**Cache Memory**

Cache is a small high-speed memory. Stores data from some frequently used addresses (of main memory).

**Cache hit** Data found in cache. Results in data transfer at maximum speed.

**Cache miss** Data not found in cache. Processor loads data from M and copies into cache. This results in extra delay, called miss penalty.

Hit ratio = percentage of memory accesses satisfied by the cache.

Miss ratio = 1-hit ratio
**Cache Line**

Cache is partitioned into **lines** (also called blocks). Each line has **4-64** bytes in it. During data transfer, **a whole line** is read or written.

Each line has a **tag** that indicates the address in M from which the line has been copied.

<table>
<thead>
<tr>
<th>Index</th>
<th>Tag</th>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2</td>
<td>ABC</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>DEF</td>
</tr>
</tbody>
</table>

**Cache**

Cache hit is detected through **an associative search** of all the tags. Associative search provides a **fast response** to the query:

"**Does this key match with any of the tags?**"

Data is read only if a match is found.
Types of Cache

1. Fully Associative
2. Direct Mapped
3. Set Associative

Fully Associative Cache

"No restriction on mapping from M to C."

Associative search of tags is expensive.

Feasible for very small size caches only.
The secret of success
Program locality.

Cache line replacement
To fetch a new line after a miss, an existing line must be replaced. Two common policies for identifying the victim block are

- LRU (Least Recently Used)
- Random

Estimating Average Memory Access Time

Average memory access time =
Hit time + Miss rate × Miss penalty

Assume that
Hit time = 5 ns
Miss rate = 10%
Miss penalty = 100 ns.
The average memory access time = 15 ns.
Better performance at a cheaper price.
Direct-Mapped Cache

A given memory block can be mapped into one and only cache line. Here is an example of mapping

<table>
<thead>
<tr>
<th>Cache line</th>
<th>Main memory block</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0, 8, 16, 24, ... 8n</td>
</tr>
<tr>
<td>1</td>
<td>1, 9, 17, 25, ... 8n+1</td>
</tr>
<tr>
<td>2</td>
<td>2, 10, 18, 26, ... 8n+2</td>
</tr>
<tr>
<td>3</td>
<td>3, 11, 19, 27, ... 8n+3</td>
</tr>
</tbody>
</table>

**Advantage**
No need of expensive associative search!

**Disadvantage**
Miss rate may go up due to possible increase of mapping conflicts.
Set-Associative Cache

Two-way Set-associative cache

\[ C \rightarrow M \]

\[ \begin{align*}
\text{set 0} & \\
\text{set 1} & \\
\text{Set 3} & 
\end{align*} \]

\emph{N-way set-associative cache}

Each M-block can now be mapped into any one of a set of \( N \) C-blocks. The sets are predefined. Let there be \( K \) blocks in the cache. Then

\[ N = 1 \quad \text{Direct-mapped cache} \]
\[ N = K \quad \text{Fully associative cache} \]

Most commercial cache have \( N= 2, 4, \) or \( 8. \)
Cheaper than a fully associative cache.
Lower miss ratio than a direct mapped cache.

\textit{But direct-mapped cache is the fastest.}
Address translation: an example

Main memory size = 2 KB
Block size = 8 bytes
Cache size = 64 bytes
Set size = 2
No. of sets in cache = 4

No of sets = 4 = 2^2  Block size = 8 = 2^3
6    2
|   |   3
|   |
|---|---|---|
|Tag| index| offset|

Memory address

To locate an M-block in cache, check the tags in the set \( S = (M\text{-block}) \mod (\text{number of sets}) \) i.e. the index field.
**Specification of a cache memory**

<table>
<thead>
<tr>
<th></th>
<th>Block size</th>
<th>Hit time</th>
<th>Miss penalty</th>
<th>Miss rate</th>
<th>Cache size</th>
<th>Cache speed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4-64 byte</td>
<td>1-2 cycle</td>
<td>8-32 cycles</td>
<td>1-20%</td>
<td>8KB-64KB</td>
<td>0.5 ns (8 GB/sec)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>6-10 cycles</td>
<td></td>
<td>128KB-2 MB</td>
<td>0.75 ns (6 GB/sec)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>2-22 cycles</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

What happens to the cache during a write operation?