MEMORY HIERARCHY →

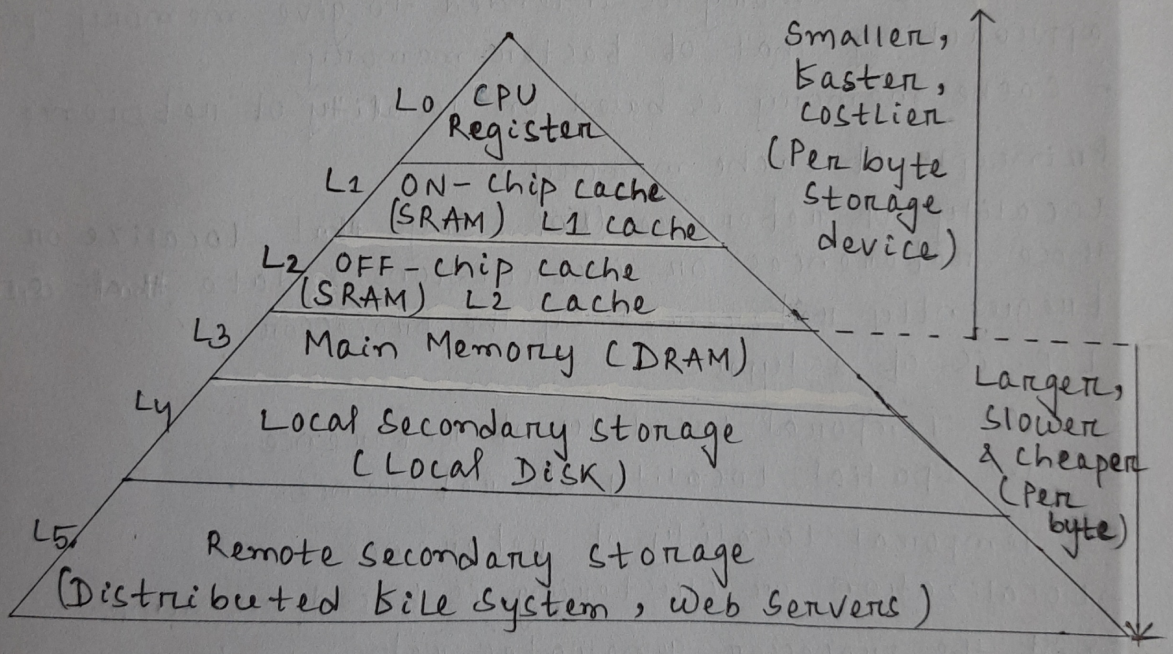
The hierarchial arrangement of storage in current computer architecture is called memory hierarchy. Some fundamental properties of hardware & software are,

(a) Fast storage technology cost more per byte & have less capacity.

(b) Gap between CPU & main memory speed is widening.

(c) Well-written programs tend to exhibit good locality & the type of localities are - temporal & spatial.

Therefore these fundamental properties suggest an approach for organizing memory & storage system as a memory hierarchy.



→ CPU Register holds words derived from L1 cache.

→ L1 cache holds lines read from L2 cache memory.

→ L2 cache holds lines or blocks read from main memory

→ Main memory holds disk blocks retrieved from Local disk.

→ Local disk holds files read from disk on remote network servers.

→ As the capacity increases for storing information in memory, then the access time also increases and consequently the cost per bits for storing also increases.

Cache Memory – It is a high speed memory, small in size that increase the processing speed of processor by making current program & data available to the processor in a faster rate. Cache memory is used to store temporarily those data which are frequently used.

Main Memory – The memory unit that communicate directly with the processor is called main memory. The main memory stores those data that are currently needed by the processor.

Auxillary Memory – It is a low cost storage device to serve as a back up for storing the information that is not currently needed by the processor.

CACHE MEMORY

→ A cache memory is intended to give memory speed approaching that of faster memory.
→ Cache memory is based on locality of reference principle

Principle of cache memory →
Locality of reference (LOR) says that localize or cluster those references or instructions & data that are frequently referenced by the processor.
LOR is of 2 types –

(a) Temporal Locality of reference.
(b) Spatial Locality of reference.

a) Temporal Locality of reference –
→ Localization or clustering is based on time.
→ If the processor generates reference for an instruction or data now then there is a possibility that the same instruction or data or word is also referenced again in near future by the processor.

b) Spatial Locality of reference –
→ Localization or clustering based on space.
→ If the processor generates reference for a word in memory then there is possibility that it's near by or neighbour words can also be referenced by the Processor in near future.

Cache Memory Process of Retrieval →

1. Cache memory is intended to give memory speed approaching that of faster memory.

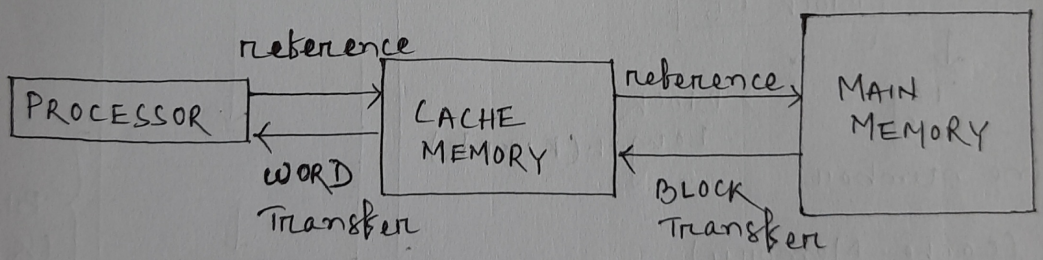
2. The cache contain a copy of main memory. When the processor attempts to read a word from memory, a check is made to determine if the word is in the cache or not.

→ If the word is present in the cache memory then the word is delivered to the processor.

→ If the word is not present in cache memory then a block of main memory consisting of some fixed no. of words, where the searched word is present is read from main memory to cache memory & then the searched word is delivered to the processor.

3. Because of the phenomenon of locality of reference, when a block of data is fetched or read into the cache memory to satisfy a single memory reference, it is likely that there will be future references to that same memory location or word or other memory location or other words of same block which is transferred to the cache memory.

4.



5. Structure of Main memory →

→ Main memory is addressable by n-bit. Therefore, total no. of memory locations in main memory = 2^n

⇒ No. of memory locations varies from 0 to $(2^n - 1)$

→ For mapping main memory consist of K-words in each block.

⇒ Total no. of blocks present in main memory = $\frac{2^n}{K}$

⇒ $M = \frac{2^n}{K}$

where M = no. of blocks in main memory.

6. Structure of cache memory →

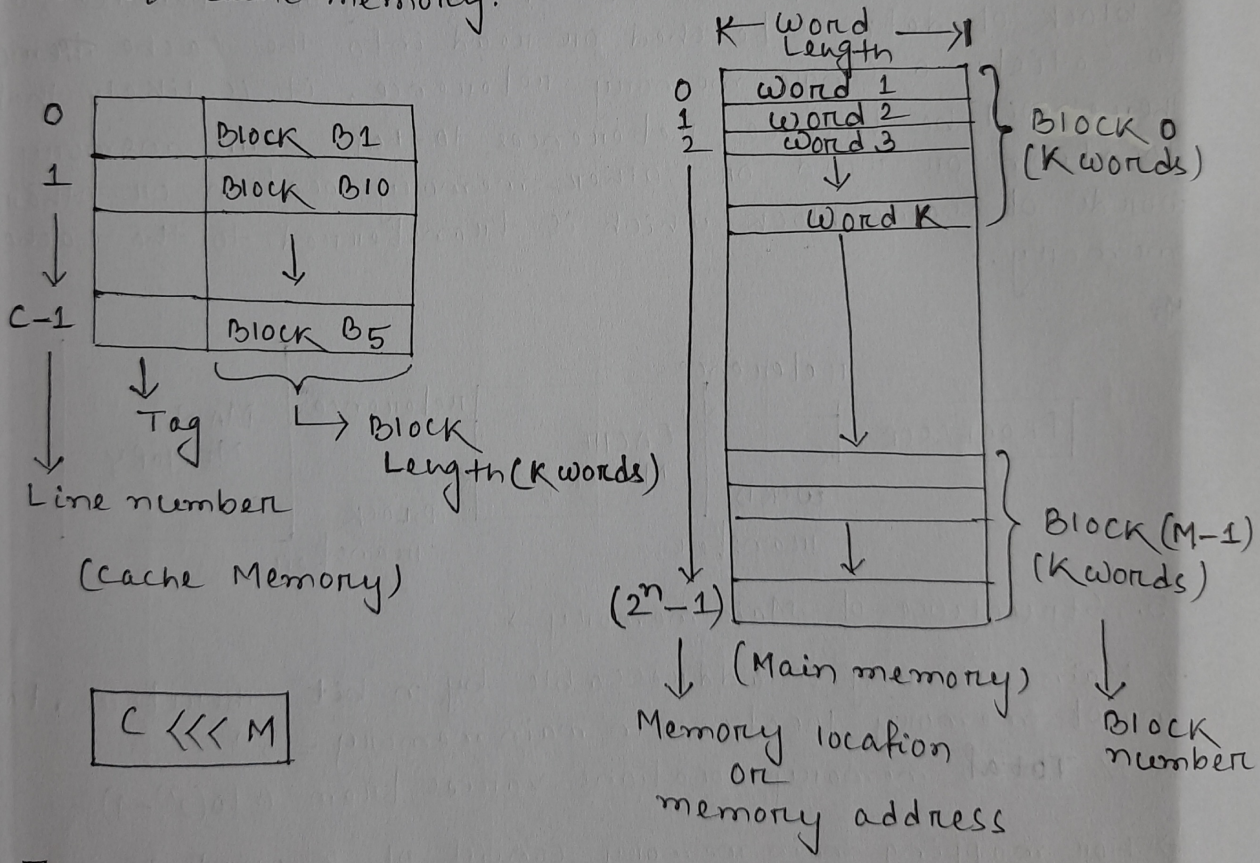
→ Cache memory consist of 'C' lines, varies from 0 to C-1. The cache memory divided into lines, where each lines consists of K-words. Each line contain K words & few bits of tag.

→ No. of words present in each line is known as line size.

→ No. of lines present in cache memory is less than no. of blocks present in main memory. i.e. $C \lll M$.

→ If a block of main memory is read then that block is transferred to one of line of cache memory.

As there are more no. of blocks than lines, so an individual line can not be dedicated to a particular block. So a tag is included in each line of cache to identify which block of main memory is stored in which line of cache memory.



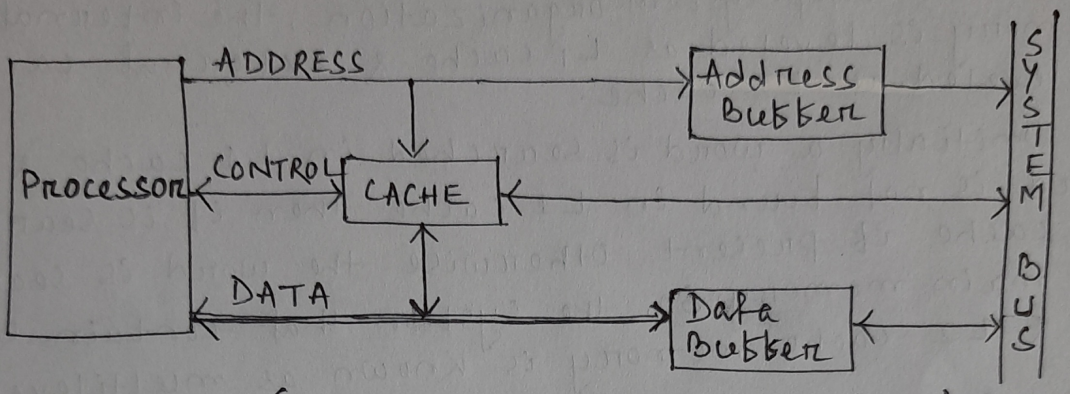
$C \lll M$

7. The processor generates the address or references of a word to be read.

→ The cache connects to the processor via data, control & address lines. The data & address lines also attached to the data & address buffer respectively which is attached to the system bus from which main memory reached.

→ when a cache hit occurs the data & address buffers are disabled & the communication is between the processor & cache (NO system bus traffic)

→ when a cache miss occurs, the desired address is loaded into the system bus & data are read through data buffer to both cache & the processor.



(Cache memory organization)

Types of Cache Memory →

1. Single cache Vs Multilevel cache.
2. ON-chip cache Vs OFF-chip cache.
3. Unified Vs split cache.

1) Single cache Vs multilevel cache -

→ If the computer system contain only one cache memory connected to the processor then it is known as single cache.

→ If more than one cache memory connected to the processor in the computer system then it is known multilevel cache memory.

2) ON-chip cache Vs OFF-chip cache memory -

→ A cache on the same chip as the processor is called ON-chip cache. It is internal to the processor. An ON-chip cache memory interact with processor by data path & address path internal to the processor. Hence access time as well as execution time will be less. Hence overall performance increases.

It is faster than OFF chip cache memory.

→ A cache that is not on the same chip as that of the processor is called OFF-chip cache memory. It is external to the Processor chip. An OFF-chip cache interact with the Processor chip via external system bus. Hence access time as well as execution time increases that consequently reduces the processor performance.

→ In some comp. system organization, the internal cache memory is leveled as L1 cache & external cache mem. is leveled as L2 cache.

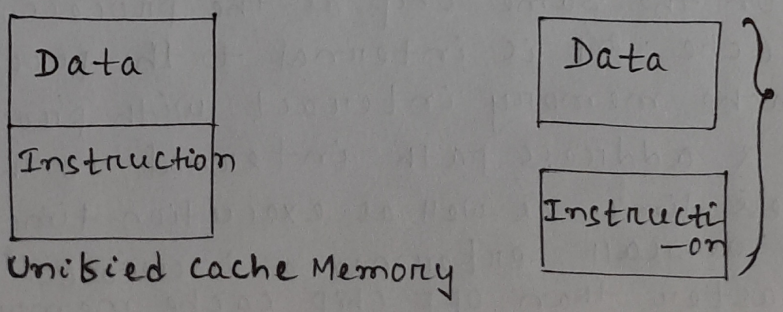
Initially a word is searched in L1 cache & if the word is not found in L1 cache then it is searched in L2 cache if present otherwise the word is searched in main memory. So, the system that contain both L1 & L2 cache memory is known as multilevel cache.

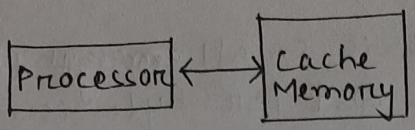
→ In some comp. system L1 is ON-chip & L2 is OFF-chip memory but some other comp. system has both L1 & L2 ON-chip & L3 is OFF-chip cache memory.

3) Unified vs split cache -

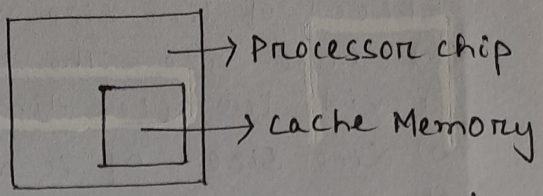
→ If a single cache memory is used to store references to both instructions & data then it is called unified cache memory. unified cache has higher hit ratio than split cache because unified cache balances both instruction & data.

→ If the cache is split into two, one is divided into two that is one cache is dedicated to store references to instruction only & the other dedicated to data is called split cache. It has smaller hit ratio. split cache is useful when parallel processing is used or to handle resource dependency in pipeline to avoid resource hazard.

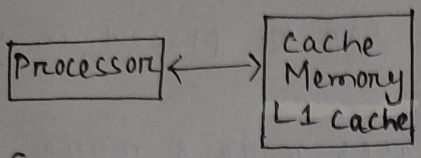




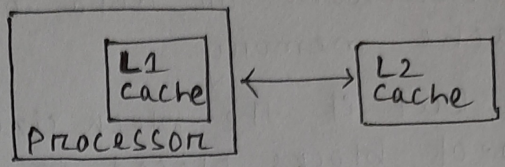
(OFF-chip cache Memory)



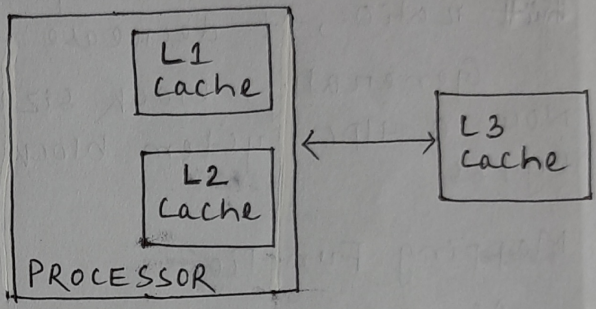
(ON-chip cache Memory)



(Single Level cache
or
Level 1 cache Memory)



or



(Multiple cache Memory)

Performance of cache Memory →

The performance of cache memory is described by two terms. They are,

(a) cache hit

(b) cache miss

→ If a searched word is found in cache memory then it is called as cache hit.

→ If the searched word is not found in cache memory then it is called as cache miss.

$$\text{Hit Ratio} = \frac{\text{Total no. of hits}}{\text{Total no. of references generated by the Processor.}}$$

$$\text{Miss Ratio} = \frac{\text{No. of miss}}{\text{Total no. of references generated by the processor.}}$$

$$\Rightarrow \boxed{\text{Hit Ratio} = 1 - \text{Miss Ratio}}$$

Effect of line size & cache size →

(a) Cache Size – The size of cache is small enough to provide less access time. But the cost per byte is more.

(b) Line Size – It is also can be considered as block size. If the block size increases then the hit ratio will also increases. This is because more no. of near by or neighbour instructions or data or words can be placed in the cache memory.

But if the block size increases more & more then no. of blocks placed in cache memory of fixed size then decreases. Consequently, rather than increase in hit ratio, it decrease the hit ratio.

Generally, block size varies from 8 to 64 bytes. Now a HPC system block size varies from 64 to 128 bytes.

Mapping Function →

As C << M, so algorithms are needed for mapping the main memory blocks (M) into cache memory lines (C). The mapping function is needed to identify which main-memory block (M) currently stored in cache memory lines (C). Alternatively, the mapping technique also shows how the cache memory is organized.

There are 3 different types of mapping functions in cache memory. These are –

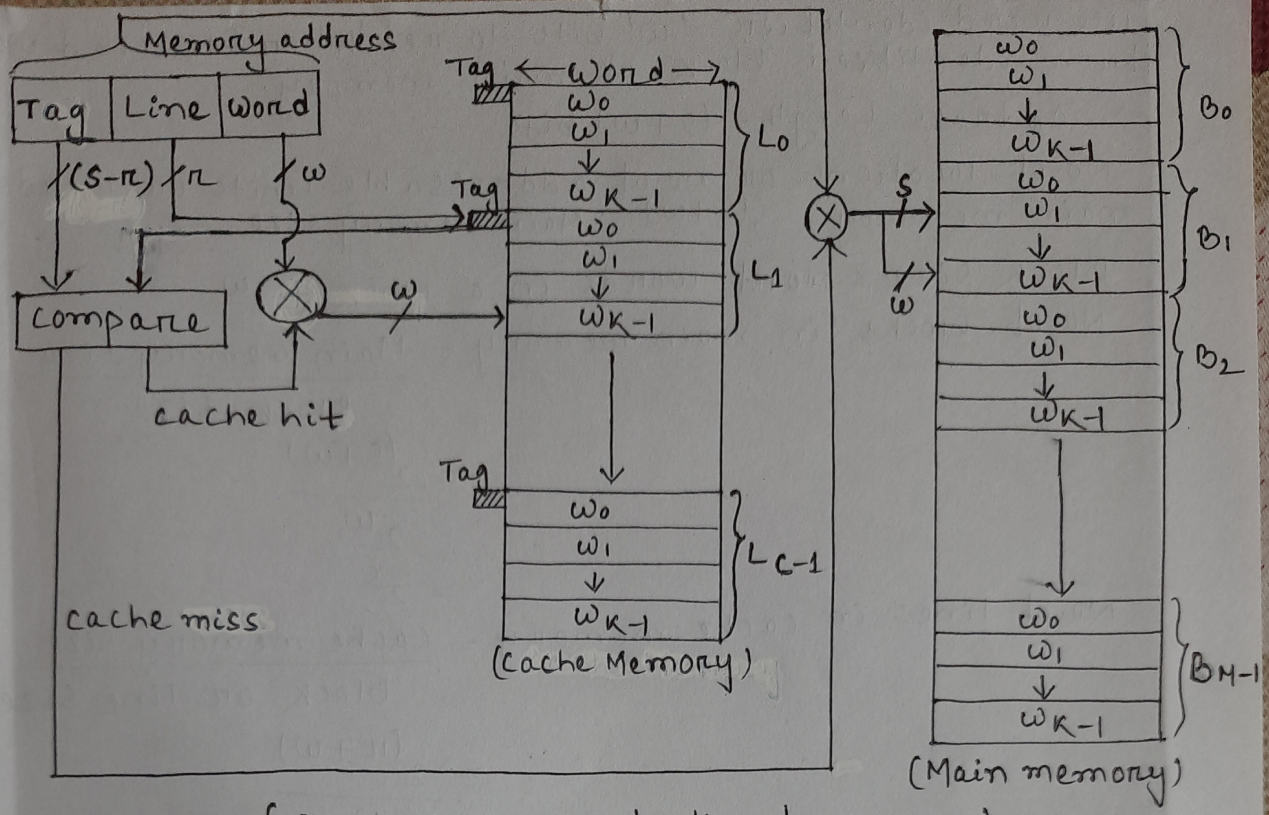
1. Direct Mapping
2. Associative Mapping
3. Set-Associative Mapping.

1. Direct Mapping –

In direct mapping, it maps each block of main memory into only one possible cache lines. The mapping is expressed as,

$$i = j \text{ modulo } m$$

where i = cache memory line number.
j = Main-memory block number.
m = no. of lines in cache memory.



(Block Diagram of direct Mapping)

Each memory address consist of 3 different fields. These are- Tag | Line | word

(a) Least significant 'w' bits are used to identify a unique word within a block of main memory.

(b) 's' bits of memory address are used to specify one of 2^s blocks of main memory.

(c) 'r' bits are used to specify one of 2^r lines of cache memory.

→ Use Line identifier 'r' bits to identify the selected line within which the block (searched) can present.

→ Now match the tag field of the memory address i.e. (S-r) bits with the tag field of the identified line.

→ if it is a cache hit after matching then use word identifier 'w' bits to read the searched word. And if it is a cache miss then use the memory address that is generated by processor i.e. (S+w) bits to read the searched word from main memory.

→ To read from main memory initially use block identifier 's' bits to identify the block where the search word is present. After identifying the block then

use word identifier 'w' bits to read the searched word from identified block of main memory.

→ Address Length = (s+w) bits

No. of locations or no. of addressable units or words in main memory = $2^{(s+w)}$ = Main memory size

Block size = No. of words in a block = 2^w

No. of blocks in main memory = $\frac{\text{Main memory size}}{\text{Block size}}$

$= \frac{2^{(s+w)}}{2^w}$

$= 2^s$

No. of lines in cache memory = $\frac{\text{cache memory size}}{\text{Block or line size}}$

$= \frac{2^{(r+w)}}{2^w}$

$= 2^r$

No. of locations or no. of addressable units or words in cache memory = $2^{(r+w)}$ = cache memory size

Block size = Line size = 2^w

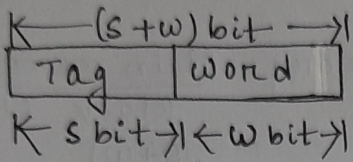
⇒ No. of words in line = 2^w.

Drawback of direct mapping –

It is simple & inexpensive to implement but the drawback here is, for any given block of main memory there is a fixed line of cache. So if a program references words repeatedly from two different blocks that map into the same line then the blocks are continuously swapped in & swapped out from from cache memory. This results in decrease of hit ratio. Such a phenomenon is known as thrashing.

2. Associative Mapping –

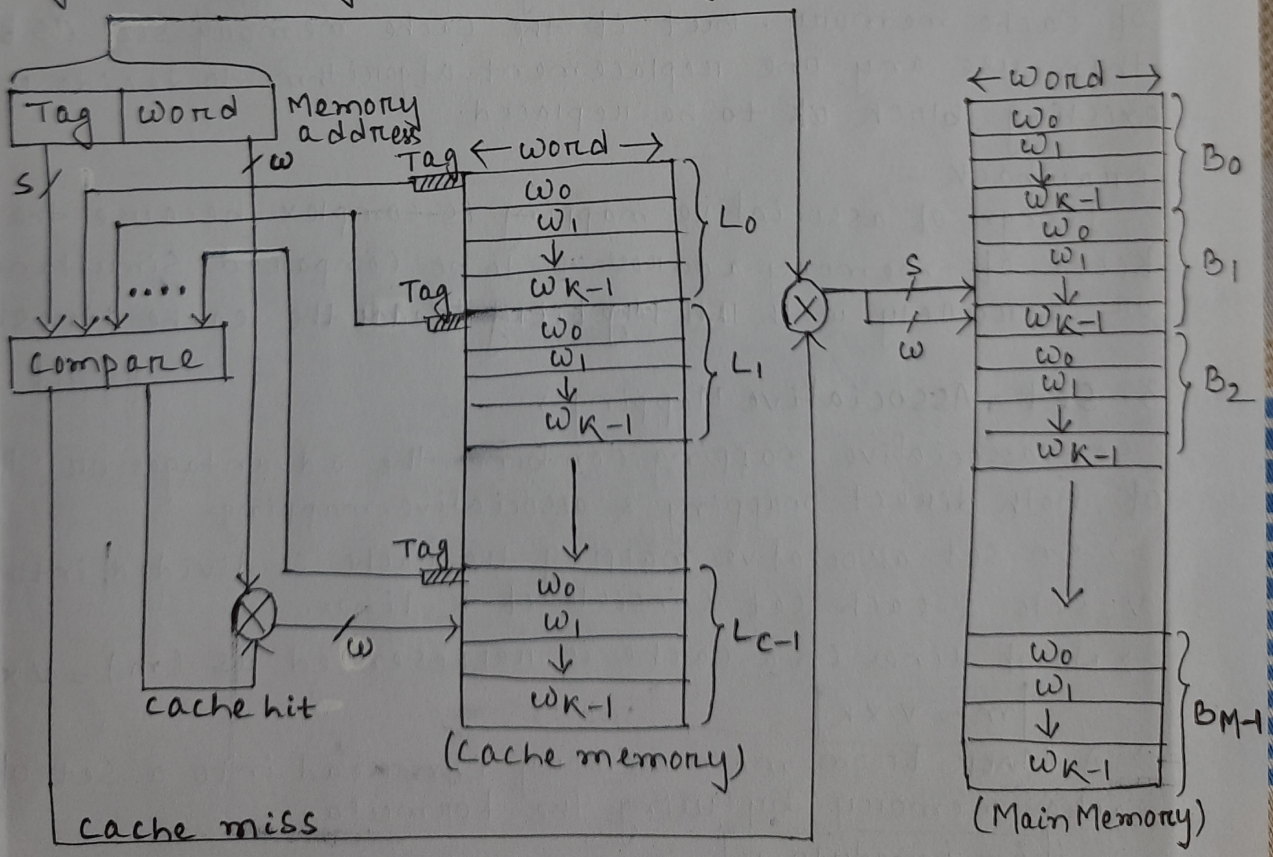
Associative mapping overcomes the drawback of direct mapping known as thrashing where the direct mapping can permits each main memory block to be loaded into a fixed line of cache memory. But in associative mapping any block of main memory can be loaded into any line of cache memory. Therefore the memory address consist of only two fields – Tag & word.



→ Tag field is of s-bit that uniquely identify block of main memory.

→ word field is of w-bits that is used to identify a word within the identified block.

→ To examine whether a block is in cache or not, the cache control logic simultaneously examine every line's tag with the tag field of memory address for a match.



(Block Diagram of associative mapping)

→ Address length = (s+w) bits
⇒ No. of addressable locations or words in main memory can be = $2^{(s+w)}$ = Main memory size.
No. of address bits in word field or word identifier field = w bits.
⇒ No. of words in the block or line = 2^w
⇒ Block size = line size = 2^w.

No. of blocks in main memory = $\frac{\text{Main memory size}}{\text{Block size}}$

$= \frac{2^{(s+w)}}{2^w}$

$= 2^s$

Or, if block identifier field is of 's' bits then no. of blocks in main memory = 2^s.

Advantage of associative mapping –
A block in main memory can be mapped to any line of cache memory. But if the cache memory size is full then use any one replacement algorithm to decide which existing block is to be replaced.

Drawback –
Design of associative mapping is complex because the tag field of memory address is to be compared simultaneously or parallely with the tag field of all the cache lines.

3. Set-Associative Mapping →
Set associative mapping combines the advantages or strength of both direct mapping & associative mapping.
→ In set associatve mapping the cache is divided into v-sets & each set consists of k-lines.
→ No. of lines in a cache is represented as (m) = v x k

$$\boxed{m = v \times k}$$

→ A block from main memory can read into a set of cache memory by using the formula,

$$\boxed{i = j \text{ modulo } v}$$

where j = Block number of main memory.

i = Set number of cache memory
v = no. of sets present in cache memory.

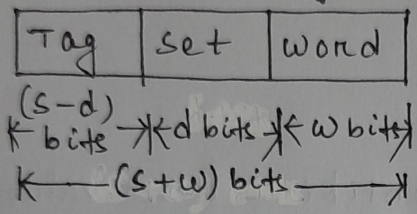
Now, m can be expressed as follows,

$m = K \times v$

$\Rightarrow \boxed{m = K \times 2^d} \quad (\because v = 2^d)$

→ Set associative is also known as K-way set associative mapping. Block Bj of main memory mapped to any line of a fixed set i of cache memory.

→ Memory address of set-associative mapping consist of 3 different fields. They are,



→ Using set field identifier value 'd' bits, identify one of the set of the cache memory.

If set identifier is of d-bits then no. of sets in the cache memory = 2^d.

Now, the tag field (s-d) bits are used to specify which block of main memory is present in which line of the identified cache memory.

After identifying the line of cache memory, the word identifier w-bits are used to specify or identify a word in the line or block.

→ In set associative mapping the tag in memory address is compared with tag of every line of a set that is specified in memory address. (matching is done with K-tags of a set).

→ In fully associative mapping the tag in memory address is very large or there is only one set in cache memory. Hence only the associative mapping is known as the fully associative mapping. (K = 1 in associative or fully associative mapping).

→ Address length = (s+w) bits.

⇒ No. of addressable locations or words = $2^{(s+w)}$

⇒ Main memory size = $2^{(s+w)}$

Word identifier = w bits
⇒ no.of words in a block or line = 2^w.

No.of blocks in main memory = $\frac{\text{Main memory size}}{\text{Block size}}$

$= \frac{2^{(s+w)}}{2^w}$

$= 2^s$

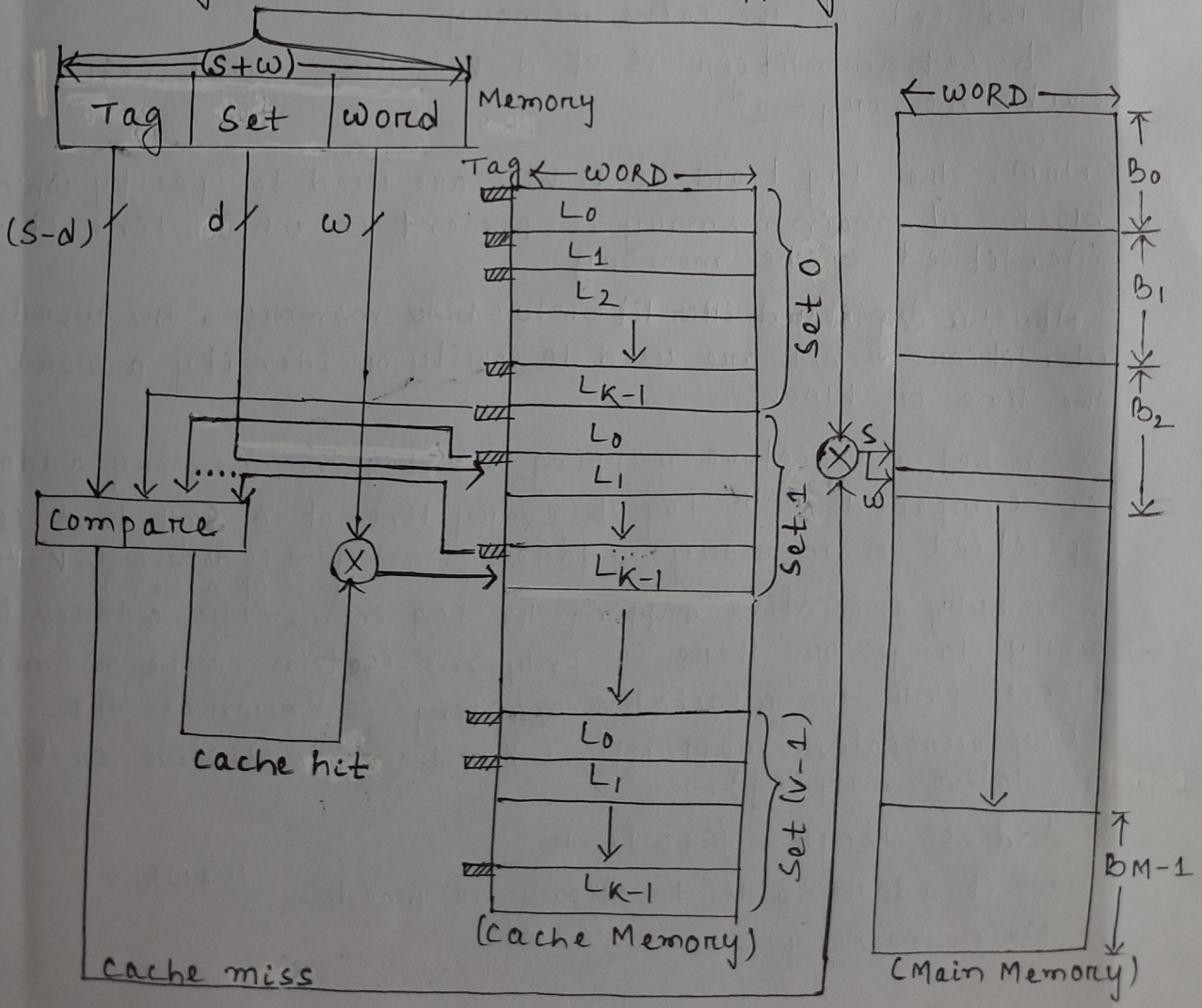
No.of lines in a set = K & no.of sets in cache memory is = V. Hence total no.of lines present in cache memory can be, K x v.

$m = K \times v$

⇒ $\boxed{m = K \times 2^d}$

where $v = 2^d$ & d = set identifier field value.

Block diagram of set-associative mapping →



Replacement Algorithm →

If the cache memory is full & a new block is read from main memory into cache memory then one of the existing block of cache memory is replaced by the new incoming block from main memory.

Different types of replacement algorithms are,

1. LRU
2. LFU
3. FIFO
4. optimal
5. Random

Writing Policy →

If a block in cache memory is replaced then the following cases can be happened -

→ If the old block which is going to be replaced has not been altered, then only the new block overwrite the old existing one.

→ If atleast one write operation has been performed on a word in a line then that line of cache must be write onto block of main memory which is called updating main memory & this writing should be done before bringing a new block onto the cache line.

→ If a cache word is altered then the corresponding main memory word is invalid. If the main memory word has altered, then the corresponding word in cache is invalid.

→ Another problem also arise when multiprocessor used, in this case each processor has its local cache memory & one shared main memory. Then if a word is altered in one cache, it will invalidate that corresponding word in other cache memory.

→ There are different writing policies. They are,

1. Write through
2. write back
3. write once

1. Write through – All write operations are made to main memory as well as to the cache memory. This is done to keep the main memory always valid.

Therefore in multiprocessor system all the processors need to monitor the main memory to maintain consist – ency within its own local cache memory.

Drawback – As monitoring to main memory is done by processors to maintain consistency this increase memory traffic.

2. Write back – In write back policy all the updates are made in cache memory. When an update occur in a word in cache memory, then an update bit associated with the line is set. And when a block is to be replaced from cache then it is written back to main memory if & only if the update bit is set.

Inspite of all these writing policies in multiprocessor system, the local cache memory may contain invalid data. So the ystem which prevents these problem is called cache coherency.

Drawback – As main memory is not always updated immediately after every write operation in cache memory, so some portion of main memory is invalid. Therefore write back policy is used in complex circuit design.