

Improving Cache Performance

Computer Organization

Monday, 26 September 2022

Many slides adapted from:
Computer Organization and Design,
Patterson & Hennessy
5th Edition, © 2014, MK
and from Prof. Mary Jane Irwin, PSU



TÉCNICO LISBOA

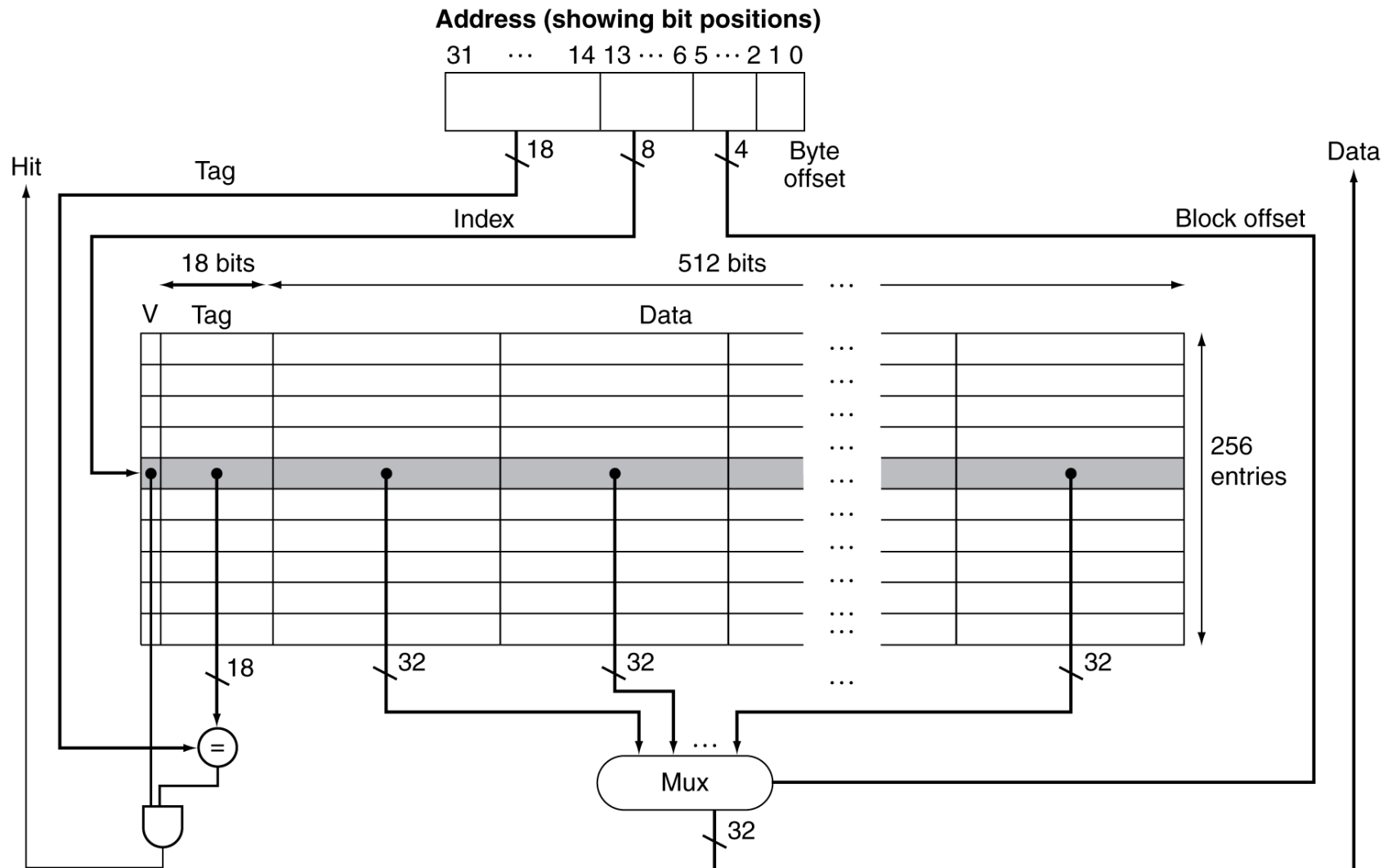
Summary

- Previous Class
 - Memory hierarchy
 - Caches
- Today:
 - Cache Policies
 - Improving Cache Performance

Example: Intrinsic FastMATH

- Embedded MIPS processor
 - 12-stage pipeline
 - Instruction and data access on each cycle
- Split cache:
 - separate I-cache and D-cache
 - Each 16KB: 256 blocks \times 16 words/block
 - D-cache: write-through or write-back
- SPEC2000 miss rates
 - I-cache: 0.4%
 - D-cache: 11.4%
 - Weighted average: 3.2%

Example: Intrinsity FastMATH



Cache Misses

- On cache hit, CPU proceeds normally
- On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access

Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference):
 - First access to a block, “cold” fact of life, not a whole lot you can do about it. If you are going to run “millions” of instruction, compulsory misses are insignificant
 - Solution: increase block size (increases miss penalty; very large blocks could increase miss rate)
- **Capacity**:
 - Cache cannot contain all blocks accessed by the program
 - Solution: increase cache size (may increase access time)
- **Conflict** (collision):
 - Multiple memory locations mapped to the same cache location
 - Solution 1: increase cache size
 - Solution 2: increase associativity (stay tuned) (may increase access time)

Write policies: Write-Through

- On data-write hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through also updates memory
- But makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
 - Holds data waiting to be written to memory
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

Write policies: Write-Back

- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory
 - Can use a write buffer to allow replacing block to be read first

Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
 - Allocate on miss: fetch the block
 - Write around: don't fetch the block
 - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
 - Usually fetch the block

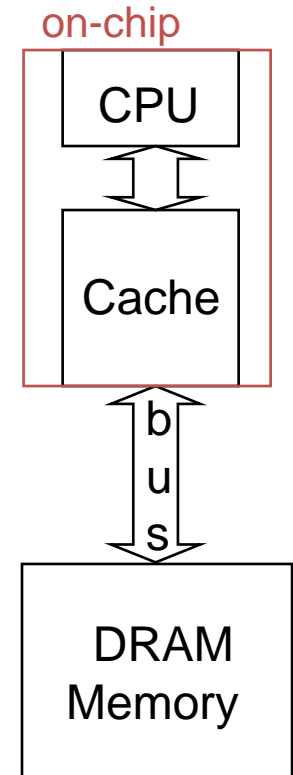
Loading Policies

How should a block be loaded into cache?

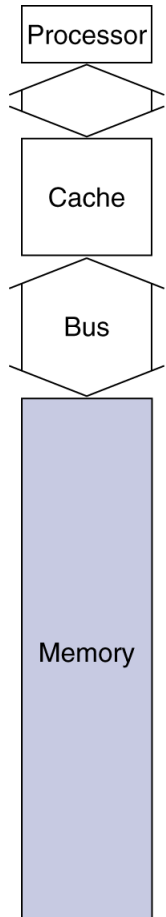
- Blocking
 - the requested word is only sent to the processor after the whole block has been loaded into cache
 - Simpler to implement
 - According to spatial locality, the next access will be to the same block
- Non Blocking
 - Greater impact in caches where the block loading implies several memory accesses.
 - Early Restart
 - fetch the words in normal order, but as soon as the requested word of the block arrives, send it to the processor and let the processor continue execution;
 - Critical Word First
 - request the missed word first from memory and send it to the processor as soon as it arrives; let the processor continue execution while filling the rest of the words in the block.

Check@home: Main Memory Supporting Caches

- Use DRAMs for main memory
 - Fixed width (e.g., 1 word)
 - Connected by fixed-width clocked bus
 - Bus clock is typically slower than CPU clock
 - Example cache block read
 - 1 bus cycle for address transfer
 - 15 bus cycles per DRAM access
 - 1 bus cycle per data transfer
 - For 4-word block, 1-word-wide DRAM
 - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$



Check@home: Increasing Memory Bandwidth

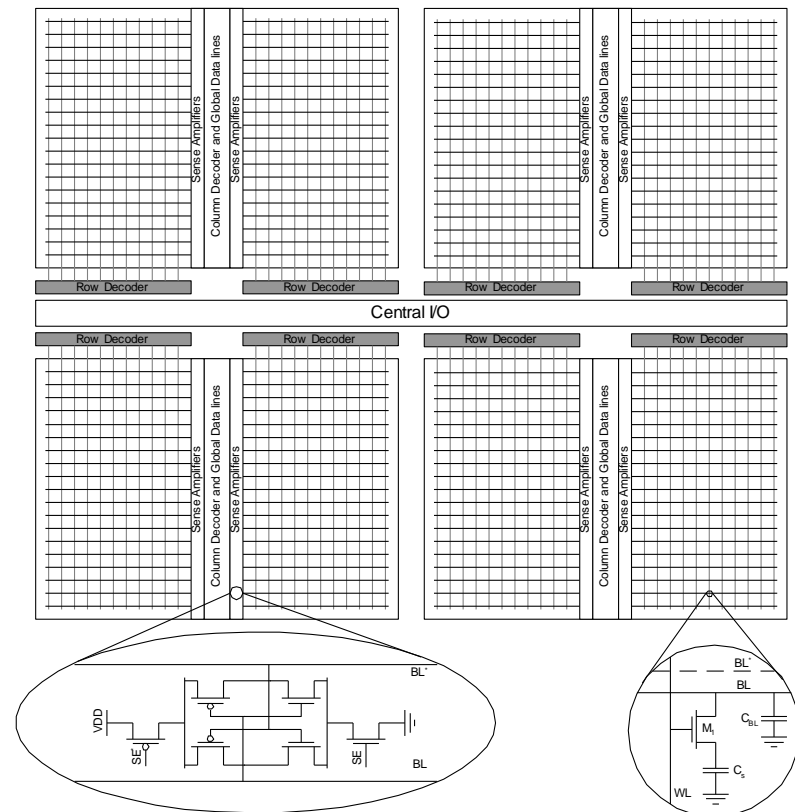


a. One-word-wide memory organization

- 4-word wide memory
 - Miss penalty = $1 + 15 + 1 = 17$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$
- 4-bank interleaved memory
 - Miss penalty = $1 + 15 + 4 \times 1 = 20$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$

Other topics: Advanced DRAM Organization

- Bits in a DRAM are organized as a rectangular array
 - DRAM accesses an entire row
 - Burst mode
 - supply successive words from a row with reduced latency



Other topics: Advanced DRAM Organization

Double Data Rate (DDR) DRAM (2000)

Transfer on rising and falling clock edges

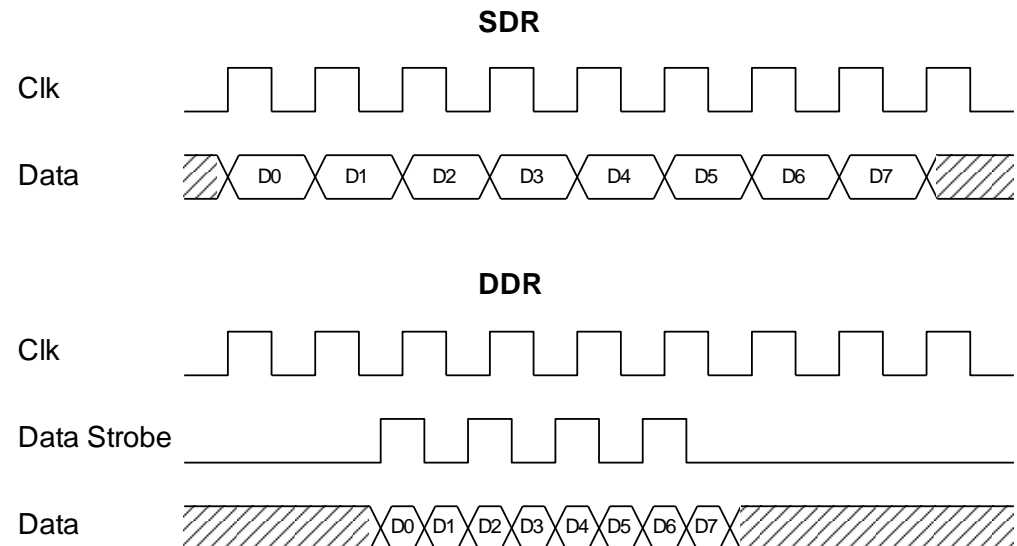
2x Speed without increasing CLK frequency

– Advantages

- Faster Transitions
- Lower power (1.5V vs 2.5V)
- Data strobe
- Bandwidth

– Problems

- Synchronization



Other topics: Advanced DRAM Organization

- Double Data Rate 2 (DDR2) DRAM (2003)
 - Even lower power than DDR (halved, 1.5V)
 - Increased I/O Pins (240-pin DIMM)
 - 4-bit pre-fetch
- Double Data Rate III (DDR3) DRAM (2007)
 - 8 bit internal buffer
 - Higher frequencies
- Double Data Rate IV (DDR4) DRAM (2011)
 - Bank Grouping
 - Lower Power (1.2V)
 - Auto self refresh (based on temperature)

Measuring Cache Performance

- Components of CPU time
 - Program execution cycles
 - Includes cache hit time
 - Memory stall cycles
 - Mainly from cache misses
- Assuming cache hit costs are included as part of the normal CPU execution cycle, then:

$$\begin{aligned}\text{CPU time} &= IC \times CPI \times CC \\ &= IC \times \underbrace{(CPI_{\text{ideal}} + \text{Memory-stall cycles})}_{CPI_{\text{stall}}} \times CC\end{aligned}$$

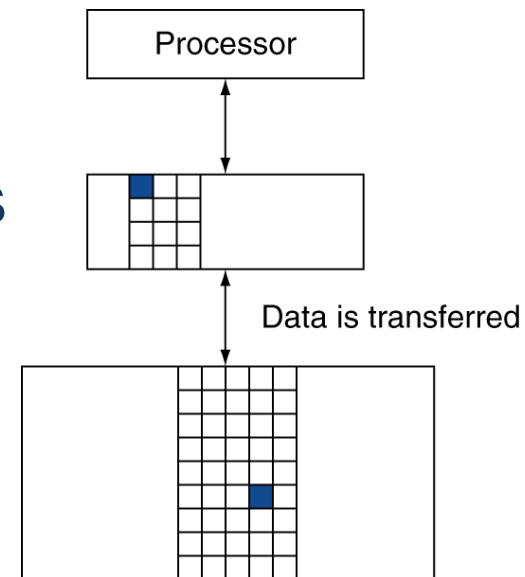
- Assuming a combined read/write miss rate:
Memory-stall cycles =
accesses/program \times miss rate \times miss penalty

Terminology

- **Block** (or line): the minimum unit of information that is present (or not) in a cache
- **Hit Rate**: the fraction of memory accesses found in a level of the memory hierarchy
- **Miss Rate**: the fraction of memory accesses *not* found in a level of the memory hierarchy $\Rightarrow 1 - (\text{Hit Rate})$
- **Miss Penalty**: Time to replace a block in that level with the corresponding block from a lower

Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = $2 + 2 + 1.44 = 5.44$
 - Ideal CPU is $5.44/2 = 2.72$ times faster



Average Access Time

- Hit time is also important for performance

- Average memory access time (AMAT)

$$\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Example

CPU with 1ns clock, hit time = 1 cycle,
miss penalty = 20 cycles, I-cache miss rate = 5%

➤ $\text{AMAT} = 1 + 0.05 \times 20 = 2\text{ns}$

- 2 cycles per instruction

Performance Summary

- When CPU performance increases
 - Miss penalty becomes more significant
- Decreasing based on CPI
 - 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

Reducing Cache Miss Rates: Associative Cache

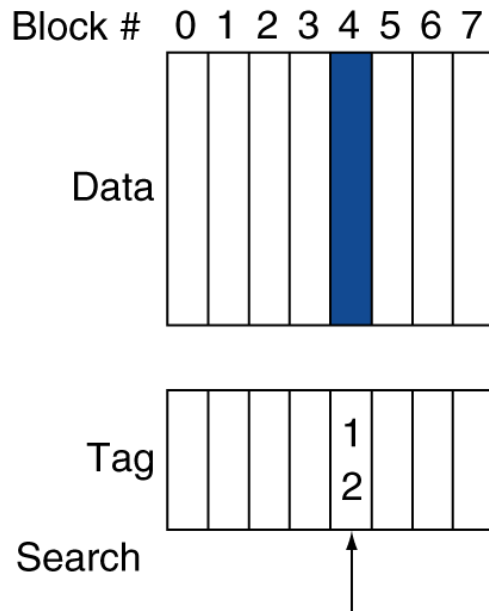
- Allow for more flexible block placement:
 - In a **direct mapped cache** a memory block maps to exactly one cache block
 - At the other extreme, we can allow a memory block to be mapped to *any* cache block – **fully associative cache**
 - A compromise is to divide the cache into **sets** each of which 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 in any way of that set (so there are n choices)

Associative Caches

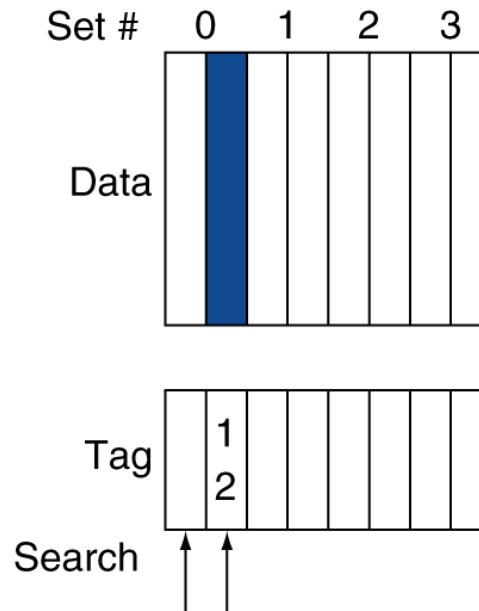
- Fully associative
 - Allow a given block to go in any cache entry
 - Requires all entries to be searched at once
 - Comparator per entry (expensive)
- n -way set associative
 - Each set contains n entries
 - Block number determines which set
 - (Block number) *modulo* (#Sets in cache)
 - Search all entries in a given set at once
 - n comparators (less expensive)

Associative Cache Example

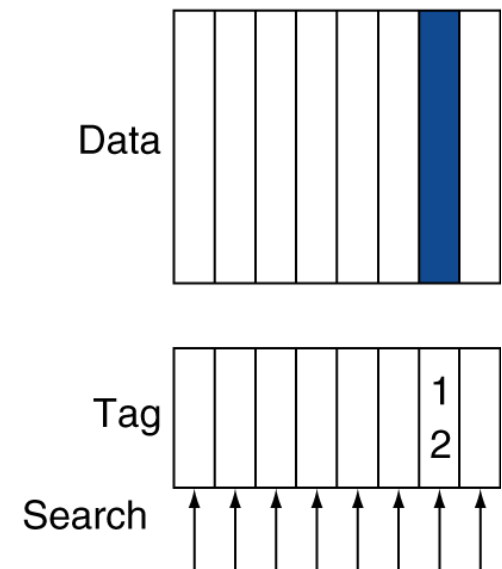
Direct mapped



Set associative



Fully associative



Spectrum of Associativity

For a cache with 8 entries:

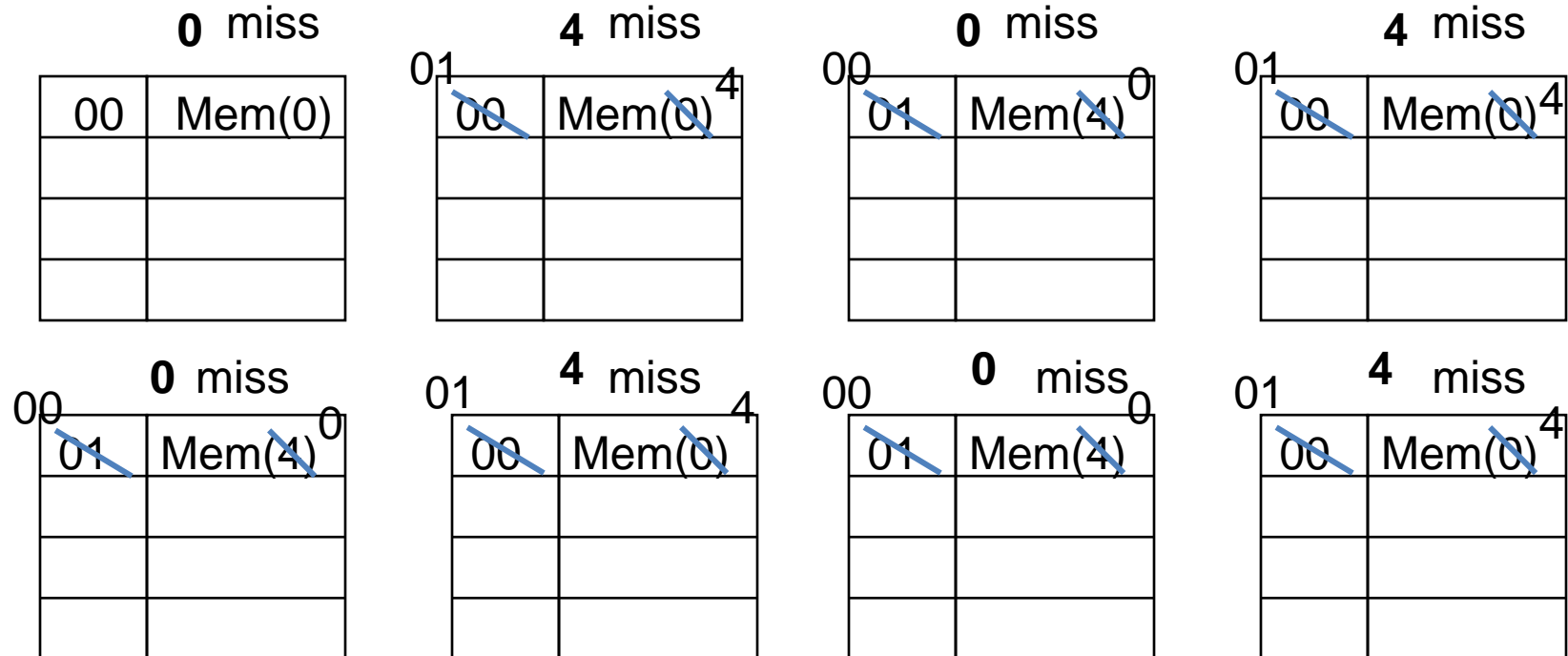
**One-way set associative
(direct mapped)**

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Direct mapped - 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

2-way set associative - 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

0 miss

000	Mem(0)

4 miss

000	Mem(0)
010	Mem(4)

0 hit

000	Mem(0)
010	Mem(4)

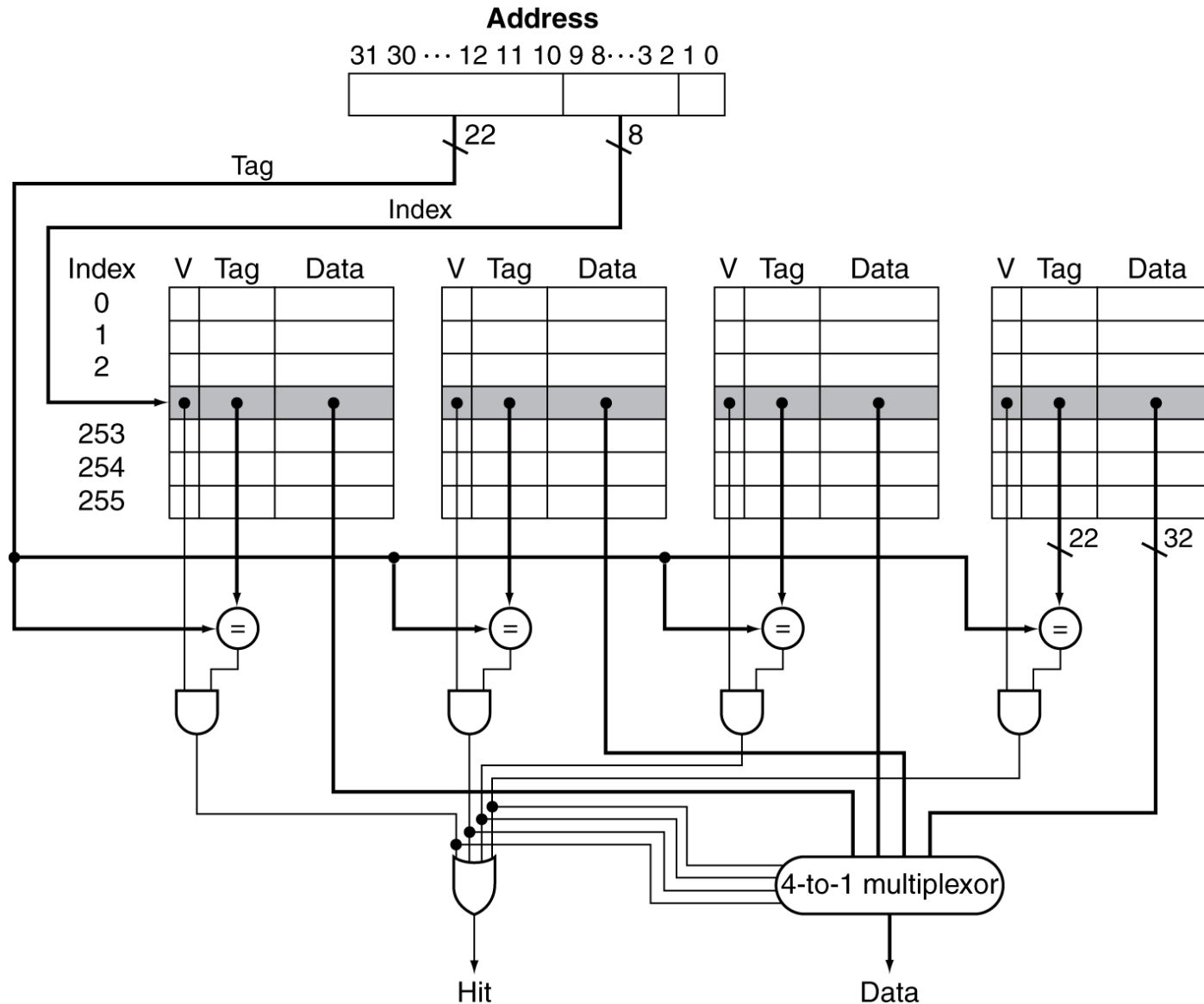
4 hit

000	Mem(0)
010	Mem(4)

□ 8 requests, 2 misses

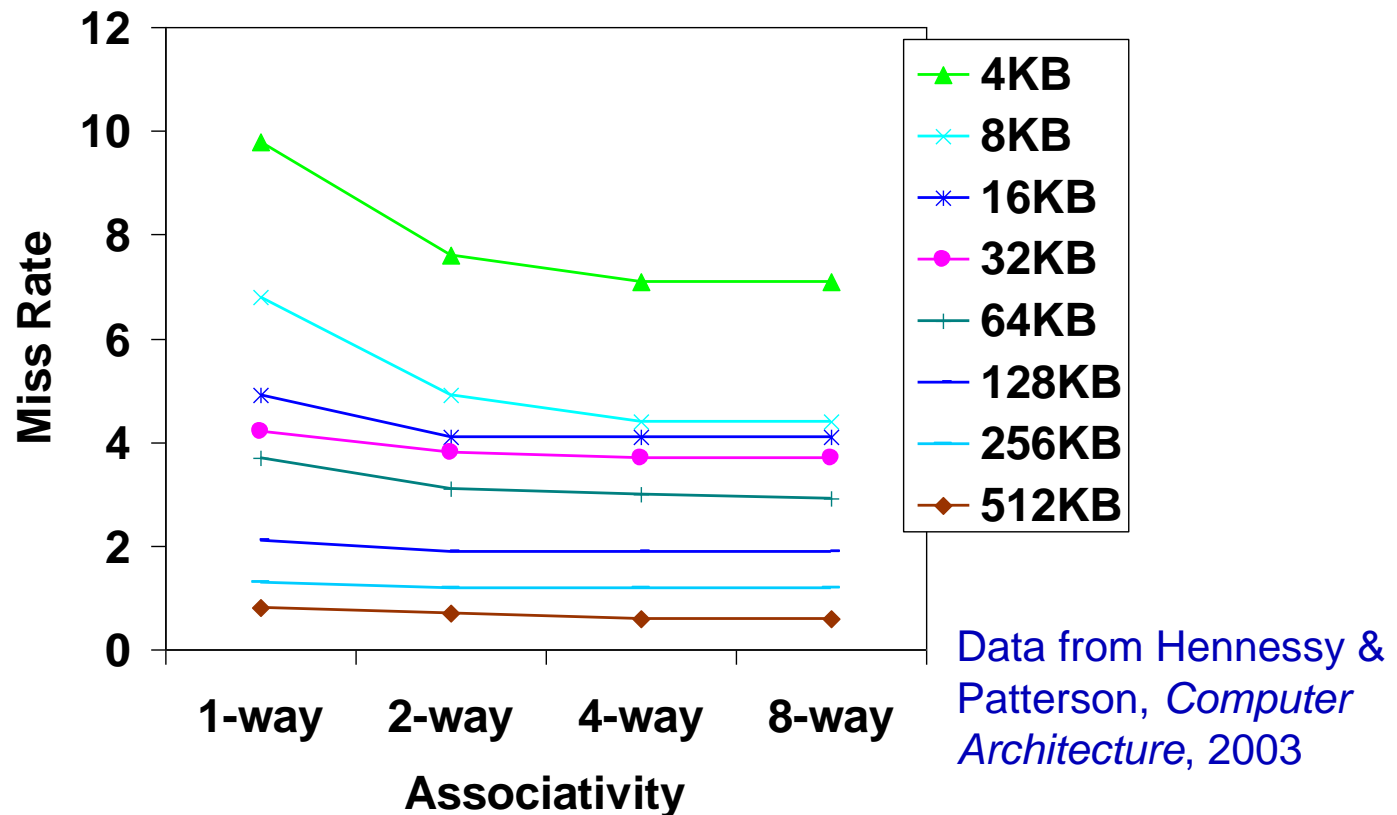
Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

Set Associative Cache Organization



How Much Associativity

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



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

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)
 - Choose the one unused for the longest time
 - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
 - Gives approximately the same performance as LRU for high associativity

Interactions with Advanced CPUs

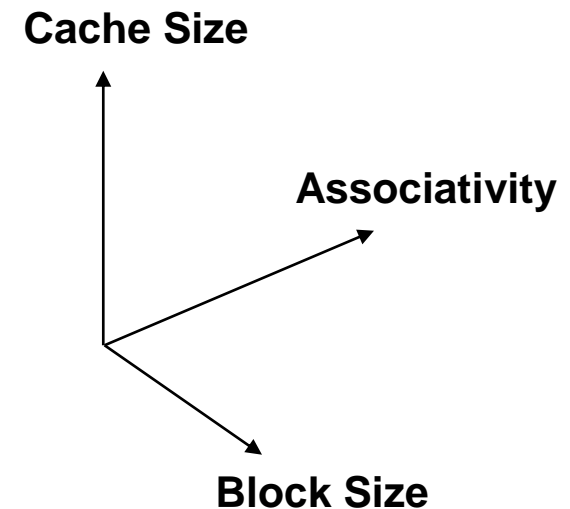
- Out-of-order CPUs can execute instructions during cache miss
 - Pending store stays in load/store unit
 - Dependent instructions wait in reservation stations
 - Independent instructions continue
- Effect of miss depends on program data flow
 - Much harder to analyze
 - Use system simulation

Victim Cache

- Instead of completely discarding each block when it has to be replaced, temporarily keep it in a **victim buffer**.
- Rather than stalling on a subsequent cache miss, the contents of the buffer are checked on a subsequent miss to see if they have the desired data before going to the next lower-level of memory.
 - Small cache (e.g., 4 to 16 positions)
 - Fully associative
 - Particularly efficient for small direct mapped caches (more than 25% reduction of the miss rate in a 4kB cache).

Summary: The Cache Design Space

- Several interacting dimensions
 - cache size
 - block size
 - associativity
 - replacement policy
 - write-through vs write-back
 - write allocation



Next Class

- Optimizing cache usage

Improving Cache Performance

Computer Organization

Monday, 26 September 2022

Many slides adapted from:
Computer Organization and Design,
Patterson & Hennessy
5th Edition, © 2014, MK
and from Prof. Mary Jane Irwin, PSU



TÉCNICO LISBOA