

5.1 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. 5-1. 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 de-codes 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.

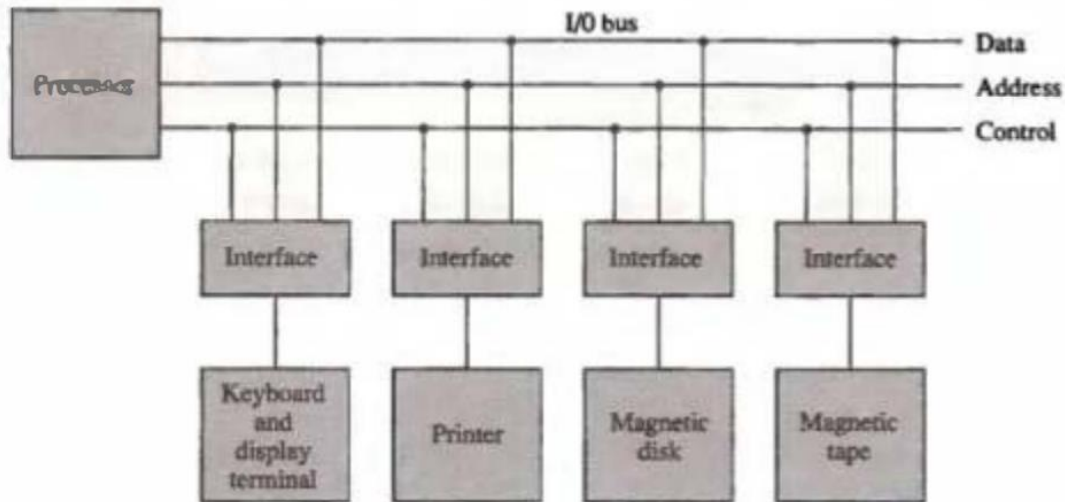


Fig 5.1.1: Connection between I/O bus to input-output device

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.

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.

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 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 reduces the memory address range available.

In a memory-mapped I/O organization there are no specific input 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.

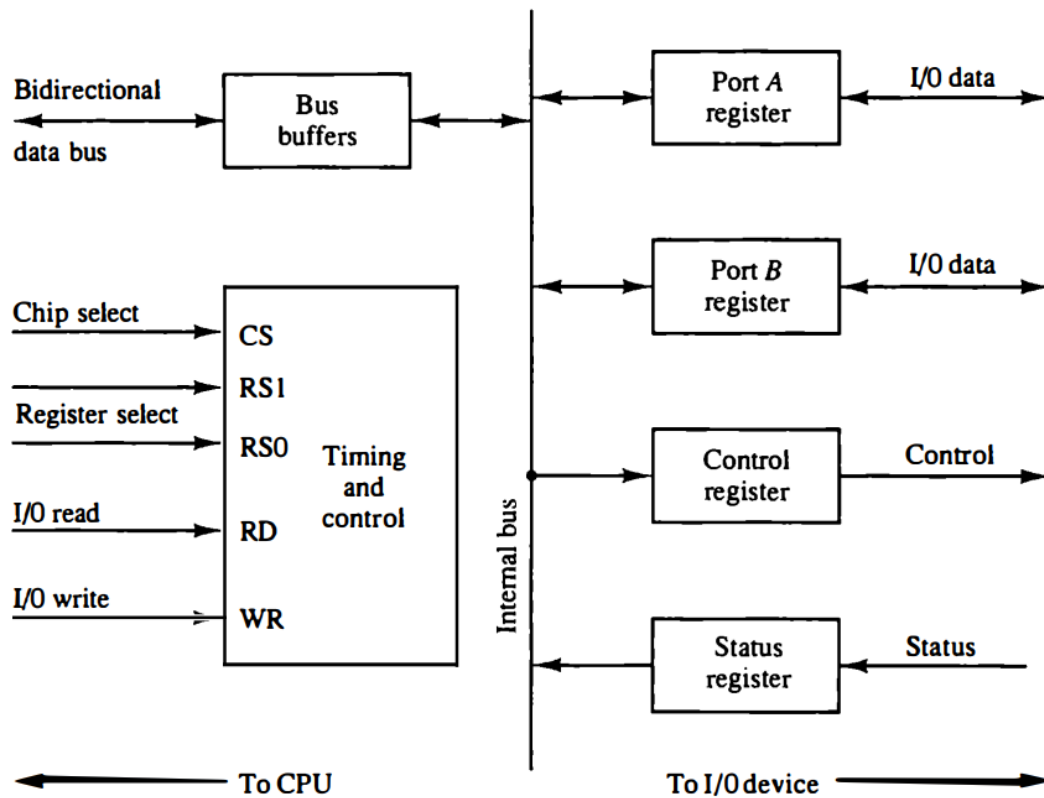
Example of I/O Interface

An example of an I/O interface unit is shown in block diagram form in Fig. 5.2. 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 I/O read and write are two control lines that specify an input or output, respectively. The four registers communicate directly with the I/O device attached to the interface.

The I/O data to and from the device can be transferred into either port A or port B. The interface may operate with an 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 I/O device by sending a word to the appropriate interface register. In a system like this, the function code in the I/O bus is not needed because control is sent to the control register, status information is received from the status register, and data are transferred to and from ports A and B registers. Thus the transfer of data, control, and status information is always 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.

The control register receives control information from the CPU. By loading appropriate bits into the control register, the interface and the I/O device attached to it 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 port-A has received a new data item from the I/O 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 registers select inputs RSI 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.



CS	RS1	RS0	Register selected
0	x	x	None: data bus in high-impedance
1	0	0	Port A register
1	0	1	Port B register
1	1	0	Control register
1	1	1	Status register

Figure 5.1.2 Example of I/O interface unit.

2 Asynchronous Data Transfer

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

The strobe pulse method and the handshaking method of asynchronous data transfer are not restricted to I/O transfers. In fact, they are used extensively on numerous occasions requiring the transfer of data between two independent units. In the general case we consider the transmitting unit as the source and the receiving unit as the destination. For example, the CPU is the source unit during an output or a write transfer and it is the destination unit during an input or a read transfer. It is customary to specify the asynchronous transfer between two independent units by means of a **timing diagram** that shows the timing relationship that must exist between the control signals and the data in the buses. The sequence of control during an asynchronous transfer depends on whether the transfer is initiated by the source or by the destination unit.

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 3(a) shows a source-initiated transfer.

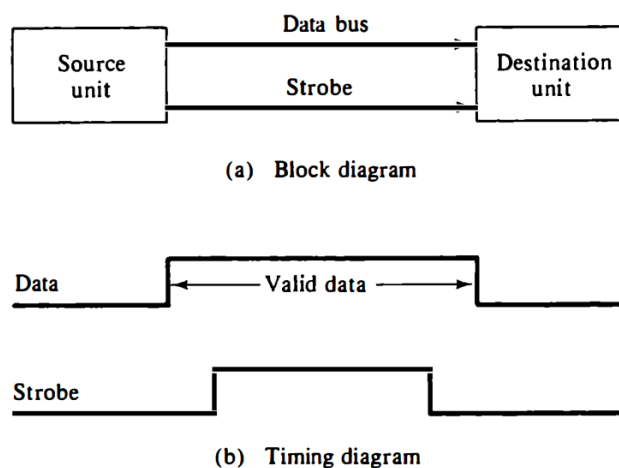


Figure 3 Source initiated strobe for data transfer.

The data bus carries the binary information from source unit to the destination unit. 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(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 remains 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 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.

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 VO device is commonly controlled by a set of handshaking lines.

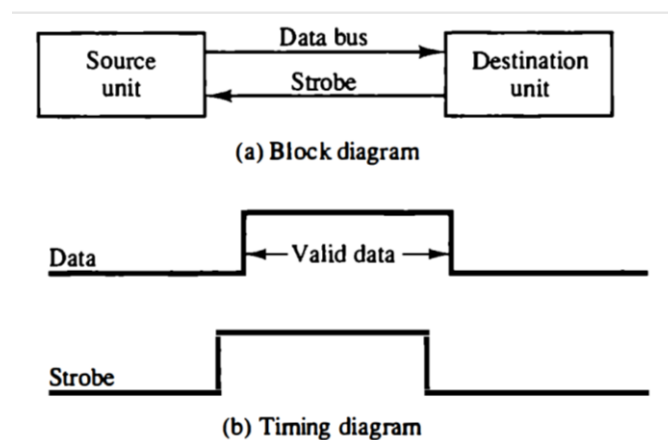
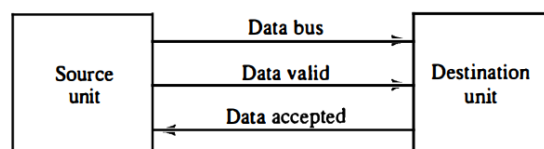


Figure 4 Destination initiated strobe for data transfer.

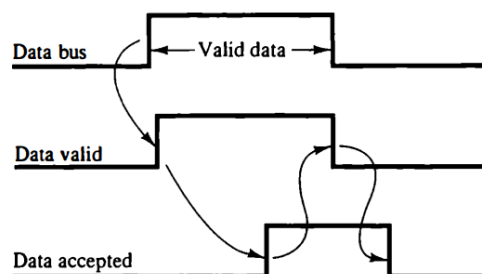
Handshaking

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.

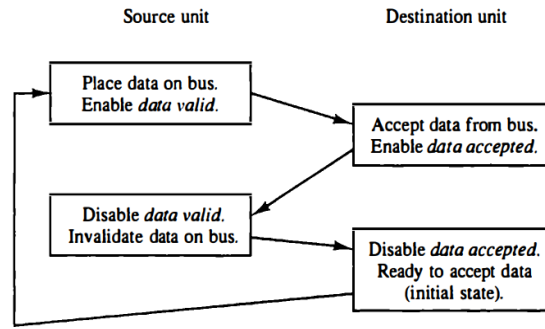
Figure 5 shows the data transfer procedure when initiated by the source. The two handshaking lines are data valid, which is generated by the source unit, and data accepted, generated by the destination unit. The timing diagram shows the exchange of signals between the two units. The sequence of events 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



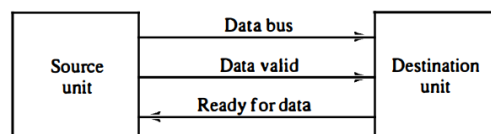
(b) Timing diagram



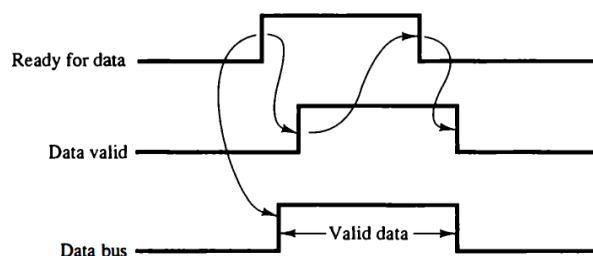
(c) Sequence of events

Figure 5 Source - initiated transfer using handshaking

The destination-initiated transfer using handshaking lines is shown in Fig. 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. 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.



(a) Block diagram



(b) Timing diagram

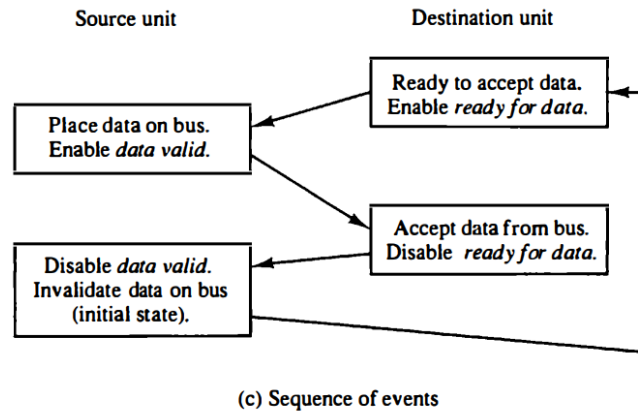


Figure 6 Destination-initiated transfer using handshaking.

3. 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 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.

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.

Transfer of data under programmed I/O is between CPU and peripheral. In direct memory access (DMA), the interface transfers data into and out of the memory unit through the memory bus. The CPU initiates the transfer by supplying the interface with the starting address and the number of words needed to be transferred and then proceeds to execute other tasks. When the transfer is made, the DMA requests memory cycles through the memory bus. When the request is granted by the memory controller, the DMA transfers the data directly into memory. The CPU merely delays its memory access operation to allow the direct memory I/O transfer. Since peripheral speed is usually slower than processor speed, I/O-memory transfers are infrequent compared to processor access to memory.

Many computers combine the interface logic with the requirements for direct memory access into one unit and call it an I/O processor (IOP). The IOP can handle many peripherals through a DMA and interrupt facility. In such a system, the computer is divided into three separate modules: the memory unit, the CPU, and the IOP.

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. 7. 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.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 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. 8. 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.

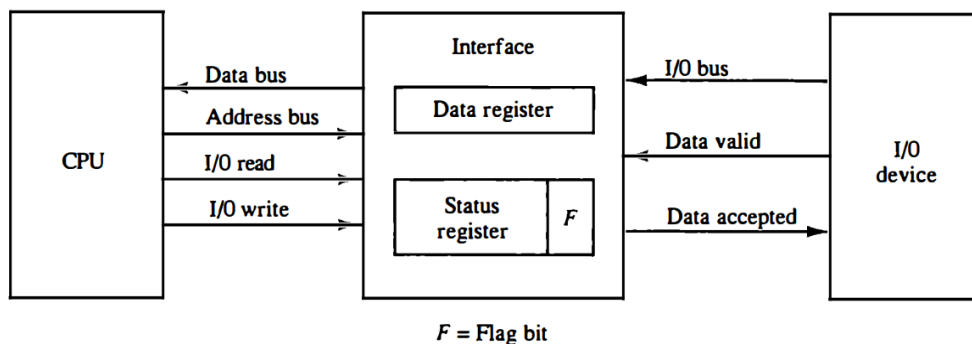


Figure 7 Data transfer from I/O device to CPU

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. 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 $1 \mu\text{s}$. Assume that the input device transfers its data at an average rate of 100 bytes per second. This is equivalent to one byte every $10,000 \mu\text{s}$. 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 tasks.

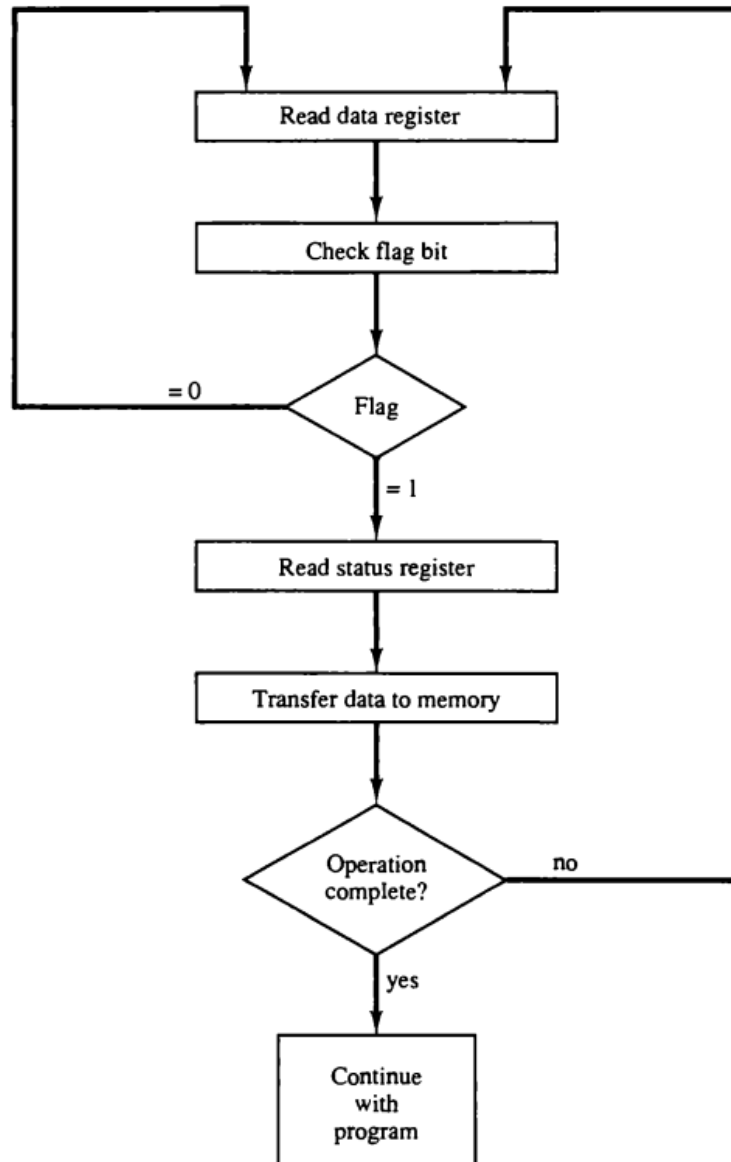


Figure 8 Flowchart for CPU program to input data.

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 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. In some cases, a control command is then given to execute a device function such as stop tape or print characters. Error checking and other useful steps often accompany the transfers. In interrupt-controlled transfers, the I/O software must issue commands to the peripheral to interrupt when ready and to service the interrupt when it occurs. In DMA transfer, the I/O software must initiate the DMA channel to start its operation.

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. They are usually included within the operating system. Most operating systems are supplied with a variety of I/O programs to support the particular line of peripherals offered for the computer. I/O routines are usually available as operating system procedures and the user refers to the established routines to specify the type of transfer required without going into detailed machine language programs.

4 Priority Interrupt

Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal. The CPU responds to the interrupt request by storing the return address from PC into a memory stack and then the program branches to a service routine that processes the required transfer. Some processors also push the current PSW (program status word) onto the stack and load a new PSW for the service routine. We neglect the PSW here in order not to complicate the discussion of I/O interrupts.

In a typical application a number of I/O devices are attached to the computer, with each device being able to originate an interrupt request. The first task of the interrupt system is to identify the source of the interrupt. There is also the possibility that several sources will request service simultaneously. In this case the system must also decide which device to service first.

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