Chapter 4c: Set-Associative Cache

Gene Cooperman

Copyright ©2026 Gene Cooperman, gene@ccs.neu.edu
This text may be copied as long as the copyright notice remains and no text is modified.

Contents

1 Set-Associative Cache **1**
1.1 The Idea in One Sentence . . . 1
1.2 The Address: Same Three Fields as Before . . . 2
1.3 The Cache Array . . . 2
1.4 Lookup: First Direct-Mapped, Then Fully Associative . . . 2
1.5 Example: The Thrashing Problem Goes Away . . . 3
1.6 One Design, Three Names . . . 4
1.7 Summary . . . 4

Copyright © 2026 Gene Cooperman, gene@ccs.neu.edu
This text may be copied as long as the copyright notice remains and no text is modified.

THIS IS STILL A WORK IN PROGRESS.

1 Set-Associative Cache

Chapters 4a and 4b described two extremes:

- The **fully associative cache** lets any data block go into any cache line. It never suffers from conflict misses, but it needs **N comparators**, one per cache line.
- The **direct-mapped cache** forces each data block into exactly one cache line. It needs only **one comparator**, but two busy data blocks can fight over the same cache line (**thrashing**).

Real CPU caches use a compromise, the **set-associative cache**. The good news is that this chapter has no new mechanisms. A set-associative cache is just the two earlier designs glued together.

1.1 The Idea in One Sentence

> **Use the index bits to select a small group of cache lines (as in a direct-mapped cache), and then search that small group in parallel (as in a fully associative cache).**

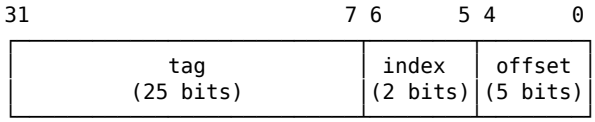
The small group of cache lines is called a **set**. A cache whose sets each contain k cache lines is called a **k-way set-associative cache**. The phrase *k-way* simply means that each set holds k cache lines. When we need to name a position within a set, we call it a **slot**: slot 0, slot 1, and so on.

For software readers, a hash table with buckets is a good analogy. The index bits act as the hash function: they choose a bucket (a set). Each bucket can hold up to k entries (data blocks), and a lookup searches only the k entries in that one bucket. A direct-mapped cache is a hash table in which every bucket holds exactly one entry, so a collision always evicts. A fully associative cache is a hash table with just one bucket, which holds everything.

1.2 The Address: Same Three Fields as Before

The address is split into **tag**, **index**, and **offset**, exactly as in Chapter 4b. The only change is in what the index selects: a **set**, rather than a single cache line.

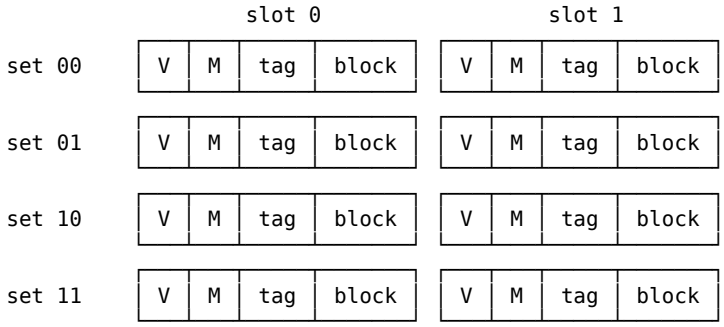
Using the same example as Chapter 4b (32-bit addresses, 32-byte data blocks, and 4 sets), the split is unchanged:



In general, for a cache with 2^s sets and 2^b bytes per data block, there are s index bits and b offset bits. All of the remaining high-order bits are the tag. The number of cache lines per set, k, does not appear in this formula: k changes how many cache lines sit inside each set, not how many sets there are.

1.3 The Cache Array

Here is a 2-way set-associative cache with 4 sets. Compare it to the cache array of Chapter 4b: it is the same picture, except that each row now holds **two** cache lines (two slots) instead of one.

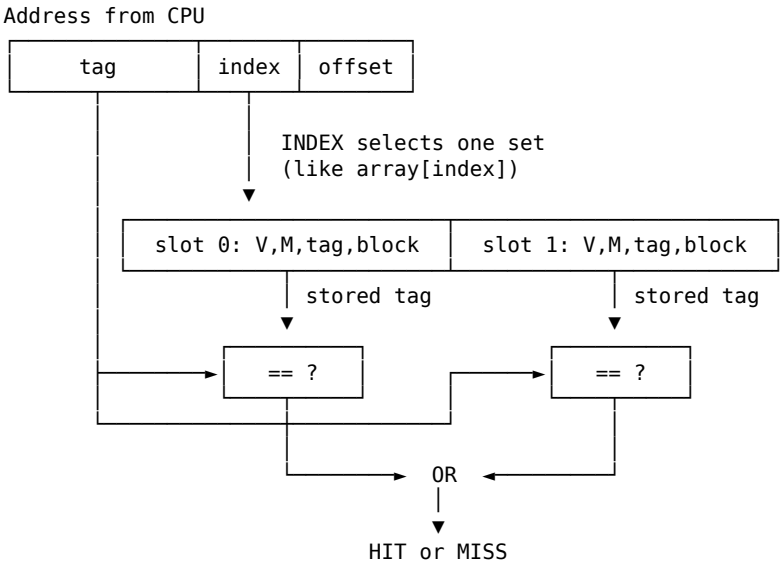


Each cache line is exactly what it was before: a valid bit, a modified bit, a stored tag, and a 32-byte data block.

1.4 Lookup: First Direct-Mapped, Then Fully Associative

When the CPU executes `lw` or `sw` with address `ADDR`:

1. **Direct-mapped step:** The index bits of `ADDR` select one set, just like an array subscript `cache[index]`.
2. **Fully associative step:** The tag of `ADDR` is compared against the stored tags of the k cache lines in that set, all in parallel, using **k comparators**.
3. If some cache line in the set has V=1 and a matching tag, it is a **HIT**, and the offset bits select the byte within its data block. Otherwise, it is a **MISS**.



On a miss, the new data block must go somewhere in the selected set:

- If some cache line in the set has V=0, use it.
- Otherwise, evict the **least recently used** cache line *within that set* (LRU, as in Chapter 4a, but choosing among only k cache lines instead of N).
- As always, if the evicted cache line has M=1, write its data block back to RAM first.

1.5 Example: The Thrashing Problem Goes Away

In Chapter 4b, we saw that two addresses with the same index bits can thrash in a direct-mapped cache. Consider these two addresses, using the address split above:

- A = `0x0000_0044` = `...0000 0000 0100 0100`: tag = `0x0000000`, index = `10`, offset = 4
- B = `0x0000_0844` = `...0000 1000 0100 0100`: tag = `0x0000010`, index = `10`, offset = 4

Both addresses have index `10`. In the 4-line direct-mapped cache of Chapter 4b, a program alternating between A and B would miss on every access. In the 2-way set-associative cache, both data blocks fit side by side in set `10`:

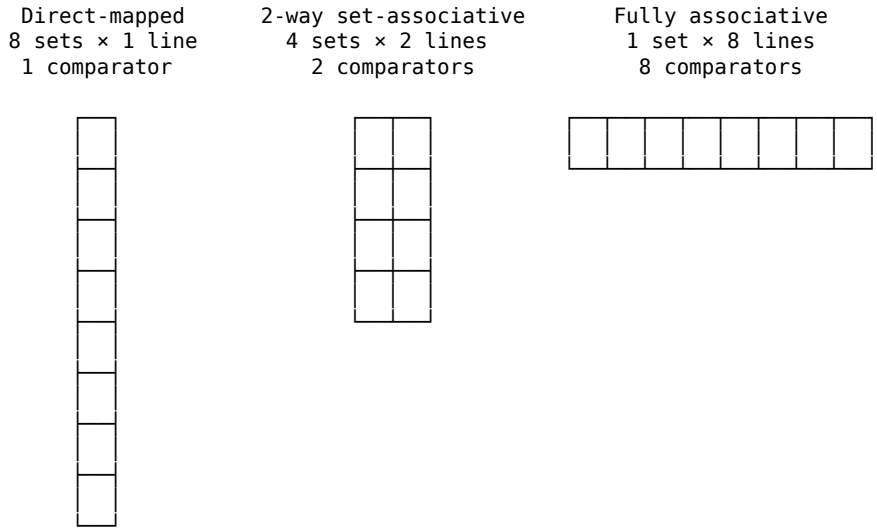
Access	Set 10, slot 0	Set 10, slot 1	Result
A	A	(empty)	MISS
B	A	B	MISS
A	A	B	HIT
B	A	B	HIT
…	A	B	HIT

After two unavoidable first-time misses, every access is a hit.

Of course, if a program alternates among **three** busy data blocks that all map to the same set, then a 2-way cache can still thrash. That is why real CPU caches are typically 4-way to 16-way. For example, a common L1 data cache is 8-way: it has 64 sets of 8 cache lines each, with 64-byte data blocks: 64 × 8 × 64 bytes = 32 KB.

1.6 One Design, Three Names

The set-associative cache contains the other two designs as special cases. Here are three arrangements of the same 8 cache lines:



- A **direct-mapped** cache is a **1-way** set-associative cache: every set has just one cache line.
- A **fully associative** cache is a set-associative cache with just **one set**: there are no index bits, and every cache line is searched.

Moving from left to right, each data block has more cache lines to choose from (fewer conflict misses), but more comparators fire on each access (more energy and chip area). A set-associative cache lets the designer pick a point in between.

1.7 Summary

Concept	Description
Set	A small group of cache lines, selected by the index bits
Slot	One position (cache line) within a set: slot 0, slot 1, …
k-way set-associative	Each set holds k cache lines (“k-way” just means k cache lines per set)
Address fields	TAG | INDEX | OFFSET, as in direct-mapped; the index now selects a set
Lookup	Index selects a set (direct-mapped step); k comparators check the set in parallel (fully associative step)
Comparators	k per access, no matter how many sets the cache has
Eviction	LRU among the k cache lines of the selected set; write-back if M=1
Conflict misses	Still possible, but only when more than k busy data blocks share one set

Concept	Description
Special cases	1-way = direct-mapped; one set = fully associative