Generations of Cache

1980: no cache in µproc;

- 1989 first Intel µproc with a cache on chip.
- 1995 2-level cache on chip

Instructions Per Cycle Lost to Memory

1st Alpha	340 ns/5.0 ns = 68 clks x 2 or	136
2nd Alpha	266 ns/3.3 ns = 80 clks x 4 or	320
3rd Alpha	180 ns/1.7 ns =108 clks x 6 or	648
Today	**80 ns/0.25 ns = 320 clks x 4 or**	**1280**

2

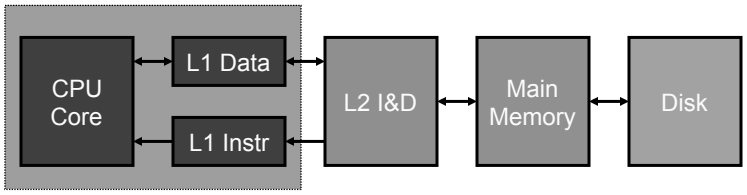
Bridging the Gap

- The "Memory Gap"
  - Processor and memory speed are disconnected and the problem is continuing to grow.
- How do we overcome????
- More ILP (by way of OOO) to overcome long latency cache misses
- Caches

3

Memory Hierarchies

- Principle of locality: *Most programs do not access all data or code uniformly, access a small number of addresses at any one time.*
- Smaller hardware is faster - Leads to a memory hierarchy with multiple levels



Typically, on-chip $$$$, Fastest!* | *$$$, Fast | *$$, Slow* | *$, Slowest*

4

Locality

- Temporal locality - locality in time

  If an item is referenced, it will likely be referenced again soon.

- Spatial locality - locality in space

  It an item is referenced, items whose addresses are close by will tend to be referenced soon.

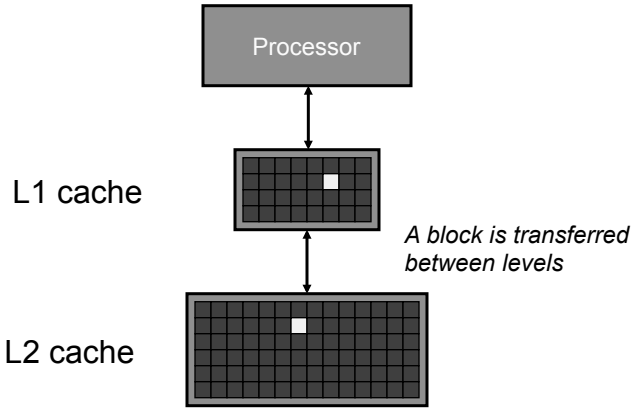
5

Memory Hierarchy Terminology

- Block: Minimum unit of information present or not present in a level (also called a "cache line")
- Hit: Data appears in a block in the upper level
  - Hit rate: Fraction of memory accesses found in upper level
  - Hit time: Time to access the upper level
- Miss: Data retrieved from a lower level
  - Miss rate = 1 - (Hit rate)
  - Miss penalty: Time to replace a block in the upper level + time to deliver the block

6

Upper and Lower Memory Levels



For data cache, a typical block size is 32B to 64B

7

Fundamental Questions for Memory Hierarchy Design

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

8

Processor Caches

9

Processor Caches

- Modern processors: 2-3 levels, some 4 levels
- L1 characteristics (on-chip SRAM)
  - Split instruction and data, 16K-32K, 1 to 8-way assoc., private
  - Very fast access: 1-4 cycles
- L2 characteristics (on-chip SRAM)
  - Unified, 256K - 2MB, 8 to 16-way assoc., private/shared
  - Fast access: 2-20 cycles
- L3 characteristics (on-chip/module SRAM)
  - Unified, 2 MB to 45 MB, shared, central/distributed
  - Moderately fast: 10-36+ cycles (location/hit type dependent)
  - May be banked (improving bandwidth)
- L4 characteristics (package/module eDRAM)
  - May be used for multiple purposes, e.g., GPU or CPU
  - 128MB or larger size, similar to DRAM but faster access

10

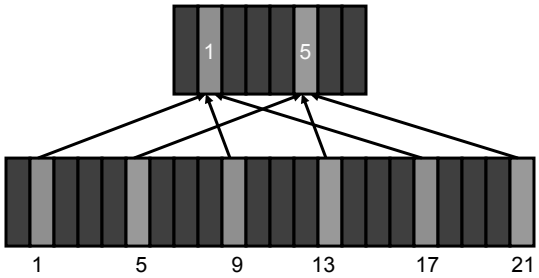
Block Placement

- Direct mapped - a block maps to a specific location in the cache
- Set associative - a block maps to one of a set of specific locations in the cache
- Fully associative - a block maps to any location in the cache

11

Direct Mapped

- Each memory location maps to exactly one location in the cache.

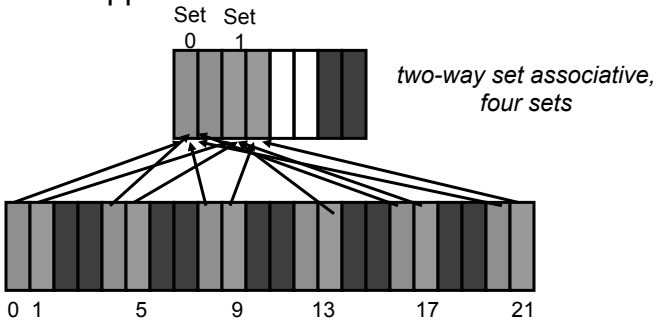


- *Cache block* = *block address* mod *# cache blocks*

12

Set Associative

- Block can appear in a restricted set of locations

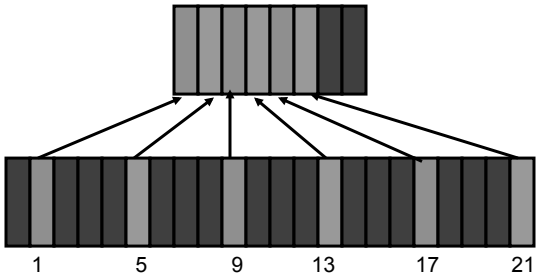


- *Cache block* = *block address* mod *# of sets*
- *n-way associative: n* blocks per set

13

Fully Associative

- Block can appear in any location



- With *m* blocks, *m*-way set associative
- Direct-mapped is one-way set associative

14

Block Identification

- We need a way to identify a block
  - E.g., direct-mapped: any one of several memory locations map to the same location; so how do we identify whether a particular memory address is in the cache?
- Each block (line) in cache has an *address tag*
- Check that the tag matches the block address from the CPU (to check for hit or miss)
- Cache blocks also need a *valid bit*
  - Identify whether the line has valid data (address)
  - Bit cleared: can t have a match on this line

15

Forming the Tag

- Block offset: Desired data from the block
- Index: Selects the set (i.e., block for DM)
- Tag: Compared to check block is the right one

- How big is the offset, index, and tag for a 32B block, 32-bit word, a two-way set associative cache with 32 lines?
- Offset = 3 bits (+2 for 4 bytes in a word), Index = 4 bits, Tag = 32-3-2-4=23 bits

16

Tag Comparison

- Only need to check the tag - why???
- Index is redundant because it is used to select the set checked
- Offset is unnecessary because the entire block is either present or not
- Keeping cache size the same and increasing the associativity, what happens to tag size???

It increases!

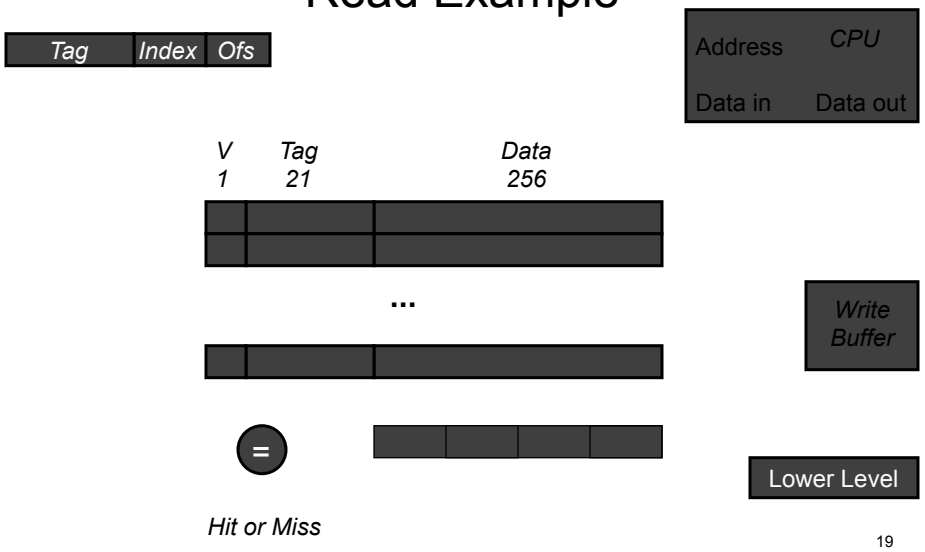
17

Tag Comparison

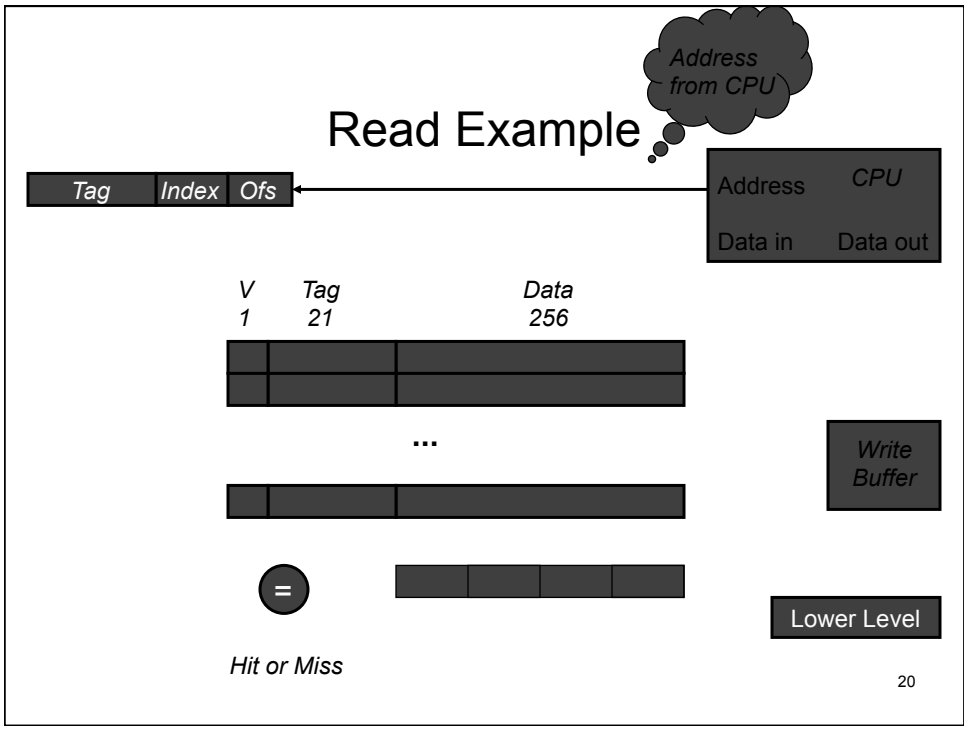
- Tag check can be done in parallel with reading the cache line
- Doesn't hurt when the tag doesn't match - just ignore the read data
- Helps when the tag matches - latency of tag comparison overlapped with line read
- Can we do this for writes????
  No! We can't modify a block until we check tags

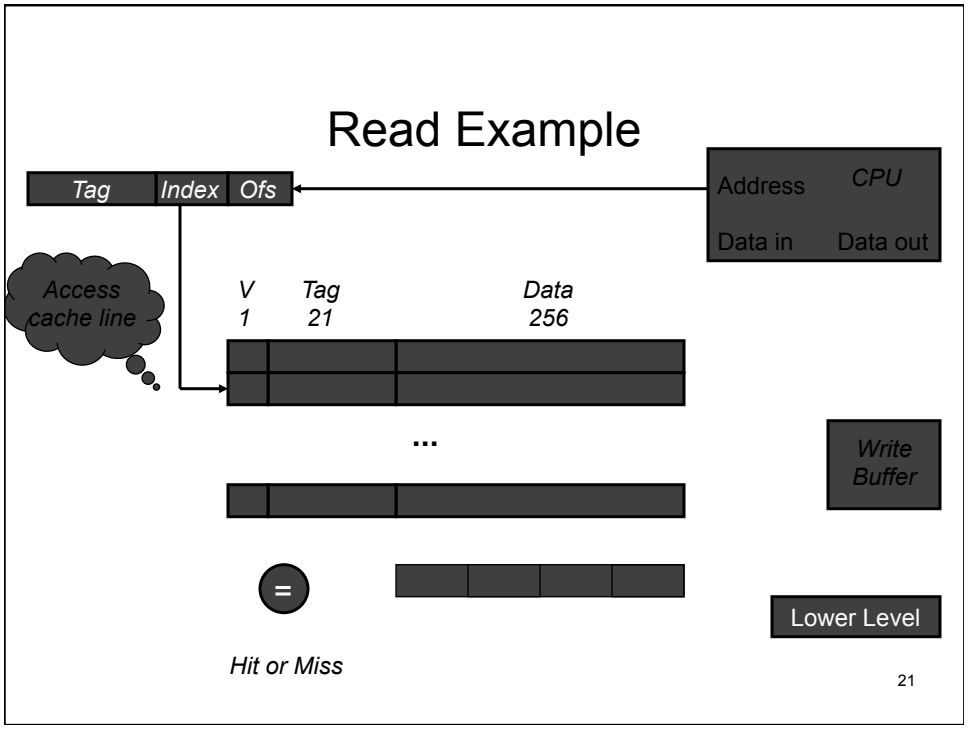
18

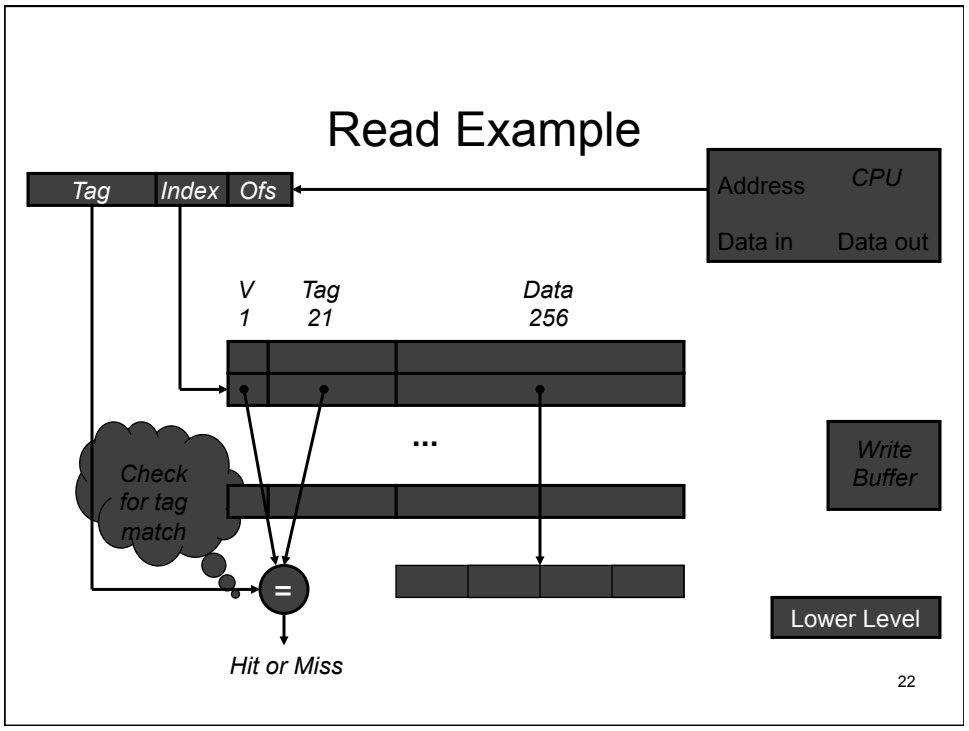
Read Example

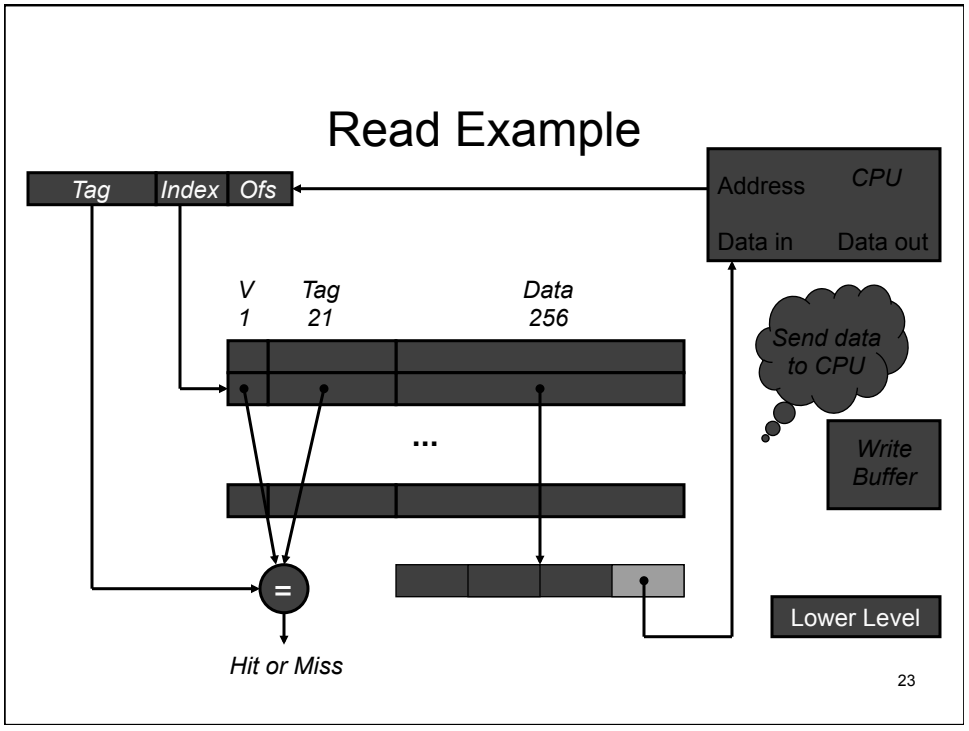


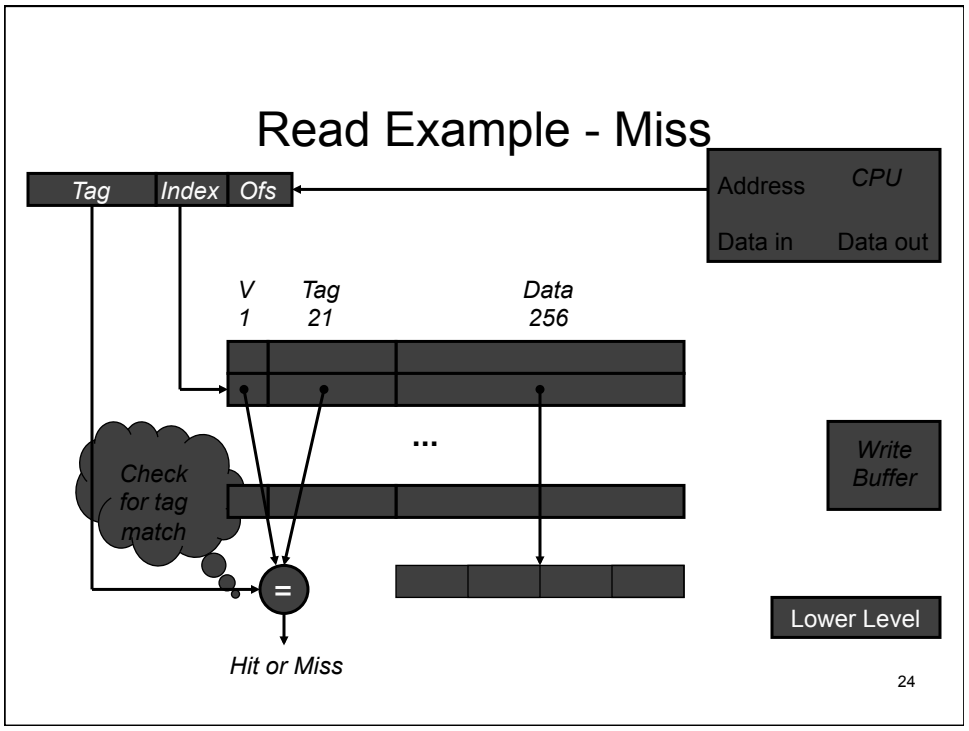
19

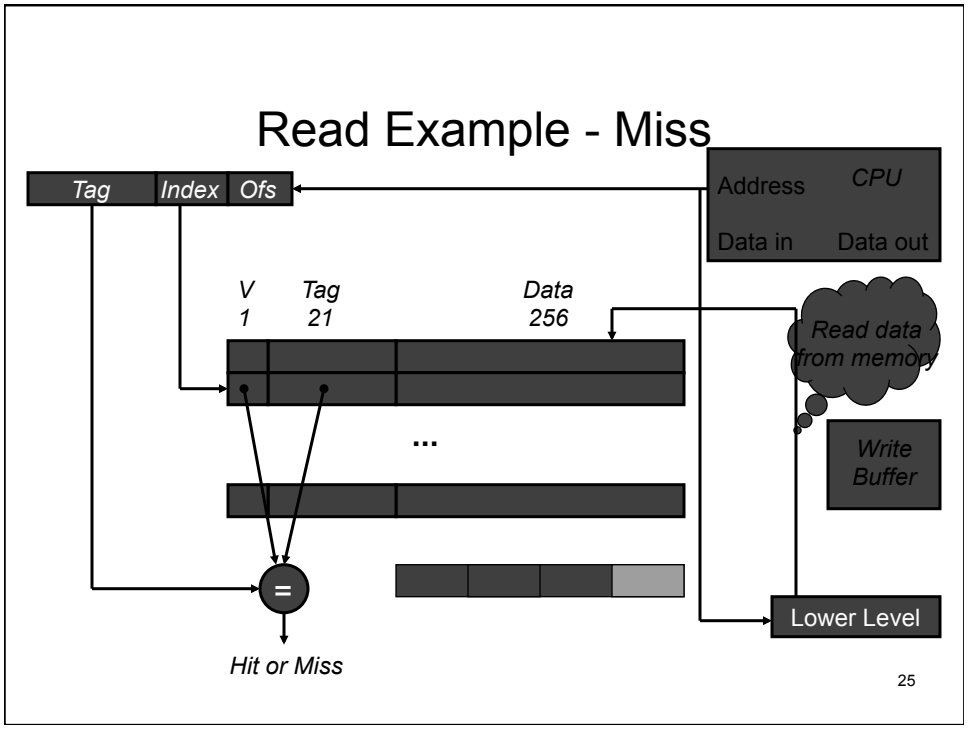

Address from CPU
Read Example
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
Tag
Data
1
21
256
...
Write Buffer
=
Lower Level
Hit or Miss
20

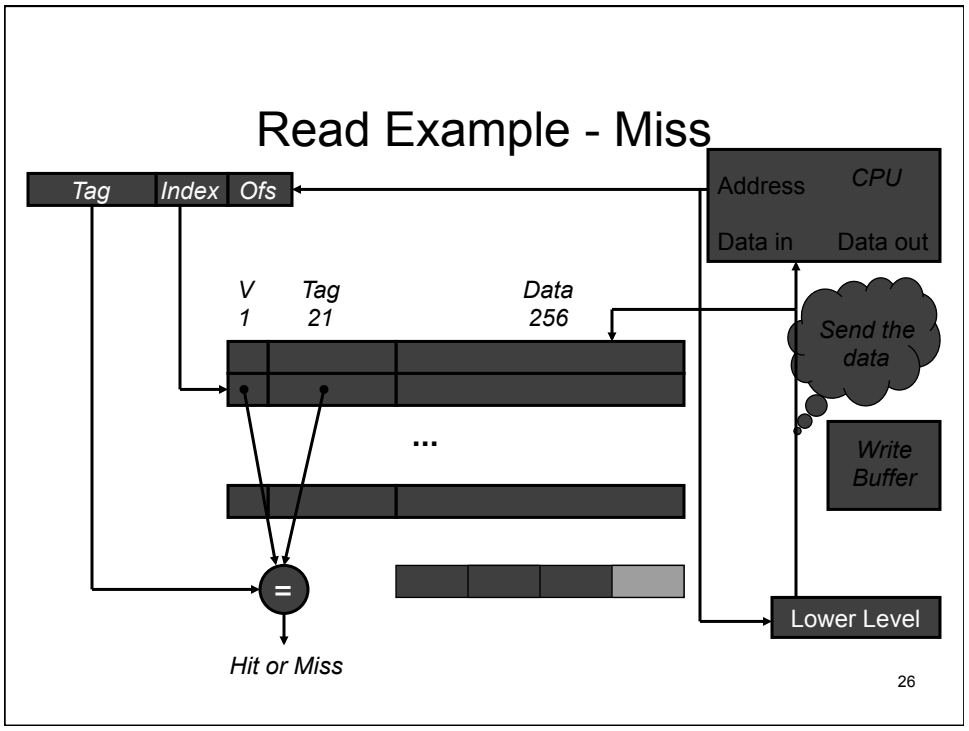

Read Example
Tag
Index
Ofs
Address
CPU
Data in
Data out
Access cache line
V
Tag
Data
1
21
256
...
Write Buffer
=
Lower Level
Hit or Miss
21

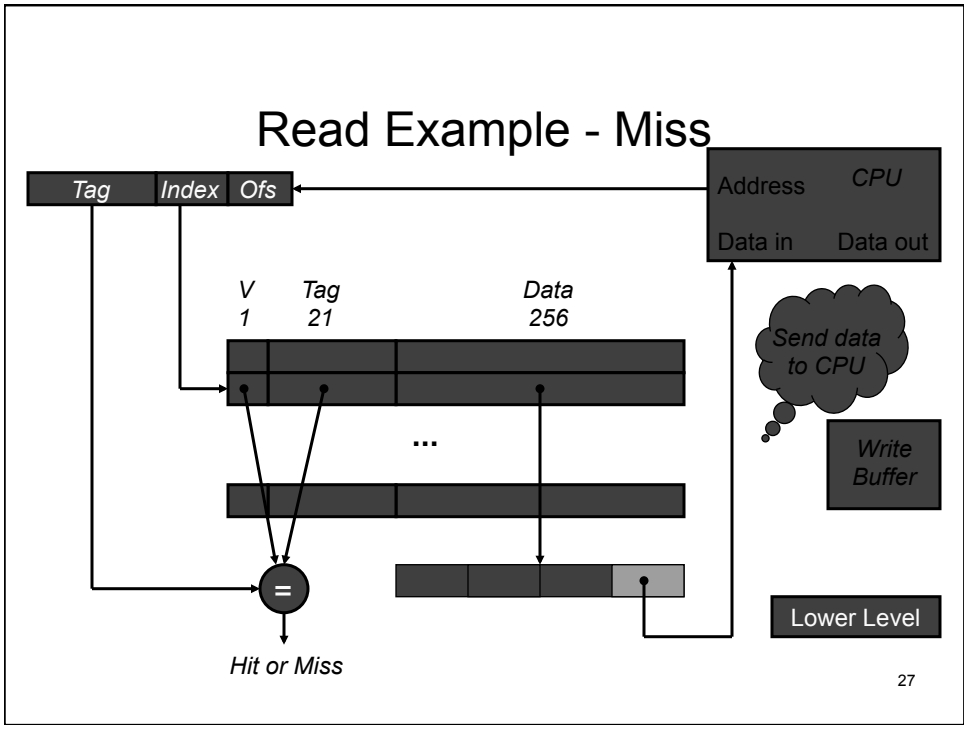

Read Example
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
...
Check for tag match
Write Buffer
=
Lower Level
Hit or Miss
22


Read Example
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
Send data to CPU
...
Write Buffer
=
Lower Level
Hit or Miss
23

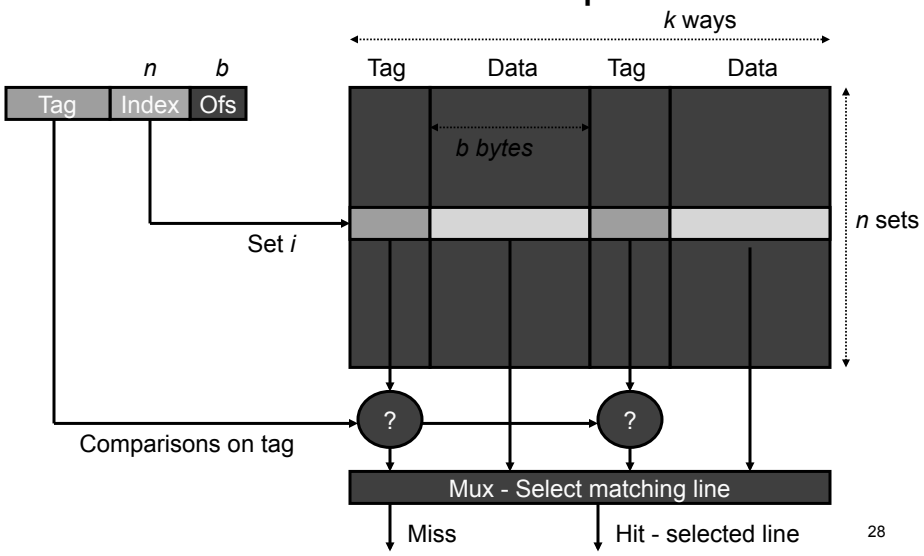

Read Example - Miss
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
...
Check for tag match
=
Write Buffer
Lower Level
Hit or Miss
24


Read Example - Miss
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
Read data from memory
...
Write Buffer
=
Lower Level
Hit or Miss
25


Read Example - Miss
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
Send the data
Write Buffer
...
=
Lower Level
Hit or Miss
26


Read Example - Miss
Tag
Index
Ofs
Address
CPU
Data in
Data out
V
1
Tag
21
Data
256
Send data to CPU
Write Buffer
...
=
Lower Level
Hit or Miss
27

Cache Lookup



28

Block Replacement

- On a miss, a block must be selected for eviction.
- Direct-mapped: The one the new block maps to.
- Set or Fully Associative:
  - Random: Spread allocation uniformly
    - » Nondeterminism can be a problem
    - » Pseudo-random to force determinism
  - Least recently used (LRU): Block replaced is one that has been unused the longest.
    - » Usually approximated
    - » Follows corollary: recently used blocks are most likely to be used again

29

LRU Approximation

- Regular LRU
  - Counter updated on every cycle
  - Countered cleared when accessed
  - Highest counter value is least recently used
  - Typically too expensive - too many bits, constant update
  - Random can work well
- Approximation
  - An access bit per block in set
  - Set on an access
  - Cleared when all bits set, except most recent
  - Replace block with cleared bit

30

Write Strategy

- How cache lines are updated on a write
  - Can't do parallel tag compare with write
  - Have to modify specified data (e.g., byte, halfword, word, quadword)
- Write policies
  - Write through: Information is updated in both the block in the cache and the lower-level memory
  - Write back: Information is written back to the lower level only when a modified block is replaced.
    - » Dirty bit: Keeps track of whether a line needs to be written to a lower level; ensures only modified lines get written to memory

31

Write Through vs. Write Back

- Write back
  - Writes happen at full cache speed (no stalls to lower level)
  - Multiple writes to same block require only one write to a lower memory level
  - Reduces bandwidth requirements between levels (less contention)
- Write Through
  - Read misses never trigger writes to lower levels
  - Next lower level has current copy of data (consistency for I/O and multiprocessors)
  - Simplest design

32

Write Stalls

- On a read miss, we stall waiting for the line (for now - this will change in a few slides)
- For writes, we can continue as soon as the data is written
- Write buffer: Holds stored data for write to cache
- Effect: Concurrently execute during a write

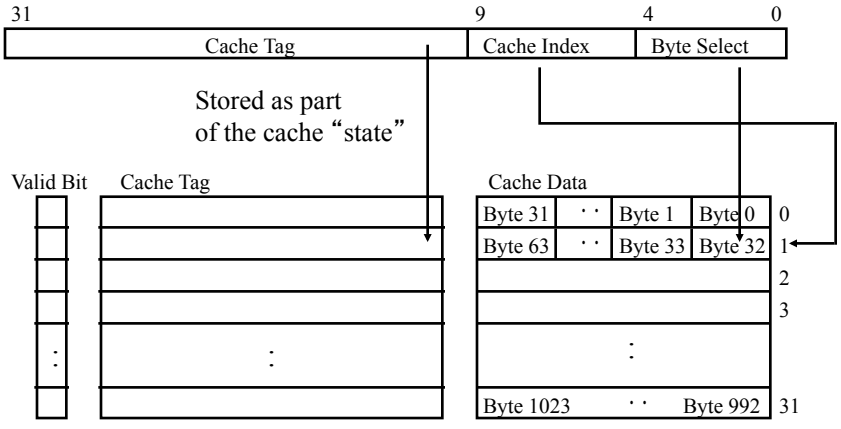
33

Handling a Write Miss

- *What if we write to a block not in the cache????*
- Write allocate: Block is loaded on a write miss, followed by write-hit actions
- Write around: Block modified in lower level and not loaded into memory
- Write back, write allocate - tries to capture future writes to that block in upper level
- Write through, write around - subsequent writes have to go to lower level anyway

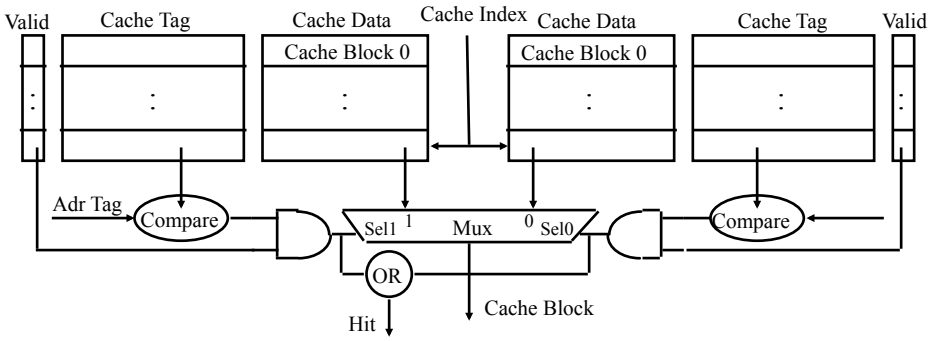
34

Direct Mapped Organization



35

Two-Way Set Associative Organization



Disadvantage to Associative Caches

- *n*-way set associative vs. direct mapped
- *n* comparators vs. 1
- Extra MUX delay for the data
- Data available AFTER hit or miss detection
- In direct mapped cache, cache block is available BEFORE hit or miss detection
  - Assume a hit and continue. Recover on a miss

Cache Performance

- Evaluating performance
  - Miss rate: It misses the point!
  - It's the miss penalty *and* the miss rate
- Average memory access time (AMAT)
  - Time (AMAT) = Hit time + Miss rate * Miss penalty
- But… what matters is the actual cycles…
  - Still evaluate with actual performance
  - Memory effects can impact instruction execution

38

Split Caches

- Unified cache: Instruction and data live in the same memory (can map to same lines)
- Split cache: Separate instruction and data caches (instructions and data don't map to same lines)
- L1 usually separate I&D caches
  - Easy bandwidth improvement (IF & MEM on same cycle)
  - Different cache geometries for I and D
    - » I and D streams have different characteristics
- Lower levels usually unified
  - Bigger memories, single path to lower levels

39

Evaluating Split Cache Performance

Comparison of split vs. unified - compare same relative size caches

Separate I&D removes conflicts between I&D blocks, but fixes the size of each cache. How does this affect miss rates?

Size	I Cache	D Cache	Unified
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

40

Evaluating Split Cache Performance

- Suppose…..

- 16 KB I&D caches vs. 32 KB unified cache
- Hit - 1 cycle, Miss - 50 cycles
- Unified cache: Load/Store hit takes 2 cycles (single port to the cache - structural hazard)
- Write through with a write buffer

- What is the average memory access time for each cache organization????

41

Example Access Times

Overall miss rate for split cache
- 75% references are instructions
- Overall miss rate = (75% * 0.64%) + (25% * 6.47%) = 2.10%

Miss rate for 32 KB unified is 1.99% (experiment)

Access time = % I * (Hit time + Miss rate * Miss penalty) +
% D * (Hit time + Miss rate * Miss penalty)

$Time_{Split}$ =
75% * (1 + 0.64% * 50) + 25% * (1 + 6.47% * 50) = 2.05
$Time_{Unified}$ =
75% * (1 + 1.99% * 50) + 25% * (1 + 1 + 1.99% * 50) = 2.24

42

Factoring in Memory Performance

CPU time =
(CPU cycles + Memory stall cycles) * Clock Cycle time

Hit cycles included in CPU cycles.
Memory stalls can be defined in terms of memory accesses.

Stall cycles =
Reads * Read miss rate * Read miss penalty +
Writes * Write miss rate * Write miss penalty

Stall cycles = Memory accesses * Miss rate * Miss penalty
Simplified version, combing writes and reads in a single term

43

Factoring in Memory Performance

CPU time =

IC * (CPI + Mem. accesses/instr. * Miss rate * Miss Penalty) * Clock cycle time

Basic CPU equation factoring in memory performance

Misses per instructions =

(Memory accesses * Miss rate) / Instructions

Architecture dependent metric - e.g., x86 vs. SPARC

Often reported as misses per kilo (1000) instructions: MPKI

CPU time =

IC * (CPI + Memory stall cycles / instr) * Clock cycle time

Factors in misses per instruction (stalls)

44

Example of Memory Evaluation

- Miss penalty is 50 clock cycles
- Instructions normally take 2 cycles (ignoring memory stalls)
- Miss rate is 2% and 1.33 memory references per instruction
- What is the performance with the cache?

CPU time = IC * (CPI + Stalls/Instr) * Clock cycle time

= IC * (2 + (1.33 * 2% * 50)) * Clock cycle time

= IC * 3.33 * Clock cycle time

CPI increases from 2 to 3.33 with the cache

What happens without the caches but 50 cycle memory?

45

Impact of Cache Performance

- For a CPU with low CPI and fast clock, cache performance is particularly important:
  - Lower CPI implies the higher the relative impact of a fixed number of cache miss cycles
  - For identical memory hierarchies, a machine with a fast clock cycle has a higher number of cache miss cycles. Hence, memory portion of CPI is higher.

46

Improving Cache Performance

- What should be tackled to improve cache performance????

① Reducing cache misses
② Reducing cache miss penalty
③ Reducing cache hit time

47

Reducing Cache Misses

- Types of misses - the three C's
  - Compulsory
    - » First access to a block when it's not in the cache, it must be loaded
  - Capacity
    - » If cache can't contain all blocks in the working set, capacity misses occur because blocks are discarded later
  - Conflict
    - » For set associative or direct-mapped caches when multiple blocks map to same location causing some block to be discarded that is loaded later

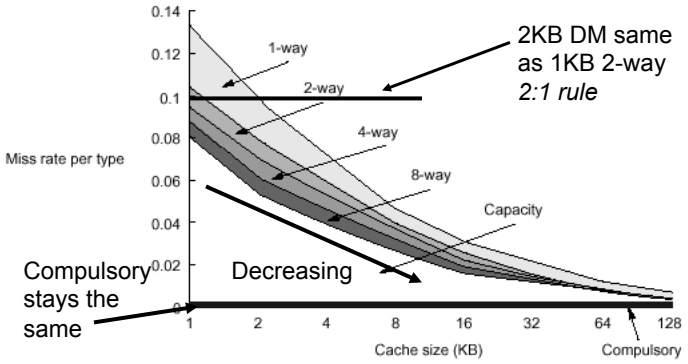
48

Dealing with Cache Miss Types

- Conflicts
  Increase associativity
  Fully associative has no conflict misses
- Capacity
  Enlarge the cache
  Thrashing possible when capacity is too small
- Compulsory
  Increase block size (effectively prefetching data)
  Independent of cache size (why?)

49

Absolute Miss Rate



Three C Model

- Simple model about average miss behavior
- Doesn't tell you about individual misses
- Changing cache size spreads references out to more blocks
  - Conflict and capacity misses are affected
  - E.g., A *capacity miss* may become a *conflict miss*
- Says nothing about
  - Replacement policy
  - Miss penalty
  - Hit time

Larger Block Size

- Increase block size - what happens???
  Reduces compulsory misses
- Effectively, needed data is brought into the cache on some other miss, thereby reducing the misses
- Larger blocks exploit what???
  Spatial locality

52

Downside: Larger Blocks

- Increased miss penalty (more time to fill)
- Reduced number of blocks
  - For a given cache size
  - More conflict misses and maybe even capacity misses
- At a point, may actually *increase* the miss rate
- May not be willing to pay increased miss penalty for decreased miss rate

53

Block Size Selection

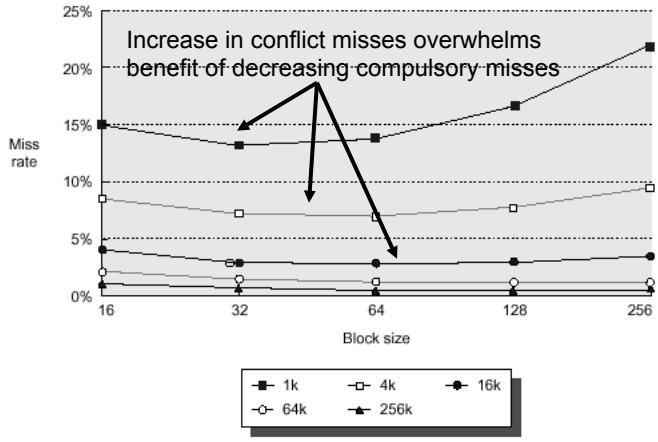
- Block size depends on lower level's
  - Latency
  - Bandwidth
- High latency, high bandwidth - what size???

  Increased line size - get more data for small increase in miss penalty
- Low latency, low bandwidth

  Smaller block size b/c little benefit from increased miss penalty (may fetch unneeded data)

54

Effect of Block Size



55

Higher Associativity

- Increasing associativity of cache
  - Decreases conflict misses
- Eight-way set associative does nearly as well as a fully associative cache (from book)
- 2:1 rule - A direct-mapped cache of size N does nearly as well as a two-way set associative cache of size N/2

Downside: Higher Associativity

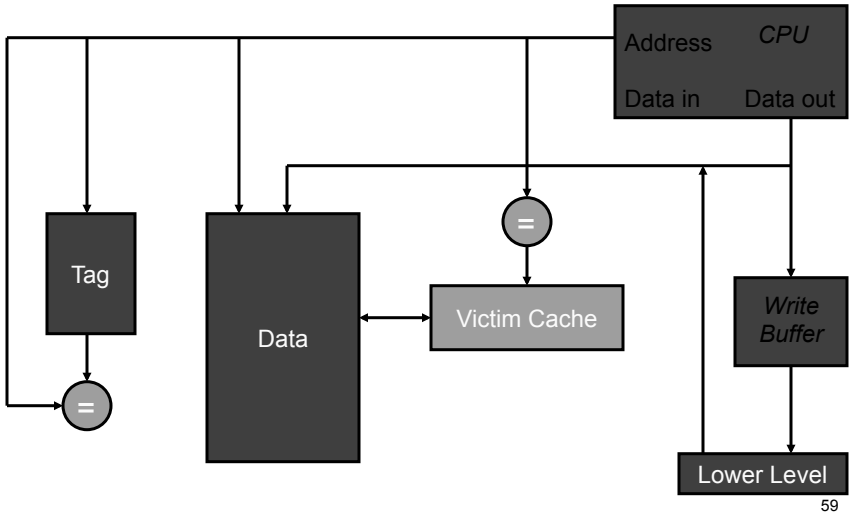
- Increased hit latency
  - More levels in MUXes to select data
- Increased power consumption
  - More banks are accessed
  - Can get around by accessing tag arrays first before reading the data (when tag matches, select the matching data array)
- Usual trade-off
  - Lower miss rate, higher hit latency and more power

Victim Cache

- A small, fully associative cache between a cache and its fill path
- Victim cache contains blocks discarded from a miss (so called ''victims'')
- Victim cache checked on a miss for data before going to lower level
- On a miss and hit in VC, swap victim block with block in cache
- No clock rate effect!

58

Victim Cache Architecture



59

Victim Cache

- Improves miss rate by keeping blocks that might have been mistakenly evicted
- Effectively increases associativity - reduces the conflict misses
- A four-entry victim cache works well
  - 20 - 95% of conflict misses in a 4KB direct-mapped cache are removed
- Typically used with direct-mapped cache to achieve effect of a set associative cache

60

Way Prediction

- Speed of direct-mapped with miss rate of n-way set associative cache
- Works like a direct mapped cache initially
  - Predict likely Way to hit
  - Check appropriate entry for a hit
  - If a hit, return data
- On a miss
  - Check other cache ways *in same level* for a hit
  - 2-way: The second entry's index can be formed by inverting most significant bit of index field
  - No clock rate effect - speed of DM for first access

61

Way Prediction

- Fast hit: Hit in the first way (speed of DM)
- Slow hit ("pseudo hit"): Hit in the other ways

- Concept can be applied to Direct Mapped
  - So called pseudo-associative cache
  - Fast hits in direct mapped cache can become slow hits in the pseudo-associative cache

- What happens when we have lots of slow hits for the same data?
  - Update way prediction (similar to branch prediction)
  - Swap fast hit entry and pseudo hit entry on a pseudo hit (may cause thrashing)

62

Hardware Prefetching

- Prefetching: Load items before they are needed
  - Instructions and data
  - Loaded into cache or buffer between upper and lower levels

- Instruction prefetching (one possible scheme)
  - On a miss, get missed block and next subsequent block
  - Hold subsequent block in buffer until needed (avoids evicting a possibly needed block)
  - On a miss to a prefetched block, it is copied from the "instruction stream buffer" to the cache
  - 4 blocks in instruction stream buffer: instruction hit rate increases by 50%

63

Data Prefetching

- Apply the same idea
- On a miss, prefetch the next block
  - A twist: Detect data access pattern
  - E.g., stride of array accesses
  - Prefetch the next predicted block based on the past data access pattern
- Multiple stream buffers for data
  - Each prefetch at a different address
  - 4 buffers increase data hit rate by 43%

64

Software Prefetching

- Compiler inserts prefetch instructions
  - Register prefetch: Prefetched data loaded into a register
  - Cache prefetch: Prefetched data loaded into the cache
- Requires some support
  - Non-faulting prefetch instructions (“nonbinding prefetch”)
  - Nonblocking caches: processor can continue while prefetching the data
- Overlap execution with prefetching of data
  - Loops - schedule prefetches on earlier iterations so data available on later ones

65

Reducing Cache Miss Penalty

- *Much research* has focused on cache miss rates
- But the latency of a miss also matters
- You can't really separate the two to get a meaningful understanding of what is happening
- A number of optimizations on how to handle misses to improve their latency
- Also, increase the latency of the lower level!

66

Read Priority over Write

- Write-through cache - write buffer avoids stalling
- Write buffer may hold a value that is being loaded
- Read miss can wait for buffer to empty
  - for WT cache, WB (few words) almost always has data in it
  - Don't wait: Check contents of write buffer before sending the read miss to a lower level
- Write back costs - whole cache line written
  - On replacement, write old line, copy new line, CPU executes
  - Better: move old line to a write buffer, copy new line, CPU executes, write old line to memory when bus available
  - On a miss, CPU must check the write buffer

67

Early Restart - Critical Word First

- CPU just needs one word
- Why wait for entire cache line?

- Early restart - send data to CPU directly from memory on a read miss (no second access)
- Critical word first - request missed word first and fill cache line while CPU executing

- Benefits designs with long cache lines (or, rather, cache fill time is high relative to CPU speed)

69

Lockup-Free Caches

- Out-of-order completion
  - Don't stall CPU waiting for cache miss to be serviced
  - Continue executing instructions

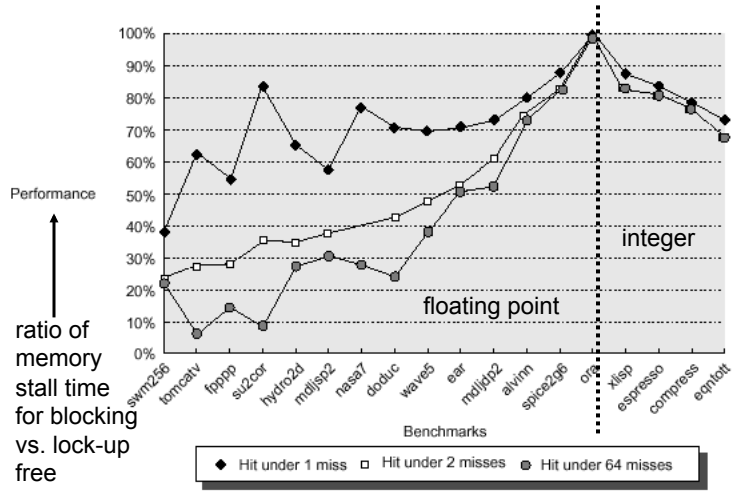
- Lockup-free cache: Allows data cache to continue supplying cache hits during cache misses - "hit under miss"

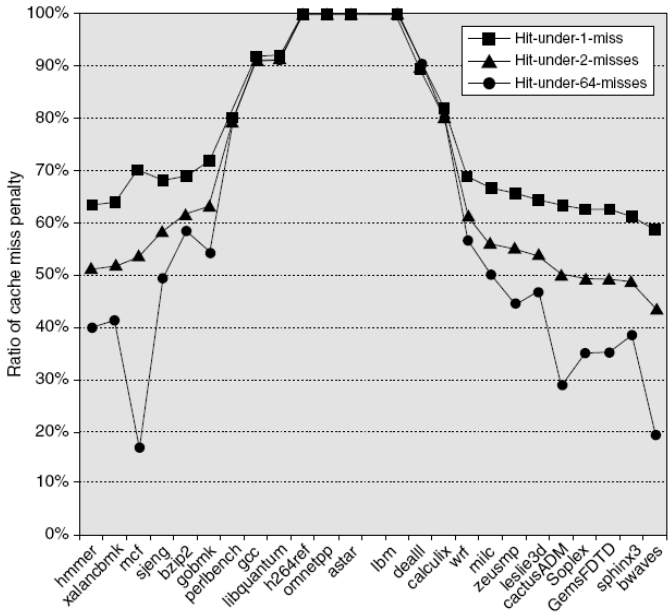
- Multiple misses: Cache overlaps miss handling of separate misses (buffers loads)

- Multiple outstanding accesses: Complexity!

70

Blocking vs. Lock-up Free Cache





Second-Level Caches

- Trade off
  - Make L1 cache faster to keep up with CPU
  - Make L1 cache larger to reduce miss rate
- What's the expense of doing this?
- Instead, do both by adding a L2 cache
  - First level: Fast and small: keeps up with CPU
  - Second level: Captures many misses that would go to main memory
  - Reduces miss penalty of L1
  - Hardware is relatively straightforward
    - » Add additional cache unit between first level and main memory.

73

Performance Analysis

- Second-level miss rate measured on misses from the first level cache
- Local miss rate - Number of misses to the cache divided by total number of memory accesses
- Global miss rate - Number of misses in cache divided by total number of memory accesses generated by the CPU (2nd level's global miss rate is: $\text{Miss rate}_{L1} * \text{Miss rate}_{L2}$)

74

Performance Analysis

Extend the memory access time for one level cache to handle two levels

How can we compute the overall access time for both levels of cache?

75

Performance Analysis

L1 average access time is???

Time = Hit time$_{L1}$ + Miss rate$_{L1}$ * Miss penalty$_{L1}$

For L2, we know that

Miss rate$_{L1}$ is rate of *accesses* to L2

Miss penalty$_{L1}$ is *average access time* for L2

We can rewrite miss penalty$_{L1}$ in terms of L2

Miss penalty$_{L1}$ = Hit time$_{L2}$ + Miss rate$_{L2}$ * Miss penalty$_{L2}$

Then, overall access time is

Time = Hit time$_{L1}$ + Miss rate$_{L1}$ *(Hit time$_{L2}$ +Miss rate$_{L2}$ *Miss penalty$_{L2}$)

76

Reducing Hit Time

- Tag comparison is expensive part of hit time
  - Read tag memory, compare tag to address
- Small and simple caches
- Small: Keep cache small enough so it fits entirely on chip - avoid the off-chip access expense
- Simple: Direct mapped caches
  - Overlap tag comparison with delivery of data
- High clock rates: Encourages *small caches that can be accessed in a single cycle*

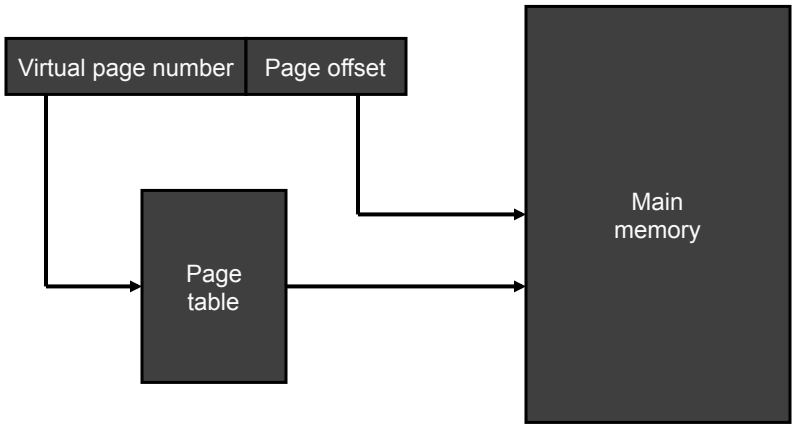
77

Avoid Address Translation

- Virtual address
  - Maps "logical address" to physical address in memory
  - Virtual address on disk mapped to physical address in main memory
- Hits more common than misses
  - Avoid address translation since most are hits
  - Argues for caching on virtual address ("virtual cache")
- Virtual vs. physical cache
  - Addressed using virtual address (tag comparison on vaddr.)
  - No address translation before cache access

78

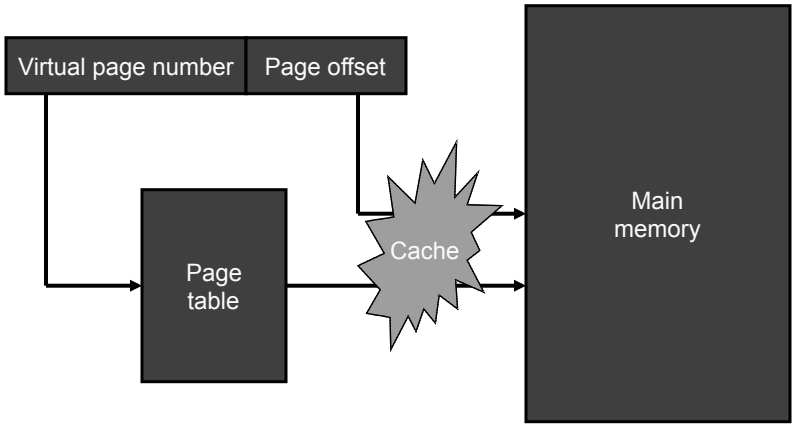
Virtual Address Translation



Map logical address into a physical address located in memory

79

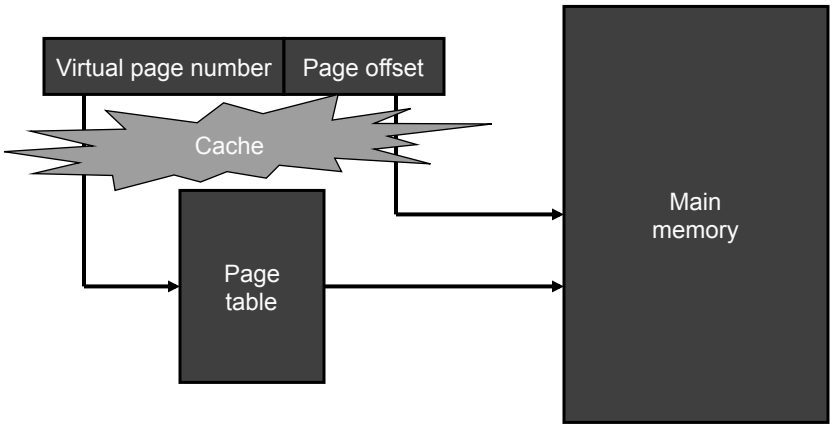
Virtual Address Translation



Physical cache - after mapping address

80

Virtual Address Translation



Virtual cache - addresses based virtual address

81

Disadvantages: Virtual Caches

- What happens with a context switch????
- On process switch, cache must be flushed since virtual addresses from one process map to another
- How can we avoid the ambiguity????
- Avoiding flushes
  - Attach PID to cache address tag
  - Only flush when a PID is reassigned

82

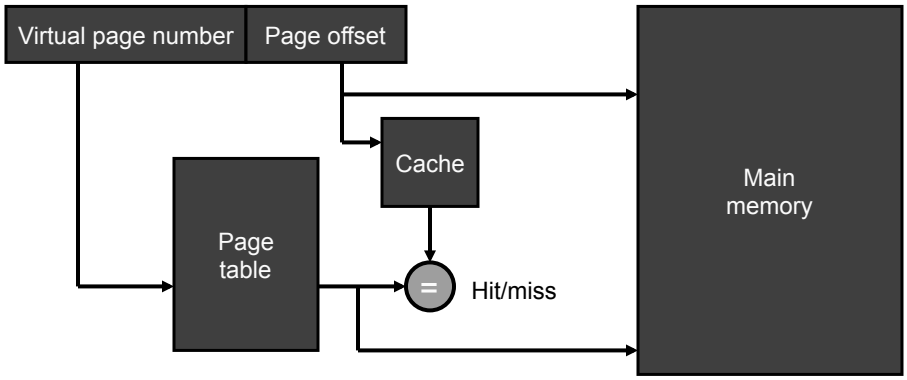
Disadvantages: Virtual Caches

- Another problem - aliasing
  - OS and user programs may use different virtual addresses for the same physical address
  - Two copies of data in cache at once
  - Must update both; or ensure that every block has an unique physical address (no duplicates)
- Yet another problem - I/O lives in physical address space
  - I/O physical addresses would have to be mapped to virtual addresses for a virtual cache
  - Mapping needed to tell that an address is in I/O space

Solutions: Virtual Caches

- Fast hits: Separate translation and cache access
  - Separate pipeline stages
    - » Address translation stage
    - » Cache access stage (with physical address)
  - Fast cycle time with slow cache access (2 cycles)
    - » I.e., increased access time with a fast clock rate
  - Increases misprediction branch penalty
- A final solution: Use page offset to index cache
  - So called "virtually indexed, physically tagged"
  - Used in Alpha 21064 and others

Virtually Indexed, Physically Tagged



Page offset indexes cache, tags checked after address translation finishes

85

Virtually Indexed, Physically Tagged

- Features
  - Tag comparison on physical address
  - Tag read can be overlapped with address translation
- Operation
  - Page offset is not translated during virtual address translation
  - Reading tags done concurrently with address translation
  - Tag comparison occurs on physical address after translation
- Limitation
  - DM cache can be no larger than a page size
  - Increase associativity to keep index within one page size

86

Pipelining Writes

- Write hits take longer than reads - why????
- Tag comparison before writing the data

- Pipelining writes
  - Separate tag and data memories
  - On a write, compare with tag
  - With a *different* address, a write can proceed on a line with the tag comparison
  - I.e., tag comparison for write *i* and write for write *i-1*

Cache Optimization Summary

What	Miss Rate	Miss Penalty	Hit Time	Comp?	Notes
Larger block size	+	-		0	
Higher associativity	+		-	1	MIPS R10K 4way
Victim caches	+			2	HP7200
Pseudo-assoc.	+			2	L2 MIPS R10K
HW prefetch	+			2	Instr common
SW prefetch	+			3	Lock-up free
Read priority		+		1	Most
Critical word		+		1	Most
Nonblocking		+		3	Most
2nd level cache		+		2	Most
Small, simple	-		+	0	Pentium
Addr. Translation			+	2	
Pipelining writes			+	1	

Block Size Example

- 40 cycle memory overhead, 16 bytes delivered every 2 cycles.
- E.g., 16 bytes in 42 cycles, 32 bytes in 44 cycles
- Baseline caches miss rates
  - 1K, 32B line: 13.34% 64K, 32B line: 1.35%
  - 1K, 64B line: 13.76% 64K, 64B line: 1.06%
- What block size has the minimum access time for these four caches?

Block Size Example

- Miss penalty based on line size - what is it???
  - 32B: 40 cycle overhead + (32/16)*2 cycles = 44 cycles
  - 64B: 40 cycle overhead + (64/16)*2 cycles = 48 cycles
- Average memory access time
  - Time = Hit time + Miss rate * Miss penalty

1K, 32B:	1 cycle + (13.34%*44 cycles) =	6.870 cycles
1K, 64B:	1 cycle + (13.76%*48 cycles) =	7.605 cycles
64K, 32B:	1 cycle + (1.35%*44 cycles) =	1.594 cycles
64K, 64B:	1 cycle + (1.06%*48 cycles) =	1.509 cycles

Associativity Example

- Suppose increasing associativity effects the clock rate:
  - 2-way 10% penalty (1.1 * base clock time)
  - 4-way 12% penalty (1.12 * base clock time)
  - 8-way 14% penalty (1.14 * base clock time)
- Hit time is 1 clock cycle, miss penalty for direct mapped cache is 50 cycles
- Miss rates for a 16K cache
  - DM: 2.9% 2-way: 2.2%
  - 4-way: 2.0% 8-way: 1.8%
- What is the access time for each cache?

91

Associativity Example

Time = Hit time + Miss rate * Miss penalty

DM:	1.0 + 2.9%*50 =	2.45
2-way:	1.1 + 2.2%*50 =	2.2
4-way:	1.12 + 2.0%*50 =	2.12
8-way:	1.14 + 1.8%*50 =	2.04

92

Second Level Cache Example

- 2-way set associative increases CPU clock cycle time by 10% (thus, hit time goes up by 10%)
- Hit time for L2 direct mapped is 10 cycles
- Local miss rate for L2 DM is 25%
- Local miss rate for 2-way associative is 20%
- Miss penalty for L2 is 50 cycles

- What's the impact of the second-level cache's associativity on the L1 cache's miss penalty?

93

Second Level Cache Example

For direct mapped L2 cache, L1's miss penalty is:
Miss penalty L1 = L2 hit time + L2 miss rate * L2 miss penalty
Miss penalty L1 = 10 + 25% * 50 = 22.5 clock cycles

Let's add associativity - 10% clock cycle penalty, 20% L2 miss rate:
Miss penalty L1 = 10.1 + 20% * 50 = 20.1 clock cycles

Clock cycle has to be an integral number of cycles, so 10 or 11 :
Miss penalty L1 = 10 + 20% * 50 = 20.0 clock cycles
Miss penalty L2 = 11 + 20% * 50 = 21.0 clock cycles

94