
ECE7995

(6) Improving Cache Performance

[Adapted from Mary Jane Irwin's slides (PSU)]

Measuring Cache Performance

- Assuming cache hit costs are included as part of the normal CPU execution cycle, then

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

IC: instruction count, CPI: cycles per instruction: CC: clock cycle time

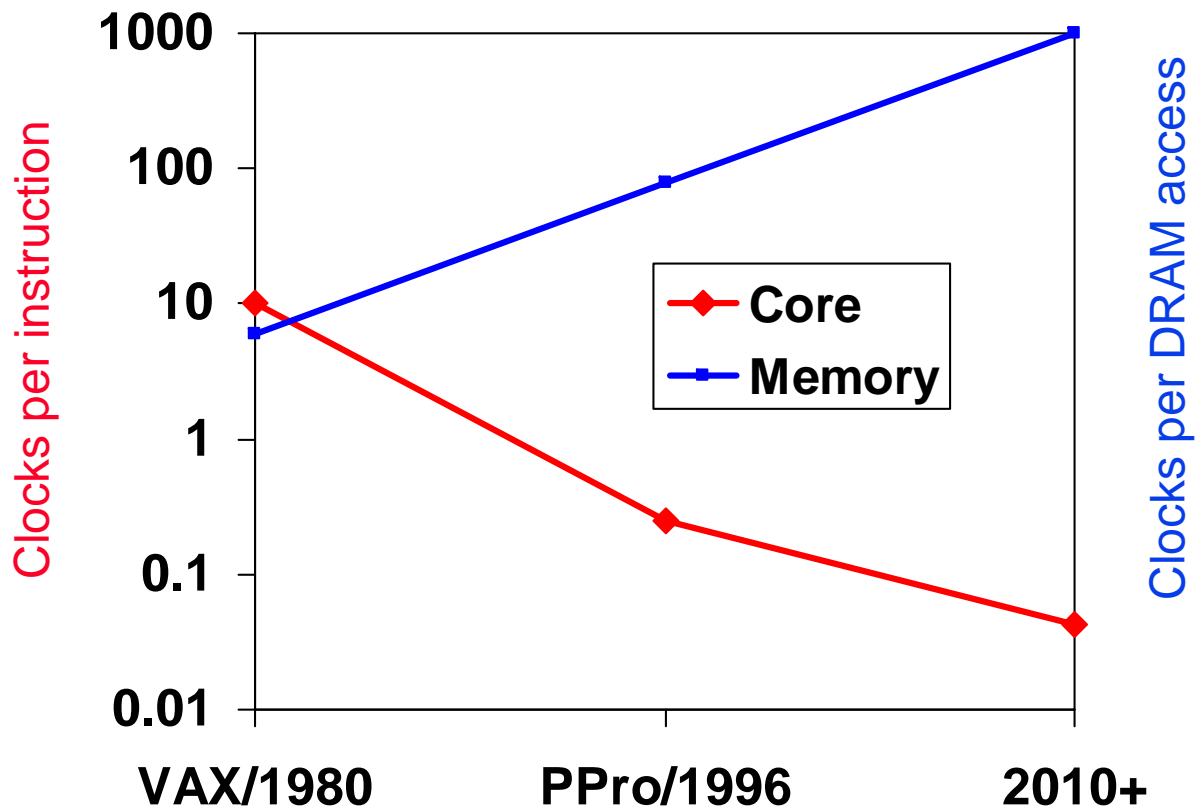
- Memory-stall cycles come from cache misses

$$\text{Memory-stall cycles} = \text{data accesses/program} \times \text{miss rate} \times \text{miss penalty}$$

- Average Memory Access time = Hit Rate x Hit Time + Miss Rate x Miss Penalty

Review: The “Memory Wall”

- ❑ Logic vs DRAM speed gap continues to grow



Impacts of Cache Performance

- ❑ Relative cache penalty increases as processor performance improves (faster clock rate and/or lower CPI)

- The memory speed is unlikely to improve as fast as processor cycle time. When calculating CPI_{stall} , the cache miss penalty is measured in *processor* clock cycles needed to handle a miss
- The lower the CPI_{ideal} , the more pronounced the impact of stalls

- ❑ A processor with a CPI_{ideal} of 2, a 100 cycle miss penalty, 36% load/store instr's, and 2% I\$ and 4% D\$ miss rates

$$\text{Memory-stall cycles} = 2\% \times 100 + 36\% \times 4\% \times 100 = 3.44$$

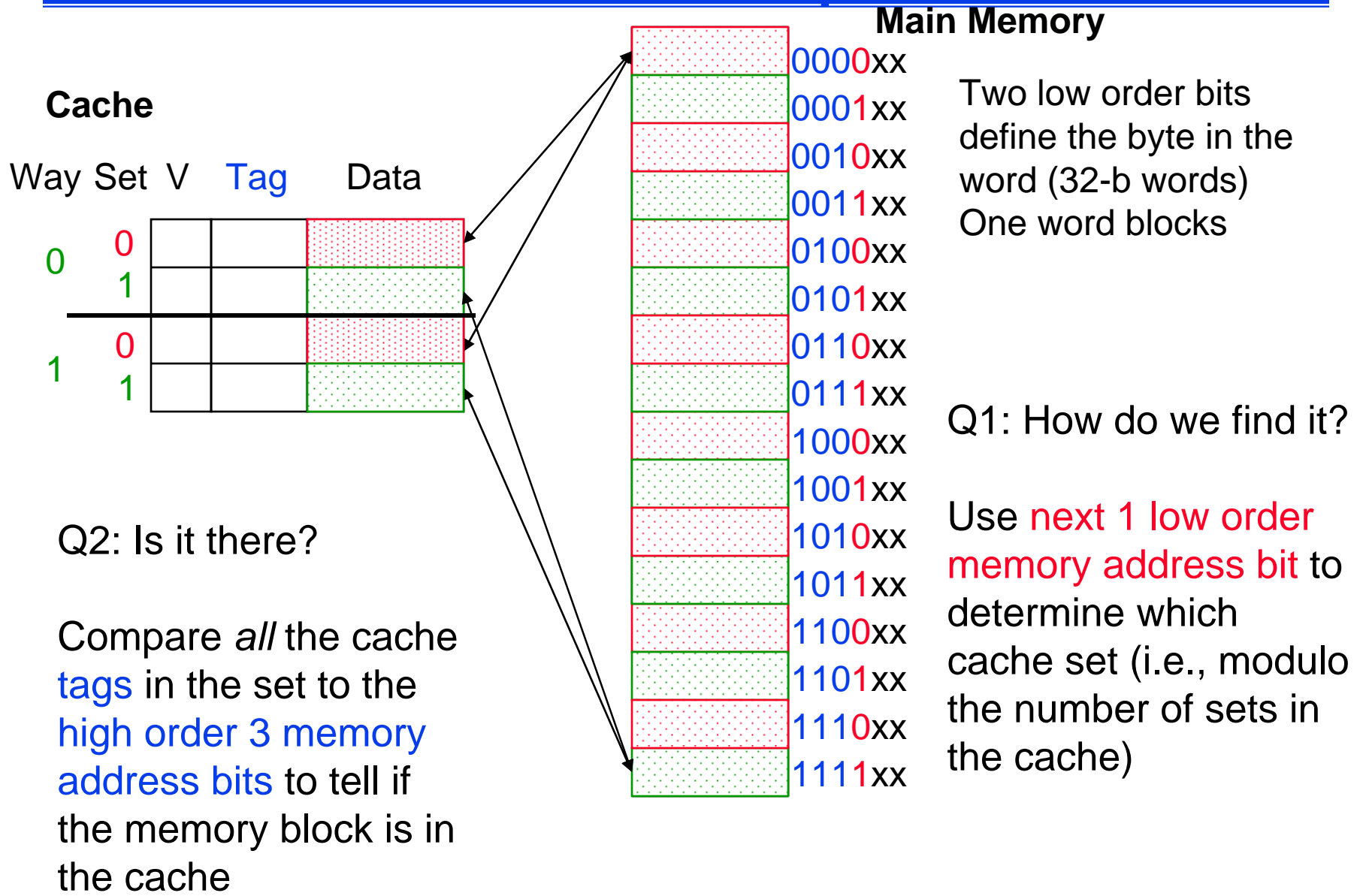
$$\text{So } CPI_{stalls} = 2 + 3.44 = 5.44$$

- ❑ What if the CPI_{ideal} is reduced to 1? 0.5? 0.25?
- ❑ What if the processor clock rate is doubled (doubling the miss penalty)? For miss penalty of 200, memory stall cycles = $2\% \times 200 + 36\% \times 4\% \times 200 = 6.88$

Reducing Cache Miss Rates #1

1. Allow more flexible block placement
 - ❑ In a **direct mapped cache** a memory block maps to exactly one cache block
 - ❑ At the other extreme, 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 in any way of that set (so there are n choices)
 $(\text{block address}) \bmod (\# \text{ sets in the cache})$

Set Associative Cache Example



Another Reference String Mapping

❑ 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)

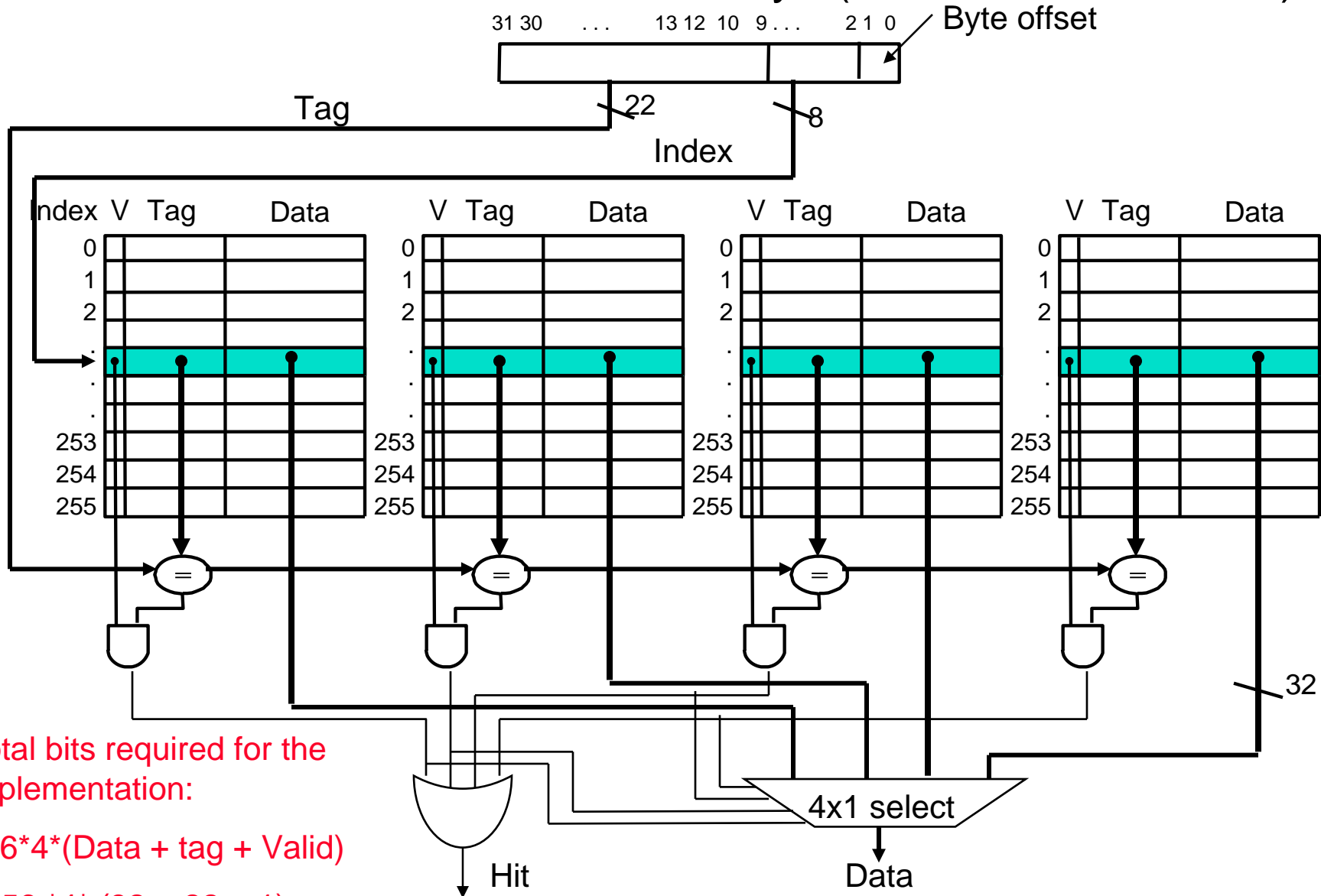
- 8 requests, 2 misses

Try another access string: 0 1 2 3 0 8 11 0 3.

❑ 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!

Four-Way Set Associative Cache

□ $2^8 = 256$ sets each with four ways (each with one block)



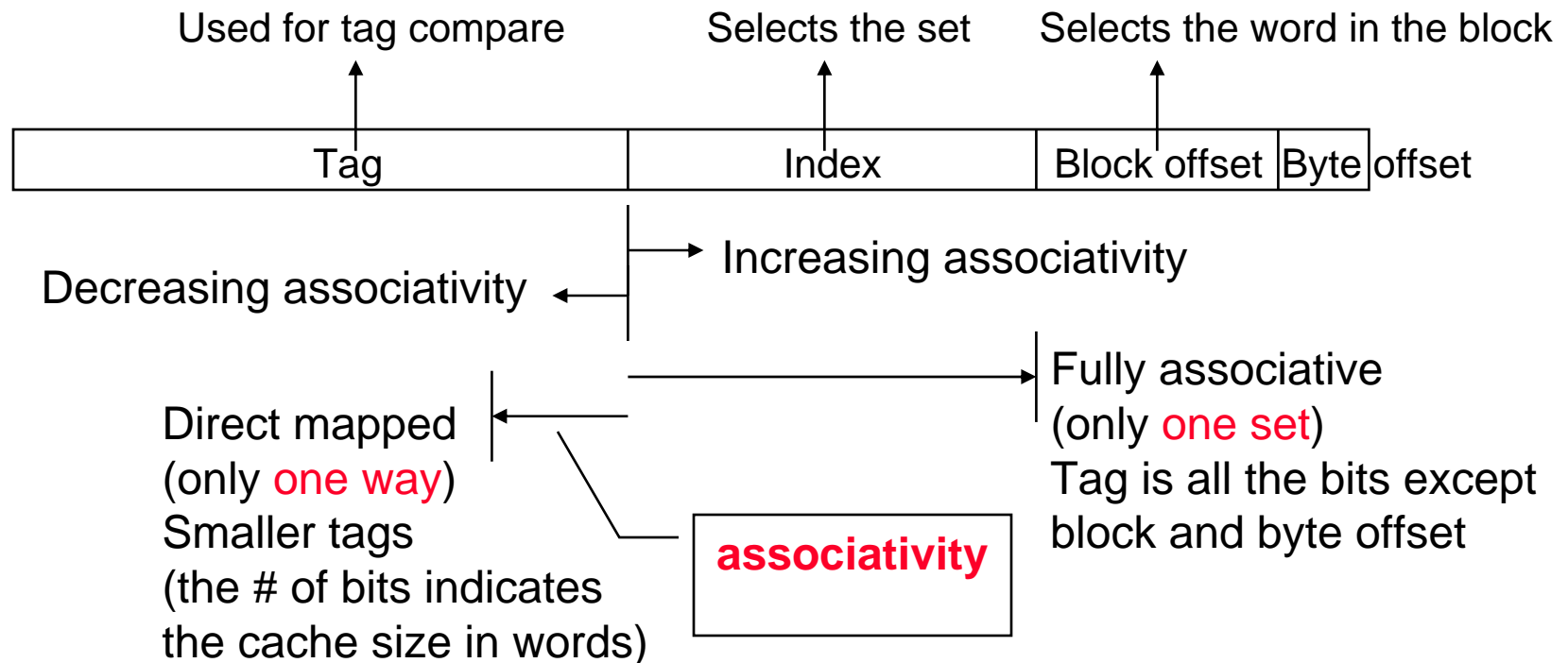
Total bits required for the implementation:

$$256 * 4 * (\text{Data} + \text{tag} + \text{Valid})$$

$$= 256 * 4 * (32 + 22 + 1)$$

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 of 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



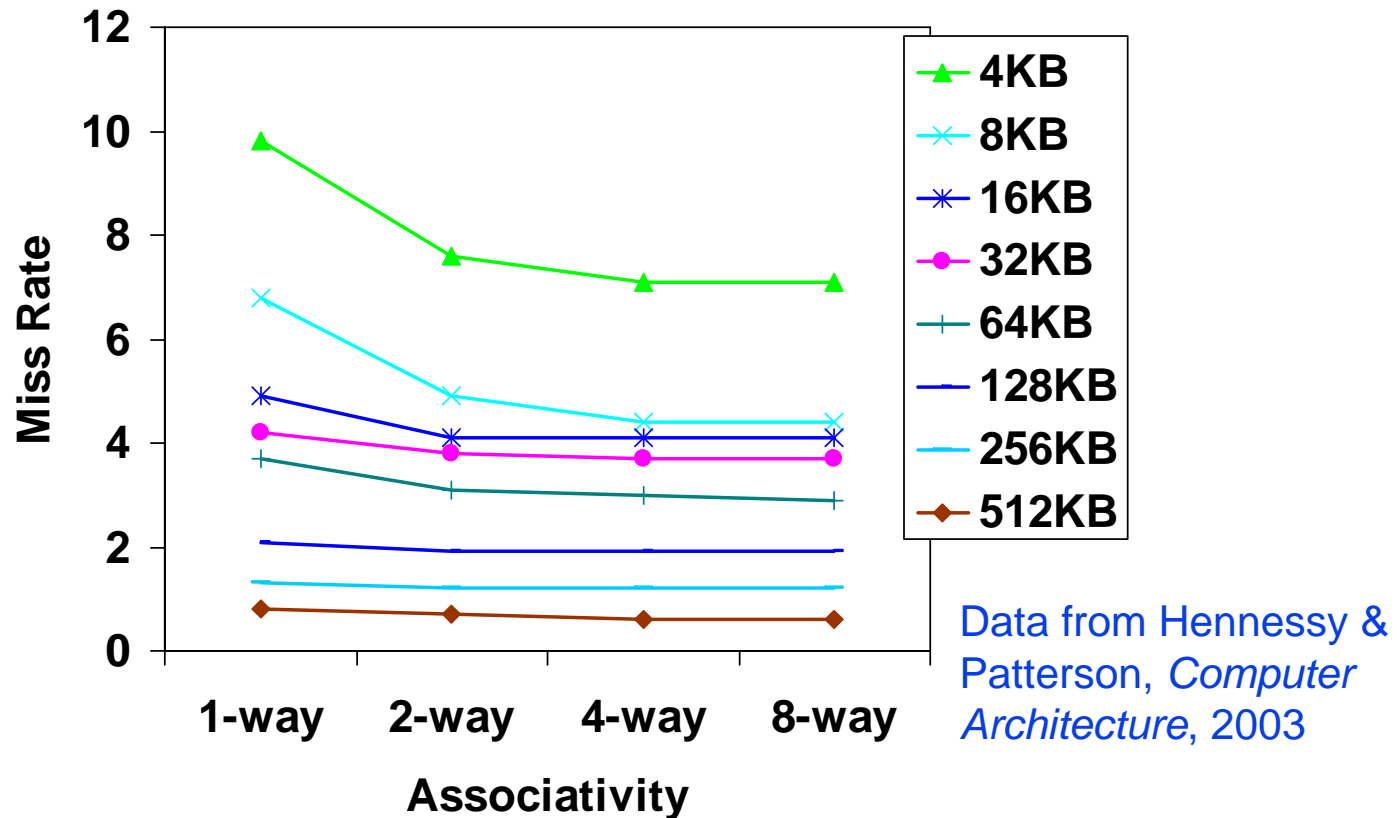
Costs of Set Associative Caches

- ❑ When a miss occurs, which way's block do we pick for replacement?
 - Least Recently Used (LRU): the block replaced is the one that has been unused for the longest time
 - Must have hardware to keep track of when each way's block was used relative to the other blocks in the set
 - For 2-way set associative, takes **one bit per set** → set the bit when a block is referenced (and reset the other way's bit)

- ❑ 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 it's possible to just assume a hit and continue and recover later if it was a miss (remember that cache hit ratio is usually larger than 95%)

Benefits of Set Associative Caches

- ❑ 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)

Reducing Cache Miss Rates #2

2. Use multiple levels of caches
 - ❑ With advancing technology has more than enough room on the die for bigger L1 caches *or* for a second level of caches – normally a **unified** L2 cache (i.e., 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), 36% load/stores, a 2% (4%) L1 I\$ (D\$) miss rate, add a UL2\$ that has a 25 cycle miss penalty and a 0.5% miss rate (over program's I & D accesses)

$$CPI_{stalls} = 2 + .02 \times 25 + .36 \times .04 \times 25 + .005 \times 100 + .36 \times .005 \times 100 = 3.54$$

(as compared to 5.44 with no L2\$)

Multilevel Cache Design Considerations

- ❑ Design considerations for L1 and L2 caches are very different
 - Primary cache should focus on **minimizing hit time** in support of a shorter clock cycle
 - Smaller with smaller block sizes, small-associativity
 - Secondary cache(s) should focus on **reducing miss rate** to reduce the penalty of long main memory access times
 - Larger with larger block sizes

- ❑ 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
 - L2\$ local miss rate \gg the global miss rate

Key Cache Design Parameters

	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates (global for L2)	2% to 5%	0.1% to 2%

Two Machines' Cache Parameters

	Intel P4	AMD Opteron
L1 organization	Split I\$ and D\$	Split I\$ and D\$
L1 cache size	8KB for D\$, 96KB for trace cache (~I\$)	64KB for each of I\$ and D\$
L1 block size	64 bytes	64 bytes
L1 associativity	4-way set assoc.	2-way set assoc.
L1 replacement	~ LRU	LRU
L1 write policy	write-through	write-back
L2 organization	Unified	Unified
L2 cache size	512KB	1024KB (1MB)
L2 block size	128 bytes	64 bytes
L2 associativity	8-way set assoc.	16-way set assoc.
L2 replacement	~LRU	~LRU
L2 write policy	write-back	write-back

4 Questions for the Memory Hierarchy

- ❑ Q1: Where can a block be placed in the upper level?
(Block placement)
- ❑ Q2: How is a block found if it is in the upper level?
(Block identification)
- ❑ Q3: Which block should be replaced on a miss?
(Block replacement)
- ❑ Q4: What happens on a write?
(Write strategy)

Q1&Q2: Where can a block be placed/found?

	# of sets	Blocks per set
Direct mapped	# of blocks in cache	1
Set associative	(# of blocks in cache)/ associativity	Associativity (typically 2 to 16)
Fully associative	1	# of blocks in cache

	Location method	# of comparisons
Direct mapped	Index	1
Set associative	Index the set; compare set's tags	Degree of associativity
Fully associative	Compare all blocks tags	# of blocks

Q3: Which block should be replaced on a miss?

- ❑ Easy for direct mapped – only one choice
- ❑ Set associative or fully associative
 - Random
 - LRU (Least Recently Used)

- ❑ For a 2-way set associative cache, random replacement has a miss rate about 1.1 times higher than LRU.

- ❑ LRU is too costly to implement for high levels of associativity (> 4-way) since tracking the usage information is costly

Q4: What happens on a write?

- Write-through – The information is written to both the block in the cache and to the block in the next lower level of the memory hierarchy
 - Write-through is always combined with a write buffer so write waits to lower level memory can be eliminated (as long as the write buffer doesn't fill)
- Write-back – The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.
 - Need a dirty bit to keep track of whether the block is clean or dirty
- Pros and cons of each?
 - Write-through: read misses don't result in writes (so are simpler and cheaper)
 - Write-back: repeated writes require only one write to lower level

Improving Cache Performance

1. Reduce the time to hit in the cache

- smaller cache
- direct mapped cache
- smaller blocks

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 discarded blocks

Improving Cache Performance (cont'd)

3. Reduce the miss penalty

- smaller blocks
- use a write buffer to hold dirty blocks being replaced so 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 not tied to CPU clock rate
- faster backing store/improved memory bandwidth
 - wider buses
 - memory interleaving, page mode DRAMs

Summary: The Cache Design Space

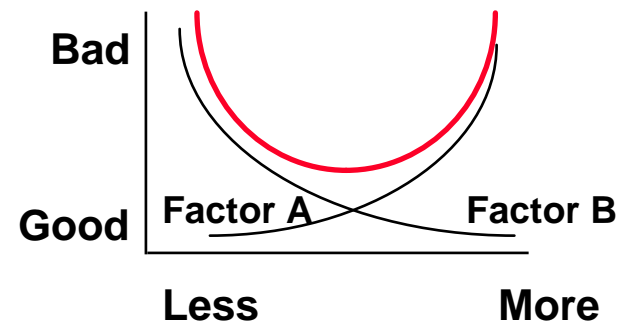
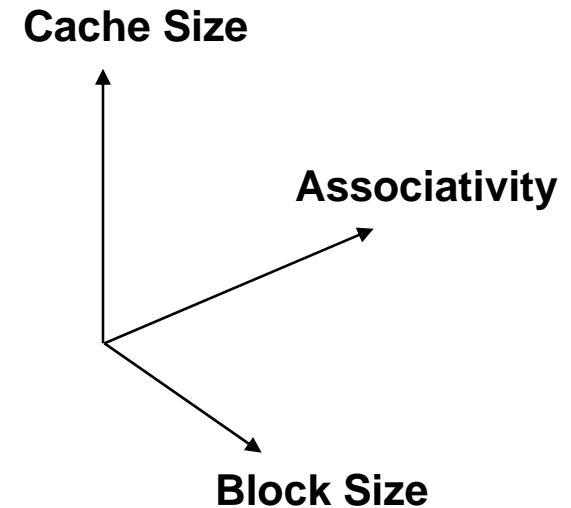
❑ Several interacting dimensions

- cache size
- block size
- associativity
- replacement policy
- write-through vs write-back

❑ The optimal choice is a compromise

- depends on access characteristics
 - workload
 - use (I-cache, D-cache, TLB)
- depends on technology / cost

❑ Simplicity often wins



Practice Questions:

- Here is a series of an address references given as word address: 2, 3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11. Assuming a direct-mapped cache with 16 one-word blocks that is initially empty, label each reference in the list as a hit or miss and show the final contents of the cache.

2-miss, 3-miss, 11-miss, 16-miss, 21-miss, 13-miss, 64-miss, 48-miss, 19-miss, 11-hit, 3-miss, 22-miss, 4-miss, 27-miss, 6-miss, 11-set.

Cache set	Address
0000	48
0001	
0010	2
0011	3
0100	4
0101	21
0110	6
0111	
1000	
1001	27
1010	
1011	11
1100	
1101	13
1110	
1111	

Practice Questions:

- Using the same series of references: 2, 3, 11, 16, 21, 13, 64, 48, 19, 11, 3, 22, 4, 27, 6, and 11, show the hits and misses and final cache contents for a directly-mapped cache with four-word blocks and a total size of 16 words.

2-miss, 3-hit, 11-miss, 16-miss, 21-miss, 13-miss, 64-miss, 48-miss,
19-miss, 11-hit, 3-miss, 22-hit, 4-miss, 27-miss, 6-hit, 11-miss

Cache set	Address
00	[0, 1, 2, 3]
01	[4, 5, 6, 7]
10	[8, 9, 10, 11]
11	[12, 13, 14, 15]

Practice Questions:

- Suppose a computer's address size is k bits (using byte addressing), the cache size is S bytes, the block size is B bytes, and the cache is A -way set associative. Assume that B is a power of 2, so $B = 2^b$. Figure out what the following quantities are in terms of S , B , A , b , and k : the number of sets in the cache, the number of index bits in the address, and the number of bits needed to implement the cache.

Address size:	k bits
Cache size:	S bytes/cache
Block size:	$B = 2^b$ bytes/block
Associativity:	A blocks/set

Number of sets in the cache:

$$\begin{aligned}\text{Sets/cache} &= \frac{(\text{Bytes/cache})}{(\text{Blocks/set}) \times (\text{Bytes/block})} \\ &= \frac{S}{AB}\end{aligned}$$

Number of address bits needed to index a particular set of the cache:

$$\begin{aligned}\text{Cache set index bits} &= \log_2 (\text{Sets/cache}) \\ &= \log_2 \left(\frac{S}{AB} \right) \\ &= \log_2 \left(\frac{S}{A} \right) - b\end{aligned}$$

Practice Questions:

Number of bits needed to implement the cache:

Tag address bits/block = (Total address bits) – (Cache set index bits)
– (Block offset bits)

$$= k - \left(\log_2 \left(\frac{S}{A} \right) - b \right) - b$$
$$= k - \log_2 \left(\frac{S}{A} \right)$$

Number of bits needed to implement the cache = sets/cache \times associativity \times (data + tag + valid):

$$= \frac{S}{AB} \times A \times \left(8 \times B + k - \log_2 \left(\frac{S}{A} \right) + 1 \right)$$
$$= \frac{S}{B} \times \left(8B + k - \log_2 \left(\frac{S}{A} \right) + 1 \right)$$
