COMPUTER ORGANIZATION AND DESIGN
The Hardware/Software Interface
5th Edition

Chapter 5

Set-Associative Cache Architecture

Performance Summary

- When CPU performance increases:
  - Miss penalty becomes more significant.
  - Greater proportion of time spent on memory stalls.
- Increasing clock rate:
  - Memory stalls account for more CPU cycles.
- Can't neglect cache behavior when evaluating system performance.

Review: Reducing Cache Miss Rates #1

Allow more flexible block placement

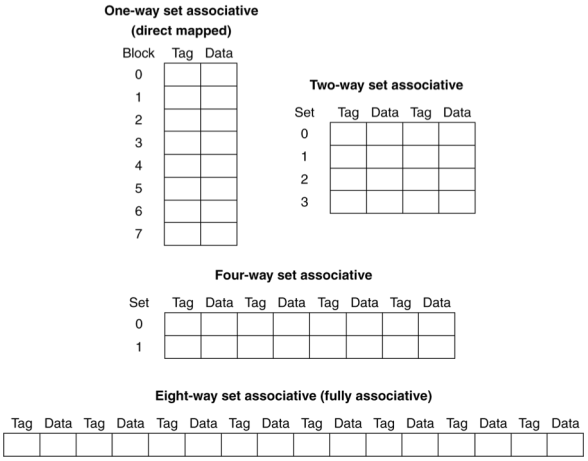
- In a ***direct mapped cache*** a memory block maps to exactly one cache block.
- At the other extreme, we could allow a memory block to be mapped to *any* cache block – ***fully associative cache.***
- A compromise is to divide the cache into ***sets,*** each of which consists of n "ways" (***n-way set associative***). A memory block maps to a unique set - specified by the index field - and can be placed any where in that set.

Associative Caches

- Fully associative cache:
  - Allow a given block to go in any cache entry.
  - On a read, requires all blocks to be searched in parallel.
  - One comparator per entry (expensive).
- *n*-way set associative:
  - Each set contains *n* entries.
  - Block number determines which set the requested item is located in.
  - Search all entries in a given set at once.
  - *n* comparators (less expensive).

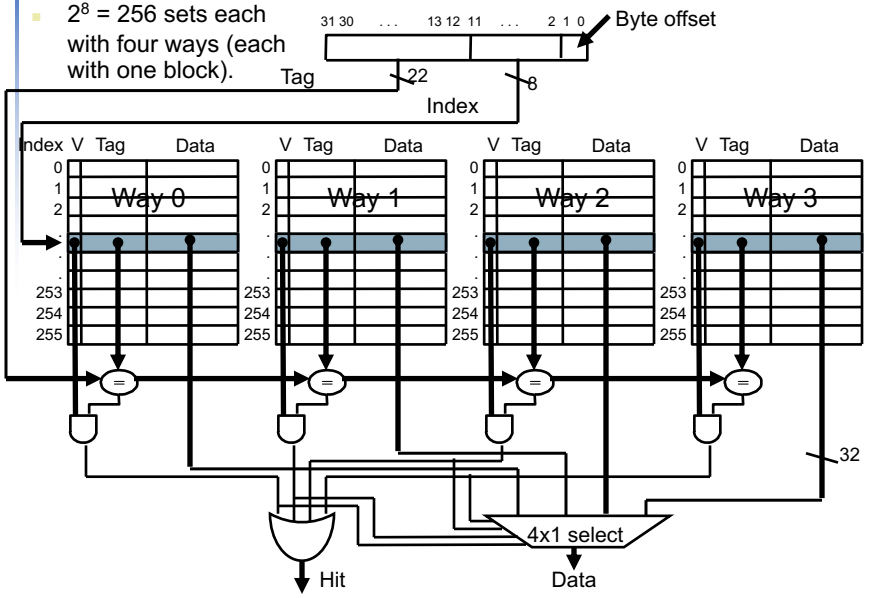
Spectrum of Associativity

- For a cache with 8 entries:



Four-Way Set Associative Cache

- 2^8 = 256 sets each with four ways (each with one block).

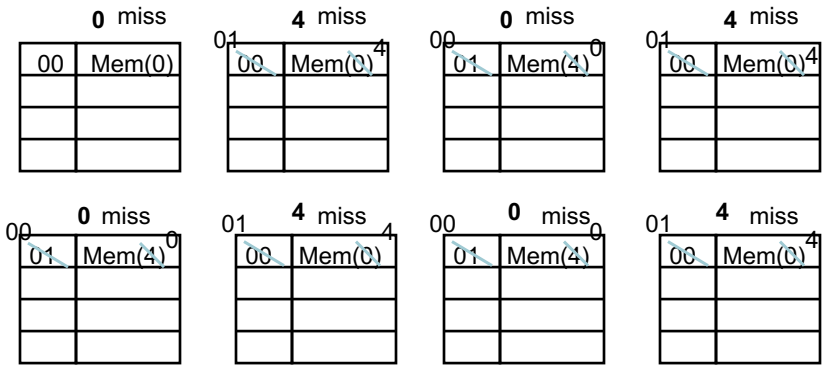


Another Direct-Mapped Catch Example

- Consider the main-memory word reference string

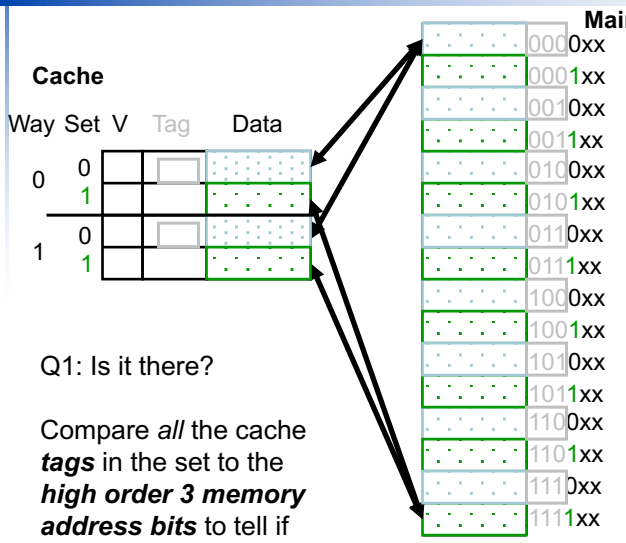
Start with an empty cache - all blocks initially marked as not valid.

0 4 0 4 0 4 0 4



- 8 requests, 8 misses
- Ping-pong effect due to conflict misses - two memory locations that map into the same cache block.

Set Associative Cache Example



- Each word is 4 bytes.
- Two low-order bits select the byte in the word.
- Two Sets per Way.

Q1: Is it there?

Compare *all* the cache ***tags*** in the set to the ***high order 3 memory address bits*** to tell if the memory block is in the cache.

Q2: How do we find it?

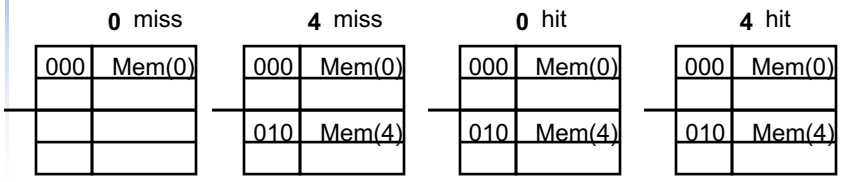
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache).

2-way Set Associative Memory

- Consider the main memory word reference string

Start with an empty cache - all blocks initially marked as not valid.

0 4 0 4 0 4 0 4

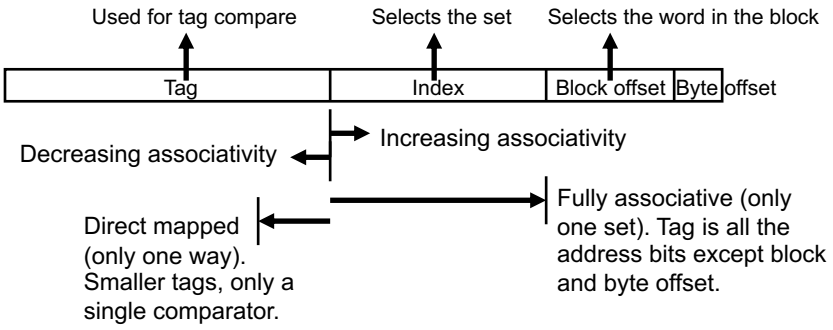


  - 8 requests, 2 misses

- Solves the ping pong effect in a direct-mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist.

Range of Set Associative Caches

- For a fixed size cache, each increase by a factor of two in associativity doubles the number of blocks per set (i.e., the number or ways) and halves the number of sets – decreases the size of the index by 1 bit and increases the size of the tag by 1 bit.



How Much Associativity is Right?

- Increased associativity decreases miss rate
  - But with diminishing returns.
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Costs of Set Associative Caches

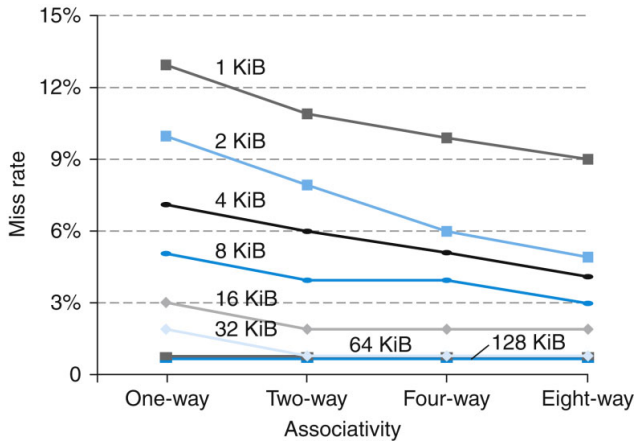
- N-way set associative cache costs:
  - N comparators - delay and area.
  - MUX delay - set selection - before data is available.
  - Data available ***after*** set selection, and Hit/Miss decision. In a direct-mapped cache, the cache block is available ***before*** the Hit/Miss decision:
    - So its not possible to just assume a hit and continue and recover later if it was a miss.
- When a miss occurs, which way's block do we pick for replacement?

Replacement Policy

- Direct mapped - no choice.
- Set associative:
  - Prefer non-valid entry, if there is one.
  - Otherwise, choose among entries in the set.
- Least-recently used (LRU) is common:
  - Choose the one unused for the longest time:
    - Simple for 2-way, manageable for 4-way, too complicated beyond that.
- Random
  - Oddly, gives about the same performance as LRU for high associativity.

Benefits of Set Associative Caches

- The choice of direct-mapped or set-associative depends on the cost of a miss versus the benefit of a hit.



- Largest gains are in going from direct mapped to 2-way (20%+ reduction in miss rate).

Reducing Cache Miss Rates #2

Use multiple levels of caches

- With advancing technology, we have more than enough room on the die for bigger L1 caches *or* for a second level of cache – normally a ***unified*** L2 cache - it holds both instructions and data - and in some cases even a unified L3 cache.
- For our example, CPI_{ideal} of 2, 100 cycle miss penalty (to main memory) and a 25 cycle miss penalty (to UL2$), 36% load/stores, a 2% (4%) L1 I$ (D$) miss rate, add a 0.5% UL2$ miss rate.

$$CPI_{stalls} = 2 + .02\times25 + .36\times.04\times25 + .005\times100 + .36\times.005\times100 = 3.54$$

(as compared to 5.44 with no L2$)

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are different:
  - Primary cache should focus on ***minimizing hit time*** in support of a shorter clock cycle:
    - Smaller capacity with smaller block sizes.
  - Secondary cache(s) should focus on ***reducing miss rate*** to reduce the penalty of long main memory access times:
    - Larger capacity with larger block sizes.
    - Higher levels of associativity.
- The miss penalty of the L1 cache is significantly reduced by the presence of an L2 cache – so it can be smaller (i.e., faster) but have a higher miss rate.
- For the L2 cache, hit time is less important than miss rate:
  - The L2$ hit time determines L1$'s miss penalty.

Memory Sort Example

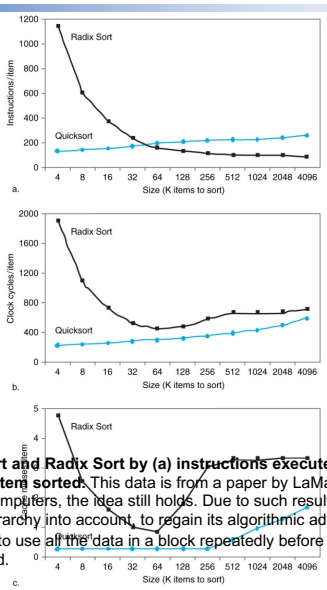


FIGURE 5.18 Comparing Quicksort and Radix Sort by (a) instructions executed per item sorted, (b) time per item sorted, and (c) cache misses per item sorted. This data is from a paper by LaMarca and Ladner [1996]. Although the numbers would change for newer computers, the idea still holds. Due to such results, new versions of Radix Sort have been invented that take memory hierarchy into account, to regain its algorithmic advantages (see Section 5.11). The basic idea of cache optimizations is to use all the data in a block repeatedly before it is replaced on a miss. Copyright © 2009 Elsevier, Inc. All rights reserved.

Two Machines' Cache Parameters

Characteristic	ARM Cortex-A8	Intel Nehalem
L1 cache organization	Split instruction and data caches	Split instruction and data caches
L1 cache size	32 KiB each for instructions/data	32 KiB each for instructions/data per core
L1 cache associativity	4-way (I), 4-way (D) set associative	4-way (I), 8-way (D) set associative
L1 replacement	Random	Approximated LRU
L1 block size	64 bytes	64 bytes
L1 write policy	Write-back, Write-allocate(?)	Write-back, No-write-allocate
L1 hit time (load-use)	1 clock cycle	4 clock cycles, pipelined
L2 cache organization	Unified (instruction and data)	Unified (instruction and data) per core
L2 cache size	128 KiB to 1 MiB	256 KiB (0.25 MiB)
L2 cache associativity	8-way set associative	8-way set associative
L2 replacement	Random(?)	Approximated LRU
L2 block size	64 bytes	64 bytes
L2 write policy	Write-back, Write-allocate (?)	Write-back, Write-allocate
L2 hit time	11 clock cycles	10 clock cycles
L3 cache organization	–	Unified (instruction and data)
L3 cache size	–	8 MiB, shared
L3 cache associativity	–	16-way set associative
L3 replacement	–	Approximated LRU
L3 block size	–	64 bytes
L3 write policy	–	Write-back, Write-allocate
L3 hit time	–	35 clock cycles

Cortex-A8 Data Cache Miss Rates

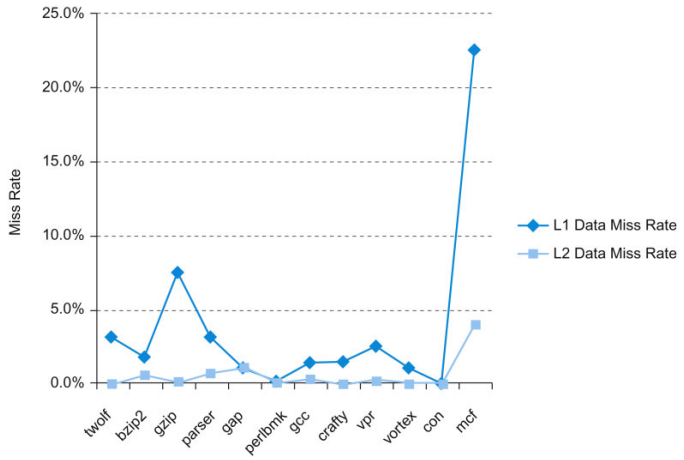


FIGURE 5.45 Data cache miss rates for ARM Cortex-A8 when running Minnespec, a small version of SPEC2000. Applications with larger memory footprints tend to have higher miss rates in both L1 and L2. Note that the L2 rate is the global miss rate; that is, counting all references, including those that hit in L1. (See Elaboration in Section 5.4.) Mcf is known as a cache buster.

Intel Core i7 920 Data Cache Miss Rates

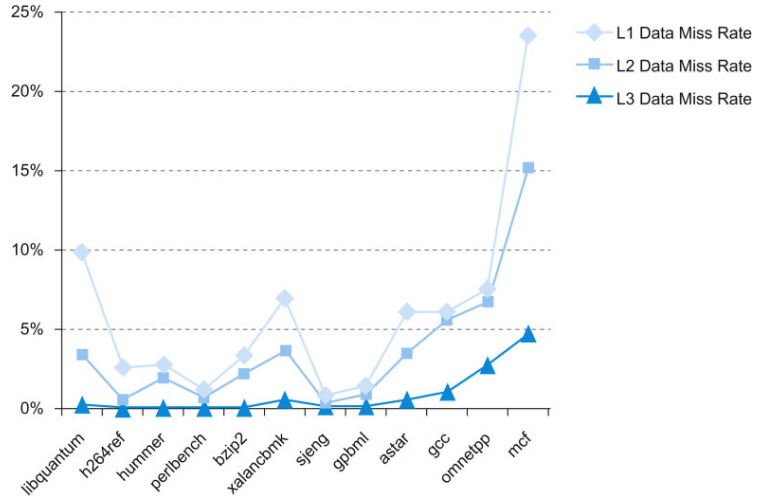


FIGURE 5.47 The L1, L2, and L3 data cache miss rates for the Intel Core i7 920 running the full integer SPECCPU2006 benchmarks.

Summary: Improving Cache Performance

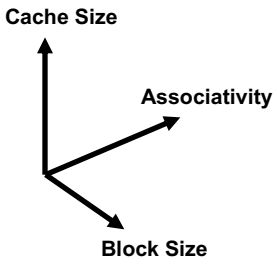
1. Reduce the time to hit in the cache:
   - Smaller cache.
   - Direct mapped cache.
   - Smaller blocks.
   - For writes:
     - No write allocate – no "hit" on cache, just write to write buffer.
     - Write allocate – to avoid two cycles (first check for hit, then write) pipeline writes via a delayed write buffer to cache.
2. Reduce the miss rate:
   - Bigger cache.
   - More flexible placement (increase associativity).
   - Larger blocks (16 to 64 bytes typical).
   - ***Victim cache*** – small buffer holding most recently replaced blocks.

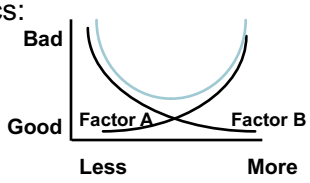
Summary: Improving Cache Performance

3. Reduce the miss penalty:
   - Smaller blocks.
   - Use a write buffer to hold dirty blocks being replaced so you don't have to wait for the write to complete before reading.
   - Check write buffer (and/or victim cache) on read miss – may get lucky.
   - For large blocks fetch, critical word first.
   - Use multiple cache levels – L2 cache is often not tied to CPU clock rate.

Summary: The Cache Design Space

- Several interacting dimensions:
  - Cache size.
  - Block size.
  - Associativity.
  - Replacement policy.
  - Write-through vs. write-back.
  - Write allocation.
- The optimal choice is a compromise
  - Depends on access characteristics:
    - Hard to predict.
  - Depends on technology / cost.
- Simplicity often wins.





Concluding Remarks

- Fast memories are small, large memories are slow:
  - We want fast, large memories. ☹
  - Caching gives this illusion. ☺
- Principle of locality:
  - Programs use a small part of their memory space frequently.
- Memory hierarchy
  - L1 cache ↔ L2 cache ↔ … ↔ DRAM memory ↔ disk