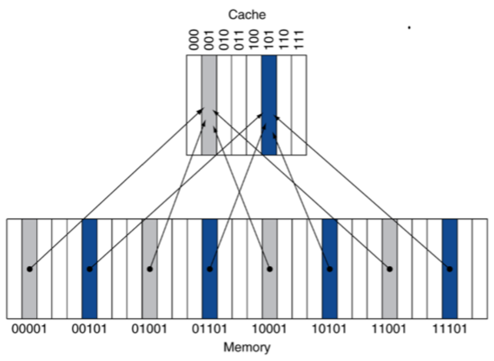
Cache Organization

Cache organization refers to how data memory is placed into the cache and how its retreived. This includes mapping strategeis, replacement policies, and write mechanisms. These choices affect performance miss rates and complexity. The organization must work in tandem with principals locality to optimize access speed.

Direct Caching



in a Direct Mapped Cache, each memory block is assigned to exactly one cache block. The index is calculated using the block address modulo the number of cache blocks. The low-order address bits are used to determine the index when the number of blocks is a power of 2. Each entry contains a tag, which is the high order bits of the memory address, and a valid bit which is initially 0 and set when valid data.

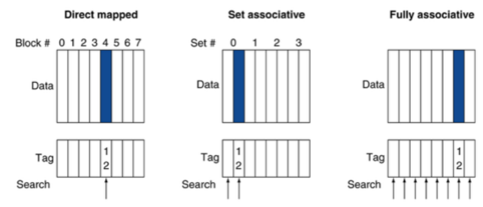
A cache hit occurs when the valid bit is set and the stored tag matches the tag portion of the requested address. A cache miss occurs when the valid bit is not set or the tags do not match, requiring a memory access and potential pipeline stall.

On a miss, the original PC value (PC-4) is sent to memory to begin data retreival. The system waits for the memory access to complete. Once the data arrives, it is written into the cache. The tag and valid bit are updated. Instruction execution is then restarted and can fetch the data from cache.

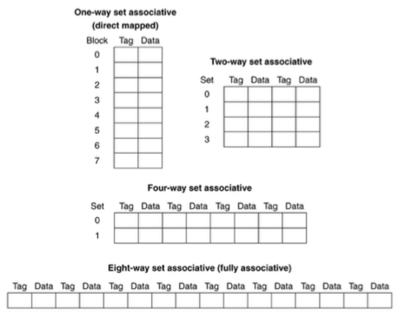
Types of cache misses:
- A compulsary miss occurs on the first access to a block and is unavoidable.
- A capacity miss happens when the cache cannot store all the blocks required
- A conflict miss occurs when multiple blocks map to the same cache location

Associativity

In a Set Associative Cache each block can go into one of several locations within a specific set. In a fully associative cache, any block can go into any location



Associativaty affects flexibility and complexity. For example in an 8-entry cache, 1-way associativity is direct mapped and 8-way is full associative


One-way set associative (direct mapped)
Block Tag Data
0 1 2 3 4 5 6 7
Two-way set associative
Set Tag Data Tag Data
0 1 2 3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0 1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

In a set associative cache, each memory address is divided into tag, index, and byte offset. the index determines the set. Each set holds multiple entries and the tag is compared to each entry in the set to find a match.

In direct mapped caches each block has a fixed location. In set associative caches replacement is only needed when all entries in a set are full. If a non-valid entry is available, it is used. If not, the system may use Least Recently Used (LRU), which replaces the block that has not been accessed for the longest time. LRU is simple in 2-way caches but becomes complex at 4-way or higher. Random placement is another option and performs similarly to LRU

Increased associativity reduces miss rates but has diminishing returns.

Cache Placement:

First, know how many sets the cache has. To do this, divide the total number of blocks by the number of ways (blocks per set). For example, if a cache has 32 blocks and is 4-way set associative, it would have 8 sets.

To figure out which set each address goes into, calculate the index by using the formula: address % number of sets.

Once you know which set to use, check if the block is already in that set. If it is, it's a cache hit. If its not, and theres still space in the set, you insert the block into the first available slot. This is considered a cold miss and doesnt count as replacement

Writing

A cache must manage how data is written. This involves deciding wether update main memory immediately or delay a write, and how to handle write misses. These choices affect consistency, performance and complexity.

When the CPU performs a store instruction, it writes data to memory. With a cache in place, this raises the issue of whether the write goes to cache, main memory, or both. If data is only written to the cache, main memory may have a different value, causing inconsistency.

Write Strategies:

- Write through updates both cache and main memory simultaneously. Its simple but slow. A write buffer is often used to reduce stalls.
- Write back only updates the data in the cache during a write hit; main memory is not updated right away. Each block is tracked with a dirty bit to indicate if it has been modified since being loaded from memory. If a block is clean, it can be replaced without writing to memory. If a block is dirty, it must be written back to memory before being replaced.