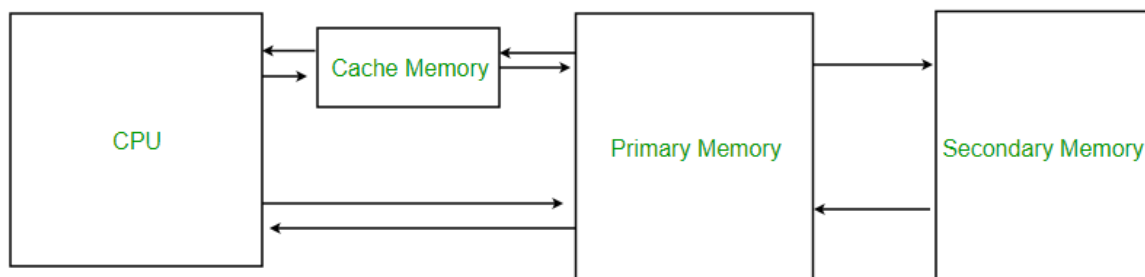


Cache Memory is a special very high-speed memory. It is used to speed up and synchronize with high-speed CPU. Cache memory is costlier than main memory or disk memory but more economical than CPU registers. Cache memory is an extremely fast memory type that acts as a buffer between RAM and the CPU. It holds frequently requested data and instructions so that they are immediately available to the CPU when needed. Cache memory is used to reduce the average time to access data from the Main memory. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various different independent caches in a CPU, which store instructions and data.



Levels of memory:

- **Level 1 or Register** – It is a type of memory in which data is stored and accepted that are immediately stored in CPU. Most commonly used register is accumulator, Program counter, address register etc.
- **Level 2 or Cache memory** – It is the fastest memory which has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory** – It is memory on which computer works currently. It is small in size and once power is off data no longer stays in this memory.
- **Level 4 or Secondary Memory** – It is external memory which is not as fast as main memory but data stays permanently in this memory.

Cache Performance: When the processor needs to read or write a location in main memory, it first checks for a corresponding entry in the cache.

- If the processor finds that the memory location is in the cache, a **cache hit** has occurred and data is read from the cache.
- If the processor **does not** find the memory location in the cache, a **cache miss** has occurred. For a cache miss, the cache allocates a new entry and copies in data from main memory, then the request is fulfilled from the contents of the cache.

The performance of cache memory is frequently measured in terms of a quantity called **Hit ratio**.

$$\text{Hit ratio} = \text{hit} / (\text{hit} + \text{miss}) = \text{no. of hits} / \text{total accesses}$$

We can improve Cache performance using higher cache block size, and higher associativity, reduce miss rate, reduce miss penalty, and reduce the time to hit in the cache.

Cache Mapping: There are three different types of mapping used for the purpose of cache memory which is as follows: Direct mapping, Associative mapping, and Set-Associative mapping. These are explained below.

A. Direct Mapping

The simplest technique, known as direct mapping, maps each block of main memory into only one possible cache line. or In Direct mapping, assign each memory block to a specific line in the cache. If a line is previously taken up by a memory block when a new block needs to be loaded, the old block is trashed. An address space is split into two parts index field and a tag field. The cache is used to store the tag field whereas the rest is stored in the main memory. Direct mapping's performance is directly proportional to the Hit ratio.

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

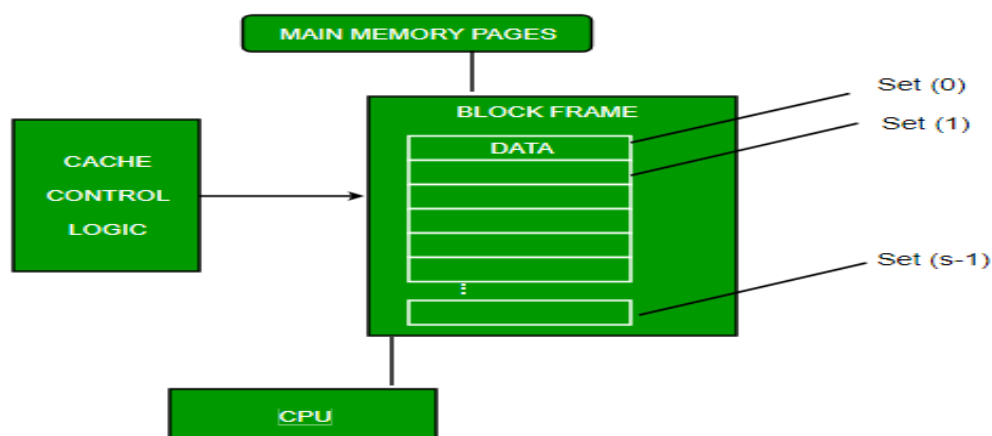
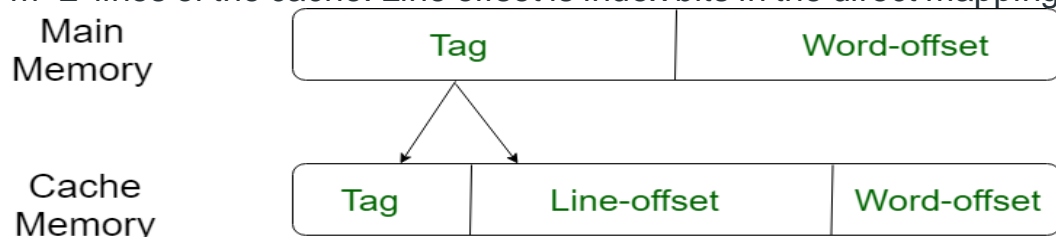
where

i =cache line number

j = main memory block number

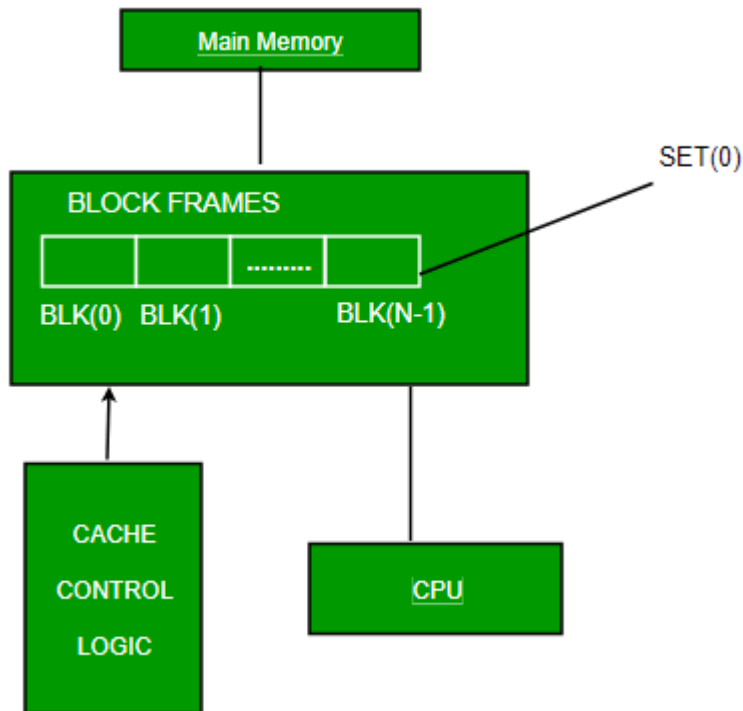
m =number of lines in the cache

1. For purposes of cache access, each main memory address can be viewed as consisting of three fields. The least significant w bits identify a unique word or byte within a block of main memory. In most contemporary machines, the address is at the byte level. The remaining s bits specify one of the 2^s blocks of main memory. The cache logic interprets these s bits as a tag of $s-r$ bits (most significant portion) and a line field of r bits. This latter field identifies one of the $m=2^r$ lines of the cache. Line offset is index bits in the direct mapping.



B. Associative Mapping

In this type of mapping, the associative memory is used to store content and addresses of the memory word. Any block can go into any line of the cache. This means that the word id bits are used to identify which word in the block is needed, but the tag becomes all of the remaining bits. This enables the placement of any word at any place in the cache memory. It is considered to be the fastest and the most flexible mapping form. In associative mapping the index bits are zero.



C. Set-associative Mapping

This form of mapping is an enhanced form of direct mapping where the drawbacks of direct mapping are removed. Set associative addresses the problem of possible thrashing in the direct mapping method. It does this by saying that instead of having exactly one line that a block can map to in the cache, we will group a few lines together creating a **set**. Then a block in memory can map to any one of the lines of a specific set. Set-associative mapping allows that each word that is present in the cache can have two or more words in the main memory for the same index address. Set associative cache mapping combines the best of direct and associative cache mapping techniques. In set associative mapping the index bits are given by the set offset bits. In this case, the cache consists of a number of sets, each of which consists of a number of lines. The relationships are

$$m = v * k$$

$$i = j \text{ mod } v$$

where

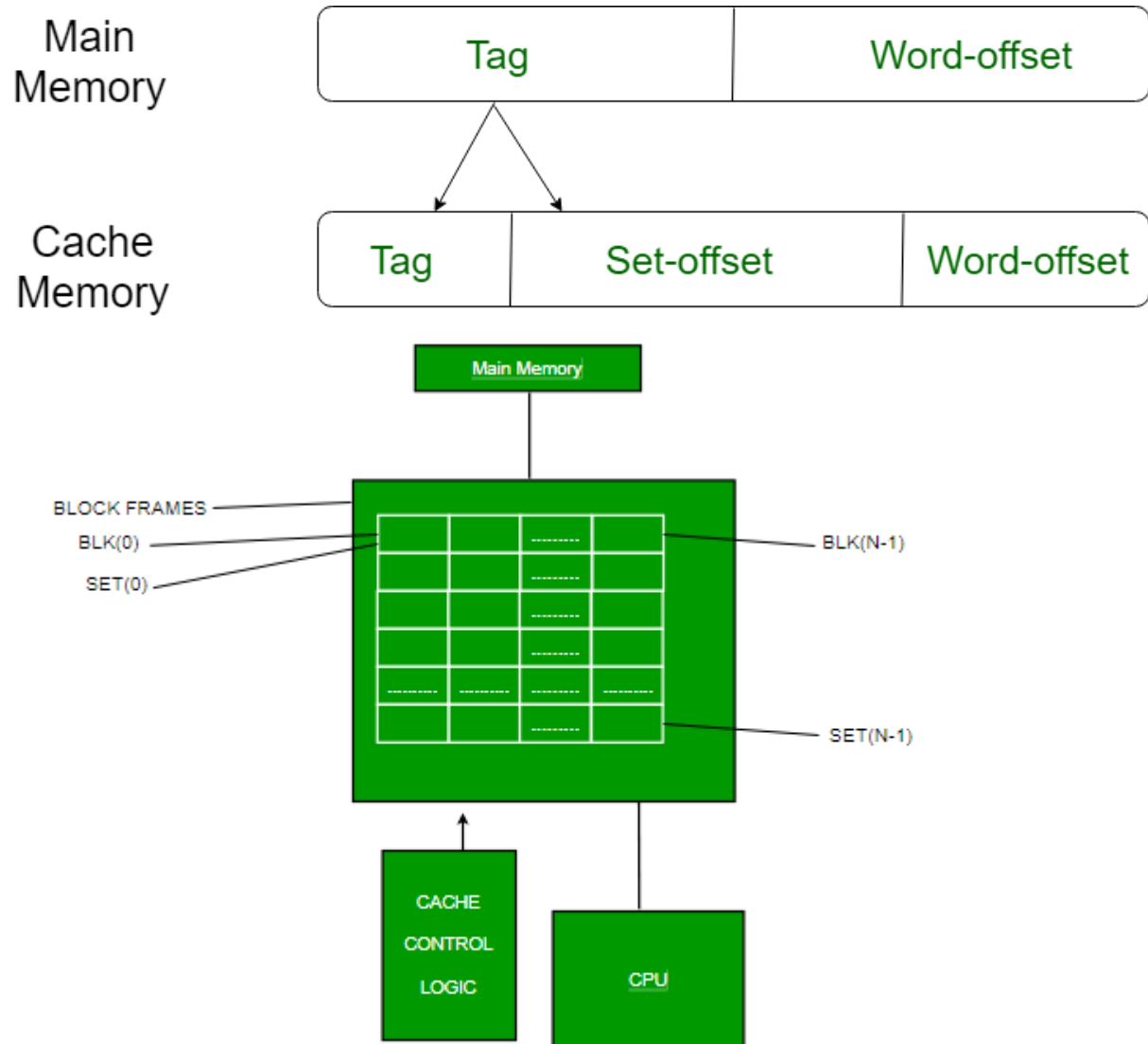
i =cache set number

j =main memory block number

v =number of sets

m =number of lines in the cache number of sets

k =number of lines in each set



Application of Cache Memory:

1. Usually, the cache memory can store a reasonable number of blocks at any given time, but this number is small compared to the total number of blocks in the main memory.
2. The correspondence between the main memory blocks and those in the cache is specified by a mapping function.

3. **Primary Cache** – A primary cache is always located on the processor chip. This cache is small and its access time is comparable to that of processor registers.
4. **Secondary Cache** – Secondary cache is placed between the primary cache and the rest of the memory. It is referred to as the level 2 (L2) cache. Often, the Level 2 cache is also housed on the processor chip.
5. **Spatial Locality of reference** This says that there is a chance that the element will be present in close proximity to the reference point and next time if again searched then more close proximity to the point of reference.
6. **Temporal Locality of reference** In this Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load word in main memory but complete page fault will be loaded because the spatial locality of reference rule says that if you are referring to any word next word will be referred to in its register that's why we load complete page table so the complete block will be loaded