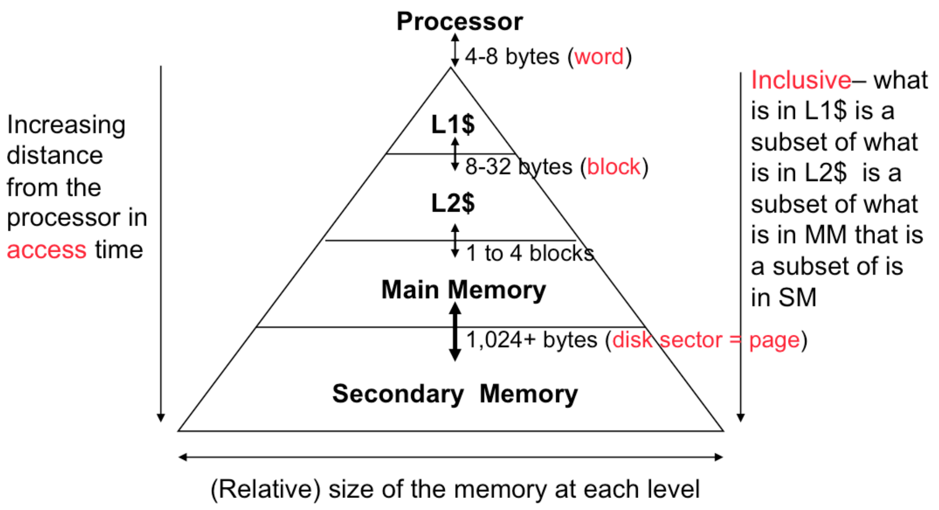
Improving Cache Performance

Other handouts
To handout next time

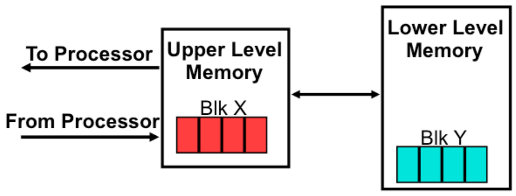
Review: The Memory Hierarchy

- Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology


Processor
4-8 bytes (word)
L1$
8-32 bytes (block)
L2$
1 to 4 blocks
Main Memory
1,024+ bytes (disk sector = page)
Secondary Memory
Increasing distance from the processor in access time
Inclusive– what is in L1$ is a subset of what is in L2$ is a subset of what is in MM that is a subset of is in SM
(Relative) size of the memory at each level

Review: Principle of Locality

- Temporal Locality
  - Keep most recently accessed data items closer to the processor
- Spatial Locality
  - Move blocks consisting of contiguous words to the upper levels



- Hit Time << Miss Penalty
  - Hit: data appears in some block in the upper level (Blk X)
    - Hit Rate: the fraction of accesses found in the upper level
    - Hit Time: RAM access time + Time to determine hit/miss
  - Miss: data needs to be retrieved from a lower level block (Blk Y)
    - Miss Rate = 1 - (Hit Rate)
    - Miss Penalty: Time to replace a block in the upper level with a block from the lower level + Time to deliver this block's word to the processor
    - Miss Types: Compulsory, Conflict, Capacity

Measuring Cache Performance

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

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

- Memory-stall cycles come from cache misses (a sum of read-stalls and write-stalls)

wichtig

$$\text{Read-stall cycles} = \text{reads/program} \times \text{read miss rate} \times \text{read miss penalty}$$

$$\text{Write-stall cycles} = (\text{writes/program} \times \text{write miss rate} \times \text{write miss penalty}) + \text{write buffer stalls}$$

- For write-through caches, we can simplify this to

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

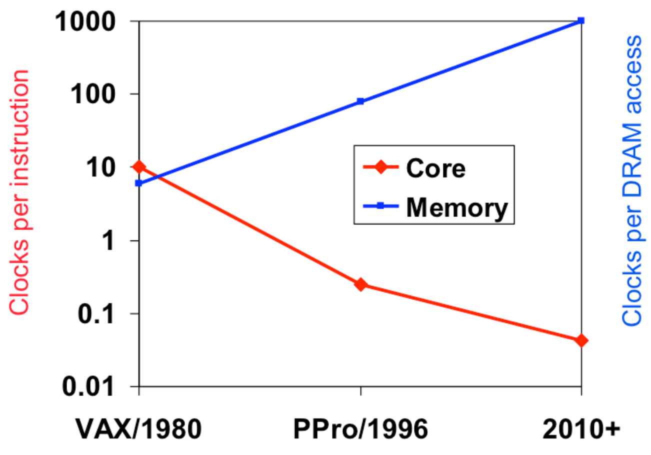
Reasonable write buffer depth (e.g., four or more words) and a memory capable of accepting writes at a rate that significantly exceeds the average write frequency means write buffer stalls are small

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

CPI = clock cycles per instruction

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

  Memory-stall cycles = 2% × 100 + 36% × 4% × 100 = 3.44

  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 ideal CPI = 1, then CPIstall = 4.44 and the amount of execution time spent on memory stalls would have risen from 3.44/5.44 = 63% to 3.44/4.44 = 77%

For miss penalty of 200, memory stall cycles = 2% 200 + 36% x 4% x 200 = 6.88 so that CPIstall = 8.88

This assumes that hit time is not a factor in determining cache performance. A larger cache would have a longer access time (if a lower miss rate), meaning either a slower clock cycle or more stages in the pipeline for memory access.

Reducing Cache Miss Rates #1

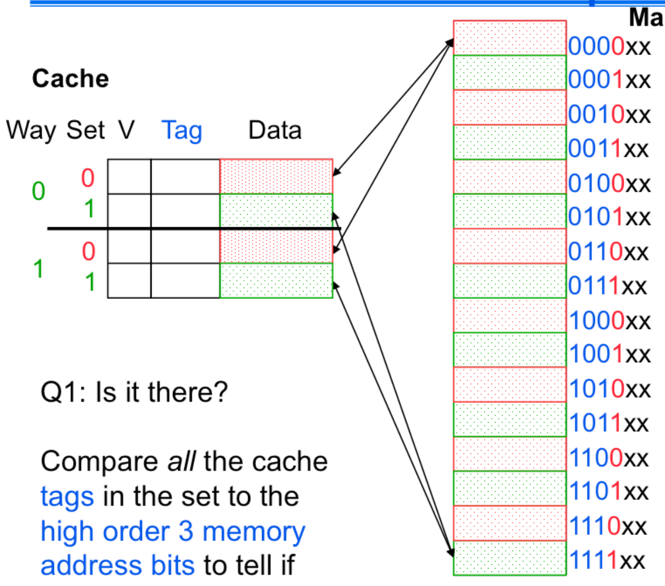
1. Allow more flexible block placement

- 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 on 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)

For lecture

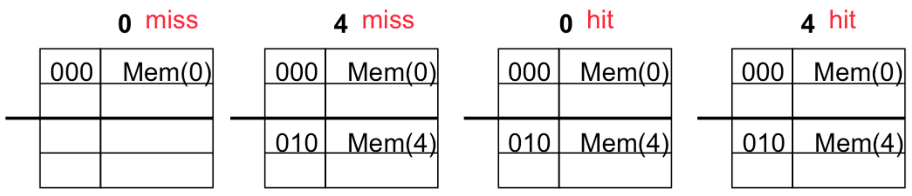
Valid bit indicates whether an entry contains valid information – if the bit is not set, there cannot be a match for this block

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



- 8 requests, 2 misses

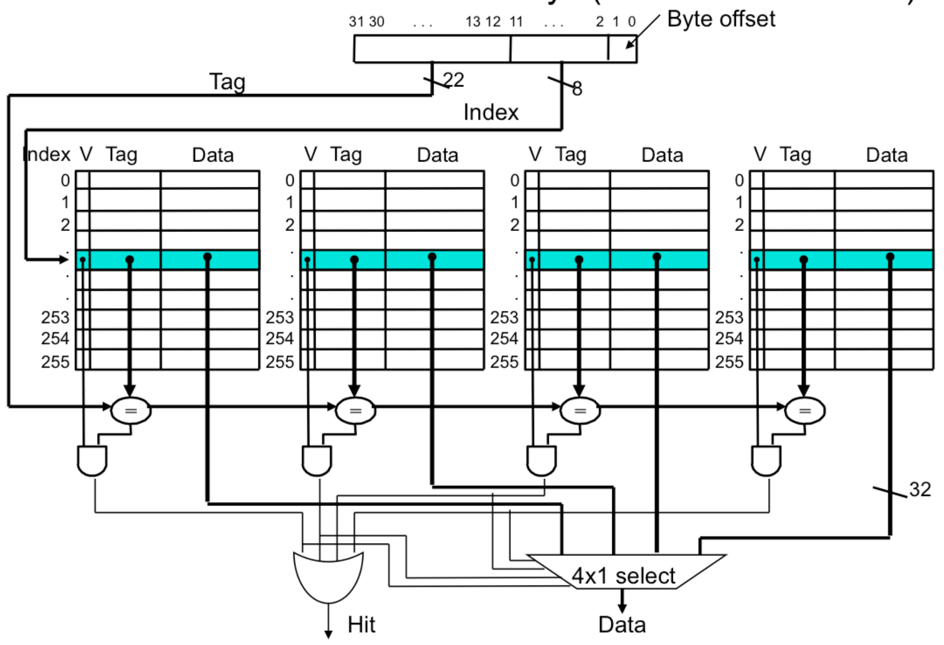
- Solves the ping pong effect in a direct mapped cache due to conflict misses since now two memory locations that map into the same cache set can co-exist!

For lecture

Another sample string to try 0 1 2 3 0 8 11 0 3

Four-Way Set Associative Cache

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



This is called a 4-way set associative cache because there are four cache entries for each cache index. Essentially, you have four direct mapped cache working in parallel.

This is how it works: the cache index selects a set from the cache. The four tags in the set are compared in parallel with the upper bits of the memory address.

If no tags match the incoming address tag, we have a cache miss.

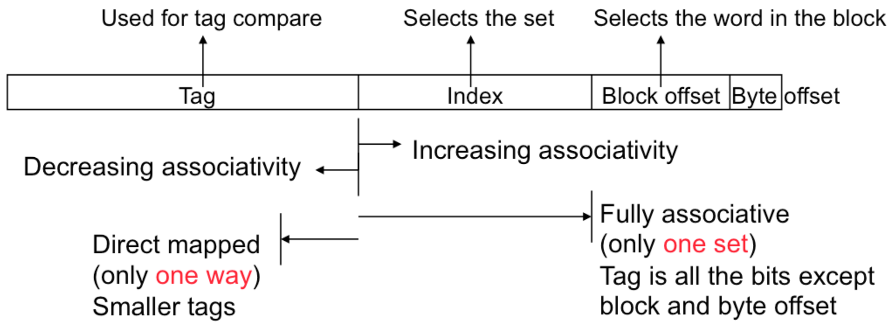
Otherwise, we have a cache hit and we will select the data from the way where the tag matches occur.

This is simple enough. What are its disadvantages?

+1 = 36 min. (Y:16)

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



For lecture

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

For a 4-way we can have one bit for the LRU pair of blocks and
then one bit for which of the blocks is LRU. Too costly in general N-ways

First of all, a N-way set associative cache will need N comparators instead of just one comparator (use the right side of the diagram for direct mapped cache).

A N-way set associative cache will also be slower than a direct mapped cache because of this extra multiplexer delay.

Finally, for a N-way set associative cache, the data will be available AFTER the hit/miss signal becomes valid because the hit/mis is needed to control the data MUX.

For a direct mapped cache, that is everything before the MUX on the right or left side, the cache block will be available BEFORE the hit/miss signal (AND gate output) because the data does not have to go through the comparator.

This can be an important consideration because the processor can now go ahead and use the data without knowing if it is a Hit or Miss. Just assume it is a hit.

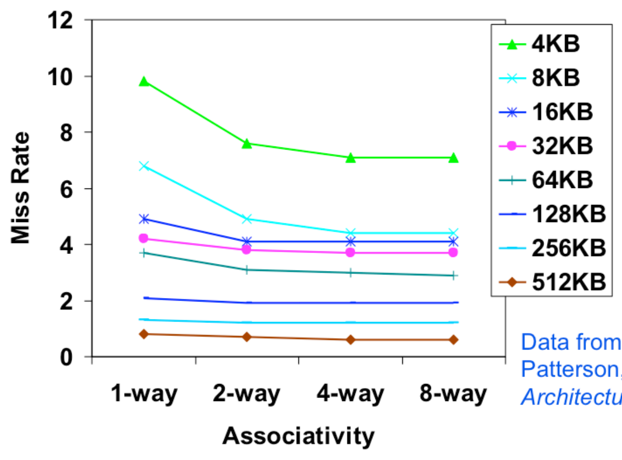
Since cache hit rate is in the upper 90% range, you will be ahead of the game 90% of the time and for those 10% of the time that you are wrong, just make sure you can recover.

You cannot play this speculation game with a N-way set-associative cache because as I said earlier, the data will not be available to you until the hit/miss signal is valid.

+2 = 38 min. (Y:18)

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)

As cache sizes grow, the relative improvement from associativity increases only slightly; since the overall miss rate of a larger cache is lower, the opportunity for improving the miss rate decreases and the absolute improvement in miss rate from associativity shrinks significantly.

Reducing Cache Miss Rates #2

2. Use multiple levels of caches

- As technology advances, we 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
- For our example, $\text{CPI}_{\text{ideal}}$ of 2, 100 cycle miss penalty (to main memory), 36% load/stores, a 2% (4%) L1I$ (D$) miss rate, add a UL2$ that has a 25 cycle miss penalty and a 0.5% miss rate

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

(as compared to 5.44 with no L2$)

Also reduces cache miss penalty

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

Global miss rate – the fraction of references that miss in all levels of a multilevel cache. The global miss rate dictates how often we must access the main memory.

Local miss rate – the fraction of references to one level of a cache that miss

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

A trace cache finds a dynamic sequence of instructions including taken branches to load into a cache block. Thus, the cache blocks contain dynamic traces of the executed instructions as determined by the CPU rather than static sequences of instructions as determined by memory layout. It folds branch prediction into the cache.

LRU = last recently used

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 does not fill up)
- *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 do not result in writes (so are simpler and cheaper)
  - Write-back: repeated writes require only one write to lower level

Read misses do not result in writes because in the write-back we need to evict and write the block to the lower level before updating the block in the cache

Improving Cache Performance

0. 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

1. Reduce the miss rate

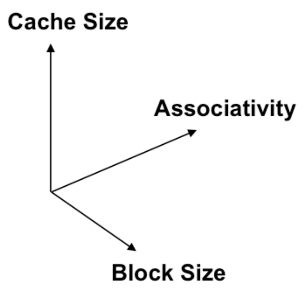
- bigger cache
- more flexible placement (increase associativity)
- larger blocks (16 to 64 bytes typical)
- victim cache – small buffer holding most recently discarded blocks

Improving Cache Performance

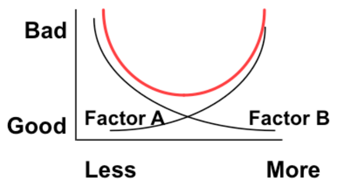
2. Reduce the miss penalty
   - smaller blocks
   - use a write buffer to hold dirty blocks being replaced so do not have to wait for the write to complete before reading
   - check write buffer (and/or victim cache) on read miss – may get lucky
   - for large blocks fetch critical word first
   - use 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

No fancy replacement policy is needed for the direct mapped cache.

As a matter of fact, that is what causes direct mapped trouble to begin with: only one place to go in the cache--causes conflict misses.