EE 4683/5683: COMPUTER ARCHITECTURE

Lecture 6A: Cache Design

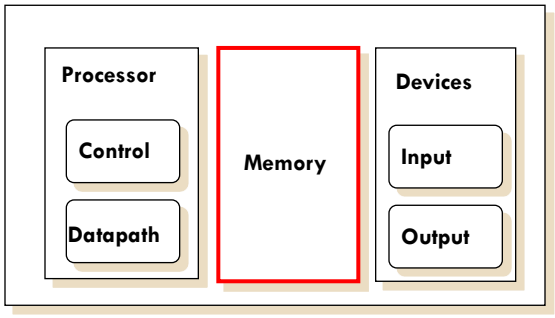
Avinash Kodi, kodi@ohio.edu

Agenda

- Review: Memory Hierarchy
- Review: Cache Organization
  - Direct-mapped
  - Set- Associative
  - Fully-Associative

Major Components of a Computer

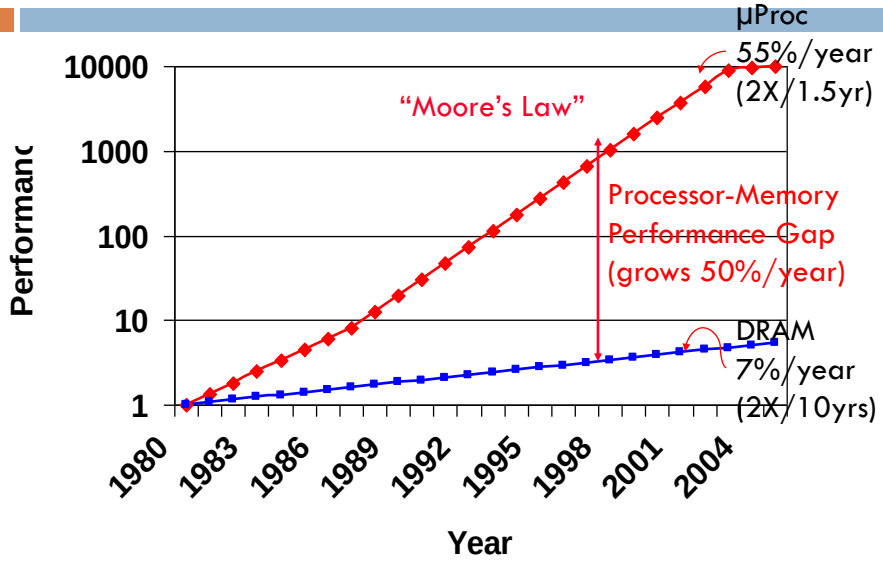
3



- Processor issues address (and data for writes)
- Memory returns data (or acknowledgement for writes)

Processor-Memory Performance Gap

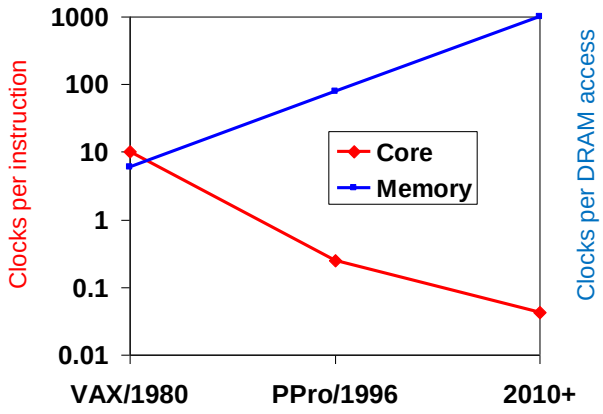
4



The "Memory Wall"

5

- Logic vs DRAM speed gap continues to grow

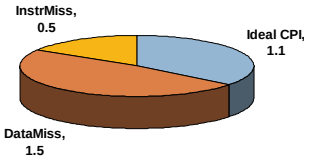


Impact of Memory Design on Performance: Example

6

- Suppose a processor executes at
  - ideal CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control

  and that 10% of data memory operations miss with a 50 cycle miss penalty



- CPI = ideal CPI + average stalls per instruction = 1.1(cycle) + (0.30 (datamemops/instr) x 0.10 (miss/datamemop) x 50 (cycle/miss))
  = 1.1 cycle + 1.5 cycle = 2.6
- A 1% instruction miss rate would add an *additional* 0.5 to the CPI!

Memory Hierarchy Goals

7

- We want lots of memory and fast access to data
  - Choices – SRAM, DRAM, Disk, Tapes
- Option 1: Fast SRAMs?
  - SRAM costs $10 for Megabyte & access time is 5nsec
- Option 2: Slow DRAMs?
  - DRAMs cost $0.06 per Megabyte & access time is 50 nsec
- Option 3: Slower Disks?
  - Disk storage costs $0.0004 per Megabyte & access time is 3,000,000 nsec
- Option 4: Magnetic Tapes?
  - Disk storage costs $0.0002 per Megabyte & access time is 100,000,000,000 nsec

Memory Hierarchy Goals

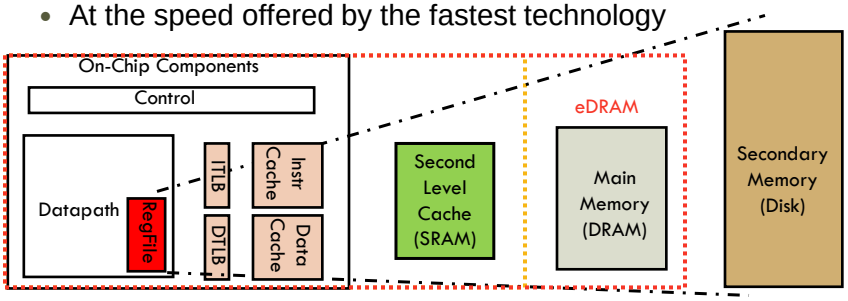
8

- How do you create a memory with an illusion of being large, cheap, and fast (most of the time)?
  - With hierarchy
  - With parallelism
- Use a small SRAM (Cache)
  - small means fast and cheap!
- Use a larger amount of DRAM (Main Memory)
- Use a really big amount of disk storage (disks becoming cheaper)
- Use tapes or optical disks to backup disks

A Typical Memory Hierarchy

9

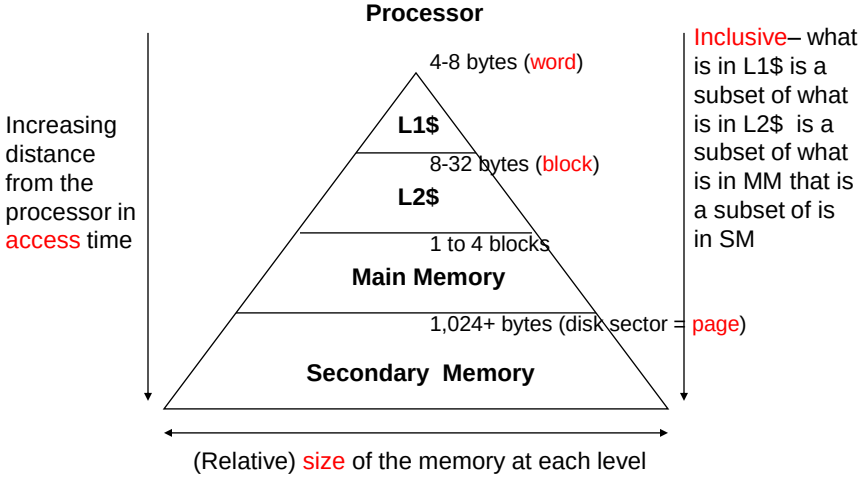
- By taking advantage of the principle of locality
  - Can present the user with as much memory as is available in the cheapest technology
  - At the speed offered by the fastest technology



Speed (%cycles):	½'s	1's	10's	100's	1,000's
Size (bytes):	100's	K's	1-2M's	1-2G's	G's to T's
Cost:	highest				lowest

Characteristics of the Memory Hierarchy

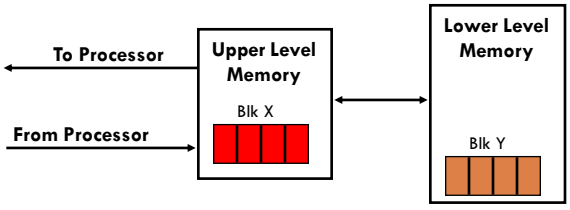
10



The Memory Hierarchy: Why Does it Work?

11

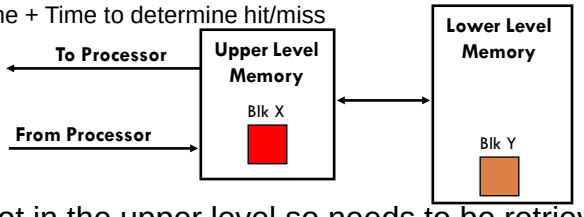
- Temporal Locality (Locality in Time):
  - ⇒ Keep most recently accessed data items closer to the processor
- Spatial Locality (Locality in Space):
  - ⇒ Move blocks consisting of contiguous words to the upper levels



The Memory Hierarchy: Terminology

12

- Hit: data is in some block in the upper level (Blk X)
  - Hit Rate: the fraction of memory accesses found in the upper level
  - Hit Time: Time to access the upper level which consists of RAM access time + Time to determine hit/miss



- Miss: data is not in the upper level so needs to be retrieve from a block in the lower level (Blk Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in the upper level + Time to deliver the block the processor
  - Hit Time << Miss Penalty

How is the Hierarchy Managed?

13

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - virtual to physical address mapping assisted by the hardware (TLB)
  - by the programmer (files)

Cache

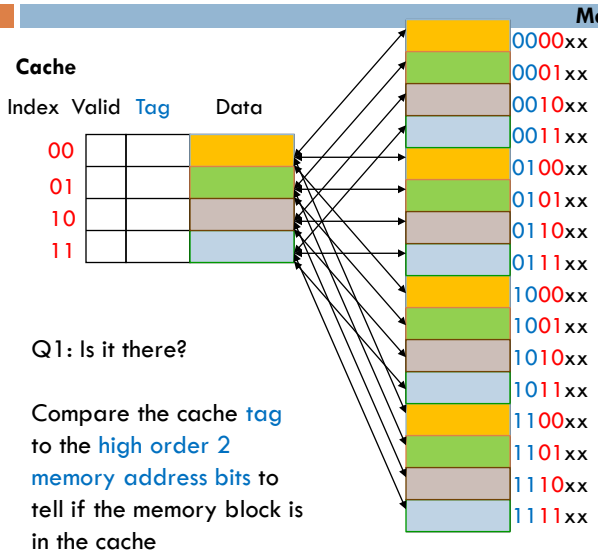
14

- Two questions to answer (in hardware):
  - Q1: How do we know if a data item is in the cache?
  - Q2: If it is, how do we find it?
- Direct mapped
  - For each item of data at the lower level, there is exactly one location in the cache where it might be - so lots of items at the lower level must share locations in the upper level
  - Address mapping:

    (block address) modulo (# of blocks in the cache)
  - First consider block sizes of one word

Caching: A Simple First Example

15



Two low order bits define the byte in the word (32-b words)

Q1: Is it there?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

Q2: How do we find it?

Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

(block address) modulo (# of blocks in the cache)

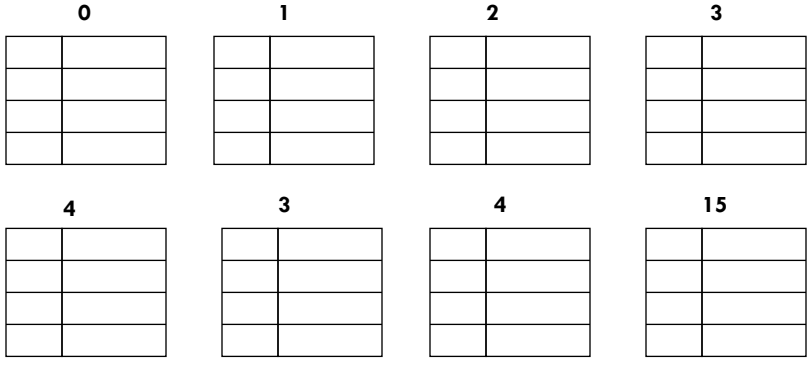
Direct Mapped Cache

16

- Consider the main memory word reference string

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

0 1 2 3 4 3 4 14



Direct Mapped Cache

17

- Consider the main memory word reference string

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

0 1 2 3 4 3 4 15

0 miss

00	Mem(0)

1 miss

00	Mem(0)
00	Mem(1)

2 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)

3 miss

00	Mem(0)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 miss

01 ~~00~~ → 4

~~00~~	~~Mem(0)~~
00	Mem(1)
00	Mem(2)
00	Mem(3)

3 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

4 hit

01	Mem(4)
00	Mem(1)
00	Mem(2)
00	Mem(3)

15 miss

11 ~~00~~ → 15

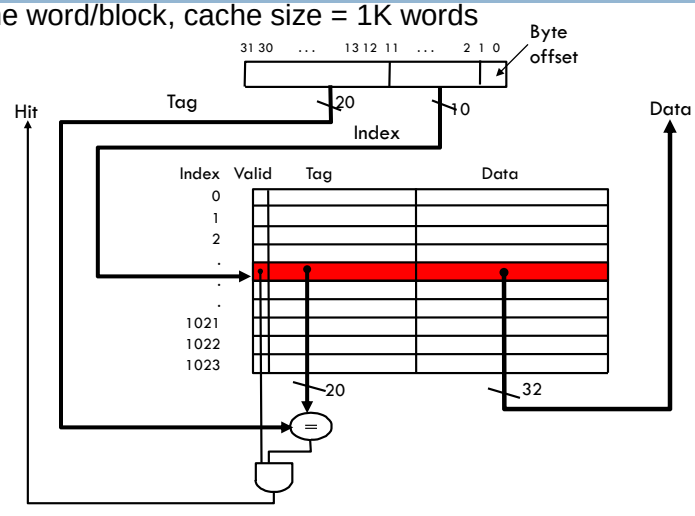
01	Mem(4)
00	Mem(1)
00	Mem(2)
~~00~~	~~Mem(3)~~

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

18

- One word/block, cache size = 1K words



What kind of locality are we taking advantage of?

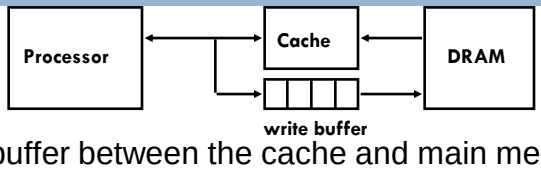
Handling Cache Hits

19

- Read hits (I$ and D$)
  - this is what we want!
- Write hits (D$ only)
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache contents to the next level in the memory hierarchy when that cache block is "evicted")
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through) so don't need a dirty bit
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a write buffer, so only have to stall if the write buffer is full

Write Buffer for Write-Through Caching

20



- Write buffer between the cache and main memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: writes contents of the write buffer to memory
- The write buffer is just a FIFO
  - Typical number of entries: 4
  - Works fine if store frequency (w.r.t. time) << 1 / DRAM write cycle
- Memory system designer's nightmare
  - When the store frequency (w.r.t. time) → 1 / DRAM write cycle leading to write buffer saturation
    - One solution is to use a write-back cache; another is to use an L2 cache

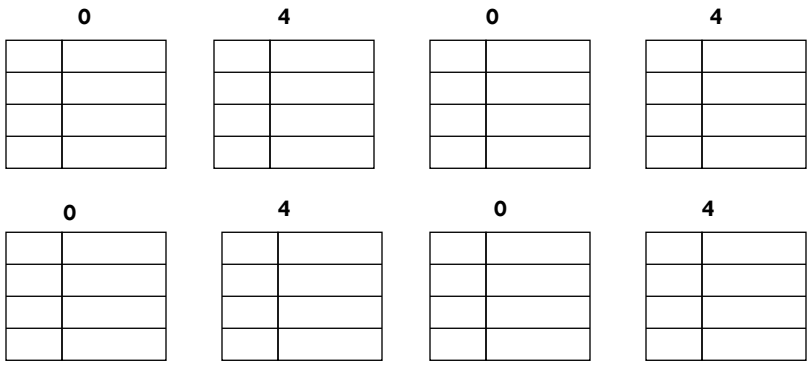
Another Reference String Mapping

21

- 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



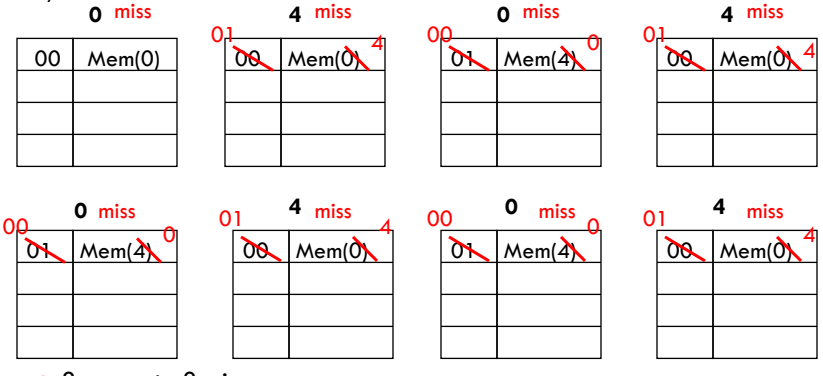
Another Reference String Mapping

22

- 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

Sources of Cache Misses

23

- 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
- Conflict (collision):
  - Multiple memory locations mapped to the same cache location
  - Solution 1: increase cache size
  - Solution 2: increase associativity
- Capacity:
  - Cache cannot contain all blocks accessed by the program
  - Solution: increase cache size

Handling Cache Misses

24

- Read misses (I$ and D$)
  - stall the entire pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- Write misses (D$ only)
  1. stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume

  or (normally used in write-back caches)

  2. Write allocate – just write the word into the cache updating both the tag and data, no need to check for cache hit, no need to stall

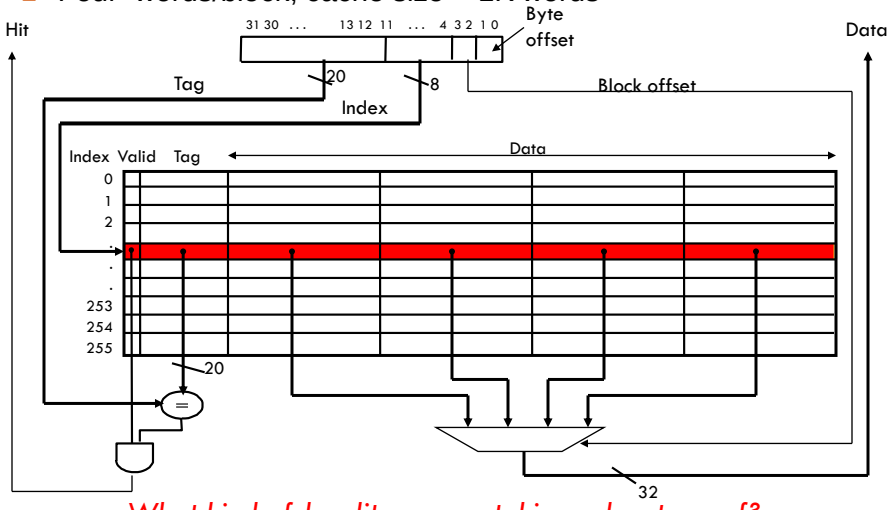
  or (normally used in write-through caches with a write buffer)

  3. No-write allocate – skip the cache write and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn’t full; must invalidate the cache block since it will be inconsistent (now holding stale data)

Multiword Block Direct Mapped Cache

25

- Four words/block, cache size = 1K words



What kind of locality are we taking advantage of?

Taking Advantage of Spatial Locality

27

- Let cache block hold more than one word

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

0 1 2 3 4 3 4 15

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

~~00~~ 01	~~Mem(1)~~	~~Mem(0)~~
00	Mem(3)	Mem(2)

5 4

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

11 01	Mem(5)	Mem(4)
~~00~~	~~Mem(3)~~	~~Mem(2)~~

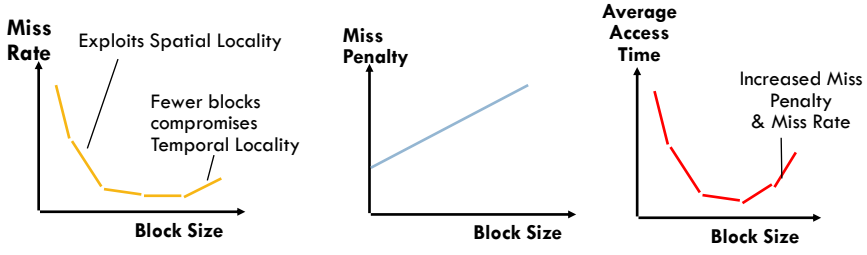
15 14

- 8 requests, 4 misses

Block Size Tradeoff

28

- Larger block sizes take advantage of spatial locality but
  - If the block size is too big relative to the cache size, the miss rate will go up
  - Larger block size means larger miss penalty
    - Latency to first word in block + transfer time for remaining words



Average Memory Access Time

= Hit Time + Miss Penalty x Miss Rate

Reducing Cache Miss Rates #1

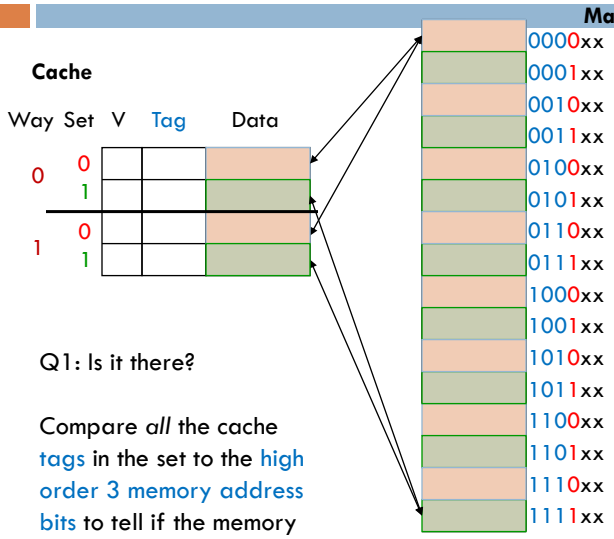
1. Allow more flexible block placement

- 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)

(block address) modulo (# sets in the cache)

Set Associative Cache Example



Two low order bits define the byte in the word (32-b words) One word blocks

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)

Another Reference String Mapping

- Consider the main memory word reference string

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

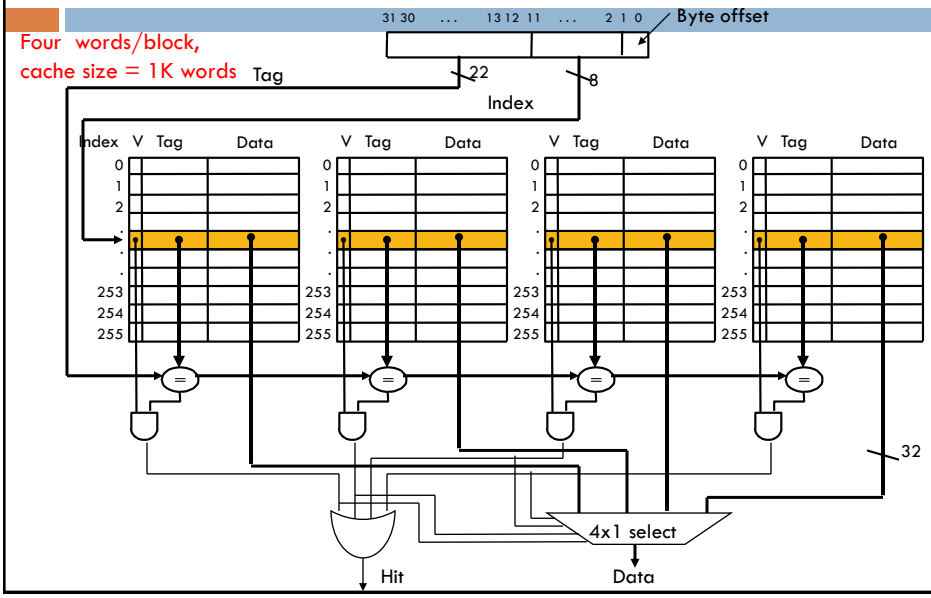
0 4 0 4 0 4 0 4

0 miss		4 miss		0 hit		4 hit	
000	Mem(0)	000	Mem(0)	000	Mem(0)	000	Mem(0)
		010	Mem(4)	010	Mem(4)	010	Mem(4)

- 8 requests, 2 misses

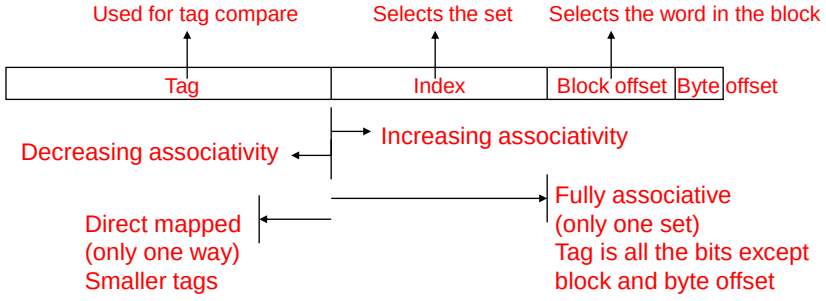
- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

Four-Way Set Associative Cache



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

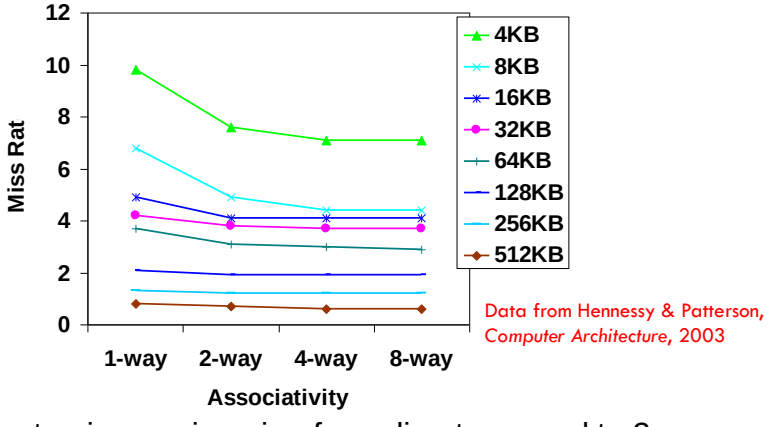


Costs of Set Associative Caches

- 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 its not possible to just assume a hit and continue and recover later if it was a miss

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

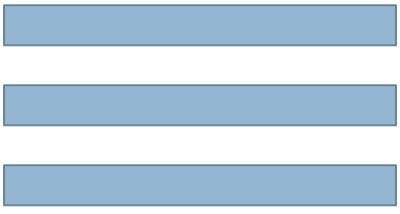


Data from Hennessy & Patterson, *Computer Architecture*, 2003

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

Class Problem

- Main Memory size is 256K words, Cache size is 16K words, block size is 32 bytes
- Show the mapping in terms of tag, index and block offset for
  - Direct-mapped
  - 4-way set-associative
  - Fully associative



Class Problem Solution

- Main Memory size is 256K words, Cache size is 16K words, block size is 32 bytes
- Show the mapping in terms of tag, index and block offset for
  - Direct-mapped

Tag = Log_2(256K x 4) – (Index + Block Offset) = 20 – (11 + 5) = 4	Index = Cache size – Block Offset = Log_2(16K x 4) - Log_2(32) = 16 – 5 = 11	Block Offset = Log_2(32) = 5

  - 4-way set-associative

Tag = Log_2(256K x 4) – (Index + Block Offset) = 20 – (9+5) = 6	Index = Cache size – (Block Offset + Associativity) = Log_2(16K x 4) – (Log_2(32) - Log_2(4)) = 16 – (5 – 2) = 9	Block Offset = Log_2(32) = 5

  - Fully associative

Tag = Log_2(256K x 4) – (Block Offset) = 20 – 5 = 15	Block Offset = Log_2(32) = 5

Reducing Cache Miss Rates #2

2. Use multiple levels of caches

- With advancing technology have 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

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
  - 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 >> than the global miss rate

Evaluating Performance

41

- Instead of *miss rate*, misses per instruction is used often

$$Misses\ per\ Instruction = \frac{Miss\ rate \times Memory\ accesses}{Instruction}$$

- AMAT can be redefined for multi-level caches as follows

$$AMAT = Hit\ Time_{L1} + Miss\ Rate_{L1} \times (Hit\ Time_{L2} + Miss\ Rate_{L2} \times Miss\ Penalty_{L2})$$

- Memory-stall clock cycles

$$Memory - stall\ clock\ cycles = \frac{Misses}{Instruction} \times Miss\ Penalty$$

Class Problem

42

- Assume the miss rate of an instruction cache is 2% and the miss rate if the data cache is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster would a processor run with a perfect cache that never missed, assume frequency of stores and loads is 36%?

- Instruction Miss Cycles = I x 2% x 100 = 2 I
  - Data Miss Cycles = I x 36% x 4% x 100 = 1.44 I
  - Total Memory Stall Cycles = 2 + 1.44 = 3.44 I
  - CPU time with stalls/CPU time with perfect cache = CPIstall/CPIperfect = 5.44/2 = 2.72

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

0. Reduce the hit time 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

1. 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

2. 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
  - write allocation
- The optimal choice is a compromise
  - depends on access characteristics
    - workload
    - use (I-cache, D-cache, TLB)
  - depends on technology / cost
- Simplicity often wins