# UNIT-III

**MEMORY AND INPUT/OUTPUT ORGANIZATION**

## Memory Organization: Memory Hierarchy Main Memory Auxiliary Memory Associative Memory Cache Memory



## UNIT - IV

**Input/output Organization: Input-Output Interface Asynchronous Data Transfer Modes of Transfer**



**Priority Interrupt**

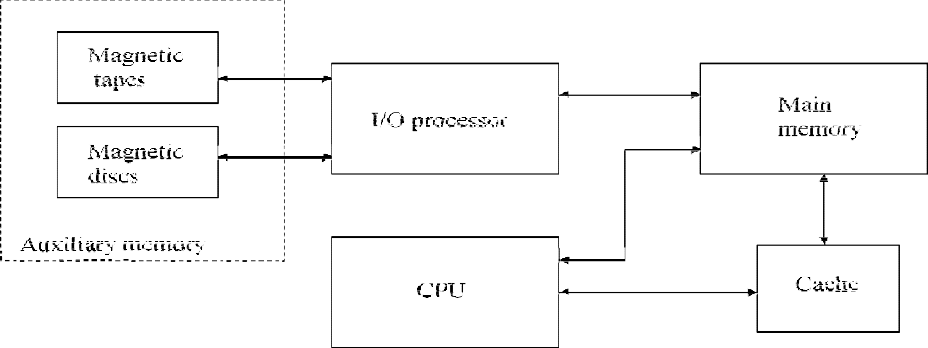


**Direct Memory Access (DMA).**



**MEMORY HIERARCHY**

* The memory unit is an essential component in any digital computer since it is needed for storing programs and data. A very small computer with a limited application may be able to fulfill its intended task without the need of additional storage capacity.
* The memory unit that communicates directly with the CPU is called the **main memory**. Devices that provide backup storage are called **auxiliary memory**. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information. Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.
* The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.



Memory hierarchy in computer system

* The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices through an I/O processor.
* When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.
* A special very-high speed memory called a **cache** is sometimes used to increase the speed of processing by making current programs and data available to the CPU at a rapid rate. The cache memory is employed in computer systems to compensate for the speed differential between main memory access time and processor logic.
* The overall goal of using a memory hierarchy is to obtain the highest-possible average access speed while minimizing the total cost of the entire memory system.
* Auxiliary and cache memories are used for different purposes. The cache holds those parts of the program and data that are most heavily used, while the auxiliary memory holds those parts that are not presently used by the CPU. Moreover, the CPU has direct access to both cache and main memory but not to auxiliary memory. The transfer from auxiliary to main memory is usually done by means of direct memory access of large blocks of data. The typical access time ratio between cache and main memory is about 1 to 7. For example, a typical cache memory may have an access time of 100ns, while main memory access time may be 700ns. Auxiliary memory average access time is usually 1000 times that of main memory. Block size in auxiliary memory typically ranges from256 to 2048 words, while cache block size is typically from 1 to 16 words.
* Many operating systems are designed to enable the CPU to process a number of independent programs concurrently. This concept, called **multiprogramming**, refers to the existence of two or more programs in different parts of the memory hierarchy at the same time.
* In a multiprogramming system, when one program is waiting for input or output transfer, there is another program ready to utilize the CPU.

# ASSOCIATIVE MEMORY

* The number of accesses to memory depends on the location of the item and the efficiency of the search algorithm. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.
* The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address.
* A memory unit accessed by content is called an ***associative memory or content addressable memory (CAM).***
* When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading.
* An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, associative memories are used in applications where the search time is very critical and must be very short.

## HARDWARE ORGANIZATION

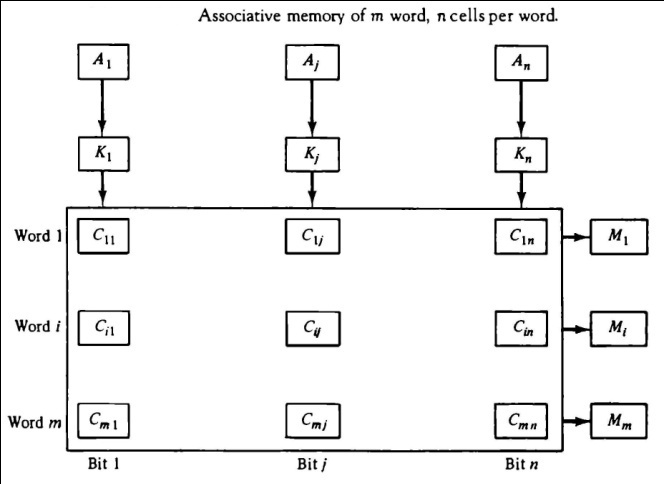
* The block diagram of an associative memory is shown in Fig.
* It consists of a memory array and logic for *m* words with *n* bits per word. The argument register A and key register K each have n bits, one for each bit of a word. The match register M has m bits, one for each memory word.
* Each word in memory is compared in parallel with the content of the argument register. The words that match the bits of the argument register set a corresponding bit in the match register.
* After the matching process, those bits in the match register that have been set indicate the fact that their corresponding words have been matched.
* Reading is accomplished by a sequential access to memory for those words whose corresponding bits in the match register have been set.
* The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word if the key register contains all 1’s. Otherwise, only those bits in the argument that have 1’s in their corresponding position of the key register are compared.
* To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has 1’s in these positions.

A 101 111100

K 111 000000

Word 1 100 111100 no match

Word 2 101 000001 match

* The relation between the memory array and external registers in an associative memory is shown in Fig.
* The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. Thus cell Cij is the cell for bit j in word i.
* A bit Aj in the argument register is compared with all the bits in column j of the array provided that Kj

= 1. This is done for all columns j = 1, 2,…,n.

* If a match occurs between all the unmasked bits of the argument and the bits in word i, the corresponding bit Mi in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0

## READ OPERATION

* If more than one word in memory matches the unmasked argument field, all the matched words will have 1’s in the corresponding bit position of

the match register. It is then necessary to scan the bits of the match register one at a time. The matched words are read in sequence by applying a read signal to each word line whose corresponding Mi bit is a 1.

* In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field. By connecting output Mi directly to the read line in the same word position (instead of the M register), the content of the matched word will be presented automatically at the output lines and no specialread command signal is needed.

## WRITE OPERATION

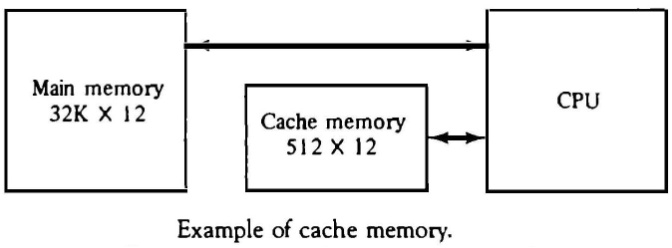
* An associative memory must have a write capability for storing the information to be searched.
* Writing in an associative memory can take different forms, depending on the application. If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence. This will make the device a random-access memory for writing and a content addressable memory for reading. The advantage here is that the address for input can be decoded as in a random-access memory. Thus instead of having m address lines, one for each word in memory, the number of address lines can be reduced by the decoder to d lines, where m = 2d.
* If unwanted words have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active and inactive words. This register, sometimes called a *tag register*.
* For every active word stored in memory, the corresponding bit in the tag register is set to 1. A word is deleted from memory by clearing its tag bit to 0.
* Words are stored in memory by scanning the tag register until the first 0 bit is encountered. This gives the first available inactive word and a position for writing a new word. After the new word is stored in memory it is made active by setting its tag bit to 1. An unwanted word when deleted from memory can be cleared to all 0’s if this value is used to specify an empty location.

## CACHE MEMORY

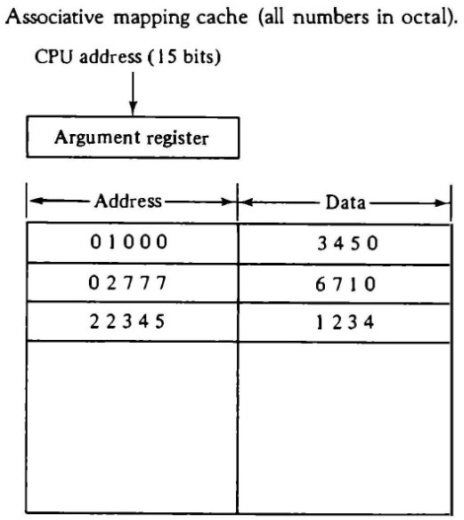
* Locality of Reference: The references to memory at any given time interval tends to be confined within a localized area.
* When a program loop is executed, the CPU repeatedly refers to the set of instructions in memory that constitute the loop.
* Every time a given subroutine is called, its set of instructions is fetched from memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions.
* Iterative procedures refer to common memory locations and array of numbers are confined within a local portion of memory
* If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory. The cache is the fastest component in the memory hierarchy and approaches the speed of CPU components.
* When the CPU needs to access memory, the cache is examined. If the word is found in the cache, it is read from the fast memory. If the word addressed by the CPU is not found in the cache, the main memory is accessed to read the word. A block of words containing the one just accessed is then transferred from main memory to cache memory
* The performance of cache memory is frequently measured in terms of a quantity called **hit ratio**. When the CPU refers to memory and finds the word in cache, it is said to produce a hit. If the word is not found in cache, it is in main memory and it counts as a miss. *The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio.*
* The average memory access time of a computer system can be improved considerably by use of a cache*.*
* The transformation of data from main memory to cache memory is referred to as a mapping process. Three types of mapping procedures are :

1. Associative mapping
2. Direct mapping
3. Set-associative mapping.

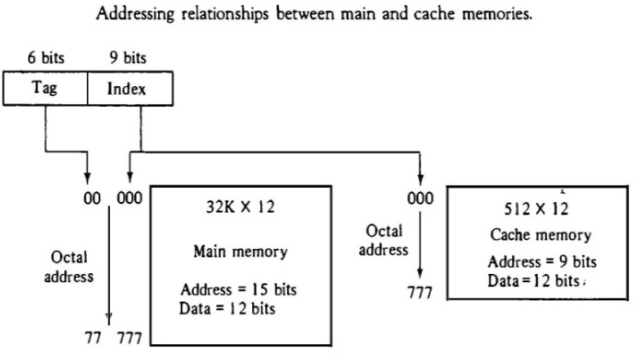
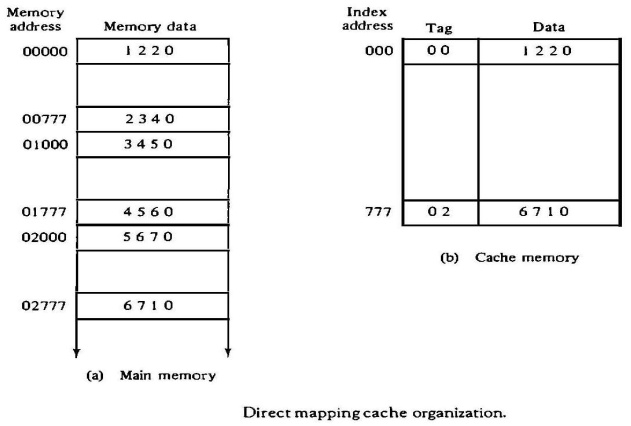
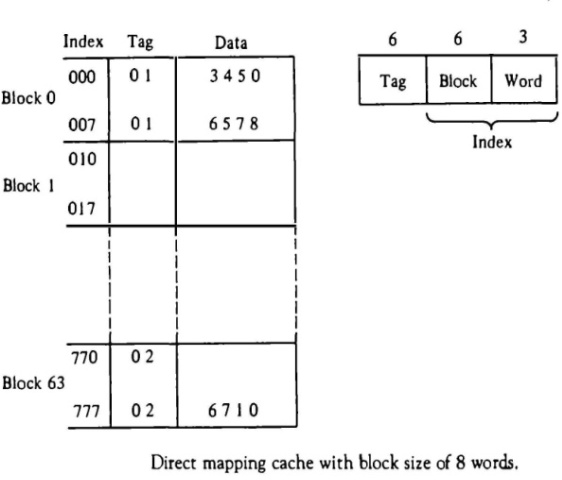
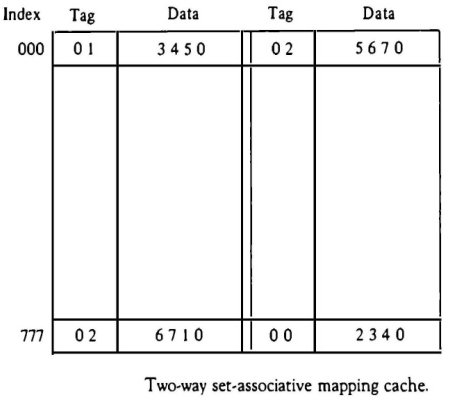
* Consider the following memory organization:



## ASSOCIATIVE MAPPING

* The faster and most flexible cache organization use an associative memory. The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.
* A CPU address of 15 bits is placed in the argument register and the associative memory is searched for a matching address. If the address is found, the corresponding 12-bit data is read and sent to the CPU.
* If no match occurs, the main memory is accessed for the word. The address-data pair is then transferred to the associative cache memory. If the cache is full, an address−data pair must be displaced to make room for a pair that is needed and not presently in the cache.
* The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache. A simple procedure is to replace cells of the cache in round- robin order whenever a new word is requested from main memory. This constitutes a first-in first-out (FIFO) replacement policy.

## DIRECT MAPPING

* Associative memories are expensive compared to random-access memories because of the added logic associated with each cell.
* Direct mapping uses RAM instead of CAM.
* The n-bit memory address is divided into two fields: k bits for the index field and n-k bits for the tag field. The direct mapping cache organization uses the n-bit address to access the main memory and the k-bit index to access the cache.
* The internal organization of the words in the cache memory is as shown in Fig
* Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache.
* The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value.
* The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly.
* Suppose that the CPU now wants to access the word at address 02000. The index address is 000, so it is sued to access the cache. The two tags are then compared. The cache tag is 00 but the address tag is 02, which does not produce a match. Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU. The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.
* The direct-mapping uses a block size of one word. The same organization but using a block size of 8 words is shown in Fig.
* The index field is now divided into two parts: the block field and the word field. The tag field stored within the cache is common to all eight words of the same block.
* Every time a miss occurs, an entire block of eight words must be transferred from main memory to cache memory. Although this takes extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs. **SET-ASSOCIATIVE MAPPING**
* Set-associative mapping is an improvement over the direct-mapping organization in that each word of cache can store two or more words of memory under the same index address.
* Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.
* Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is 2(6 + 12) = 36 bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512 × 36. It can accommodate 1024
* The words stored at addresses 01000 and 02000 of main memory are stored in cache memory at index address 000. Similarly, the words at addresses 02777 and 00777 are stored in cache at index address 777.
* When the CPU generates a memory request, the index value of the address is used to access the cache. The tag field of the CPU address is then compared with both tags in the cache to determine if a match occurs.
* The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache.
* When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value. The most common replacement algorithms used are: random replacement, first-in first out (FIFO), and least recently used (LRU).

WRITING INTO CACHE

* An important aspect of cache organization is concerned with memory write requests. If the operation is a write, there are two ways that the system can proceed.
* The simplest and most commonly used procedure is to up data main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the ***write-through method***. This method has the advantage that main memory always contains the same data as the cache,. This characteristic is important in systems with direct memory access transfers.
* The second procedure is called the ***write-back method***. In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the words are removed from the cache it is copied into main memory. The reason for the write-back method is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache. It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory. CACHE INITIALIZATION
* The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, built in effect it contains some non-valid data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.
* The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again. The introduction of the valid bit means that a word in cache is initialization condition has the effect of forcing misses from the cache until it fills with valid data.

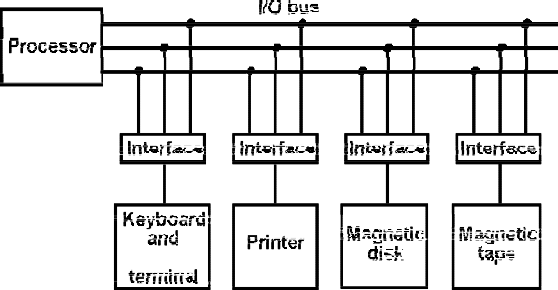
**UNIT IV**

**INPUT OUTPUT ORGANIZATION**

# 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 need.
  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.

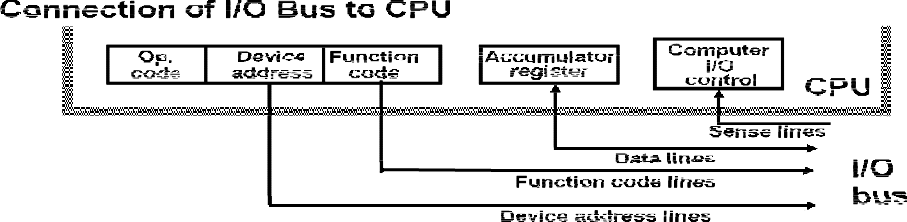
## I/O BUS AND INTERFACE MODULES

* A typical communication link between the processor and several peripherals is shown in Fig.
* 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, interprets them for the peripheral, and provides signals for the peripheral controller. It also synchronizes the data

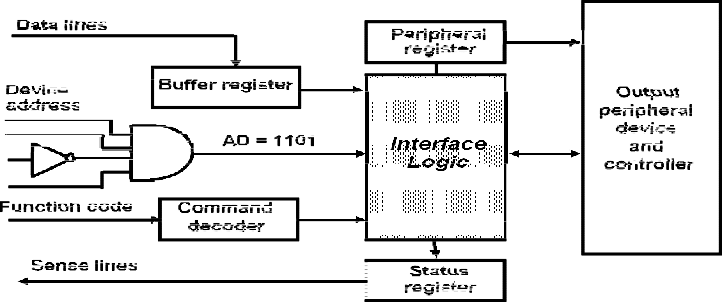
flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller that operates the particular electromechanical device.

* 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 their interface.





* At the same time the processor provides a function code in the control lines.



* There are four types of commands that an interface may receive. They are classified as control, status, data output, and data input.
* 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.
* A data output command causes the interface to respond by transferring data from the bus into one of its registers.
* The data input command is the opposite of the data output. In this case the interface receives an item of data from the peripheral and places it in its buffer register.

## 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.

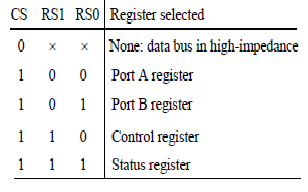
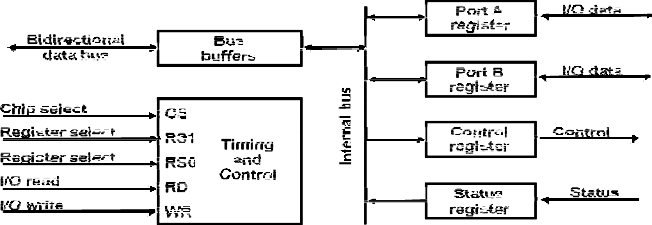
In the first method, the computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This is done in computers that provide a separate I/O processor (IOP) in addition to the central processing unit (CPU). The memory communicates with both the CPU 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

## 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.
* 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 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.
* In a memory-mapped I/O organization there is 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

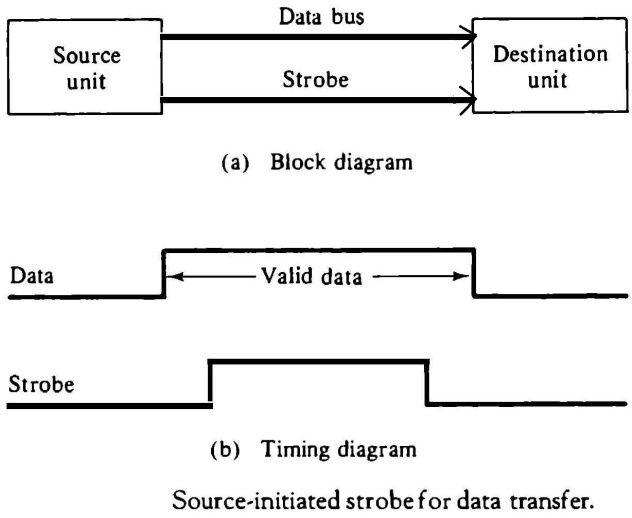
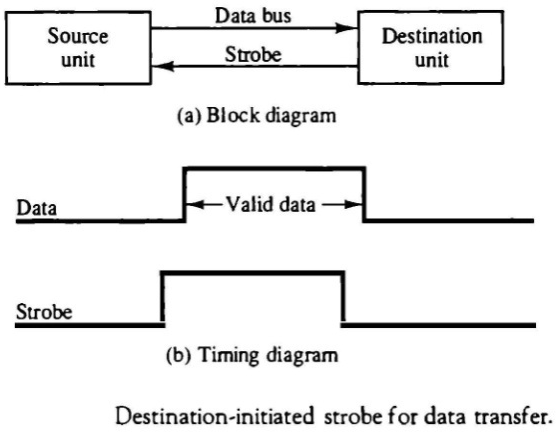


* It consists of two data registers called ports, a control register, a status register, bus buffers, and timing and control circuits. The interface communicates with the CPU through the data bus.
* The chip select and register select inputs determine the address assigned to the interface. The I/O read 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..
* 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 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.
* The interface registers communicate with the CPU through the bidirectional data bus.
* The address bus selects the interface unit through the chip select and the two register select inputs. A circuit must be provided externally (usually, a decoder) to detect the address assigned to the interface registers. This circuit enables the chip select (CS) input when the interface is selected by the address bus. The two register select inputs RS1 and RS0 are usually connected to the two least significant lines of the lines 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 I/O read signal is enabled. The CPU transfers binary information into the selected register via the data bus when the I/O write input is enabled.

# ASYNCHRONOUS DATA TRANSFER

* The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator.
* 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.
* One way of achieving this is by means of a **strobe pulse** supplied by one of the units to indicate to the other unit when the transfer has to occur. Another method commonly used is to accompany each data item being transferred with a control signal that indicates the presence of data in the bus. The unit receiving the data item responds with another control signal to acknowledge receipt of the data. This type of agreement between two independent units is referred to as **handshaking.**

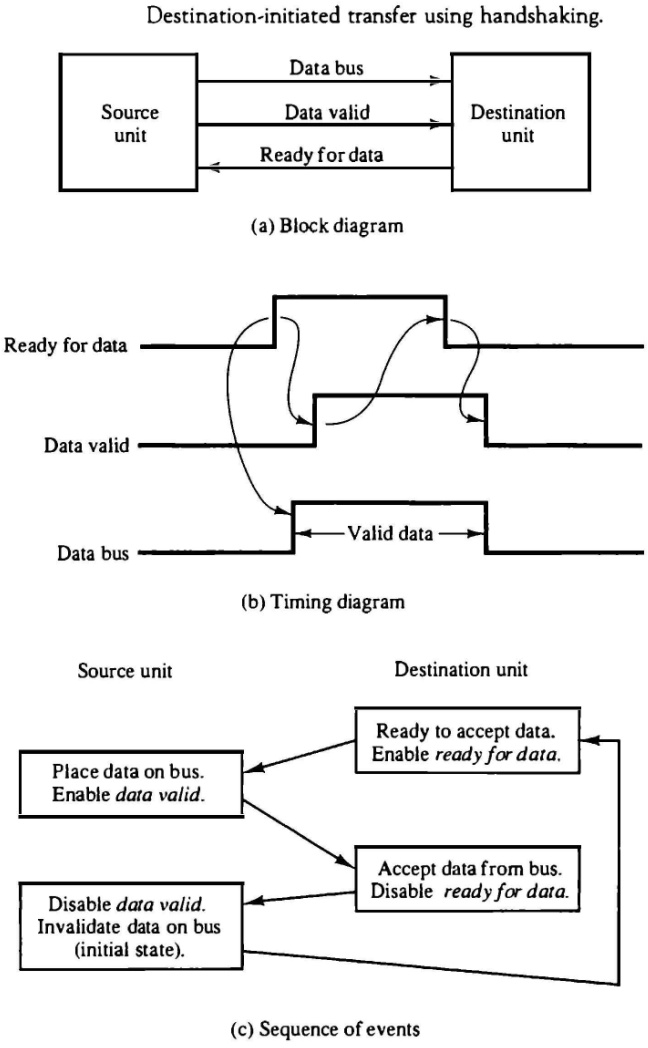
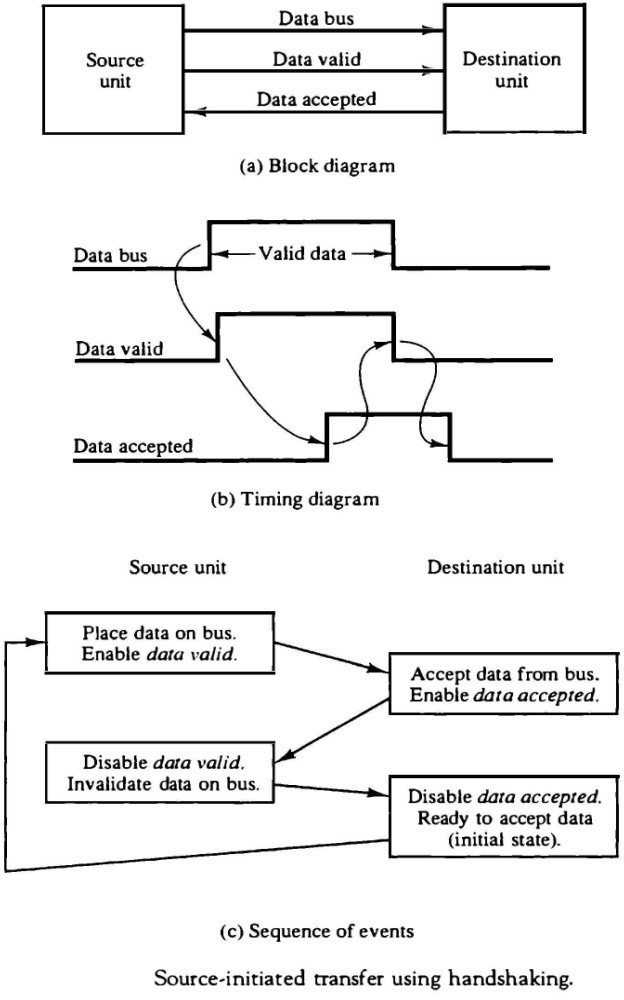
## STROBE CONTROL

* The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either the source or the destination unit.
* The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus.
* As shown in the timing diagram the source unit first places the data on the data bus. After a brief delay to ensure that the data settle to a steady value, the source activates the strobe pulse.
* The information on the data bus and the strobe signal remain in the active state for a sufficient time period to allow the destination unit to receive the data. Often, the destination unit uses the falling edge of the strobe pulse to transfer the contents of the data bus into one of its internal registers.
* The following Figure 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 transfer of data between the CPU and an interface unit is similar to the strobe transfer. Data transfer between an interface and an I/O device is commonly controlled by a set of handshaking lines

## 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 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 valued 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.
* 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. In the destination- initiated transfer the source does not place data on the bus until after it receives the ready for data signal from the destination unit.
* The handshaking scheme provides a high degree of flexibility and reality 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



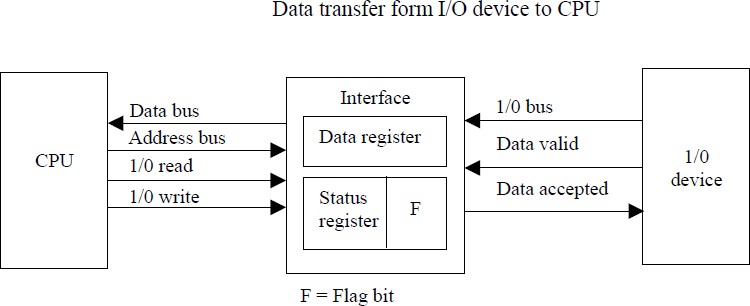
**MODES OF TRANSFER**

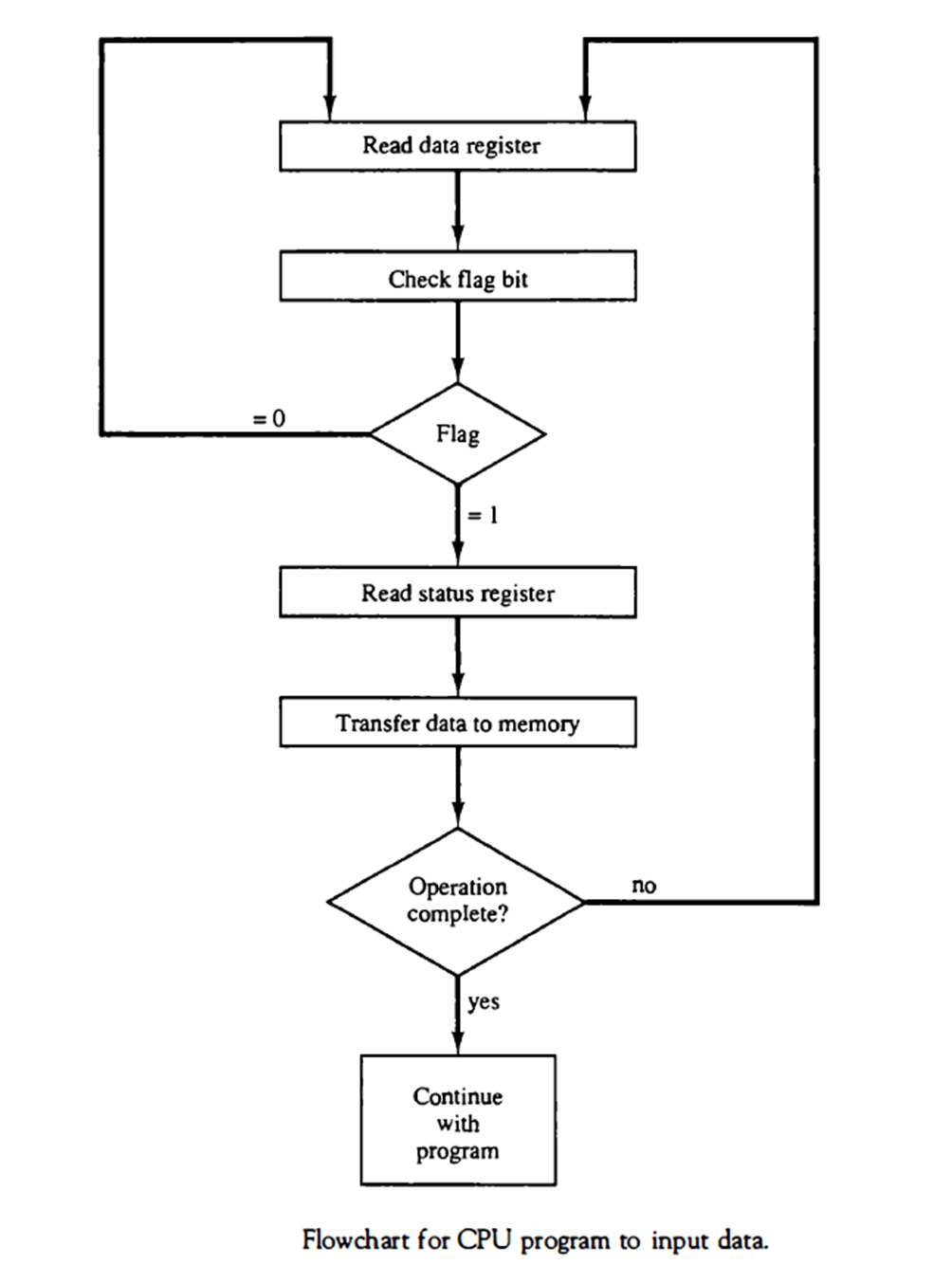
* Binary information received from an external device is usually stored in memory for later processing. Information transferred from the central computer into an external device originates in the memory unit. The CPU merely executes the I/O instructions and may accept the data temporarily, but the ultimate 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; other 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 CU 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 DMPA 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.
* 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.
* 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.

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 form an I/O device and store them in a memory buffer.
* 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.

## 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 tone unit to another. In principle, there are two methods for accomplishing this. One is called vectored interrupt and the other, no 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.

## PRIORITY INTERRUPT

* Data transfer between the CPU and an I/O device is initiated by the CPU. 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.
* Numbers of I/O devices are attached to the computer; several sources will request service simultaneously. The first task of the interrupt system is to identify the source of the interrupt and decide which device to service first
* A priority interrupts is a system to determine which interrupt is to be served first when two or more requests are made simultaneously. Also determines which interrupts are permitted to interrupt the computer while another is being serviced. Higher priority interrupts can make requests while servicing a lower priority interrupt
* Establishing the priority of simultaneous interrupts can be done by software or hardware.
* Priority Interrupt by Software(Polling)
* Priority is established by the order of polling the devices(interrupt sources)
* Flexible since it is established by software
* Low cost since it needs a very little hardware
* Very slow
* Priority Interrupt by Hardware

- Require a priority interrupt manager which accepts all the interrupt requests to determine the highest priority request

* Fast since identification of the highest priority interrupt request is identified by the hardware. Each interrupt source has its own interrupt vector to access directly to its own service routine
* The hardware priority function can be established by either a serial or a parallel connection of interrupt lines. The serial connection is also known as the daisy chaining method.

## DAISY-CHAINING PRIORITY

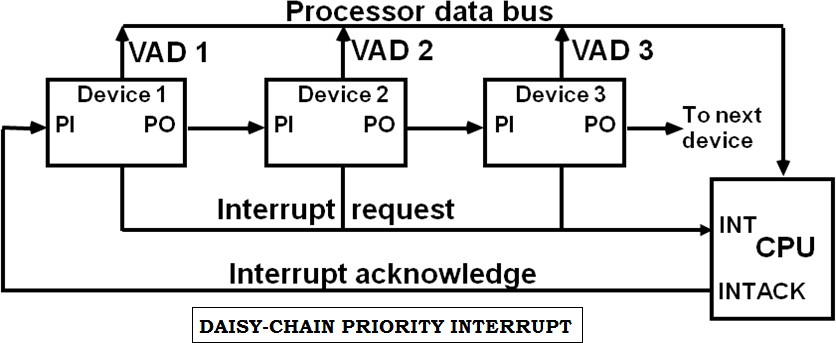
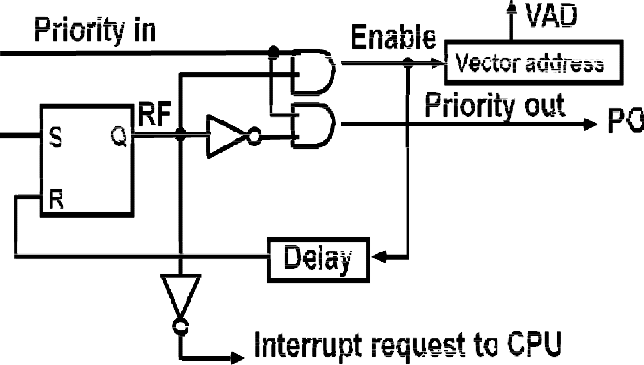
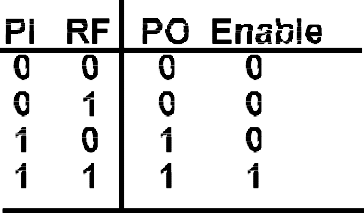
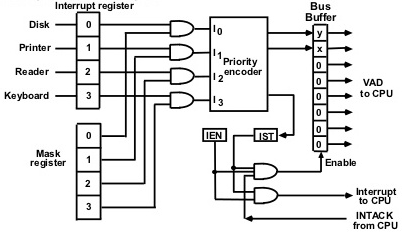
* The daisy-chaining method of establishing priority consists of a serial connection of all devices that request an interrupt. The device with the highest priority is placed in the first position, followed by lower-priority devices up to the device with the lowest priority, which is placed last in the chain.
* 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.
* 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 following figure shows the internal logic that must be included with in each device when connected in the daisy-chaining scheme.

Fig: One stage of the daisy- chain priority arrangement

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

## PARALLEL PRIORITY INTERRUPT

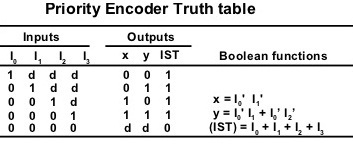
* The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device.
* Priority is established according to the position of the bits in the register. In addition to the interrupt register the circuit may include a mask register whose purpose is to control the status of each interrupt request.
* The mask register can be programmed to disable lower-priority interrupts while a higher- priority device is being serviced. It can also provide a facility that allows a high-priority device to interrupt the CPU while a lower-priority device is being serviced.
* The priority logic for a system of four interrupt sources is shown in Fig.



* It consists of an interrupt register whose individual bits are set by external conditions and cleared by program instructions.
* 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.



## Interrupt Cycle

* The interrupt enable flip-flop IEN can be set or cleared by program instructions.
* When IEN is cleared, the interrupt request coming from IST is neglected by the CPU.
* The program-controlled IEN bit allows the programmer to choose whether to use the interrupt facility. If an instruction to clear IEN has been inserted in the program, it means that the user does not want his program to be interrupted. An instruction to set IEN indicates that the interrupt facility will be used while the current program is running.
* Most computers include internal hardware that clears IEN to 0 every time an interrupt is acknowledged by the processor
* At the end of each instruction cycle the CPU checks IEN and the interrupt signal from IST. If either is equal to 0, control continues with the next instruction.
* If both IEN and IST are equal to 1, the CPU goes to an interrupt cycle.
* During the interrupt cycle the CPU performs the following sequence of microoperations: SP  SP-1 Decrement stack pointer

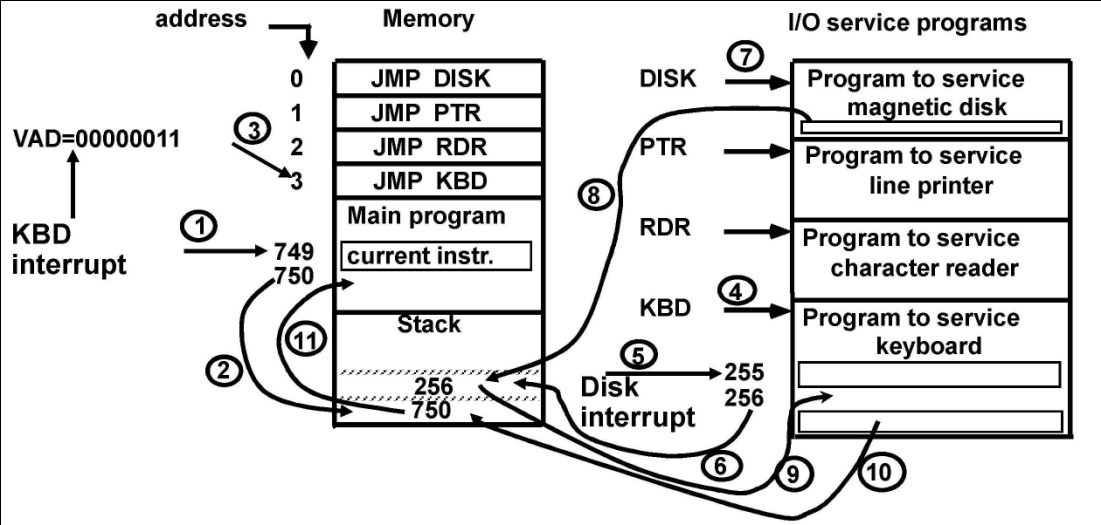
M[SP] PC Push PC into stack

INTACK 1 Enable interrupt acknowledge PC  VAD Transfer vector address to PC

lEN 0 Disable further interrupts Go to fetch next instruction

## Software Routines

* A priority interrupt system is a combination of hardware and software techniques
* The following figure shows the programs that must reside in memory for handling the interrupt system.



## Initial and Final Operations

* Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system

## Initial Sequence

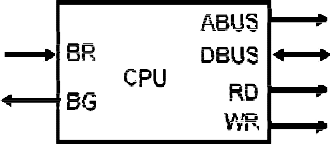
1. Clear lower level Mask reg. bits
2. IST 0
3. Save contents of CPU registers
4. IEN1
5. Go to Interrupt Service Routine

## Final Sequence

1. IEN 0
2. Restore CPU registers
3. Clear the bit in the Interrupt Reg
4. Set lower level Mask reg. bits
5. Restore return address into PC, and IEN 1

* The initial and final operations are referred to as **overhead operations** or **housekeeping chores**. They are not part of the service program proper but are essential for processing interrupts.
* All overhead operations can be implemented by software. This is done by inserting the proper instructions at the beginning and at the end of each service routine. Some of the overhead operations can be done automatically by the hardware

# 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 would improve the speed of transfer. This transfer technique is called **direct memory access (DMA)**.
* During DMA transfer, the CPU is idle and 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 memory.
* The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessors is to disable the buses through special control signals.

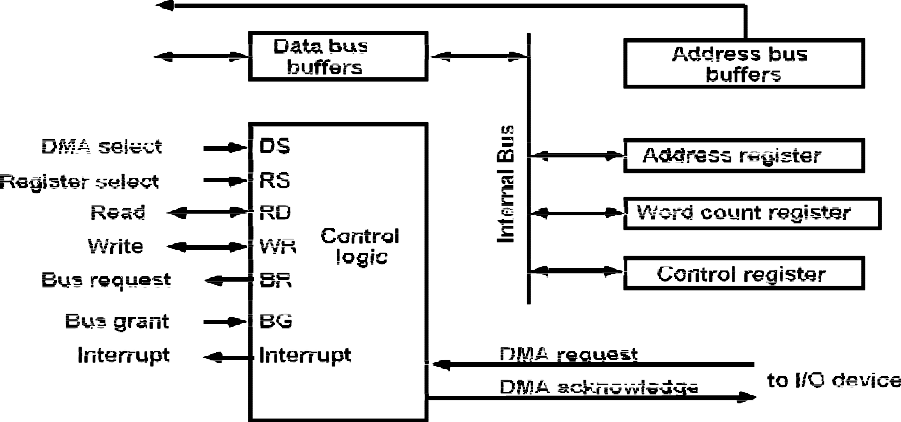


* The bus request (BR) input is used by the DMA controller to request the CPU to cease control of the buses. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, the data bus, and the read and write lines into a high- impedance state behaves like an open circuit, which means that the output is disconnected and does not have a logic significance.
* The CPU activates the Bus grant (BG) output to inform the external DMA that the buses are in the high-impedance state. The DMA that originated the bus request can now take control of the buses to conduct memory transfers without processor intervention. When the DMA terminates the transfer, it disables the bus request line. The CPU disables the bus grant, takes control of the buses, and returns to its normal operation.
* When the DMA takes control of the bus system, it communicates directly with the memory. The transfer can be made in several ways. In DMA **burst transfer**, a block sequence consisting of a number of memory words is transferred in a continuous burst while the DMA controller is master of the memory buses. 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.
* An alternative technique called **cycle stealing** allows the DMA controller to transfer one data word at a time after which it must return control of the buses to the CPU. The CPU merely delays its operation for one memory cycle to allow the direct memory I/O transfer to “steal” one memory cycle.

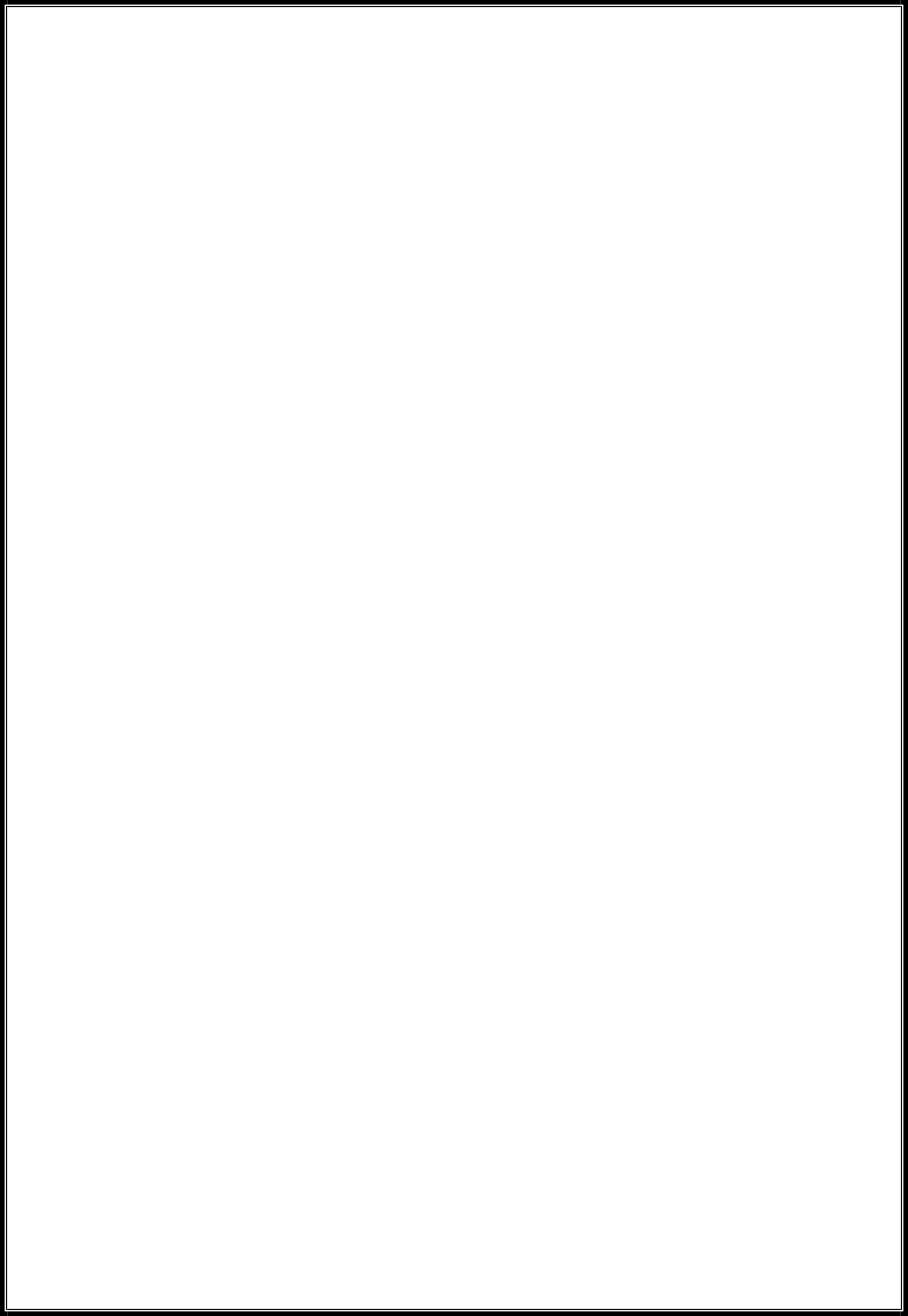
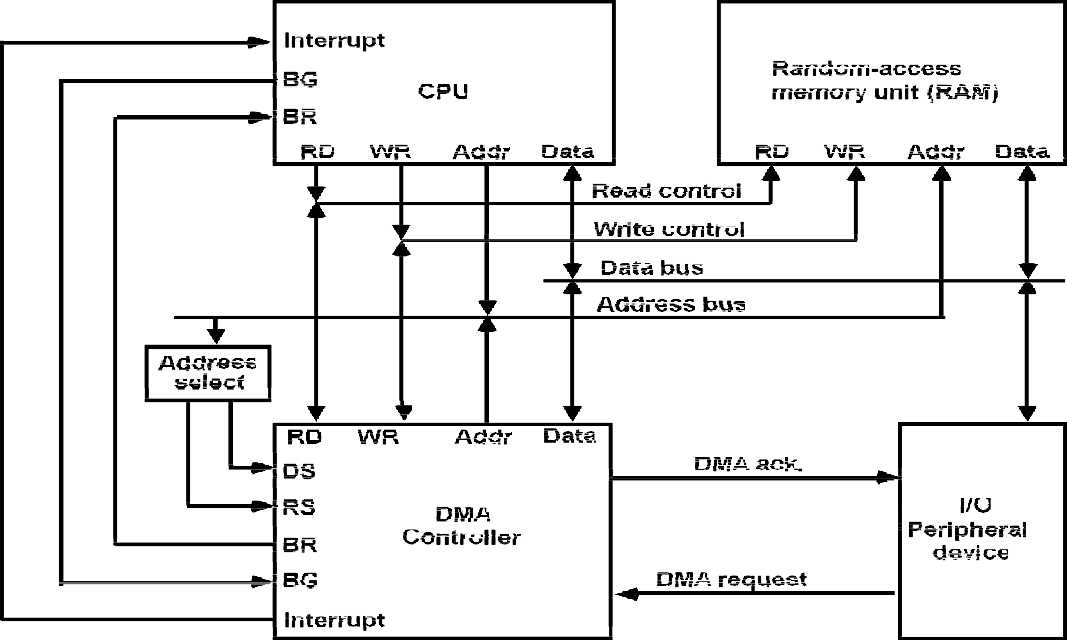
## DMA CONTROLLER

* The following figure shows the block diagram of a typical DMA controller





* The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (register select) inputs. The RD (read) and WR (write) inputs are bidirectional.
* When the BG (bus grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers.
* When BG = 1, the CPU has relinquished(ceased) 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 three registers: an address register, a word count register, and a control register. The address register contains an address to specify the desired location in memory. The address bits 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 is incremented after each word that is transferred to memory. The word count register holds the number of words to be transferred. This register is decremented by one after each word transfer and internally tested for zero.
* The control register specifies the mode of transfer. All registers in the DMA 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.
* 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:

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

* The starting address is stored in the address register. The word count is stored in the word count register, and the control information in the control register.
* Once the DMA is initialized, the CPU stops communicating with the DMA unless it receives an interrupt signal or if it wants to check how many words have been transferred.

## DMA Transfer

* The position of the DMA controller among the other components in a computer system is illustrated in following fig.
* 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. Once the DMA receives the start control command, it can start the transfer between the peripheral device and the memory.
* When the peripheral device sends a DMA request, the DMA controller activates the BR line, informing the CPU to relinquish the buses. The CPU responds with its BG line, informing the DMA that its buses are disabled.
* The DMA then puts the current value of its address register into the address bus, initiates the RD or WR signal, and sends a DMA acknowledge to the peripheral device.
* Note that the RD and WR lines in the DMA controller are bidirectional. The direction of transfer depends on the status of the BG line. When BG = 0, the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When BG = 1, the RD and WR and output lines from the DMA controller to the random-access memory to specify the read or write operation for the data.
* When the peripheral device receives a DMA acknowledge, it puts a word in the data bus (for write) or receives a word from the data bus (for read). Thus the DMA controls the read or write operations and supplies the address for the memory.
* The peripheral unit can then communicate with memory through the data bus for direct transfer between the two units while the CPU is momentarily disabled.
* For each word that is transferred, the DMA increments its address register and decrements its word count register. If the word count does not reach zero, the DMA checks the request line coming from the peripheral.
* For a high-speed device, the line will be active as soon as the previous transfer is completed. A second transfer is then initiated, and the process continues until the entire block is transferred.
* If the peripheral speed is slower, the DMA request line may come somewhat later. In this case the DMA disables the bus request line so that the CPU can continue to execute its program. When the peripheral requests a transfer, the DMA requests the buses 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. The zero value of this register indicates that all the words were transferred successfully. The CPU can read this register at any time to check the number of words already transferred.
* A DMA controller may have more than one channel. In this case, each channel has a request and acknowledges pair of control signals which are connected to separate peripheral devices. Each channel also has its own address register and word count register within the DMA controller.
* A priority among the channels may be established so that channels with high priority are serviced before channels with lower priority.
* DMA transfer is very useful in many applications.
* It is used for fast transfer of information between magnetic disks and memory.
* It is also useful for updating the display in an interactive terminal.