Unit - V

INPUT-OUTPUT ORGANIZATION

Objective:

- To familiarize the working of different I/O devices connected to the system and their inter communication.

Syllabus:

INPUT-OUTPUT ORGANIZATION: Peripheral Devices, input-output interface, asynchronous data transfer, modes of transfer- programmed I/O, priority interrupt, direct memory access, Input –Output Processor (IOP).

Learning Outcomes:

At the end of the unit student will be able to:

1. Summarize different ways in which data transfer takes place between I/O devices and Processor.
2. Differentiate between synchronous and asynchronous communication.

Learning Material

5.1 PERIPHERAL DEVICES

- The input-output subsystem of a computer, referred to as I/O.
- Input or output devices attached to the computer are also called peripherals.
- Among the most common peripherals are keyboards, display units, and printers.
- Devices that are under the direct control of the computer are said to be connected on-line.
- 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.

5.2 INPUT-OUTPUT INTERFACE

- Input-output interface provides a method for transferring information between internal storage and external I/0 devices.
- Peripherals connected to a computer need special communication links for interfacing them with the CPU.
- The purpose of the communication link is to resolve the differences that exist between the CPU 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.

To resolve the above 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.

5.2.1 I/O Bus and Interface Modules

- Here the I/O bus consists of data lines, address lines, and control lines.
- Each peripheral device has associated with it an interface unit.
- Each interface decodes the address and control received from the I/O bus, and provides signals for the peripheral controller. It also synchronizes the data flow and supervises the data transfer between peripheral and processor.

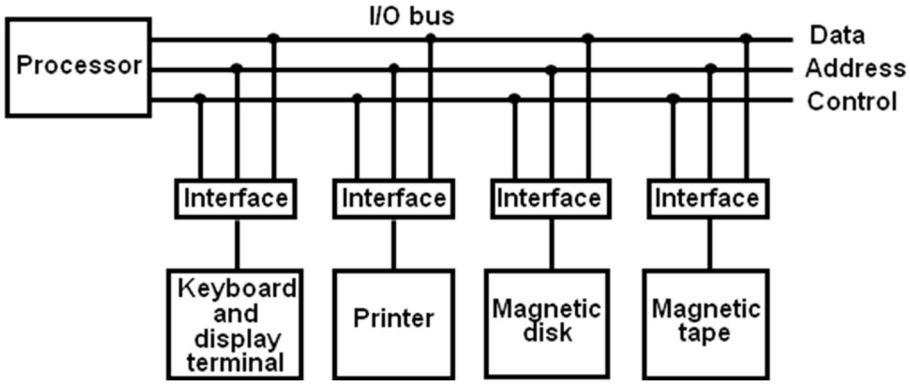


Figure 1: Connection of I/O bus to input output devices

- 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 1/0 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.
- There are four types of commands that an interface may receive. They are classified as:
    1. Control command
    2. Status command
    3. Data output command

4. Data input command

- A control command is issued to activate the peripheral and to inform it what to do.
- 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
- A data output command causes the interface to respond by transferring data from the bus into one of its registers.
- The data input command causes the interface receive an item of data from the peripheral and places it in its buffer register.

5.2.2 I/O Bus 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.

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

Isolated I/O:

- The distinction between a memory transfer and I/O transfer is made through separate read and 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.

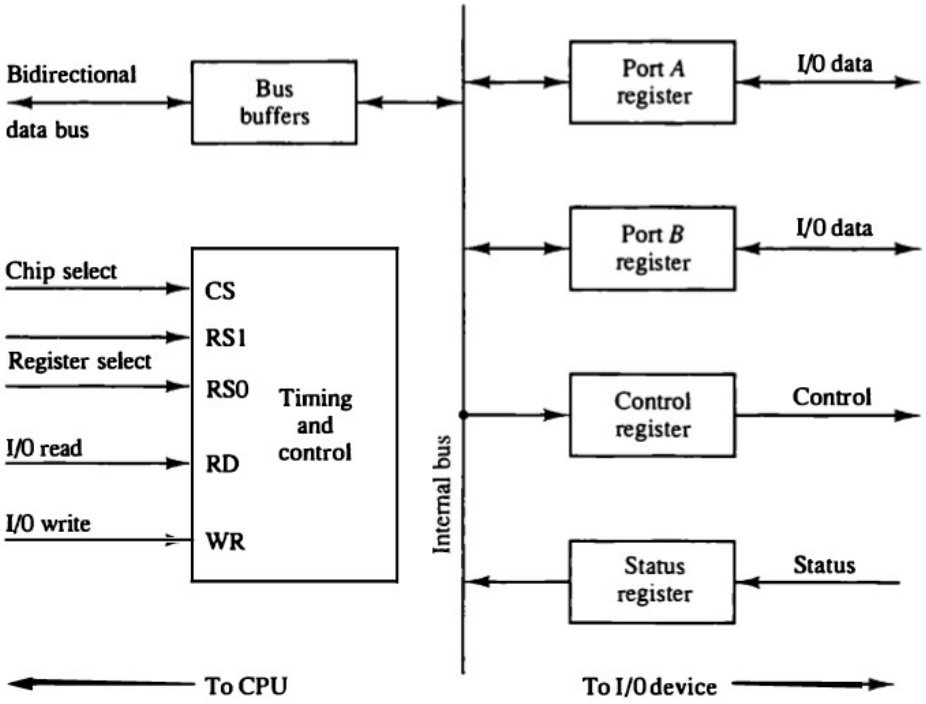
Memory-Mapped I/O:

- It uses the same address space for both memory and I/O.
- In this is the case computers has 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

5.2.4 Example of I/O Interface

- An example of an I/O interface unit is shown in block diagram form in figure 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 bidirectional 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 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.
- This circuit gets enabled when the chip select (CS) line is active. The CS is active when the address generated by the processor is for this device.



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

Figure 2: Example of I/O interface unit

5.3. ASYNCHRONOUS DATA TRANSFER

- The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
- Two units, such as a CPU and an I/O interface, are designed independent 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.
- 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.
- Two methods are available:
  1. Strobe Control
  2. Handshaking

5.3.1. Strobe Control

- A strobe pulse supplied by one of the units to indicate to the other unit when the transfer has to occur.
- The strobe control method of asynchronous data transfer employs a single control line.
- The strobe may be activated by either the source or the destination unit.

5.3.1.1 Source initiated strobe

- The following figure 3(a) shows a source-initiated transfer.
- The data bus carries the binary information from source unit to the destination unit. The strobe informs the destination unit when a valid data word is available in the bus.

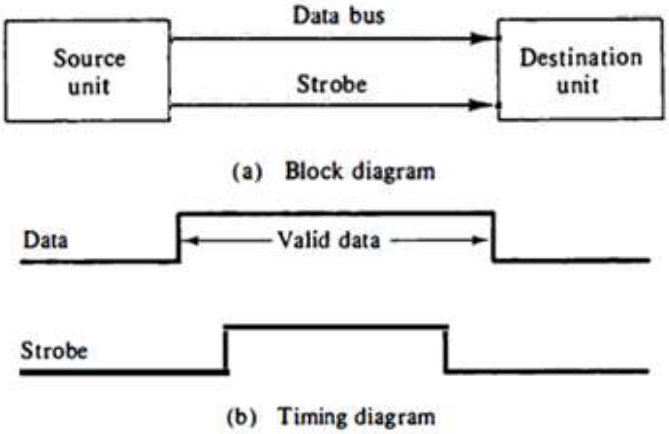


Figure 3: Source initiated strobe for data transfer

- The timing diagram is as shown in the above figure 3(b), the source unit first places the data on the data bus. After a brief delay the source activates the strobe pulse.
- The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data.
- Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.
- The source removes the data from the bus after a brief period by disabling 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.

5.3.1.2 Destination initiated strobe

- The following 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 longenough 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.

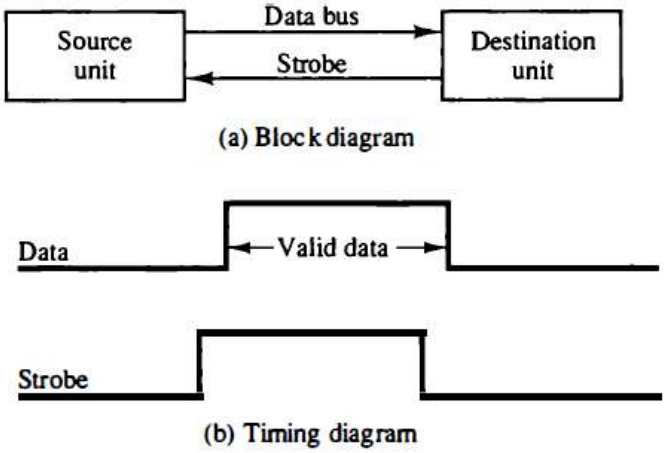


Figure 4: Destination initiated strobe for data transfer

5.3.2 Handshaking

- The disadvantage of the strobe method is that the source unit that initiates the transfer has no knowledge of the destination unit has actually received the data item or not from 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 these problems by introducing a second control signal that provides a reply to the unit that initiates the transfer.
- The basic principle of the two-way handshaking method of data transfer is as follows:
    - One control line is in the same direction as the data flow i.e., from the source to the destination. It is used by the source to inform the destination about the data validity.
    - The other control line is in the other direction i.e., from the destination to the source. It is used by the destination unit to inform the source regarding data acceptance.
    - The sequence of control signals exchanged during the transfer depends on the unit that initiates the transfer.

5.3.2.1 Source initiated transfer using handshaking

- The following 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.

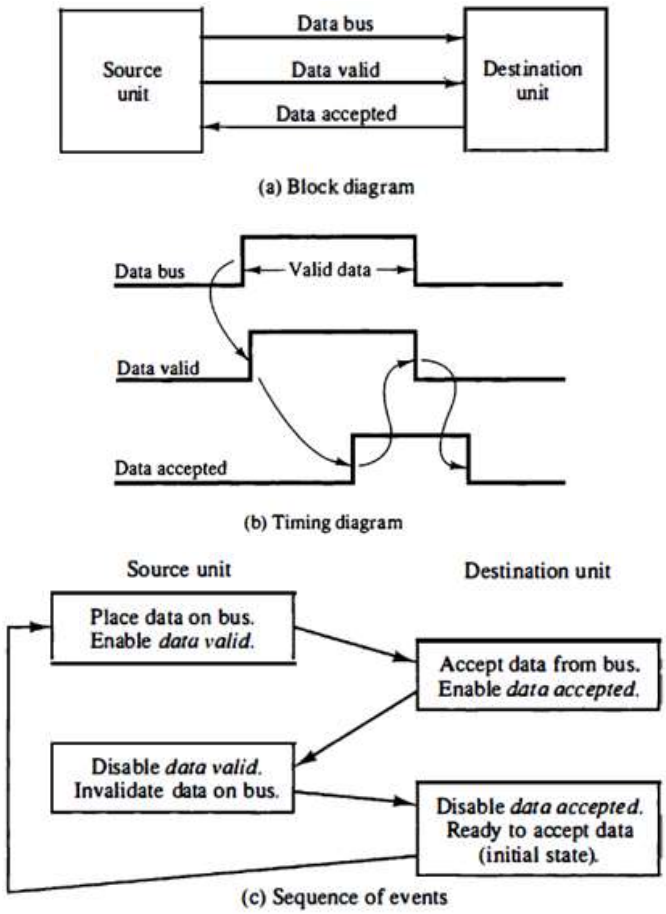


Figure 5: Source initiated transfer using handshaking

5.3.2.2 Destination initiated transfer using handshaking

- The destination-initiated transfer using handshaking is shown in the following figure 6.
- The source unit in this case does not place data on the bus till it receives the ready for data signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source-initiated case.
- The only difference between the source-initiated and the destination-initiated transfer is in their choice of initial state.

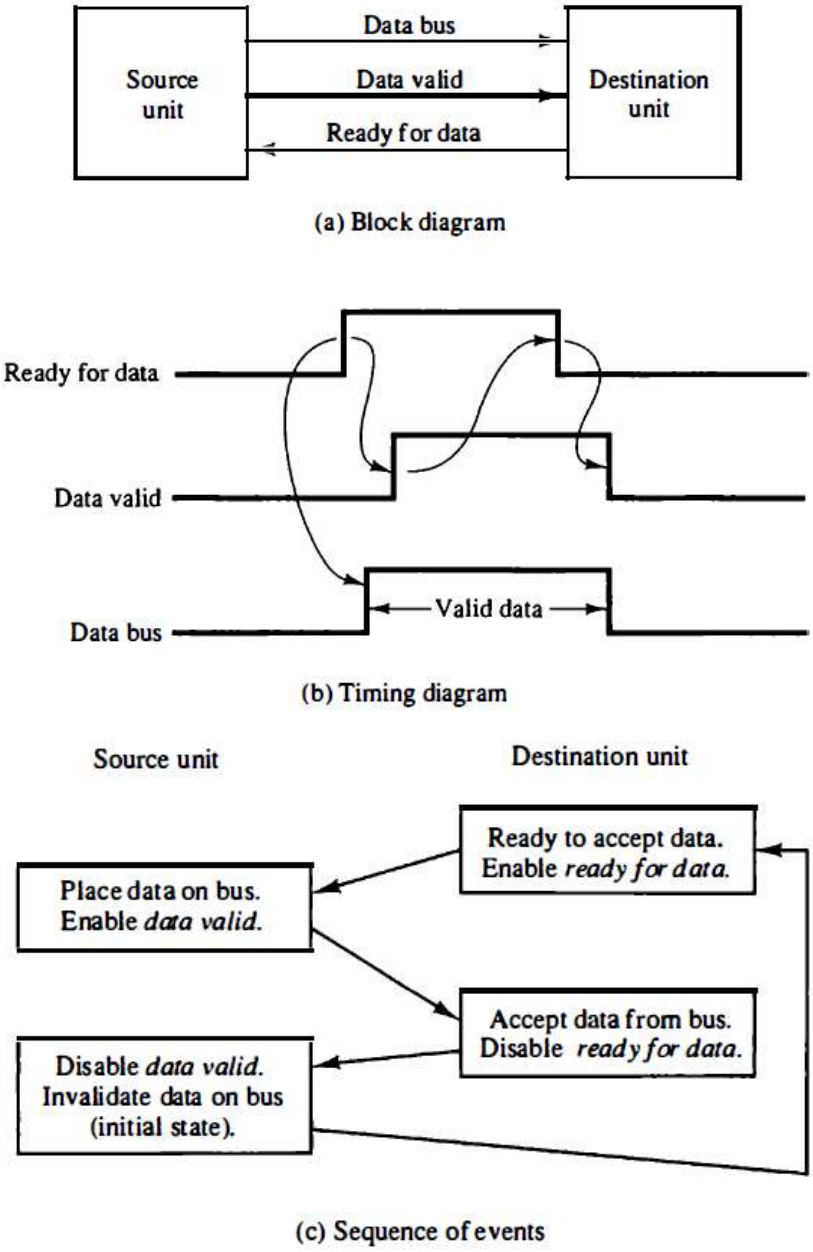
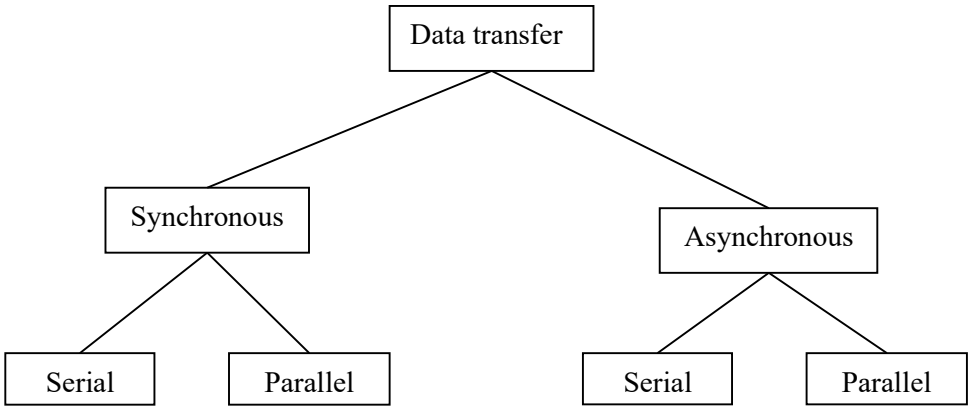


Figure 6: Destination initiated transfer using handshaking



5.3.3 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.
- 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 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.

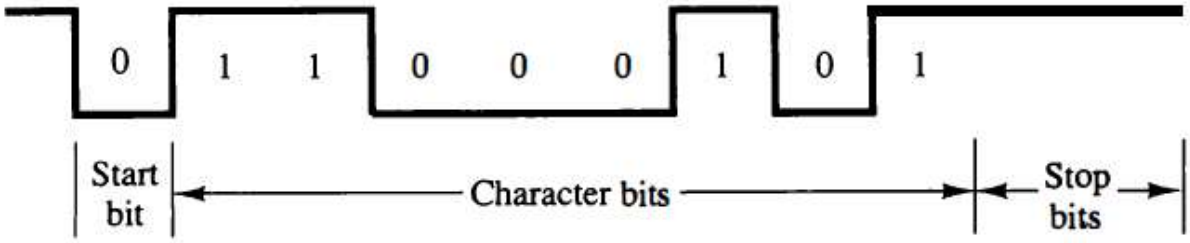


Figure 7: Asynchronous serial transmission

- 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.

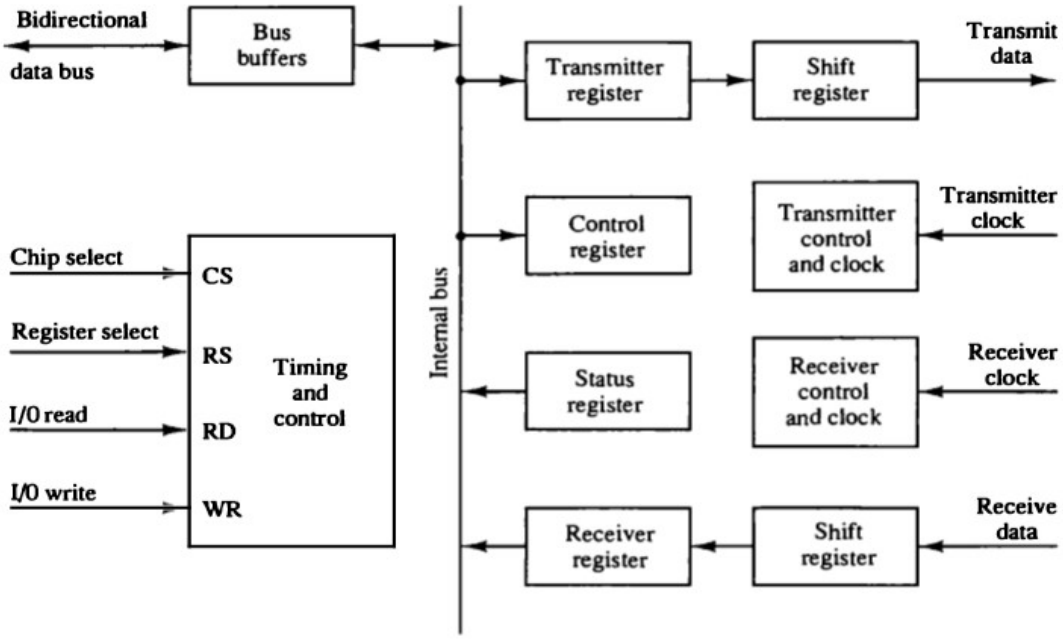
  2. The initiation of a character transmission is detected from the start bit, which is always 0.

  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.
- After the character bits one or two *stop bits* are transmitted.
- At the end of the character the line is held at the 1-state for a period of at least one or two bit times so that both the transmitter and receiver can resynchronize.
- The interface between computer and similar interactive terminals is called an asynchronous communication interface or a universal asynchronous receiver transmitter (UART).

5.3.4 Asynchronous Communication Interface

- The block diagram of an asynchronous communication interface is shown in Fig 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 (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.



CS	RS	Operation	Register selected
0	×	×	None: data bus in high-impedance
1	0	WR	Transmitter register
1	1	WR	Control register
1	0	RD	Receiver register
1	1	RD	Status register

Figure 8: Block diagram of a typical asynchronous communication interface

- The operation of the transmitter portion of the interface is that the CPU reads the status register and checks the flag to see if the transmitter register is empty.
- If status register is empty, the CPU transfers a character to the transmitter register and the interface clears the flag to mark the register full.
- The operation of the receiver portion of the interface is similar. The received 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.

5.3.5 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.
- 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.
- The logic diagram of a typical 4 x 4 FIFO buffer is shown in Fig 9.

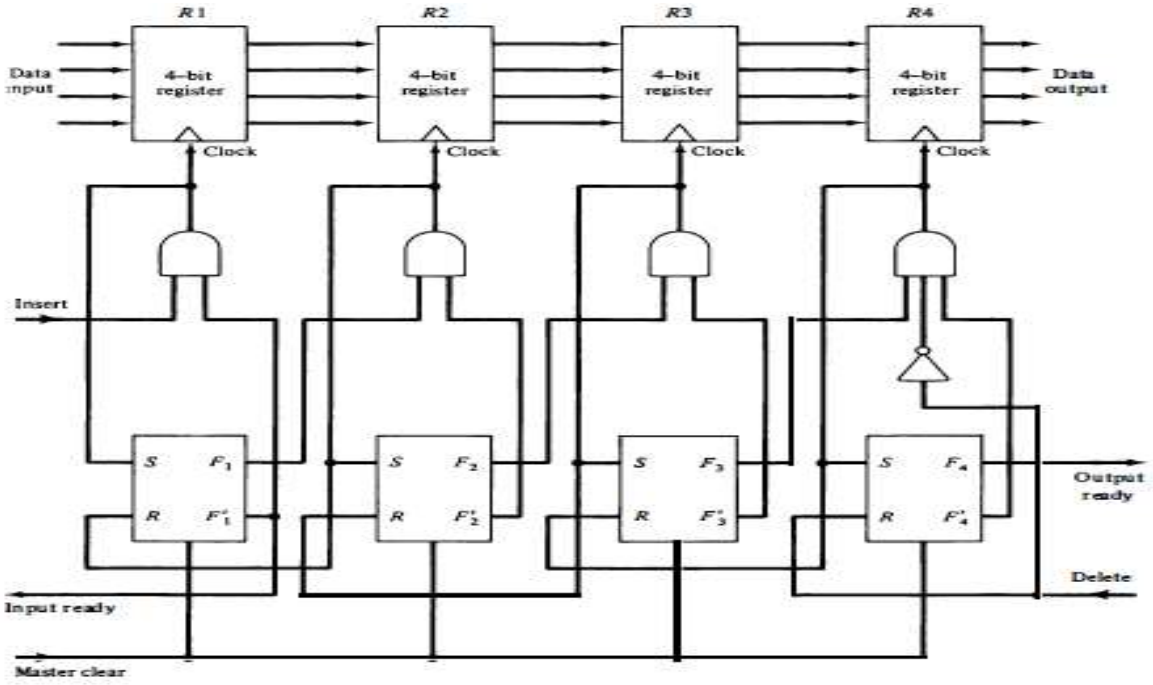


Figure 9: First-In First-Out Buffer

- It consists of four 4-bit registers RI, I = 1, 2, 3, 4, and a control register withflip-flops F_i, i = 1, 2, 3, 4, one for each register.
- A flip-flop F_i 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 F_i indicates that the corresponding register does not contain valid data.
- If F_i= 1 and the F_{i+1}'=1 then a clock is generated causing register R_{i+1} to accept the data from register R_i. (i.e. $R_{i+1} \leftarrow R_i$).
- The same clock transition sets F_{i+1}=1 and resets F_i'=1. This causes the control flag to move one position to the right together with the data.

- 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.
- The same clock sets F_1, 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 F_1 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.
- 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 F_4, causing output ready to disable, indicating that the data on the output are no longer valid.

5.4 MODES OF TRANSFER

- Data transfer between the central computer and 110 devices may be handled in a variety of modes.
- 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)

5.4.1. 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.
- In the programmed I/O method, the I/Odevice 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. 10.

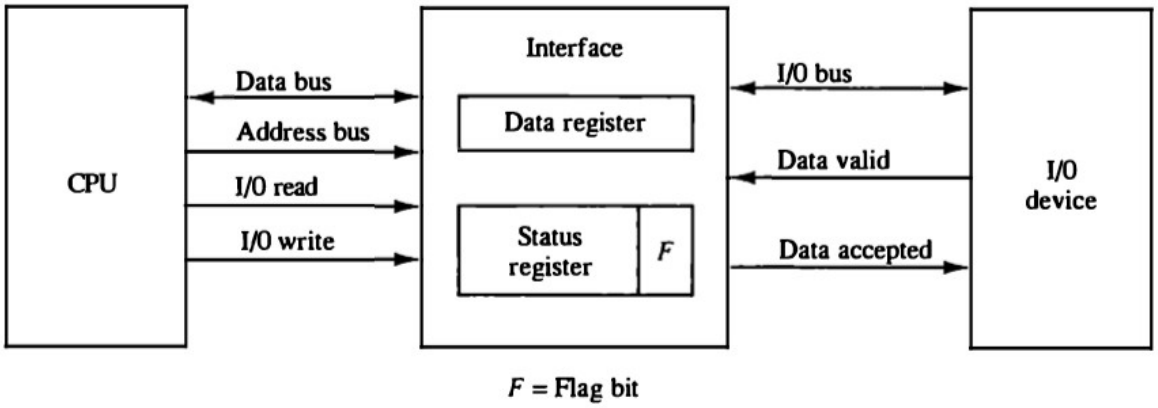


Figure 10: Data transfer from I/O device to CPU

- When a byte of data is available, the device places it on 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 For "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.
- A flowchart of the program that must be written for the CPU is shown in Fig 11.
- 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.

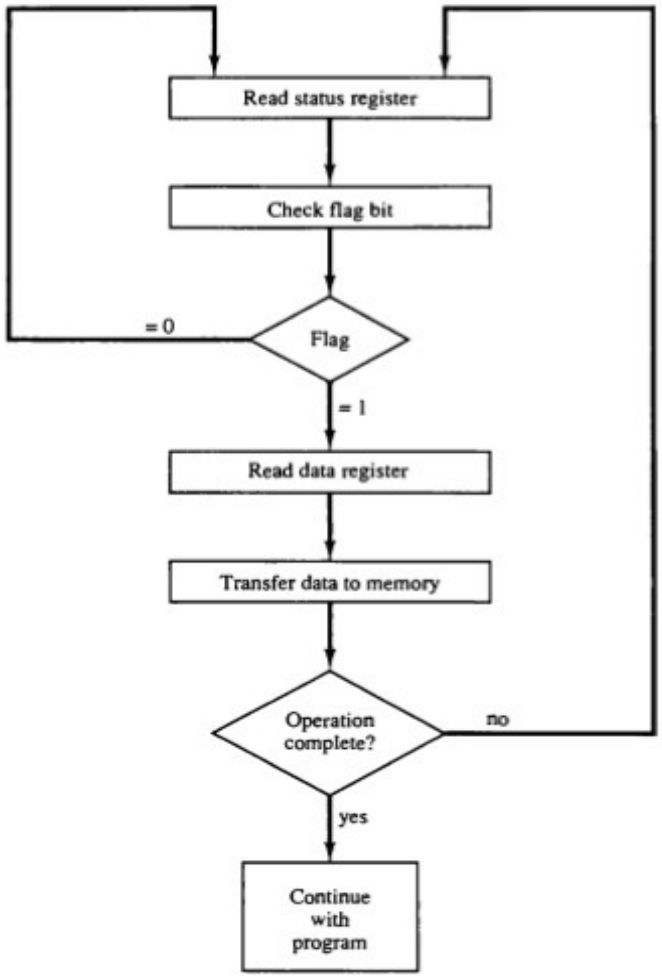


Figure 11: Flowchart for CPU program to input data

- The programmed I/O method is particularly useful in small low-speed computers or in systems that are dedicated to monitor a device continuously.

5.4.2. Interrupt-initiated I/O

- In the programmed I/O method, the CPU stays in a 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.
- 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.

5.4.2.1 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.
- 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.
- 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 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 in two ways.

  1. Serial connection (daisy chaining)
  2. Parallel connection

5.4.3 Daisy-Chaining Priority (Serial connection)

- 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 devices and the CPU is shown in Fig. 12.

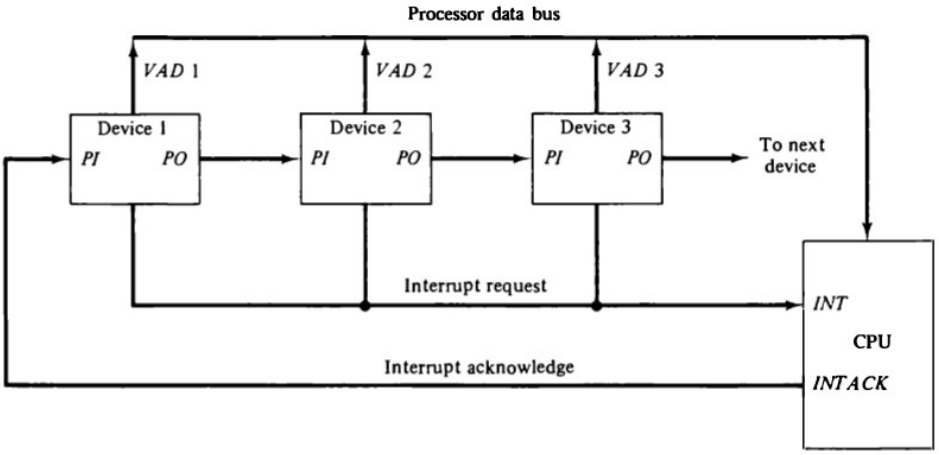


Figure 12: Daisy-Chain Priority interrupt

- 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.
- A device with a 0 in its PI input generates a 0 in its PO output to inform the next-lower-priority device that the acknowledge signal has been blocked.
- A device that is requesting an interrupt and has a 1 in its PI input will intercept the acknowledge signal by placing a 0 in its PO output.
- If the device does not have pending interrupts, it transmits the acknowledge signal to the next device by placing a 1 in its PO output.
- Thus the device with PI = 1 and PO = 0 is the one with the highest priority that is requesting an interrupt, and this device places its VAD on the data bus.
- The daisy chain arrangement gives the highest priority to the device that receives the interrupt acknowledge signal from the CPU.
- The following figure 13 shows the internal logic that must be included within each device when connected in the daisy-chaining scheme.

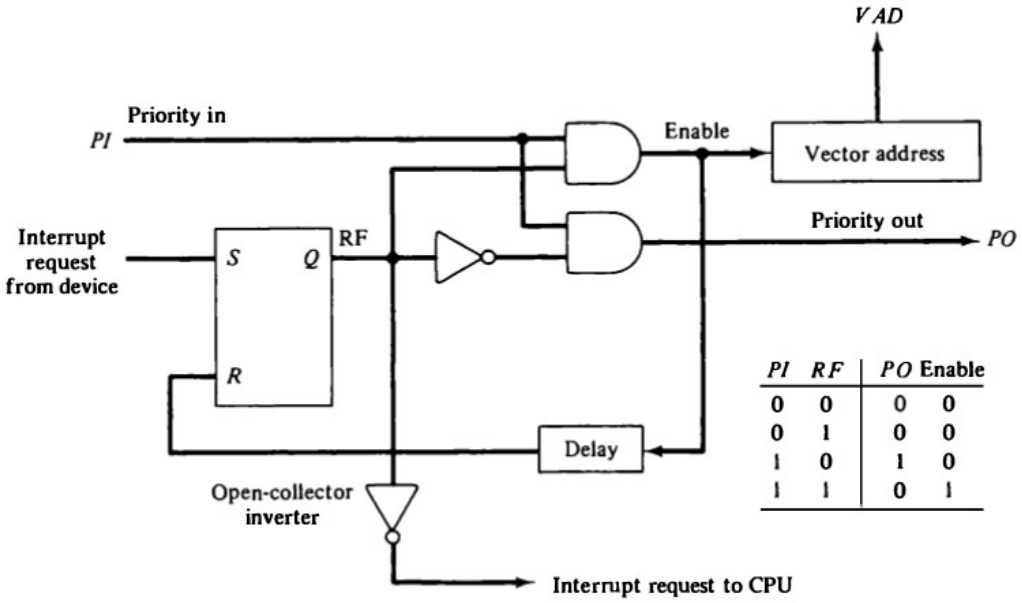


Figure 13: One stage of the daisy-chain priority arrangement

- The device sets its RF flip-flop when it wants to interrupt the CPU.
- If PI = 0, both PO and the enable line to VAD are equal to 0, irrespective of the value of RF.
- If PI = 1 and RF = 0, then PO = 1 and the vector address is disabled. This condition passes the acknowledge signal to the next device through PO.

- The device is active when PI = 1 and RF = 1. This condition places a 0 in PO and enables the vector address for the data bus.
- The RF flip-flop is reset after a sufficient delay to ensure that the CPU has received the vector address.

5.4.4 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 priority logic for a system of four interrupt sources is shown in Fig. 14. It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.

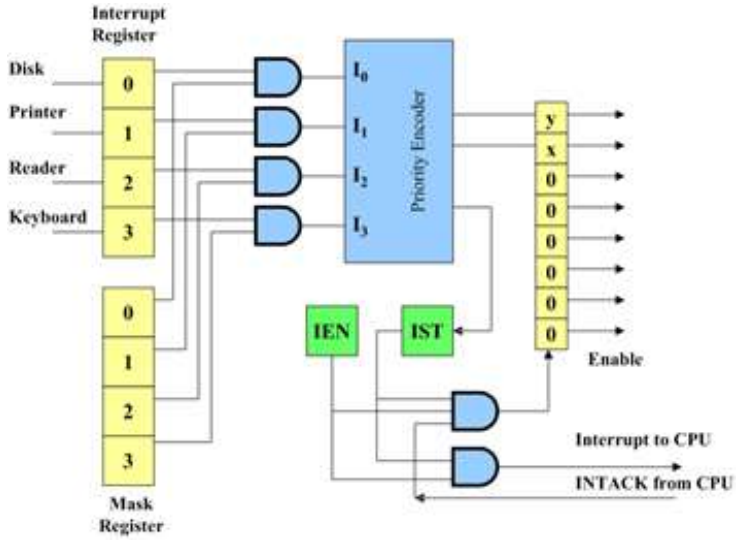


Figure 14: Priority interrupt hardware

- 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 2.

TABLE 2: Priority Encoder Truth Table

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

- The output of the priority encoder is used to form part of the vector address for each interrupt source.

5.5 DIRECT MEMORY ACCESS (DMA)

- The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU.
- Removing the CPU from the path and letting the peripheral device manage the memory buses directly is facilitated by DMA.
- A DMA controller takes over the buses to manage the transfer directly between the I/O device and memory.
- The CPU can be allowed to perform its usual operations without any frequent interruptions from I/O devices by using special control signals.
- These control signals (bus request) from the DMA controller requests the CPU to relinquish control of the buses and place them in high impedance state.

- The CPU activates the (bus grant) output to inform the external DMA that the buses are in high impedance state.
- The DMA can now take control of the buses to conduct memory transfers without processors intervention.
- Upon completion, the DMA disables the bus request line following which CPU disables the bus grant, takes control of the buses and returns to its normal operation.
- The DMA can directly communicate with memory after the bus grant is issued by the CPU. Data transfer can be made by transferring data in burst or an alternative technique called cycle stealing.
- Burst transfer is useful while communicating with fast devices and transfer can be done in a continues burst while the DMA controller is the master of the buses.
- In cycle stealing DMA controller transfers one data word at a time, after which it must return control of the buses to the CPU.

5.5.1 DMA Controller

- The DMA controller needs the usual circuits of an interface to communicate with CPU and I/O device. In addition it needs a word count register, and a set of address lines.
- The address register and address lines are used for direct communication with memory. The word count register specifies the number of words that must be transferred. The data transfer may be done directly between the device and memory under control of DMA.

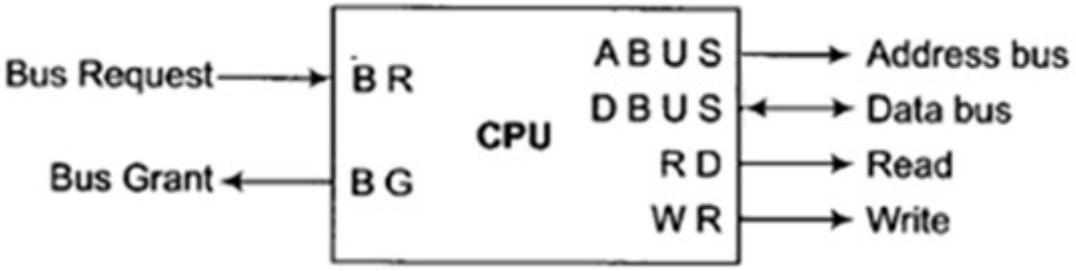


Figure 15: CPU BUS Signals for DMA Transfer

- The unit communicates with CPU via the data bus and control lines. The registers in DMA are selected by the CPU through the address bus by enabling the DS and RS inputs.
- The RD(read) and WR(write) inputs are bidirectional. When BG=1, the CPU has relinquished the buses and the DMA can communicate directly with the memory by specifying an address in the address bus and activating the RD or WR control.
- The DMA communicates with the external peripheral through the request and acknowledge lines by using a prescribed handshaking procedure.
- The DMA controller has 3 registers: an address register, a word count register, and a control register, and a control register.

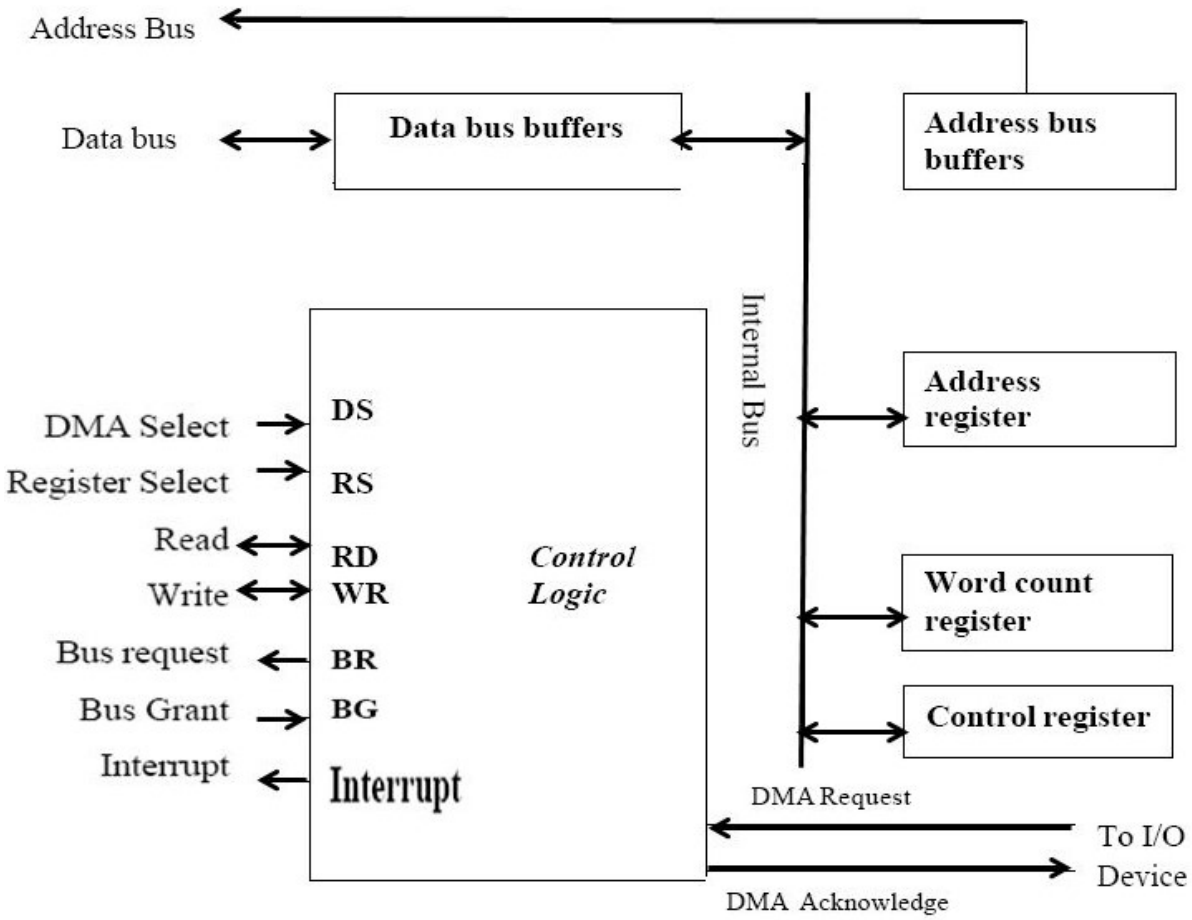


Figure 16: Block diagram for DMA Controller

- The address bits from the address register go through bus buffers into the address bus. The address register is incremented after each word that is transferred to memory.
- The word count register holds the number of words to be transferred and is decremented after each transfer.
- The control register specifies the mode of transfer. The CPU can read from or write into the DMA registers via the data bus.
- The DMA is first initialized by the CPU. After that, the DMA starts and continues to transfer data between memory and peripheral unit until an entire block is transferred.
- The initialization process is essentially a program consisting of I/O instructions that include the address for selecting particular DMA registers.
- The CPU initializes the DMA by sending the following information through the data bus:

i. The starting address of the memory block where data are available or where data are to be stored
ii. The word count, which is the number of words in the memory block
iii. Control to specify the mode of transfer such as read and write
iv. A control to start the DMA transfer

5.5.2 DMA Transfer

- The CPU communicates with the DMA through the address and data buses as with any interface unit.
- The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus.
- When the peripheral device sends a DMA request, the DMA controller activates the BR line, requesting the CPU to relinquish the system bus.
- The CPU responds by sending a BG signal informing the DMA that its buses are disabled. The DMA then puts the address in address register on to the address lines initiates the RD or WR signal, and sends DMA acknowledge to the peripheral device.
- When the peripheral device receives a DMA acknowledge, it puts a word in the data bus or receives a word from the data bus.
- The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while CPU is momentarily disabled.

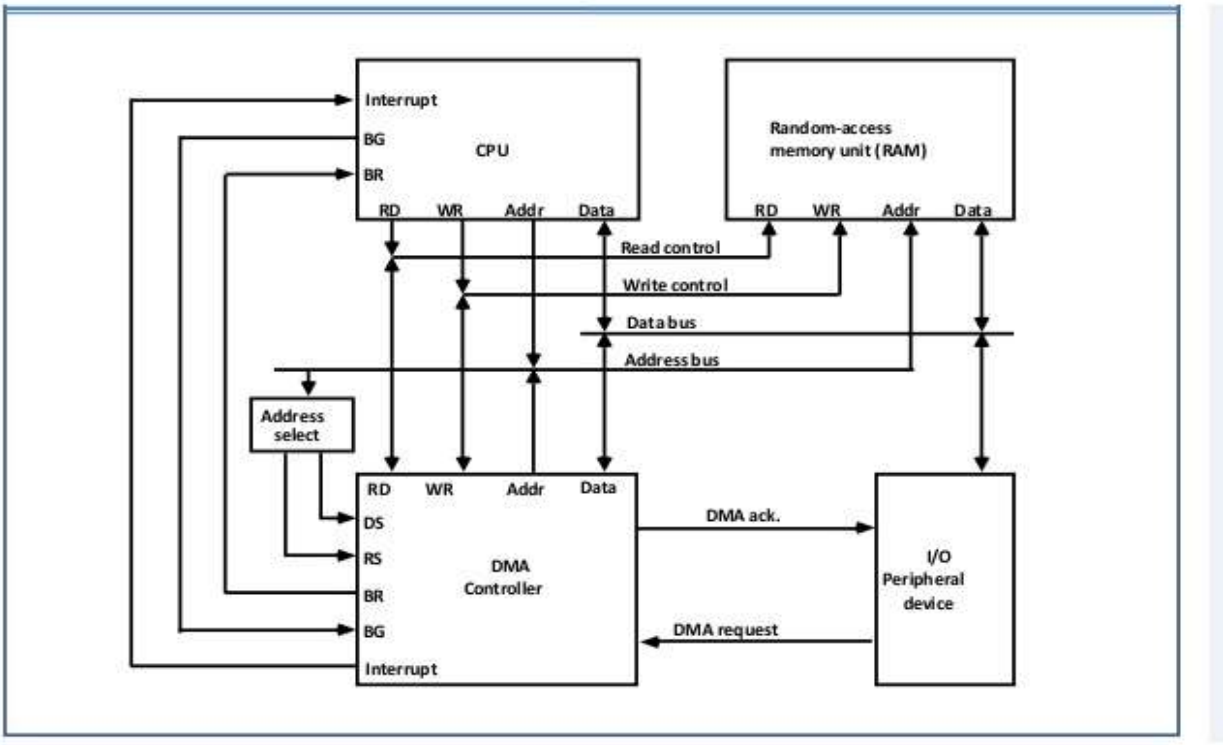


Figure 17: DMA Transfer in a computer system

- For each word that is transferred, the DMA increments its address registers and decrements its word count register.
- If the word count register does not reach zero, the DMA checks the request line coming from the peripheral. When the peripheral requests a transfer, the DMA requests the bus again.
- If the word count register reaches zero, the DMA stops any further transfer and removes its bus request.
- It also informs the CPU of the termination by means of an interrupt. When the CPU responds to the interrupt, it reads the content of the word count register to check for successful transmission.
- DMA transfer is useful in fast transfer of information between magnetic disks and memory.

5.6 INPUT-OUTPUT PROCESSOR (IOP)

- An Input-Output Processor (IOP) may be classified as a processor with direct memory access capability that communicates with all I/O devices.
- Here the computer system can be divided into a memory, and a number of processors comprised of CPU and one or more IOPs.
- The IOP is similar to a CPU except that it is designed to handle the details of the I/O processing.
- The IOP can perform arithmetic, logic, branching and code translation.

- The block diagram of a computer with two processors is shown in Fig. 18. The memory unit occupies a central position and can communicate with each processor by means of DMA.

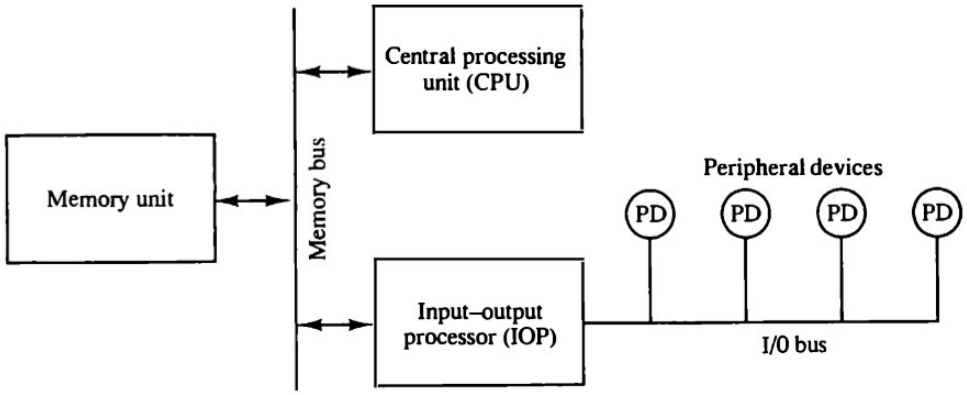


Figure 18: Block diagram of a computer with I/O processor

- The CPU is responsible for processing data needed in computational task.
- The IOP provides a path for transformation of data between various peripheral devices and the memory unit.
- The data formats of peripheral devices differ from memory and CPU data formats. The IOP must structure data words from many different sources.
- The communication between the IOP and the devices attached to it is similar to the program control method of transfer.
- Communication with the memory is similar to the direct memory access method.
- 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.
- Instructions that are read from memory by an IOP are sometimes called commands, to distinguish them from instructions that are read by the CPU.

5.6.1 CPU-IOP Communication

- The communication between CPU and IOP may take different forms, depending on the particular computer considered.
- In most cases the memory unit acts as a message center where each processor leaves information for the other.
- The sequence of operations may be carried out as shown in the flowchart of Fig. 19.

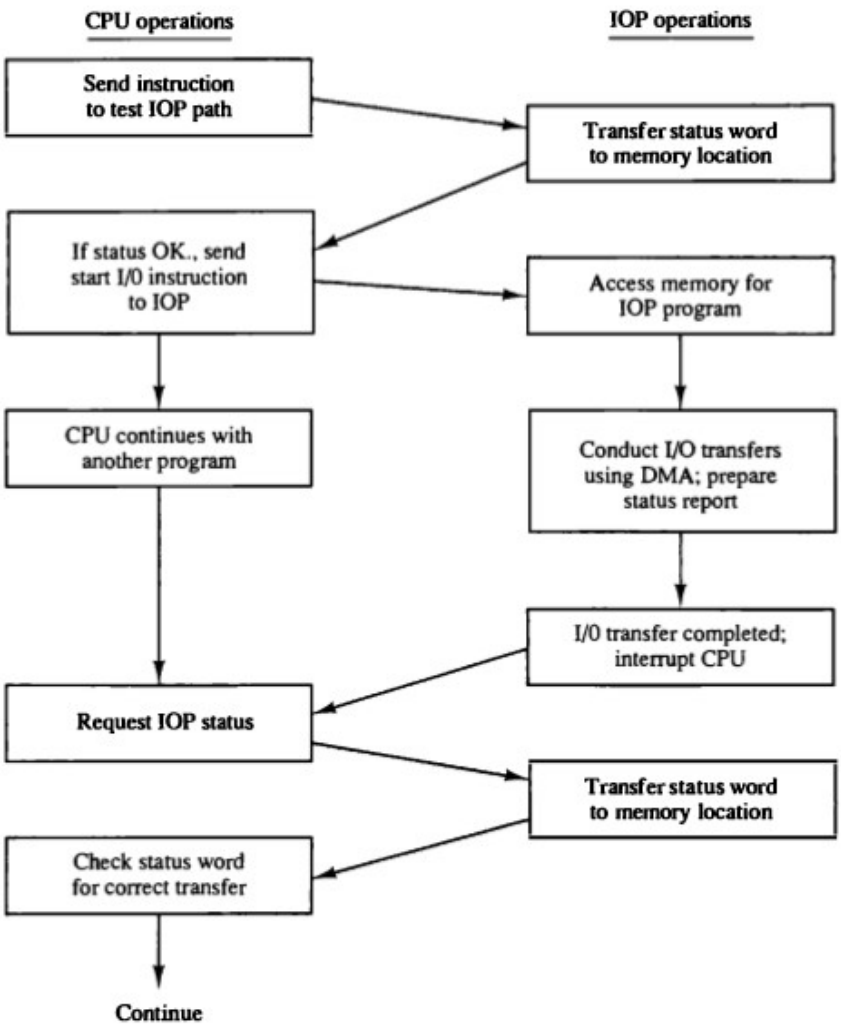


Figure 19: CPU-IOP communication