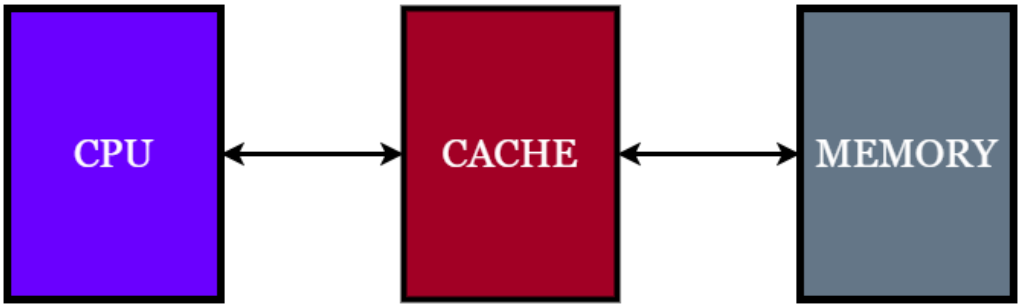
CACHE READ AND CACHE WRITE

By Yash

Modern CPUs contain a hierarchy of storage systems, with the fastest and smallest being the registers, followed by various levels of cache (L1, L2, L3, etc.), main memory (RAM), and then disk storage. The purpose of caches is to keep frequently accessed data close to the CPU to reduce the time it takes to fetch that data.



Cache Read (Load)

1. **Request Initiation**: When the CPU needs to read a data item, it first checks if the item is in the L1 cache.
2. **Cache Hit**: If the data item is found in the cache, it's called a "cache hit." The CPU can quickly access the data, and the process ends.
3. **Cache Miss**: If the data item is not found in the cache (a "cache miss"), the cache controller will check the next level of cache (e.g., L2). If it's not there, it might check the next level (e.g., L3) and so on. If it's not found in any cache level, the CPU will fetch it from main memory.
4. **Fetching and Populating**: After fetching the data from main memory (or a lower cache level), it is placed into the L1 cache (and potentially other cache levels) so that future reads of that data item are faster.

Cache Write (Store)

1. **Request Initiation**: When the CPU needs to write a data item, it will look for the location in the L1 cache.
2. **Write Policies**:
    - **Write-Through**: The data is written to the L1 cache and simultaneously to the main memory (or to the next cache level). This ensures data consistency but can be slower because every write to the cache also involves a write to the next level.
    - **Write-Back (or Write-Behind)**: The data is written only to the cache. The modified cache line is marked as "dirty." The data will be written back to main memory (or a lower cache level) only when the cache line is evicted. This is faster but can lead to data inconsistency if the cache data is not eventually written back to main memory.
3. **Cache Hit on Write**: If the location is found in the cache, the data is written to the cache based on the cache's write policy.

4. **Cache Miss on Write**: If the location isn't found in the cache, there are a couple of strategies:
    - **Write Allocate (or Fetch on Write)**: The cache line is fetched into the cache and then written. This is based on the idea that if you're writing to a location, you're likely to read from or write to it again soon.
    - **No-Write Allocate (or Write-No-Allocate)**: The data is written to the main memory (or a lower cache level) directly without bringing it into the cache.

Speed: Cache Read vs. Cache Write

The relative speeds of cache reads and writes depend on several factors, including the cache's write policy and the specific architecture of the CPU and memory system. In general:

- **Cache Read**: The speed is mainly determined by the latency of finding and retrieving the data. If there's a cache hit, the read is very fast. If there's a miss, the latency can be much higher as the data has to be fetched from a lower cache level or main memory.
- **Cache Write**: The speed can be influenced significantly by the write policy. Write-back caches can make writes appear very fast since the data is only written to the cache and not immediately to main memory. Write-through can be slower since every write to the cache must also be written to the next level.

Cache writes (stores) can seem to be faster than cache reads (loads) because the actual write-back to main memory can be delayed.

So which is faster?

In Write-Back Policy:

- **Cache Write**: Often appears faster because the data is only written to the cache initially, and the actual write-back to main memory is deferred until necessary.

- **Cache Read**: The speed is dependent on whether there's a cache hit or miss. A cache hit is fast, while a cache miss (requiring data to be fetched from main memory) can be slower.

In Write-Through Policy:

- **Cache Write**: Can be slower because every write to the cache also involves a write to the next level (either a lower cache level or main memory).

- **Cache Read**: The speed is dependent on cache hits or misses.

For systems using a **write-back policy**, cache writes (stores) can often seem faster than cache reads (loads) due to the deferred nature of writing back to main memory.

For systems using a **write-through policy**, cache writes might seem slower than reads, especially if there's a high rate of cache hits for reads.