Caches

### LKN Caches

L Number of **bytes** per cache line.

K Number of **cache lines** per set.

N Number of **sets**.

N = 1 ⇒ **Fully Associative**

K = 1 ⇒ **Direct Mapped**

else ⇒ **Set Associative**

|  |  |
| --- | --- |
| **Fully Associative** | **Direct-Mapped** |
| ✓ Full cache utilisation.  ✓ Better cache hit rate.  ✘ Slow replacement policy.  ✘ More power-hungry.  ✘ Expensive hardware implementation. | ✓ More power efficient.  ✓ Cheap hardware implementation.  ✘ Lower cache hit rate. |

32 0

|  |  |  |  |
| --- | --- | --- | --- |
| Tag | | Set Number | Offset |
|  | | log2N | log2L |

### Searching a Cache

1. Calculate the **set number** from the address.
2. Calculate the **tag** from the address.
3. Compare this tag with the tag in each cache line of the set:
   1. If the tag exists (**cache hit**), return the data at the offset index.
   2. If the tag does not exist (**cache miss**), read the data from memory and replace a victim cache line in the set.

The K cache lines are accessed concurrently so that the data can be routed quickly to the output buffers on a cache hit.

### Write-Through vs. Write-Back

|  |  |
| --- | --- |
| **Write-Through** | **Write-Back** |
| **Write Hit**:  Update cache line **and** main memory.  **Write Miss**:  Update main memory **only**.  *or*  Fill a victim cache line from memory.  Write changes to cache line **and** memory. | **Write Hit**:  Update cache line **only**.  (Update memory when line is replaced.)  **Write Miss**:  Fill a victim cache line from memory.  (Write this cache line to memory before replacement if dirty.) |

### Cache Misses

1. Compulsory Misses that occur because the cache is **empty** at the start.
2. Capacity Misses that occur because of cache **size**.
3. Conflict Misses that occur because of cache **associativity**.

Total Misses = Compulsory + Capacity + Conflict

### Victim Cache

This organisation achieves the performance of a

2-way cache with the implementation cost of a direct-mapped cache.

This cache organisation uses two caches:

1. A large direct-mapped cache.
2. A small fully-associative **victim cache**.

On a cache **miss**, search the victim cache before searching the next level cache in memory hierarchy.

When data is ejected from the direct-mapped cache, it is saved in the victim cache.

### Cache Coherency

If the data in main memory is changed, these changes need to be reflected in the cache.

The cache **snoops** on the bus and observes changes to memory. If relevant cache lines are affected, they are invalidated. The next time the CPU accesses the location, the cache lines will be updated.

Virtual Caches

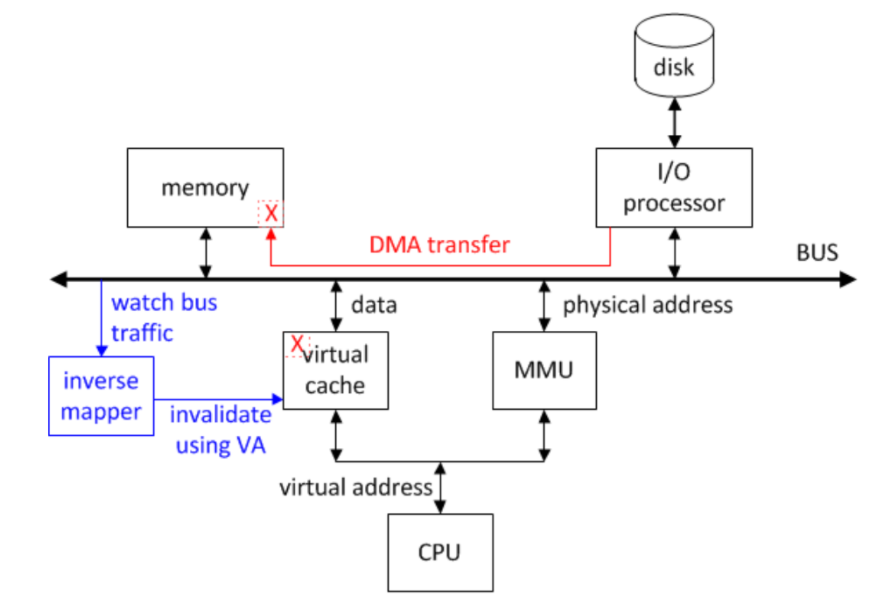
A virtual cache stores virtual addresses as opposed to physical addresses.

|  |  |
| --- | --- |
| Advantages | Disadvantages |
| Eliminates the need for address translation. | Aliasing - Processes use the same virtual addresses.  On TLB miss, can’t fill TLB from cache.  More difficult to maintain cache coherency. |

### Cache Coherency

Virtual addresses are stored in the cache, but physical addresses are used on the bus.

Physical addresses must be converted on the bus using an **inverse mapper**.



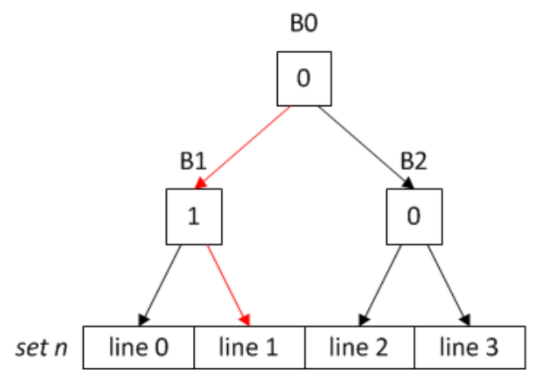
In order to solve this, both virtual **and** physical tags could be stored in each cache line.

CPU uses virtual addresses, bus uses physical addresses.

### Pseudo LRU

The Intel 486 uses a pseudo least-recently-used access sequence in order to determine a victim cache line.

Extra bits are stored in each cache set to represent a tree-like structure.



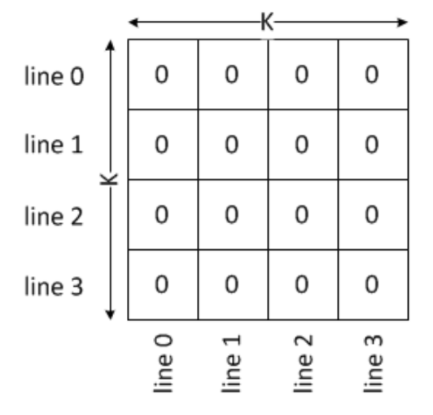
If 0, go left.

If 1, go right.

When a victim cache line is reached, flip the bits of the path traversed.

### Real LRU

Using a K2 matrix of bits for each set, we can represent the least recently used cache line.



**If cache hit:**

Set row to all 0s

Set column to all 1s

The least-recently-used cache line contains a row of 0s.

Cache Trace Analysis

Methods of **address trace collection**:

* Logic Analyser
* Software Machine Simulator
* Instruction Trace Mechanism
* Microcode Modification

Multiple analyses per run can be accomplished by noting the position of each hit in the cache set. The first K hit counts can be summed together to calculate the hits for a K-way cache.

### Trace Stripping

A reduced trace can be created by only outputting addresses that produce misses. In a 1-way cache a hit doesn’t change the state of the cache, so misses represent **all** the state changes.

Addresses which map onto set 0 in a 4 set cache will map onto the same set on an 8 set cache.