Interrupt-initiated I/O
    3. Direct memory access (DMA)

Interrupt-initiated I/O

- An alternative to the CPU constantly monitoring the flag is to let the interface inform the computer when it is ready to transfer data. This mode of transfer uses the interrupt facility. While the CPU is running a program, it does not check the flag. However, when the flag is set, the computer is momentarily interrupted from proceeding with the current program and is informed of the fact that the flag has been set. The CPU deviates from what it is doing to take care of the input or output transfer. After the transfer is completed, the computer returns to the previous program to continue what it was doing before the interrupt.
- The CPU responds to the interrupt signal by storing the return address from the program counter into a memory stack and then control branches to a service routine that processes the required VO transfer. The way that the processor chooses the branch address of the service routine varies from one unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, nonvectored interrupt. In a non vectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector. In some computers the interrupt vector is the first address of the VO service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the VO service routine is stored.