Cache memories

A cache is a small, fast memory which is transparent to the processor.

- The cache duplicates information that is in main memory.
- With each data block in the cache, there is associated an identifier or tag. This allows the cache to be content addressable (e.g., like translation lookaside buffers).


table
<table>
<thead>
<tr>
<th>Tag</th>
<th>Cache</th>
</tr>
</thead>
<tbody>
<tr>
<td>37</td>
<td></td>
</tr>
<tr>
<td>26</td>
<td>information</td>
</tr>
<tr>
<td>49</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
</tr>
</tbody>
</table>

- A replacement rule must be used to purge entries from the cache when the space is needed for new information.
- Caches are smaller and faster than main memory. **Backing**, or secondary storage, on the other hand, is larger and slower.

- Speed differences between cache memory and virtual memory.
  - Main memory is about 10–50 times slower than a cache,
  - but secondary memory is ≈ 1000 times slower than main memory.
- A cache miss is the term analogous to a page fault.
  - Cache misses must be handled much more quickly than page faults. Thus, they are handled in hardware.
Caches can be organized according to four different strategies:
- Direct
- Fully associative
- Set associative
- Sectored

A cache implements several different policies for retrieving and storing information, one in each of the following categories:
- Fetch policy—determines when information is loaded into the cache.
- Replacement policy—determines what information is purged when space is needed for a new entry.
- Write policy—determines how soon information in the cache is written to lower levels in the memory hierarchy.

Cache memory organization
Information is moved into and out of the cache in blocks. When a block is in the cache, it occupies a cache line. Blocks are usually larger than one byte (or word),

- to take advantage of locality in programs, and
- because memory may be organized so that it can overlap transfers of several words at a time.

The block size is the same as the line size of the cache.

A placement policy determines where a particular block can be placed when it goes into the cache. E.g., is a block of memory eligible to be placed in any line in the cache, or is it restricted to a single line?

In our examples, we assume—
- The cache contains 2048 words,
Thus it has ______ lines.

- Main memory is made up of 256K words, or 16384 blocks. Thus an address consists of

We want to structure the cache to achieve a high hit ratio.

- Hit—the referenced information is in the cache.
- Miss—referenced information is not in cache, must be read in from main memory.

\[
\text{Hit ratio} = \frac{\text{Number of hits}}{\text{Total number of references}}
\]

As noted above, there are four different placement policies.

**Direct**

Only 1 choice of where to place a block.

block \( i \) \( \rightarrow \) line \( i \mod 128 \)

Each line has its own tag associated with it.

When the line is in use, the tag contains the high-order seven bits of the main-memory address of the block.
To search for a word in the cache,

1. Determine what line to look in (easy; just select bits 10–4 of the address).

2. Compare the leading seven bits (bits 17–11) of the address with the tag of the line. If it matches, the block is in the cache.

3. Select the desired word from the line.

**Advantages:**

- Fast lookup (only one comparison needed).
- Cheap hardware (no associative comparison).
- Easy to decide

**Disadvantage:** Contention for lines.
Fully associative
A full 128 choices of where to place a block.

block \( i \) \( \rightarrow \) any free (or purgeable) cache location

Each line has its own tag associated with it.

When the line is in use, the tag contains the high-order *fourteen* bits of the main-memory address of the block.

To search for a word in the cache,

1. Simultaneously compare the leading 14 bits (bits 17-4) of the address with the tag of all lines. If it matches any one, the block is in the cache.
2. Select the desired word from the line.

*Advantages:*

Minimal contention for lines.
Wide variety of replacement algorithms feasible.

Disadvantage:

The most expensive of all organizations, due to the high cost of associative-comparison hardware.

A flowchart of cache operation: The process of searching a fully associative cache is very similar to using a directly mapped cache. Let us consider them in detail.

Note that this diagram assumes a separate TLB.

Which steps would be different if the cache were directly mapped?
Set associative

1 < $n$ < 128 choices of where to place a block.

A compromise between direct and fully associative strategies.

The cache is divided into $s$ sets, where $s$ is a power of 2.

block $i \rightarrow$ any line in set $i \mod s$

Each line has its own tag associated with it.

When the line is in use, the tag contains the high-order eight bits of the main-memory address of the block. (The next six bits can be derived from the set number.)

To search for a word in the cache,
1. Select the proper set \((i \mod s)\).

2. Simultaneously compare the leading 8 bits (bits 17–10) of the address with the tag of all lines in the set. If it matches any one, the block is in the cache.

   At the same time, the (first bytes/words of) the lines are also being read out so they will be accessible at the end of the cycle.

3. If a match is found, gate the data from the proper block to the cache-output buffer.

4. Select the desired word from the line.

- All reads from the cache occur as early as possible, to allow maximum time for the comparison to take place.
- Which line to use is decided late, after the data have reached high-speed registers, so the processor can receive the data fast.

To attain maximum speed in accessing data, we would like to start searching the cache \textit{at the same time} we are looking up the page number in the TLB.
When the bit-selection method is used, both can be done at once if the page number is disjoint from the set number.

This means that

- the number of bits \( k \) in the set number
- + the number of bits \( j \) which determine the byte (or word) within a line
- must be \( \leq \) the number of bits \( d \) in the displacement field.

(If the page size is \( 2^d \), then there will be \( d \) bits in the displacement field.)

Factors influencing line lengths:

- Long lines \( \Rightarrow \) higher hit ratios.
- Long lines \( \Rightarrow \) less memory devoted to tags.
- Long lines \( \Rightarrow \) longer memory transactions (undesirable in a multiprocessor).
- Long lines \( \Rightarrow \) more write-backs (explained below).

For most machines, line sizes between 16 and 64 bytes perform best.

If there are \( b \) lines per set, the cache is said to be \( b \)-way set associative.

The logic to compare 2, 4, or 8 tags simultaneously can be made quite fast.

But as \( b \) increases beyond that, cycle time starts to climb, and the higher cycle time begins to offset the increased associativity.
Almost all caches built today are either direct mapped, or 2- or 4-way set-associative.

As cache size is increased, does a high degree of set associativity become more important?

- A high degree of set associativity gives more freedom of where to place a block in the cache. So it tends to decrease the chance of a conflict requiring a purge.
- A large cache also tends to decrease the chance that two blocks will map to the same line. So it tends to decrease the chance of a conflict requiring a purge.

Consider a pathological case: A cache with 128 lines and a program that references every 128th block. Each reference causes a miss in a direct-mapped cache.

- Increasing the set associativity increases the hit ratio.
- Increasing the cache size also increases the hit ratio.

So as cache size is increased, high associativity becomes less important.

Sectored

With a sectored cache, main memory is partitioned into sectors, each containing several blocks.

The cache is partitioned into sector frames, each containing several lines. (The number of lines/sector frame = the number of blocks/sector.)

When block \(b\) of a new sector \(c\) is brought in,
- it is brought into line \(b\) within some sector frame \(f\), and
- the rest of the lines in sector frame \(f\) are marked invalid.

Thus, if there are \(S\) sector frames, there are \(S\) choices of where to place a block.

There are 128 lines in the cache. Assume there are 16 blocks per sector. Then there are only 8 sector frames in the cache.
Thus, when a frame is in use, its tag contains the high-order ___ bits of the main-memory address of the block.

Since there are relatively few tags, all tags can be searched simultaneously.

Sectored caches were popular in the early days of mainframes. They have made a comeback recently, with the increasing popularity of on-chip caches, where space is severely limited.

*Diminishing returns:* As the cache size grows larger, it becomes more and more difficult to improve performance by increasing the size of the cache:
The 30% rule: Doubling cache size reduces the number of misses by 30%.

Cache memory need not grow linearly with physical memory. (Program locality doesn’t grow linearly with physical memory.)

Modern processor architecture uses two, or even three, levels of caches.

The first level captures most of the hits.
The second level is still much faster than main memory.
Performance is similar to what would be obtained with a cache as fast as the first level.
Cost is similar to that of the second level.