

UNIT -III

MEMORY ARCHITECTURE

3.1 MEMORY HIERARCHY

The memory unit is an essential component in any digital computer since it is needed for storing programs and data. A very small computer with a limited application may be able to fulfill its intended task without the need of additional storage capacity. 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 a typical computer. Moreover, most computer users accumulate and continue to accumulate large amounts of data-processing software. Not all accumulated information is needed by the processor at the same time. Therefore, it is more economical to use low-cost storage devices to serve as a backup for storing the information that is not currently used by the CPU.

3.1.1 Main Memory

The memory unit that communicates directly with the CPU is called the main memory.

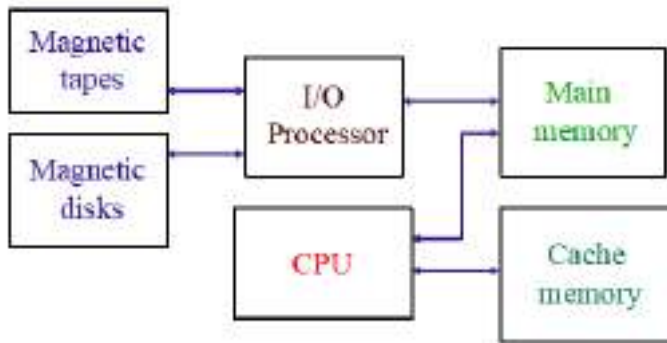
3.1.2 Auxiliary Memory

Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system, programs, large data files, and other backup information. Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

The total memory capacity of a computer can be visualized as being a hierarchy of components. The memory hierarchy system consists of all storage devices employed in a computer system from the slow but high-capacity auxiliary memory to a relatively faster main memory, to an even smaller and faster cache memory accessible to the high-speed processing logic.

Figure 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 through an I/O processor. When programs not residing in main memory are needed by the CPU, they are brought in from auxiliary memory. Programs not currently needed in main memory are transferred into auxiliary memory to provide space for currently used programs and data.



3.1.3 cache memory

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. A technique used to compensate for the mismatch in operating speeds is to employ an extremely fast, small cache between the CPU and main memory whose access time is close to processor logic clock cycle time. The cache is used for storing segments of programs currently being executed in the CPU and temporary data frequently needed in the present calculations.

By making programs and data available at a rapid rate, it is possible to increase the performance rate of the computer.

3.2 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 during the computer operation. The principal technology used for the main memory is based on semiconductor integrated circuits.

3.2.1 RAM

Integrated circuit RAM chips are available in two possible operating modes, static and dynamic. The static RAM 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 tends 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.

3.2.2 ROM

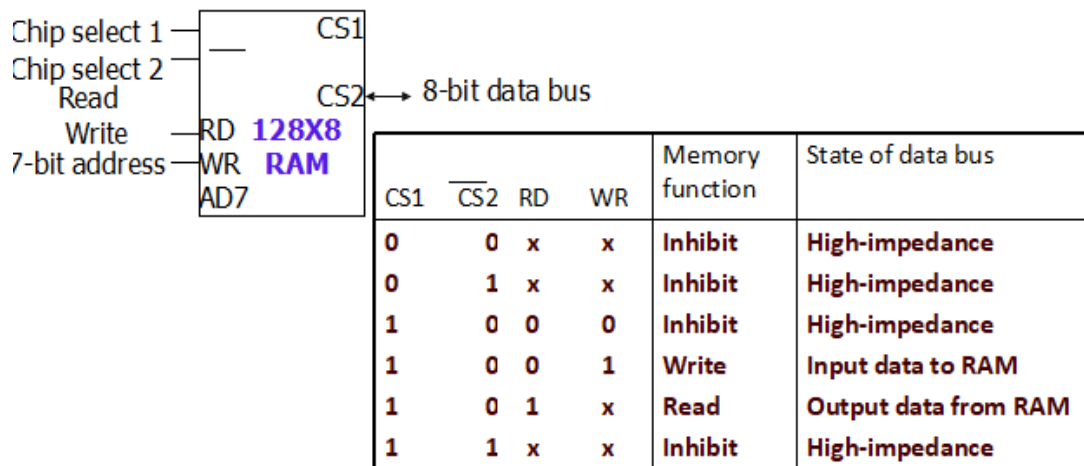
Most of the main memory in a general-purpose computer is made up of RAM integrated circuit chips, but a portion of the memory may be constructed with ROM chips. Originally, RAM was used to refer

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. Among other things, the ROM portion of main memory is needed for 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 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 are available in a variety of sizes. If the memory needed for the computer is larger than the capacity of one chip, it is necessary to combine a number of chips to form the required memory size. To demonstrate the chip interconnection, we will show an example of a 1024 x 8 memory constructed with 128 x 8 RAM chips and 512 x 8 ROM chips.

3.2.3 RAM 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 block diagram of a RAM chip is shown in Fig. 3-2. 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. When the chip is selected, the two binary states in this line specify the two operations of read or write.



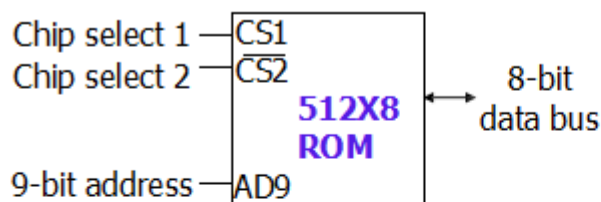
The function table listed in Fig. specifies the operation of the RAM chip. The unit is in operation only when $\text{CS1} = 1$ and $\text{CS2} = 0$. The bar on top of the second select variable indicates that this input is enabled when it is equal to 0. If the chip select inputs are not enabled, or if 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 $\text{CS1} = 1$ and $\text{CS2} = 0$, 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. The RD and WR signals control the memory operation as well as the bus buffers associated with the bidirectional data bus. 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.

For this reason, the diagram specifies a 512-byte ROM, while the RAM has only 128 bytes. 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 $\text{CS1} = 1$ and $\text{CS2} = 0$ 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. Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

3.2.3.1 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. To demonstrate with a particular example, assume that a computer system needs 512 bytes of RAM and 512 bytes of ROM. The RAM and ROM chips to be used are specified in Figs. 3-2 and 3-3. The memory address map for this configuration is shown in Table 3-1. 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 clearly 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. The equivalent hexadecimal address for each chip is obtained from the information under the address bus assignment. The address bus lines are subdivided into groups of four bits each so that each group can be represented with a hexadecimal digit. The first hexadecimal digit represents lines 13 to 16 and is always 0. The next hexadecimal digit represents lines 9 to 12, but lines 11 and 12 are always 0. The range of hexadecimal addresses for each component is determined from the x's associated with it. These x's represent a binary number that can range from an all-0's to an all-1's value.



CS1	CS2	Memory function	State of data bus
0	0	Inhibit	High-impedance
0	1	Inhibit	High-impedance
1	0	Read	Output data from ROM
1	1	Inhibit	High-impedance

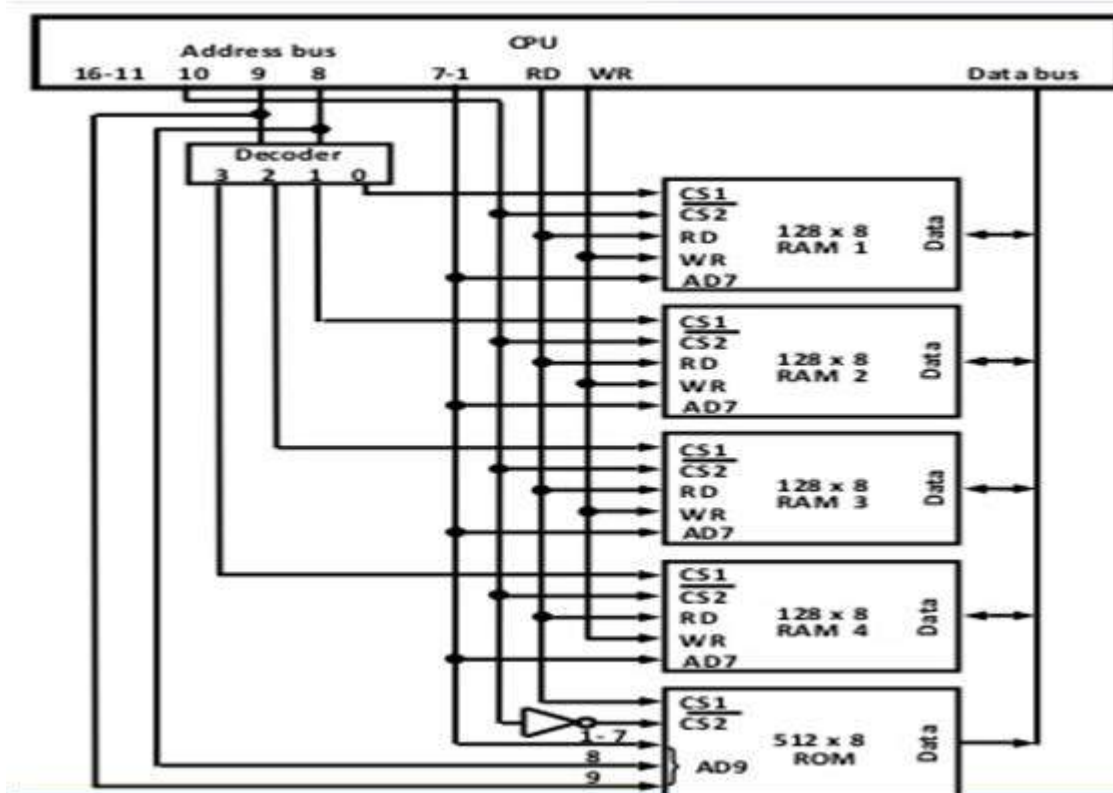
3.2.4 Memory Connection to CPU

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. The connection of memory chips to the CPU is shown in Fig. 12-4. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM. It implements the memory map of Table 12-1. Each RAM receives the seven low-order bits of the address bus to select one of 128 possible bytes. The particular RAM chip selected is determined from lines 8 and 9 in the address bus. This is done through a 2 x 4 decoder whose outputs go to the CS1 inputs in each RAM chip. Thus, when address lines 8 and 9 are equal to 00, the first RAM chip is selected. When 01, the second RAM chip is selected, and so on. The RD and WR outputs from the microprocessor are applied to the inputs of each RAM chip. 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. The other chip select input in the ROM is connected to the RD control line for the ROM chip to be enabled only during a read operation. Address bus lines 1 to 9 are applied to the input

address of ROM without going through the decoder. This assigns addresses 0 to 511 to RAM and 512 to 1023 to ROM. The data bus of the ROM has only an output capability, whereas the data bus connected to the RAMs can transfer information in both directions. The example just shown gives an indication of the interconnection complexity that can exist between memory chips and the CPU. The more chips that are connected, the more external decoders are required for selection among the chips. The designer must establish a memory map that assigns addresses to the various chips from which the required connections are determined.

3.3 AUXILIARY MEMORY

The most common auxiliary memory devices used in computer systems are



tems are magnetic disks and tapes. Other components used, but not as frequently, are magnetic drums, magnetic bubble memory, and optical disks. To understand fully the physical mechanism of auxiliary memory devices one must have a knowledge of magnetics, electronics, and electromechanical systems. Though the physical properties of these storage devices can be quite complex, their logical properties can be characterized and compared by a few parameters. 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

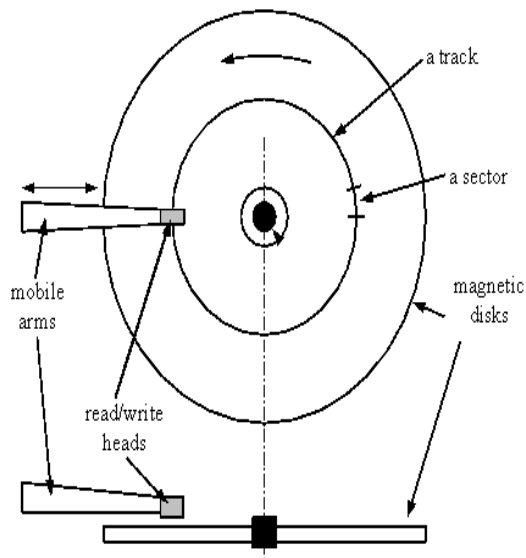
the beginning of the record. Magnetic drums and disks are quite similar in operation. Both consist of high-speed rotating surfaces coated with a magnetic recording medium. The rotating surface of the drum is a cylinder and that of the disk, a round flat plate. The recording surface rotates at uniform speed and is not started or stopped during access operations. Bits are recorded as magnetic spots on the surface as it passes a stationary mechanism called a write head. Stored bits are detected by a change in magnetic field produced by a recorded spot on the surface as it passes through a read head. The amount of surface available for recording in a disk is greater than in a drum of equal physical size. Therefore, more information can be stored on a disk than on a drum of comparable size. For this reason, disks have replaced drums in more recent computers.

3.3.1 Magnetic disks

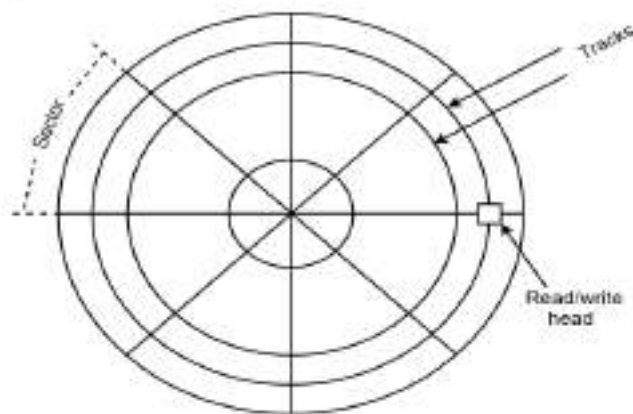
Magnetic disk is a circular plate constructed of metal or plastic coated with magnetized material. Often both sides of the disk are used and several disks may be stacked on one spindle with read/write heads available on each surface. All disks rotate together at high speed and are not stopped or started 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. The subdivision of one disk surface into tracks and sectors is shown in Fig. 3-5.

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 only in 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.

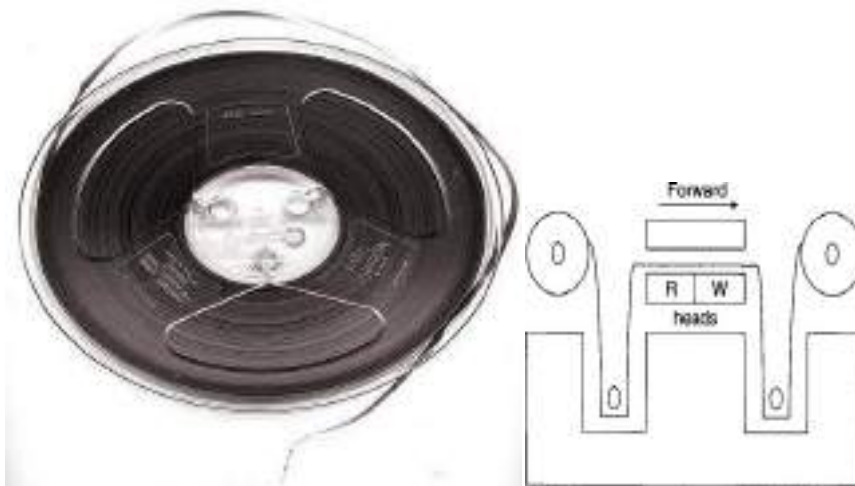


Magnetic disks



3.3.2 Magnetic Tapes

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 be 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.



3.4 ASSOCIATIVE MEMORY

Any data-processing applications require the search of items in a table stored in memory. An assembler program searches the symbol address table in order to extract the symbol's binary equivalent. An account number may be searched in a file to determine the holder's name and account status. The established way to search a table is to store all items where they can be addressed in sequence. The search procedure is a strategy for choosing a sequence of addresses, reading the content of memory at each address, and comparing the information read with the item being searched until a match occurs. The number of accesses to memory depends on the location of the item and the efficiency of the search algorithm. Many search algorithms have been developed to minimize the number of accesses while searching for an item in a random or sequential access memory.

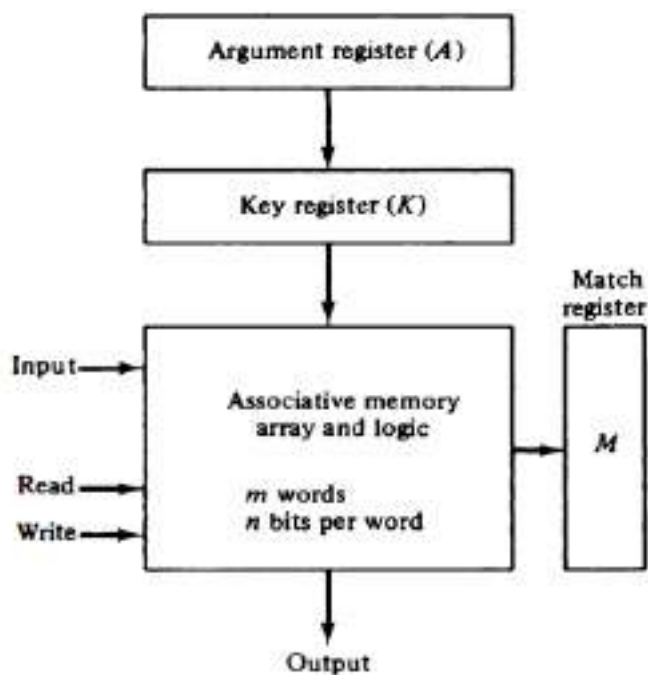
The time required to find an item stored in memory can be reduced considerably if stored data can be identified for access by the content of the data itself rather than by an address. A memory unit accessed by content is called an **associative memory** or **content**

addressable memory (CAM). 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. When a word is to be read from an associative memory, the content of the word, or part of the word, is specified. The memory locates all words which match the specified content and marks them for reading. Because of its organization, the associative memory is uniquely suited to do parallel searches by data association. Moreover, searches can be done on an entire word or on a specific field within a word. An associative memory is more expensive than a random access memory because each cell must have storage capability as well as logic circuits for matching its content with an external argument. For this reason, associative memories are used in applications where the search time is very critical and must be very short.

3.5 Hardware Organisation

The block diagram of an associative memory is shown in Fig. 3-6. 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 1's. Otherwise, only those bits in the argument that have 1's in their corresponding position of the key register are compared. Thus the key provides a mask or identifying piece of information which specifies how the reference to memory is made.



To illustrate with a numerical example, suppose that the argument register A and the key register K have the bit configuration shown below. Only the three leftmost bits of A are compared with memory words because K has 1's in these positions.

A

K

Word 1

Word 2

101 111 100

111 000000

100 111 100

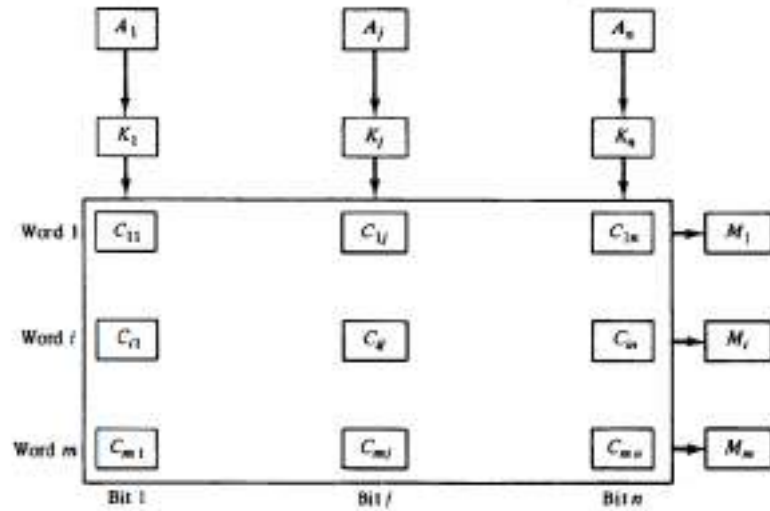
101 000001

no match

match

Word 2 matches the unmasked argument field because the three leftmost bits of the argument and the word are equal. The relation between the memory array and external registers in an associative memory is shown in Fig. 3-7. 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 C₉ is the cell for bit j in word i. A bit A_i in the argument register is compared with all the bits in

column j of the array provided that $K_i = 1$. This is done for all columns $j = 1, 2, \dots, n$. If a match occurs between all the unmasked bits of the argument and the bits in word i , the corresponding bit M_i 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.

Fig: Associative memory of m word, n cells per word.

The internal organization of a typical cell C_{ij} is shown in Fig. 3-8. It consists of a flip-flop storage element F_{ij} and the circuits for reading, writing, and matching the cell. The input bit is transferred into the storage cell during a write operation. The bit stored is read out during a read operation. The match logic compares the content of the storage cell with the corresponding unmasked bit of the argument and provides an output for the decision logic that sets the bit in M_i .

3.3.3 Match Logic

The match logic for each word can be derived from the comparison algorithm for two binary numbers. First, we neglect the 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 $A_i = F_{ij}$ for $j = 1, 2, \dots, n$. 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 $x_i = 1$ if the pair of bits in position j are equal; otherwise, $x_i = 0$. For a word i to be equal to the argument in A we must have all x_i 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

$$M_i = x_1 x_2 x_3 \cdots x_n$$

and constitutes the AND operation of all pairs of matched bits in a word.

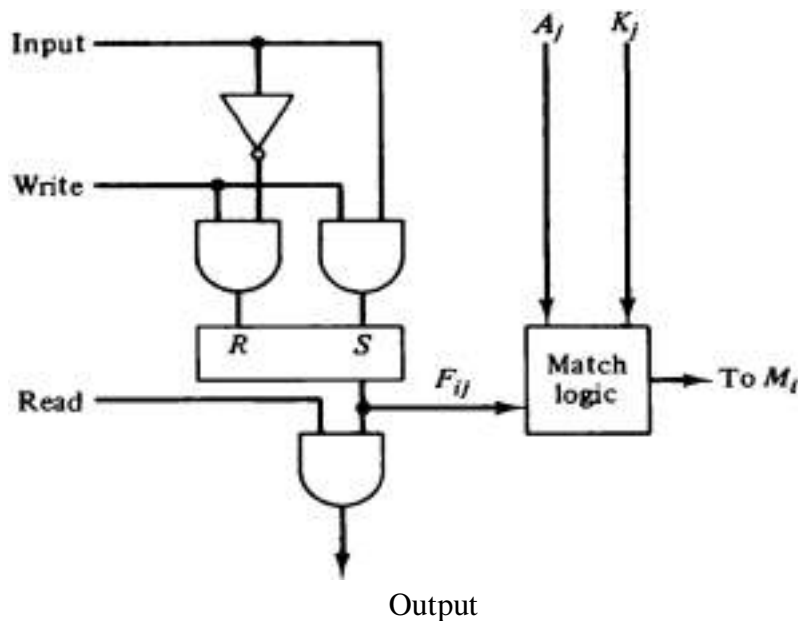


Fig: One cell of associative memory

We now include the key bit K_i in the comparison logic. The requirement is that if $K_i = 0$, the corresponding bits of A_i and f_{ij} need no comparison. Only when $K_i = 1$ must they be compared. This requirement is achieved by ORing each term with K_i , thus:

$$\{ \begin{aligned} X \cdot X + K' &= 1 \text{ if } K_i = 1 \text{ if } K_i = 0 \end{aligned}$$

When $K_i = 1$, we have $K_i' = 0$ and $x_i + 0 = x_i$. When $K_i = 0$, then $K_i' = 1$ and $x_i + 1 = 1$. A term $(x_i + K_i')$ will be in the 1 state if its pair of bits is not compared. This is necessary because each term is ANDed with all other terms so that an output of 1 will have no effect. The comparison of the bits has an effect only when $K_i = 1$.

The match logic for word i in an associative memory can now be expressed by the following Boolean function:

$$M_i = (x_1 + K_1')(x_2 + K_2')(x_3 + K_3') \cdots (x_n + K_n')$$

Each term in the expression will be equal to 1 if its corresponding $K_i = 0$. If $K_i = 1$, the term will be either 0 or 1 depending on the value of X_i . A match will occur and M_i will be equal to 1 if all terms are equal to 1. If we substitute the original definition of x_i , the Boolean function above can be expressed as follows:

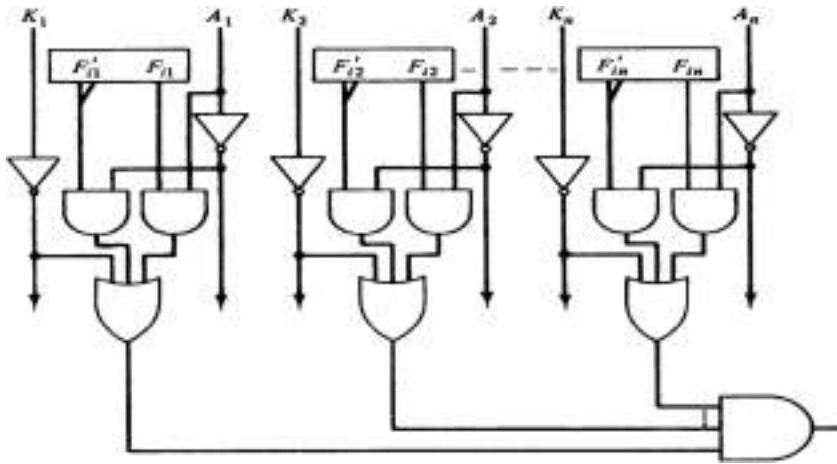
"

$$M_i = \prod (A_i f_{ij} + K_i) = 1$$

where \prod is a product symbol designating the AND operation of all n terms. We need m such functions, one for each word $i = 1, 2, 3, \dots, m$. The circuit for matching one word is shown in Fig. 12-9. Each cell requires two AND gates and one OR gate. The inverters for A_i and K_i are needed once for each column and are used for all bits in the column. The output of all OR gates in the cells of the same word go to the input of a common AND gate to generate the match signal for M_i . M_i will be logic 1 if a match occurs and 0 if no match occurs. Note that if the key register contains all 0's, output M_i will be a 1 irrespective of the value of A or the word. This occurrence must be avoided during normal operation.

3.3.4 read operation

If more than one word in memory matches the unmasked argument field, all the matched words will have 1's in the corresponding bit position of the match register. It is then necessary to scan the bits of the match register one at a time. The matched words are read in sequence by applying a read signal to each word line whose corresponding M, bit is a 1.



Fog :Match logic for one word of associative memory.

In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field. By connecting output M directly to the read line in the same word position (instead of the M register), the content of the matched word will be presented automatically at the output lines and no special read command signal is needed. Furthermore, if we exclude words having a zero content, an all-zero output will indicate that no match occurred and that the searched item is not available in memory.

3.4.5 write operation

An associative memory must have a write capability for storing the information to be searched. Writing in an associative memory can take different forms, depending on the application. If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence. This will make the device a random access memory for writing and a content addressable memory for reading. The advantage here is that the address for input can be decoded as in a random access memory. Thus instead of having m address lines, one for each word in memory, the number of address lines can be reduced by the decoder to d lines, where $m = 2^d$.

If unwanted words have to be deleted and new words inserted one at a time, there is a need for a special register to distinguish between active and inactive words. This register, sometimes called a tag register, would have as many bits as there are words in the memory. For every active word stored in memory, the corresponding bit in the tag register is set to 1. A word is deleted from memory by clearing its tag bit to 0. Words are stored in memory by scanning the tag register until the first 0 bit is encountered. This gives the first available inactive word and a position for writing a new word. After the new word is stored in memory it is made active by setting its tag bit to 1. An unwanted word when deleted from memory can be cleared to all 0's if this value is used to specify an empty location. Moreover, the words that have a tag bit of 0 must be masked (together with the K_i bits) with the argument word so that only active words are compared.

3.5 CACHE MEMORY

Analysis of a large number of typical programs has shown that the references to 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. The reason for this property may be understood considering that a typical computer program flows in a straight-line fashion with program loops and subroutine calls encountered frequently. 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, thus reducing the total execution time of the program. Such a fast small memory is referred to as a cache memory. It is placed between the CPU and main memory as illustrated in Fig. 3-1. The cache memory access time is less than the access time of main memory by a factor of 5 to 10. 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 (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.

The performance of cache memory is frequently measured in terms of a quantity called hit ratio. When the CPU refers to memory and finds the word in cache, it is said to produce a hit. If the word is not found in cache, it is in main memory and it counts as a miss. The ratio of the number of hits divided by the total CPU references to memory (hits plus misses) is the hit ratio. The hit ratio is best measured experimentally by running representative programs in the computer and measuring the number of hits and misses during a given interval of time. 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 cache. 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. For example, a computer with cache access time of 100 ns, a main memory access time of 1000 ns, and a hit ratio of 0.9 produces an average access time of

200 ns. This is a considerable improvement over a similar computer without a cache memory, whose access time is 1000 ns.

The basic characteristic of cache memory is its fast access time. Therefore, very little or no time must be wasted when searching for words in the cache. 3.5 mapping 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.

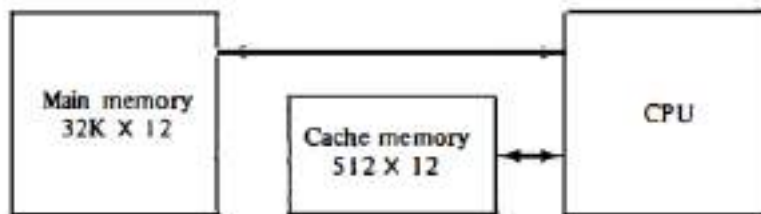


Fig: Example of cache memory

3.5.1 Associative mapping

The fastest and most flexible cache organization uses an associative memory. This organization is illustrated in Fig. 12-1 1. 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. The decision as to what pair is replaced is determined from the replacement algorithm that the designer chooses for the cache. A simple procedure is to replace cells of the cache in round-robin order whenever a new word is requested from main memory. This constitutes a **first-in first-out (FIFO) replacement policy**.

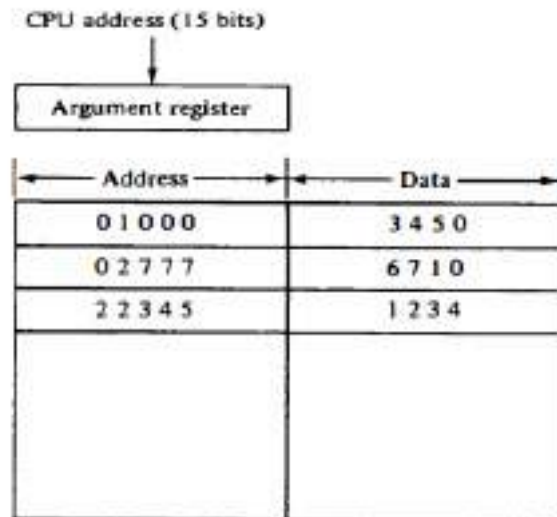


Fig : Associative mapping cache (all numbers in octal).

3.5.2 Direct Mapping

Associative memories are expensive compared to random-access memories because of the added logic associated with each cell. The possibility of using a random-access memory for the cache is investigated in Fig. 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 form the tag field. The figure shows that main memory needs an address that includes both the tag and the index bits. The number of bits in the index field is equal to the number of address bits required to access the cache memory.

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. The internal organization of the words in the cache memory is as shown in Fig. 12-13(b). Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache.

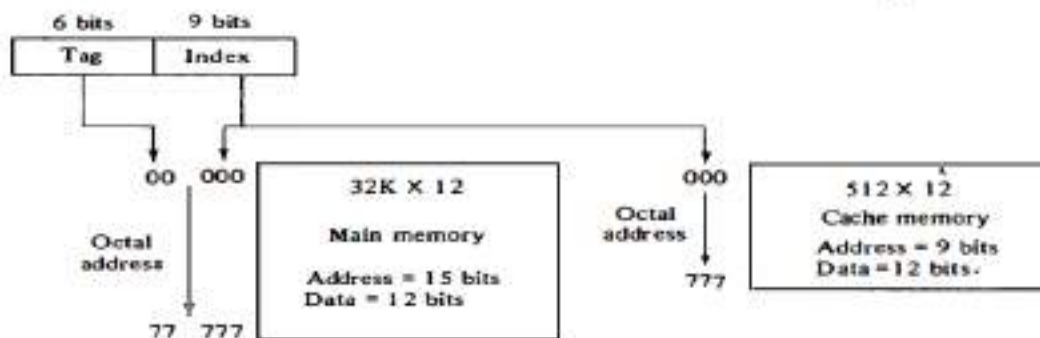


Fig: Addressing relationships between main and cache memories.

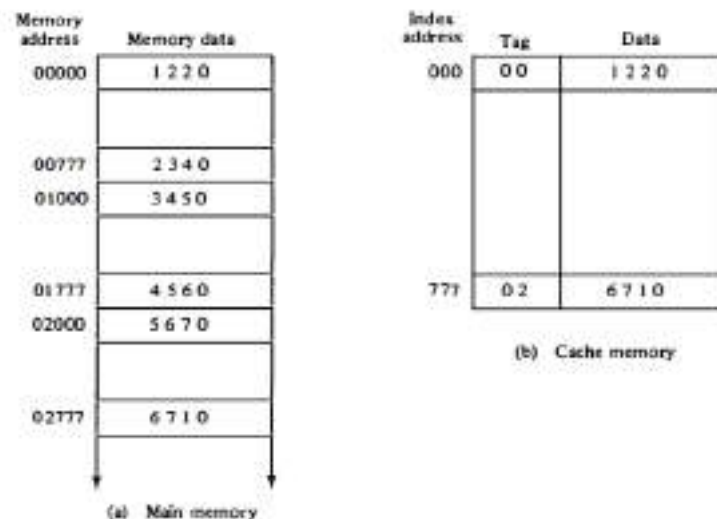


Fig: Direct mapping cache organization.

The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If there is no match, there is a miss and the required word is read from main memory. It is then stored in the cache together with the new tag, replacing the previous value. The disadvantage of direct mapping is that the hit ratio can drop considerably if two or more words whose addresses have the same index but different tags are accessed repeatedly. However, this possibility is minimized by the fact that such words are relatively far apart in the address range (multiples of 512 locations in this example.) To see how the direct-mapping organization operates, consider the numerical example shown in Fig. 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 word at index address 000 is then replaced with a tag of 02 and data of 5670. The direct-mapping example just described uses a block size of one word.

The same organization but using a block size of B words is shown in Fig.

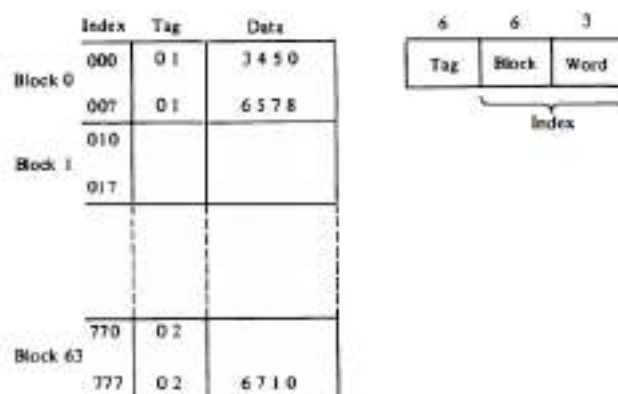


Fig : Direct mapping cache with block size of 8 words.

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 $64 \times 8 = 512$. 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. Although this takes extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.

3.5.3 Set-Associative Mapping

It was mentioned previously that the disadvantage of direct mapping is that two words with the same index in their address but with different tag values cannot reside in cache memory at the same time. 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. An example of a set-associative cache organization for a set size of two is shown in Fig. 12-15. Each index address refers to two data words and their associated tags. Each tag requires six bits and each data word has 12 bits, so the word length is $2(6 + 12) = 36$ bits. An index address of nine bits can accommodate 512 words. Thus the size of cache memory is 512×36 . It can accommodate 1024 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.

Index	Tag	Data	Tag	Data
000	01	3450	02	5670
777	02	6710	00	2340

Fig : Two way set associative mapping cache

The octal numbers listed in Fig. 12-15 are with reference to the main memory contents illustrated in Fig. 12-13(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. However, an increase in the set size increases the number of bits in words of cache and requires more

complex comparison logic. When a miss occurs in a set-associative cache and the set is full, it is necessary to replace one of the tag-data items with a new value.

The most common replacement algorithms used are: random replacement, first-in, first out (FIFO), and least recently used (LRU). With the random replacement policy the control chooses one tag-data item for replacement at random. The FIFO procedure selects for replacement the item that has been in the set the longest. The **LRU algorithm** selects for replacement the item that has been least recently used by the CPU. Both FIFO and LRU can be implemented by adding a few extra bits in each word of cache.

3.5.4 Writing into Cache

An important aspect of cache organization is concerned with memory write requests. 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 updated in parallel if it contains the word at the specified address. This is called the **write-through method**. This method has the advantage that the main memory always contains the same data as the cache. This characteristic is important in systems with direct memory access transfers. It ensures that the data residing in main memory are valid at all times so that an I/O device communicating through DMA would receive the most recent updated data.

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 reason for the write-back method is that during the time a word resides in the cache, it may be updated several times; however, as long as the word remains in the cache, it does not matter whether the copy in main memory is out of date, since requests from the word are filled from the cache. It is only when the word is displaced from the cache that an accurate copy need be rewritten into main memory. Analytical results indicate that the number of memory writes in a typical program ranges between 10 and 30 percent of the total references to memory.

3.6 VIRTUAL MEMORY

In a memory hierarchy system, programs and data are first stored in auxiliary memory. Portions of a program or data are brought into main memory as they are needed by the CPU. 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.

3.6.1 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 in computers with virtual memory.

As an illustration, 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 $220 = 1024K$ 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. 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. (Remember that for efficient transfers, auxiliary storage moves an entire record to the main memory.)

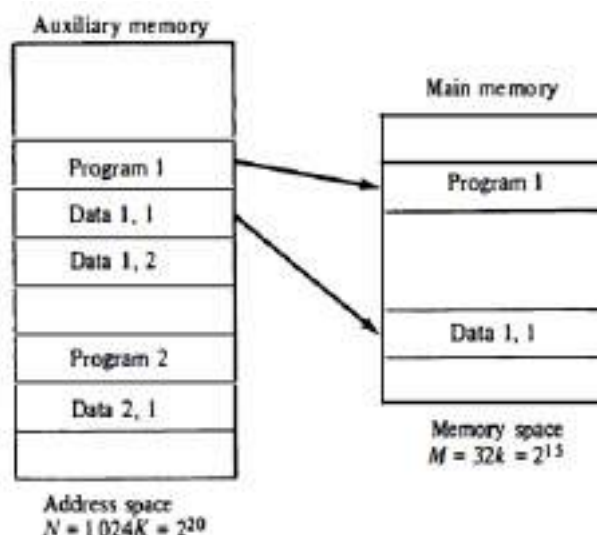


Fig: Relation between address and memory space in a virtual memory system.

A table is then needed, as shown in Fig. 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.

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 program running at half speed. A third alternative is to use an associative memory as explained below.

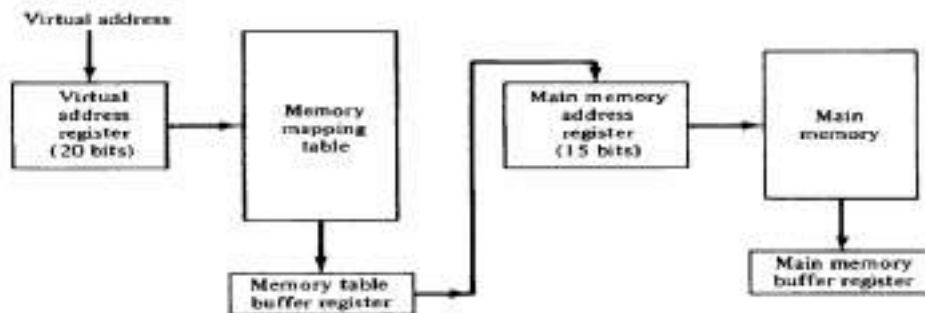


Fig: Memory table for mapping a virtual address.

3.5.6 Address Mapping Using Pages

The table implementation of the address mapping is simplified if 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. For example, if a page or block consists of 1K words, then, using the previous example, address space is divided into 1024 pages and main memory is divided into 32 blocks. Although both a page and a block are split into groups of 1K words, a page refers to the organization of address space, while a block refers to the 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 a block.

Consider a 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. 12-18. 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 2^p 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. 3-18, 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. 3-19. 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. A 0 in the presence bit indicates that this page is not available in main memory. The CPU references a word in memory with a virtual address of 13 bits. The three 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. 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.

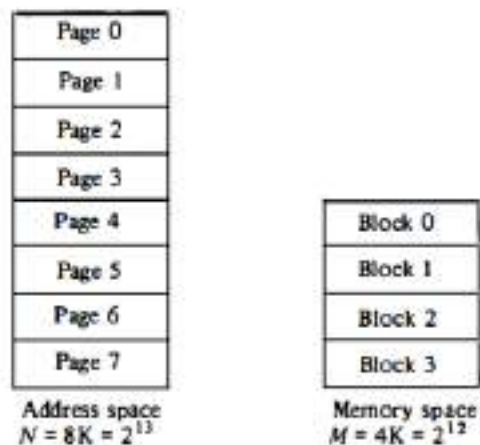


Fig: Address space and memory space split into groups of 1K words.

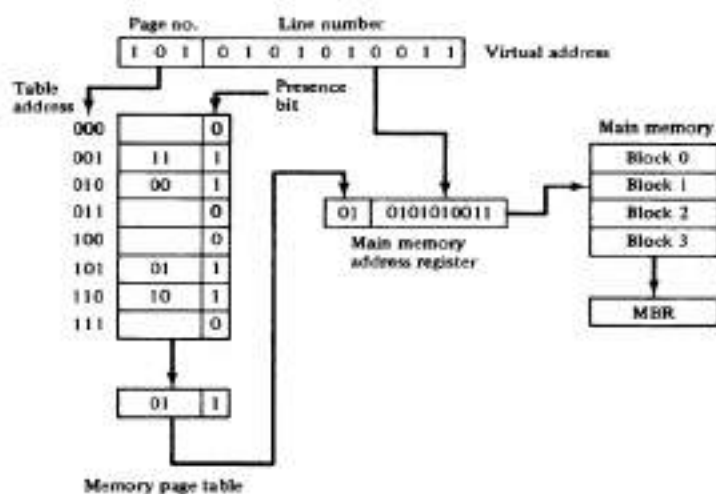


Fig: Memory table in a paged system.

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. 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.

3.5.7 Associative Memory Page Table

A random-access memory page table is inefficient with respect to storage utilization. In the example of Fig. 12-19 we observe that eight words of memory are needed, one for each page, but at least four

words will always be marked empty because the main memory cannot accommodate more than four blocks. 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 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 a 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 its corresponding block number. 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.

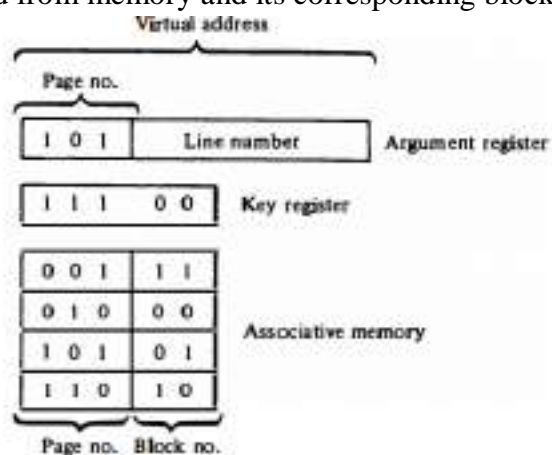


Fig: An associative memory page table.

Consider again the case of eight pages and four blocks as in the example. We replace the random access memory-page table with an associative memory of four words as shown in Fig. 12-20. 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.

3.7 Page Replacement

A virtual memory system is a combination of hardware and software techniques. 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.

The hardware mapping mechanism and the memory management software together constitute the architecture of a virtual 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 VO operation, the operating system assigns this task to the VO 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 remove 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 page 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 from memory too frequently.

INPUT-OUTPUT ORGANIZATION

Peripheral Devices

The Input / output organization of computer depends upon the size of computer and the peripherals connected to it. The I/O Subsystem of the computer, provides an efficient mode of communication between the central system and the outside environment

The most common input output devices are:

- i) Monitor
- ii) Keyboard
- iii) Mouse
- iv) Printer
- v) Magnetic tapes

INPUT - OUTPUT INTERFACE

Input Output Interface provides a method for transferring information between internal storage and external I/O devices.

Peripherals connected to a computer need special communication links for interfacing them with the central processing unit.

The purpose of communication link is to resolve the differences that exist between the central computer and each peripheral.

The Major Differences are:-

1. Peripherals are electro mechanical and electromagnetic devices and CPU and memory are electronic devices. Therefore, a conversion of signal values may be needed.
2. The data transfer rate of peripherals is usually slower than the transfer rate of CPU and consequently, a synchronization mechanism may be needed.
3. Data codes and formats in the peripherals differ from the word format in the CPU and memory.
4. The operating modes of peripherals are different from each other and must be controlled so as not to disturb the operation of other peripherals connected to the CPU.

To Resolve these differences, computer systems include special hardware components between the CPU and Peripherals to supervise and synchronizes all input and out transfers

- These components are called Interface Units because they interface between the processor bus and the peripheral devices.

I/O BUS and Interface Module

It defines the typical link between the processor and several peripherals. The I/O Bus consists of data lines, address lines and control lines. The I/O bus from the processor is attached to all peripherals interface. To communicate with a particular device, the processor places a device address on address lines. Each Interface decodes the address and control received from the I/O bus, interprets them for peripherals and provides signals for the peripheral controller. It is also synchronizes the data flow and supervises the transfer between peripheral and processor. Each peripheral has its own controller. For example, the printer controller controls the paper motion, the print timing. The control lines are referred as I/O command.

The commands are as following:

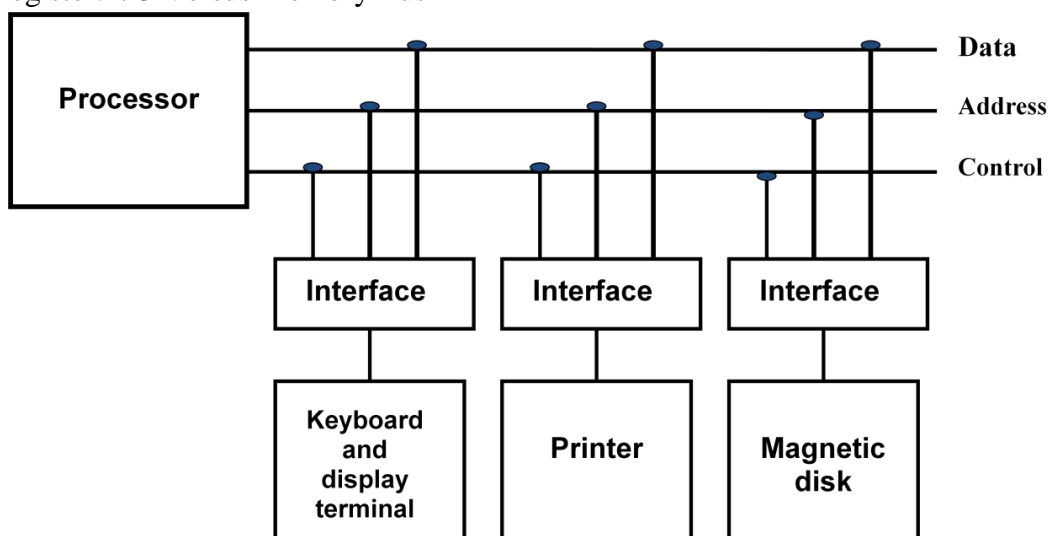
Control command- A control command is issued to activate the peripheral and to inform it what to do.

Status command- A status command is used to test various status conditions in the interface and the peripheral.

Data Output command- A data output command causes the interface to respond by transferring data from the bus into one of its registers.

Data Input command- The data input command is the opposite of the data output.

In this case the interface receives an item of data from the peripheral and places it in its buffer register. I/O Versus Memory Bus



Connection of I/O bus to input-output devices

To communicate with I/O, the processor must communicate with the memory unit. Like the I/O bus, the memory bus contains data, address and read/write control lines. There are 3 ways that computer buses can be used to communicate with memory and I/O:

- Use two Separate buses , one for memory and other for I/O.

- ii. Use one common bus for both memory and I/O but separate control lines for each.
- iii. Use one common bus for memory and I/O with common control lines. I/O Processor

In the first method, the computer has independent sets of data, address and control buses one for accessing memory and other for I/O. This is done in computers that provide a separate I/O processor (IOP). The purpose of IOP is to provide an independent pathway for the transfer of information between external device and internal memory.

ASYNCHRONOUS DATA TRANSFER

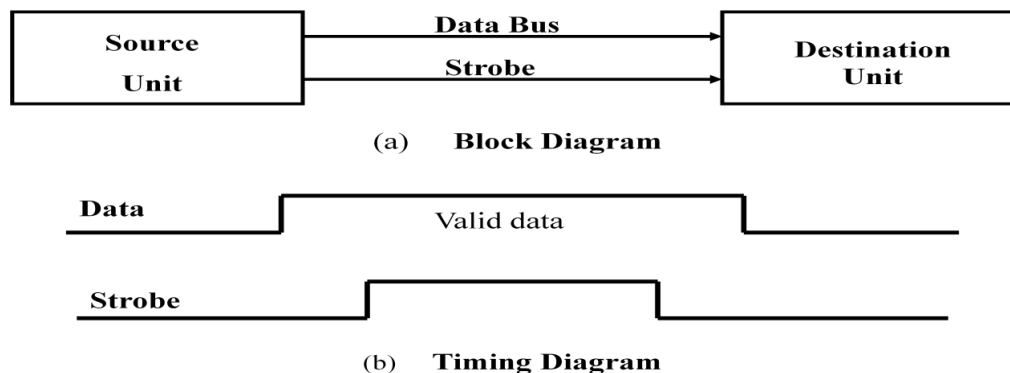
This Scheme is used when speed of I/O devices does not match with microprocessor, and timing characteristics of I/O devices is not predictable. In this method, processes initiate the device and check its status. As a result, CPU has to wait till I/O device is ready to transfer data. When device is ready CPU issues instruction for I/O transfer. In this method two types of techniques are used based on signals before data transfer.

- i. Strobe Control
- ii. Handshaking

Strobe Control

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

Data Transfer Initiated by Source Unit:



Source-Initiated strobe for Data Transfer

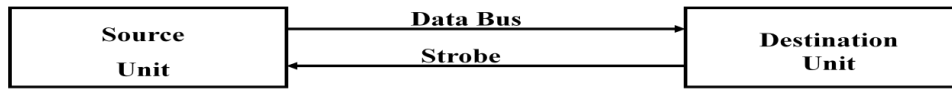
In the block diagram fig. (a), the data bus carries the binary information from source to destination unit. Typically, the bus has multiple lines to transfer an entire byte or word. The strobe is a single line that informs the destination unit when a valid data word is available.

The timing diagram fig. (b) the source unit first places the data on the data bus. The information on the data bus and strobe signal remains in the active state to allow the destination unit to receive the data.

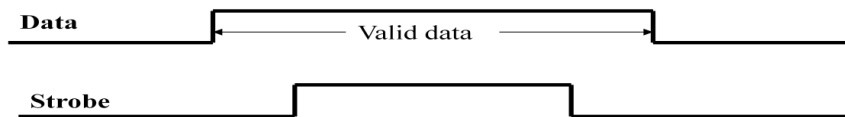
Data Transfer Initiated by Destination Unit:

In this method, the destination unit activates the strobe pulse, to informing the source to provide the data. The source will respond 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. When accepted the destination unit then disables the strobe and the source unit removes the data from the bus.



(a) Block Diagram



(b) Timing Diagram

Destination-Initiated strobe for Data Transfer

Disadvantage of Strobe Signal :

The disadvantage of the strobe method is that, the source unit initiates the transfer has no way of knowing whether the destination unit has actually received the data item that was placed in the bus. Similarly, a destination unit that initiates the transfer has no way of knowing whether the source unit has actually placed the data on bus. The Handshaking method solves this problem.

Handshaking method:

The handshaking method solves the problem of strobe method by introducing a second control signal that provides a reply to the unit that initiates the transfer.

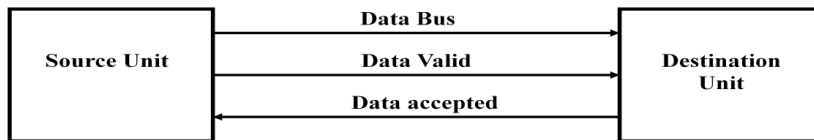
Principle of Handshaking:

The basic principle of the two-wire handshaking method of data transfer is as follow:

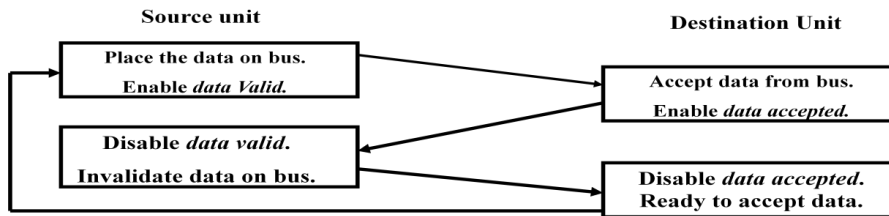
One control line is in the same direction as the data flows in the bus from the source to destination. It is used by source unit to inform the destination unit whether there is a valid data in the bus. The other control line is in the other direction from the destination to the source. It is used by the destination unit to inform the source whether it can accept the data. The sequence of control during the transfer depends on the unit that initiates the transfer.

Source Initiated Transfer using Handshaking:

The sequence of events shows four possible states that the system can be at any given time. The source unit initiates the transfer by placing the data on the bus and enabling its *data valid* signal. The *data accepted* signal is activated by the destination unit after it accepts the data from the bus. The source unit then disables its *data accepted* signal and the system goes into its initial state.



(a) Block Diagram

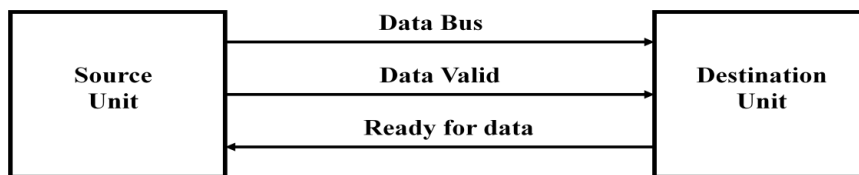


(b) Sequence of events

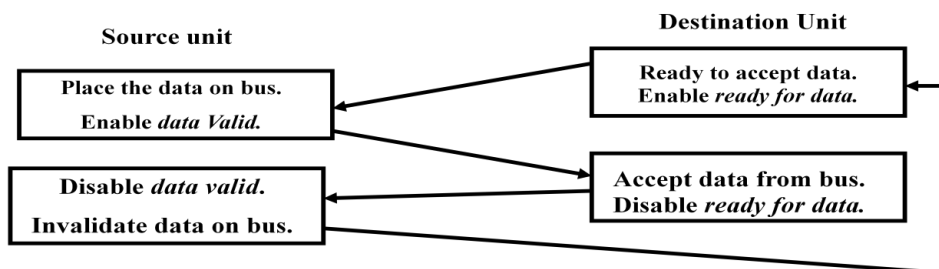
Destination Initiated Transfer Using Handshaking:

The name of the signal generated by the destination unit has been changed to *ready for data* to reflect its new meaning. The source unit in this case does not place data on the bus until after it receives the *ready for data* signal from the destination unit. From there on, the handshaking procedure follows the same pattern as in the source initiated case.

The only difference between the Source Initiated and the Destination Initiated transfer is in their choice of Initial state.



(a) Block Diagram



(b) Sequence of events

Destination-Initiated transfer using Handshaking

Advantage of the Handshaking method:

- The Handshaking scheme provides degree of flexibility and reliability because the successful completion of data transfer relies on active participation by both units.
- If any of one unit is faulty, the data transfer will not be completed. Such an error can be detected by means of a *Timeout mechanism* which provides an alarm if the data is not completed within time.

Asynchronous Serial Transmission:

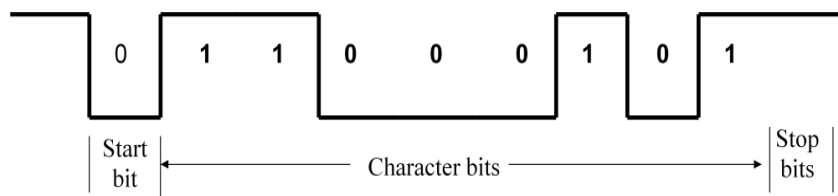
The transfer of data between two units is serial or parallel. In parallel data transmission, n bit in the message must be transmitted through n separate conductor path. In serial transmission, each bit in the message is sent in sequence one at a time.

Parallel transmission is faster but it requires many wires. It is used for short distances and where speed is important. Serial transmission is slower but is less expensive.

In Asynchronous serial transfer, each bit of message is sent a sequence at a time, and binary information is transferred only when it is available. When there is no information to be transferred, line remains idle. In this technique each character consists of three points:

- Start bit
- Character bit
- Stop bit

- Start Bit- First bit, called start bit is always zero and used to indicate the beginning character.
- Stop Bit- Last bit, called stop bit is always one and used to indicate end of characters. Stop bit is always in the 1- state and frame the end of the characters to signify the idle or wait state.
- Character Bit- Bits in between the start bit and the stop bit are known as character bits. The character bits always follow the start bit.



Asynchronous Serial Transmission

Serial Transmission of Asynchronous is done by two ways:

- a) Asynchronous Communication Interface
- b) First In First out Buffer

Asynchronous Communication Interface:

It works as both a receiver and a transmitter. Its operation is initialized by CPU by sending a byte to the control register. The transmitter register accepts a data byte from CPU through the data bus and transferred to a shift register for serial transmission. The receive portion receives information into another shift register, and when a complete data byte is received it is transferred to receiver register. CPU can select the receiver register to read the byte through the data bus. Data in the status register is used for input and output flags.

First In First Out Buffer (FIFO):

A First In First Out (FIFO) Buffer is a memory unit that stores information in such a manner that the first item is in 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. When placed between two units, the FIFO can accept data from the source unit at one rate, rate of transfer and deliver the data to the destination unit at another rate. If the source is faster than the destination, the FIFO is useful for source data arrive in bursts that fills out the buffer. FIFO is useful in some applications when data are transferred asynchronously.

MODES OF DATA TRANSFER

Transfer of data is required between CPU and peripherals or memory or sometimes between any two devices or units of your computer system. To transfer a data from one unit to another one should be sure that both units have proper connection and at the time of data transfer the receiving unit is not busy. This data transfer with the computer is Internal Operation.

All the internal operations in a digital system are synchronized by means of clock pulses supplied by a common clock pulse Generator. The data transfer can be

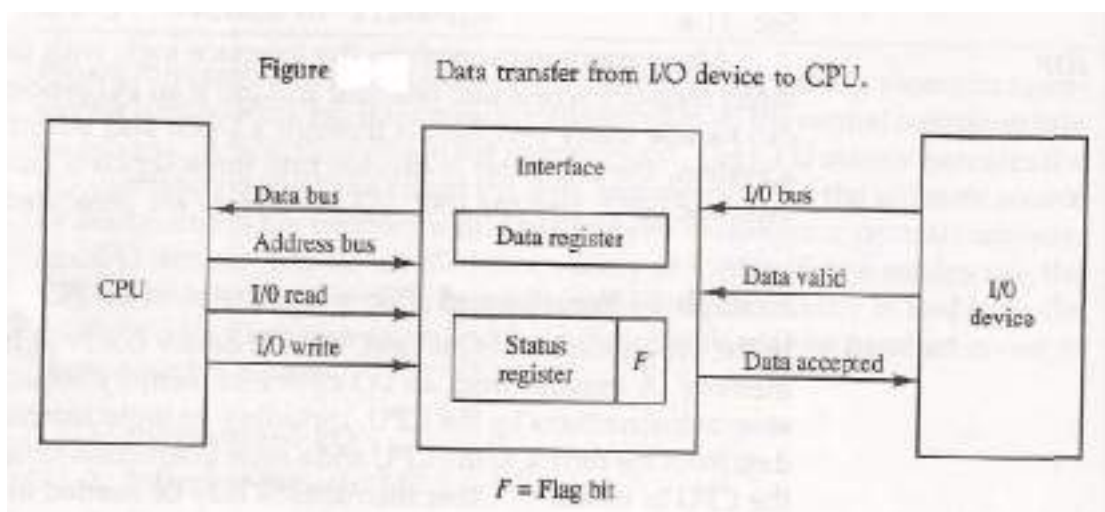
- i. Synchronous or
- ii. Asynchronous

When both the transmitting and receiving units use same clock pulse then such a data transfer is called Synchronous process. On the other hand, if there is not concept of clock pulses and the sender operates at different moment than the receiver then such a data transfer is called Asynchronous data transfer. The data transfer can be handled by various modes. Some of the modes use CPU as an intermediate path, others transfer the data directly to and from the memory unit and this can be handled by 3 following ways:

- i. Programmed I/O
- ii. Interrupt-Initiated I/O
- iii. Direct Memory Access (DMA)

PROGRAMMED I/O

In this mode of data transfer the operations are the results in I/O instructions which is a part of computer program. Each data transfer is initiated by a instruction in the program. Normally the transfer is from a CPU register to peripheral device or vice-versa. Once the data is initiated the CPU starts monitoring the interface to see when next transfer can made. The instructions of the program keep close

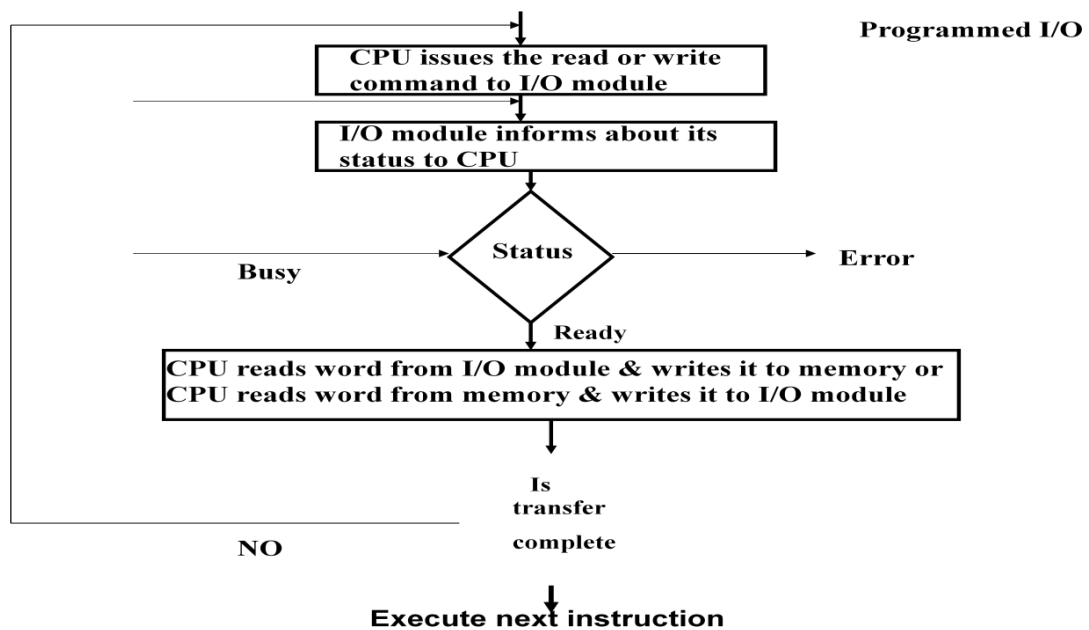


tabs on everything that takes place in the interface unit and the I/O devices.

- The transfer of data requires three instructions:

1. Read the status register.
2. Check the status of the flag bit and branch to step 1 if not set or to step 3 if set.
3. Read the data register.

In this technique CPU is responsible for executing data from the memory for output and storing data in memory for executing of Programmed I/O as shown in Flowchart:-



Drawback of the Programmed I/O :

The main drawback of the Program Initiated I/O was that the CPU has to monitor the units all the times when the program is executing. Thus 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 the CPU time is wasted a lot in keeping an eye to the executing of program. To remove this problem an Interrupt facility and special commands are used.

INTERRUPT-INITIATED I/O

In this method an interrupt facility an interrupt command is used to inform the device about the start and end of transfer. In the meantime the CPU executes other program. When the interface determines that the device is ready for data transfer it generates an Interrupt Request and sends it to the computer. When the CPU receives such a signal, it temporarily stops the execution of the program and branches to a service program to process the I/O transfer and after completing it returns back to task, what it was originally performing.

- In this type of IO, computer does not check the flag. It continue to perform its task.
- Whenever any device wants the attention, it sends the interrupt signal to the CPU.
- CPU then deviates from what it was doing, store the return address from PC and branch to the address of the subroutine.
- There are two ways of choosing the branch address:

- Vectored Interrupt
- Non-vectored Interrupt
- In vectored interrupt the source that interrupt the CPU provides the branch information. This information is called interrupt vectored.
- In non-vectored interrupt, the branch address is assigned to the fixed address in the memory.

Priority Interrupt:

- There are number of IO devices attached to the computer.
- They are all capable of generating the interrupt.
- When the interrupt is generated from more than one device, priority interrupt system is used to determine which device is to be serviced first.
- Devices with high speed transfer are given higher priority and slow devices are given lower priority.
- Establishing the priority can be done in two ways:
 - Using Software
 - Using Hardware
- A pooling procedure is used to identify highest priority in software means.

Polling Procedure :

- There is one common branch address for all interrupts.
- Branch address contain the code that polls the interrupt sources in sequence. The highest priority is tested first.
- The particular service routine of the highest priority device is served.
- The disadvantage is that time required to poll them can exceed the time to serve them in large number of IO devices.

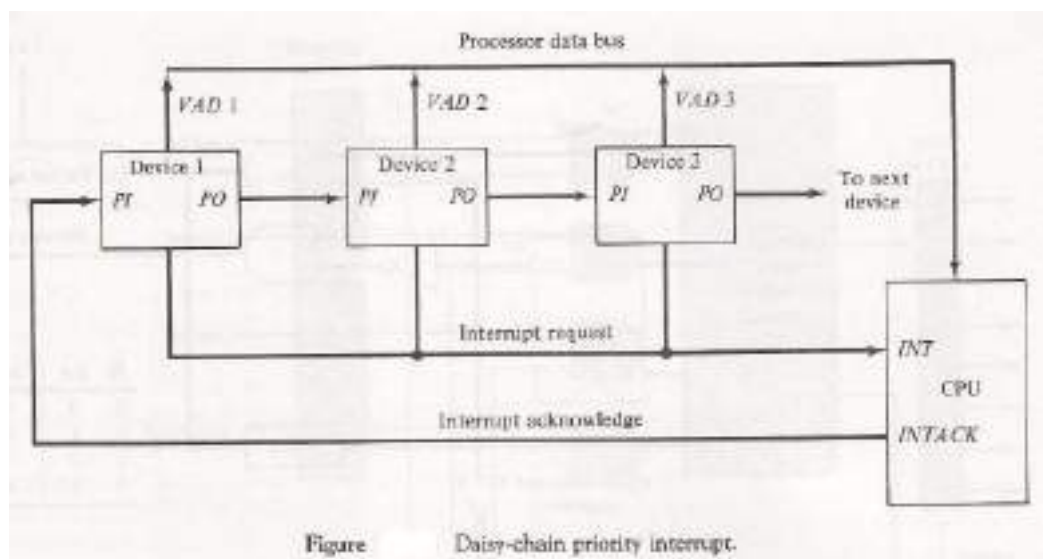
Using Hardware:

- Hardware priority system function as an overall manager.
- It accepts interrupt request and determine the priorities.
- To speed up the operation each interrupting devices has its own interrupt vector.

- No polling is required, all decision are established by hardware priority interrupt unit.
- It can be established by serial or parallel connection of interrupt lines.

Serial or Daisy Chaining Priority:

- Device with highest priority is placed first.
- Device that wants the attention send the interrupt request to the CPU.
- CPU then sends the INTACK signal which is applied to PI(priority in) of the first device.
- If it had requested the attention, it place its VAD(vector address) on the bus. And it block the signal by placing 0 in PO(priority out)
- If not it pass the signal to next device through PO(priority out) by placing 1.
- This process is continued until appropriate device is found.
- The device whose PI is 1 and PO is 0 is the device that send the interrupt request.

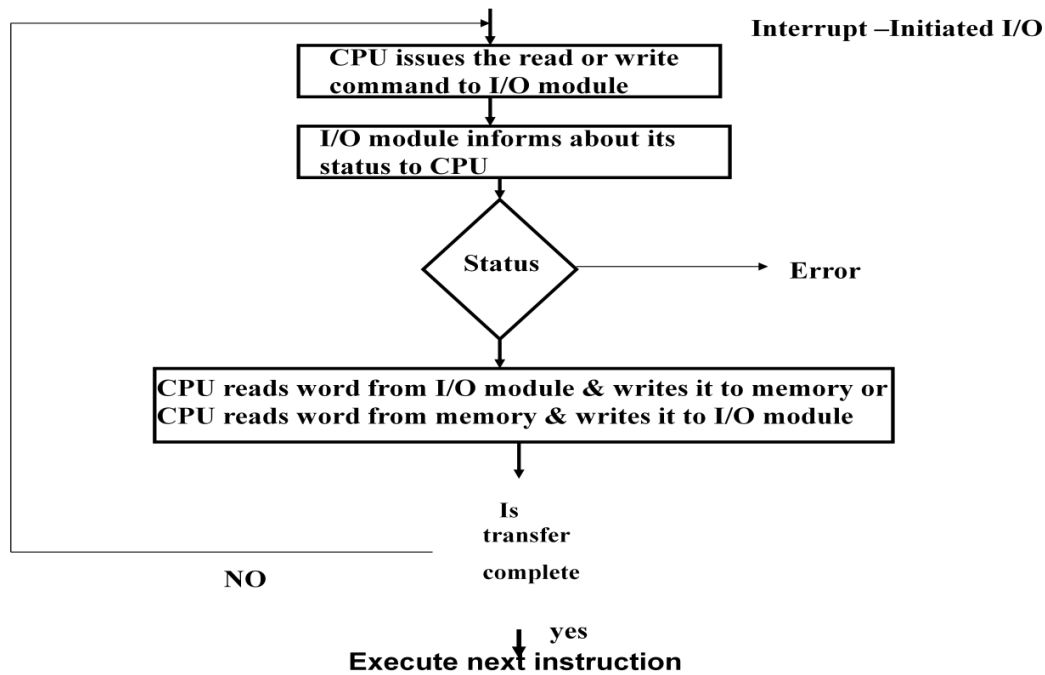


Parallel Priority Interrupt :

- It consist of interrupt register whose bits are set separately by the interrupting devices.
- Priority is established according to the position of the bits in the register.
- Mask register is used to provide facility for the higher priority devices to interrupt when lower priority device is being serviced or disable all lower priority devices when higher is being

served. Corresponding interrupt bit and mask bit are ANDed and applied to priority encoder.

- Priority encoder generates two bits of vector address.
- Another output from it sets IST (interrupt status flip flop).
- The Execution process of Interrupt-Initiated I/O is represented in the flowchart:



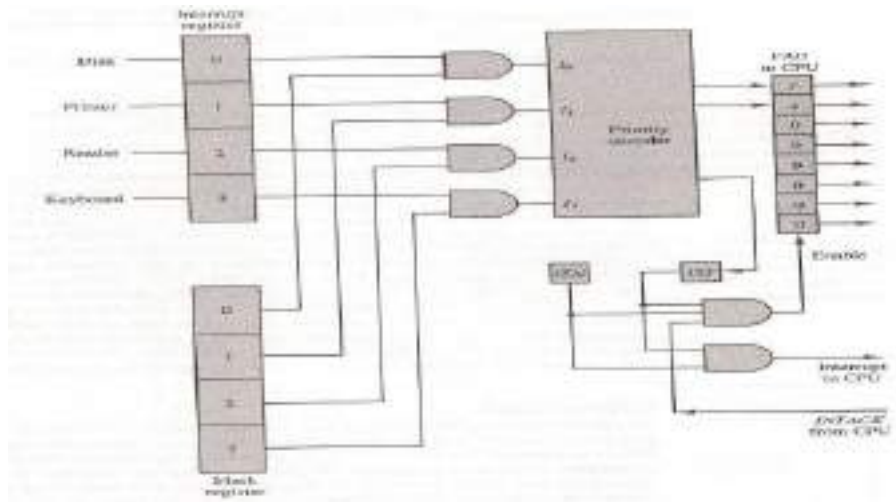
Priority Encoder Truth Table						
Inputs				Outputs		
I_0	I_1	I_2	I_3	x	y	IST
1	×	×	×	0	0	1
0	1	×	×	0	1	1
0	0	1	×	1	0	1
0	0	0	1	1	1	1
0	0	0	0	×	×	0

Boolean functions

$$x = I_0' I_1'$$

$$y = I_0' I_1 + I_0' I_2$$

$$(IST) = I_0 + I_1 + I_2 + I_3$$



DIRECT MEMORY ACCESS (DMA)

In the Direct Memory Access (DMA) the interface transfer the data into and out of the memory unit through the memory bus. The transfer of data between a fast storage device such as magnetic disk and memory is often limited by the speed of the CPU. Removing the CPU from the path and letting the peripheral device manage the memory buses directly would improve the speed of transfer. This transfer technique is called Direct Memory Access (DMA).

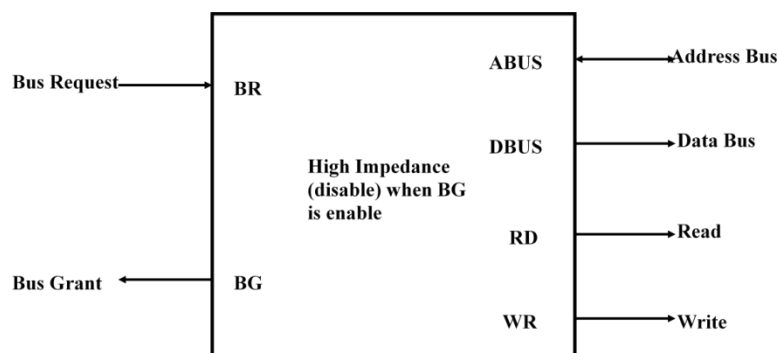
During the DMA transfer, the CPU is idle and has no control of the memory buses. A DMA Controller takes over the buses to manage the transfer directly between the I/O device and memory.

The CPU may be placed in an idle state in a variety of ways. One common method extensively used in microprocessor is to disable the buses through special control signals such as:

■ Bus Request (BR)

■ Bus Grant (BG)

These two control signals in the CPU that facilitates the DMA transfer. The *Bus Request (BR)* input is used by the *DMA controller* to request the CPU. When this input is active, the CPU terminates the execution of the current instruction and places the address bus, data bus and read write lines into a *high*



CPU bus Signals for DMA Transfer

Impedance state. High Impedance state means that the output is disconnected. The CPU activates the *Bus Grant (BG)* output to inform the external DMA that the *Bus Request (BR)* can now take control of the buses to conduct memory transfer without processor. When the DMA terminates the transfer, it disables the *Bus Request (BR)* line. The CPU disables the *Bus Grant (BG)*, takes control of the buses and return to its normal operation.

The transfer can be made in several ways that are:

i. DMA Burst transfer

ii. Cycle Stealing

- i) DMA Burst transfer:- In DMA Burst transfer, a block sequence consisting of a number of memory words is transferred in continuous burst while the DMA controller is master of the memory buses.
- ii) Cycle Stealing :- Cycle stealing allows the DMA controller to transfer one data word at a time, after which it must return control of the buses to the CPU.

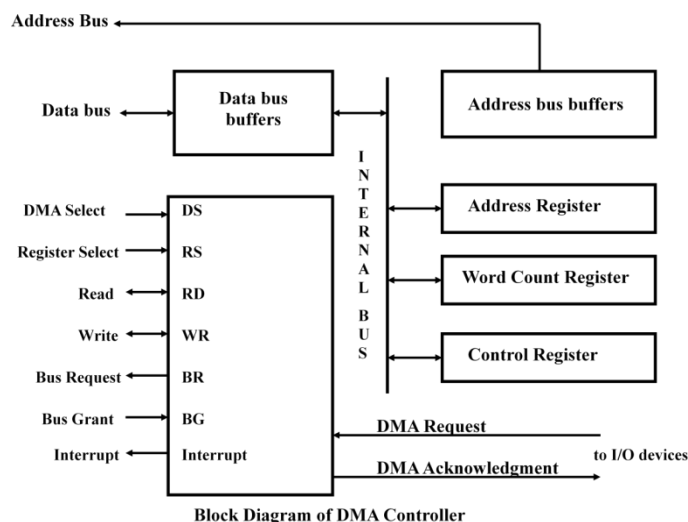
DMA Controller:

The DMA controller needs the usual circuits of an interface to communicate with the CPU and I/O device. The DMA controller has three registers:

- i. Address Register- Address Register contains an address to specify the desired location in memory
- ii. Word Count Register- WC holds the number of words to be transferred. The register is incre/decre by one after each word transfer and internally tested for zero.
- iii. Control Register :- Control Register specifies the mode of transfer

The unit communicates with the CPU via the data bus and control lines. The registers in the DMA are selected by the CPU through the address bus by enabling the DS (DMA select) and RS (Register select) inputs. The RD (read) and WR (write) inputs are bidirectional.

When the BG (Bus Grant) input is 0, the CPU can communicate with the DMA registers through the data bus to read from or write to the DMA registers. When BG =1, the DMA can communicate directly with the memory by specifying an address in the address bus



and activating the RD or WR control.

DMA Transfer:

The CPU communicates with the DMA through the address and data buses as with any interface unit. The DMA has its own address, which activates the DS and RS lines. The CPU initializes the DMA through the data bus. Once the DMA receives the start control command, it can transfer between the peripheral and the memory. When $BG = 0$ the RD and WR are input lines allowing the CPU to communicate with the internal DMA registers. When $BG=1$, the RD and WR are output lines from the DMA controller to the random access memory to specify the read or write operation of data.