UNIT - III: INPUT-OUTPUT ORGANIZATION

Input-Output Organization: Peripheral Devices, Input-Output Interface Modes of Transfer, Priority Interrupt, Direct memory Access, Input–Output Processor (IOP), Serial communication, Introduction to Peripheral Component Interconnect (PCI) bus.

3.1 Peripheral devices:

The input–output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment. Programs and data must be entered into computer memory for processing and results obtained from computations must be recorded or displayed for the user. A computer serves no useful purpose without the ability to receive information from an outside source and to transmit results in a meaningful form.

Input or output devices attached to the computer are also called peripherals. Among the most common peripherals are keyboards, display units, and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes. Peripherals are electromechanical and electromagnetic devices of some complexity.

Monitor and Keyboard

Video monitors are the most commonly used peripherals. They consist of a keyboard as the input device and a display unit as the output device. There are different types of video monitors, but the most popular use a cathode ray tube (CRT). The CRT contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of the tube. The beam can be deflected horizontally and vertically. To produce a pattern on the screen, a grid inside the CRT receives a variable voltage that causes the beam to hit the screen and make it glow at selected spots. Horizontal and vertical signals deflect the beam and make it sweep across the tube, causing the visual pattern to appear on the screen.

In the block mode, the edited text is first stored in a local memory inside the terminal. The text is transferred to the computer as a block of data. Printers provide a permanent record on paper of computer output data or text.

Printer

There are three basic types of character printers: daisywheel, dot matrix, and laser printers. The daisywheel printer contains a wheel with the characters placed along the circumference. To print a character, the wheel rotates to the proper position and an energized magnet then presses the letter against the ribbon. The dot matrix printer contains a set of dots along the printing mechanism. For example, a 5 x 7 dot matrix printer that prints 80 characters per line has seven horizontal lines, each consisting of 5 x 80 = 400 dots. Each dot can be printed or not, depending on the specific characters that are printed on the line. The laser printer uses a rotating photographic drum that is used to imprint the character images. The pattern is then transferred onto paper in the same manner as a copying machine.

Magnetic tape:

Magnetic tapes are used mostly for storing files of data: for example, a company's payroll record Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read–write mechanism. It is one of the cheapest and slowest methods for storage and has the advantage that tapes can be removed when not in use.

Magnetic disks

Magnetic disks have high-speed rotational surfaces coated with magnetic material. Access is achieved by moving a read–write mechanism to a track in the magnetized surface. Disks are used mostly for bulk storage of programs and data.
Other input and output devices encountered in computer systems are digital incremental plotters, optical and magnetic character readers, analog-to-digital converters, and various data acquisition equipment.

The input—output organization of a computer is a function of the size of the computer and the devices connected to it. The difference between a small and a large system is mostly dependent on the amount of hardware the computer has available for communicating with peripheral units and the number of peripherals connected to the system.

**ASCII Alphanumeric Characters**

Input and output devices that communicate with people and the computer are usually involved in the transfer of alphanumeric information to and from the device and the computer. The standard binary code for the alphanumeric characters is ASCII (American Standard Code for Information Interchange). It uses seven bits to code 128 characters as shown in Table 11-1. The seven bits of the code are designated by b1 through b7, with b7 being the most significant bit. The letter A, for example, is represented in ASCII as 1000001 (column 100, row 0001). The ASCII code contains 94 characters that can be printed and 34 nonprinting characters used for various control functions. The printing characters consist of the 26 uppercase letters A through Z, the 26 lowercase letters, the 10 numerals 0 through 9, and 32 special printable characters such as %, *, and $. The 34 control characters are designated in the ASCII table with abbreviated names. They are listed again below the table with their functional names. The control characters are used for routing data and arranging the printed text into a prescribed format. There are three types of control characters: format effectors, information separators, and communication control characters. Format effectors are characters that control the layout of printing. They include the familiar typewriter controls, such as backspace (BS), horizontal tabulation (HT), and carriage return (CR). Information separators are used to separate the data into divisions like paragraphs and pages. They include characters such as record separator (RS) and file separator (FS). The communication control characters are useful during the transmission of text between remote terminals. Examples of communication control characters are STX (start of text) and ETX (end of text), which are used to frame a text message when transmitted through a communication medium. ASCII is a 7-bit code, but most computers manipulate an 8-bit quantity as a single unit called a byte. Therefore, ASCII characters most often are stored one per byte. The extra bit is sometimes used for other purposes, depending on the application. For example, some printers recognize 8-bit ASCII characters with the most significant bit set to 0. Additional 128 8-bit characters with the most significant bit set to 1 are used for other symbols, such as the Greek alphabet or italic type font. When used in data communication, the eighth bit may be employed to indicate the parity of the binary-coded character.
3.2 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 special communication links for interfacing them with the central processing unit. The purpose of the communication link is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

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

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise and synchronize all input and output
transfers. These components are called *interface units* because they interface between the processor bus and the peripheral device. In addition, each device may have its own controller that supervises the operations of the particular mechanism in the peripheral.

**I/O Bus and Interface Modules**

A typical communication link between the processor and several peripherals is shown in Fig. 11-1. The I/O bus consists of data lines, address lines, and control lines. The magnetic disk, printer, and terminal are employed in practically any general-purpose computer. Each peripheral device has associated with it an interface unit. Each interface decodes the address and control received from the I/O bus, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular 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.

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

**I/O versus Memory Bus**

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

1. Use two separate buses, one for memory and the other for I/O.
2. Use one common bus for both memory and I/O but have separate control lines for each.
3. Use one common bus for memory and I/O with common control lines.
In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU). The memory communicates with both the CPU and the IOP through a memory bus. The IOP communicates also with the input and output devices through a separate I/O bus with its own address, data and control lines. The purpose of the IOP is to provide an independent pathway for the transfer of information between external devices and internal memory. The I/O processor is sometimes called a data channel.

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

Many computers use one common bus to transfer information between memory or I/O and the CPU. The distinction between a memory transfer and I/O transfer is made through separate read and write lines. The CPU specifies whether the address on the address lines is for a memory word or for an interface register by enabling one of two possible read or write lines. The I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

**Isolated I/O**

In the isolated I/O configuration, the CPU has distinct input and output instructions, and each of these instructions is associated with the address of an interface register. When the CPU fetches and decodes the operation code of an input or output instruction, it places the address associated with the instruction into the common address lines. At the same time, it enables the I/O read (for input) or I/O write (for output) control line. This informs the external components that are attached to the common bus that the address in the address lines is for an interface register and not for a memory word. On the other hand, when the CPU is fetching an instruction or an operand from memory, it places the memory address on the address lines and enables the memory read or memory write control line. This informs the external components that the address is for a memory word and not for an I/O interface.

The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by interface address assignment since each has its own address space. The other alternative is to use the same address space for both memory and I/O. This is the case in computers that employ only one set of read and write signals and do not distinguish between memory and memory-mapped I/O addresses. This configuration is referred to as memory-mapped I/O. The computer treats an interface register as being part of the memory system. The assigned addresses for interface registers cannot be used for memory words, which reduce the memory address range available.

In a memory-mapped I/O organization there are no specific inputs or output instructions. The CPU can manipulate I/O data residing in interface registers with the same instructions that are used to manipulate memory words. Each interface is organized as a set of registers that respond to read and write requests in the normal address space. Typically, a segment of the total address space is reserved for interface registers, but in general, they can be located at any address as long as there is not also a memory word that responds to the same address. Computers with memory-mapped I/O can use memory-type instructions to access I/O data. It allows the computer to use the same instructions for either input—output transfers or for memory transfers. The advantage is that the load and store instructions used for reading and writing from memory can be used to input and output data from I/O registers. In a typical computer, there are more memory-reference instructions than I/O instructions. With memory-mapped I/O all instructions that refer to memory are also available for I/O.

**Example of I/O Interface**
An example of an I/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.

11-3 Asynchronous Data Transfer

Two units, such as a CPU and an I/O interface, are designed independently of each other. If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be synchronous. In most cases, the internal timing in each unit is independent from the other in that each uses its own private clock for internal registers. In that case, the two units are said to be asynchronous to each other. This approach is widely used in most computer systems.

Asynchronous data transfer between two independent units requires that control signals be transmitted between the communicating units to indicate the time at which data is being transmitted. One way of achieving this is by means of a **strobe** pulse supplied by one of the units to indicate to the other unit when the transfer has to occur. Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as **handshaking**.

**Strobe control**

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit. Figure 11-3(a) shows a source-initiated 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.

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.

**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
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.

![Diagram of data exchange](image)

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.

**Figure 11-6** Destination-initiated transfer using handshaking.

The handshaking scheme provides a high degree of flexibility and reliability because the successful completion of a data transfer relies on active participation by both units. If one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a timeout mechanism, which produces an alarm if the data transfer is not completed 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.

**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 requires 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 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 transmits synchronized with each other.

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.

**Asynchronous Communication Interface**

The block diagram of an asynchronous communication interface is shown in Fig. 11-8. It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by means of a control byte that is loaded into its control register. The transmitter register accepts a data byte from the CPU through the data bus. This byte is transferred to a shift register for serial transmission. The receiver portion receives serial information into another shift register, and when a complete data byte is accumulated, it is transferred to the receiver register. The CPU can select the receiver
register to read the byte through the data bus. The bits in the status register are used for input and output flags and for recording certain errors that may occur during the transmission. The CPU can read the status register to check the status of the flag bits and to determine if any errors have occurred. The chip select and the read and write control lines communicate with the CPU. The chip select (CS) input is used to select the interface through the address bus. The register select (RS) is associated with the read (RD) and write (WR) controls. Two registers are write-only and two are read-only. The register selected is a function of the RS value and the RD and WR status, as listed in the table accompanying the diagram.

![Block diagram of a typical asynchronous communication interface.](image)

<table>
<thead>
<tr>
<th>CS</th>
<th>RS</th>
<th>Operation</th>
<th>Register selected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>×</td>
<td>×</td>
<td>None: data bus in high-impedance</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>WR</td>
<td>Transmitter register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>WR</td>
<td>Control register</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>RD</td>
<td>Receiver register</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>RD</td>
<td>Status register</td>
</tr>
</tbody>
</table>

The operation of the transmitter portion of the interface is as follows. The CPU reads the status register and checks the flag to see if the transmitter register is empty. If it is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full. The first bit in the transmitter shift register is set to 0 to generate a start bit. The character is transferred in parallel from the transmitter register to the shift register and the appropriate numbers of stop bits are appended into the shift register. The transmitter register is then marked empty the character can now be transmitted one bit at a time by shifting the data in the shift register at the specified baud rate. The CPU can transfer another character to the transmitter register after checking the flag in the status register. The interface is said to be double buffered because a new character can be loaded as soon as the previous one starts transmission.

The operation of the receiver portion of the interface is similar. The receive data input is in the 1-state when the line is idle. The receiver control monitors the receive-data line for a 0 signal to detect the occurrence of a start bit. Once a start bit has been
detected, the incoming bits of the character are shifted into the shift register at the prescribed baud rate after receiving the data bits, the interface checks for the parity and stop bits. The character without the start and stop bits is then transferred in parallel from the shift register to the receiver register. The flag in the status register is set to indicate that the receiver register is full. The CPU reads the status register and checks the flag, and if set, it reads the data from the receiver register.

The interface checks for any possible errors during transmission and sets appropriate bits in the status register. The CPU can read the status register at any time to check if any errors have occurred. Three possible errors that the interface checks during transmission are *parity error, framing error, and over-run error*. Parity error occurs if the number of l's in the received data is not the correct parity. A framing error occurs if the right number of stop bits is not detected at the end of the received character. An overrun error occurs if the CPU does not read the character from the receiver register before the next one becomes available in the shift register. Overrun error results in a loss of characters in the received data stream.

**First-In, First-Out Buffer**

A first-in, first-out (FIFO) buffer is a memory unit that stores information in such a manner that the item first in is the item first out. A FIFO buffer comes with separate input and output terminals. The important feature of this buffer is that it can input data and output data at two different rates and the output data are always in the same order in which the data entered the buffer. When placed between two units, the FIFO can accept data from the source unit at one rate of transfer and deliver the data to the destination unit at another rate. If the source unit is slower than the destination unit, the buffer can be filled with data at a slow rate and later emptied at the higher rate. If the source is faster than the destination, the FIFO is useful for those cases where the source data arrive in bursts that fill out the buffer but the time between bursts is long enough for the destination unit to empty some or all the information from the buffer. Thus a FIFO buffer can be useful in some applications when data are transferred asynchronously. It piles up
data as they come in and gives them away in the same order when the data are needed.

The logic diagram of a typical 4 x 4 FIFO buffer is shown in Fig. 11-9. It consists of four 4-bit registers RI, I = 1, 2, 3, 4, and a control register with flip-flops Fi, i = 1, 2, 3, 4, one for each register. The FIFO can store four words of four bits each. The number of bits per word can be increased by increasing the number of bits in each register and the number of words can be increased by increasing the number of registers.

A flip-flop Fi in the control register that is set to 1 indicates that a 4-bit data word is stored in the corresponding register RI. A 0 in Fi indicates that the corresponding register does not contain valid data. The control register directs the movement of data through the registers. Whenever the Fi bit of the control register is set (Fi = 1) and the Fi+1 bit is reset (Fi+1 = 0), a clock is generated causing register R(I + 1) to accept the data from register RI. The same clock transition sets Fi+1 to 1 and resets Fi to 0. This causes the control flag to move one position to the right together with the data. Data in the registers move down the FIFO toward the output as long as there are empty locations ahead of it. This ripple-through operation stops when the data reach a register RI with the next flip-flop Fi+1, being set to 1, or at the last register R4. An overall master clear is used to initialize all control register flip-flops to 0.

Data are inserted into the buffer provided that the input ready signal is enabled. This occurs when the first control flip-flop F1 is reset, indicating that register R1 is empty. Data are loaded from the input lines by enabling the clock in R1 through the insert control line. The same clock sets F1, which disables the input ready control, indicating that the FIFO is now busy and unable to accept more data. The ripple-through process begins provided that R2 is empty. The data in R1 are transferred into R2 and F1 is cleared. This enables the input ready line, indicating that the inputs are now available for another data word. If the FIFO is full, F1 remains set and the input ready line stays in the 0 state. Note that the two control lines input ready and insert constitute a destination-initiated pair of handshake lines.

The data falling through the registers stack up at the output end. The output ready control line is enabled when the last control flip-flop F4 is set, indicating that there are
valid data in the output register R4. The output data from R4 are accepted by a destination unit, which then enables the delete control signal. This resets for causing output ready to disable, indicating that the data on the output are no longer valid. Only after the delete signal goes back to 0 can the data from R3 move into R4. If the FIFO is empty, there will be no data in R3 and F4 will remain in the reset state. Note that the two control lines output ready and delete constitute a source-initiated pair of handshake lines.

4. 3 Modes of Transfer
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)

1. 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 I/O 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 I/O 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.

![Data transfer from I/O device to CPU](image)

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

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 I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer. A program that stores input characters in a memory buffer using the instructions

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

2. Interrupt-Initiated I/O

In the programmed I/O method, the CPU stays in a program loop until the I/O unit indicates that it is ready for data transfer. This is a time-consuming process since it keeps the processor busy needlessly. It can be avoided by using an interrupt facility and special commands to inform the interface to issue an interrupt request signal when the data are available from the device. In the meantime the CPU can proceed to execute another program. The interface meanwhile keeps monitoring the device. When the interface determines that the device is ready for data transfer, it generates an interrupt request to the computer. Upon detecting the external interrupt signal, the CPU momentarily stops the task it is processing, branches to a service program to process the I/O transfer, and then returns to the task it was originally performing.

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 I/O transfer. The way that the processor chooses the branch address of the service routine varies from one unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, non-vectored interrupt. In a non-vectored interrupt, the branch address is assigned to a fixed location in memory. In a vectored interrupt, the source that interrupts supplies the branch information to the computer. This information is called the interrupt vector. In some computers the interrupt vector is the first address of the I/O service routine. In other computers the interrupt vector is an address that points to a location in memory where the beginning address of the I/O service routine is stored.

**Software Considerations**

The previous discussion was concerned with the basic hardware needed to interface I/O devices to a computer system. A computer must also have software routines for controlling peripherals and for transfer of data between the processor and peripherals. I/O routines must issue control commands to activate the peripheral and to check the device status to determine when it is ready for data transfer. Once ready, information is transferred item by item until all the data are transferred.

Software control of input—output equipment is a complex undertaking. For this reason I/O routines for standard peripherals are provided by the manufacturer as part of the computer system.

### 4.4 Priority Interrupt

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 high-speed transfers such as magnetic disks are given high priority, and slow devices such as keyboards receive low priority. When two devices interrupt the computer at the same time, the computer services the device, with the higher priority first.

Establishing the priority of simultaneous interrupts can be done by software or hardware. A polling procedure is used to identify the highest-priority source by software means. In this method there is one common branch address for all interrupts. The program that takes care of interrupts begins at the branch address and polls the interrupt sources in sequence. The order in which they are tested determines the priority of each interrupt. The highest-priority source is tested first, and if its interrupt signal is on, control branches to a service routine for this source. Otherwise, the next-lower-priority source is tested, and so on. Thus the initial service routine for all interrupts consists of a program that tests the interrupt sources in sequence and branches to one of many possible service routines. The particular service routine reached belongs to the highest-priority device among all devices that interrupted the computer. The disadvantage of the software method is that if there are many interrupts, the time required to poll them can exceed the time available to service the I/O device. In this situation a hardware priority-interrupt unit can be used to speed up the operation.
A hardware priority-interrupt unit functions as an overall manager in an interrupt system environment. It accepts interrupt requests from many sources, determines which of the incoming requests has the highest priority, and issues an interrupt request to the computer based on this determination. To speed up the operation, each interrupt source has its own interrupt vector to access its own service routine directly. Thus no polling is required because all the decisions are established by the hardware priority-interrupt unit. The hardware priority function can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the daisy-chaining method.

**Daisy-Chaining Priority**

The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain. This method of connection between three devices and the CPU is shown in 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. The CPU responds to an interrupt request by enabling the interrupt acknowledge line. This signal is received by device 1 at its PI (priority in) input. The acknowledge signal passes on to the next device through the PO (priority out) output only if device 1 is not requesting an interrupt. If device 1 has a pending interrupt, it blocks the acknowledge signal from the next device by placing a 0 in the PO output. It then proceeds to insert its own interrupt vector address (VAD) into the data bus for the CPU to use during the interrupt cycle.

Figure 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 RE. 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.

![Figure 11-12 Daisy-chain priority interrupt.](image-url)
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 INTACK signal from the CPU enables the bus buffers in the output register and a vector address VAD is placed into the data bus.

**Priority Encoder**

The priority encoder is a circuit that implements the priority function. The logic of the priority encoder is such that if two or more inputs arrive at the same time, the input having the highest priority will take precedence. The truth table of a four-input priority encoder is given in Table 11-2. The x's in the table designate don't-care conditions.
The output of the priority encoder is used to form part of the vector address for each interrupt source. The other bits of the vector address can be assigned any value. For example, the vector address can be formed by appending six zeros to the \( x \) and \( y \) outputs of the encoder. With this choice the interrupt vectors for the four I/O devices are assigned binary numbers 0, 1, 2, and 3.

### Interrupt Cycle

The interrupt enable flip-flop IEN shown in Fig. 11-14 can be set or cleared by program instructions. When IEN is cleared, the interrupt request coming from IST is neglected by the CPU. The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted. An instruction to set IEN indicates that the interrupt facility will be used while the current program is running. Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor.

During the interrupt cycle the CPU performs the following sequence of micro-operations:

- \( \text{SP} \leftarrow \text{SP} - 1 \) Decrement stack pointer
- \( \text{M}[\text{SP}] \leftarrow \text{PC} \) Push PC into stack
- \( \text{IN TACK} \leftarrow 1 \) Enable interrupt acknowledge
- \( \text{PC} \leftarrow \text{VAD} \) Transfer vector address to PC
- \( \text{IEN} \leftarrow 0 \) Disable further interrupts

Go to fetch next instruction

### Software Routines

A priority interrupt system is a combination of hardware and software techniques. So far we have discussed the hardware aspects of a priority interrupt system. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers.

Figure 11-15 shows the programs that must reside in memory for handling the interrupt system. Each device has its own service program that can be reached through a jump (IMP) instruction stored at the assigned vector address. The symbolic name of each routine represents the starting address of the service program. The stack shown in the diagram is used for storing the return address after each interrupt.
Initial and Final Operations

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system. The initial sequence of each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

- Clear lower-level mask register bits.
- Clear interrupt status bit IST.
- Save contents of processor registers.
- Set interrupt enable bit IEN.
- Proceed with service routine.

The final sequence in each interrupt service routine must have instructions to control the interrupt hardware in the following manner:

- Clear interrupt enable bit IEN.
- Restore contents of processor registers.
- Clear the bit in the interrupt register belonging to the source that has been serviced.
- Set lower-level priority bits in the mask register.
- Restore return address into PC and set IEN.

4.5 Direct Memory Access (DMA)

5.14 Direct Memory Access (DMA)

- Block of data transfer from high speed devices, Drum, Disk, Tape
- DMA controller - Interface which allows I/O transfer directly between Memory and Device, freeing CPU for other tasks
- CPU initializes DMA Controller by sending memory address and the block size (number of words)
CPU bus signals for DMA transfer

Block diagram of DMA controller

DMA I/O OPERATION

Starting an I/O
- CPU executes instruction to
  Load Memory Address Register
  Load Word Counter
  Load Function(Read or Write) to be performed
  Issue a GO command

Upon receiving a GO Command DMA performs I/O operation as follows independently from CPU

Input
1. Input Device <- R (Read control signal)
2. Buffer(DMA Controller) <- Input Byte; and
   assembles the byte into a word until word is full
3. M <- memory address, W(Write control signal)
4. Address Reg <- Address Reg +1; WC(Word Counter) <- WC - 1
5. If WC = 0, then Interrupt to acknowledge done, else go to [1]

Output
1. M <- M Address, R
   M Address R <- M Address R + 1, WC <- WC - 1
2. Disassemble the word
3. Buffer <- One byte; Output Device <- W, for all disassembled bytes
4. If WC = 0, then Interrupt to acknowledge done, else go to [1]
CYCLE STEALING

While DMA I/O takes place, CPU is also executing instructions

DMA Controller and CPU both access Memory -> Memory Access Conflict

Memory Bus Controller

- Coordinating the activities of all devices requesting memory access
- Priority System

Memory accesses by CPU and DMA Controller are interwoven, with the top priority given to DMA Controller
-> Cycle Stealing

Cycle Steal

- CPU is usually much faster than I/O(DMA), thus CPU uses the most of the memory cycles
- DMA Controller steals the memory cycles from CPU
- For those stolen cycles, CPU remains idle
- For those slow CPU, DMA Controller may steal most of the memory cycles which may cause CPU remain idle long time
◆ DMA Transfer (I/O to Memory)

1) I/O Device sends a DMA request
2) DMAC activates the BR line
3) CPU responds with BG line
4) DMAC sends a DMA acknowledge to the I/O device
5) I/O device puts a word in the data bus (for memory write)
6) DMAC writes a data to the address specified by Address register
7) Decrement Word count register
8) Word count register = 0, EOT interrupt CPU
9) Word count register ≠ 0, DMAC checks the DMA request from I/O device
4.6 Input-Output Processor (IOP)

Instead of having each interface communicate with CPU, a computer may incorporate one or more external processors and assign them the task of communicating directly with all I/O devices. An input output processor (IOP) may be classified as a processor with direct memory access capability that communicates with I/O devices. Each IOP takes care of input and output tasks, relieving the CPU from the housekeeping chores involved in I/O transfers.

The IOP is similar to a CPU except that it is designed to handle the details of I/O processing. Unlike the DMA controller, the IOP can fetch and execute its own instructions. IOP instructions are specifically designed to facilitate I/O transfers. In addition, the IOP can perform other processing tasks, such as arithmetic, logic branching, and code translation.

The block diagram of a computer with two processors is shown in Fig. below. The CPU is usually assigned the task of initiating the I/O program. From then on the IOP operates independent of the CPU and continues to transfer data from external devices and memory.

The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources. For example, it may be necessary to take four bytes from an input device and pack them into one 32-bit word before the transfer to memory. After the input data are assembled into a memory word, they are transferred from IOP directly into memory by “stealing” one memory cycle from the CPU. Similarly, an output word transferred from memory to the IOP is directed from the IOP to the output device at the device rate and bit capacity.

In most computer systems, the CPU is the master while the IOP is a slave processor. The CPU is assigned the task of initiating all operations, but I/O instructions are executed in the IOP. CPU instructions provide operations to start an I/O transfer and also to test I/O status conditions needed for making decisions on various I/O activities. The IOP, in turn, typically asks for CPU attention by means of an interrupt. It also responds to CPU requests by placing a status word in a prescribed location in memory to be examined later by a CPU program. When an I/O operation is desired, the CPU informs the IOP where to find the I/O program and then leaves the transfer details to the IOP.
Instructions that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU. Commands are prepared by experienced programmers and are stored in memory. The command words constitute the program for the IOP. The CPU informs the IOP where to find the commands in memory when it is time to execute the I/O program.

**CPU—IOP Communication**

The sequence of operations may be carried out as shown in the flowchart of Fig. 11-20.

The CPU sends an instruction to test the IOP path. The IOP responds by inserting a status word in memory for the CPU to check. The bits of the status word indicate the condition of the IOP and I/O device, such as IOP overload condition, device busy with another transfer, or device ready for I/O transfer. The CPU refers to the status word in memory to decide what to do next. If all is in order, the CPU sends the instruction to start I/O transfer. The memory address received with this instruction tells the IOP where to find its program.

The CPU can now continue with another program while the IOP is busy with the I/O program. Both programs refer to memory by means of DMA transfer. When the IOP terminates the execution of its program, it sends an interrupt request to the CPU. The CPU responds to the interrupt by issuing an instruction to read the status from the IOP. The IOP responds by placing the contents of its status report into a specified memory location. The status word indicates whether the transfer has been completed or if any errors occurred during the transfer. From inspection of the bits in the status word, the CPU determines if the I/O operation was completed satisfactorily without errors.

**IBM 370 I/O Channel**

The I/O processor in the IBM 370 computer is called a channel. A typical computer system configuration includes a number of channels with each channel attached to one or
more I/O devices. There are three types of channels: multiplexer, selector, and block-multiplexer. The multiplexer channel can be connected to a number of slow- and medium-speed devices and is capable of operating with a number of I/O devices simultaneously. The selector channel is designed to handle one I/O operation at a time and is normally used to control one high-speed device. The block-multiplexer channel combines the features of both the multiplexer and selector channels. It provides a connection to a number of high-speed devices, but all I/O transfers are conducted with an entire block of data as compared to a multiplexer channel, which can transfer only one byte at a time.

The CPU communicates directly with the channels through dedicated control lines and indirectly through reserved storage areas in memory. Figure 11-21 shows the word formats associated with the channel operation. The I/O instruction format has three fields: operation code, channel address, and device address.

![I/O instruction format](image)

The addressed channel responds to each of the I/O instructions and executes it. It also sets one of four condition codes in a processor register called PSW (processor status word). The CPU can check the condition code in the PSW to determine the result of the I/O operation. The meaning of the four condition codes is different for each I/O instruction. But, in general, they specify whether the channel or the device is busy, whether or not it is operational, whether interruptions are pending, if the I/O operation had started successfully, and whether a status word was stored in memory by the channel.

The format of the channel status word is shown in Fig. 11-21(b). It is always stored in location 64 in memory. The key field is a protection mechanism used to prevent unauthorized access by one user to information that belongs to another user or to the operating system. The address field in the status word gives the address of the last command word used by the channel. The count field gives the residual count when the transfer was terminated. The count field will show zero if the transfer was completed successfully. The status field identifies the conditions in the device and the channel and any errors that occurred during the transfer.

The format of the channel command word is shown in Fig. 11-21(c). The data address field specifies the first address of a memory buffer and the count field gives the number of bytes involved in the transfer. The command field specifies an I/O operation and the flag bits provide additional information for the channel. The command field corresponds to an operation code that specifies one of six basic types of I/O operations:

- **Write.** Transfer data from memory to I/O device.
- **Read.** Transfer data from I/O device to memory.
- **Read backwards.** Read magnetic tape with tape moving backward.
- **Control.** Used to initiate an operation not involving transfer of data, such as rewinding of tape or positioning a disk-access mechanism.
- **Sense.** Informs the channel to transfer its channel status word to memory location.
- **Transfer in channel.** Used instead of a jump instruction. Here the data address field specifies the address of the next command word to be executed by the channel.

A memory map showing all pertinent information for I/O processing is illustrated in Fig. 11-22. The operation begins when the CPU program encounters a start I/O instruction. The TOP then goes to memory location 72 to obtain a channel address word. This word contains the starting address of the I/O channel program. The channel then proceeds to execute the program specified by the channel command words. The channel constructs a status word during the transfer and stores it in location 64. Upon interruption, the CPU can refer to memory location 64 for the status word.

**Intel 8089 IOP**
The Intel 8089 I/O processor is contained in a 40-pin integrated circuit package. Within the 8089 are two independent units called channels. The 8089 IOP has 50 basic instructions that can operate on individual bits, on bytes, or 16-bit words. The IOP can execute programs in a manner similar to a CPU except that the instruction set is specifically chosen to provide efficient input—output processing. The instruction set includes general data transfer instructions, basic arithmetic and logic operations, conditional and unconditional branch operations, and subroutine call and return capabilities. The set also includes special instructions to initiate DMA transfers and issue an interrupt request to the CPU.
A microcomputer system using the Intel 8086/8089 pair of integrated circuits is shown in Fig. 11-23. The 8086 functions as the CPU and the 8089 as the IOP. The two units share a common memory through a bus controller connected to a system bus, which is called a "multibus" by Intel. The IOP uses a local bus to communicate with various interface units connected to I/O devices. The CPU communicates with the IOP by enabling the channel attention line. The select line is used by the CPU to select one of two channels in the 8089. The IOP gets the attention of the CPU by sending an interrupt request.

The CPU and IOP communicate with each other by writing messages for one another in system memory. The CPU prepares the message area and signals the IOP by enabling the channel attention line. The IOP reads the message, performs the required I/O functions, and executes the appropriate channel program. When the channel has completed its program, it issues an interrupt request to the CPU.

The communication scheme consists of program sections called "blocks," which are stored in memory as shown in Fig. 11-24. Each block contains control and parameter information as well as an address pointer to its successor block. The address of the control block is passed to each IOP channel during initialization. The busy flag indicates whether the IOP is busy or ready to perform a new I/O operation. The CCW (channel command word) is specified by the CPU to indicate the type of operation required from the IOP. The CPU and IOP work together through the control and parameter blocks.

The CPU then fills in the information in the parameter block and writes a "start operation" command in the CCW. After the communication blocks have been set up, the CPU enables the channel attention signal to inform the IOP to start its I/O operation. The CPU then continues with another program. The IOP responds to the channel attention signal by placing the address of the control block into its program counter. The IOP refers
to the control block and sets the busy flag. It then checks the operation in the CCW. The PB (parameter block) address and TB (task block) address are then transferred into internal IOP registers. The IOP starts executing the program in the task block using the information in the parameter block. The entries in the parameter block depend on the I/O device. The parameters listed in Fig. 11-24 are suitable for data transfer to or from a magnetic disk. The memory address specifies the beginning address of a memory buffer. The byte count gives the number of bytes to be transferred. The device address specifies the particular I/O device to be used. The track and sector numbers locate the data on the disk. When the I/O operation is completed, the IOP stores its status bits in the status word location of the parameter block and interrupts the CPU. The CPU can refer to the status word to check if the transfer has been completed satisfactorily.

4.7 Serial communication

A data communication processor is an I/O processor that distributes and collects data from many remote terminals connected through telephone and other communication lines. It is a specialized I/O processor designed to communicate directly with data communication networks. A communication network may consist of any of a wide variety of devices, such as printers, interactive display devices, digital sensors, or a remote computing facility.

The most striking difference between an I/O processor and a data communication processor is in the way the processor communicates with the I/O devices. An I/O processor communicates with the peripherals through a common I/O bus that is comprised of many data and control lines. A data communication processor communicates with each terminal through a single pair of wires. Both data and control information are transferred in a serial fashion with the result that the transfer rate is much slower. The task of the data communication processor is to transmit and collect digital information to and from each terminal, determine if the information is data or control and respond to all requests according to predetermined established procedures. The processor, obviously, must also communicate with the CPU and memory in the same manner as any I/O processor.

The way that remote terminals are connected to a data communication processor is via telephone lines or other public or private communication facilities. Since telephone lines were originally designed for voice communication and computers communicate in terms of digital signals, some form of conversion must be used. The converters are called data sets, acoustic couplers, or modems (from "modulator-demodulator"). A modem converts digital signals into audio tones to be transmitted over telephone lines and also converts audio tones from the line to digital signals for machine use. Various modulation schemes as well as different grades of communication media and transmission speeds are used. A communication line may be connected to a synchronous or asynchronous interface, depending on the transmission method of the remote terminal. An asynchronous interface receives serial data with start and stop bits in each character.

Synchronous transmission does not use start-stop bits to frame characters and therefore makes more efficient use of the communication link. High-speed devices use synchronous transmission to realize this efficiency. The modems used in synchronous transmission have internal clocks that are set to the frequency that bits are being transmitted in the communication line. For proper operation, it is required that the clocks in the transmitter and receiver modems remain synchronized at all times.

Contrary to asynchronous transmission, where each character can be sent separately with its own start and stop bits, synchronous transmission must send a continuous message in order to maintain synchronism. The message consists of a group of bits transmitted sequentially as a block of data. The entire block is transmitted with special control characters at the beginning and end of the block. The control characters at the
beginning of the block supply the information needed to separate the incoming bits into individual characters.

One of the functions of the data communication processor is to check for transmission errors. An error can be detected by checking the parity in character received. Another procedure used in asynchronous terminals involving a human operator is to echo the character.

In synchronous transmission, where an entire block of characters is transmitted, each character has a parity bit for the receiver to check. After the entire block is sent, the transmitter sends one more character that constitutes a parity over the length of the message. This character is called a longitudinal redundancy check (LRC) and is the accumulation of the exclusive-OR of all transmitted characters.

If the receiver finds an error in the transmitted block, it informs the sender to retransmit the same block once again. Another method used for checking errors in transmission is the cyclic redundancy check (CRC). This is a polynomial code obtained from the message bits by passing them through a feedback shift register containing a number of exclusive-OR gates. This type of code is suitable for detecting burst errors occurring in the communication channel.

Data can be transmitted between two points in three different modes: simplex, half-duplex, or full-duplex. A simplex line carries information in one direction only. A half-duplex transmission system is one that is capable of transmitting in both directions but data can be transmitted in only one direction at a time. A full-duplex transmission can send and receive data in both directions simultaneously.

The communication lines, modems, and other equipment used in the transmission of information between two or more stations is called a data link. The orderly transfer of information in a data link is accomplished by means of a protocol. A data link control protocol is a set of rules that are followed by interconnecting computers and terminals to ensure the orderly transfer of information. The purpose of a data link protocol is to establish and terminate a connection between two stations, to identify the sender and receiver, to ensure that all messages are passed correctly without errors, and to handle all control functions involved in a sequence of data transfers. Protocols are divided into two major categories according to the message-framing technique used. These are character-oriented protocol and bit-oriented protocol.

**Character-Oriented Protocol**

The SYN character serves as synchronizing agent between the transmitter and receiver. Once a SYN character is detected, the receiver has framed a character. From here on the receiver counts every eight bits and accepts them as a single character. Moreover, when the transmitter is idle and does not have any message characters to send, it sends a continuous string of SYN characters. The receiver recognizes these characters as a condition for synchronizing the line and goes into a synchronous idle state.

![Figure 11-25 Typical message format for character-oriented protocol.](image)

Messages are transmitted through the data link with an established format consisting of a header field, a text field, and an error-checking field. A typical message format for a character-oriented protocol is shown in Fig. 11-25. The two SYN characters assure proper synchronization at the start of the message. Following the SYN characters is the header, which starts with an SOH (start of heading) character. The header consists of address and control information. The STX character terminates the header and signifies the beginning of the text transmission. The text portion of the message is variable in
Data Transp

Data Transp

data transparency

character-oriented protocol

binary information

ASCII text

character-oriented protocols

DLE (data link escape)

double characters

DLE SOH

DLE ETX

IDLE bit pattern

2DLE 00010000

receiver

DLE characters

8-bit pattern

data transparency

Digital Equipment Corporation

byte count field

frame boundaries

special 8-bit number

flag

zoo insertion

zero insertion

5 consecutives

transmission

error-free

bit-oriented protocol

frame format

address and control sequence

information field

CRC (cyclic redundancy check)

adding flag

beginning flag

interfering flag

15 consecutives

control messages

minimum 32 bits

transmit long messages

error-free

zero insertion

serial bit stream

bit-oriented protocol

transmit any length
As a consequence, no pattern of 01111110 is ever transmitted between the beginning and ending flags.

The address field, which is used by the primary station to designate the secondary station address. When a secondary station transmits a frame, the address tells the primary station which secondary station originated the frame.

The control field comes in three different formats, as shown in Fig. 11-27. The information transfer format is used for ordinary data transmission. Each frame transmitted in this format contains send and receive counts. A station that transmits sequenced frames counts and numbers each frame. This count is given by the send count $N_s$. A station receiving sequenced frames counts each error-free frame that it receives. This count is given by the receive count $N_r$. The $N_s$ count advances when a frame is checked and found to be without errors. The receiver confirms accepted numbered information frames by returning its $N_r$ count to the transmitting station.

The P/F bit is used by the primary station to poll a secondary station to request that it initiate transmission. It is used by the secondary station to indicate the final transmitted frame. Thus the P/F field is called P (poll) when the primary station is transmitting but is designated as F (final) when a secondary station is transmitting. Each frame sent to the secondary station from the primary station has a P bit set to 0. When the primary station is finished and ready for the secondary station to respond, the P bit is set to 1. The secondary station then responds with a number of frames in which the F bit is set to 0. When the secondary station sends the last frame, it sets the F bit to 1. Therefore, the P/F bit is used to determine when data transmission from a station is finished.

The supervisory format of the control field is recognized from the first two bits being 1 and 0. The next two bits indicate the type of command. This follows by a P/F bit and a receive sequence frame count. The frames of the supervisory format do not carry an information field. They are used to assist in the transfer of information in that they confirm the acceptance of preceding frames carrying information, convey ready- or busy conditions, and report frame numbering errors.

The unnumbered format is recognized from the first two bits being 11. The five code bits available in this format can specify up to 32 commands and responses. The primary station uses the control field to specify a command for a secondary station. The secondary station uses the control field to transmit a response to the primary station. Unnumbered-format frames are employed for initialization of link functions, reporting procedural errors, placing stations in a disconnected mode, and other data link control operations.

4.8 Introduction to peripheral component Interconnect (PCI) 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 in 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 length one.
- 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 burst of data is called a “transaction”.

**Fig:** Use of a PCI bus in a Computer system

![Diagram of PCI bus in a computer system](image)
Data Transfer Signals on PCI Bus:

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

Individual word transfers are called “phases”.

**Fig: Read operation on 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 drivers from AD lines.
- The selected target enable its drivers on AD lines and fetches the requested data to be placed on the bus.
- 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 the transaction.
• During clock cycle 3, the initiator asserts IRDY #, to indicate that it is ready to receive data.
• 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 to 6.
• The indicator uses FRAME # to indicate the duration of the burst, since it read 4 words, the initiator negates FRAME # during clock cycle 5.
• After sending the 4th word, the target disconnects its drivers and negates DEVSEL # during clockcycle 7.

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

- It indicates the pause in the middle of the transaction.
- 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 cycle 8.

**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.
- The initialization s/w read these ROM’s whenever the S/M is powered up or reset
- 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 to AD31).
- 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 to 0.
- 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.
Two types of Serial Communication

- Asynchronous Serial Communication
  - Interacts with devices outside of the computer
    - Ex: modem connecting to another computer
  - Transmit individual bytes instead of large blocks
  - Do not share a common clock.

- Synchronous Serial transmission
  - Transmits block of data in frames.
    - Frames are had head in front of the data and a tail at the end of the data.
  - The head and tail contain information that allows the two computers to synchronize their clocks.