The Fundamental Knowledge of System Design — (5) — Cache

The cache is an intermediate layer introduced to bridge the gap between high-speed devices and low-speed devices, to increase application processing speed. Storing a copy of the previously fetched data or computed results increases processing speed.

It is the fifth series of the fundamentals knowledge of system design. You can read my previous articles.

The fundamental knowledge of System Design — (1)

Today, I will share the fundamental knowledge of system design.

medium.com

The fundamental knowledge of System Design — (2)

Please clap and share this article if you like it.

medium.com

The fundamental knowledge of System Design — (3)

Servers are the core of today's computational world. Server performance depends on throughput and latency. Generally…

medium.com

The fundamental knowledge of System Design — (4)

System Availability = Availability = Uptime ÷ (Uptime + downtime)

medium.com

Why there is a need for a cache in this computer? It is because:

1. The capacity, access speed, and the cost of various storage devices in computer systems vary greatly
2. A deep understanding of cache helps us write better performance code. A cache is a hardware concept, which is located between the processor and the memory and is mainly used to solve the problem of the mismatch between the processor and the memory speed. So, programs written must have good locality especially temporal locality and spatial locality.

Let us take a bottom-up approach to understand the various layers of cache.

Computer Architecture

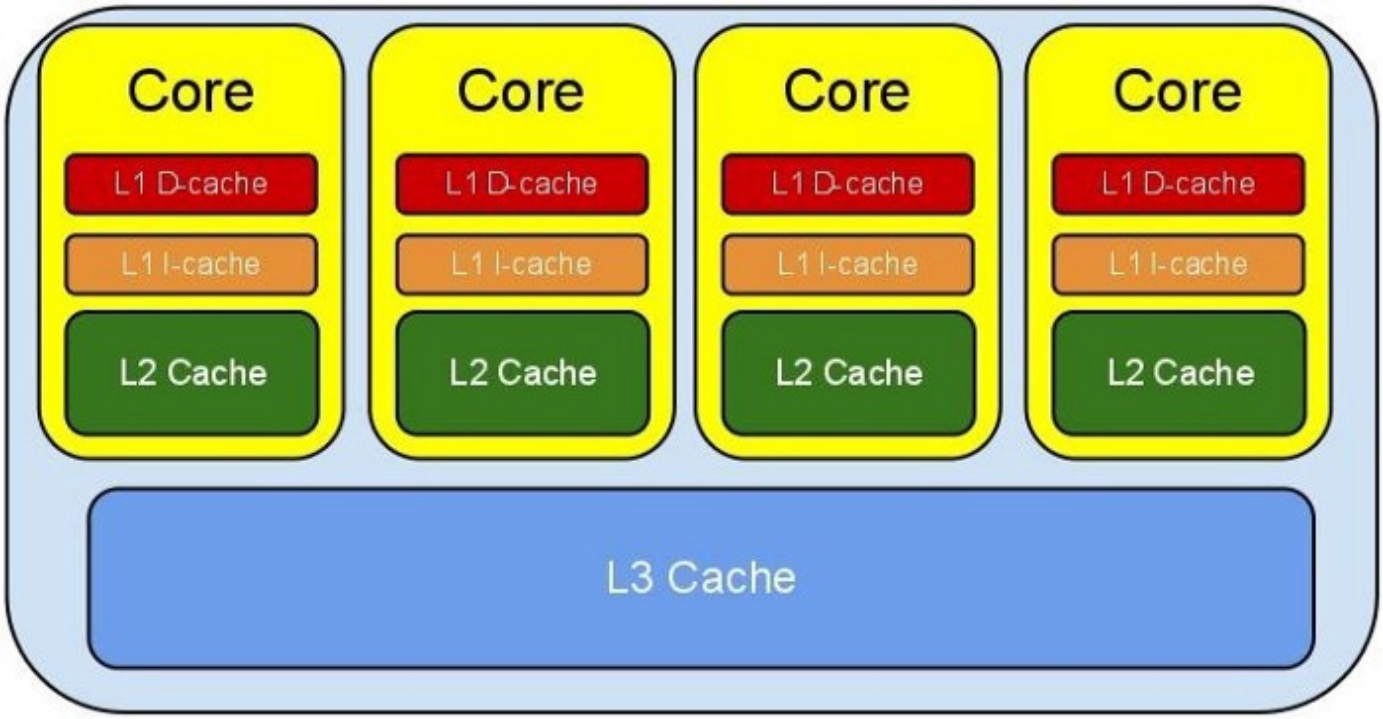
A computer is a machine that can be programmed to input, process, and output useful information, or either store it…

medium.com

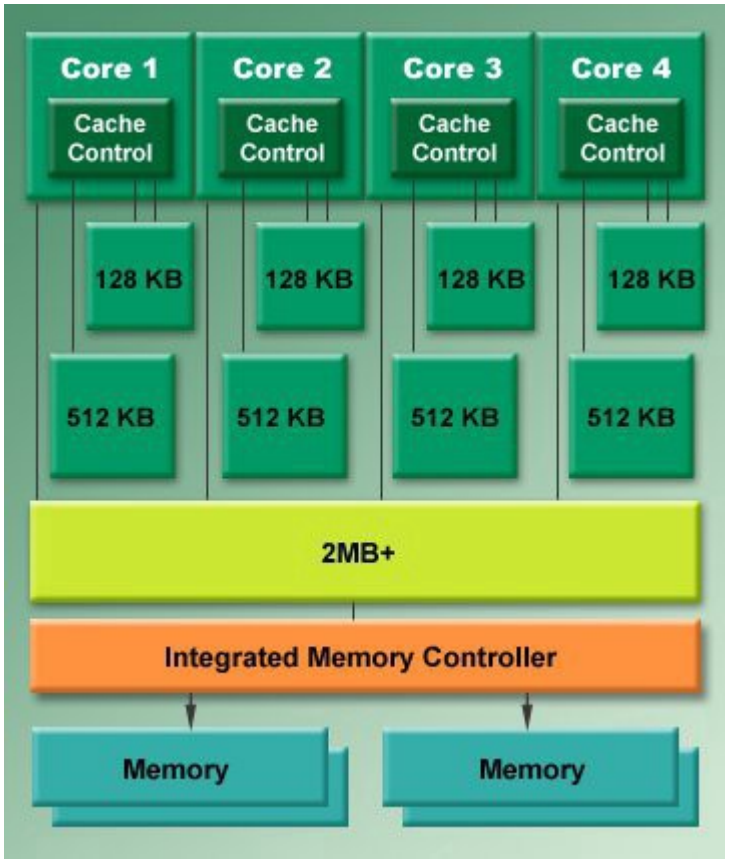
You can read my previous article to have a better understanding of computer structure.

Cache

We all know that the current CPU multi-core technology will have several levels of cache, and the current CPU will have 3 levels of memory (L1, L2, L3).



https://www.quora.com/How-many-caches-are-there-in-a-CPU



https://www.quora.com/How-many-caches-are-there-in-a-CPU

Level 1 Cache (L1 cache)

It is responsible for caching instructions and caching data. The capacity and structure of the level 1 cache have a great impact on the performance of the CPU, but due to its complex structure. It is about 256 KB.

Level 2 Cache (L2 cache)

The capacity of the L2 cache will affect the performance of the CPU too. The larger capacity of the L2 cache can provide better system performance for the CPU. For example, Intel's eighth-generation i7–8700 processor has 6 cores, and each core has a 256 KB L2 cache, which is exclusive to each core so that the total number of L2 caches reaches 1.5MB.

Level 3 cache (L3 cache)

It is responsible for further reducing the delay of the memory and improving the performance of massive data calculations. Level 3 cache is shared by cores and can make a large capacity.

Notes:

- The L1 cache is divided into 2 types, namely the instruction cache and the data cache. L2 cache and L3 cache do not distinguish between instructions and data.
- L1 and L2 are cached in each CPU core, and L3 is shared by all CPU cores.
- The closer L1, L2, and L3 are to the CPU, the faster the speed of the CPU is, or vice versa.

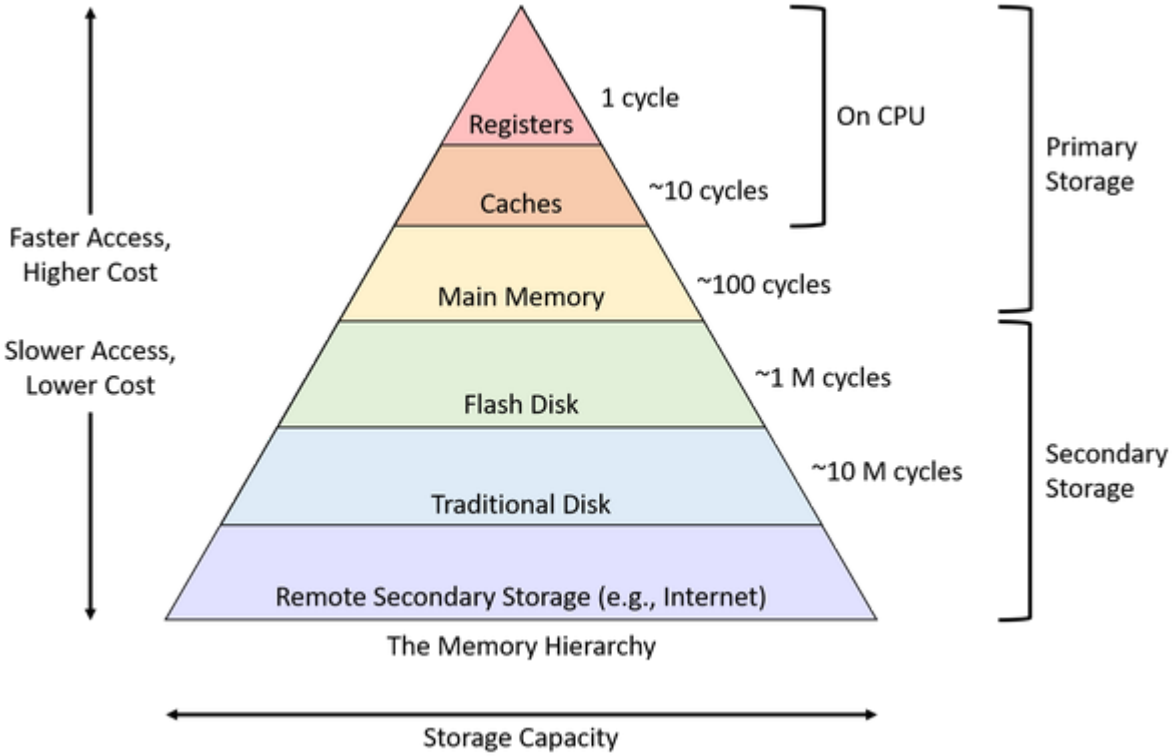
Access Speed

1. **L1 Cache Access Speed**: 4 CPU clock cycles
2. **L2 Cache Access Speed**: 11 CPU clock cycles
3. **L3 Cache Access Speed**: 39 CPU clock cycles
4. **RAM Memory Access Speed**: 107 CPU clock cycles

We can see that the speed of L1 is 27 times that of RAM, the access size of L1 and L2 is basically KB level, and L3 is MB level. For example, Intel Core i7–8700K, is a 6-core CPU,

L1 on each core is 64KB (32KB each for data and instructions), L2 is 256 K, and L3 has 2MB.

The Computer Memory Hierarchy Diagram



https://www.quora.com/What-is-Memory-hierarchy

It is a pyramid-shaped structure for computer memory. The concept of cache has been expanded now, there is not only a cache between the processor and main memory, but also a disk cache between memory and hard disk, or even a certain cache between hard disk and network, which is known as temporary internet folder or network content cache. Any structure that is located between 2 types of hardware with large differences in speed and is used to coordinate the difference in data transmission speed between the 2 devices can be called a cache.

- **Hardware Cache**: CPU cache, GPU cache, DSP
- **Software Cache**: Disk cache, Web cache, etc

Different storage technologies have different price and performance tradeoffs. The slower the access speed is, the cheaper the price of the memory is.

The Principle of Locality References

- Processors access only a small set of instructions and data at any time

1. **Temporal Locality (In time)**

- All those data and instructions which are recently executed have high chances of execution again.

2. Spatial Locality (In space)

- All those data and instructions are stored nearby to the recently executed instruction

Difference between Spatial Locality and Temporal Locality :

S.No.	Spatial Locality	Temporal Locality
1.	In Spatial Locality, nearby instructions to recently executed instruction are likely to be executed soon.	In Temporal Locality, a recently executed instruction is likely to be executed again very soon.
2.	It refers to the tendency of execution which involve a number of memory locations .	It refers to the tendency of execution where memory location that have been used recently have a access.
3.	It is also known as locality in space.	It is also known as locality in time.
4.	It only refers to data item which are closed together in memory.	It repeatedly refers to same data in short time span.
5.	Each time new data comes into execution.	Each time same useful data comes into execution.
6.	Example : Data elements accessed in array (where each time different (or just next) element is being accessing).	Example : Data elements accessed in loops (where same data elements are accessed multiple times).

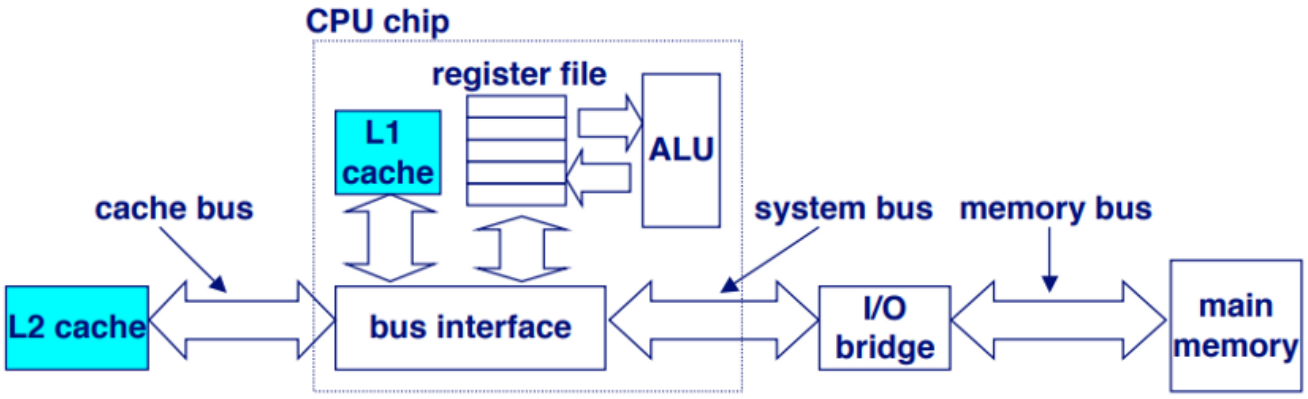
https://www.geeksforgeeks.org/difference-between-spatial-locality-and-temporal-locality/

However, it is impossible for us to cache all the data and instructions in the level 1 cache, but only a part of the data and instructions. At this time, it is necessary to take advantage of

the nature of the locality. The spatial locality will tell us that adjacent data are often accessed together. So, data is transmitted between layers in units of blocks (a continuous piece of data). What if the level 1 cache is already full? The easiest way is to select another block and insert it randomly. But it is not necessarily the best way to do so. The temporal locality tells us that the most recent and least recently accessed data should be cached and should be replaced.

The General Concept of Caching

We assume that the system has only multi-level caches. Generally, it operates by checking the level 1 cache first, if it hit, the processor proceeds at high speed. If the level 1 cache misses, the level 2 cache is checked, and so on, before the main memory is checked. There are 2 types of caches, namely an inclusive cache (the data in the L1 may also be in the L2 cache) and an exclusive cache (the data is guaranteed to be in at most one of the L1 and L2 caches. The advantage of an exclusive cache is that they don't have repeatable data in L1 and L2 caches and can store more data compared to the inclusive caches. The advantage of an inclusive cache is that the data at an address may exist in multiple levels of the cache.

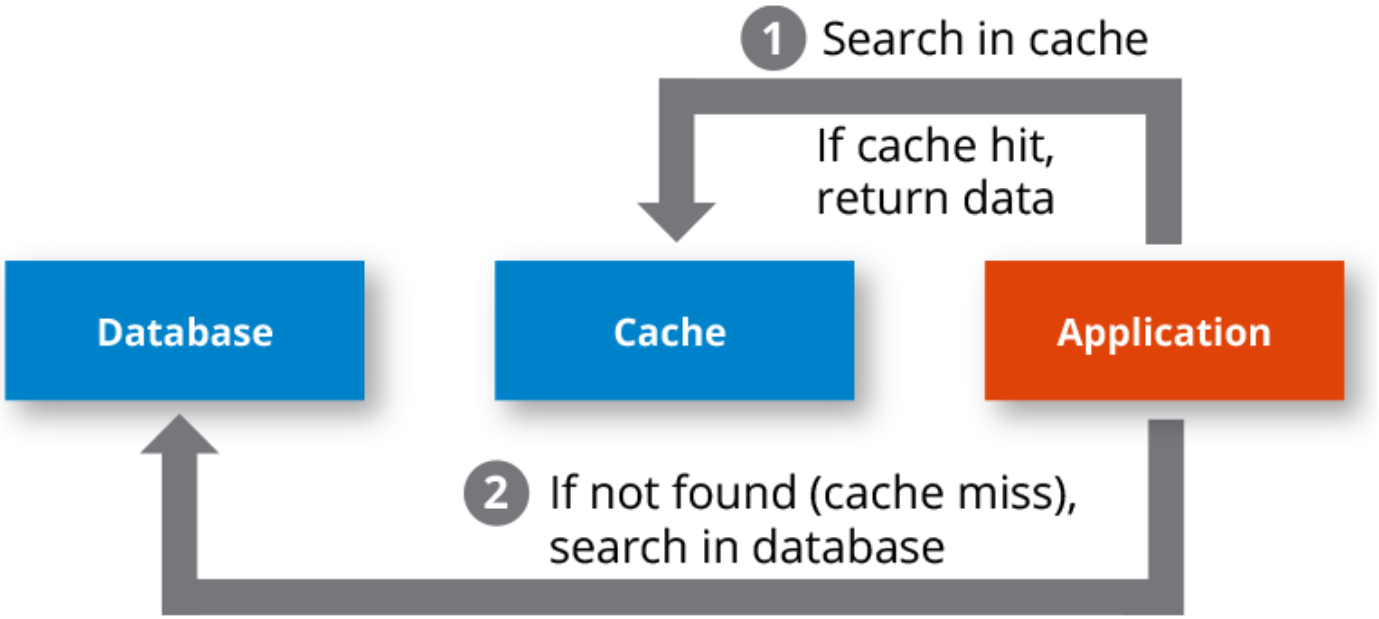


https://www.cs.utexas.edu/users/fussell/courses/cs429h/lectures/Lecture_18-429h.pdf

The processor sends the address to the cache

1. **Cache Hit** — Data for the address in the cache, and returns data according to the block's offset

2. **Cache Miss** — Data is not in the cache, fetch data from memory, send it back to the processor, and retain this data in the cache, do a cache line fill. The processor must deal with variable memory access time



https://hazelcast.com/glossary/cache-miss/

- Also, we need to consider the tradeoff between cache latency and hit rate. Basically, larger caches have better hit rates but longer latency. Hence, many computers use multi-level caches, with small fast caches backed up by larger slower caches.
- Cache misses will add latency.

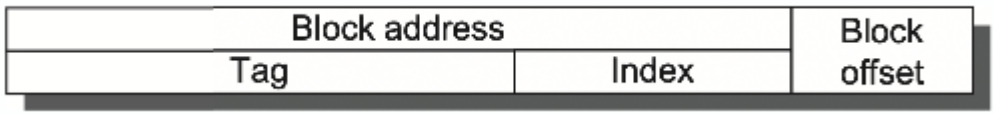
The Way to Reduce Cache Misses

- We can apply appropriate cache replacement policies

1. **FIFO (First In First Out)**
2. **LIFO (Last In First Out)**
3. **LRU (Least Recently Used)**
4. **MRU (Most Recently Used)**

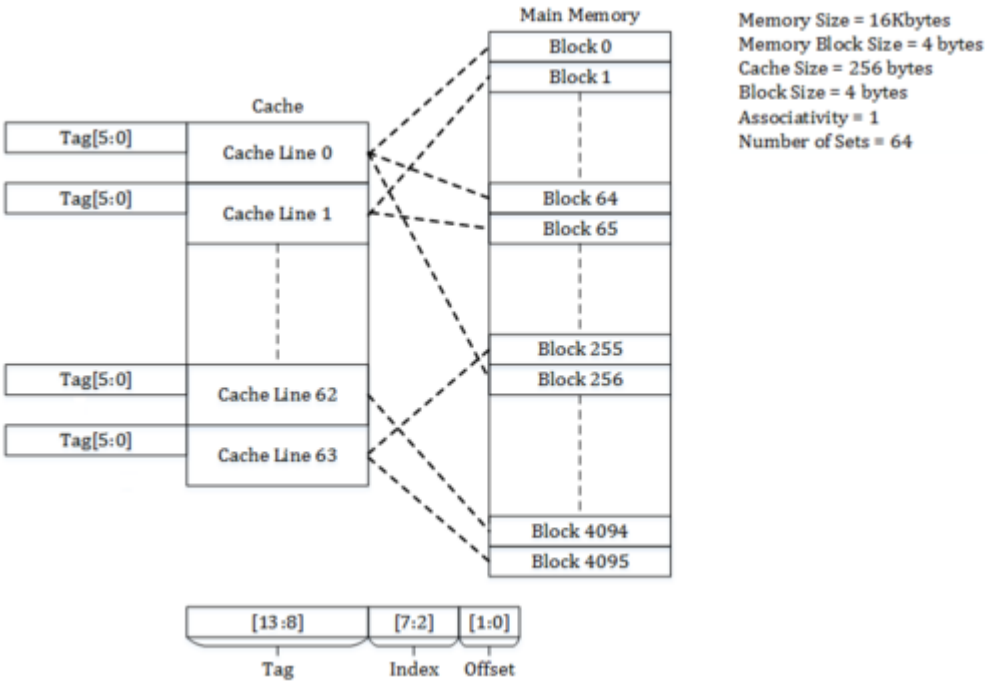
The Composition of Address

To understand how a register works, it is a must to understand what a registry is made of, and how it communicates with the processor. An address consists of 2 blocks: block address and block offset (select the bytes in the data block). The block address is composed of 2 parts: Tag(see if the required data is hit)+ Index (select a group).



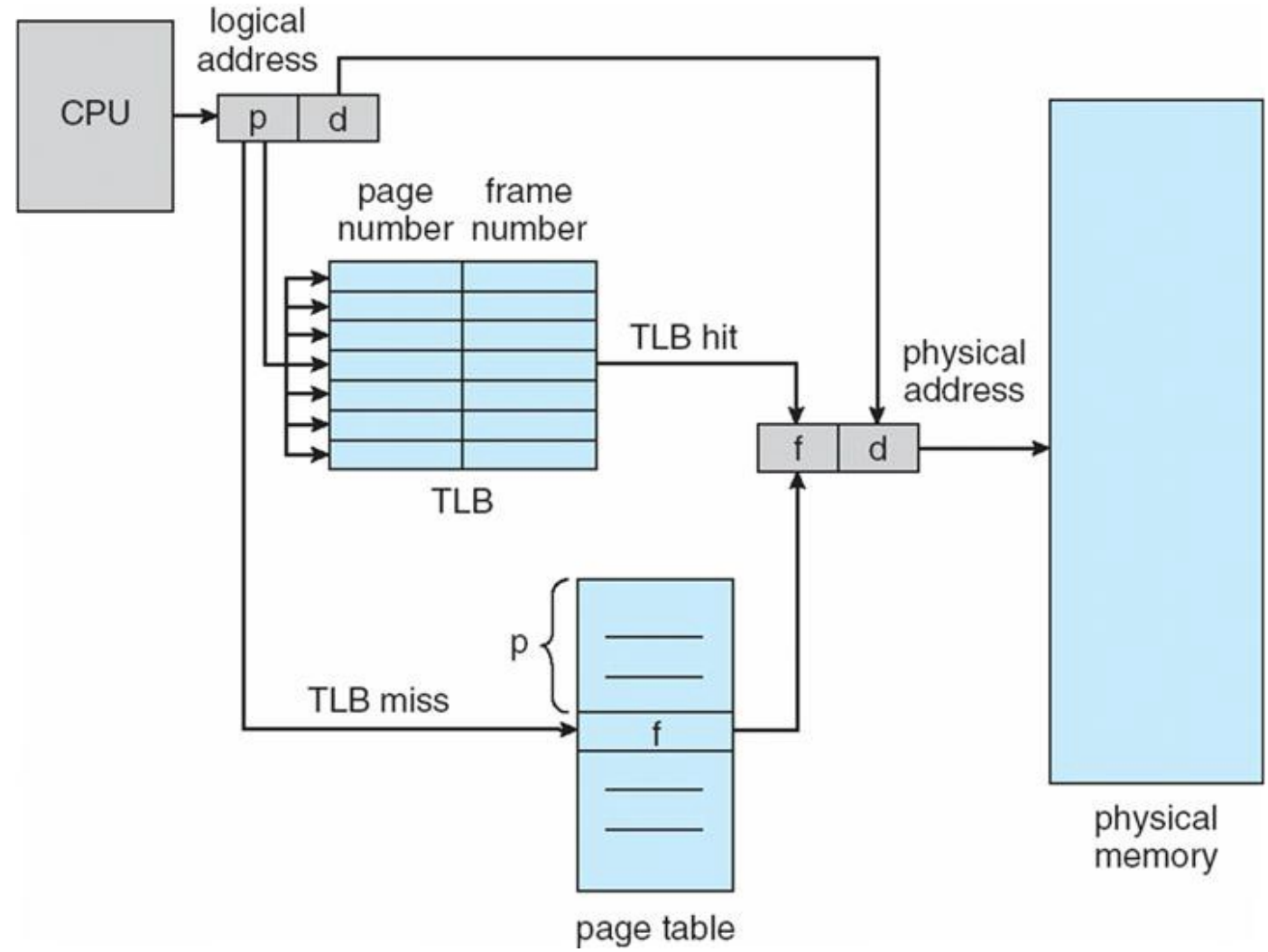
Access Method

The cache is divided into many cache lines. The size of the cache line varies from 16 Byte to 128 Byte. The general size is 64 Byte. So there is a cache of 512KB, which can be divided into 8192 cache lines.



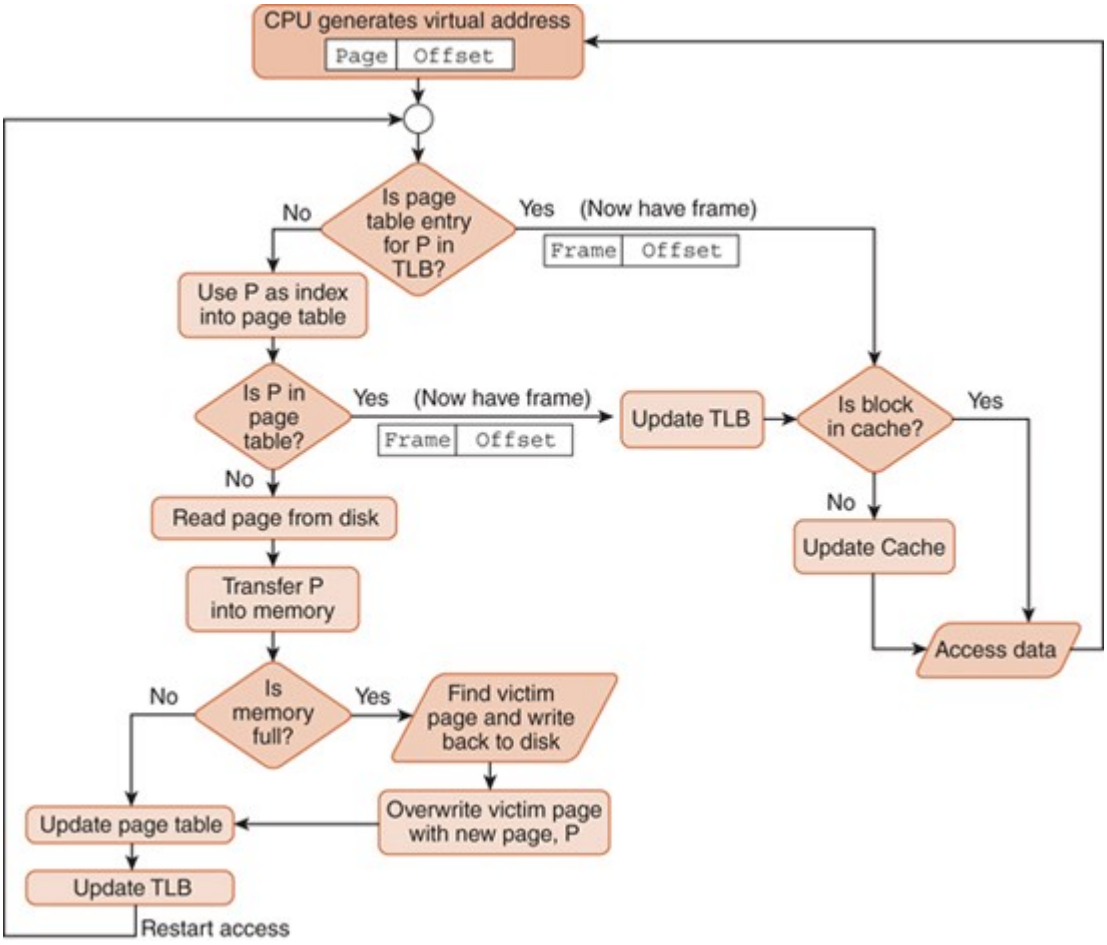
https://en.wikipedia.org/wiki/Cache_placement_policies

So how does an address access map to the cache? The addresses are divided into logical addresses, linear addresses, and physical addresses.



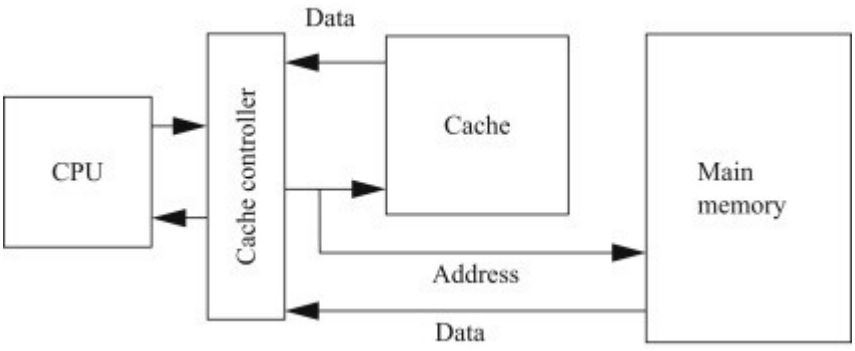
https://www.cs.odu.edu/~price/cs471/public_html/spring17/lectures/MainMemory.htm

The diagram shows the process from a logical address to a linear address to a physical address. The conversion process from linear addresses to physical addresses requires the use of page tables. The page table consists of many entries, each called a page table entry, and the entire page table is maintained by the operating system and placed in memory or on disk. Memory access takes hundreds of clock cycles, and it would be a waste of time to look at the memory page table for every address translation. So, modern computers have introduced **translation lookaside buffers** (TLBs) to speed up the process. The CPU will check the TLB every time the address is converted. If there is only one address translation, there is no need to fetch the memory page table. In the case of a TLB hit, the physical address can be selected.



https://stackoverflow.com/questions/37825859/cache-miss-a-tlb-miss-and-page-fault

Cache Associativity



https://www.sciencedirect.com/topics/computer-science/set-associative-cache

The replacement policy determines where a particular memory block can be placed when it goes into the cache. To place a block in the cache, the set is determined by the index bits

derived from the address of the memory block. The tag is stored in the tag field associated with the set.

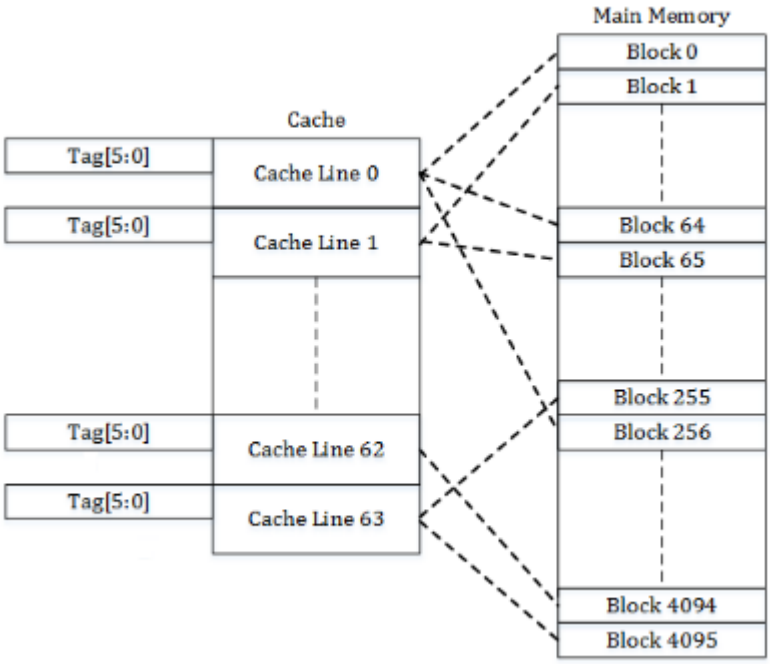
Set>Index bits>Tag

Associativity is a tradeoff. When the cache is searching for a hit, all the cache lines must be searched. It requires more power, space consumption, and time. On the other hand, caches with more associativity suffer fewer misses. Doubling the associativity, from direct mapped to 2-way, or from 2-way to 4-way, has about the same effect on hit rate as doubling the cache size. However, associativity increases beyond 4-way have much less effect on the hit rate.

There are 3 main types of mapping relationships.

1. **Direct Mapping** — Each block in the main memory can only be placed in a unique location in the cache. It is characterized by low space utilization, the highest conflict probability, and the simplest implementation.
2. **Fully Associative Mapping** — Any block in the main memory can be placed anywhere in the cache. It is characterized by high space utilization, the lowest conflict probability, and the most complex implementation.
3. **Group-Associate Mapping** — Each block in the main memory can be placed anywhere in a unique group in the cache. Group associate mapping is a compromise between direct mapping and fully associative mapping.

Direct Mapping Cache

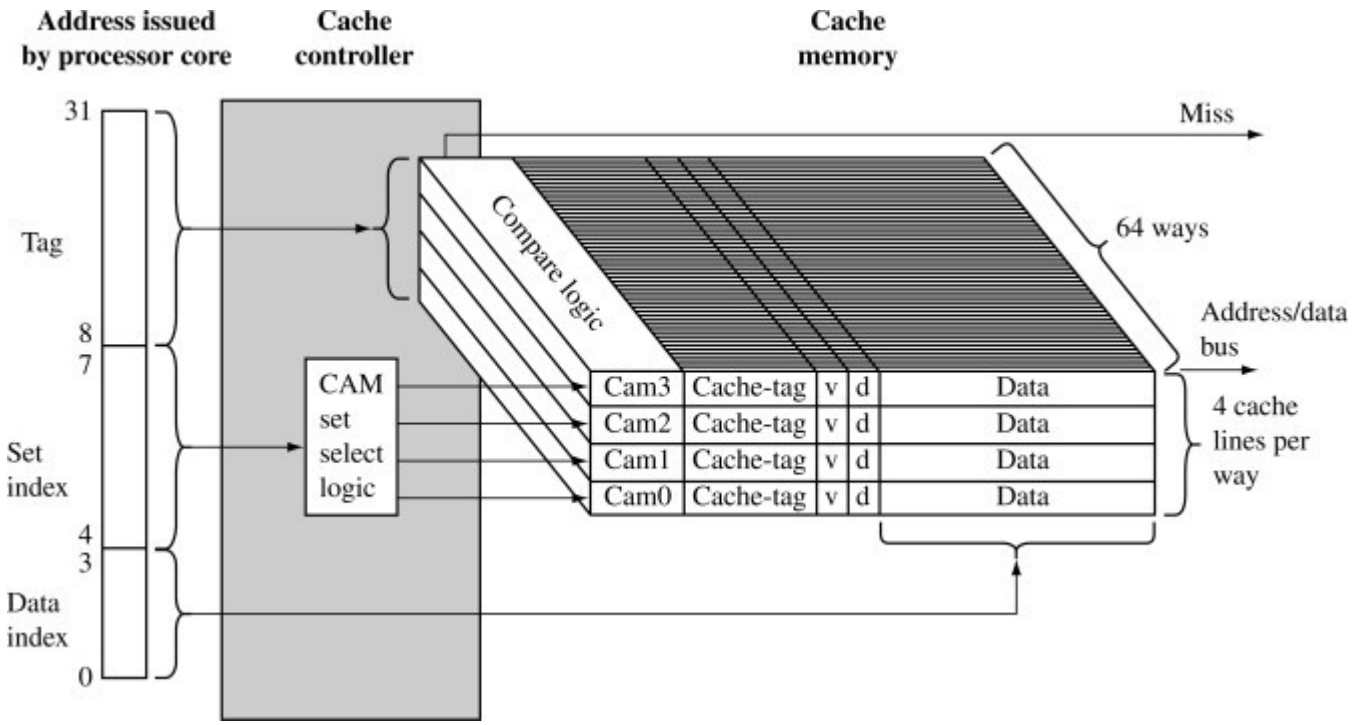


https://en.algorithmica.org/hpc/cpu-cache/associativity/

The size of the cache is called the cache size, which represents the size of the maximum data that the cache can place. We divide the cache evenly into many equal blocks, each block size is called a cache line, and its size is cache line size. Each cache line corresponds to a unique tag, and the tag stores the remaining part of the entire address (bit width minus the bits used by index and offset). The combination of a tag, index, and offset can uniquely determine an address. Each address can be directly and only mapped to a certain cache line immediately. For example, a 64 bytes cache is divided into 64 blocks equally, then the cache line is 1 byte, so we have 64 cache lines totally. If we divide the 64 bytes into 8 blocks equally, then the cache line is 8 bytes, so we have 8 cache lines for each block totally. The general cache line size is 4–128 bytes. If a block conflict occurs, the original block in the cache will be replaced unconditionally. The processor will find the corresponding cache block number directly according to the index bit in the address, take out the tag corresponding to all cache lines in the current cache block, and then compare it with the tag in the address. If the valid bit of the block number is 1, it means a hit, otherwise, it is a miss. (1 means valid, 0 means invalid). The cache controller can confirm if the current cache line data is valid according to the valid bit.

Cache>Cache Block>Cache Lines

Fully Associative Mapping



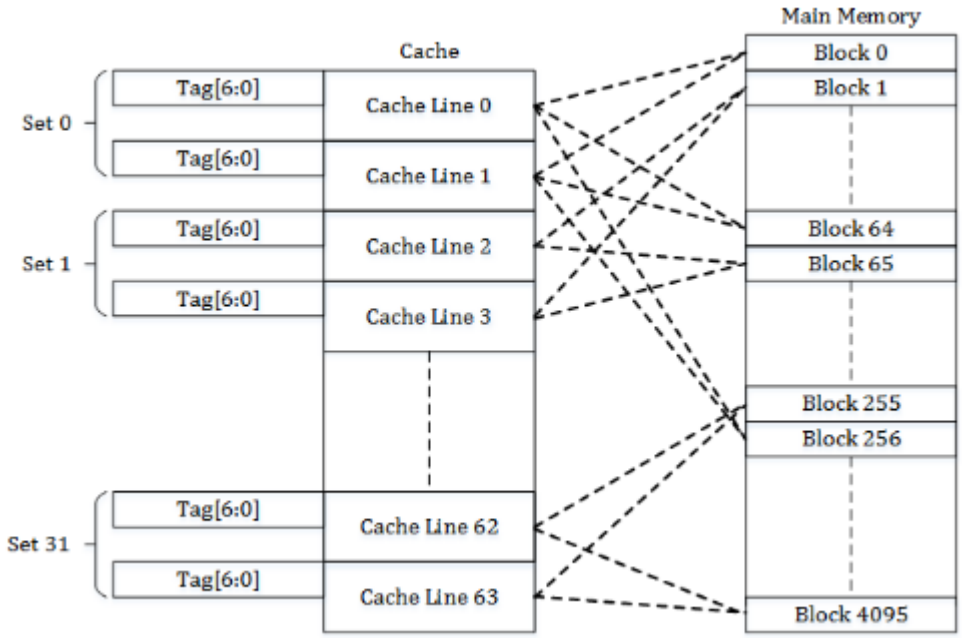
https://www.sciencedirect.com/topics/computer-science/fully-associative-cache

This cache is a fully-connected cache. All cache lines are in a group, so there is no need for the set index in the address. It is because there is only one group for you to choose from. The processor compares the tags corresponding to all cache lines according to the tag of the address. If it is equal, it means cache hit. Hence, in a fully-associated cache, data at any address can be cached in any cache line. It can minimize the frequency of cache thrashing, but the hardware cost is higher. Obviously, it provides flexibility (shortens the block conflict) and a high hit rate.

Group Associative Mapping

Divide the cache into groups of the same size, and a cache block in the main memory can be loaded into any position in the group.

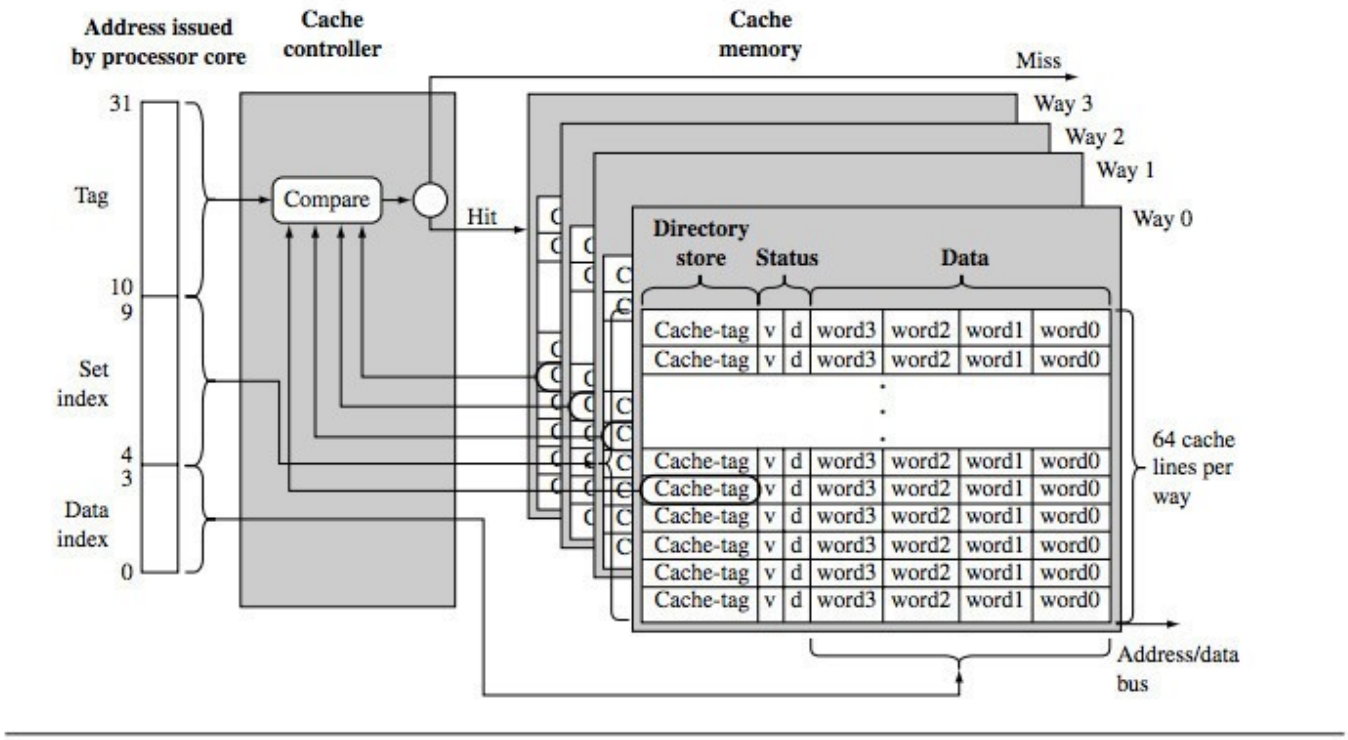
Two-way Set Associative Cache



https://en.algorithmica.org/hpc/cpu-cache/associativity/

Assume 64 bytes cache size and a cache line size of 8 bytes. Therefore, the two-way-set-associative cache divides the cache equally into 2 parts, every 32 bytes. Each part contains 4 cache lines. All cache lines with the same index will be grouped together. The processor will find the corresponding cache block number directly according to the index bit in the address, take out the tags corresponding to all cache lines in the current cache block, and then compare it with the tag in the address. If the valid bit of the block number is 1, it means a hit, otherwise, it is a miss. (1 means valid, 0 means invalid). The cache controller can confirm if the current cache block is valid according to the valid bit. The hardware cost of a two-way set-associative cache is higher than that of a direct-mapping cache. It is because it compares tags each time, it needs to compare tags corresponding to multiple cache lines. Some computers may perform parallel comparisons to increase the comparison speed, which increases the complexity of hardware design. But, it helps reduce the possibility of cache thrashing.

Four-way Set Associative Cache



https://www.researchgate.net/figure/4-KB-4-way-set-associative-cache-with-256-cache-lines_fig3_318860805

Assume a four-way set-associative cache, that is, each location in the cache stores 4 lines of data, its data size is 8 KB and each cache line size is 16 bytes. There will be 128 locations because 128 locations * 4 lines/location * 16 bytes/line = 8 KB. The concept is similar to a 2-way set-associative cache.

There are 4 important characteristics of the register.

1. **Cache Placement (Put data in the buffer)** — the direct-mapping, fully-associative mapping, and set-associative cache mapping
2. **Cache Searching (Search for data in the cache)** —the processors use the tag bits in the address tp compare with the tags in the cache. If it is matching, it means a cache hit.
3. **Cache Replacement (Replace data in the buffer)** — FIFO (First In First Out), LIFO (Last In First Out), LRU (Least Recently Used), MRU (Most Recently Used)

4. **Cache Write Policy (Write data to the main memory through the buffer)** — **Hit** (Write-through, write-back), **Miss** (write-allocate, and no-write allocate)

Cache Write Policy

Cache Update — The cache update strategy refers to how a write operation should update data when a cache hit occurs.

1. **Write-Through** — When the processor executes the store instruction and hits the cache, we update the data in the cache and update the data in the main memory. So that, the data in the cache are always consistent.
2. **Write-Back** — When the processor executes the store instruction and hits the cache, we only update the data in the cache. The modified data of each cache line will be stored in the dirty bit (the D next to the cache line). The data in the main memory is only updated when the cache line is replaced or when a clean operation is performed. Therefore, the data in the cache and the main memory may be inconsistent.

Cache Allocation Policy — It refers to when we should allocate cache lines for data. It is divided into 2 operations: read and write.

1. **Cache-Write Allocation** — It is only considered when the CPU writes data with cache misses. When we don't support write allocation, the write instruction just updates the main memory data. When write allocation is supported, we first load data from the main memory into the cache line, and then update the data in the cache line.
2. **Cache-no-write allocation/Read Allocation** — When the CPU reads data, a cache miss occurs, in which case a cache line is allocated to cache the data read from the main memory.

Illustration (Provided by University Berkeley)

For computers, memory accesses are like going to the library,

Finding the necessary information in the page of a book,

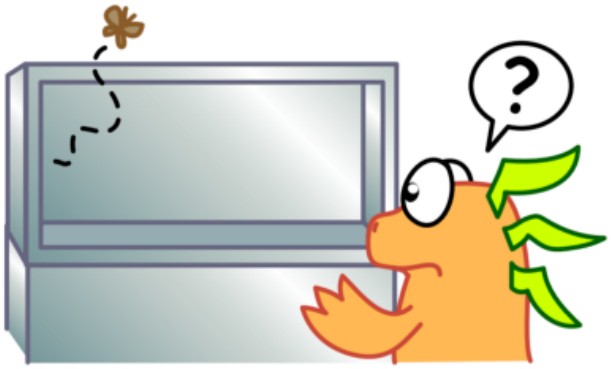
And going back home to do the work involving that information.

While computers don't mind going back and forth like this for data, it usually means users have to do a lot of waiting.

Fortunately for users, computers have caches, which is the equivalent of keeping copies of the books needed on a shelf near the workspace. Through a number of mechanisms, caches give the illusion of being able to access memory very quickly!

http://csillustrated.berkeley.edu/illustrations.php

Sometimes, the cache doesn't have the memory block the computer's looking for. When this happens, it's called a cache miss. There are three causes of cache misses. Just remember the three C's:



Compulsory

Compulsory misses happen when a block is referenced for the first time. The computer can't get a block that doesn't exist yet!

Capacity

The block is not in the cache because there is no space in the cache for it. Caches are of finite size, after all.

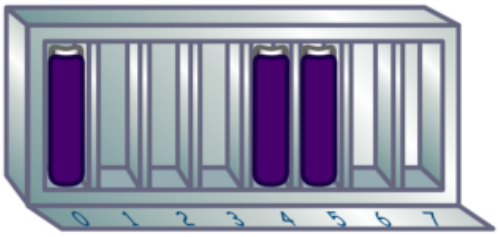
Conflict

These types of misses happen only in direct-mapped and set-associative caches. Multiple blocks can be mapped to a set, forcing evictions when the set is full.

http://csillustrated.berkeley.edu/illustrations.php

Just as bookshelves come in different shapes and sizes, caches can also take on a variety of forms and capacities. But no matter how large or small they are, caches fall into one of three categories: direct mapped, n-way set associative, and fully associative.

Direct Mapped



Tag	Index	Offset

A cache block can only go in one spot in the cache. It makes a cache block very easy to find, but it's not very flexible about where to put the blocks.

2-Way Set Associative

Tag	Index	Offset

This cache is made up of sets that can fit two blocks each. The index is now used to find the set, and the tag helps find the block within the set.

4-Way Set Associative

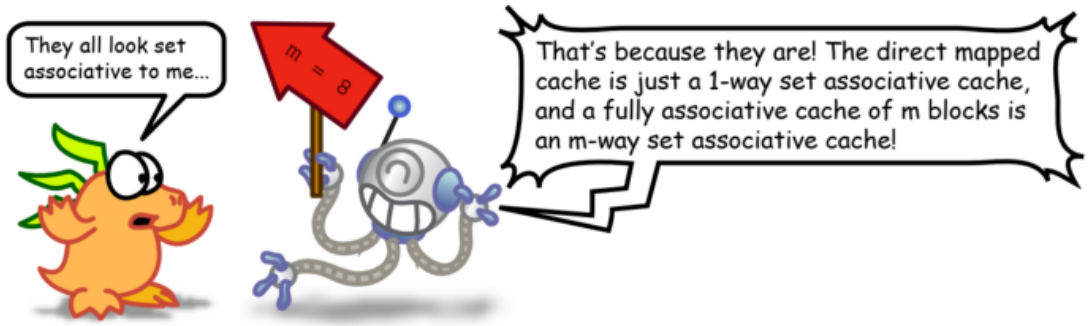
Tag	Index	Offset

Each set here fits four blocks, so there are fewer sets. As such, fewer index bits are needed.

Fully Associative

Tag	Offset

No index is needed, since a cache block can go anywhere in the cache. Every tag must be compared when finding a block in the cache, but block placement is very flexible!



Ketrina Yim

http://csillustrated.berkeley.edu/illustrations.php