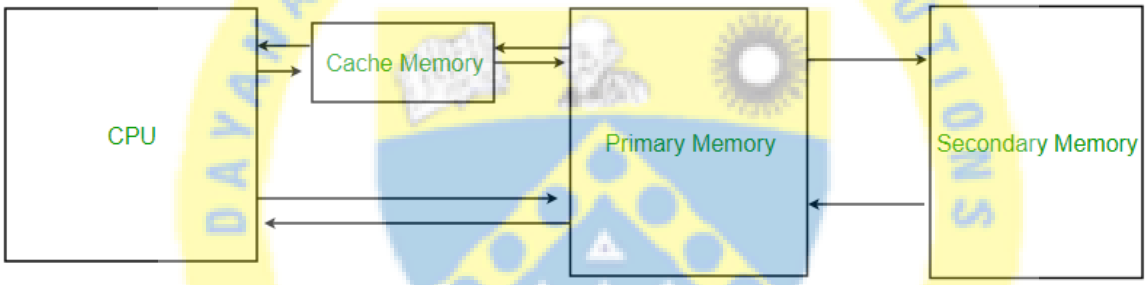
Cache Memory in Computer Organization

Cache Memory is a special very high-speed memory. It is used to speed up and synchronizing with high-speed CPU. Cache memory is costlier than main memory or disk memory but 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 which 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 memory organization

A cache memory is a fast random access memory where the computer hardware stores copies of information currently used by programs (data and instructions), loaded from the main memory. The cache has a significantly shorter access time than the main memory due to the applied faster but more expensive implementation technology. The cache has a limited volume that also results from the properties of the applied technology. If information fetched to the

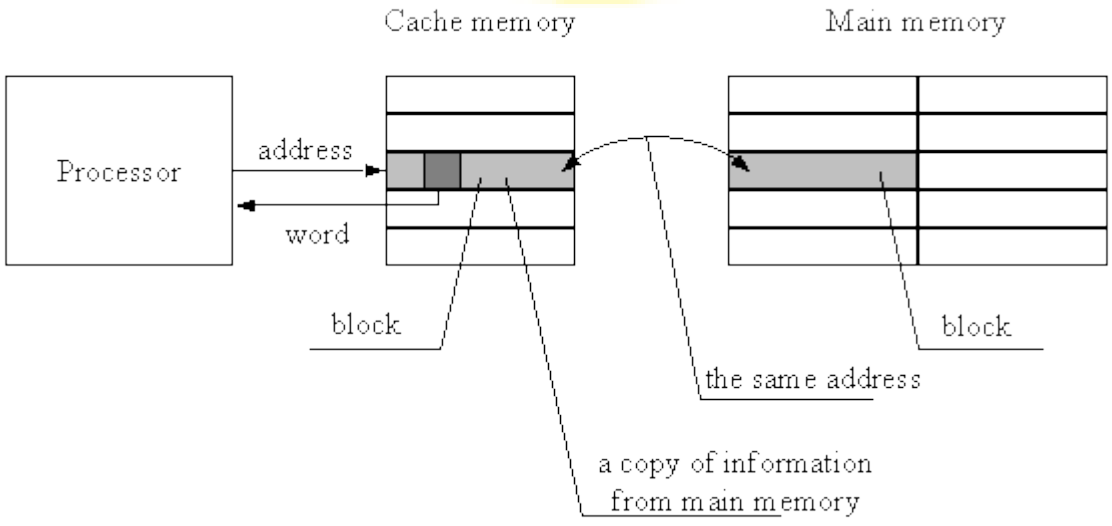
cache memory is used again, the access time to it will be much shorter than in the case if this information were stored in the main memory and the program will execute faster.

Time efficiency of using cache memories results from the locality of access to data that is observed during program execution. We observe here time and space locality:

- **Time locality** consists in a tendency to use many times the same instructions and data in programs during neighbouring time intervals,
- **Space locality** is a tendency to store instructions and data used in a program in short distances of time under neighbouring addresses in the main memory.

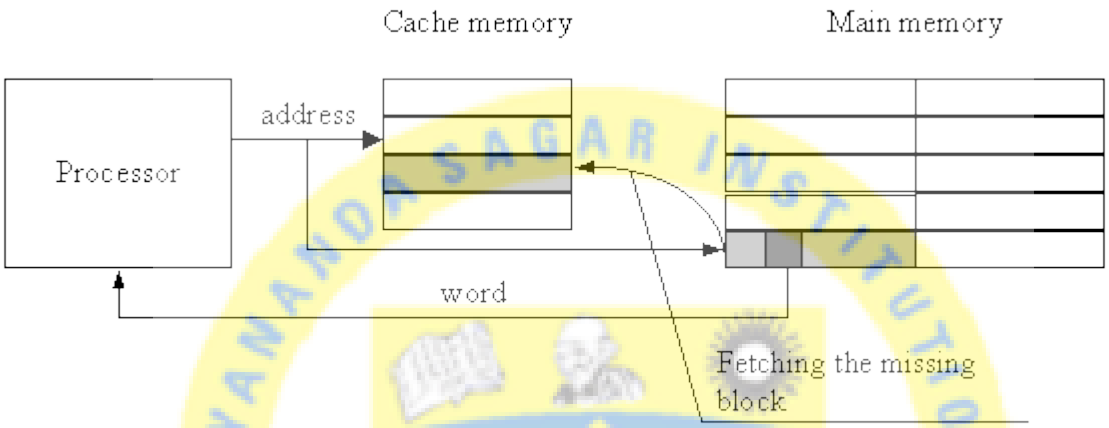
Due to these localities, the information loaded to the cache memory is used several times and the execution time of programs is much reduced. Cache can be implemented as a multi-level memory. Contemporary computers usually have two levels of caches. In older computer models, a cache memory was installed outside a processor (in separate integrated circuits than the processor itself). The access to it was organized over the processor external system bus. In today's computers, the first level of the cache memory is installed in the same integrated circuit as the processor. It significantly speeds up processor's co-operation with the cache. Some microprocessors have the second level of cache memory placed also in the processor's integrated circuit. The volume of the first level cache memory is from several thousands to several tens of thousands of bytes. The second level cache memory has volume of several hundred thousand bytes. A cache memory is maintained by a special processor subsystem called **cache controller**.

If there is a cache memory in a computer system, then at each access to a main memory address in order to fetch data or instructions, processor hardware sends the address first to the cache memory. The cache control unit checks if the requested information resides in the cache. If so, we have a "hit" and the requested information is fetched from the cache. The actions concerned with a read with a hit are shown in the figure below.



Read implementation in cache memory on hit

If the requested information does not reside in the cache, we have a "miss" and the necessary information is fetched from the main memory to the cache and to the requesting processor unit. The information is not copied in the cache as single words but as a larger block of a fixed volume. Together with information block, a part of the address of the beginning of the block is always copied into the cache. This part of the address is next used at readout during identification of the proper information block. The actions executed in a cache memory on "miss" are shown below.



Read implementation in cache memory on miss

To simplify the explanations, we have assumed a single level of cache memory below. If there are two cache levels, then on "miss" at the first level, the address is transferred in a hardwired way to the cache at the second level. If at this level a "hit" happens, the block that contains the requested word is fetched from the second level cache to the first level cache. If a "miss" occurs also at the second cache level, the blocks containing the requested word are fetched to the cache memories at both levels. The size of the cache block at the first level is from 8 to several tens of bytes (a number must be a power of 2). The size of the block in the second level cache is many times larger than the size of the block at the first level.

The cache memory can be connected in different ways to the processor and the main memory:

- as an additional subsystem connected to the system bus that connects the processor with the main memory,
- as a subsystem that intermediates between the processor and the main memory,
- as a separate subsystem connected with the processor, in parallel regarding the main memory.

The third solution is applied the most frequently.

We will now discuss different kinds of information organization in cache memories.

There are three basic methods used for mapping of information fetched from the main memory to the cache memory:

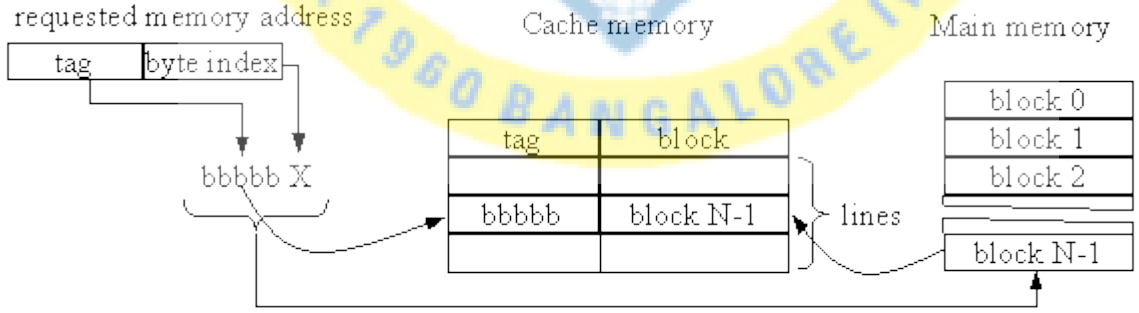
- **associative mapping**
- **direct mapping**
- **set-associative mapping.**

In today's computers, caches and main memories are byte-addressed, so we will refer to byte-addressed organization in the sections on cache memories that follow.

Cache memory with associative mapping

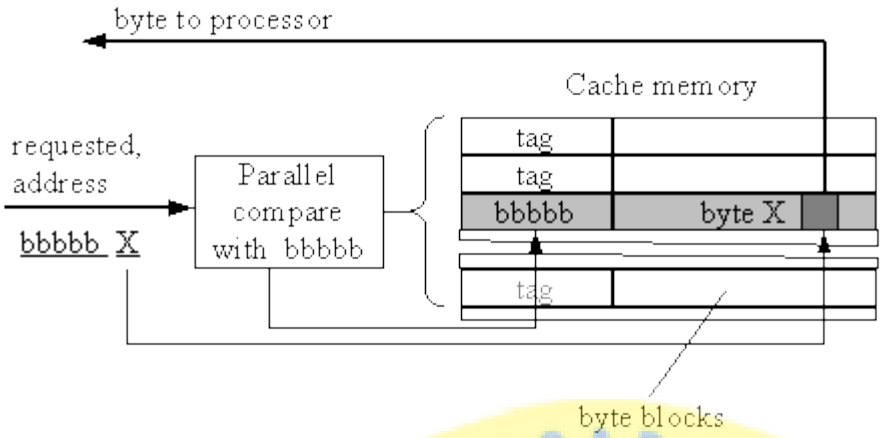
With the associative mapping of the contents of cache memory, the address of a word in the main memory is divided into two parts: the tag and the byte index (offset). Information is fetched into the cache in blocks. The byte index determines the location of the byte in the block whose address is generated from the tag bits, which are extended by zeros in the index part (it corresponds to the address of the first byte in the block. In the number of bits in the byte index is *n* then the size of the block is a power of 2 with the exponent *n*. The cache is divided into lines. In each line one block can be written together with its tag and usually some control bits. It is shown in the figure below.

When a block is fetched into the cache (on miss), the block is written in an arbitrary free line. If there is no free line, one block of information is removed from the cache to liberate one line. The block to be removed is determined according to a selected strategy, for example the least used block can be selected. To support the block selection, each access to a block residing in the cache, is registered by changing the control bits in the line the block occupies.



Information organization in cache with associative mapping

The principle of the read operation in cache memory is shown below. The requested address contains the tag (bbbbb) and the byte index in the block (X). The tag is compared in parallel with all tags written down in all lines. If a tag match is found in a line, we have a hit and the line contains the requested information block. Then, based on the byte index, the requested byte is selected in the block and read out into the processor. If none of the lines contains the requested tag, the requested block does not reside in the cache. The missing block is next fetched from the main memory or an upper level cache memory.

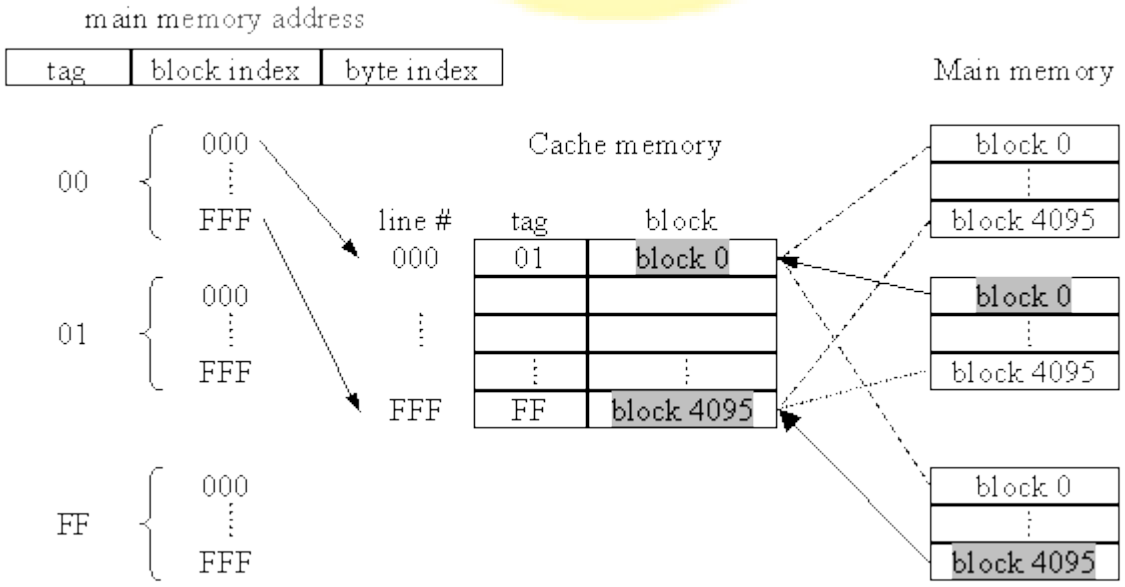


Read of a byte on hit in a cache with associative mapping

The functioning of a cache with associative mapping is based on the associative access to memory. The requested data are found by a parallel comparison of the requested tag with tags registered in cache lines. For a big number of lines, the comparator unit is very large and costly. Therefore, the associative mapping is applied in cache memories of a limited sizes (i.e. containing not too many lines).

Cache memory with direct mapping

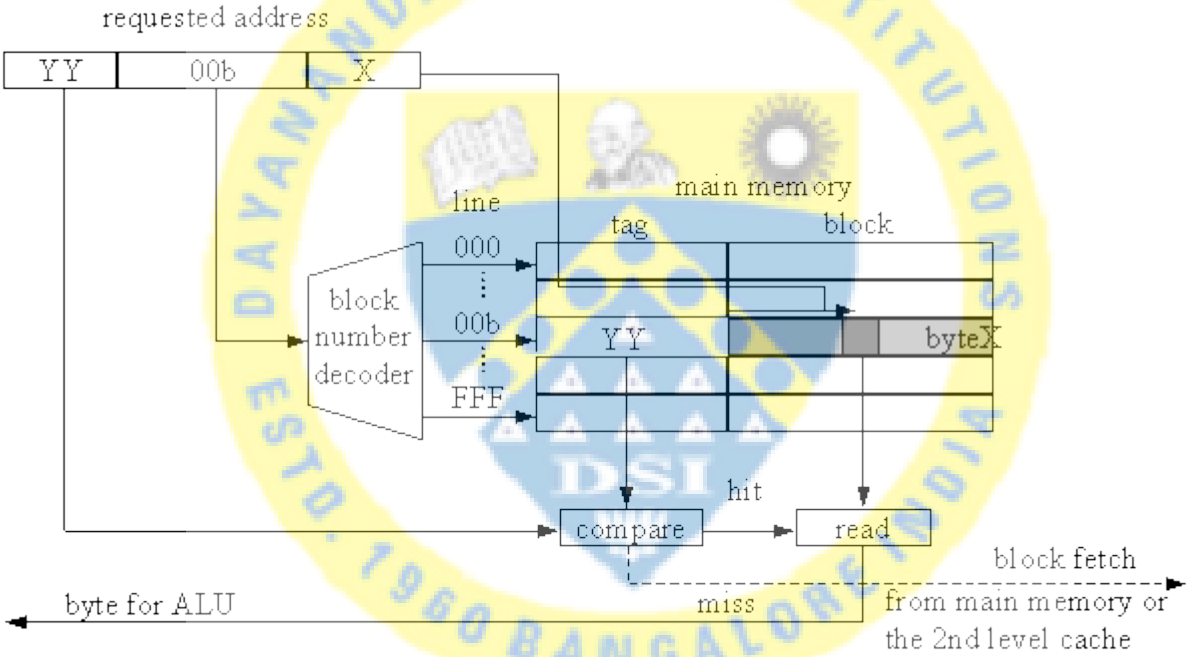
The name of this mapping comes from the direct mapping of data blocks into cache lines. With the direct mapping, the main memory address is divided into three parts: a tag, a block index and a byte index. In a given cache line, only such blocks can be written, whose block indices are equal to the line number. Together with a block, the tag of its address is stored. It is easy to see that each block number matches only one line in the cache.



Information organization in cache with direct mapping

The readout principle in cache with direct mapping is shown below. The block index (middle part of the address) is decoded in a decoder, which selects lines in the cache. In a selected line, the tag is compared with the requested one. If a match is found, the block residing in the line is exactly that which has been requested, since in the main memory there are no two blocks with the same block indices and tags.

We have a hit in this case and the requested byte is read in the block. If there was no tag match, it means that either there is no block yet in the line or the residing block is different to the requested one. In both cases, the requested block is fetched from the main memory or the upper level cache. Together with the fetched block, its tag is stored in the cache line.



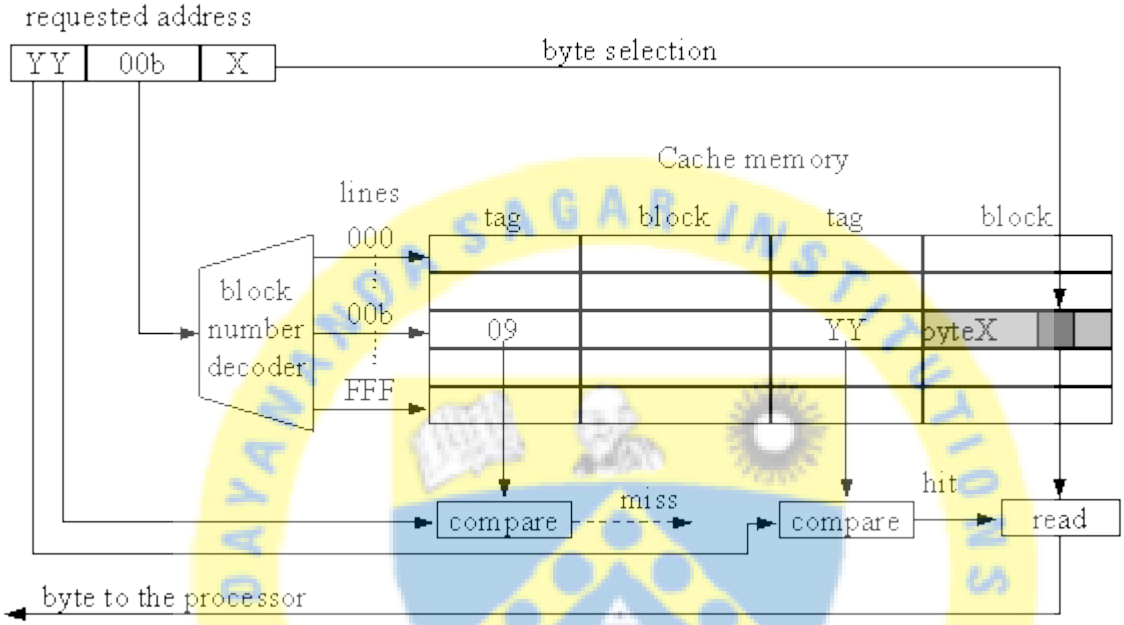
Read of a byte in a cache with direct mapping

With direct mapping, all blocks with the same index have to be written into the same cache line. It can cause frequent block swapping in cache lines, since only one block can reside in a line at a time. It is called **block thrashing** in the cache. For large data structures used in programs, this phenomenon can substantially decrease the efficiency of cache space use. The solution shown in next section eliminates this drawback.

Cache memory with set associative mapping

With this mapping, the main memory address is structured as in the previous case. We have there a tag, a block index and a byte index. The block into line mapping is the same as for the

direct mapping. But in a set associative mapping many blocks with different tags can be written down into the same line (a set of blocks). Access to blocks written down in a line is done using the associative access principle, i.e. by comparing the requested tag with all tags stored in the selected line. From both mentioned features, the name of this mapping is derived. The figure below shows operations during a read from a cache of this type.



Read of a byte in a cache with set associative mapping

First, the block index of the requested address is used to select a line in a cache. Next, comparator circuits compare the requested tag with all stored in the line. On match, the requested byte is fetched from the selected block and sent to the processor. On miss (no match), the requested block is fetched from the main memory or the upper level cache. The new block is stored in a free block slot in the line or in the slot liberated by a block sent back to the main memory (or the upper level cache). To select a block to be removed, different strategies can be applied. The most popular is the LRU (least-recently used) strategy, where the block not used for the longest time is removed. Other strategies are: FIFO (first-in-first-out) strategy - the block that is stored during the longest time is selected or LFU (least-frequently used) strategy where the least frequently modified block is selected. To implement these strategies, some status fields are maintained associated with the tags of blocks.

Due to the set associative mapping, block thrashing in cache is eliminated to the large degree. The number of blocks written down in the same cache line is from 2 to 6 with the block size of 8 to 64 bytes.

Memory updating methods after cache modification

A cache memory contains copies of data stored in the main memory. When a change of data in a cache takes place (ex. a modification due to a processor write) the contents of the main memory and cache memory cells with the same address, are different. To eliminate this lack of data coherency two methods are applied:

- **write through**, the new cache contents is written down to the main memory immediately after the write to the cache memory,
- **write back**, the new cache contents is not written down to the main memory immediately after the change, but only when the given block of data is replaced by a new block fetched from the main memory or an upper level cache. After a data write to the cache, only state bits are changed in the modified block, indicating that the block has been modified (a dirty block).

The write back updating is more time efficient, since the block cells that were modified many times while being in the cache, are updated in the main memory only once.

Sector Mapping

In sector mapping, the main memory and the cache are both divided into sectors; each sector is composed of a number of blocks. Any sector in the main memory can map into any sector in the cache and a tag is stored with each sector in the cache to identify the main memory sector address. However, a complete sector is not transferred to the cache or back to the main memory as one unit. Instead, individual blocks are transferred as required. On cache sector miss, the required block of the sector is transferred into a specific location within one sector. The sector location in the cache is selected and all the other existing blocks in the sector in the cache are from a previous sector. Sector mapping might be regarded as a fully associative mapping scheme with valid bits, as in some microprocessor caches. Each block in the fully associative mapped cache corresponds to a sector, and each byte corresponds to a "sector block".

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 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.**

Hit ratio = hit / (hit + miss) = no. of hits/total accesses

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