Memory Hierarchy :→

* Memory Hierarchy is an enhancement to organize the memory such that it can minimize the access time.
* The Memory Hierarchy was developed based on a program behaviour known as locality of references.
* The figure below clearly demonstrates the different levels of memory hierarchy.

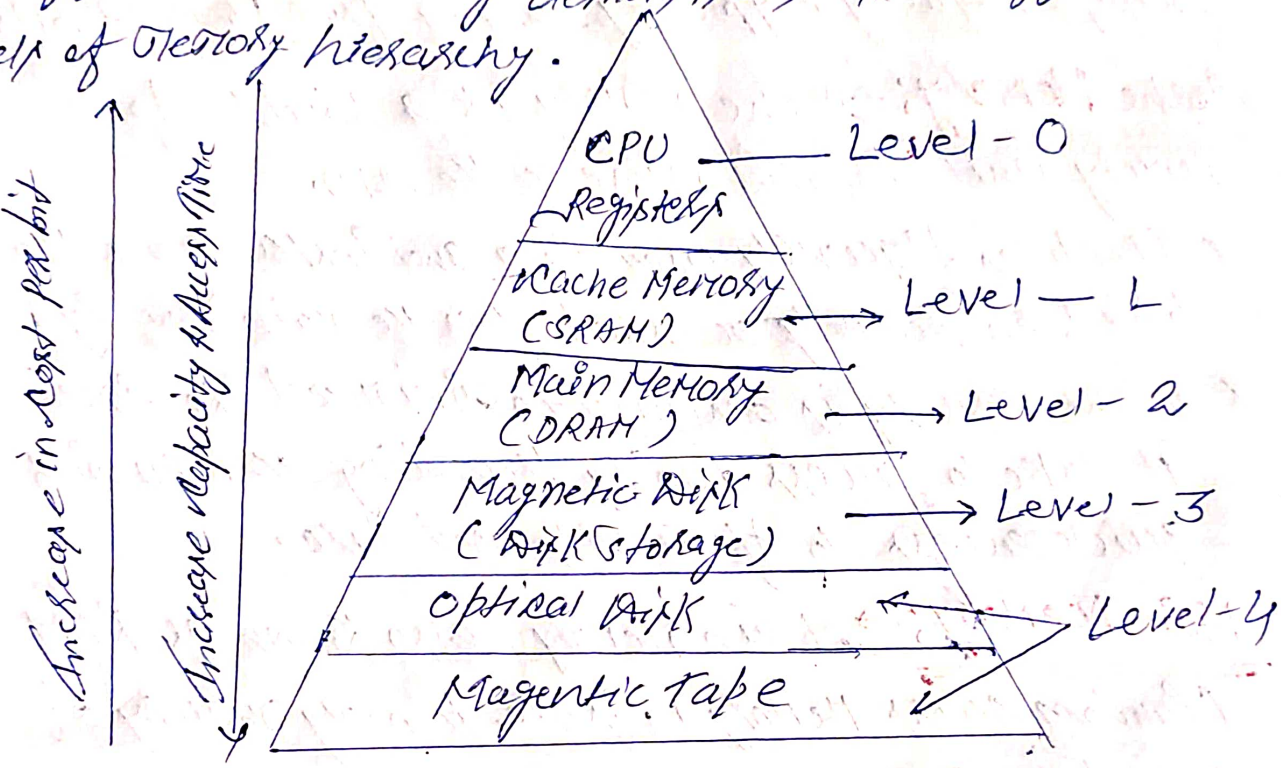


Fig: Memory Hierarchy Design

* This Memory Hierarchy Design is divided into 2 (Two) main types:-

1. External Memory or Secondary Memory:-

* Comprising of Magnetic Disk, Optical Disk, Magnetic Tape i.e. peripheral storage devices which are accessible by the processor via I/O Module.

2. Internal Memory or Primary Memory:→

* Comprising of Main Memory, Cache Memory & CPU Registers. This is directly accessible by the processor.

* There are typically four levels of memory in a memory hierarchy:

Registers: Registers are small, high-speed memory units located in the CPU.

* They are used to store the most frequently used data and instructions.
* Registers have the fastest access time and the smallest storage capacity, typically ranging from 16 to 64 bits.

Cache Memory: Cache memory is a small, fast memory unit located close to the CPU.

* It stores frequently used data and instructions that have been recently accessed from the main memory.
* Cache memory is designed to minimized the time it take to access data by providing the CPU with quick access to frequently used data.

Main Memory: → * Main memory, also known as RAM (Random Access Memory), is the primary memory of a computer system.

* It has a larger storage capacity than cache memory, but it is slower.
* Main memory is used to store data and instructions, that are currently in use by the CPU.
* Types of Main Memory:—
  - Static RAM: → It stores the binary information in F-Fs and information remains valid until power is supplied.
* It has faster access time and is used in implementing cache memory

- Dynamic RAM:→ It stores the binary information as a charge on the capacitor. It requires refreshing circuitry to maintain the charge on the capacitor after few milliseconds. It contains more memory cells per unit area as compared to SRAM.

Secondary Storage:→

* Secondary Storage, such as HDD, and SSD, is non-volatile memory unit that has a larger storage capacity than main memory.
* It is used to store data and instructions that are not currently in use by the CPU.
* Secondary storage has the slowest access time and is typically the least expensive type of memory in the memory hierarchy.

We can infer the following characteristics of Memory Hierarchy design from above figure.

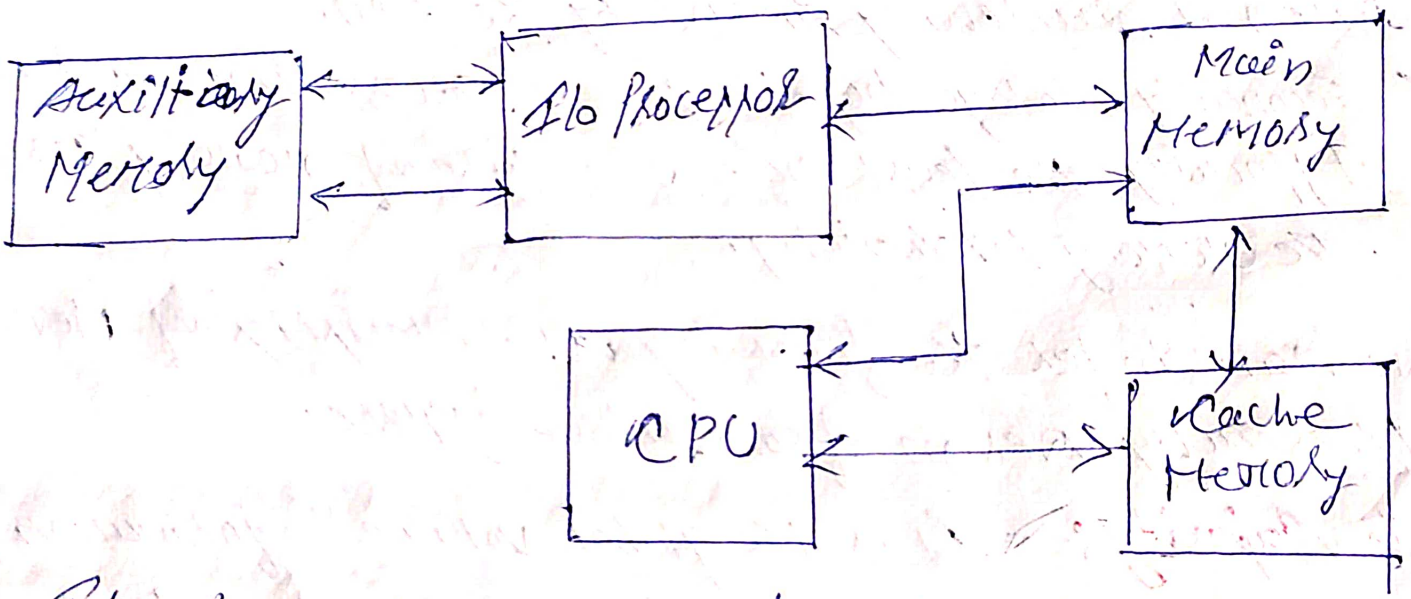
1. Capacity:— It is the global volume information the memory can store. As we move from top to bottom in the hierarchy, the capacity increases.
2. Access time:— It is the time interval between the read/write request and the availability of the data. As we move from top to bottom in the hierarchy, the access time increases.
3. Performance:—

Earlier when the computer system was designed without memory hierarchy design the speed gap increases between the CPU register

Registers and Main Memory due to large difference in access time.

* This results in lower performance of the system and thus enhancement was required.
* This enhancement was made in the form of Memory Hierarchy design because of which the performance of the system increases. (Cache Memory used)

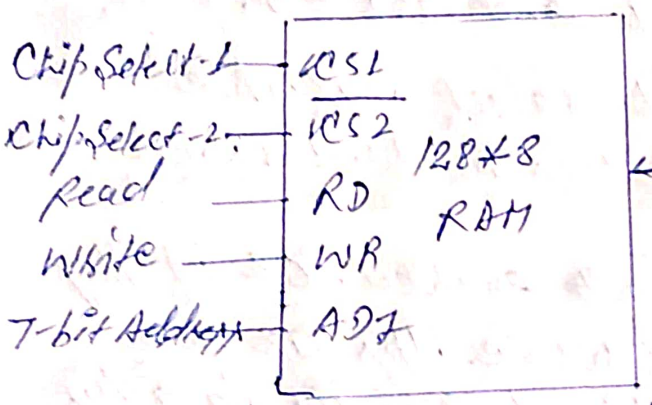
4. Cost per bit: → * As we move from bottom to top in the hierarchy, the cost per bit increases i.e. internal memory is costlier than external memory.



* I/O processors manages data transfer between auxiliary memory and main memory.

RAM and ROM Chips:

* A RAM chip is better suited for communication with the CPU if it has one or more control i/ps 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.
* The block diagram of a RAM chip is shown in below 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.



(a) Block Diagram

CS1	CS2	RD	WR	Memory Function	State of Data Bus
0	0	X	X	Inhibit	High-Impedance
0	1	X	X	Inhibit	High-Impedance
1	0	0	0	Inhibit	High-Impedance
1	0	0	1	Write	Input data to RAM
1	0	1	X	Read	Output data from RAM
1	1	X	X	Inhibit	High-Impedance

(b) Function-table

Fig: Typical RAM chip

* The Read and write i/ps specify the memory operation and the two chip select (cc) control i/p's are for enabling the chip only when it is selected by the cpu.
* 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 i/ps are sometimes combined into one line, labeled R/W. When the chip is selected, the two binary states in this line specify the two operation of Read or write.
* The function table listed in the previous page specifies the operation of the RAM chip. The unit is in operation only when $CS1 = 1$ and $\overline{CS2} = 0$.
* 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 i/p's are not enabled, or if they are enabled but the Read or write i/p's are not enabled, the memory is inhibited and its data bus is in a high impedance state.
* When $CS1 = 1$ and $\overline{CS_2} = 0$, the memory can be placed in a write or Read mode.
* When the WR i/p is enabled, the memory stores a byte from the data bus into a location specified by the address i/p lines.
* When RD i/p 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 buffer is associated with the bidirectional data bus.

* A ROM chip is organised 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 below diagram.
* For a same-size chips, 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.

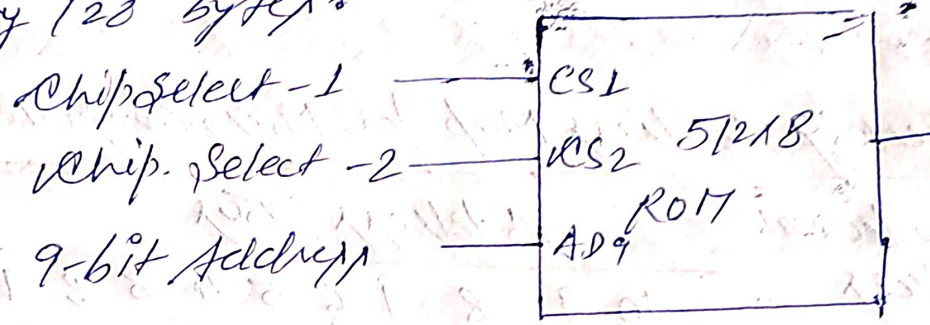


Fig: Typical ROM chip.

* The nine address lines in the ROM chip specify any one of the 512 bytes stored in it. The two chip select inputs must be CS1 = 1 and CS2 = 0 for the unit to operate.
* Otherwise, the data bus is in a high-impedance state. There is no need for a read or write control because the unit can only read.
* Thus when the chip is enabled by the two select i/ps, the byte selected by the address lines appears on the data bus.

Memory Address Map:→

* The addressing of a 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 needed 512 bytes of RAM and 512 bytes of ROM. The memory address map for this configuration is shown in below table:

Table: Memory Address Map for Micro-computer

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 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 input 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 different address.
* For this particular example we chose 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 RAM and ROM addresses 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 address for each chip is obtained from the information

from the Byte section enables the address bus assigned.

* The address bus lines are subdivided into groups of four lines each so that each group can be represented with a hexadecimal digit.
* The first hexadecimal digit represent lines 8 to 15 and it always 0.
* The next hexadecimal digit represents lines 4 to 7 but lines 11 and 12 are always 0.
* ~~The next hexadecimal digit etc~~
* The range of hexadecimal address for each component is determined from the X's associated with it. These X's represents a binary number that can range from an all 0's to an all 1's value

Memory Connection to CPU:→

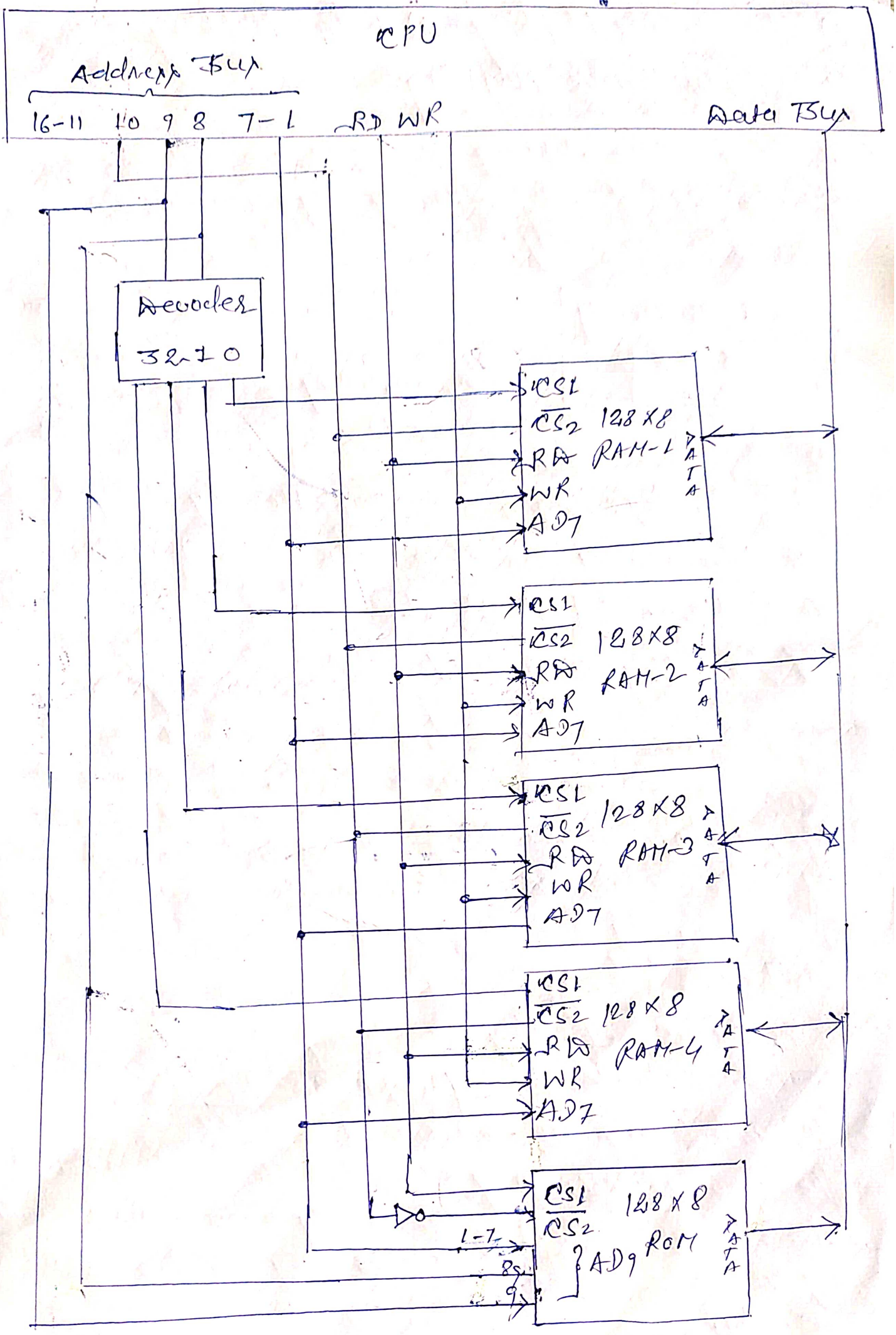


Fig : Memory connection to the CPU

Cache Memory :→

* If the active portions of the program and data are placed in a fast small memory, the average memory access time can be reduced, thus reducing the total execution time of the program. Such a small fast memory is referred to as a cache memory.
* It is placed between the CPU and main memory as illustrated in memory hierarchy.
* 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 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.
* For Ex: 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 avg. access time of 200 ns. Without a cache memory, whose access time is 1000 ns.
* The transformation of data from main memory to cache memory is referred to as a mapping process.
* Three types mapping procedure for cache memory.

1) Associative Mapping 2) Direct Mapping
3) Set-Associative Mapping

* For these three Mapping procedure we will use a specific example of a memory organization as shown in below fig:

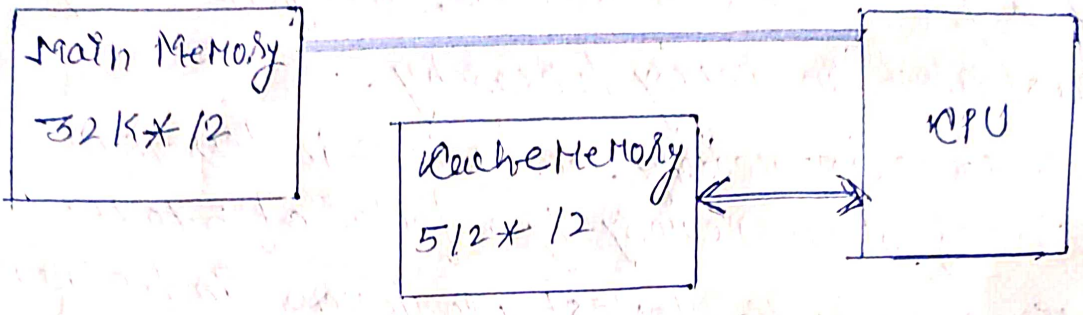


Fig: Example of Cache Memory.

* 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 words stored in cache, there is a duplicate copy in main memory. The CPU communicate with both memories. It first sends a 15-bit address to cache. If there is a hit, the CPU accept the 12-bit data from cache. If there is a miss, the CPU reads the words from main memory and the word is then transferred to cache.

Associative Mapping:→

* The fastest and most flexible cache organization uses an associative memory.
* The associative memory stores both the address and content (data) of the memory word. This permits any location in cache to store any word from main memory.

* 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.

CPU address (15 bits)

↓

Argument Register	
← Address →	← Data →
01000	3450
02777	6710
22345	1234

* 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 for replacement pair is done on the basis of replacement algorithm that designer chooses for the cache.

Direct Mapping :-

* Associative memories are expensive compared to RAMs because of the added logic associated with each cell.
* The possibility of using a RAM for the cache is investigated in fig: below.

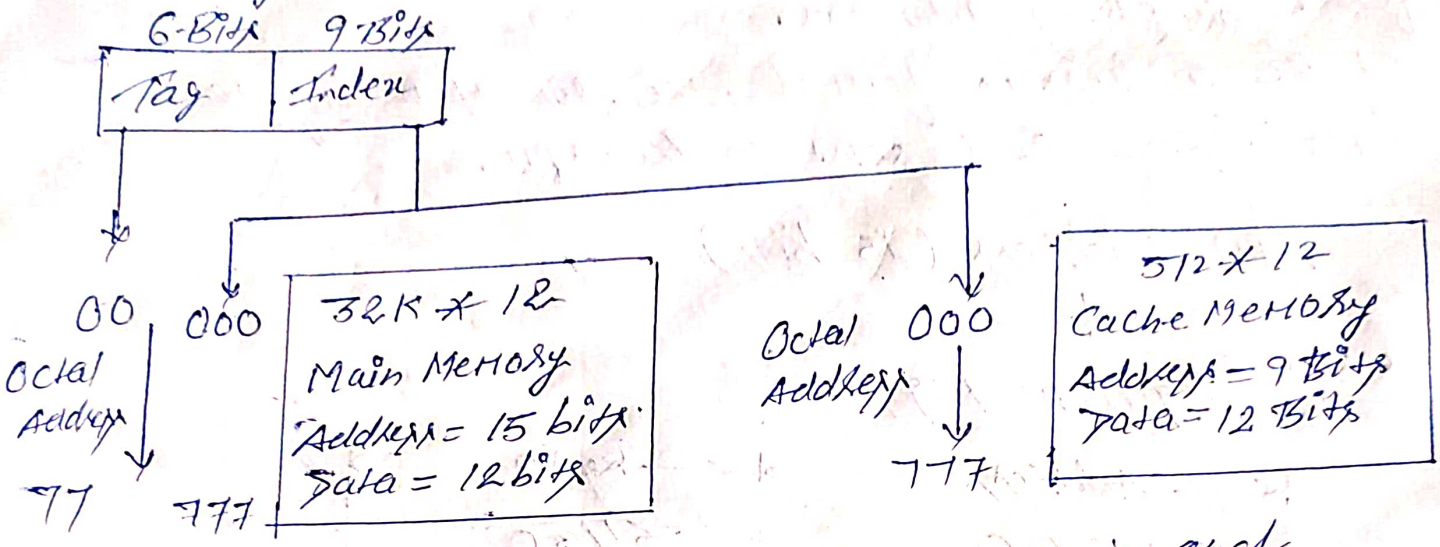


Fig: Addressing Relationship between Main and Cache Memory.

* 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 number of bits in the index field is equal to the number of address required to access the cache memory.
* In general case, there are 2K words in cache memory and 2n words in Main Memory.
* The n-bit memory address is divided into two fields : K bits for the index field an n-K bits for the tag field.
* The direct mapping cache organisation uses the n-bit address to access the main memory and

the 9-bit index to access the cache.

* The internal organization of the words in the cache memory is shown in fig b.

* Each word in cache consists of the data word and its associated tag. ~~Whenever a word is read from the cache, its associated tag is compared with the tag of the address.~~

* 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 there is no match, there is 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.

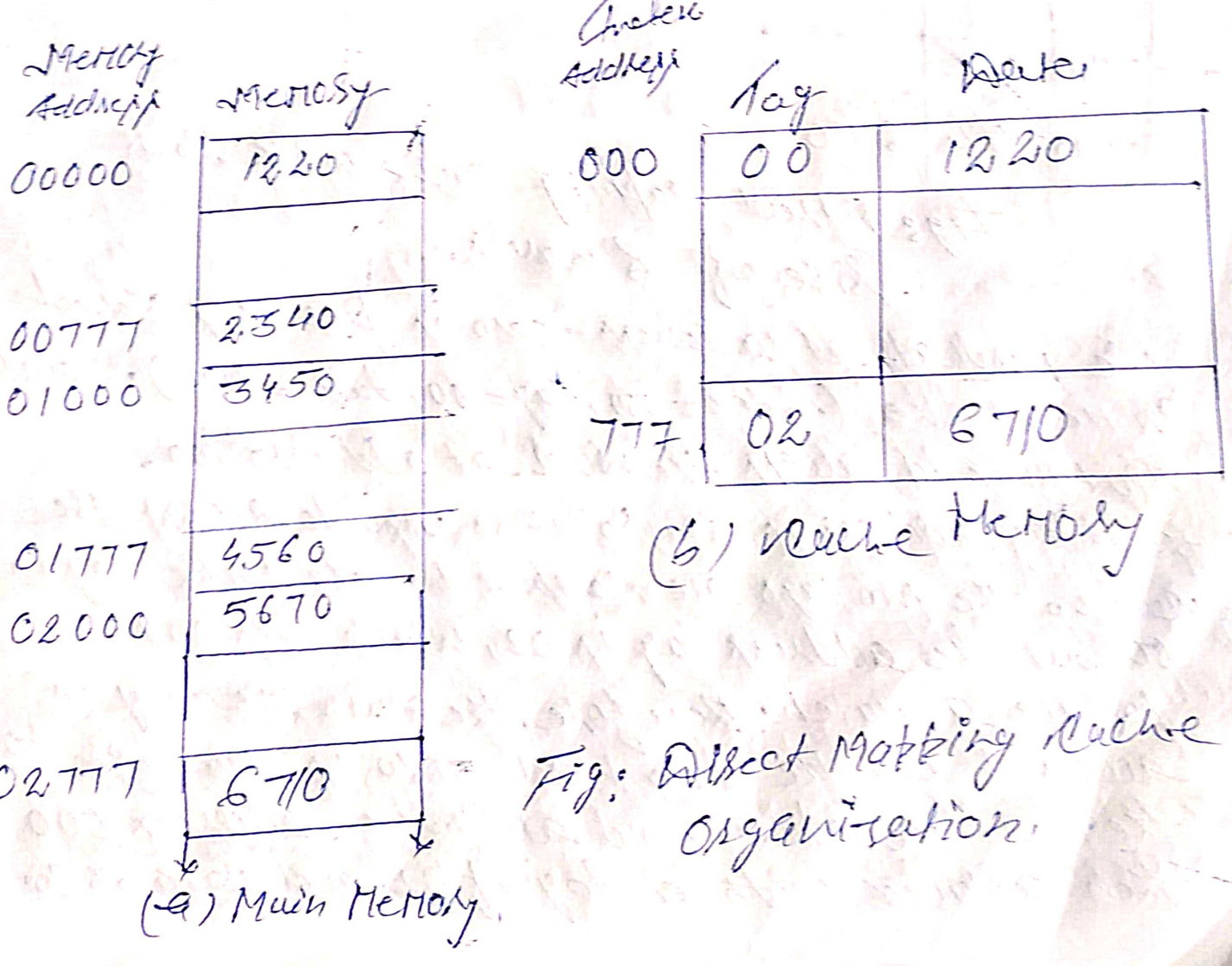
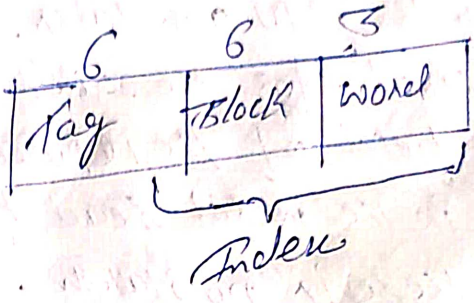


Fig: Direct Mapping Cache organization.

* 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 the previous example).

* The direct mapping is shown in previous fig.



	Index	Tag	Data
Block 0	000	01	3450
	007	01	6578
Block 1	010		
	017		
Block 63	770	02	
	777	02	6710

Fig. Direct Mapping Cache with block size of 8 words.

* The word at address zero is presently stored in the cache (index = 000, tag = 00, data = 1220). And CPU 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 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 transfer to the CPU. The cache word at index address 000 is then replaced with a tag of 02 and data 5670.

* The direct mapping example just described uses a block size of one word. The same organization but using a block size of 8 words is shown in fig on the previous page.
* 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*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.
* It requires extra time, the hit ratio will most likely improve with a larger block size because of the sequential nature of computer programs.

Set-Associative Mapping:

* Set-associative mapping is an improvement over the direct mapping organization in that each word of cache can store two or more words of memory under the same index address.
* Each data word is stored together with its tag and the number of tag-data items in one word of cache is said to form a set.
* Each index address refers to two data words and their associated tags.

Each tag requires six bits and each data word has 12 bits, so the word length is 2(6+12) = 36 bits. An index address of nine bits can accomodate 512 words. Thus the size of cache memory is 512 x 36. It can accomodate 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 accomodate 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 number listed in above fig are with reference to the main memory contents.
* 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 name "Set-associative"

* The hit ratio will improve as the size increases because more word with the same index but different tags can reside in cache.
* However, an increase in the set size increase the number of bits in word 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 item with a new value. The most common replacement algorithm are, FIFO, LRU.

Writing into cache:→

* When the CPU finds a word in cache during a read operation, the main memory is not involved in the transfer.
* However, if the operation is a write, there are two ways that the system can proceed.
* The simplest and most commonly used procedure is to update main memory with every memory write operation with cache memory being updated with every memory write operation, with cache memory being updated in parallel if it contains the word at the specific address. This is called the write-through method.
* This method has the advantage that main memory always contains the same data as the cache. This characteristic is important in system 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 update 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 needs be rewritten into main memory.

* Number of memory writes in a typical program ranges between 10 and 50 percent of the total reference to memory.

Cache Initialization:→

* The cache is initialized when the 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, but in effect it contains some nonvalid data. It is customary to include with each word in cache a valid bit to indicate whether or not the word contains valid data.

→ The cache is initialized by clearing all the valid bits to 0. The valid bit of a particular cache

word is set to 1 the first time this word is loaded from Main Memory and stays set unless the cache has to be Initialized again.

* The Introduction of the valid bit means that a word in cache is not replaced by another word unless the valid bit is set to 1 and a mismatch of tags occurs.
* If the valid bit happens to be 0, the new word automatically replaces the invalid data. Thus the Initialization condition has the effect of forcing misses from the cache until it fills with the valid data.