# UNIT-V

**Syllabus:**

**The Memory System:** Memory Hierarchy, Main memory, Auxiliary memory, Associative memory, Cache memory, Virtual memory.

**Input-Output Organization:** Peripheral Devices, Input-Output Interface, Asynchronous data transfer, modes of transfer, priority interrupts, direct memory access, Introduction to Multi Processors: Characteristics and Interconnection structures, Introduction to pipelining.

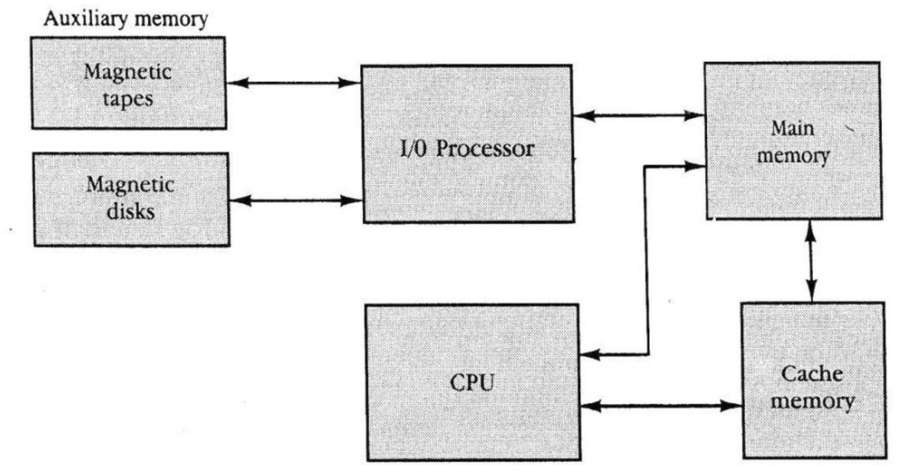
## Memory Hierarchy:

The memory unit is an essential component in any digital computer since it is needed for storing **programs and data**. Most general–purpose computers would run more efficiently if they were equipped with additional storage beyond the capacity of the main memory. There is just not enough space in one memory unit to accommodate all the programs used in typical computer. 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.

Figure 6.1 illustrates the components in a typical memory hierarchy. At the bottom of the hierarchy are the relatively slow magnetic tapes used to store removable files. Next are the magnetic disks used as backup storage. The main memory occupies a central position by being able to communicate directly with the CPU and with auxiliary memory devices throughan 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. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.

Figure 6.1 Memory Hierarchy in a computer system



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. CPU logic is usually faster than main memory access time, with the result that processing speed is limited primarily by the speed of main memory. The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently need in the present calculations.

While the I/O processor manages data transfers between auxiliary memory and main memory, the cache organization is concerned with the transfer of information between main memory and CPU. As the storage capacity of the memory increases, the cost per bit for storing binary information decreases and the access time of the memory becomes longer. The auxiliary memory has a large storage capacity, is relatively inexpensive, but has low access speed compared to main memory. The cache memory is very small, relatively expensive, and has very high access speed. Thus as the memory access speed increases, so does its relative cost.

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 „pp done by means of direct memory access of large blocks of data.

CPU process a multiprogramming 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 this way it is possible to keep all parts of the computer busy by working with several programs in sequence. In a multiprogramming system, when one program is waiting for input or output transfer, there is another program ready to utilize the CPU.

Computer programs are sometimes too long to be accommodated in the total space available in main memory. Moreover, a computer system uses many programs and all the programs cannot reside in main memory at all times. A program with its data normally resides in auxiliary memory. When the program or a segment of the program is to be executed, it is transferred to main memory to be executed by the CPU. It is the task of the operating system to maintain in main memory a portion of this information that is currently active. The part of the computer system that supervises flow of information between auxiliary memory and main memory is called the memory management system.

## Main memory:

The main memory is the central storage unit in a computer system. It is a relatively large and fast memory used to store programs and data. The main memory are memory available in two possible operating modes, static and dynamic. The static R consists essentially of internal flip-flops that store the binary information. The stored information remains valid as long as power is applied to the unit. The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. The capacitors are provided inside the chip by MOS transistors. The stored charge on the capacitors tend to discharge with time and the capacitors must be periodically recharged by refreshing the dynamic memory. Refreshing is done by cycling through the words every few milliseconds to restore the decaying charge. The dynamic RAM offers reduced power consumption and larger storage capacity in a single memory chip. The static RAM is easier to use and has shorter read and write Cycles.

RAM was used to refer to a random-access memory, but now it is used to designate a read/write memory to distinguish it from a read-only memory, although ROM is also random access. RAM is used for storing the bulk of the programs and data that are subject to change. ROM is used for storing programs that are permanently resident in the computer and for tables of constants that do not change in value once the production of the computer is completed.

The ROM portion of main memory is needed for bootstrap loader storing an initial program called a bootstrap loader. The bootstrap loader is a program whose function is to start the computer software operating when power is turned on. Since RAM is volatile, its contents are destroyed when power is turned off. The contents of ROM remain unchanged after power is c turned off and on again. The startup of a computer consists of turning the power on and starting the execution of an initial program. Thus when power is turned on, the hardware of the computer sets the program counter to the first address of the bootstrap loader. The bootstrap program loads a portion of the operating system from disk to main memory and control is then transferred to the operating system, which prepares the computer for general use.

**RAM and ROM Chips**

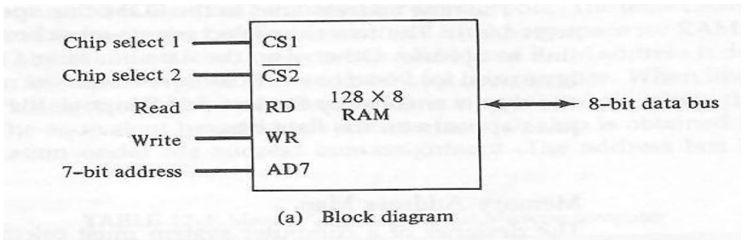
A RAM chip is better suited for communication with the CPU if it has one or more control inputs that select the chip only when needed. Another common feature is a bidirectional data bus that allows the transfer of data either from memory to CPU during a read operation, or from CPU to memory during a write operation. A bidirectional bus can be constructed with three-state buffers. A three-state buffer output can be placed in one of three possible states: a signal equivalent to logic 1, a signal equivalent to logic 0, or a high- impedance state. The logic 1 and 0 are normal digital signals. The high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

The capacity of the memory is 128 words of eight bits (one byte) per word. This requires a 7-bit address and an 8-bit bidirectional data bus. The read and write inputs specify the memory operation and the two chips select (CS) control inputs are for enabling the chip only when it is selected by the microprocessor. The availability of more than one control input to select the chip facilitates the decoding of the address lines when multiple chips are used in the microcomputer. The read and write inputs are sometimes combined into one line labeled R/W.

The function table listed in Fig. 6.2(b) specifies the operation of the RAM chip. The unit is in operation only when CS1=1 and CS2 = 0. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state.

When CS1 = 1 and CS2 = 0, the memory can be placed in a write or read mode. When the WR input is enabled, the memory stores a byte from the data bus into a location specified by the address input lines. When the RD input is enabled, the content of the selected byte is placed into the data bus.

Figure 6.2 Typical RAM chip

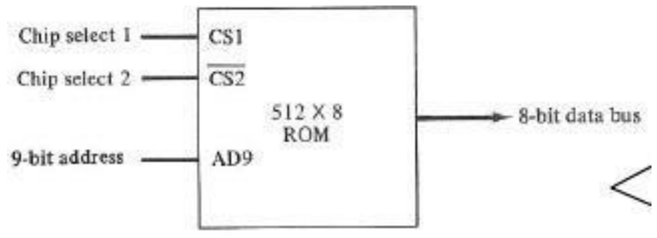


(a) Block diagram

| CS1 | $\overline{\text{CS2}}$ | RD | WR | Memory function | State of data bus |
|---|---|---|---|---|---|
| 0 | 0 | × | × | Inhibit | High-impedance |
| 0 | 1 | × | × | Inhibit | High-impedance |
| 1 | 0 | 0 | 0 | Inhibit | High-impedance |
| 1 | 0 | 0 | 1 | Write | Input data to RAM |
| 1 | 0 | 1 | × | Read | Output data from RAM |
| 1 | 1 | × | × | Inhibit | High-impedance |

A ROM chip is organized externally in a similar manner. However, since a ROM can only read, the data bus can only be in an output mode. The block diagram of a ROM chip is shown in Fig. 12.3. For the same-size chip, it is possible to have more bits of ROM than of RAM, because the internal binary cells in ROM occupy less space than in RAM

Figure 6.3 Typical ROM chip



The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The

two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate. Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read.

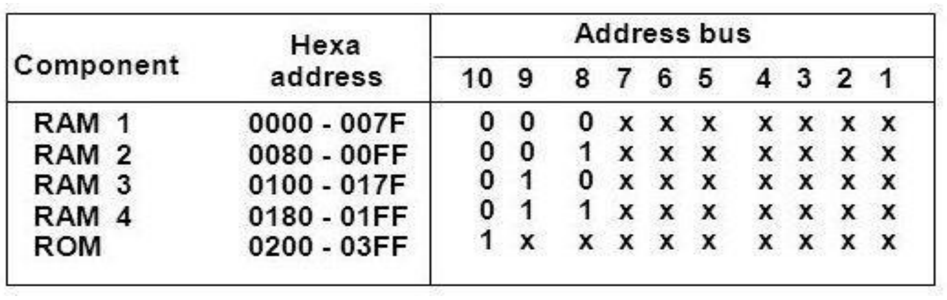
## Memory Address Map

The designer of a computer system must calculate the amount of memory required for the particular application and assign it to either RAM or ROM. The interconnection between memory and processor is then established from knowledge of the size of memory needed and the type of RAM and ROM chips available. The addressing of memory can be established by means of a table that specifies the memory address assigned to each chip. The table, called a memory address map, is a pictorial representation of assigned address space for each chip in the system.

The memory address map for this configuration is shown in Table 6-1. The component column specifies whether a RAM or a ROM chip is used. The hexadecimal address column assigns a range of hexadecimal equivalent addresses for each chip. The address bus lines are listed in the third column. Although there are 16 lines in the address bus, the table shows only 10 lines because the other 6 are not used in this example and are assumed to be zero. The small x's under the address bus lines designate those lines that must be connected to the address inputs in each chip. The RAM chips have 128 bytes and need seven address lines.

The ROM chip has 512 bytes and needs 9 address lines. The x's are always assigned to the low-order bus lines: lines 1 through 7 for the RAM and lines 1 through 9 for the ROM. It is now necessary to distinguish between four RAM chips by assigning to each a different address. For this particular example we choose bus lines 8 and 9 to represent four distinct binary combinations. Note that any other pair of unused bus lines can be chosen for this purpose. The table shows that the nine low-order bus lines constitute a memory space for RAM equal to $2^9$= 512 bytes. The distinction between a RAM and ROM address is done with another bus line. Here we choose line 10 for this purpose. When line 10 is 0, the CPU selects a RAM, and when this line is equal to 1, it selects the ROM.

Table 6-1 Memory Address Map for Micro procomputer



| Component | Hexa address | Address bus | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
| RAM 1 | 0000 - 007F | 0 | 0 | 0 | x | x | x | x | x | x | x |
| RAM 2 | 0080 - 00FF | 0 | 0 | 1 | x | x | x | x | x | x | x |
| RAM 3 | 0100 - 017F | 0 | 1 | 0 | x | x | x | x | x | x | x |
| RAM 4 | 0180 - 01FF | 0 | 1 | 1 | x | x | x | x | x | x | x |
| ROM | 0200 - 03FF | 1 | x | x | x | x | x | x | x | x | x |

**Memory Connection to CPU**

RAM and ROM chips are connected to a CPU through the data and address buses. The low-order lines in the address bus select the byte within the chips and other lines in the address bus select a particular chip through its chip select inputs. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM.

The selection between RAM and ROM is achieved through bus line 10. The RAMs are selected when the bit in this line is 0, and the ROM when the bit is 1.

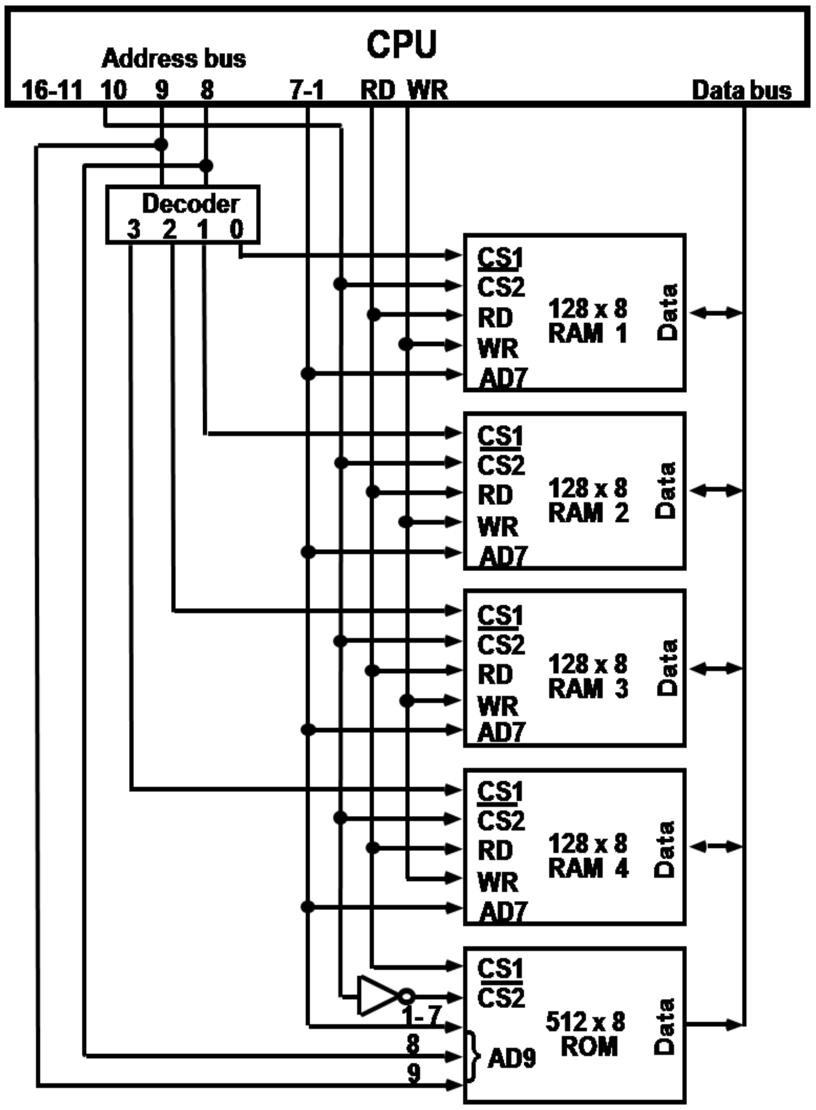


Figure 6.4 Memory connection to the CPU

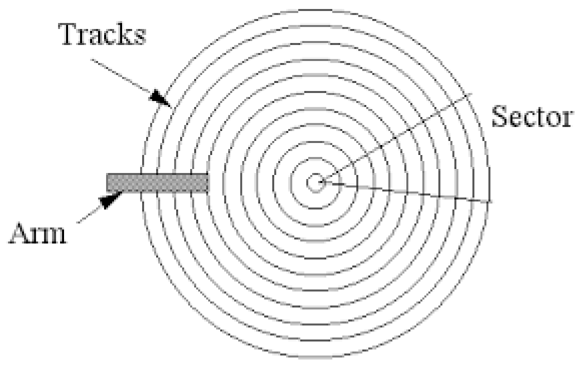
## Auxiliary memory:

The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks. The important characteristics of any device are its access mode, access time, transfer rate, capacity, and cost. The average time required to reach a storage location in memory and obtain its contents is called the access time. In electromechanical devices with moving parts such as disks and tapes, the access time consists of a seek time required to position the read-write head to a location and a transfer time required to transfer data to or from the device. Because the seek time is usually much longer than the transfer time, auxiliary storage is organized in records or blocks. A record is a specified number of characters or words. Reading or writing is always done on entire records. The transfer rate is the number of characters or words that the device can transfer per second, after it has been positioned at the beginning of the record.

### Magnetic Disks

A magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started for access purposes. Bits are stored in the magnetized surface in spots along concentric circles called **tracks. The tracks are commonly divided into sections called sectors.** In most systems, the minimum quantity of information which can be transferred is a sector.

Figure 6.5 Magnetic disk



Some units use a single read/write head for each disk surface. In this type of unit, the track address bits are used by a mechanical assembly to move the head into the specified track position before reading or writing. In other disk systems, separate read/write heads are provided for each track in each surface. The address bits can then select a particular track electronically through a decoder circuit. This type of unit is more expensive and is found onlyin very large computer systems.

Permanent timing tracks are used in disks to synchronize the bits and recognize the sectors. A disk system is addressed by address bits that specify the disk number, the disk surface, the sector number and the track within the sector. After the read/write heads are positioned in the specified track, the system has to wait until the rotating disk reaches the specified sector under the read/write head. Information transfer is very fast once the beginning of a sector has been reached. Disks may have multiple heads and simultaneous transfer of bits from several tracks at the same time.

A track in a given sector near the circumference is longer than a track near the center of the disk. If bits are recorded with equal density, some tracks will contain more recorded bits than others. To make all the records in a sector of equal length, some disks use a variable recording density with higher density on tracks near the center than on tracks near the circumference. This equalizes the number of bits on all tracks of a given sector.

Disks that are permanently attached to the unit assembly and cannot be removed by the occasional user are called hard disks. A disk drive with removable disks is called a floppy disk. The disks used with a floppy disk drive are small removable disks made of plastic coated with magnetic recording material. There are two sizes commonly used, with diameters of 5.25 and 3.5 inches. The 3.5-inch disks are smaller and can store more data than can the 5.25-inch disks. Floppy disks are extensively used in personal computers as a medium for distributing software to computer users.

**Magnetic Tape**

A magnetic tape transport consists of the electrical, mechanical, and electronic components to provide the parts and control mechanism for a magnetic-tape unit. The tape itself is a strip of plastic coated with a magnetic recording medium. Bits are recorded as magnetic spots on the tape along several tracks. Usually, seven or nine bits are recorded simultaneously to form a character together with a parity bit. Read/write heads are mounted one in each track so that data can be recorded and-read as a sequence of characters.

Magnetic tape units can he stopped, started to move forward or in reverse, or can be rewound. However, they cannot be started or stopped fast enough between individual characters. For this reason, information is recorded in. blocks referred to as records. Gaps of unrecorded tape are inserted between records where the tape can be stopped. The tape starts moving while in a gap and attains its constant speed by the time it reaches the next record. Each record on tape has an identification bit pattern at the beginning and end. By reading the bit pattern at the beginning, the tape control identifies the record number. By reading the bit pattern at the end of the record, the control recognizes the beginning of a gap. A tape unit is addressed by specifying the record number and the number of characters in the record. Records may be of fixed or variable length.

## Associative memory:

Many data-processing applications require the search of items in a table stored in memory. The established way to search a table is to store all items where they can be addressed in sequence. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.

A memory unit accessed by content is called an associative memory or content addressable memory (CAM). This type of memory is accessed simultaneously and in parallel on the basis of data content rather than by specific address or location. When a word is written in an associative memory, no address is given. The memory is capable of finding an empty unused location to store the word.

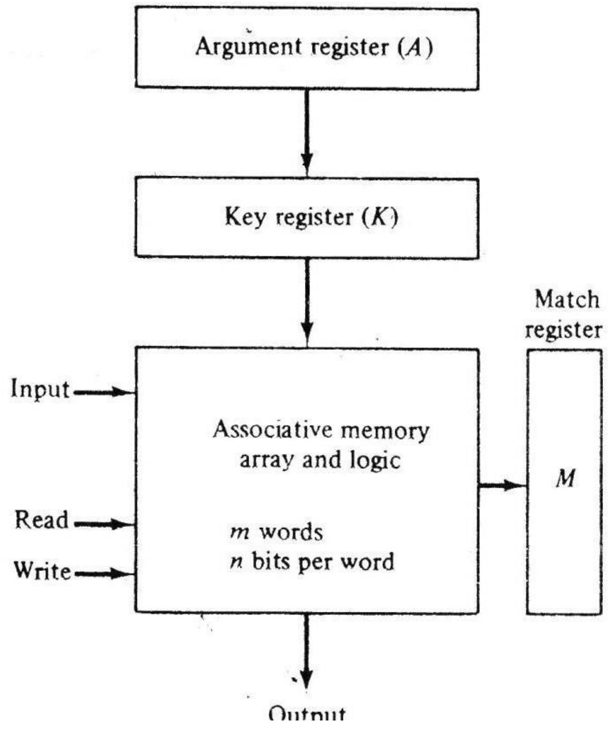
**Hardware Organization**

The block diagram of an associative memory is shown in Fig. 6.6. 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 l's. Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.

Figure 6.6 Block diagram of associative memory

To illustrate with a numerical example, suppose that the argument register A and the



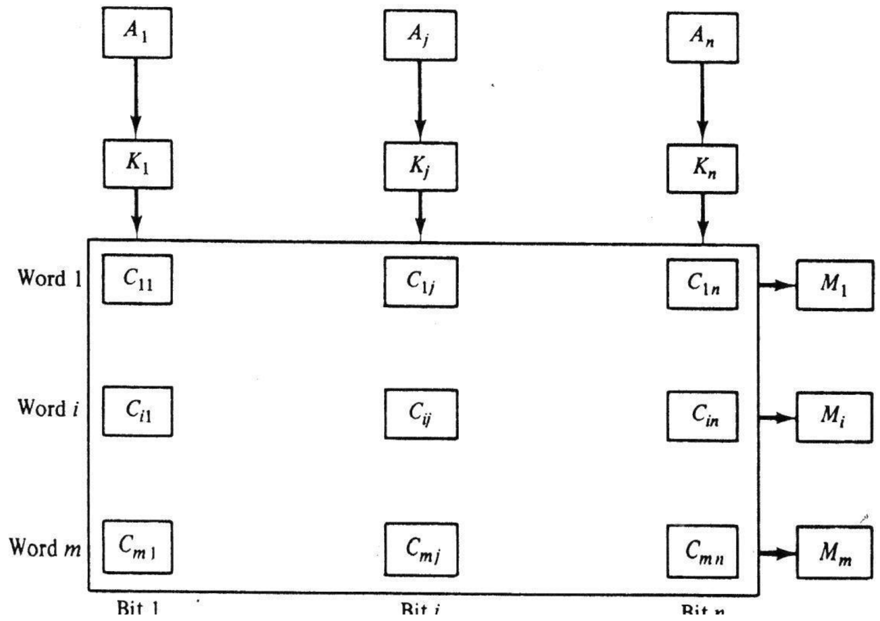
To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has l's in these positions.

| | | |
|---|---|---|
| A | 101 111100 | |
| K | 111 000000 | |
| Word 1 | 100 111100 | no match |
| Word 2 | 101 000001 | match |

Word 2 matches the unmasked argument field because the three leftmost bits of the argumentand the word are equal.

The relation between the memory array and external registers in an associative memory is shown in Fig. 6.7. 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 $C_{ij}$ is the cell for bit j in word i. A bit $A_j$ in the argument register is compared with all the bits in column j of the array provided that $K_j = 1$. 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 in the match register is set to 1. If one or more unmasked bits of the argument and the word do not match, $M_i$ is cleared to 0.

Figure 6.7 Associative memory of m word, n c ells per word



**Match Logic**

The match logic for each word can be derived from the comparison algorithm s or two binary numbers. First, we *neglect t*he key bits and compare the argument in A with the bits stored in the cells of the words.

Word i is equal to the argument in A if $A_j$=$F_{ij}$for j = 1, 2, . . . , n . Two bits are equal if they are both 1 or both 0. The equality of two bits can be expressed logically by the Boolean function

$$x_j = A_j F_{ij} + A_j F_{ij}'$$

Where $x_j = 1$ if the pair of bits in position j are equal; otherwise, $x_j = 0$.

For a word i to be equal to the argument in A we must have all $x_j$ variables equal to 1. This is the condition for setting the corresponding match bit $M_i$ to 1. The Boolean function for this condition is

Mi = $x_1 x_2 x_3 .................. x_n$

and constitutes the AND operation of all pairs of matched bits in a word. Figure 6.8 One cell of associative memory. The requirement is that if $K_j = 0$, the corresponding bits of $A_j$ and $F_{ij}$need no compatriot. This requirement is achieved b ORing each term with $K_j$ thus:

$$x_j + K_j' = \begin{cases} x_j & if\ K_j = 1 \\ 1 & if\ K_j = 0 \end{cases}$$

## Cache memory:

Analysis of a large number of typical programs has shown that the references memory at any given interval of time tend to be confined within a few localized areas in memory. This phenomenon is known as the property of locality of reference. 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 are fetched from memory. Thus loops and subroutines tend to localize the references to memory for fetching instructions. To a lesser degree, memory references to data also tend to be localized. Table- lookup procedures repeatedly refer to that portion in memory where the table is stored. Iterative procedures refer to common memory locations and array of numbers are confined within a local portion of memory. The result of all these observations is the locality of reference property, which states that over a short interval of time, the addresses generated by a typical program refer to a few localized areas of memory repeatedly, while the remainder of memory is accessed relatively infrequently.

If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced. 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.

The fundamental idea of cache organization is that by keeping the most frequently accessed instructions and data in the fast cache memory, the average memory access time will approach the access time of the cache. Although the cache is only a small fraction of the size of main memory, a large fraction of memory requests will be found in the fast cache memory because of the locality of reference property of programs.

The basic operation of the cache is as follows. 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 block size may vary from one word to about 16 words adjacent to the one just accessed.

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, ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio. Hit ratios of 0.9 and higher have been reported. This high ratio verifies the validity of the locality of reference property.

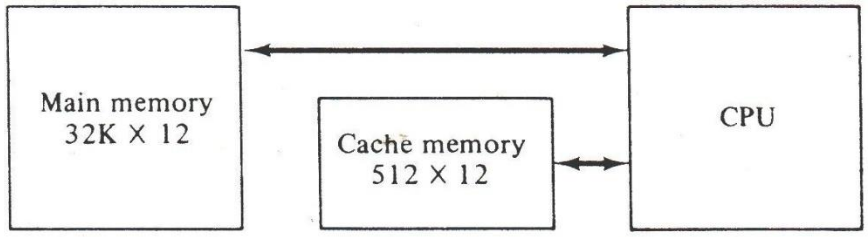
The average memory access time of a computer system can be improved considerably by use of a cached If the hit ratio is high enough so that most of the time the CPU accesses the cache instead of main memory, the average access time is closer to the access time of the fast cache memory.

The transformation of data from main memory to cache memory is referred to as a mapping process. Three types of mapping procedures are of practical interest when considering the organization of cache memory:

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

The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory. The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accepts the 12-bit data from cache. If there is a miss, the CPU reads the word from main memory and the word is then transferred to cache.

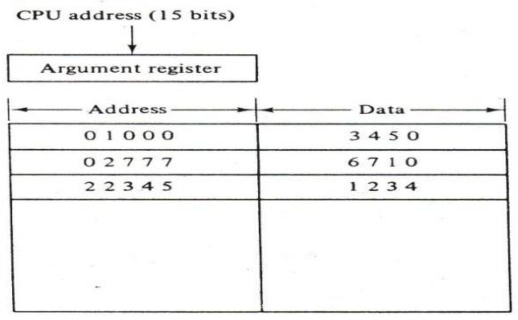
Figure 6.9 Example of cache memory



**Associative Mapping**

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. The diagram shows three words presently stored in the cache. The address value of 15 bits is shown as a five-digit octal number and its corresponding 12-bit word is shown as a four-digit octal number. 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.

Figure 6.10 associative mapping cache(all numbers in octal)



**Direct Mapping**

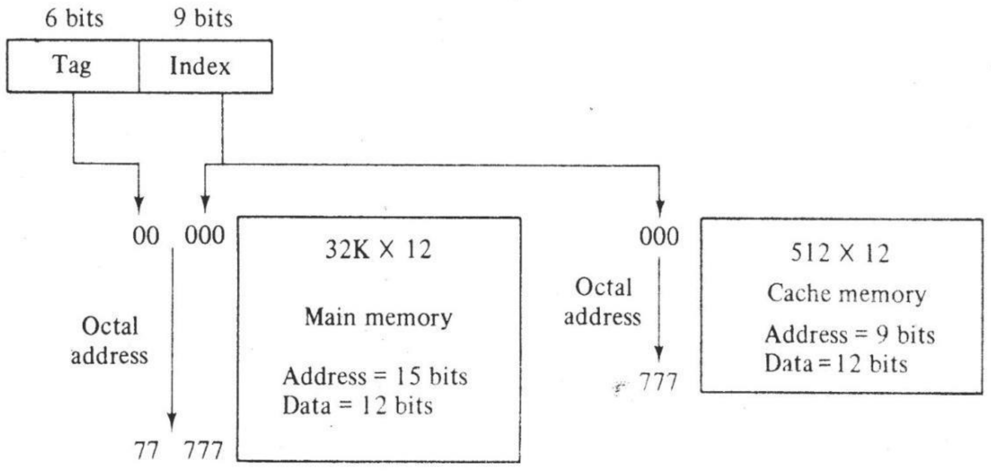


Figure 6.11 Addressing relationships between main and cache memories

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell. The CPU address of 15 bits is divided into two fields.

The nine least significant bits constitute the index field and the

Remaining six bits from the tag field. In the general case, there are $2^k$ words in cache memory and $2^n$ words in main memory. 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. 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.

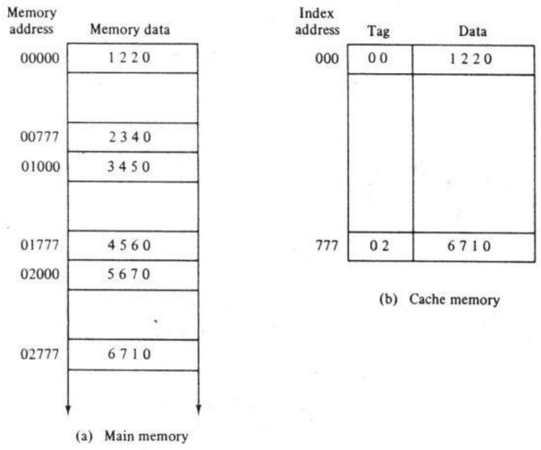


Figure 6.12 Direct mapping cache organization

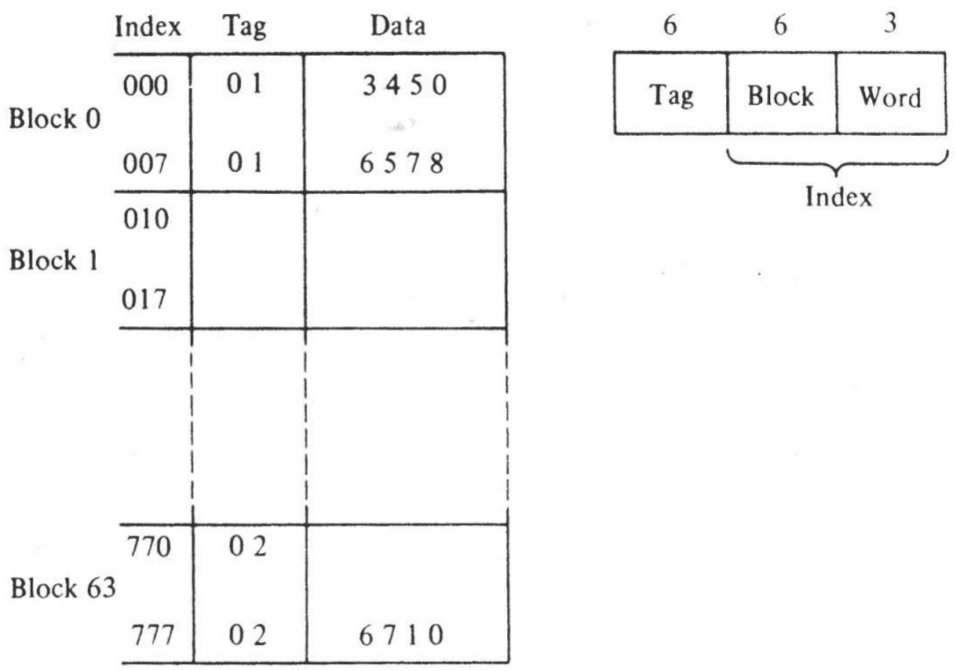
If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cachetogether with the new tag, replacing the previous value.

The word at address zero is presently stored in the cache (index = 000, tag = 00, data

= 1220). Suppose that the CPU now wants to access the word at address 02000. The index address is 000, so it is used 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 he word at index address 000 is then replaced with a tag of 02 and data of 5670.

The index field is now divided into two parts: the block field and the word field. In a 512-word cache there are 64 blocks of 8 words each, since 64 x 8 = 512. The block number is specified with a 6-bit field and the word within the block is specified with a 3-bit 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.

Figure 6.13 Direct mapping cache with block size of 8 words



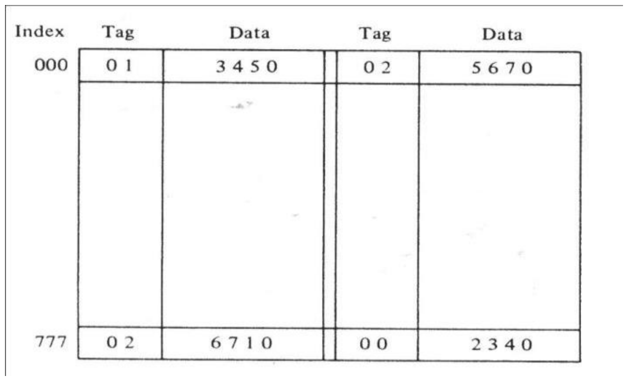
## Set-Associative Mapping

A third type of cache organization, called 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 tag requires six bits and each data word has 12 bits, so the word length is (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 words of main memory since each word of cache contains two data

words. In general, a set-associative cache of set size k will accommodate k words of main memory in each word of cache.

The octal numbers listed in Fig.6.14 are with reference to the main memory contents illustrated in Fig. 6.12(a). 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 comparison logic is done by an associative search of the tags in the set similar to an associative memory search: thus the name "set-associative." The hit ratio will improve as the set size increases because more words with the same index but different tags can reside in cache.

The most algorithms common replacement algorithms used are: random replacement, firs -in, first-out (FIFO), and least recently used (LRU)



| Index | Tag | Data | Tag | Data |
|---|---|---|---|---|
| 000 | 0 1 | 3 4 5 0 | 0 2 | 5 6 7 0 |
| | | | | |
| 777 | 0 2 | 6 7 1 0 | 0 0 | 2 3 4 0 |

Figure 6.14 Two way Set associative mapping cache

**Writing into Cache**

When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer. However, if the operation is a write, there are two ways that the system can proceed.

The simplest and most commonly used procedure is to update main memory with every memory write operation, with cache memory being up-dated in parallel if it contains the word at the specified address. This is called the write-through method.

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 word is removed from the cache it is copied into main memory.

**The 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.

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 initialization condition has the effect of forcing misses from the cache until fills with valid data.

## Virtual memory:

Virtual memory is a concept used in some large computer systems that permit the user to construct programs as though a large memory space were available, equal to the totality of auxiliary memory. Each address that is referenced by the CPU goes through an address mapping from the so-called virtual address to a physical address in main memory. Virtual memory is used to give programmers the illusion that they have a very large memory at their disposal, even though the computer actually has a relatively small main memory. A virtual memory system provides a mechanism for translating program-generated addresses into correct main memory locations. This is done dynamically, while programs are being executed in the CPU. The translation or mapping is handled automatically by the hardware by means of a mapping table.

**Address Space and Memory Space**

An address used by a programmer will be called a virtual address, and the set of such addresses the address space. An address in main memory is called a location or physical address. The set of such locations is called the memory space. Thus the address space is the set of addresses generated by programs as they reference instructions and data.

The memory space consists of the actual main memory locations directly addressable for processing. In most computers the address and memory spaces are identical. The address space is allowed to be larger than the memory space is allowed to be larger than the memory space computers with virtual memory.

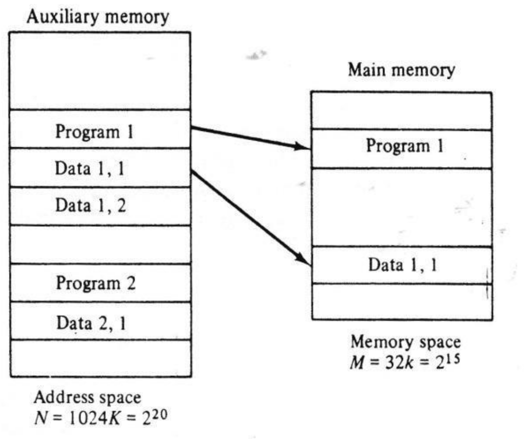
Consider a computer with a main memory capacity of 32K words (K = 1024). Fifteen bits are needed to specify a physical address in memory since 32K = $2^{15}$. Suppose that the computer has available auxiliary memory for storing $2^{20}$ = 1024K words. Thus auxiliary memory has a capacity for storing information equivalent to the capacity of 32 main memories. Denoting the address space by N and the memory space by M, we then have for this example N = 1024K and M = 32K.

In a multiprogram computer system, programs and data are transferred to and from auxiliary memory and main memory based on demands imposed by the CPU.

Suppose that program 1 is currently being executed in the CPU. Program 1 and a portion of its associated data are moved from auxiliary memory into main memory as shown in Fig. 6.15. Portions of programs and data need not be in contiguous locations in memory since information is being moved in and out, and empty spaces may be available in scattered locations in memory.

In a Virtual memory system, programmers are told that they have the total address space at their disposal. Moreover, the address field of the instruction code has a sufficient number of bits to specify all virtual addresses. In our example, the address field of an instruction code will consist of 20 bits but physical memory addresses must be specified with only 15 bits. Thus CPU, will reference instructions and data with a 20-bit address, but the information at this address must be taken from physical memory because access to auxiliary storage for individual words will be prohibitively long.

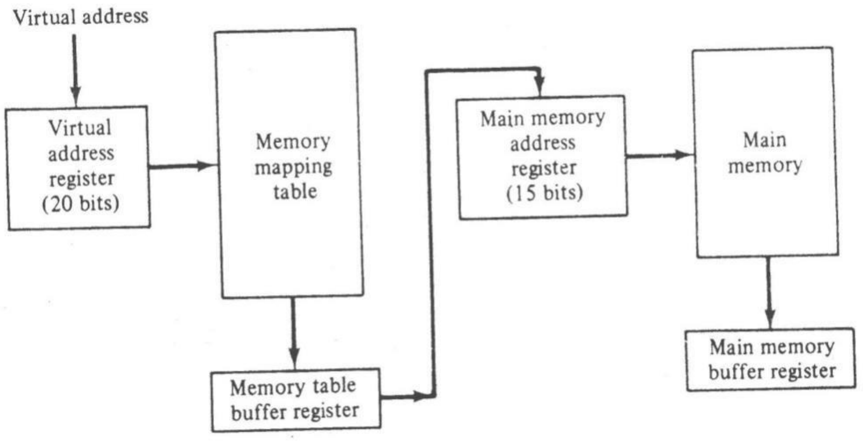
Figure 6.15 Relation between address and memory space in a virtual memory system



A table is then needed, as shown in Fig. 6.16, to map a virtual address of 20 bits to a physical address of 15 bits. The mapping is a dynamic operation, which means that every address is translated immediately as a word is referenced, by CPU.

The mapping table may be stored in a separate memory as shown in Fig.6.16 or in main memory. In the first case, an additional memory unit is required as well as one extra memory access time. In the second case, the table takes space from main memory and two accesses to memory are required with the pro ram running at half speed. A third alternative is to use an associative memory.

Figure 6.16 Memory table for mapping a virtual address


Virtual address
Virtual address register (20 bits)
Memory mapping table
Memory table buffer register
Main memory address register (15 bits)
Main memory
Main memory buffer register

**Address Mapping Using Pages.**

The information in the address space and the memory space are each divided into groups of fixed size. The physical memory is broken down into groups of equal size called blocks, which may range from 64 to 4096 words each. The term page refers to groups of address space of the same size, a page refers to the organization of address space, while a block refers to t e organization of memory space. The programs are also considered to be split into pages. Portions of programs are moved from auxiliary memory to main memory in records equal to the size of a page. The term "page frame" is sometimes used to denote ablock.

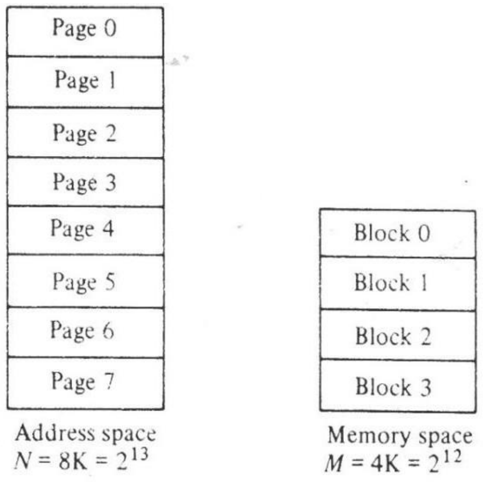
Consider computer with an address space of 8K and a memory space of 4K. If we split each into groups of 1K words we obtain eight pages and four blocks as shown in Fig.6.17. At any given time, up to four pages of address space may reside in main memory in any one of the four blocks.

The mapping from address space to memory space is facilitated if each virtual address is considered to be represented by two numbers: a page number address and a line within the page. In a computer with 2P words per page, p bits are used to specify a line address and the remaining high-order bits of the virtual address specify the page number. In the example of Fig. 6.17, a virtual address has 13 bits.

Since each page consists of $2^{10}$ = 1024 words, the high-order three bits of a virtual address will specify one of the eight pages and the low-order 10 bits give the line address within the page. Note that the line address in address space and memory space is the same; the only mapping required is from a page number to a block number

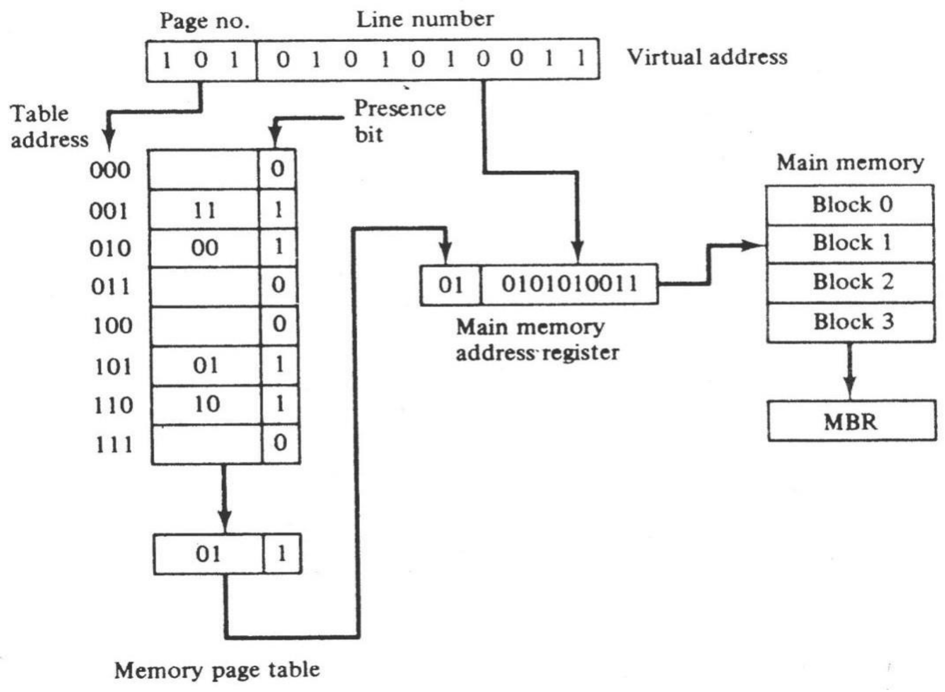
The organization of the memory mapping table in a paged system is shown in Fig.6.18. The memory-page table consists of eight words, one for each page. The address in the page table denotes the page number and the content of the word gives the block number where that page is stored in main memory. The table shows that pages 1, 2, 5, and 6 are now available in main memory in blocks 3, 0, 1, and 2, respectively. A presence bit in each location indicates whether the page has been transferred from auxiliary memory into main memory. The CPU references a word in memory with a virtual address of 13 bits. The tree high-order bits of the virtual address specify a page number and also an address for the memory-page table. The content of the word in the memory page table at the page number address is read out into the memory table buffer register.

Figure 6.17 Address space and memory space split into groups of 1Kwords



If the presence bit is a 1, the block number thus read is transferred to the two high-order bits of the main memory address register. The line number from the virtual address is transferred into the 10 low-order bits of the memory address register. A read signal to main memory transfers the content of the word to the main memory buffer register ready to be used by the CPU.

Figure 6.18 Memory table in a page system



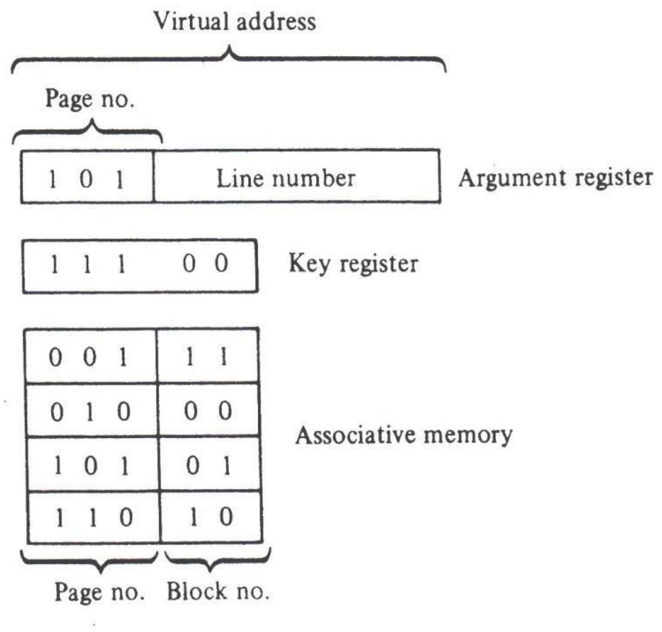
If the presence bit in the word read from the page table is 0, it signifies that the content of the word referenced by the virtual address does not reside in main memory. A call to the operating system is then generated to fetch the required page from auxiliary memory and place it into main memory before resuming computation.

**Associative Memory Page Table**

In general, a system with n pages and m blocks would require a memory-page table of n locations of which up to m blocks will be marked with block numbers and all others will be empty. As a second numerical example, consider an address space of 1024K words and memory space of 32K words. If each page or block contains 1K words, the number of pages is 1024 and the number of blocks 32. The capacity of the memory-page table must be 1024 words and only 32 locations may have a presence bit equal 1 to 1. At any given time, at least 992 locations will be empty and not in use.

A more efficient way to organize the page table would be to construct it with number of words equal to the number of blocks in main memory. In this way the size of the memory is reduced and each location is fully utilized. This method can be implemented by means of an associative memory with each word in memory containing a page number together with itscorresponding block number.

Figure 6.19 An Associative memory page table



The page field in each word is compared with the page number in the virtual address. If a match occurs, the word is read from memory and its corresponding block number is extracted.

Consider again the case of eight pages and four blocks as in the example of fig. 6.18. Each entry in the associative memory array consists of two fields. The first three bits specify a field for storing the page number. The last two bits constitute a field for storing the block number. The virtual address is placed in the argument register. The page number bits in the argument register are compared with all page numbers in the page field of the associative memory. If the page number is found, the 5-bit word is read out from memory. The corresponding block number, being in the same word, is transferred to the main memory address register. If no match occurs, a call to the operating system is generated to bring the required page from auxiliary memory.

## Page Replacement

The memory management software system handles all the software operations for the efficient utilization of memory space. It must decide (1) which page in main memory ought to be removed to make room for a new page, (2) when a new page is to be transferred from auxiliary memory to main memory, and (3) where the page is to be placed in main memory.

When a program starts execution, one or more pages are transferred into main memory and the page table is set to indicate their position. The program is executed from main memory until it attempts to reference a page that is still in auxiliary memory. This condition is called page fault. When page fault occurs, the execution of the present program is suspended until the required page is brought into main memory. Since loading a page from auxiliary memory to main memory is basically an I/O operation, the operating system assigns this task to the I/O processor. In the meantime, control is transferred to the next program in memory that is waiting to be processed in the CPU. Later, when the memory block has been assigned and the transfer completed, the original program can resume its operation.

When a page fault occurs in a virtual memory system, it signifies that the page referenced by the CPU is not in main memory. A new page is then transferred from auxiliary memory to main memory. If main memory is full, it would be necessary to remove a page from a memory block to make room for the new page. The policy for choosing pages to remove is determined from the replacement algorithm that is used. The goal of a replacement policy is to try to r move the page least likely to be referenced in the immediate future.

Two of the most common replacement algorithms used are the first in, first-out(FIFO) and the least recently used (LRU).

The FIFO algorithm selects for replacement the page that has been in memory the longest time. Each time a page is loaded into memory, its identification number is pushed into a FIFO stack. FIFO will be full whenever memory has no more empty blocks. When a new page must be loaded, the rage least recently brought in is removed. The page to be removed is easily determined because its identification number is at the top of the FIFO stack. The FIFO replacement policy has the advantage of being easy to implement. It has the disadvantage that under certain circumstances pages are removed and loaded fro-n memory too frequently).

The LRU policy is more difficult to implement but has been more attractive the assumption that the least recently used page is a better candidate for removal than the least recently loaded page as in FIFO. The LRU algorithm can be implemented by associating a counter with every page that is in main memory. When a page is referenced, its associated counter is set to zero. At fixed intervals of time, the counters associated with all pages presently in memory are incremented by 1. The least recently used page is the page with the highest count. The counters are often called aging registers, as their count indicates their age, that is, how long ago their associated pages have been referenced.

## Input-Output Organization:

### Peripheral Devices:

- The input output devices connected to the computer is called as **peripherals**. Among the most common peripherals are keyboards, display units, and printers. Peripherals that provide auxiliary storage for the system are magnetic disks and tapes.
- The input-output subsystem of a computer, referred to as I/O, provides an efficient mode of communication between the central system and the outside environment. The most familiar input device is keyboard that allows a person to enter alphanumeric information directly. Every time a key is pressed and released, the terminal sends a binary coded character to the computer.
- **Monitor:** Video monitors are the most commonly used peripherals. The keyboard is an input device and a display unit is the output device. There are different types of monitors, but the most popular use a cathode ray tube (CRT). The CRT contains an electronic gun that sends an electronic beam to a phosphorescent screen in front of tube. The beam can be deflected horizontally and vertically.
- **Printer:** Printers provide a permanent record on paper of computer output data or text. There are 3 types of character printers: **daisywheel, dot matrix, and laser printers.**
    - The daisywheel printer contains a wheel with the characters placed along the circumference. To print a character, the wheel rotates to the proper position.
    - The dot matrix printer contains a set of dots along the printing mechanism.
    - The laser printer uses a rotating photographic drum that is used to imprint the character images. This pattern transferred onto paper in the same manner as a copying machine.
- **Magnetic tape:** Magnetic tapes are used mostly for storing files of data. For example, a company's payroll record. Access is sequential and consists of records that can be accessed one after another as the tape moves along a stationary read write mechanism. It is one of the cheapest and slowest methods for storage. Magnetic tapes automatically removed when not in use.
- **Magnetic disk:** Magnetic disks have high speed rotational surfaces coated with plastic material. Access is achieved by moving a read write mechanism to a track in the magnetized surface. Disks are used mostly for bulk storage of programs and data.

## 4.2 Input-Output Interface

Input-output interface provides a method for transferring information between internal storage and external I/O devices. Peripherals connected to computer need special communication links for interfacing them with the central processing unit. Some differences between peripherals and CPU and memory unit is below:

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.

2: The data transfer rate of peripheral is usually slower than the transfer rate of CPU.

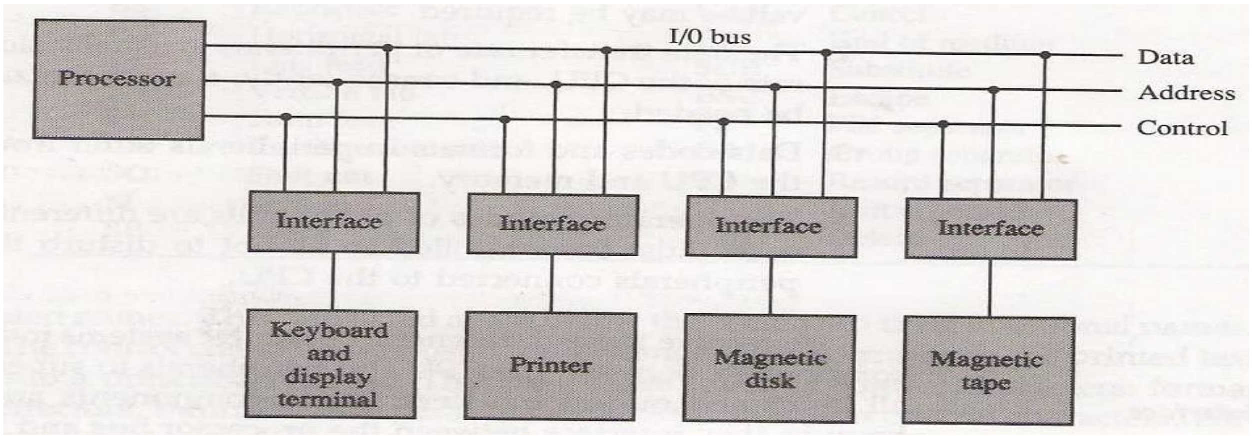
3: Data codes and formats in peripherals differ from the word format in CPU and memory.

4: The operating modes of peripherals are different from each other and each must be controlled by the CPU.

To resolve these differences, computer systems include special hardware components between the CPU and peripherals to supervise all input and output transfers. These components are called interface units because they interface between the processor bus and the peripheral device.

### 4.2.1 I/O Bus and Interface Modules

A typical communication link between the processor and several peripherals is shown below. The I/O bus consists of data bus/lines, address bus/lines, and control bus/lines. The I/O bus from the processor is attached to all peripheral interfaces.



**Connection of I/O bus to input output device**

The I/O command is a function code and is in essence an instruction that is executed in the interface and it is attached to peripheral unit. There are 4 types of interfaces: control command, status command, data output, and input.
**Control command:** To control the peripheral and interface operations.
**Status command:** To test the various status conditions in the interface and peripherals.
**Data output command:** To transfer the data from the bus into one of its registers.
**Input command:** It is opposite of the data output.

### 4.2.2 I/O versus Memory Bus

In addition to communicating with I/O, the processor must communicate the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines with IOP. The computer has independent sets of data, address, and control buses, one for accessing memory and the other for I/O. This operates done at separate I/O processor(IOP) in addition to the CPU. There are 3 ways 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 both memory and I/O with common control lines.
The purpose of IOP is to provide an independent pathway for the transfer of information between external devices and internal memory is shown in below diagram.

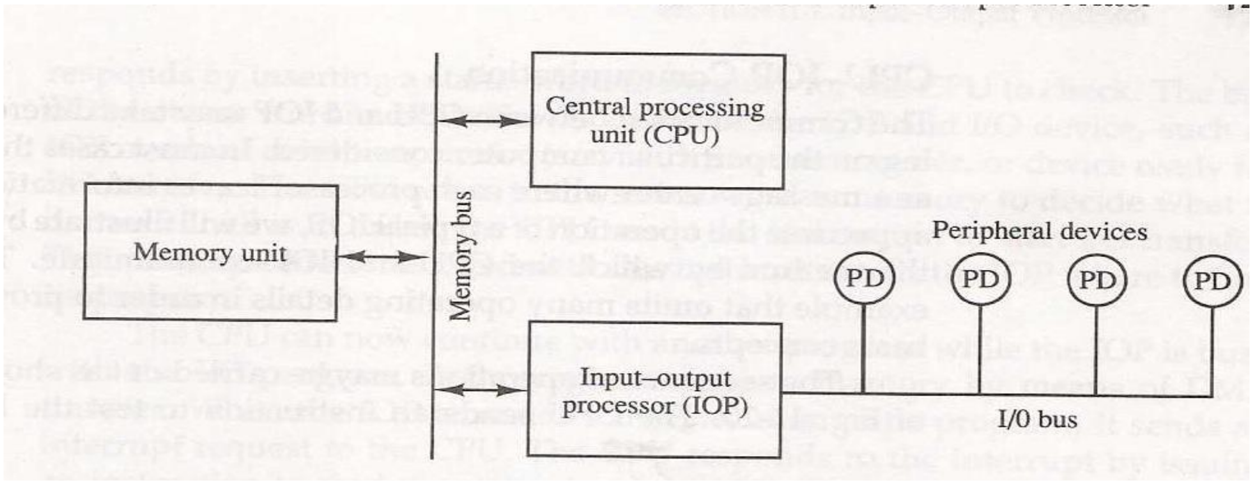


**Fig: Block diagram of computer with IO processor**

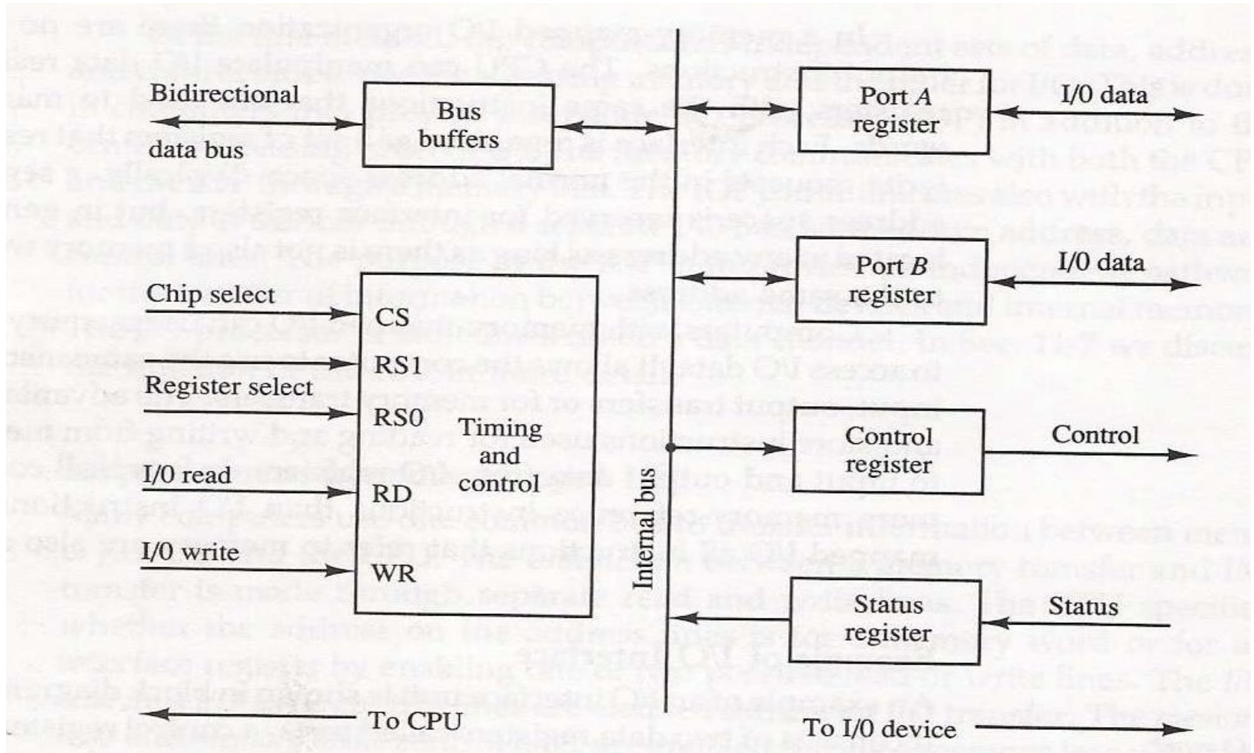
### 4.2.3 Isolated versus Memory mapped I/O

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 I/O read and I/O write control lines are enabled during an I/O transfer. The memory read and memory write control lines are enabled during a memory transfer. This configuration isolates all I/O interface addresses from the addresses assigned to memory and is referred to as the isolated I/O method for assigning addresses in a common bus.

In the isolated I/O configuration, the CPU has distinct inputs and output instructions, and each of these instructions is associated with the address of an interface register. The isolated I/O method isolates memory and I/O addresses so that memory address values are not affected by the interface register. 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 for read and write signals and do not distinguish between memory the and I/O address. This configuration is called as memory-mapped I/O. This reduces the memory address without using an interface register. The CPU can manipulate I/O instructions and memory reference instructions data directly within memory unit.

### 4.2.4 Example of I/O interface:

An example of an I/O interface unit consists of two data registers called ports, a control register, a status register, 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 I/O write are two control lines that specify an input or output operations. 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. It is shown in below diagram.



| 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 |

## Asynchronous data transfer:

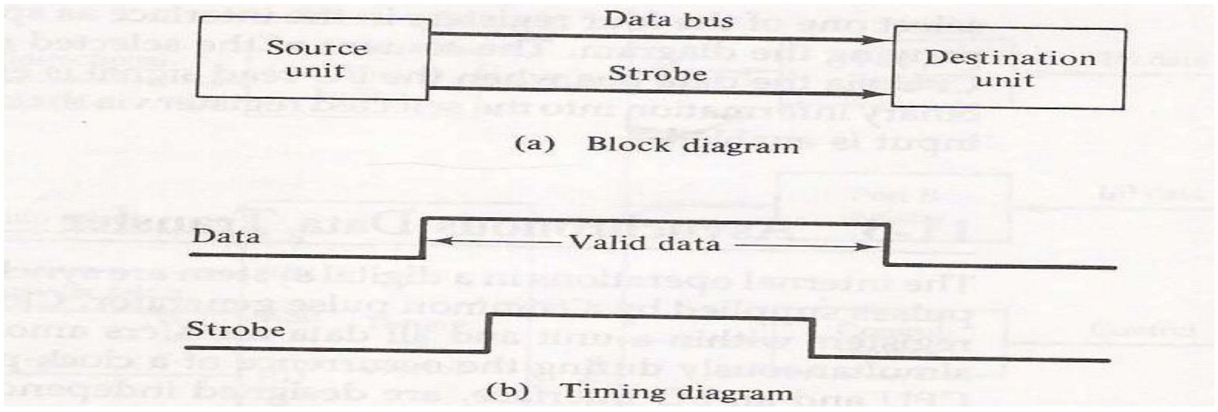
The internal operations in a digital system are synchronized by means of clock pulses supplied by a common pulse generator. That means a single clock pulse for all the devices. Clock pulses are applied to all registers within a unit and all data transfers among internal registers occur simultaneously during the occurrence of a clock pulse.

Two units such as a CPU and an I/O interface are designed independently of each other. If the registers in the interface share a common clock with the CPU registers, the transfer between the two units is said to be **synchronous.**

**Asynchronous** data transfer between two independent units requires that control signals for transferring between the communicating units to indicate the time at which data is being transmitted. Asynchronous data transfer can be classified into two types: 1. **Strobe pulse** and 2. **Hand shaking pulse.**

### 4.3.1 Strobe pulse

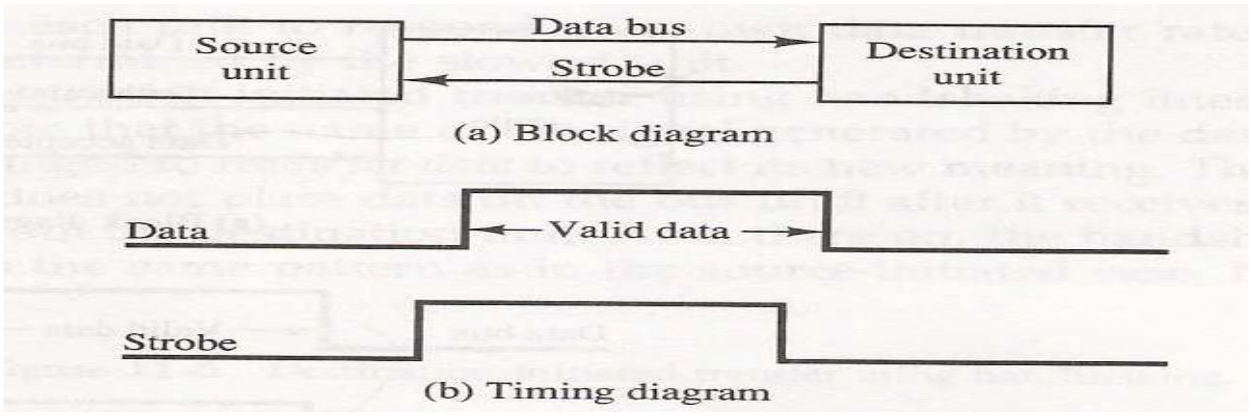
The strobe pulse and handshaking method of asynchronous data transfer are not restricted to I/O transfers. The timing diagram pass timing control signals to asynchronous data transfer. Initially data is on data bus, before going to transfer data from one unit to other unit it is checking whether given data is valid data or not with **strobe pulse.** It is shown in below diagrams. The first diagrams transfer data from source unit to destination unit. The second diagram transfer data from source unit to destination but strobe pulse is from destination unit to source unit. Both the diagrams follow timing signals.



**Source initiated strobe for data transfer**

**Strobe controlled**

The strobe control method of asynchronous data transfer employs a single control line to time each transfer. The strobe may be activated by either source or the destination unit.

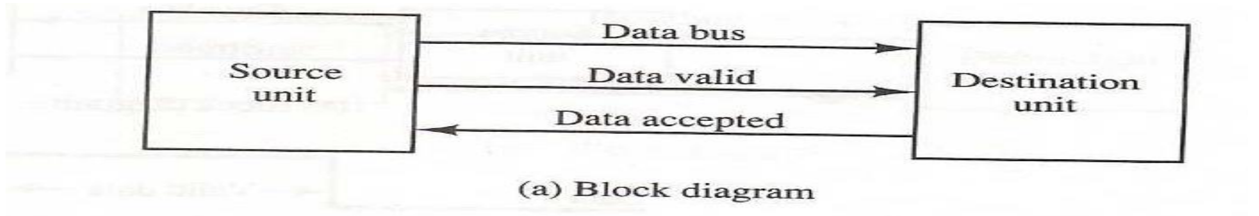


**Destination initiated strobe for data transfer**

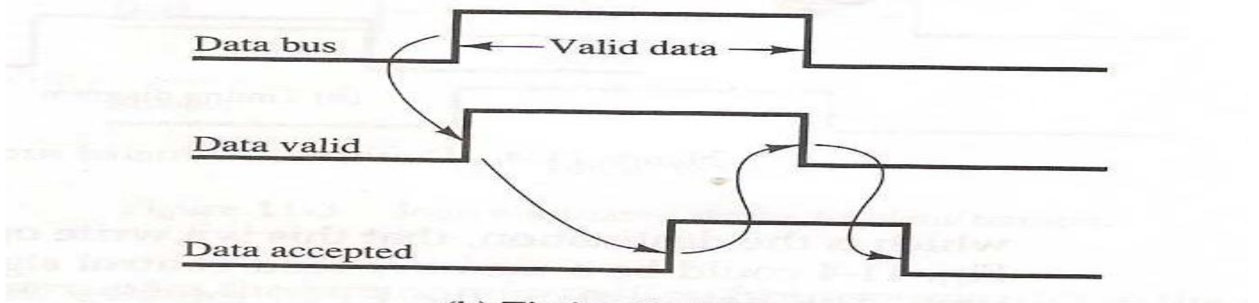
The data bus carries the binary information from source unit to the destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available in the bus. The source removes the data from the bus a brief period after it disables its strobe pulse. The data transfer initiated by 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.

### 4.3.2 Handshaking

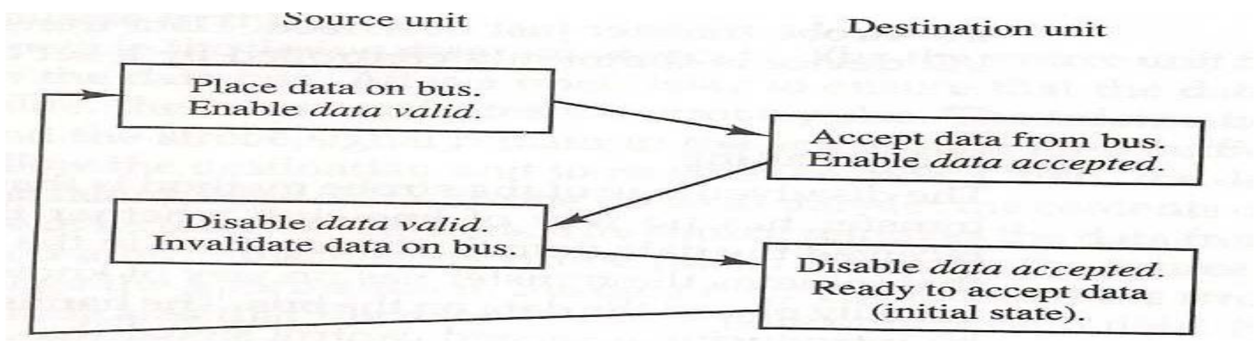
The disadvantage of the strobe method is that the source unit sent the data but it does not know whether the destination unit has actually received the data item or not. The handshaking method solves this problem by introducing a second control signal that provides a reply to the unit that initiates the transfer. This basic principle of the two-wire handshaking method of data transfer is as follows. One control line is in the same direction as the data flow in the bus from the source to the destination. It is used by the source unit to inform the destination unit whether there are valid data in the bus. The other control line is in the other direction from the destination to the source for accepting data. The two handshaking lines are **data valid** and **data accepted**. The diagrams for source to destination and destination to source unit is shown below.



(a) Block diagram



(b) Timing diagram

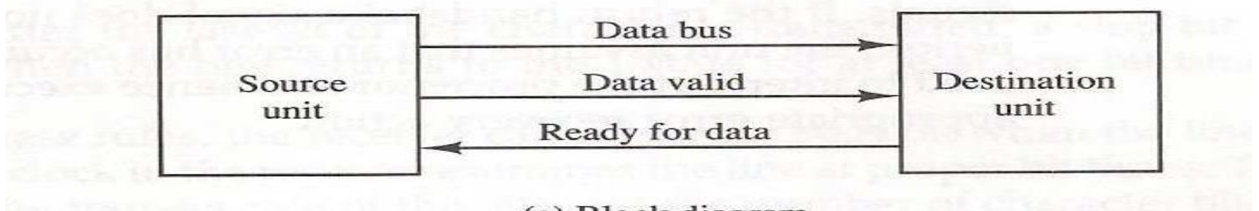

Source unit
Destination unit
Place data on bus. Enable data valid.
Accept data from bus. Enable data accepted.
Disable data valid. Invalidate data on bus.
Disable data accepted. Ready to accept data (initial state).

**Sequence of events**

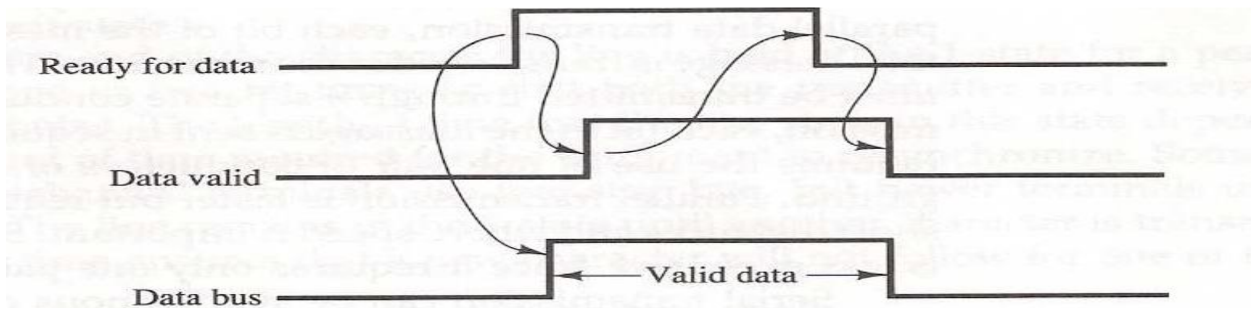
**Source initiated data transfer using handshaking**

**Destination initiated transfer using hand shaking**

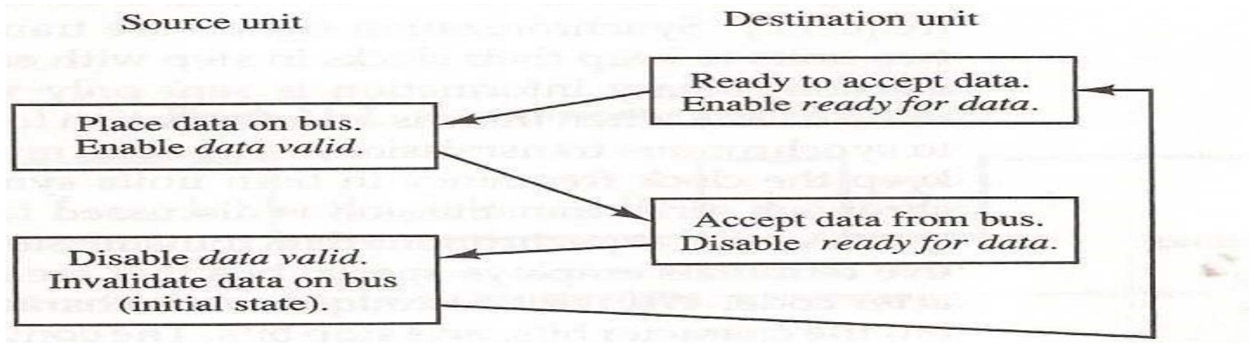
In the destination initiated transfer using hand shaking method, first ready for data is active then data on data bus and check whether data is valid or not. Later it transfers data from source to destination after it disables ready for data and data valid. It is shown in below diagrams.



(a) Block diagram



**(b) Timing diagram**



Source unit
Destination unit
Ready to accept data. Enable *ready for data*.
Place data on bus. Enable *data valid*.
Accept data from bus. Disable *ready for data*.
Disable *data valid*. Invalidate data on bus (initial state).

**(c) Sequence events**
**Destination initiated transfer using hand shaking**

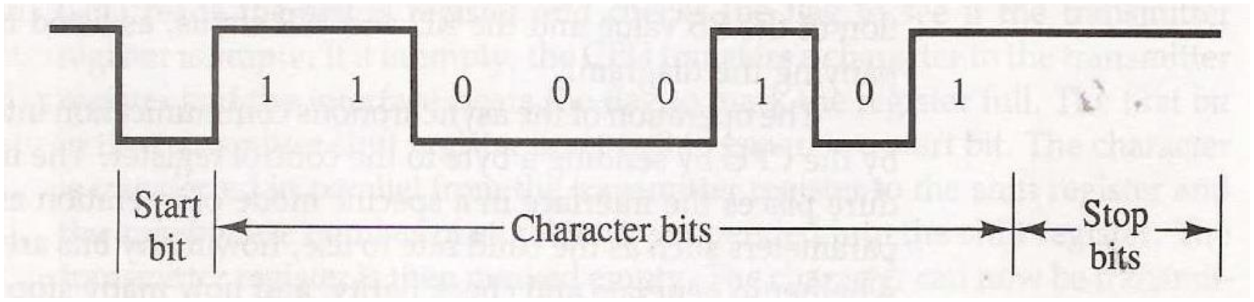
### 4.3.3 Asynchronous serial transfer

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. So it is used for short distances and where speed is important.

Serial transmission can be synchronous or asynchronous. In synchronous transmission, the two units share a common clock frequency and bits are transmitted continuously at the rate dictated by the clock pulses. 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.



**Asynchronous serial data transfer**

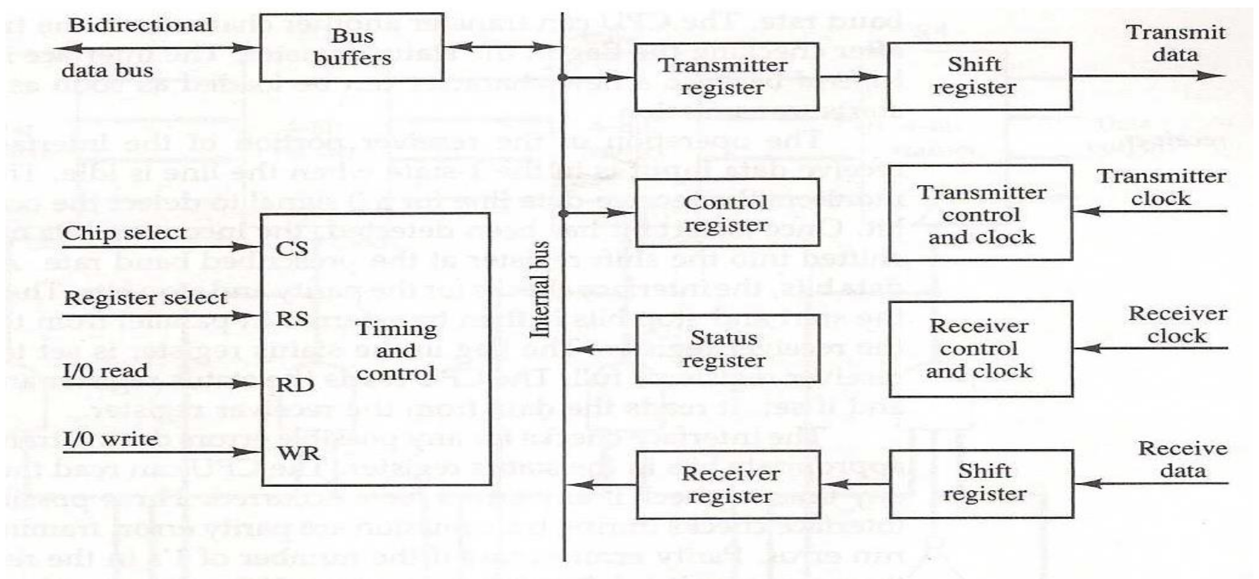
The **baud rate** is defined as the rate at which serial information is transmitted and is equivalent to the data transfer in bits per second. The circuit representation of data transfer is called as asynchronous communication interface or a universal asynchronous receiver transmitter (UART).

**4.3.4 Asynchronous Communication Interface**

The block diagram of an asynchronous communication interface is shown in below. It functions as both a transmitter and a receiver. The interface is initialized for a particular mode of transfer by mean of a control byte that is loaded into its control register.

In asynchronous communication data transfer circuit consists of bus buffer, 4 registers (transmitter register, receiver register, control register and status register), and timing control. 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 receives serial information from other shift register. The bits in the status register to check the status of the flag bits and for recording certain errors that may occur during transmission.

The chip selection and the read and write control lines communicate with the CPU. When chip selection (CS) input is 1 then it performs operations. The register selection (RS) is associated with the read (RD) and write (WR) controls. Two registers are write-only and two are read-only. It is listed in below diagram.



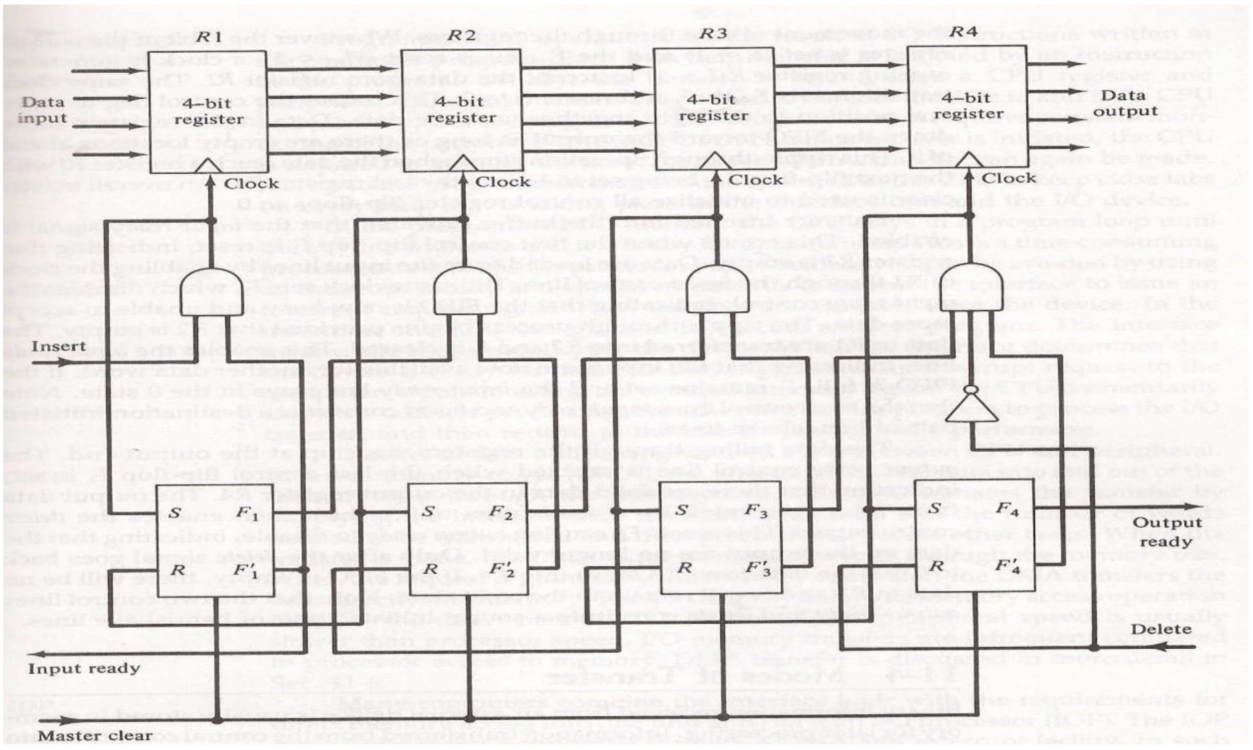
| CS | RS | Operation | Register selected |
|---|---|---|---|
| 0 | × | × | None: data bus in high-impedance |
| 1 | 0 | WR | Transmitter register |
| 1 | 1 | WR | Control register |
| 1 | 0 | RD | Receiver register |
| 1 | 1 | RD | Status register |

**4.3.5 First-in, First-out buffer**

A first in first out 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.

The logic diagram of a typical 4 * 4 FIFO buffer is shown in below figure. It consists of four registers R1,R2,R3,R4 and a control register with flip-flops Fi, i=1,2,3,4 one for each register. The FIFO can store four words of four bits each. The master clear is a clock pulse to clear the devices. A flip-flop Fi in the control register that is set to 1 indicates that a 4-bit data word is stored in the corresponding register RI. Suppose a flip-flop Fi is 0 that indicates corresponding register does not contain valid data. The control register directs the movement of data through the registers.

Data are inserted into the buffer provided that the input ready signal is enabled. When register R1 empty; then data is loaded input lines by enabling the clock in R1 through the insert control line. This procedure is continuing until four registers full. Later it deletes the data from first register first and repeats same procedure from register R1 to R4.



**Circuit diagram of 4 * 4 FIFO**

## 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. Data transfer between the central computer and I/O devices may be handled in a variety of modes. Some modes of transfer are

**1. Programmed I/O**
**2. Interrupt Initiated or driven I/O**
**3. Direct memory Access (DMA)**

### 4.4.1 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. 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.

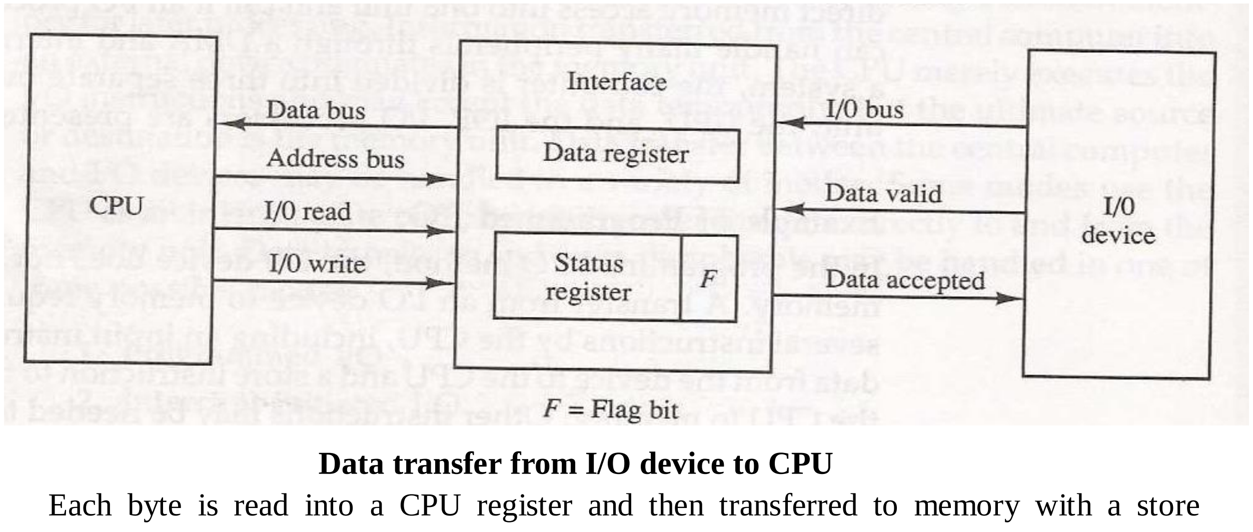
### 4.4.1.2 Example of Programmed I/O

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.

The transfer of each byte from I/O device to CPU requires three instructions:

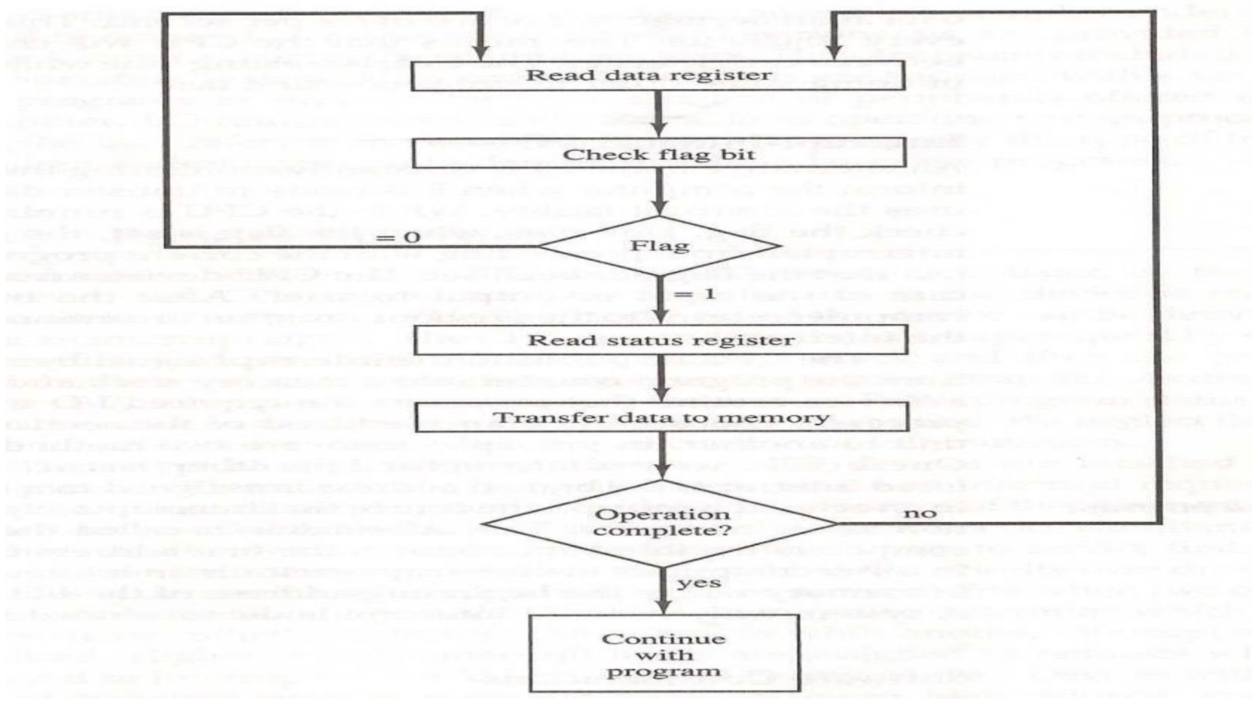
1. Read the data register
2. Check the status of the flag bit and branch to step1 if not set else go to step3
3. Read the status register.

It is shown in below diagram. It consists of data on data bus and transfer from I/O device to CPU with an interface while transferring data it is checking whether valid data or not. If data is valid then send acceptance to I/O device.



**Data transfer from I/O device to CPU**

Each byte is read into a CPU register and then transferred to memory with a store instruction. A common I/O programming task is to transfer a block of words from an I/O device and store them in a memory buffer. It is shown in below flowchart:



**Flowchart for CPU program into input data**

### 4.4.2 Interrupt Initiated I/O:

In the programmed I/O method, two problems occurred. First, 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 and sometimes the source program loop is repeated infinite times when a problem occurred in loops. So 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 mean time CPU can proceed to execute another program. When an external interrupt signal is detected, then it stops the task it is processing. The interrupts are classified into two types:

#### 4.4.2.1 Vectored Interrupt

In vectored interrupt, the source program contains an interrupt.

#### 4.4.2.2 Non Vectored Interrupt:

In non vectored interrupt, the branch address assigned to a fixed location in memory.

## Priority interrupts:

Data transfer between the CPU and an I/O device is initiated by the CPU. However, the CPU cannot start the transfer unless the device is ready to communicate with the CPU. The readiness of the device can be determined from an interrupt signal. A priority interrupt is a system that establishes a priority over various source to destination which condition is to be serviced first when two or more requests arrive simultaneously. The system may also determine which conditions are permitted to interrupt the computer while another interrupt is being serviced. Higher order priority interrupt levels are assigned to requests which if delayed or interrupted, could have serious consequences.

Priority interrupts are classified into two types:

### 4.5.1 Software priority interrupt

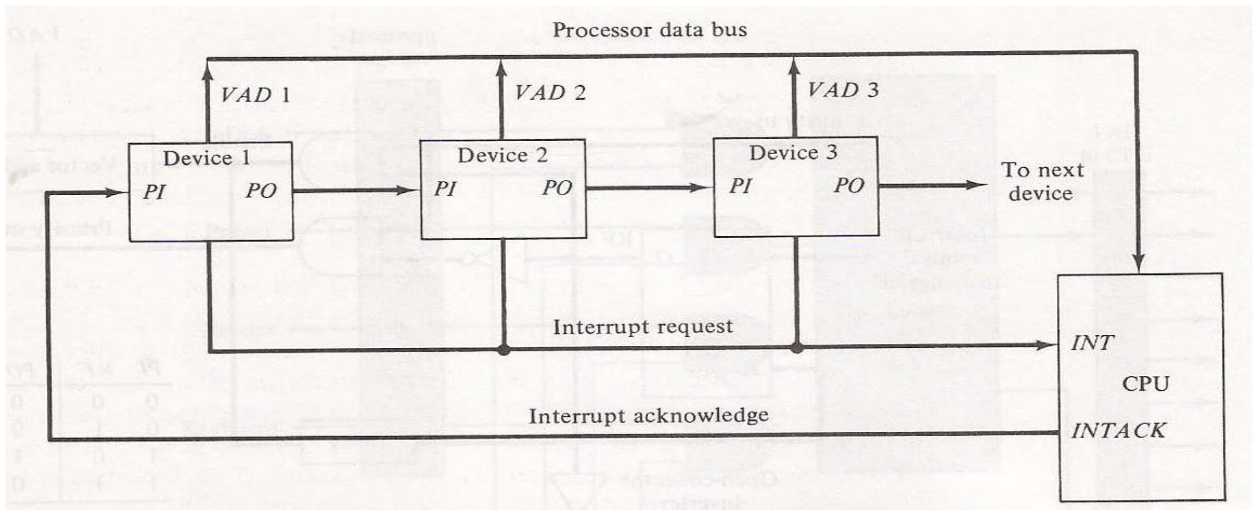
In software priority interrupt, a common memory address is used for entire memory unit. So it will takes more time for transferring data. A polling procedure is software, which is used to identify the highest-priority source.

### 4.5.2 Hardware priority interrupts

Hardware priority interrupts again classified into two types:

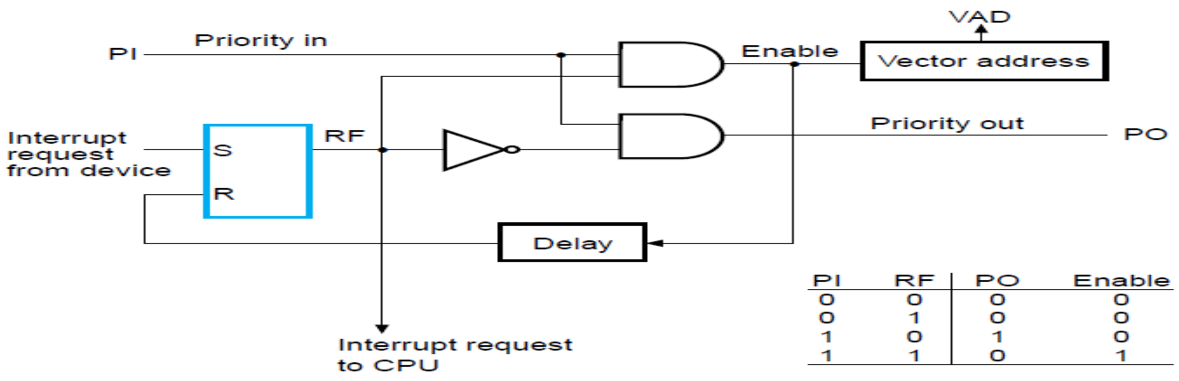
#### 4.5.2.1 Daisy-chaining priority (Serial transfer priority interrupt)

The daisy-chaining priority 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 in chain. This method provides a connection between three devices and the CPU. 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.



**Daisy chain priority interrupt**

The one stage of a Daisy chain priority interrupt contains SR-flip-flop, Priority input and priority output. When **priority in (PI)** is 0; then it will not perform any operation otherwise it performs Priority output and enable operation. It is shown in below diagram.

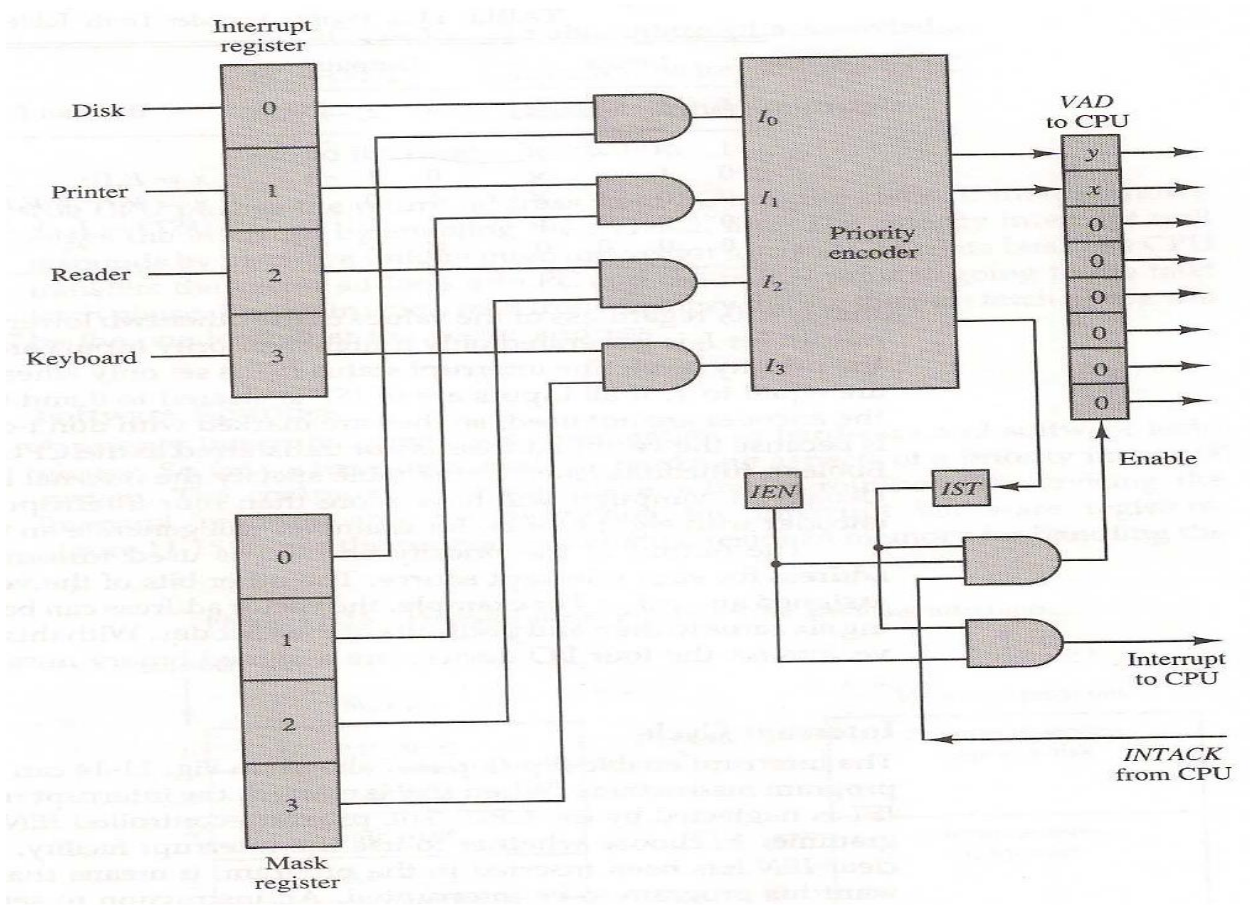


| PI | RF | PO | Enable |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 |

**One stage of the daisy chain priority arrangement**

### 4.5.2.2 Parallel priority Interrupt

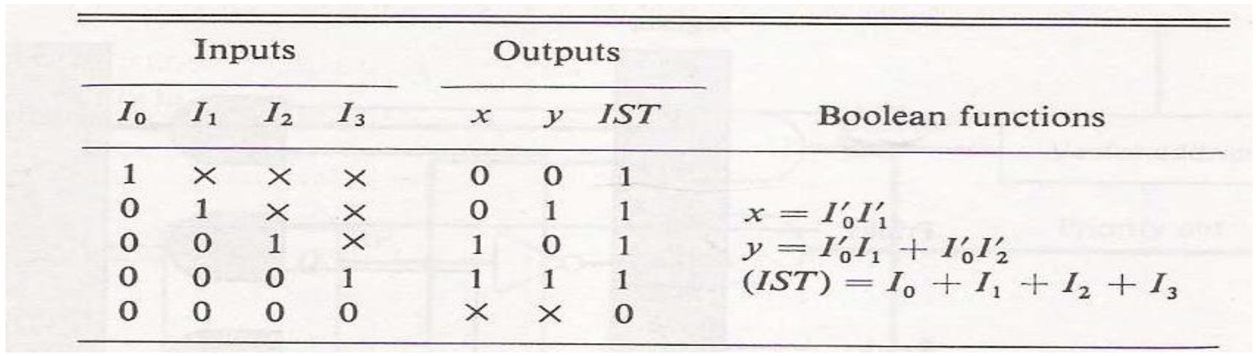
The parallel priority interrupt method uses a register whose bits are set separately by the interrupt signal from each device. Priority is established according to the position of the bits in the register. In addition to the interrupt register, the circuit may include a **mask register** whose purpose is to control the status of each interrupt request.



**Priority interrupt hardware**

### 4.5.2.3 Priority Encoder:

The priority encoder is a circuit that implements the priority function. The priority encoder takes two or more inputs arrive at same time. The input having the highest priority will take high precedence. The truth table of a four input priority encoder is shown in below diagram. It has *'s in the table designate don't care conditions. Input Io has the highest priority



| 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 | |

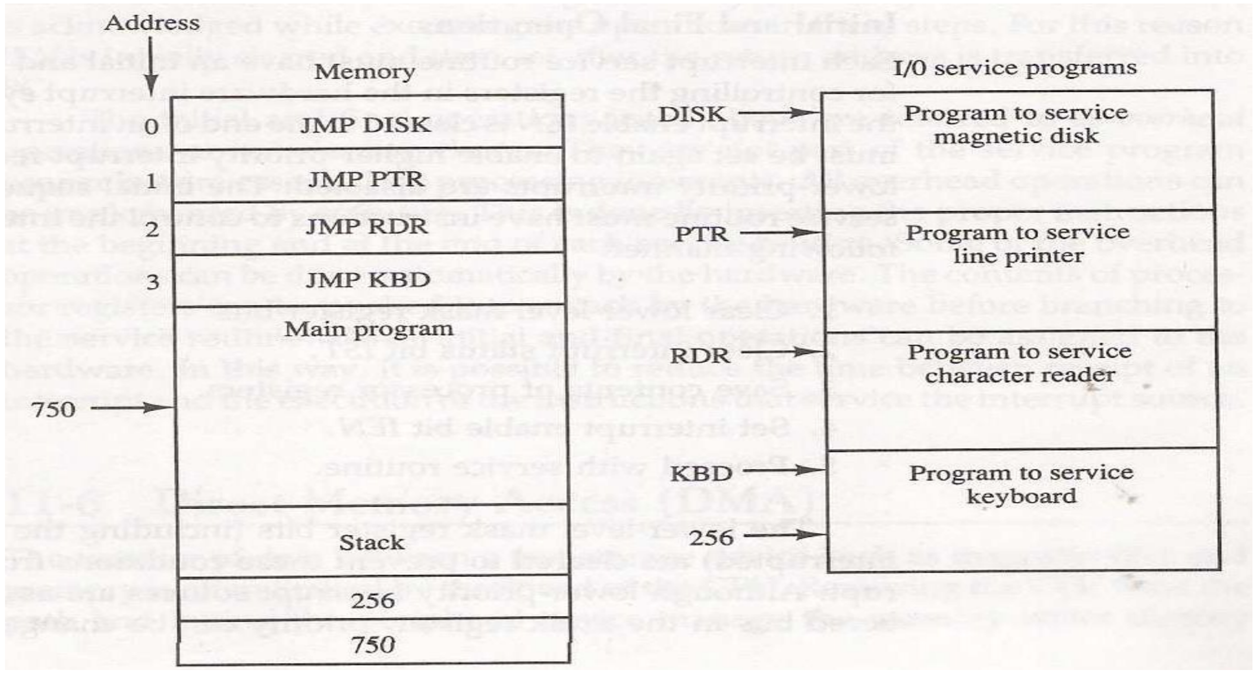
**4.5.2.4 Interrupt cycle**

The interrupt enable flip-flop IEN can be set or cleared by the program instructions. When IEN is cleared, the interrupt request coming from IST is neglected by the CPU. When IEN is set, then the interrupt facility will be used while the current program is running. It means an interrupt signal is repeatedly occurs is known as **interrupt cycle**.

**4.5.2.5 Software Routines**

A priority interrupt system is a combination of hardware and software techniques. So far we have discussed the hardware aspects of a priority interrupt system. The computer must also have software routines for servicing the interrupt requests and for controlling the interrupt hardware registers. The diagrammatic representation is shown below. Each device has its own service program that can be reached through a jump(JMP) instruction stored at the assigned vector address.

Each interrupt service routine must have an initial and final set of operations for controlling the registers in the hardware interrupt system like clear lower-level mask register, clear IST bit, set IEN bit and proceed service routine. Finally save the contents of process register.


Address
Memory
0
JMP DISK
1
JMP PTR
2
JMP RDR
3
JMP KBD
Main program
750
Stack
256
750
I/0 service programs
DISK
Program to service magnetic disk
PTR
Program to service line printer
RDR
Program to service character reader
KBD
Program to service keyboard
256

**Programs stored in memory for servicing interrupts**

## <u>Direct Memory Access:</u>

Second drawback of programmed I/O, there is no direct data transfer between CPU and peripherals. In direct memory access (DMA), removing the CPU from the path and letting the peripheral device manage the memory buses directly. Here interface register is not used for transferring 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. 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.

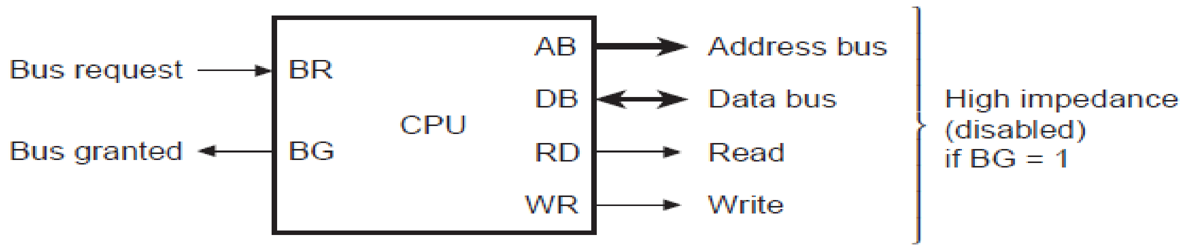
The CPU may be placed in an idle state in a variety of ways. One common method used in microprocessors is to disable the buses through special control signals: **bus request and bus grant.** The **bus request (BR)** input is used by the DMA controller to request the CPU to relinquish control of buses. When the CPU is accepted the bus request then it transfer **bus grant (BG)** signal to DMA controller. The CPU performs different address bus, data bus, read and write operations. It is shown in DMA controller block diagram below.

**DMA channel:** system pathway used by a device to transfer information directly to and from memory.

**DMA controller:**Dedicated hardware used for controlling the DMA operation

**Single-cycle mode/ Cycle stealing:**In DMA data transfer is done one byte at a time.

**Burst-mode/ burst transfer:** DMA transfer is finished when all data has been moved. That means, In DMA burst transfer, a block consisting of a number of memory words is transfer continuous burst while the DMA controller is master of the memory buses. When the request is granted by the memory controller, the DMA transfers the data directly into memory.



**CPU bus signals for DMA transfer**

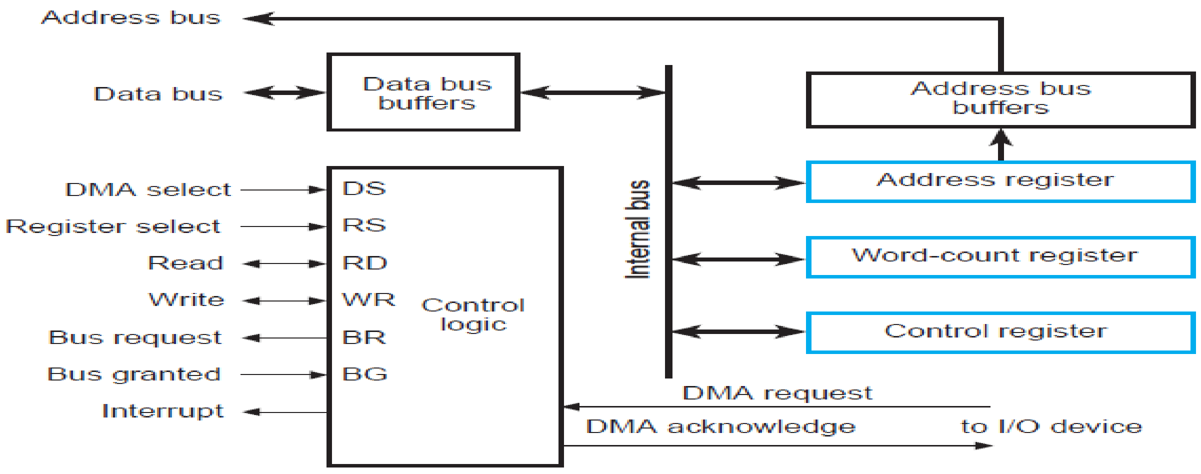
### 4.4.3.1 DMA controller

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. In addition, it needs an address register, a word count register, and a set of address lines. The address register and address lines are used for direct communication with the memory. The address register specifies the address of the memory location. 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 the DMA.

The DMA 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 Read (RD) and Write (WR) inputs are bidirectional. When BG=0 then it is communicate with the DMA register through data bus to read or write to DMA register else BG=1 then CPU sent message to DMA controller to perform operations.

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 (read) and where data to store (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.



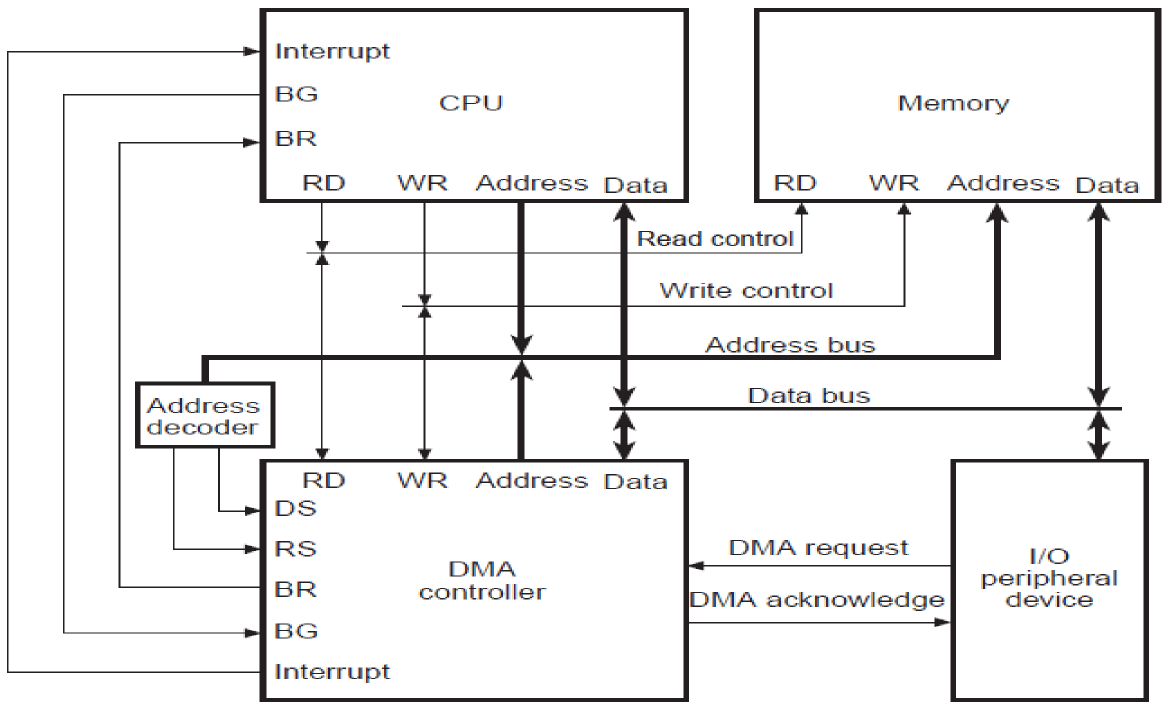
**Block diagram of DMA controller**

## DMA Transfer

The position of the DMA controller among the other components in a computer system consists of DMA controller, CPU, Random-access memory and I/O peripheral devices. The CPU is communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address and data buses as with any interface unit. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can start the transfer between the peripherals and 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 are bidirectional. For each word that is transferred, the DMA increments its address registers and decrements its word count register. If the word count reaches to zero, then stops the DMA and removes the bus request. Else the DMA check the request line coming from the peripheral.

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 with separate peripheral devices. Each channel also has its own address register and word count register within the DMA controller. DMA transfer is very useful in many applications. It is used for fast transfer of information between magnetic disks and memory. The contents of the memory can be transferred to the screen periodically by means of DMA transfer. It is shown in diagram below.



**DMA transfer in a computer system**

## Introduction to Multi Processors:

### Multiprocessor:

- A set of processors connected by a communications network

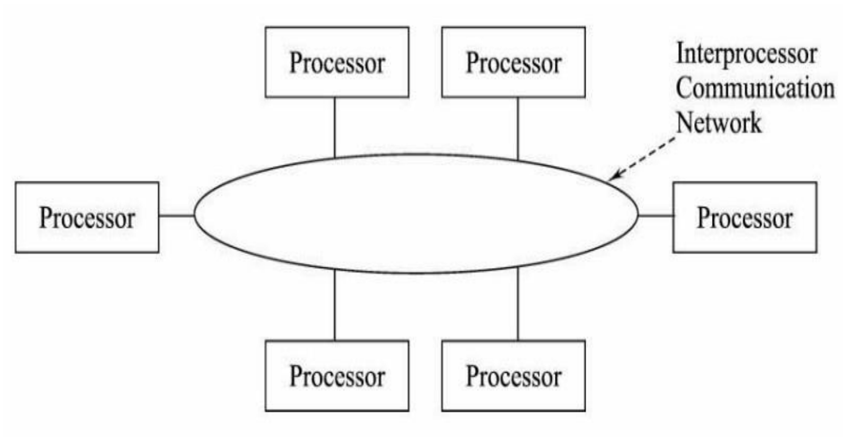


Fig. 5.1 Basic multiprocessor architecture

- A multiprocessor system is an interconnection of two or more CPU’s with memory and input-output equipment.
- Multiprocessors system are classified as multiple instruction stream,multiple data stream systems(MIMD).
- There exists a distinction between multiprocessor and multicomputer that though both support concurrent operations.
- In multicomputer several autonomous computers are connected through a network and they may or may not communicate but in a multiprocessor system there is a single OS Control that provides interaction between processors and all the components of the system to cooperate in the solution of the problem.
- VLSI circuit technology has reduced the cost of the computers to such a low Level that the concept of applying multiple processors to meet system performance requirements has become an attractive design possibility.

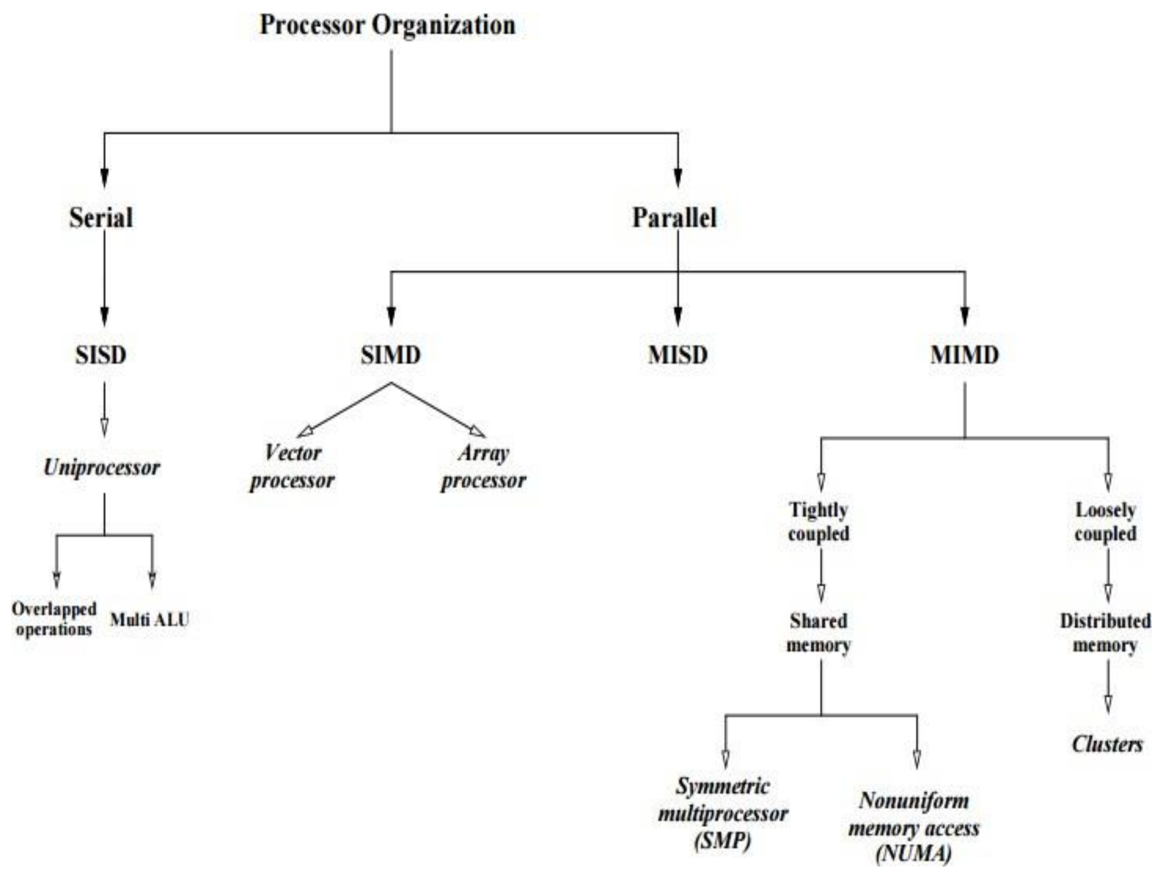


Fig. 5.2 Taxonomy of mono- mulitporcessor organizations

## <u>**Characteristics of Multiprocessors:**</u>

Benefits of Multiprocessing:

1. Multiprocessing increases the reliability of the system so that a failure or error in one part has limited effect on the rest of the system. If a fault causes one processor to fail, a second processor can be assigned to perform the functions of the disabledone.

2. Improved System performance. System derives high performance from thefact that computations can proceed in parallel in one of the two ways:

a) Multiple independent jobs can be made to operate in parallel.

b) A single job can be partitioned into multiple parallel tasks.

This can be achieved in two ways:

- The user explicitly declares that the tasks of the program be executed in parallel

- The compiler provided with multiprocessor s/w that can automaticallydetect parallelism in program. Actually it checks for Data dependency

**COUPLING OF PROCESSORS**

Tightly Coupled System/Shared Memory:

- Tasks and/or processors communicate in a highly synchronized fashion
- Communicates through a common global shared memory
- Shared memory system. This doesn’t preclude each processor from havingits own local memory(cache memory)

Loosely Coupled System/Distributed Memory

- Tasks or processors do not communicate in a synchronized fashion.
- Communicates by message passing packets consisting of an address, thedata content, and some error detection code.
- Overhead for data exchange is high
- Distributed memory system

Loosely coupled systems are more efficient when the interaction between tasks is minimal, whereas tightly coupled system can tolerate a higher degree of interaction between tasks.

Shared (Global) Memory

- A Global Memory Space accessible by all processors
- Processors may also have some local memory

Distributed (Local, Message-Passing) Memory

- All memory units are associated with processors
- To retrieve information from another processor's memory a message must besent there

Uniform Memory

- All processors take the same time to reach all memory locations

Non-uniform (NUMA) Memory

- Memory access is not uniform

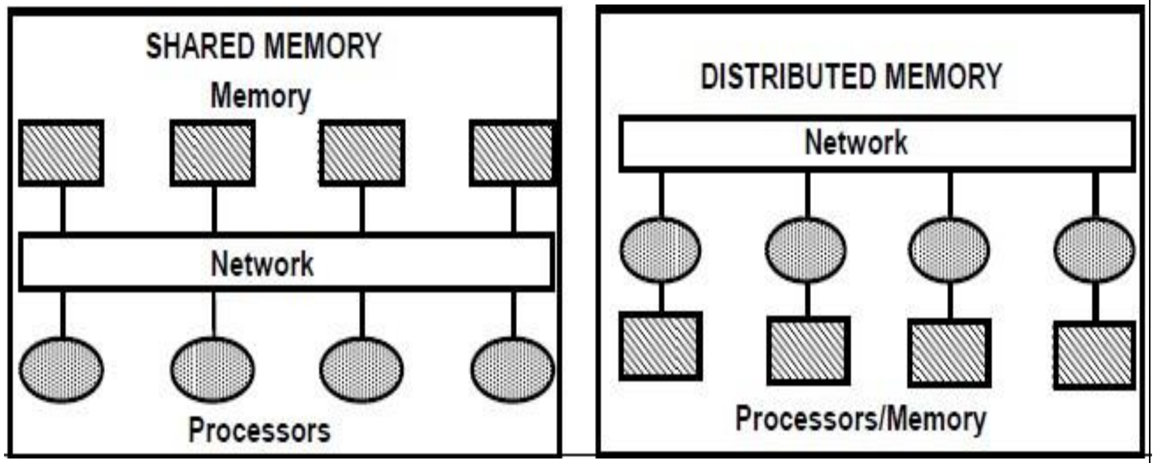


Fig. 5.3 Shared and distributed memory

Shared memory multiprocessor:

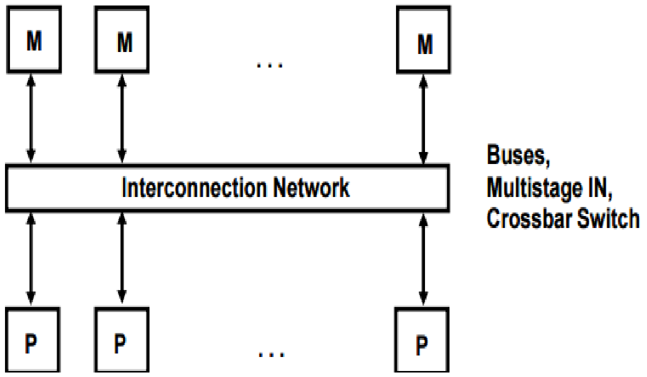


Fig 5.4 Shared memory multiprocessor

Characteristics

- All processors have equally direct access to one large memory addressspace

Limitations

- Memory access latency; Hot spot problem

## Interconnection structures:

The interconnection between the components of a multiprocessor System can have different physical configurations depending n the number of transfer paths that are available between the processors and memory in a shared memory system and among the processing elements in a loosely coupled system.

Some of the schemes are as:

- Time-Shared Common Bus
- Multiport Memory
- Crossbar Switch
- Multistage Switching Network
- Hypercube System

### a. Time shared common Bus

- All processors (and memory) are connected to a common bus or busses
- Memory access is fairly uniform, but not very scalable
- A collection of signal lines that carry module-to-module communication
- Data highways connecting several digital system elements
- Operations of Bus

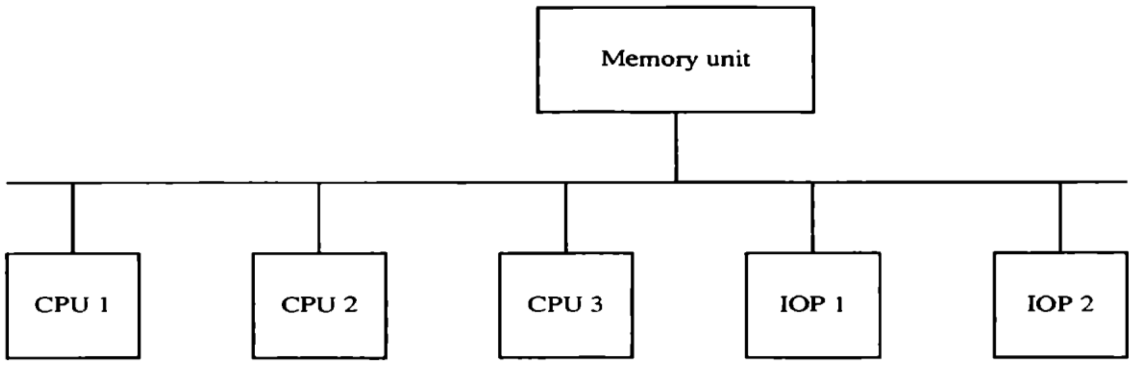


Fig. 5.5 Time shared common bus organization

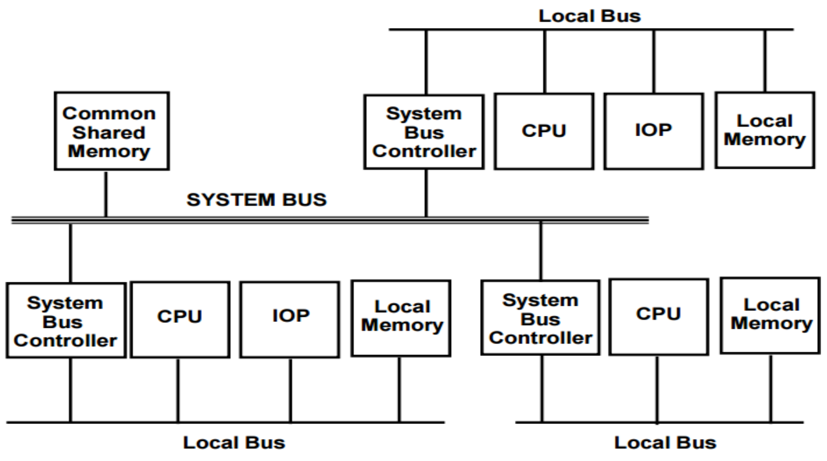


Fig. 5.6 system bus structure for multiprocessor

In the above figure we have number of local buses to its own local memory and to one or more processors. Each local bus may be connected to a CPU, an IOP, or any combinations of processors. A system bus controller links each local bus to a common system bus. The I/O devices connected to the local IOP, as well as the local memory, are available to the local processor. The memory connected to the common system bus is shared by all processors. If an IOP is connected directly to the system bus the I/O devices attached to it may be made available to all processors

Disadvantage.:

- Only one processor can communicate with the memory or anotherprocessor at any given time.
- As a consequence, the total overall transfer rate within the system is limited by the speed of the single path

### b. Multiport Memory:

Multiport Memory Module

- Each port serves a CPU

Memory Module Control Logic

- Each memory module has control logic
- Resolve memory module conflicts Fixed priority among CPUs

Advantages

- The high transfer rate can be achieved because of the multiple paths.

Disadvantages:

- It requires expensive memory control logic and a large number of cablesand connections

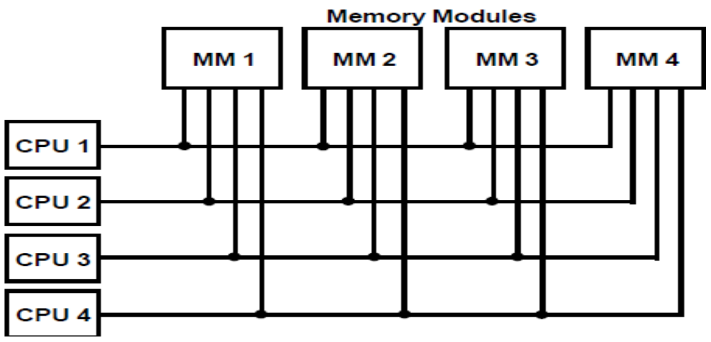
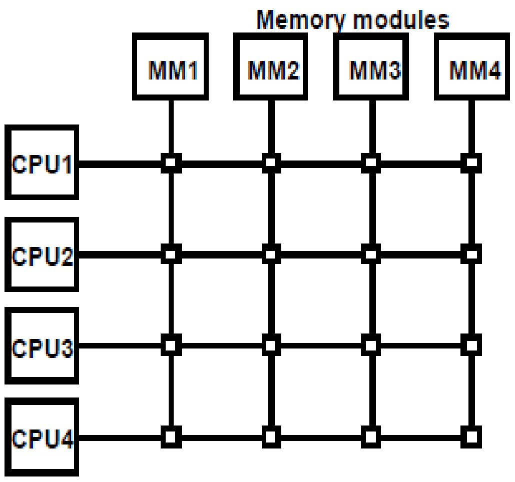
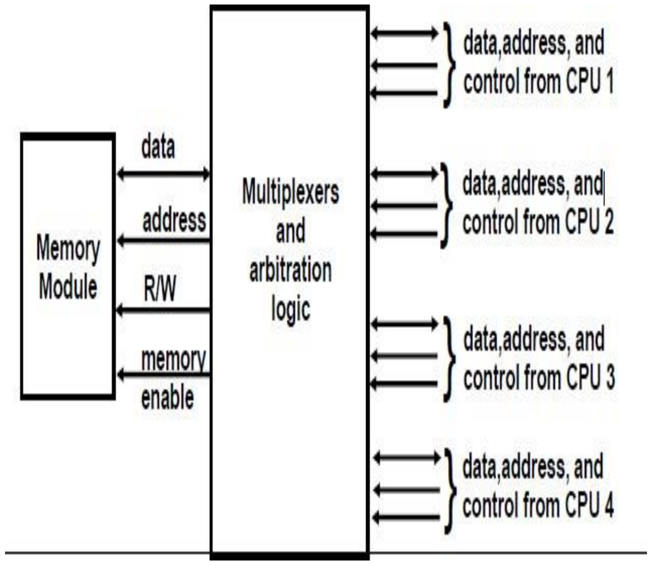


Fig. 5.7 Multiport memory

## c. Crossbar switch:

- Each switch point has control logic to set up the transfer path between a processor and a memory.
- It also resolves the multiple requests for access to the same memory onthe predetermined priority basis.
- Though this organization supports simultaneous transfers from all memory modules because there is a separate path associated with each Module.
- The H/w required to implement the switch can become quite large and complex




Memory Module
data
address
R/W
memory enable
Multiplexers and arbitration logic
data,address, and control from CPU 1
data,address, and control from CPU 2
data,address, and control from CPU 3
data,address, and control from CPU 4

**d.** a) b)

**e.** Fig. 5.8 a) cross bar switch b) Block diagram of cross bar switch

Advantage:

- Supports simultaneous transfers from all memory modules

Disadvantage:

- The hardware required to implement the switch can become quite large andcomplex.

**f. Multistage Switching Network:**

- The basic component of a multi stage switching network is a two-input, two-output interchange switch.

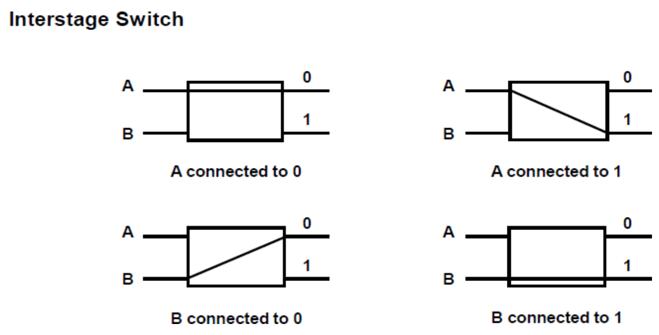


Fig. 5.9 operation of 2X2 interconnection switch

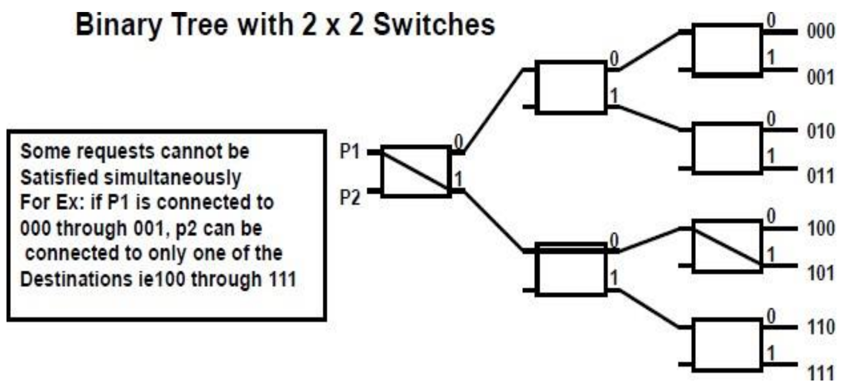
Using the 2x2 switch as a building block, it is possible to build a multistage network to control the communication between a number of sources and destinations.

- To see how this is done, consider the binary tree shown in Fig. below.
- Certain request patterns cannot be satisfied simultaneously.i.e.,

  if P1 □ 000~011, then $P_2$ □ 100~111

-

Fig 5.10 Binary tree with 2x2 switches


Binary Tree with 2 x 2 Switches
Some requests cannot be Satisfied simultaneously For Ex: if P1 is connected to 000 through 001, p2 can be connected to only one of the Destinations ie100 through 111
P1
P2
0
1
000
001
010
011
100
101
110
111

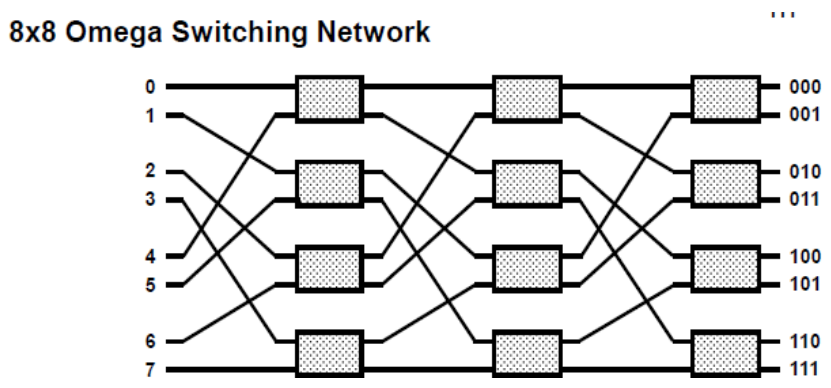


Fig. 5.11 8X8 Omega switching network

- Some request patterns cannot be connected simultaneously. i.e., any twosources cannot be connected simultaneously to destination 000 and 001
- In a tightly coupled multiprocessor system, the source is a processor and the destination is a memory module.
- Set up the path  transfer the address into memory  transfer the data
- In a loosely coupled multiprocessor system, both the source and destination are Processsing elements.
-

## g. Hypercube System:

The hypercube or binary n-cube multiprocessor structure is a loosely coupled system composed of N=2n processors interconnected in an n-dimensional binary cube.

- Each processor forms a node of the cube, in effect it contains not only a CPUbut also local memory and I/O interface.
- Each processor address differs from that of each of its n neighbors by exactlyone bit position.
- Fig. below shows the hypercube structure for n=1, 2, and 3.
- Routing messages through an *n*-cube structure may take from one to *n* links from a source node to a destination node.
- A routing procedure can be developed by computing the exclusive-OR of the source node address with the destination node address.
- The message is then sent along any one of the axes that the resulting binary value will have 1 bits corresponding to the axes on which the two nodes differ.
- A representative of the hypercube architecture is the Intel iPSC computer complex.
- It consists of 128(*n*=7) microcomputers, each node consists of a CPU, a floating point processor, local memory, and serial communication interface units

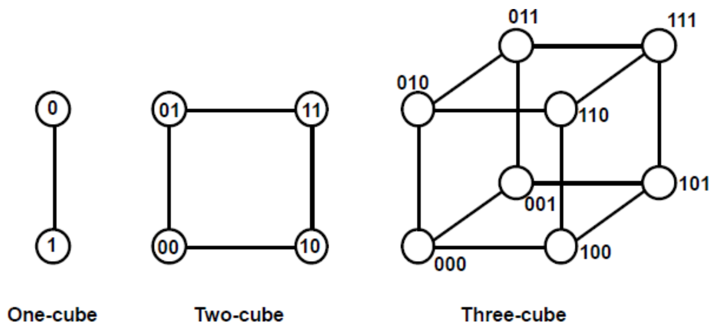


Fig. 5.12 Hypercube structures for n=1,2,3

## **Introduction to pipelining:**

Introduction to Pipelining

Pipelining is a technique of decomposing a sequential process into sub operations, with each sub process being executed in a special dedicated segment that operates concurrently with all other segments.

Example:

Suppose we need to perform multiply and add operation with a stream of numbers

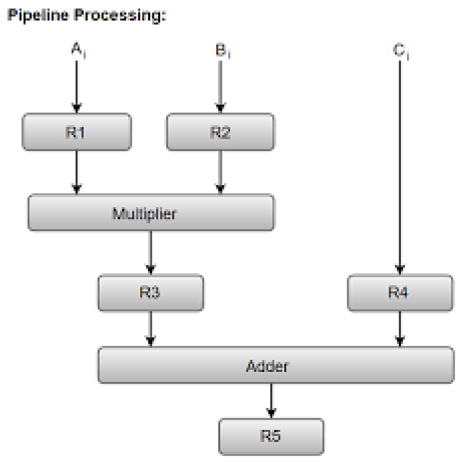
$A_i * B_i + C_i$ for $i = 1, 2, 3, \ldots\ldots., 7$

The operation to be performed on the numbers is decomposed into sub-operations with each sub-operation to be implemented in a segment within a pipeline.

The sub-operations performed in each segment of the pipeline are defined as:

| | |
|---|---|
| $R1 \leftarrow A_i,\quad R2 \leftarrow B_i$ | Input $A_i$, and $B_i$ |
| $R3 \leftarrow R1 * R2,\ R4 \leftarrow C_i$ | Multiply, and input $C_i$ |
| $R5 \leftarrow R3 + R4$ | Add $C_i$ to product |

The following block diagram represents the combined as well as the sub-operations performed in each segment of the pipeline.



Registers R1, R2, R3, and R4 hold the data and the combinational circuits operate in a particular segment. The above figure shows R1 through R5 are registers that receive new data with every clock pulse.

| Clock Pulse Number | Segment 1 | | Segment 2 | | Segment 3 |
|---|---|---|---|---|---|
| | $R1$ | $R2$ | $R3$ | $R4$ | $R5$ |
| 1 | $A_1$ | $B_1$ | — | — | — |
| 2 | $A_2$ | $B_2$ | $A_1 * B_1$ | $C_1$ | — |
| 3 | $A_3$ | $B_3$ | $A_2 * B_2$ | $C_2$ | $A_1 * B_1 + C_1$ |
| 4 | $A_4$ | $B_4$ | $A_3 * B_3$ | $C_3$ | $A_2 * B_2 + C_2$ |
| 5 | $A_5$ | $B_5$ | $A_4 * B_4$ | $C_4$ | $A_3 * B_3 + C_3$ |
| 6 | $A_6$ | $B_6$ | $A_5 * B_5$ | $C_5$ | $A_4 * B_4 + C_4$ |
| 7 | $A_7$ | $B_7$ | $A_6 * B_6$ | $C_6$ | $A_5 * B_5 + C_5$ |
| 8 | — | — | $A_7 * B_7$ | $C_7$ | $A_6 * B_6 + C_6$ |
| 9 | — | — | — | — | $A_7 * B_7 + C_7$ |

Table: Content of Registers in pipeline Example

The Above diagram shows the first clock pulse transfers A1 and B1 intoR1 and R2. The second clock pulse transfers the product of R1 and R2 into R3 and C1 into R4. The same clock pulse transfers A2 and B2 into R1 and R2.The third clock pulse operates on all three segments simultaneously.

It places A3 and B3 into R1 and R2, transfer the product of R1 and R2 into R3, transfers C2 into R4, and places the sum of R3 and R4 into R5. It takes three clock pulses to fill up pipe and retrieve the first output from R5.

**Four Segment pipeline**

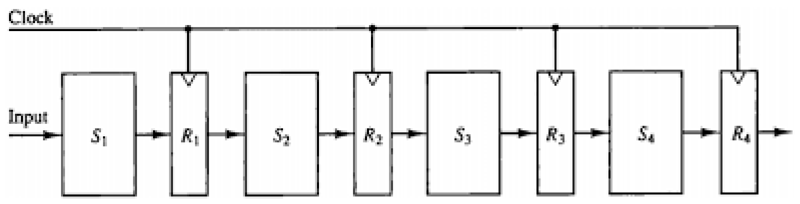
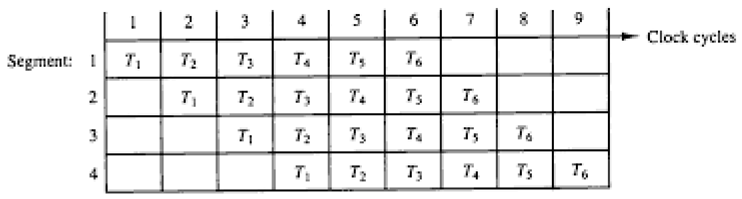


Figure 3 Four-segment pipeline.

The general structure of a four segment pipeline is shown in figure. The operand pass through all four segments in a fixed sequence. Each segment combinational circuit Si that performs a sub operation over the data stream flowing through the pipe.



| Segment: | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| 1 | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | | | |
| 2 | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | | |
| 3 | | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ | |
| 4 | | | | $T_1$ | $T_2$ | $T_3$ | $T_4$ | $T_5$ | $T_6$ |

Clock cycles

Space time diagram for pipeline.

The above space time diagram consists of Horizontal axis display the time in clock cycles and vertical axis gives the segment number. The diagram shows six tasks T1 through T6 executed in four segments. Initially task T1 is handled by segment 1. After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2. Continuing in manner.