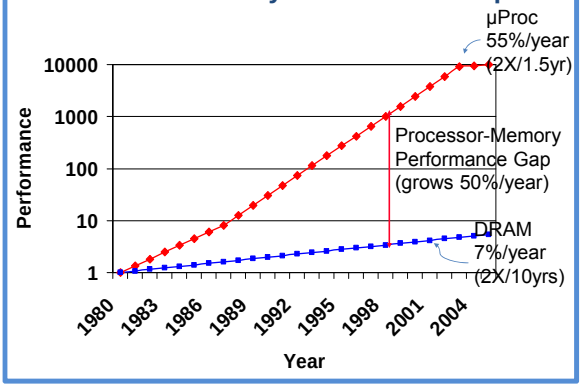
ECEN 4593
Computer Organization

Memory Hierarchy - 1

Andrew Pleszkun

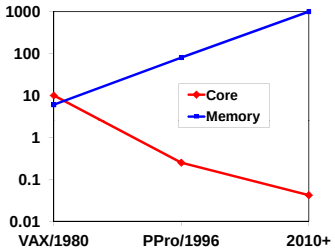
Some material taken from notes prepared by
Prof. Mary Jane Irwin - PSU

Process-Memory Performance Gap



The "Memory Wall"

- Processor vs DRAM speed disparity continues to grow



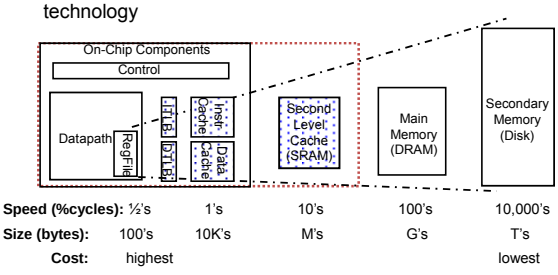
- Good memory hierarchy (cache) design is increasingly important to overall performance

The Memory Hierarchy Goal

- Fact: Large memories are slow and fast memories are small
- How do we create a memory that gives the illusion of being large, cheap and fast (most of the time)?
  - With hierarchy
  - With parallelism

A Typical 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



Memory Hierarchy Technologies

- Caches use *SRAM* for speed and technology compatibility
  - Fast (typical access times of 0.5 to 2.5 nsec)
  - Low density (6 transistor cells), higher power, expensive ($500 to $1000 per GB in 2012)
  - Static: content will last "forever" (as long as power is left on)
- Main memory uses *DRAM* for size (density)
  - Slower (typical access times of 50 to 70 nsec)
  - High density (1 transistor cells), lower power, cheaper ($10 to $20 per GB in 2012)
  - Dynamic: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - *RAS* or *Row Access Strobe* triggering the row decoder
    - *CAS* or *Column Access Strobe* triggering the column selector

The Memory Hierarchy: Why Does it Work?

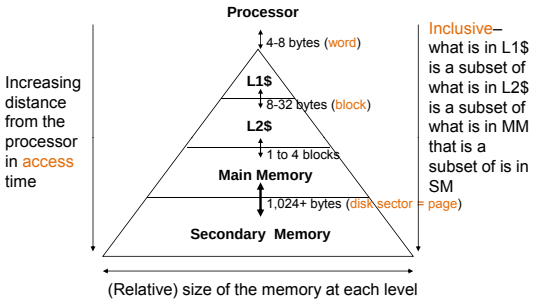
- Temporal Locality (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon
  - ⇒ Keep most recently accessed data items closer to the processor
- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon
  - ⇒ Move blocks consisting of contiguous words closer to the processor

The Memory Hierarchy: 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
  - Hit Time: Time to access that level which consists of
    Time to access the block + Time to determine hit/miss
- Miss Rate: the fraction of memory accesses *not* found in a level of the memory hierarchy ⇒ 1 - (Hit Rate)
  - Miss Penalty: Time to replace a block in that level with the corresponding block from a lower level which consists of
    Time to access the block in the lower level + Time to transmit that block to the level that experienced the miss + Time to insert the block in that level + Time to pass the block to the requestor

Hit Time << Miss Penalty

Characteristics of the Memory Hierarchy



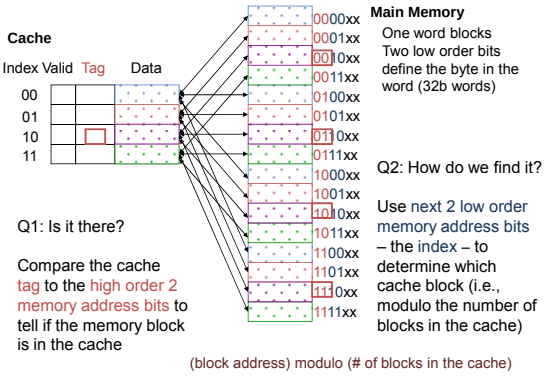
How is the Hierarchy Managed?

- 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 Basics

- 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
  - Each memory block is mapped to exactly one block in the cache
    - lots of lower level blocks must share blocks in the cache
  - Address mapping (to answer Q2):
    (block address) modulo (# of blocks in the cache)
  - Have a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (to answer Q1)

Caching: A Simple First Example



Direct Mapped Cache

- 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	Mem(4)
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)

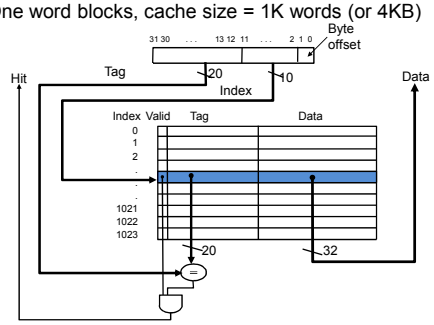
15 miss

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

- 8 requests, 6 misses

MIPS Direct Mapped Cache Example

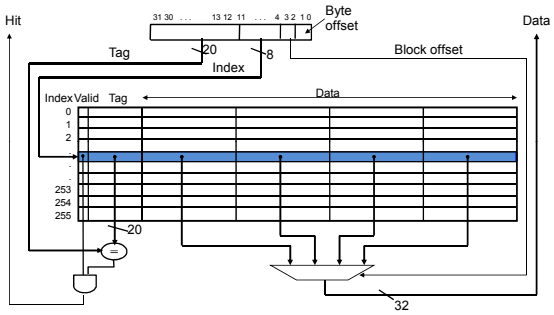
- One word blocks, cache size = 1K words (or 4KB)



What kind of locality are we taking advantage of?

Multiword Block Direct Mapped Cache

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



What kind of locality are we taking advantage of?

Taking Advantage of Spatial Locality

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

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

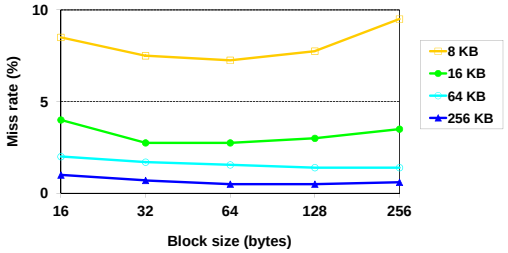
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

- 8 requests, 4 misses

Block Size Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- But in a fixed-sized cache
  - Larger blocks ⇒ fewer of them
    - More competition ⇒ increased miss rate
  - Larger blocks ⇒ pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help

Miss Rate vs Block Size vs Cache Size



- Miss rate goes up if the block size becomes a significant fraction of the cache size because the number of blocks that can be held in the same size cache is smaller (increasing capacity misses)

Cache Field Sizes

- The number of bits in a cache includes both the storage for data and for the tags
  - 32-bit byte address
  - For a direct mapped cache with 2^n blocks, *n* bits are used for the index
  - For a block size of 2^m words (2^{m+2} bytes), *m* bits are used to address the word within the block and 2 bits are used to address the byte within the word
- What is the size of the tag field?
- The total number of bits in a direct-mapped cache is then

  2^n x (block size + tag field size + valid field size)
- How many total bits are required for a direct mapped cache with 16KB of data and 4-word blocks assuming a 32-bit address?

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!
- Write hits (D$ only)
  - 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)
    - writes run at the speed of the next level in the memory hierarchy – this is slow – or can use a write buffer and stall only if the write buffer is full
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache block 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 – can use a write buffer to help "buffer" write-backs of dirty blocks

Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- Write misses (D$ 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
  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
  3. No-write allocate – skip the cache write (but must invalidate that cache block since it will now hold stale data) 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

Multiword Block Considerations

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Early restart – processor resumes execution as soon as the requested word of the block is returned
    - Requested word first – requested word is transferred from the memory to the cache (and processor) first
  - Nonblocking cache – allows the processor to continue to access the cache while the cache is handling an earlier miss
- Write misses (D$)
  - If using write allocate must *first* fetch the block from memory and then write the word to the block (or could end up with a "garbled" block in the cache (e.g., for 4 word blocks, a new tag, one word of data from the new block, and three words of data from the old block)

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

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)

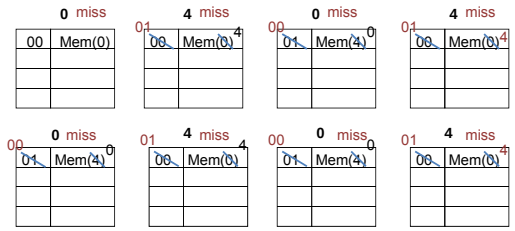
  (block address) modulo (# sets in the cache)

Another Reference String Mapping

- Consider the main memory word reference string

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

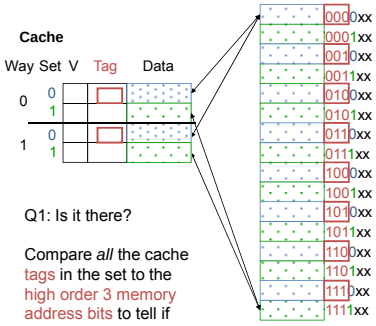
0 4 0 4 0 4 0 4



- 8 requests, 8 misses

- Ping pong effect due to conflict misses - two memory locations that map into the same cache block

Set Associative Cache Example



Main Memory

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

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?

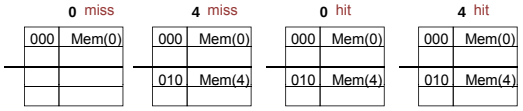
Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

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

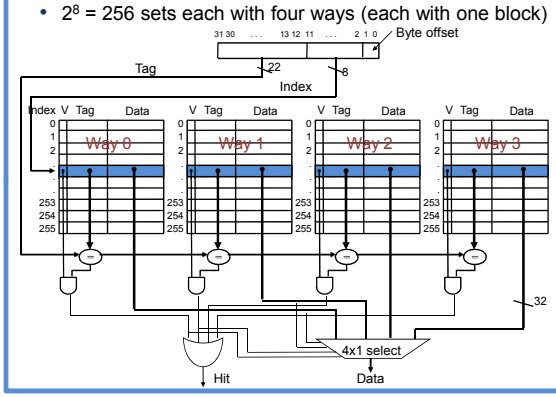


- 8 requests, 2 misses

- 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!

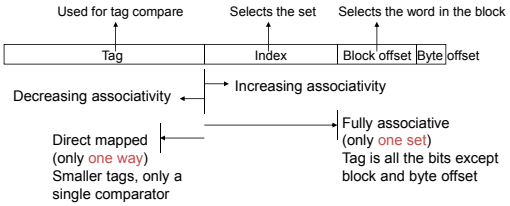
Four-Way Set Associative Cache

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



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

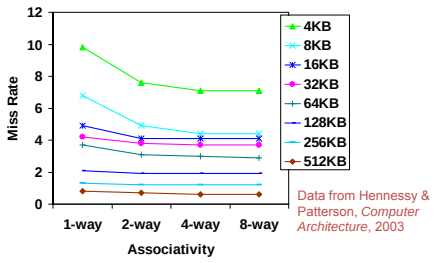


Costs of Set Associative Caches

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

Benefits of Set Associative Caches

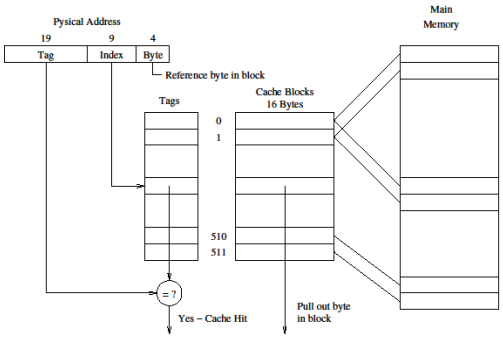
- The choice of direct mapped or set associative depends on the cost of a miss versus the cost of implementation



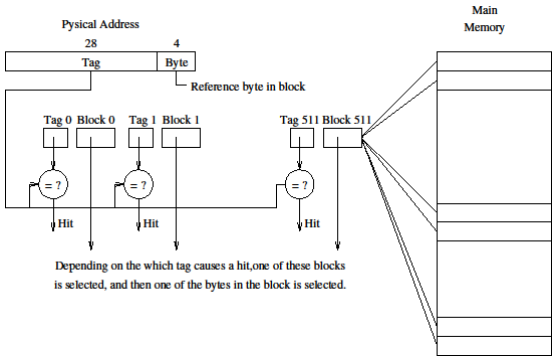
Data from Hennessy & Patterson, *Computer Architecture*, 2003

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

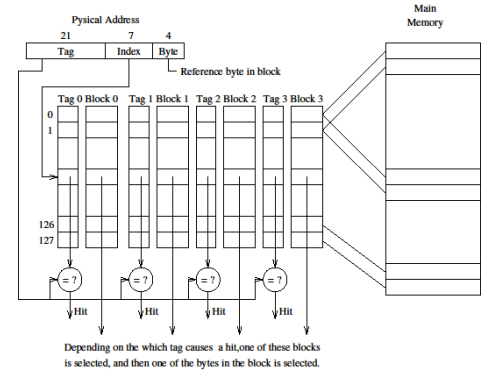
8KB Direct Mapped Cache – 16 byte block



8KB Fully Associative Cache – 16 byte block



8KB 4-way Set Associative Cache – 16B block



Victim Cache

- Observation: Not all sets need a high degree of associativity.
- The Victim Cache (VC) is a small fully associative cache placed between two levels of the cache hierarchy.
- In a sense, the VC extends the associativity of a cache.
- An item removed from a cache level is placed in the VC.
- On a miss, the VC is first checked before making a request to the next level of the hierarchy.
- The VC essentially reduces the miss time since on a hit in the VC, there is no need to go to the next level of the memory hierarchy.
- The VC is implemented as a fully associative cache using an LRU replacement policy.
- When the VC full, the LRU item is only written to the next memory level if it is dirty.