UNIT-5 (PART-A)
MEMORY ORGANIZATION

5.1. 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.

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

- Devices that provide backup storage are called auxiliary memory. The most common auxiliary memory devices used in computer systems are magnetic disks and tapes. They are used for storing system programs, large data files, and other backup information.

- Only programs and data currently needed by the processor reside in main memory. All other information is stored in auxiliary memory and transferred to main memory when needed.

- The total memory capacity of a computer can be visualized as being a hierarchy of components.

- 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.

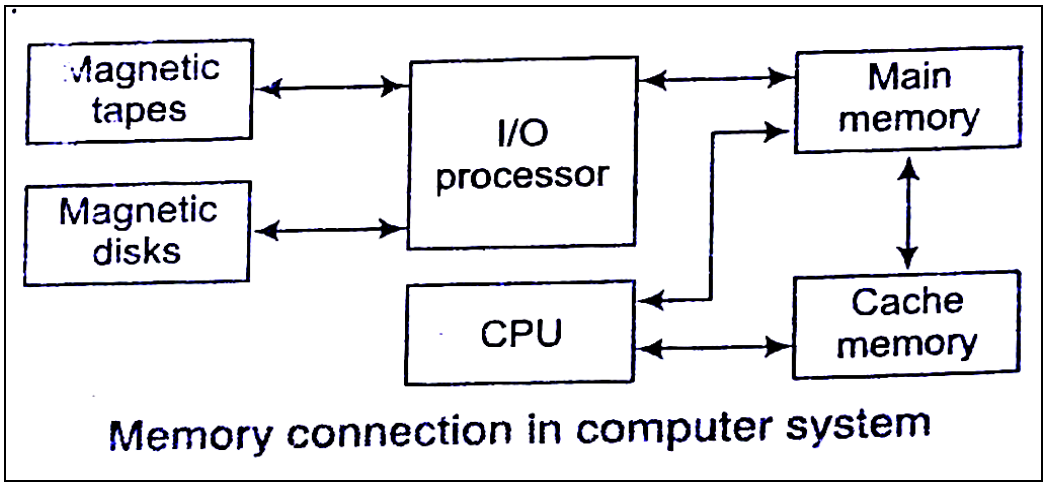


Fig :Memory hierarchy in a computer system.

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

- 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.

- Thus each is involved with a different level in the memory hierarchy system. The reason for having two or three levels of memory hierarchy is economics.

- 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.

5.2 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. 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 unit.

- The dynamic RAM stores the binary information in the form of electric charges that are applied to capacitors. 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 shorted read and write cycles. 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 one the production of the computer is completed.

- 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 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

5.2.1 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.

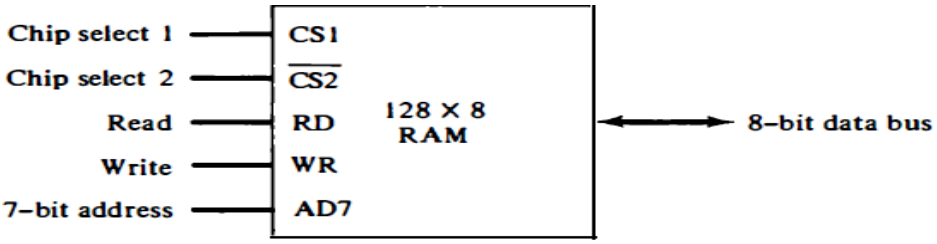
- The logic 1 and 0 are normal digital signals. The high-impedance state behaves like an open circuit, which means that the output does not carry a signal and has no logic significance.

- The block diagram of a RAM chip is shown in Fig .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 or read or write.



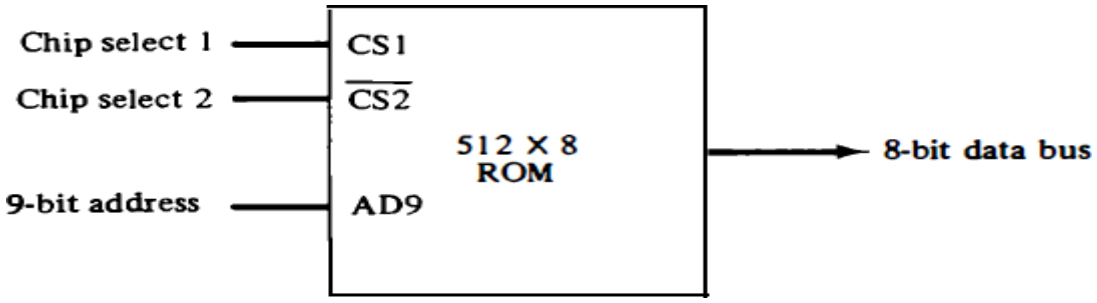
(a) Block diagram

CS1	$\overline{CS2}$	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

(b) Function table

The function table listed in Fig(b) specifies the operation of the RAM chip.

- The unit is in operation only when CSI = 1 and CS2 = 0.
- The bar on top of the second select variable indicates that this input in enabled when it is equal to 0. If the chip select inputs are not enabled, or if they are enabled but the read but the read or write inputs are not enabled, the memory is inhibited and its data bus is in a high-impedance state.
- When SC1 = 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. 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. For the same-size chip, it is possible to have more bits of 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 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.
- Thus when the chip is enabled by the two select inputs, the byte selected by the address lines appears on the data bus.

5.2.2 MEMORY ADDRESS MAP of RAM and ROM

- 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 form 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

Memory Address Map for Microprocomputer

Component	Hexadecimal 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

- 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 fro 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 form 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.

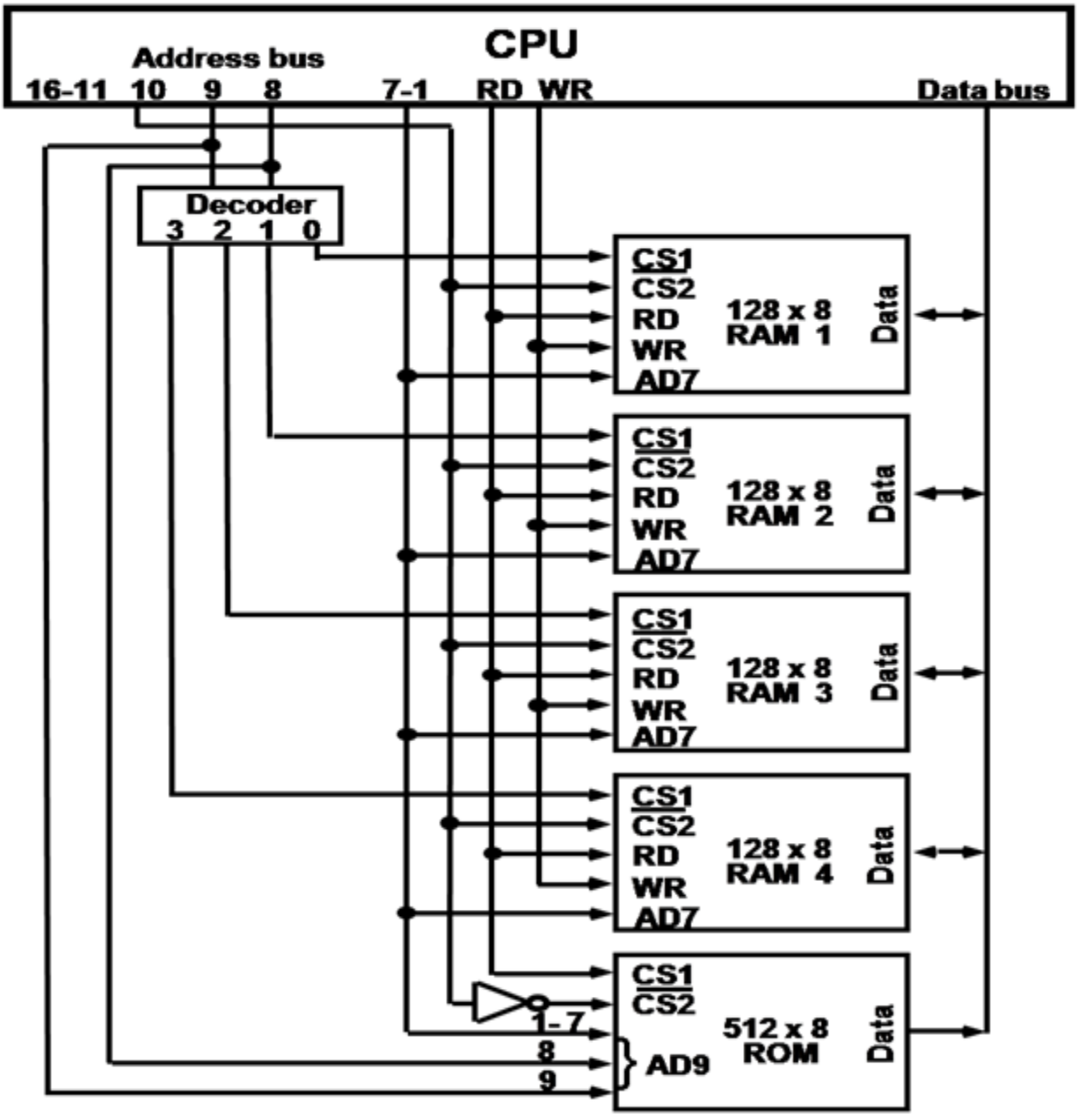
5.2.3 MEMORY CONNECTION TO CPU

- 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. This configuration gives a memory capacity of 512 bytes of RAM and 512 bytes of ROM.
- 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 2X4 decoder whose outputs go to the SCI input 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.



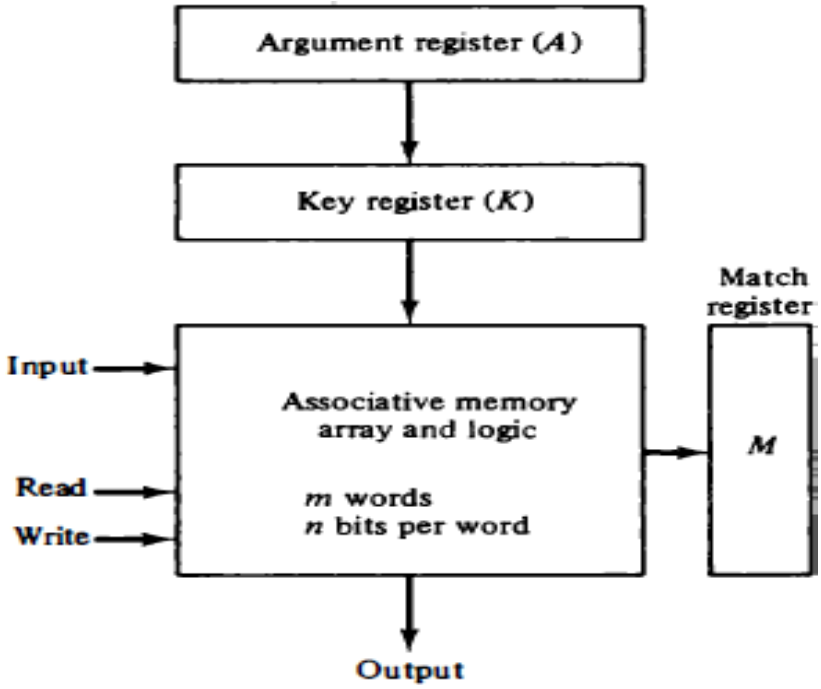
5.3 ASSOCIATIVE MEMORY

- Many 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.
- 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 locaters 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.
- An associative memory is more expensive then 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.

5.3.1 HARDWARE ORGANIZATION

The block diagram of an associative memory consists of a memory array and logic from 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.

Block diagram for Associate Memory



- 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.

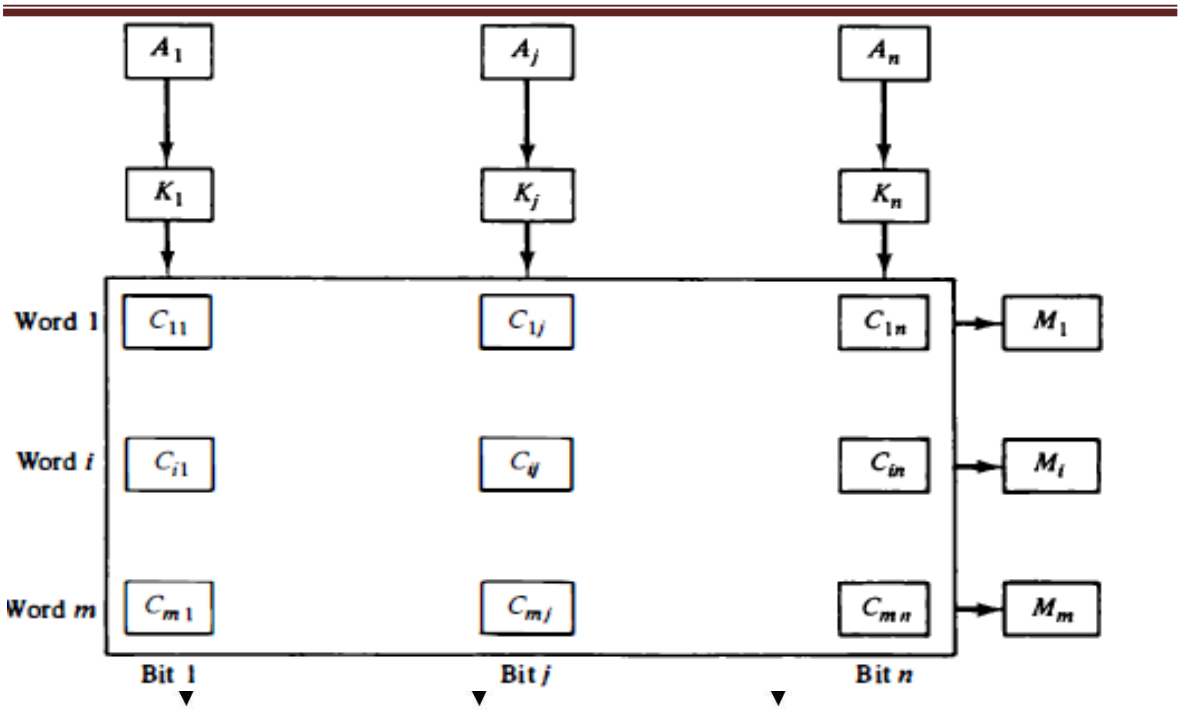
- The key register provides a mask for choosing a particular field or key in the argument word. The entire argument is compared with each memory word if the key register contains all 1's.

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

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

The relation between the memory array and external registers in an associative memory is shown in below figure.



The cells in the array are marked by the letter C with two subscripts. The first subscript gives the word number and the second specifies the bit position in the word. Thus cell Cij is the cell for bit j in word i. A bit 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 Mi in the match register is set to 1.

If one or more unmasked bits of the argument and the word do not match, Mi is cleared to 0. Flop storage element Fij 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 Mi.

5.3.2. 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 Aj = Fij 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 xj = 1 if the pair of bits in position j are equal; otherwise, xj = 0.

- For a word i to be equal to the argument in A we must have all xj variables equal to 1. This is the condition for setting the corresponding match bit Mi to 1. The Boolean function for this condition is $M_i = x_1 x_2 x_3 \ldots x_n$ and constitutes the AND operation of all pairs of matched bits in a word.

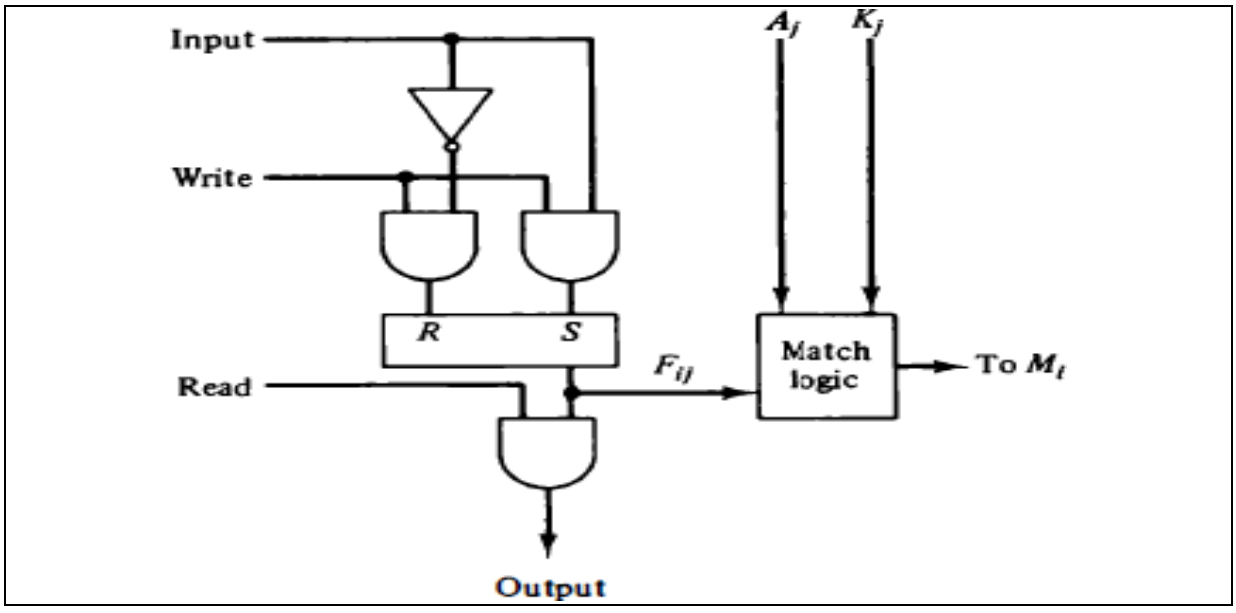


Fig : One cell associative Memory

5.3.3 READ OPERATION

- The matched words are read in sequence by applying a read signal to each word line whose corresponding Mi bit is a 1.
- In most applications, the associative memory stores a table with no two identical items under a given key. In this case, only one word may match the unmasked argument field.
- By connecting output Mi directly to the read line in the same word position (instead of the M register), the content of the matched word will be presented automatically at the output lines and no 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.

5.3.4WRITE OPERATION

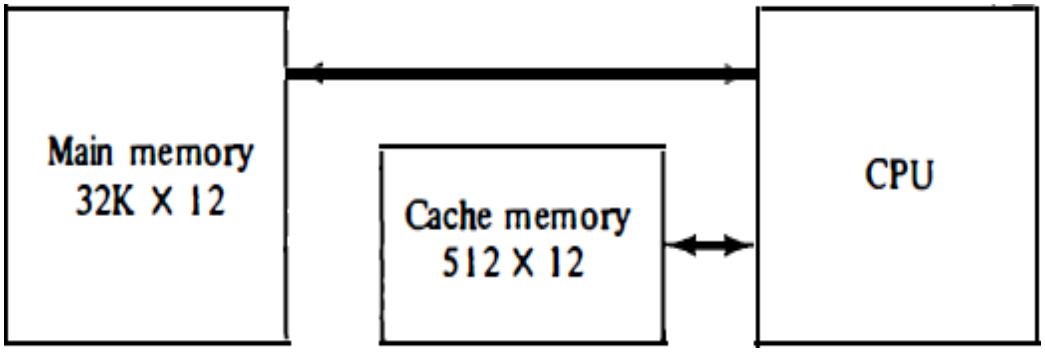
- If the entire memory is loaded with new information at once prior to a search operation then the writing can be done by addressing each location in sequence.
- This will make the device a random-access memory for writing and a content addressable memory for reading. The advantage here is that the address 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.

5.4 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. The phenomenon is known as the **property of locality of reference.**

- 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 frequently.

- 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 figure.

- 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 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.

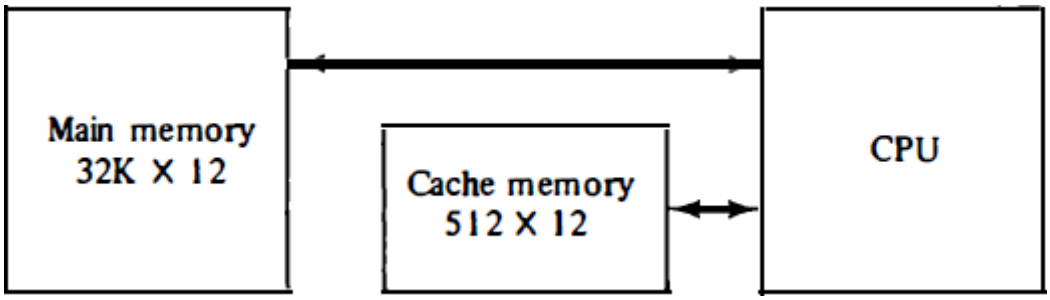
- The block size may vary from one word (the one just accessed) to about 16 words adjacent to the one just accessed. In this manner, some data are transferred to cache so that future references to memory find the required words in the fast cache memory.

- 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 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.
- 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**

To helping the discussion of these three mapping procedures we will use a specific example of a memory organization as shown in figure.

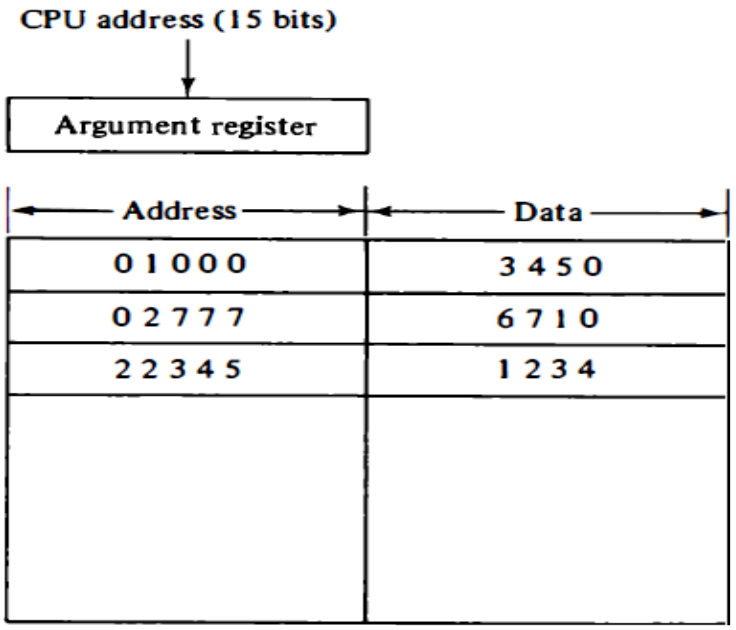


- The main memory can store 32K words of 12 bits each. The cache is capable of storing 512 of these words at any given time. For every word stored in cache, there is a duplicate copy in main memory.
- The CPU communicates with both memories. It first sends a 15-bit address to cache. If there is a hit,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

5.4.1 ASSOCIATIVE MAPPING

The fasters and most flexible cache organization uses an associative memory.

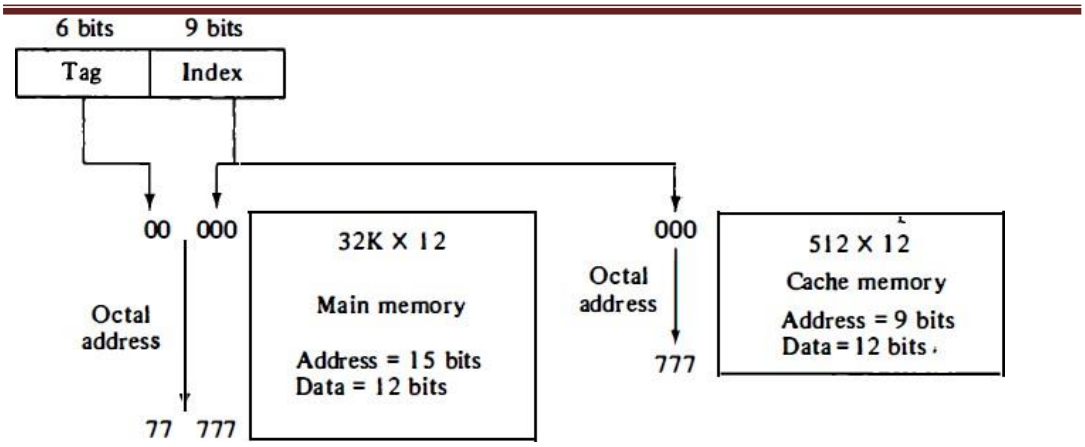
This organization is illustrated in Fig.



- 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.
- This constitutes a first-in first-out (FIFO) replacement policy.

5.4.2 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 and the index bits.

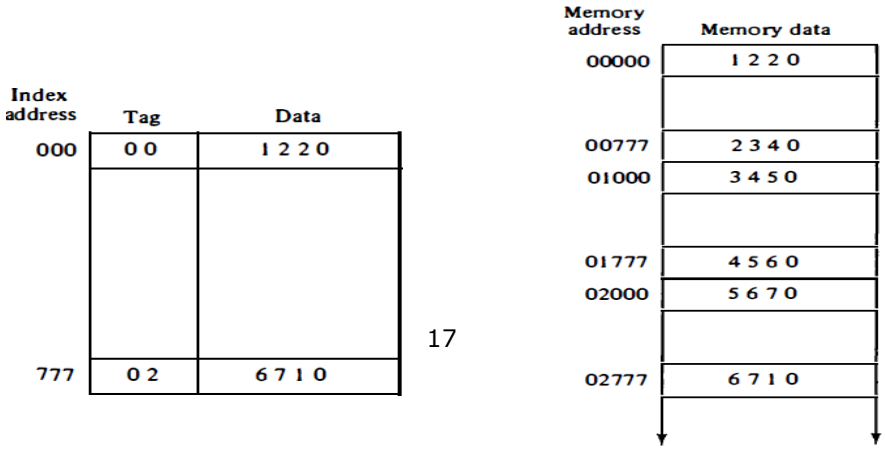
The number of bits in the index field is equal to the number of address bits required to access the cache memory.

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.

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.

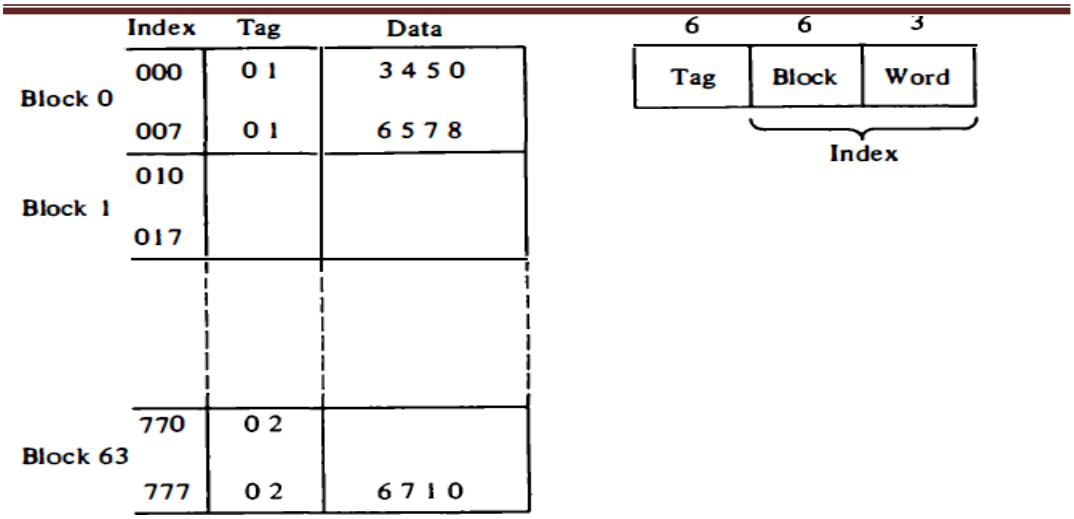


- When the CPU Each word in cache consists of the data word and its associated tag. When a new word is first brought into the cache, the tag bits are stored alongside the data bits. When the CPU generates a memory request, the index field is used for the address to access the cache.
- The tag field of the CPU address is compared with the tag in the word read from the cache. If the two tags match, there is a hit and the desired data word is in cache. If 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.

To see how the direct-mapping organization operates, consider the numerical example shown in Fig below.

- 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 sued to access the cache. The two tags are then compared. The cache tag is 00 but the address tag is 02, which does not produce a match. Therefore, the main memory is accessed and the data word 5670 is transferred to the CPU. The cache word at index address 000 is then replaced with a tag of 02 and data of 5670.

The same organization but using a block size of 8 words is shown in Fig.



- The index field is now divided into two parts: **the block field and the word field**. In a 512-word cache there are 64 block of 8 words each, since 64X8 = 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.

5.4.3 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 f ig.

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

- 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 X 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.
- With reference to the main memory content the following is illustrated. 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 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.

WRITING INTO CACHE

- For write operation, there are two ways that the system can proceed.
- The simplest and most commonly used procedure is to up data main memory with every memory write operation, with cache memory being updated in parallel if it contains the word at the specified address. This is called the **write-through method.**
- The second procedure is called the **write-back method.** In this method only the cache location is updated during a write operation. The location is then marked by a flag so that later when the words are removed from the cache it is copied into main memory.

CACHE INITIALIZATION

- One more aspect of cache organization that must be taken into consideration is the problem of initialization.
- The cache is initialized when power is applied to the computer or when the main memory is loaded with a complete set of programs from auxiliary memory. After initialization the cache is considered to be empty, built in effect it contains some non-valid data.
- It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.
- The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache word is set to 1 the first time this word is loaded from main memory and stays set unless the cache has to be initialized again.

Unit-5 (Part-B)

Pipeline-Processing

5.1. Parallel Processing

- Parallel processing is a term used to denote a large class of techniques that are used to provide simultaneous data processing tasks for the purpose of increasing the computational speed of computer system.

- Instead of processing each instruction sequentially as in a conventional computer, a parallel processing system is able to perform concurrent data processing to achieve faster execution time.

- For example, while an instruction is being executed in the ALU, the next instruction can be read from memory. The system may have two or more processors operating concurrently.

ADVANTAGES OF PARALLEL PROCESSING:

1. It speeds up the computer processing capability.

2. Increases its throughput, i.e., the amount of processing that can be accomplished during a given interval of time.

3. The amount of hardware increases with parallel processing and with it the cost of the system increases. However, technological developments have reduced hardware costs to the point where parallel processing techniques are economically feasible.

- Parallel processing can be achieved by having a multiplicity of functional units that perform identical or different operations simultaneously. Parallel processing is established by distributing the data among the multiple functional units.

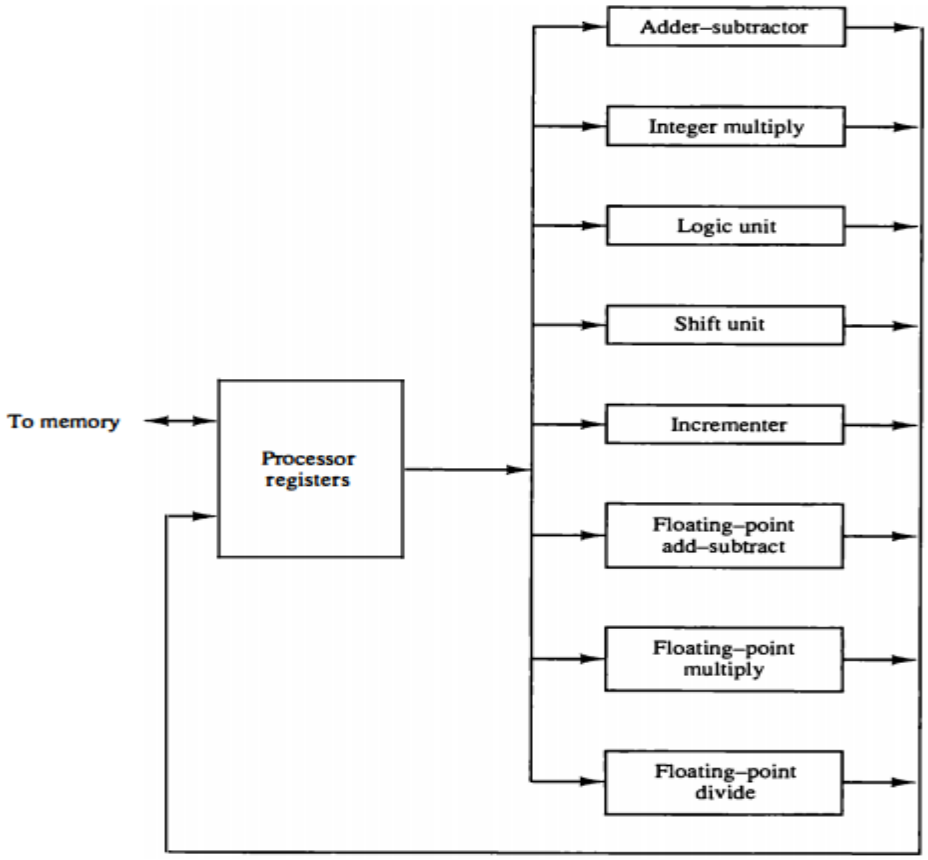
- Fig shows one possible way of separating the execution time into 8 functional units operating in parallel. The operands in the registers are applied to one of the units depending on the operation specified by the instruction associated with the operands.

- The operation performed in each functional unit is indicated in each block of the diagram. The adder and integer multiplier perform the arithmetic operations with integer numbers.

- The floating point operations are separated into three circuits operating in parallel.

- The logic, shift and increment operations can be performed concurrently on different data. All units are independent of each other, so one number can be shifted while another number is being incremented.

Figure 1 Processor with multiple functional units.



Flynn's classification divides computers into four major groups

1. Single instruction stream, single data stream (SISD)
2. Single instruction stream, multiple data stream (SIMD)
3. Multiple instruction streams, single data stream (MISD)
4. Multiple instruction stream, multiple data stream (MIMD)

SISD:

SISD represents the organization of a single computer containing a control unit, a processor unit and a memory unit. Instructions are executed sequentially and the system may or may not have internal parallel processing capabilities. Parallel processing in this case may be achieved by means of multiple functional units or by pipeline processing.

SIMD:

SIMD represents an organization that includes many processing units under the supervision of a common control unit. All processors receive the same instruction from the control unit but operate on different items of data. The shared memory unit must contain multiple modules so that it can communicate with all the processors simultaneously.

5.2. 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.

- A pipeline can be visualized as a collection of processing segments through which binary information flows. Each segment performs partial processing dictated by the way the task is partitioned.

- The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments.

5.2.1. Pipeline Organization

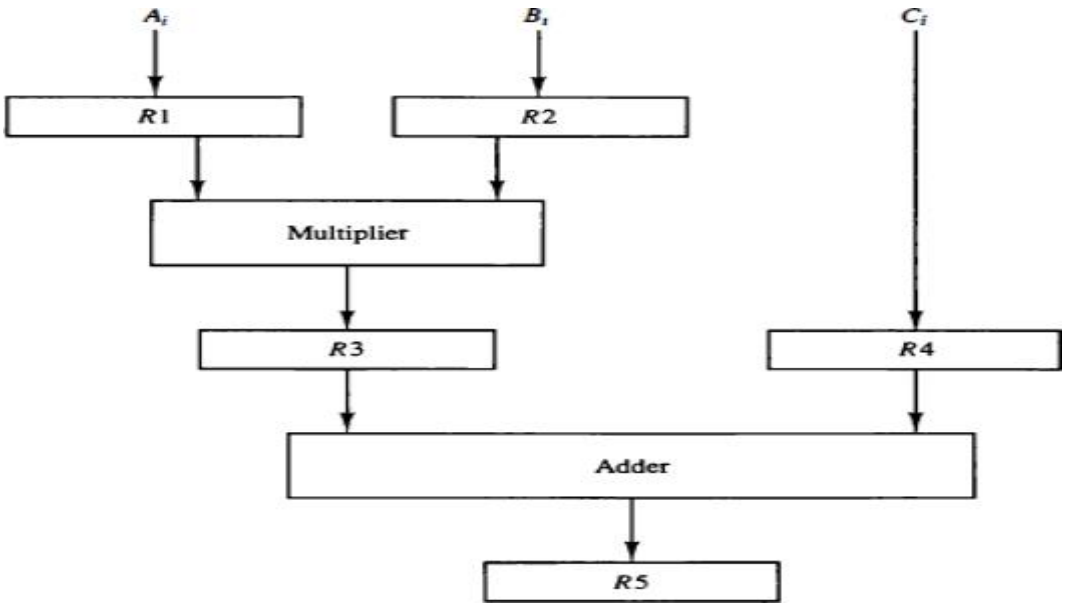
- The simplest way of viewing the pipeline structure is to imagine that each segment consists of an input register followed by a combinational circuit.

- The register holds the data and the combinational circuit performs the sub operation in the particular segment. The output of the combinational circuit is applied to the input register of the next segment.

- A clock is applied to all registers after enough time has elapsed to perform all segment activity. In this way the information flows through the pipeline one step at a time.

- Example demonstrating the pipeline organization Suppose we want to perform the combined multiply and add operations with a stream of numbers.

  Ai*Bi + Ci for i=1, 2, 3 ….7



Each sub operation is to implemented in a segment within a pipeline. Each segment has one or two registers and a combinational circuit as shown in fig.

→ R1 through r5 are registers that receive new data with every clock pulse.
→ The multiplier and adder are combinational circuits.
→ The sub operations performed in each segment of the pipeline are as follows:

R1<- Ai	R2<-Bi	Input Ai and Bi
R3<-R1*R2	R4<-Ci	multiply and input Ci
R5<-R3+R4		add Ci to product

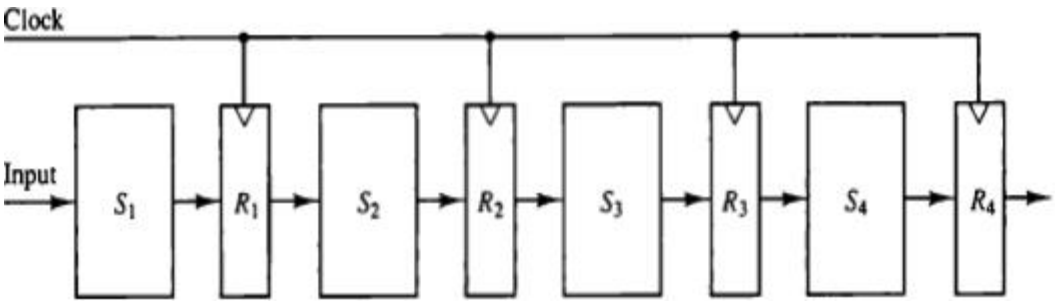
The five registers are loaded with new data every clock pulse.

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	—	—	—	—	$A_7 * B_7 + C_7$

- The first clock pulse transfers A1 and B1 into R1 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, transfers 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 the pipe and retrieve the first output from R5. From there on, each clock produces a new output and moves the data one step down the pipeline.
- This happens as long as new input data flow into the system.

5.2.2. FOUR SEGMENT Pipeline

- The general structure of four segment pipeline is shown in fig. the operands are passed through all four segments in affixed sequence.
- Each segment consists of a combinational circuit Si that performs a sub operation over the data stream flowing through the pipe.
- The segments are separated by registers Ri that hold the intermediate results between the stages. Information flows between adjacent stages under the control of a common clock applied to all the registers simultaneously.



Time - Space Diagram:

- The behavior of a pipeline can be illustrated with a space time diagram. This is a diagram that shows the segment utilization as a function of time.
- Fig The horizontal axis displays the time in clock cycles and the 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.

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

- After the first clock, segment 2 is busy with T1, while segment 1 is busy with task T2. Continuing in this manner, the first task T1 is completed after fourth clock cycle. From then on, the pipe completes a task every clock cycle.

- Consider the case where a k-segment pipeline with a clock cycle time tp is used to execute n tasks. The first task T1 requires a time equal to ktp to complete its operation since there are k segments in a pipe. The remaining n-1 tasks emerge from the pipe at the rate of one task per clock cycle and they will be completed after a time equal to (n-1) tp.

- Therefore, to complete n tasks using a k segment pipeline requires **k+ (n-1)** clock cycles.

- Consider a non pipeline unit that performs the same operation and takes a time equal to tn to complete each task. The total time required for n tasks is n tn. The speedup of a pipeline processing over an equivalent non pipeline processing is defined by the ratio

  S=ntn / (k+n-1)tp

- As the number of tasks increases, n becomes much larger than k-1, and k+n-1 approaches the value of n. under this condition the speed up ratio becomes

  S=tn/tp

- If we assume that the time it takes to process a task is the same in the pipeline and non pipeline circuits, we will have tn=ktp. Including this assumption speed up ratio reduces to

  S=ktp/tp=k

5.3. ARITHMETIC PIPELINE

An arithmetic pipeline divides an arithmetic operation into sub operations for execution in the pipeline segments. Pipeline arithmetic units are usually found in very high speed computers. They are used to implement floating point operations, multiplication of fixed point numbers, and similar computations encountered in scientific problems.

5.3.1. Pipeline Unit For Floating Point Addition And Subtraction:

The inputs to the floating point adder pipeline are two normalized floating point binary numbers.

$X=A*2^a$
$Y=B*2^b$

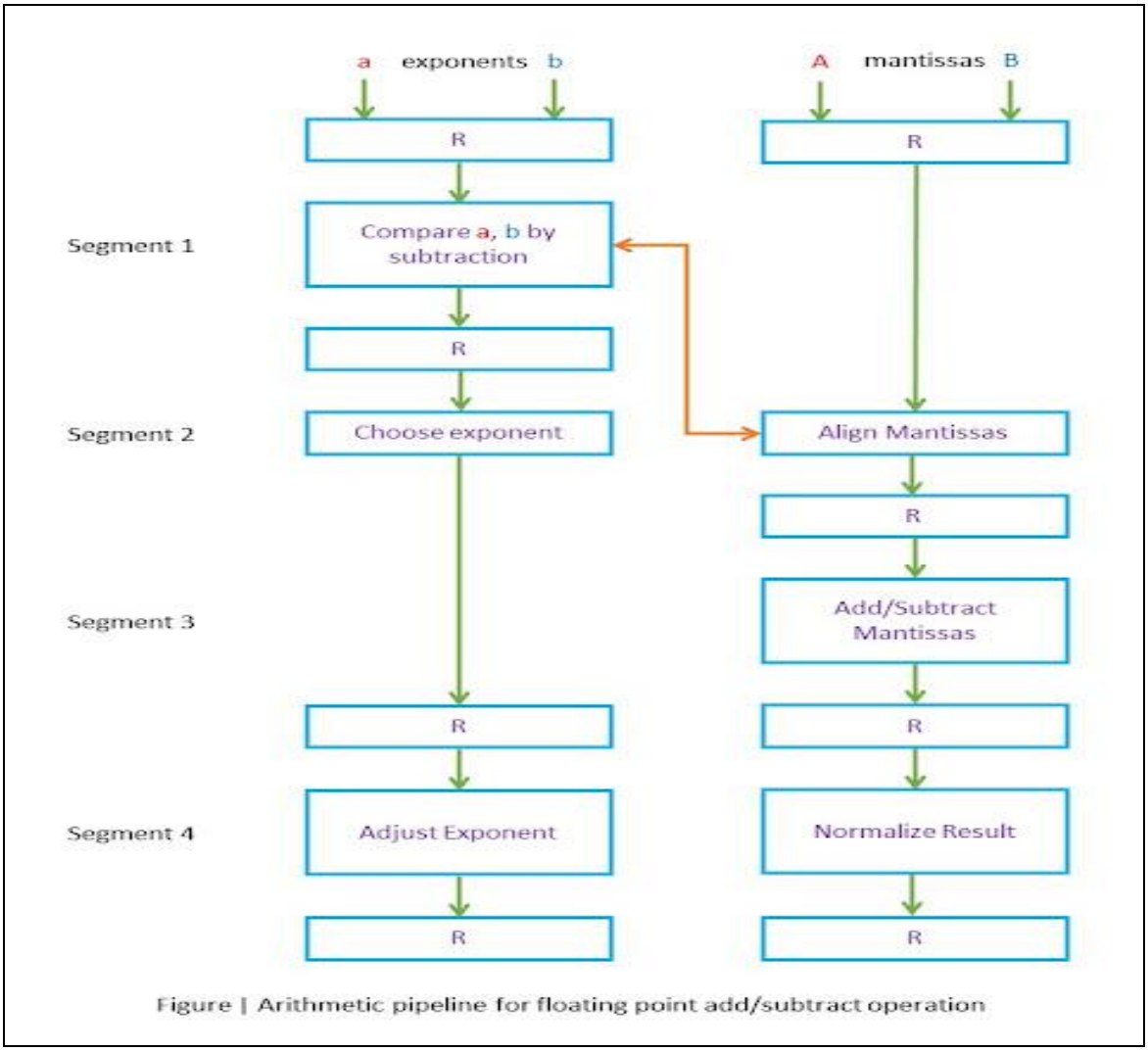


Figure | Arithmetic pipeline for floating point add/subtract operation

- A and B are two fractions that represent the mantissa and a and bare the exponents. The floating point addition and subtraction can be performed in four segments.

- The registers labeled are placed between the segments to store intermediate results. The sub operations that are performed in the **four segments** are:

  1. Compare the exponents
  2. Align the mantissa.
  3. Add or subtract the mantissas.
  4. Normalize the result.

  - The exponents are compared by subtracting them to determine their difference. The larger exponent is chosen as the exponent of the result.

  - The exponent difference determines how many times the mantissa associated with the smaller exponent must be shifted to the right. This produces an alignment of the two mantissas.

  - The two mantissas are added or subtracted in segment3.

  - The result is normalized in segment 4.

  - When an overflow occurs, the mantissa of the sum or difference is shifted to right and the exponent incremented by one.

  - If the underflow occurs, the number of leading zeroes in the mantissa determines the number of left shits in the mantissa and the number that must be subtracted from the exponent.

5.4. INSTRUCTION PIPELINE

- An instruction pipeline operates on a stream of instructions by overlapping the fetch, decode, and execute phases of instruction cycle.

- An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and executes phases to overlap and perform simultaneous operations.

- Consider a computer with an instruction fetch unit and an instruction execute unit designed to provide a two segment pipeline.

- The instruction fetch segment can be implemented by means of a first in first out (FIFO) buffer. Whenever the execution unit is not using memory, the control increments the program counter and uses it address value to read consecutive instructions from memory.

- The instructions are inserted into the FIFO buffer so that they can be executed on a first in first out basis. Thus an instruction stream can be placed in queue, waiting for decoding and processing by the execution segment.

- In general the computer needs to process each instruction with the following sequence of steps.
    1. Fetch the instruction
    2. Decode the instruction.
    3. Calculate the effective address.
    4. Fetch the operands from memory.
    5. Execute the instruction.
    6. Store the result in the proper place.

5.4.1. Four-segment Instruction pipeline

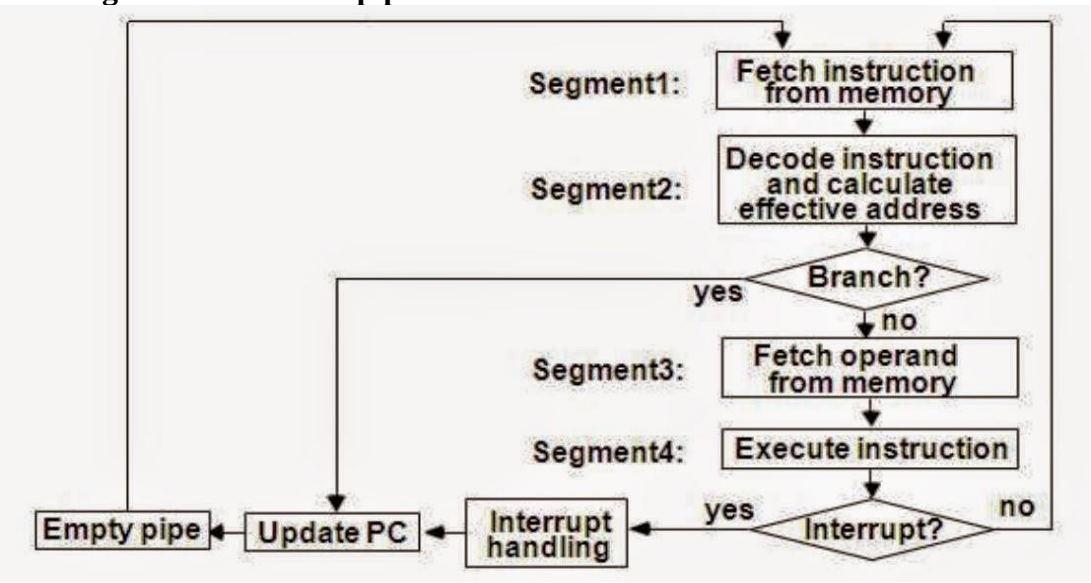


Fig shows the instruction cycle in the CPU can be processed with a four segment pipeline. While an instruction is being executed in segment 4, the next instruction in sequence is busy with fetching an operand from memory in segment 3. the effective address may be calculated in a separate arithmetic circuit for the third instruction, and whenever the memory is available, the fourth and all subsequent instructions are placed in an instruction FIFO.

Timing of Instruction Pipeline

Step:		1	2	3	4	5	6	7	8	9	10	11	12	13
Instruction:	1	FI	DA	FO	EX									
	2		FI	DA	FO	EX								
(Branch)	3			FI	DA	FO	EX							
	4				FI	-	-	FI	DA	FO	EX			
	5					-	-	-	FI	DA	FO	EX		
	6									FI	DA	FO	EX	
	7										FI	DA	FO	EX

This Fig shows the operation of the instruction pipeline. The time in the horizontal axis is divided into steps of equal duration. The four segments are represented in the diagram with an abbreviated symbol.

1. FI is the segment that fetches an instruction.
2. DA is the segment that decodes the instruction and calculates the effective address.
3. FO is the segment that fetches the operand.
4. EX is the segment that executes the instruction.

It is assumed that the processor has separate instruction and data memories so that the operation in FI and FO can proceed at the same time. In the absence of a branch instruction, each segment operates on different instructions. Thus, in step 4, instruction 1 is being executed in segment EX; the operand for instruction 2 is being fetched into segment FO; instruction 3 is being decoded in segment DA; and instruction 4 is being fetched from memory in segment FI.

Assume now this instruction is a branch instruction. As soon as this instruction is decoded in segment DA in step 4, the transfer from FI to DA of the other instructions are halted until the branch instruction is executed in step 6.

PIPELINE CONFLICTS:

1. RESOURCE CONFLICTS: They are caused by access to memory by two segments at the same time. Most of these conflicts can be resolved by using separate instruction and data memories.

2. DATA DEPENDENCY: these conflicts arise when an instruction depends on the result of a previous instruction, but this result is not yet available.

3. BRANCH DIFFERENCE: they arise from branch and other instructions that change the value of PC.

5.4.2. DATA DEPENDENCY

- A difficulty that may cause a degradation of performance in an instruction pipeline collision of data or address. A collision occurs when an instruction cannot proceed because previous instructions did not complete certain operations.

- A data dependency occurs when an instruction needs data that are not yet available. For example an instruction in the FO segment may need to fetch an operand that is being generated at the same time by the previous instruction in segment EX. Therefore, the second instruction must wait for the data to become available by the first instruction.

- An address dependency may occur when an operand address cannot be calculated because the information needed by the addressing mode is not available. For example, an instruction with register indirect mode can not proceed to fetch the operand if the previous instruction is loading the address into the register. Therefore operand access to memory must be delayed until the required address is available.

5.4.3. Handling of Branch Instructions

- One of the major problems in operating the instruction pipeline is the occurrence of the branch instructions. A branch instruction can be conditional or unconditional. An unconditional branch always alters the sequential program flow by loading the program counter with the target address. In a conditional branch, the control selects the target instruction if the condition is satisfied or the next sequential instruction if the condition is not satisfied.

- Pipeline computers employ various hardware techniques to minimize the performance of degradation caused by instruction branching.

- One way of handling a conditional branch is to pre fetch the target instruction in addition to the instruction following the branch. Both are saved until the branch is executed. If the branch condition is successful, the pipeline continues from the branch target instruction Another possibility is the use of BRANCH TARGET BUFFER .