CHAPTER

7

Input-Output Organization

7.1 INTRODUCTION

Data transfer between the computer and external device takes place through I/O mechanism. One communicates with a computer system via the I/O devices interfaced to it. The user can enter programs and data using the keyboard on a terminal, executes the programs to obtain results and finally the results may be displayed on monitor of the computer. Therefore, the I/O devices connected to a computer system provide an efficient means of communication between the computer and the outside world. These I/O devices are commonly known as *peripherals* and popular I/O devices used are keyboard, monitor, printer, disk and mouse.

In this chapter, we will consider in detail interconnection system and various ways in which I/O operations are performed.

7.2 I/O INTERFACE AND I/O DRIVER

Every computer supports a variety of peripheral devices. To use a peripheral device, two modules are required:

(a) I/O interface or I/O controller.
(b) I/O driver.

7.2.1 I/O Interface or I/O Controller

I/O interface is a hardware device provides a means for transferring information between central system (i.e. CPU and main memory) and external I/O peripheral device. Peripheral devices connected to a computer need I/O interface circuits for interfacing them with the CPU and/or memory. Each peripheral has its own I/O controller that operates the particular device. The purpose of the I/O interface is to resolve the differences that exist between the central computer and each peripheral. The major differences are:

Scanned by CamScanner

- Peripherals are mainly electromechanical and electromagnetic devices and their manner of operations is different from the operation of the CPU and main memory, which are electronic devices. Hence, a conversion of signal values may be required.
- Data codes and formats in peripheral devices are different from the code format in the CPU and memory.
- The data transfer rate of the peripheral devices is usually slower than the transfer rate of the CPU and therefore, a synchronization mechanism may be required.
- The various peripheral devices attached to a computer have different modes of operations and each must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

In order to resolve these differences, computer systems must include special hardware device called *I/O interface* with each peripheral to supervise and synchronize all I/O transfers.

The linkage of the I/O interface to the central computer is via the bus. A typical communication linkage between the central computer (i.e. CPU and main memory) and several peripheral devices is shown in Fig. 7.1. The I/O bus consists of data lines, address lines and control lines. The magnetic disk, printer, monitor and keyboard are used in practically any general-purpose computer. Each peripheral device has associated with it an I/O interface unit. A device's interface circuit constitutes of an address decoder, the data and status registers, and the control circuitry. Each I/O interface unit decodes the address and control signal received from the I/O bus, interprets them for the peripheral and provides signals for the peripheral. It synchronizes the data flow and supervises the transfer between peripheral and processor. For example, the printer controller controls the paper motion, the printing timing and the selection of printing characters. An I/O controller may be incorporated separately or may be physically integrated with the peripheral.

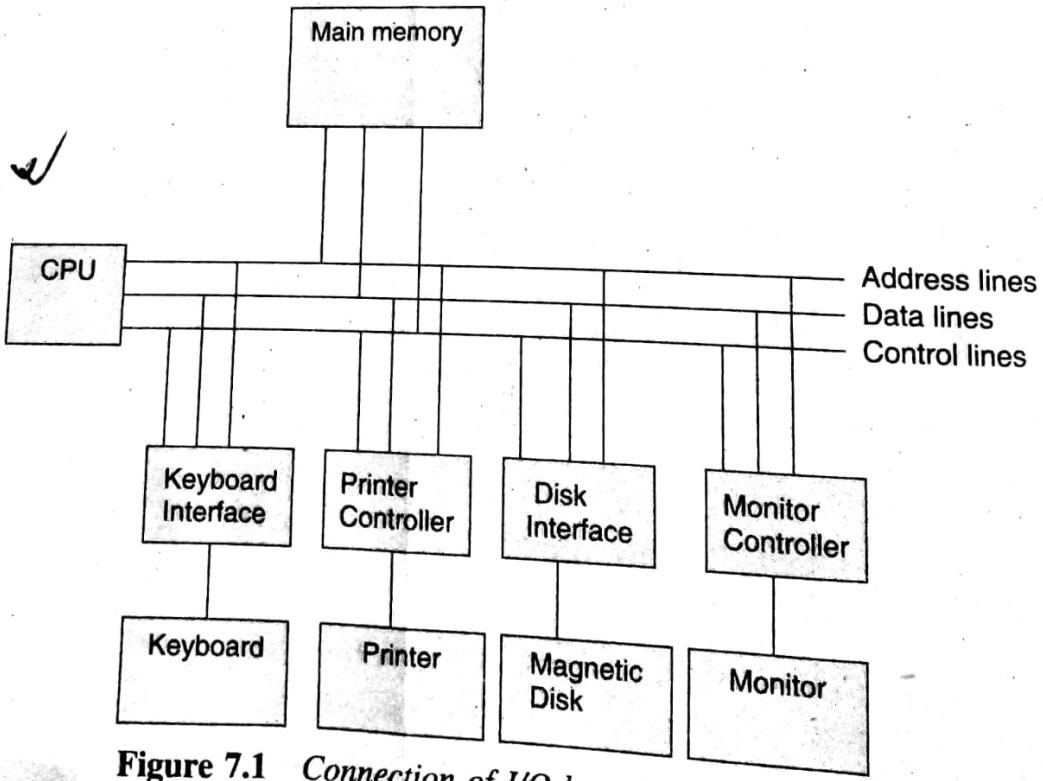

Figure 7.1 *Connection of I/O bus to I/O controllers*

Scanned by CamScanner

The I/O bus from the central computer is attached to all peripheral controllers. To communicate with a particular I/O device, the processor places a device address on the address lines. The address decoder of I/O interface monitors the address lines. When a particular I/O interface detects its own address, it activates the path between the bus lines and the peripheral device that it controls. All peripherals whose addresses do not correspond to the address in the bus are disabled by their interfaces.

When the address is made available in the address lines, at that time the processor provides an operation code in the control lines. The interface selected responds to the operation code and proceeds to execute it. The operation code is referred to as an I/O command. The meaning of the command depends on the peripheral type that the processor is addressing. There are four types of I/O commands that an I/O interface may receive when it is addressed by a processor:

1. ***Control*** It is used to enable an I/O device and to give directive what to do. For example, a magnetic tape unit may be instructed to rewind or to move forward one record. These commands are tailored to the particular type of peripheral device.

2. ***Test*** It is used to test various status conditions associated with an I/O interface and its peripheral. For example, the processor may want to know that the peripheral of interest is powered on and ready for use. It also may want to know if the most recent I/O operation is completed and if any error occurs.

3. ***Read*** This causes the I/O interface to obtain a data-item from the peripheral and places it in an internal buffer (data register). The processor can then obtain the data item by requesting that the I/O interface place it on the data bus.

4. ***Write*** This causes the I/O interface to take a data-item from the data bus and subsequently transfers that data item to the peripheral.

Some well known I/O interface devices are: SCSI (Small Computer System Interface), USB (Universal Serial Bus), IDE (Integrated Drive Electronics), RS-232C, FireWire, Centronics Interface.

There are two types of I/O interfaces available: *Serial interface* and *Parallel interface*. In *serial* interface, there is only one data line and thus data bits are transmitted serially one after other. Examples include: USB, RS-232C, and FireWire. In *parallel* interface, there are multiple data lines in parallel and thus multiple number of bits can be transmitted from the system simultaneously. Examples include: SCSI, Centronics Interface, and IDE.

7.2.2 I/O Driver

I/O driver is a software module that issues different commands to the I/O controller, for executing various I/O operations. Following are certain operations performed by different I/O drivers:

- Reading a file from a disk.
- Printing some lines by the printer.
- Displaying a message on monitor.
- Storing some data on disk.

The I/O driver program for a given peripheral device is developed only after knowing the architecture of the I/O controller device. The I/O driver program and I/O controller device together achieve the I/O operation done on behave of corresponding peripheral device. An I/O operation can be performed by calling the relevant I/O interface (or I/O controller) and passing relevant signals for

Scanned by CamScanner

that memory address values are not affected by interface address assignment since each has its own address space, as shown in Fig. 7.3. This method is followed in the IBM PC.

In the third case, computers use 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, as depicted in Fig. 7.4. The processor treats an I/O interface register as being part of the memory system. In other words, the processor uses a portion of the memory addresses to represent I/O interface. 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 I/O transfers or for memory transfers. Most of the modern computers use this technique; even though some computers like Intel 8088 support both I/O mapped I/O and memory mapped I/O techniques.

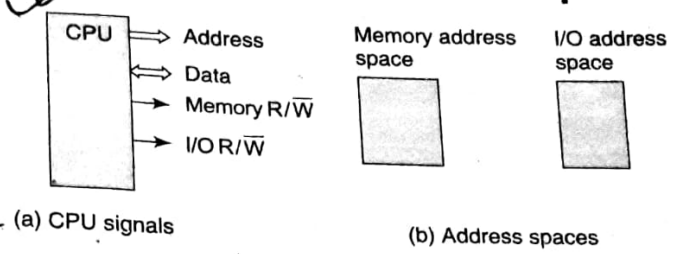

Figure 7.3 *I/O mapped I/O*

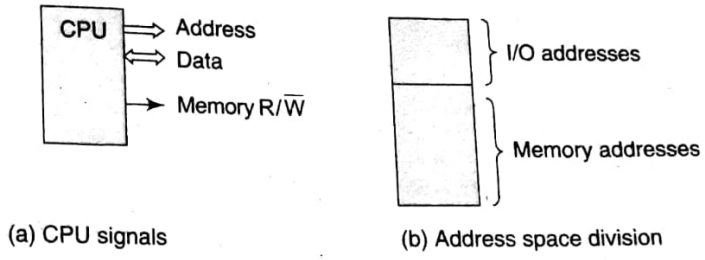

Figure 7.4 *Memory mapped I/O*

7.4 SYNCHRONOUS AND ASYNCHRONOUS DATA TRANSFERS

In order to execute a task in a computer, transfer of information among different devices is necessary. When two units have to communicate with each other for data transfer, usually one of them is the master and the other one is slave. Sometimes one word of data is transferred between two units in one clock period or some times in more than one clock period. There are two types of data transfer depending on the mechanism of timing the data: Synchronous and Asynchronous.

7.4.1 Synchronous Transfer

In this mode of data transfer, the sending and receiving units are enabled with same clock signal. The synchronous transfer is possible between two units when each of them knows the behavior of the other. The master performs a sequence of actions for data transfer in a predetermined order; each action is synchronized with the common clock. The master is designed to supply the data at a time when the slave is definitely ready for it. Usually, the master will introduce sufficient delay to take into account the slow response of the slave, without any request from the slave. The master does not expect any acknowledgement signal from the slave, when a data is sent by the master to the slave. Similarly, when a data from slave is read by the master, neither the slave informs that a data has been placed on the data bus nor the master acknowledges that a data has been read. Both master and slave perform their own task of transferring data at designated clock period. Since both devices know the behavior (response time) of each other, no difficulty arises. Prior to transferring data, the master mus

Scanned by CamScanner

logically select the slave either by sending slave's address or sending "device select" signal to the slave. But, there is no acknowledgement signal from the slave to master if device is selected.

As for example, the Fig. 7.5 shows timing diagram for synchronous read operation. The master first places slave's address in the address bus and read signal in the control line at the falling edge of a clock. The slave places data in the data bus at the raising edge of the clock. The entire read operation is over in one clock period.

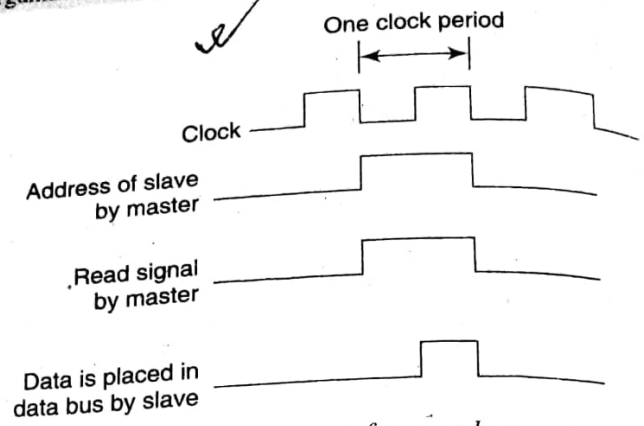

Figure 7.5 *Timing diagram for synchronous read operation*

Advantages of Synchronous Transfer

1. The design procedure is easy. The master does not wait for any acknowledge signal from the slave though the master waits for a time equal to slave's response time.
2. The slave does not generate acknowledge signal, though it obeys the timing rules as per the protocol set by the master or system designer.

Disadvantages of Synchronous Transfer

1. If a slow speed unit connected to a common bus, it can degrade overall rate of transfer in the system.
2. If the slave operates at a slow speed, the master will be idle for some time during data transfer and vice versa.

7.4.2 Asynchronous Transfer

There is no common clock between the master and slave in asynchronous transfer. Each has its own private clock for internal operations. This approach is widely used in most computers. 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 simple way is to use a strobe signal supplied by one of the units to indicate the other unit when the transfer has to occur.

Strobe Control Technique A single control line is used by the strobe control method of asynchronous data transfer to time each transfer. The strobe may be activated by either the source or the destination unit. A source-initiated transfer is depicted in Fig. 7.6. The source takes care of proper timing delay between the actual data signals and the strobe signal. The source places the data first, and after some delay, generates the strobe to inform about the data on the data bus. Before removing the data, source removes the strobe and after some delay it removes the data. By these two leading and trailing end delays, the system ensures the reliable data transfer.

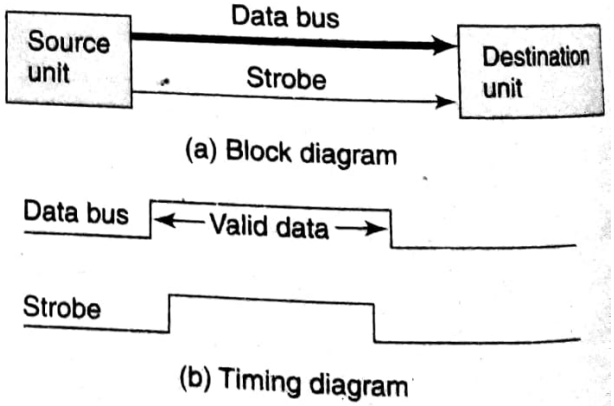

Figure 7.6 *Source-initiated strobe for data transfer*

Scanned by CamScanner

Similarly, the destination can initiate data transfer by sending strobe signal to the source unit as shown in Fig. 7.7. In response, the source unit places data on the data bus. After receiving data, the destination unit removes the strobe signal. Only after sensing the removal of strobe signal, the source removes the data from the data bus.

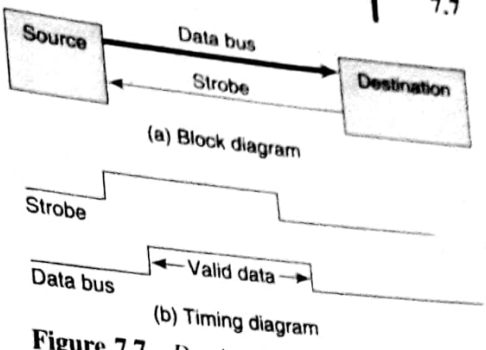

Figure 7.7 *Destination initiated strobe for data transfer*

The *disadvantage* of the strobe method is that the source unit that initiates the transfer cannot know whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer cannot know whether the source unit has actually placed the data on the bus.

Handshaking Technique To overcome this problem of strobe technique, 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 mode of transfer.

Figure 7.8 shows the data transfer method 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 source first places data and after some delay issues data valid signal. On sensing data valid signal, the destination receives data and then issues acknowledgement signal data accepted to indicate the acceptance of data. On sensing data accepted signal, the source removes data and data valid signal. On sensing removal of data valid signal, the destination removes the data accepted signal.

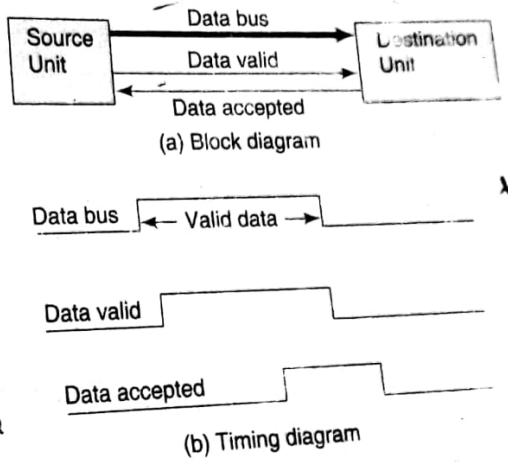

Figure 7.8 *Source-initiated transfer using handshaking*

Figure 7.9 illustrates destination initiated handshaking technique. The destination first sends the *data request* signal. On sensing this signal, the source places data and also issues the *data valid* signal. On sensing data valid signal, the destination acquires data and then removes the data request signal. On sensing this, the source removes both the data and data valid signal.

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

The disadvantages of handshaking scheme are:

1. A slow speed destination unit can hold up the bus whenever it gets a chance to communicate.
2. If one of the two communicating devices is faulty, the initiated data transfer cannot be completed."

Scanned by CamScanner

2. Most microprocessors such as Motorola 88010 and Intel 80286 follow this bus transfer mechanism.

7.5 MODES OF DATA TRANSFER

Information transferred from the central computer (i.e. CPU and main memory) into a peripheral device originates in the memory unit. Information received from a peripheral device is usually stored in memory for later processing. The CPU only 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. Three possible modes are

1. Programmed I/O.
2. Interrupt-initiated I/O.
3. Direct memory access (DMA).

The CPU executes a program to communicate with an I/O device via I/O interface registers for programmed I/O. This is a software method.

An external I/O device requests the processor to transfer data by activating a signal on the computer's interrupt line during interrupt-initiated I/O. In response, the computer executes a program called the *interrupt-service routine* (ISR) to carry out the function desired by the external I/O device. This is also a software method.

Data transfer between the computer's main memory and an external I/O device occurs without CPU involvement in direct memory access (DMA). It is a hardware method.

Scanned by CamScanner

assigns a fixed priority for the various interrupt requestor devices. For example, the IRQ0 is assigned the highest priority among the eight different interrupt requestors. Assigning decreasing order of priority from IRQ0 to IRQ7, the IRQ7 is the lowest priority. It (IRQ7) is serviced only when no other interrupt request is present.

7.5.3 Direct Memory Access (DMA)

To transfer large blocks of data at high speed, this third method is used. A special controlling unit may be provided to allow transfer a block of data directly between a high speed external device like magnetic disk and the main memory, without continuous intervention by the CPU. This method is called *direct memory access (DMA)*.

DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA *controller*. The DMA controller performs the functions that would normally be carried out by the CPU when accessing the main memory. During DMA transfer, the CPU is idle or can be utilized to execute another program and CPU has no control of the memory buses. A DMA controller takes over the buses to manage the transfer directly between the I/O device and the main memory.

The CPU can be placed in an idle state using two special control signals, HOLD and HLDA (hold acknowledge). Figure 7.13 shows two control signals in the CPU that characterize the DMA transfer. The HOLD input is used by the DMA controller to request the CPU to release control of buses. When this input is active, the CPU suspends the execution of the current instruction and places the address bus, the data bus and the read/write line into a high-impedance state. The high-impedance state behaves like an open circuit, which means that the output line is disconnected from the input line and does not have any logic significance. The CPU activates the HLDA output to inform the external DMA controller that the buses are in the high-impedance state. The control of the buses has been taken by the DMA controller that generated the bus request to conduct memory transfers without processor intervention. After the transfer of data, the DMA controller disables the HOLD line. The CPU then disables the HLDA line and regains the control of the buses and returns to its normal operation.

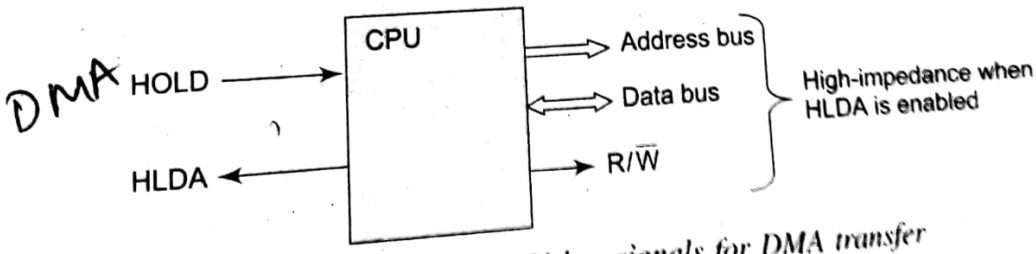

Figure 7.13 *CPU bus signals for DMA transfer*

DMA Controller To communicate with the CPU and I/O device, the DMA controller needs the usual circuits of an interface. In addition to that, it needs an address register, a word count register, a status register and a set of address lines. Three registers are selected by the controller's register select (RS) line. The address register and address lines are used for direct communication with the memory. The address register is used to store the starting address of the data block to be transferred. The word count register contains the number of words that must be transferred. This register is decremented by

Scanned by CamScanner

one after each word transfer and internally tested for zero after each transfer. Between the device and memory under control of the DMA, the data transfer can be done directly. The status register contains information such as completion of DMA transfer. All registers in the DMA controller appear to the CPU as I/O interface registers. Thus, the CPU can read from or write into the DMA registers under program control via the data bus.

While executing the program for I/O transfer, the CPU first initializes the DMA controller. After that, the DMA controller starts and continues to transfer data between memory and peripheral until an entire block is transferred. The DMA controller is initialized by the CPU by sending the following information through the data bus:

1. The starting address of the memory blocks where data are available for or where data are to be stored for write.
2. The number of words in the memory block (word count) to be read or written.
3. Read or write control to specify the mode of transfer.
4. A control to start the DMA transfer.

DMA Transfer In DMA transfer, I/O devices can directly access the main memory without intervention by the processor. Figure 7.14 shows a typical DMA system. The sequences of events involved in a DMA transfer between an I/O device and the main memory are discussed next.

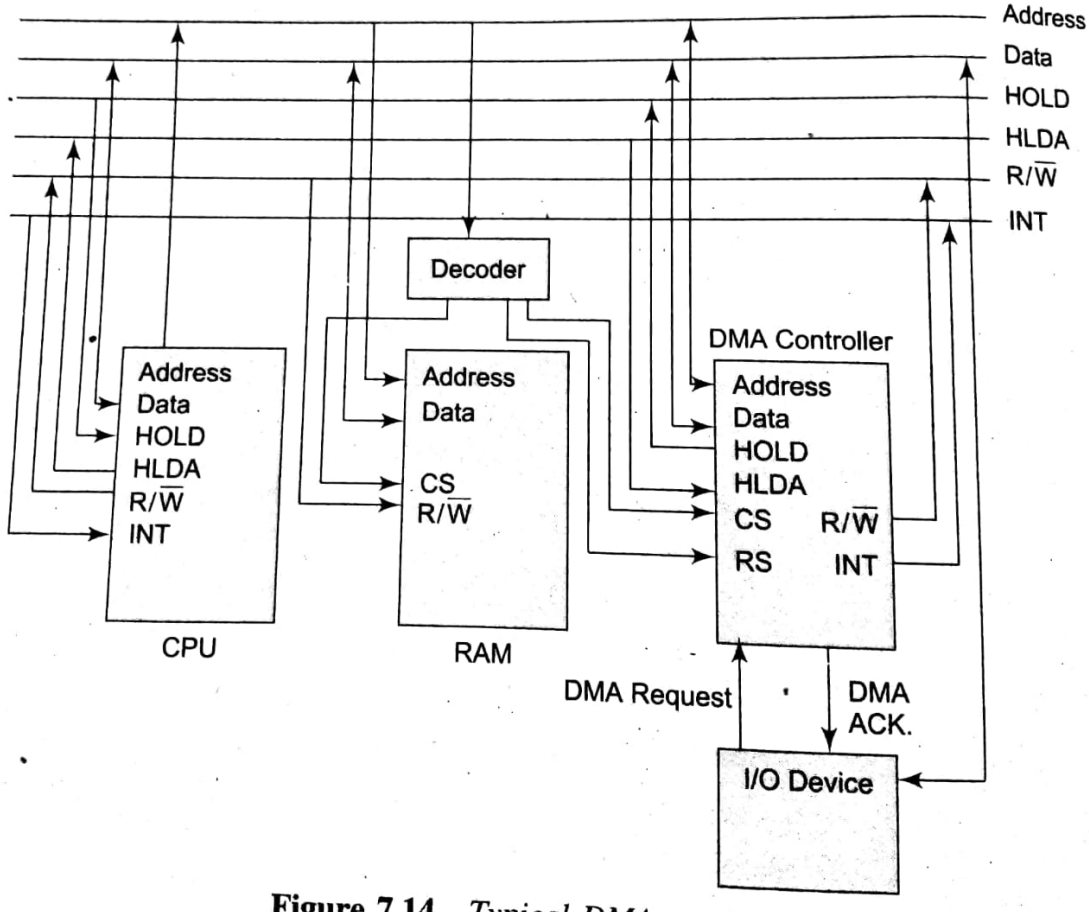

Figure 7.14 *Typical DMA system*

A DMA request signal from an I/O device starts the DMA sequence. DMA controller activates the HOLD line. It then waits for the HLDA signal from the CPU. On receipt of HLDA, the controller

Scanned by CamScanner

sends a DMA ACK (acknowledgement) signal to the I/O device. The DMA controller takes the control of the memory buses from the CPU. Before releasing the control of the buses to the controller, the CPU initializes the address register for starting memory address of the block of data, word-count register for number of words to be transferred and the operation type (read or write). The I/O device can then communicate with memory through the data bus for direct data transfer. For each word transferred, the DMA controller increments its address-register and decrements its word count register. After each word transfer, the controller checks the DMA request line. If this line is high, next word of the block transfer is initiated and the process continues until word count register reaches zero (i.e., the entire block is transferred). If the word count register reaches zero, the DMA controller stops any further transfer and removes its HOLD signal. It also informs the CPU of the termination by means of an interrupt through INT line. The CPU then gains the control of the memory buses and resumes the operations on the program which initiated the I/O operations.

Advantages of DMA It is a hardware method, whereas programmed I/O and interrupt I/O are software methods of data transfer. DMA mode has following advantages:

1. High speed data transfer is possible, since CPU is not involved during actual transfer, which occurs between I/O device and the main memory.
2. Parallel processing can be achieved between CPU processing and DMA controller's I/O operation.

DMA Transfer Modes DMA transfers can be of two types: *cycle stealing* and *block (burst) transfer*.

Memory accesses by the CPU and the DMA controllers are interlocking. Requests by DMA devices for using memory buses are always given higher priority than processor requests. Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface or a graphics display device. Since the CPU originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the CPU. Hence, this interlocking technique usually called *cycle stealing*.

When DMA controller is the master of the memory buses, a block of memory words is transferred in continuous without interruption. This mode of DMA transfer is known as *block (burst) transfer*. This mode of transfer is needed for fast devices such as magnetic disks, where data transmission cannot be stopped or slowed down until an entire block is transferred.

7.6 BUS ARBITRATION

A conflict may arise if the number of DMA controllers or other controllers or processors try to access the common bus at the same time, but access can be given to only one of those. Only one processor or controller can be bus master. The bus master is the controller that has access to a bus at an instance. To resolve these conflicts, bus arbitration procedure is implemented to coordinate the activities of all devices requesting memory transfers. *Bus arbitration* refers to a process by which the current bus master accesses and then leaves the control of the bus and passes it to another bus requesting processor unit. The selection of the bus master must take into account the needs of various devices by

Scanned by CamScanner

establishing a priority system for gaining access to the bus. The *bus arbiter* decides who would become current bus master. There are two approaches to bus arbitration:

1. Centralized bus arbitration: A single bus arbiter performs the required arbitration.
2. Distributed bus arbitration: All devices participate in the selection of the next bus master.

7.6.1 Methods of Bus Arbitration

There are three bus arbitration methods:

1. Daisy Chaining Method.
2. Polling or Rotating Priority Method.
3. Fixed Priority or Independent Request Method.

Daisy Chaining Method The daisy chaining method is a centralized bus arbitration method. During any bus cycle, the bus master may be any device - the processor or any DMA controller unit, connected to the bus. Figure 7.15 illustrates the daisy chaining method.

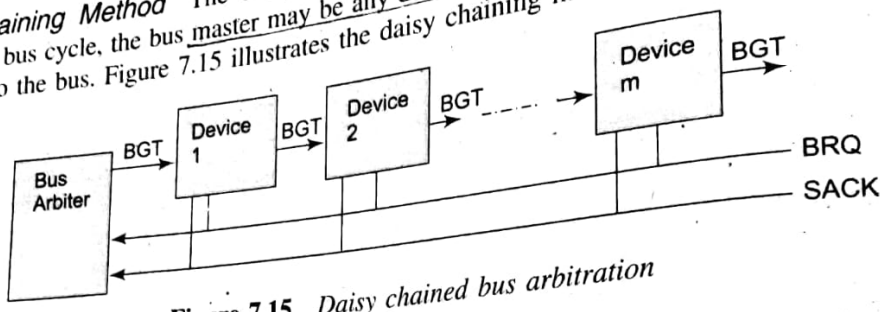

Figure 7.15 *Daisy chained bus arbitration*

All devices are effectively assigned static priorities according to their locations along a bus grant control line (BGT). The device closest to the central bus arbiter is assigned the highest priority. Requests for bus access are made on a common request line, BRQ. Similarly, the common acknowledge signal line (SACK) is used to indicate the use of bus. When no device is using the bus, the SACK is inactive. The central bus arbiter propagates a bus grant signal (BGT) if the BRQ line is high and acknowledge signal (SACK) indicates that the bus is idle. The first device, which has issued a bus request, receives the BGT signal and stops the latter's propagation. This sets the bus-busy flag in the bus arbiter by activating SACK and the device assumes bus control. On completion, it resets the bus-busy flag in the arbiter and a new BGT signal is generated if other requests are outstanding (i.e., BRQ is still active). The first device simply passes the BGT signal to the next device in the line.

The main advantage of the daisy chaining method is its simplicity. Another advantage is scalability. The user can add more devices anywhere along the chain, up to a certain maximum value.

Polling or Rotating Priority Method In this method, the devices are assigned unique priorities and compete to access the bus, but the priorities are dynamically changed to give every device an opportunity to access the bus. This dynamic priority algorithm generalizes the daisy chain implementation of static priorities discussed above. Recall that in the daisy chain scheme all devices are given static and unique priorities according to their positions on a bus-grant line (BGT) emanating from a central bus arbiter. However, in the polling scheme, no central bus arbiter exists, and the bus-grant line (BGT) is connected from the last device back to the first in a closed loop (Fig. 7.16). Whichever device is granted access to the bus serves as bus arbiter for the following arbitration (an arbitrary

Scanned by CamScanner

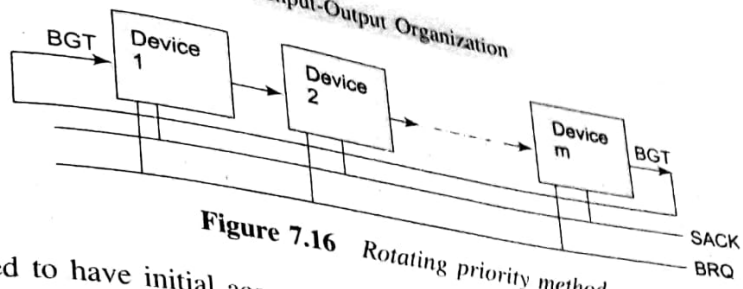

Figure 7.16 *Rotating priority method*

device is selected to have initial access to the bus). Each device's priority for a given arbitration is determined by that device's distance along the bus-grant line from the device currently serving as bus arbiter; the latter device has the lowest priority. Hence, the priorities change dynamically with each bus cycle.

The main advantage of this method is that it does not favor any particular device or processor. The method is also quite simple.

Fixed Priority or Independent Request Method In bus independent request method, the bus control passes from one device to another only through the centralized bus arbiter. Figure 7.17 shows the independent request method. Each device has a dedicated BRQ output line and BGT input line. If there are m devices, the bus arbiter has m BRQ inputs and m BGT outputs. The arbiter follows a priority order with different priority level to each device. At a given time, the arbiter issues bus grant (BGT) to the highest priority device among the devices who have issued bus requests. This scheme needs more hardware but generates fast response.

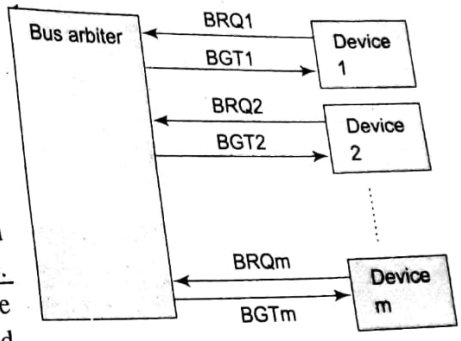

Figure 7.17 *Fixed priority bus arbitration method*

7.7 INPUT-OUTPUT PROCESSOR (IOP)

The DMA mode of data transfer reduces CPU's overhead in handling I/O operations. It also allows parallelism in CPU and I/O operations. Such parallelism is necessary to avoid wastage of valuable CPU time while handling I/O devices which are much slower compared to CPU. The concept of DMA operation can be extended to relieve the CPU further from getting involved with the execution of I/O operations. This gives rise to the development of special purpose processor called IO processor (IO channel).

The IOP is just like a CPU that handles the details of I/O operations. It is more equipped with facilities than those are available in a typical DMA controller. The IOP can fetch and execute its own instructions that are specifically designed to characterize I/O transfers. In addition to the I/O-related tasks, it can perform other processing tasks like arithmetic, logic, branching and code translation. The block diagram of an IOP is shown in Fig. 7.18. The main memory unit takes the pivotal role. It communicates with processor by means of DMA.

Scanned by CamScanner