Cache Memory in Computer Organization

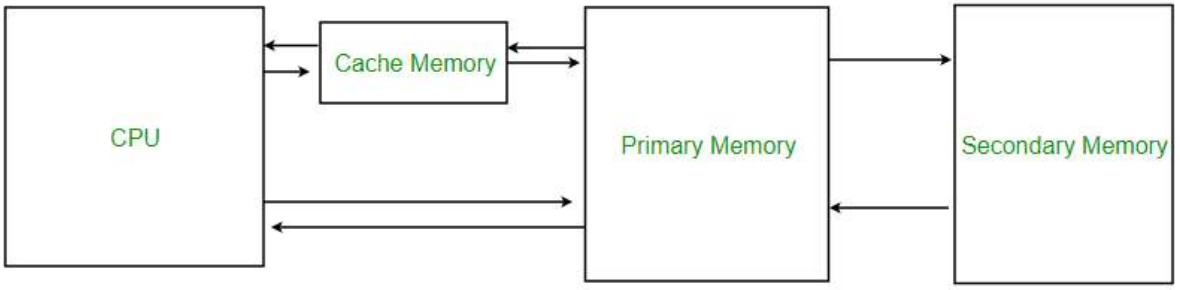
Cache memory is a small, high-speed storage area in a computer. The cache is a smaller and faster memory that stores copies of the data from frequently used main memory locations. There are various independent caches in a CPU, which store instructions and data.

- The most important use of cache memory is that it is used to reduce the average time to access data from the main memory.
- The concept of cache works because there exists locality of reference (the same items or nearby items are more likely to be accessed next) in processes.

By storing this information closer to the CPU, cache memory helps speed up the overall processing time. Cache memory is much faster than the main memory (RAM). When the CPU needs data, it first checks the cache. If the data is there, the CPU can access it quickly. If not, it must fetch the data from the slower main memory.

Characteristics of Cache Memory

- Extremely fast memory type that acts as a buffer between RAM and the CPU.
- Holds frequently requested data and instructions, ensuring that they are immediately available to the CPU when needed.
- Costlier than main memory or disk memory but more economical than CPU registers.
- Used to speed up processing and synchronize with the high-speed CPU.



Cache Memory

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 the CPU. The most commonly used register is Accumulator, Program counter , Address Register, etc.
- **Level 2 or Cache memory:** It is the fastest memory that has faster access time where data is temporarily stored for faster access.
- **Level 3 or Main Memory:** It is the memory on which the 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 that is not as fast as the main memory but data stays permanently in this memory.

Cache Performance

When the processor needs to read or write a location in the 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 the 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(H) = hit / (hit + miss) =  no. of hits/total accesses
Miss Ratio = miss / (hit + miss) =  no. of miss/total accesses = 1 - hit ratio(H)
```

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

Cache mapping refers to the method used to store data from main memory into the cache. It determines how data from memory is mapped to specific locations in the cache.

There are three different types of mapping used for the purpose of cache memory which is as follows:

- Direct Mapping
- Fully Associative Mapping
- Set-Associative Mapping

1. Direct Mapping

Direct mapping is a simple and commonly used cache mapping technique where each block of main memory is mapped to exactly one location in the cache called cache line. If two memory blocks map to the same cache line, one will overwrite the other, leading to potential cache misses. Direct mapping's performance is directly proportional to the Hit ratio.

Memory block is assigned to cache line using the formula below:

```
i = j modulo m = j % m
where,
i = cache line number
j = main memory block number
m = number of lines in the cache
```

For example, consider a memory with 8 blocks(j) and a cache with 4 lines(m). Using direct mapping, block 0 of memory might be stored in cache line 0, block 1 in line 1, block 2 in line 2, and block 3 in line 3. If block 4 of memory is accessed, it would be mapped to cache line 0 (as i = j modulo m i.e. i = 4 % 4 = 0), replacing memory block 0.

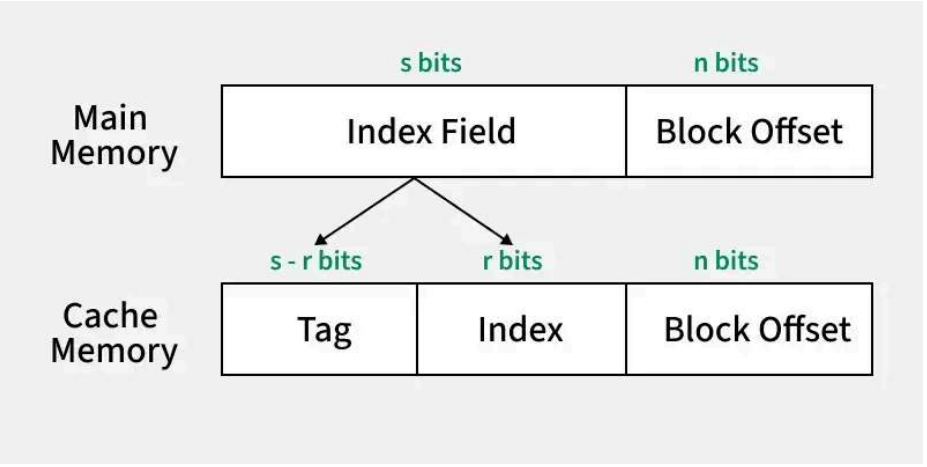
The **Main Memory** consists of memory blocks and these blocks are made up of fixed number of words. A typical address in main memory is split into two parts:

1. **Index Field:** It represent the block number. Index Field bits tells us the location of block where a word can be.

2. **Block Offset:** It represent words in a memory block. These bits determines the location of word in a memory block.

The **Cache Memory** consists of cache lines. These cache lines has same size as memory blocks. The address in cache memory consists of:

1. **Block Offset:** This is the same block offset we use in Main Memory.
2. **Index:** It represent cache line number. This part of the memory address determines which cache line (or slot) the data will be placed in.
3. **Tag:** The Tag is the remaining part of the address that uniquely identifies which block is currently occupying the cache line.



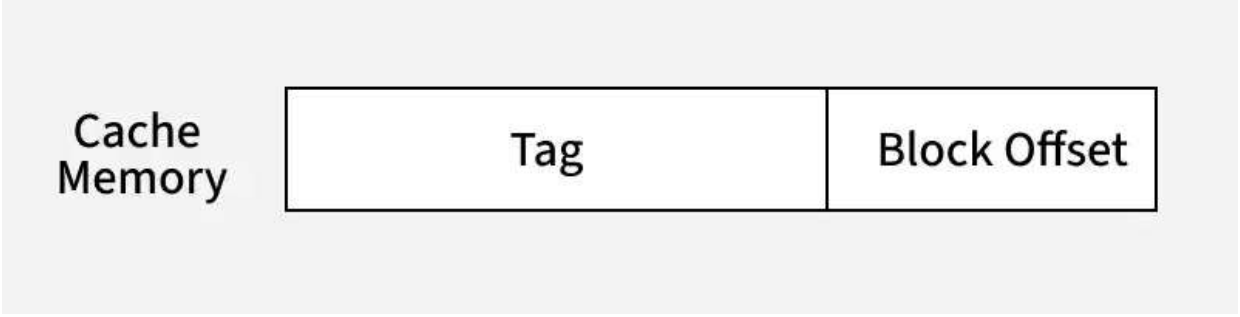
Memory Structure in Direct Mapping

The index field in main memory maps directly to the index in cache memory, which determines the cache line where the block will be stored. The block offset in both main memory and cache memory indicates the exact word within the block. In the cache, the tag identifies which memory block is currently stored in the cache line. This mapping ensures that each memory block is mapped to exactly one cache line, and the data is accessed using the tag and index while the block offset specifies the exact word in the block.

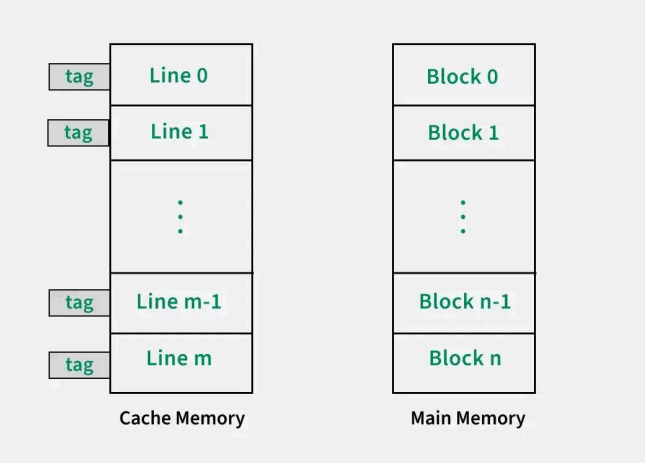
2. Fully Associative Mapping

Fully associative mapping is a type of cache mapping where any block of main memory can be stored in any cache line. Unlike direct-mapped cache, where each memory block is restricted to a specific cache line based on its index, fully associative mapping gives the cache the flexibility to place a memory block in any available cache line. This improves the hit ratio but requires a more complex system for searching and managing cache lines.

The address structure of Cache Memory is different in fully associative mapping from direct mapping. In fully associative mapping, the cache does not have an index field. It only have a tag which is same as Index Field in memory address. Any block of memory can be placed in any cache line. This flexibility means that there's no fixed position for memory blocks in the cache.



To determine whether a block is present in the cache, the tag is compared with the tags stored in all cache lines. If a match is found, it is a cache hit, and the data is retrieved from that cache line. If no match is found, it's a cache miss, and the required data is fetched from main memory.



Fully Associative Mapping

3. Set-Associative Mapping

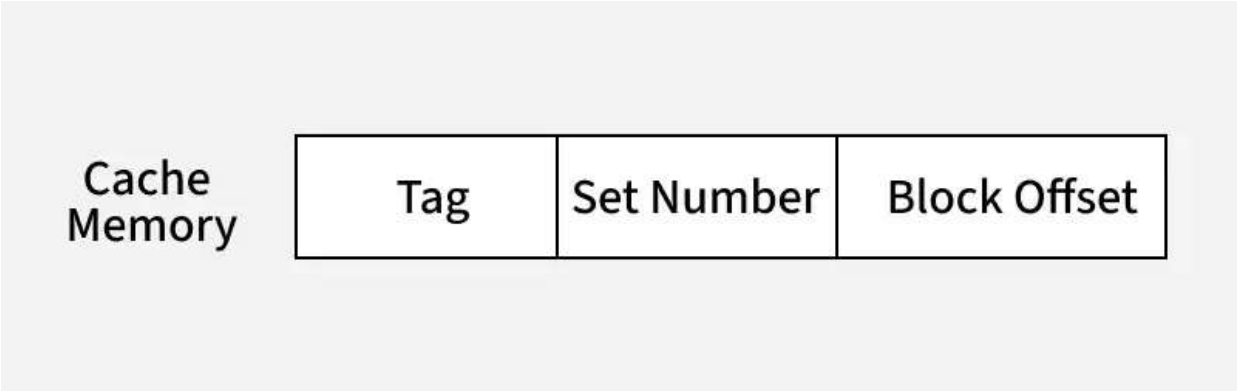
Set-associative mapping is a compromise between direct-mapped and fully-associative mapping in cache systems. It combines the flexibility of fully associative mapping with the efficiency of direct mapping. In this scheme, multiple cache lines (typically 2, 4, or more) are grouped into sets.

```
v = m / k
where,
m = number of cache lines in the cache memory
k = number of cache lines we want in each set
v = number of sets
```

Like direct mapping, now each memory block can be placed into any cache line within a specific set.

```
i = j modulo v = j % v
where,
j = main memory block number
v = number of sets
i = cache line set number
```

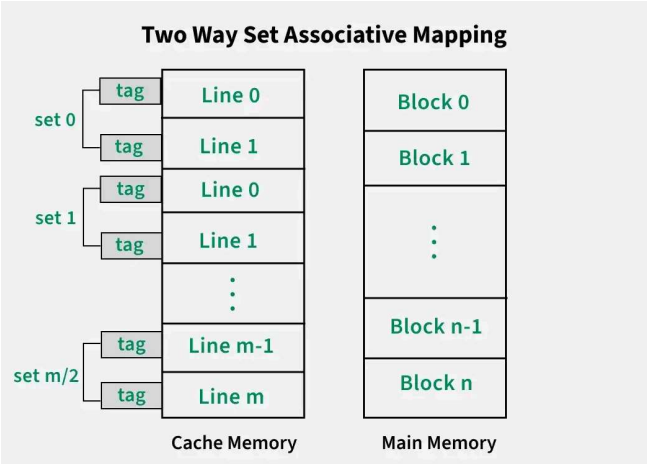
The Cache address structure is as follows:



Cache Memory in Set Associative Mapping

This reduces the conflict misses that occur in direct mapping while still limiting the search space compared to fully-associative mapping.

For example, consider a 2-way set associative cache, which means 2 cache lines make a set in this cache structure. There are 8 memory blocks and 4 cache lines, thus the number of sets will be 4/2 = 2 sets. Using direct mapping strategy first, block 0 will be in set 0, block 1 in set 1, block 2 in set 0 and so on. Then, the tag is used to search through all cache lines in that set to find the correct block (Associative Mapping).



Two Way Set Associative Cache

For more, you can refer to the Difference between Types of Cache Mapping .

Application of Cache Memory

Here are some of the applications of Cache Memory.

- **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.
- **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.
- **Spatial Locality of Reference:** Spatial Locality of Reference 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.
- **Temporal Locality of Reference:** Temporal Locality of Reference uses the Least recently used algorithm will be used. Whenever there is page fault occurs within a word will not only load the word

in the main memory but the 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.

Advantages

- Cache Memory is faster in comparison to main memory and secondary memory.
- Programs stored by Cache Memory can be executed in less time.
- The data access time of Cache Memory is less than that of the main memory.
- Cache Memory stored data and instructions that are regularly used by the CPU, therefore it increases the performance of the CPU.

Disadvantages

- Cache Memory is costlier than primary memory and secondary memory .
- Data is stored on a temporary basis in Cache Memory.
- Whenever the system is turned off, data and instructions stored in cache memory get destroyed.
- The high cost of cache memory increases the price of the Computer System.

Related Questions

Practicing the following questions will help you test your knowledge. All questions have been asked in GATE in previous years or GATE Mock Tests. It is highly recommended that you practice them.

- GATE CS 2017 (Set 1), Question 56
- GATE CS 2022, Question 54
- GATE CS 2005, Question 67
- GATE CS 2021, Question 29

- GATE CS 2023, Question 65
- GATE CS 2017 (Set 2), Question 54

Conclusion

In conclusion, cache memory plays a crucial role in making computers faster and more efficient. By storing frequently accessed data close to the CPU , it reduces the time the CPU spends waiting for information. This means that tasks are completed quicker, and the overall performance of the computer is improved. Understanding cache memory helps us appreciate how computers manage to process information so swiftly, making our everyday digital experiences smoother and more responsive.

Frequently Asked Questions on Cache Memory – FAQs

Why is cache memory important?

Cache memory is important because it helps speed up the processing time of the CPU by providing quick access to frequently used data, improving the overall performance of the computer.

How does cache memory work?

Cache memory works by storing copies of frequently accessed data from the main memory. When the CPU needs data, it first checks the cache. If the data is found (a "cache hit"), it is quickly accessed. If not (a "cache miss"), the data is fetched from the slower main memory.

Can cache memory be upgraded?

Cache memory is built into the CPU or located very close to it, so it cannot be upgraded separately like RAM. To get more or faster cache, you would need to upgrade the entire CPU.

Is cache memory volatile?

Yes, cache memory is volatile, meaning it loses all stored data when the computer is turned off, just like RAM.