Cache Architecture

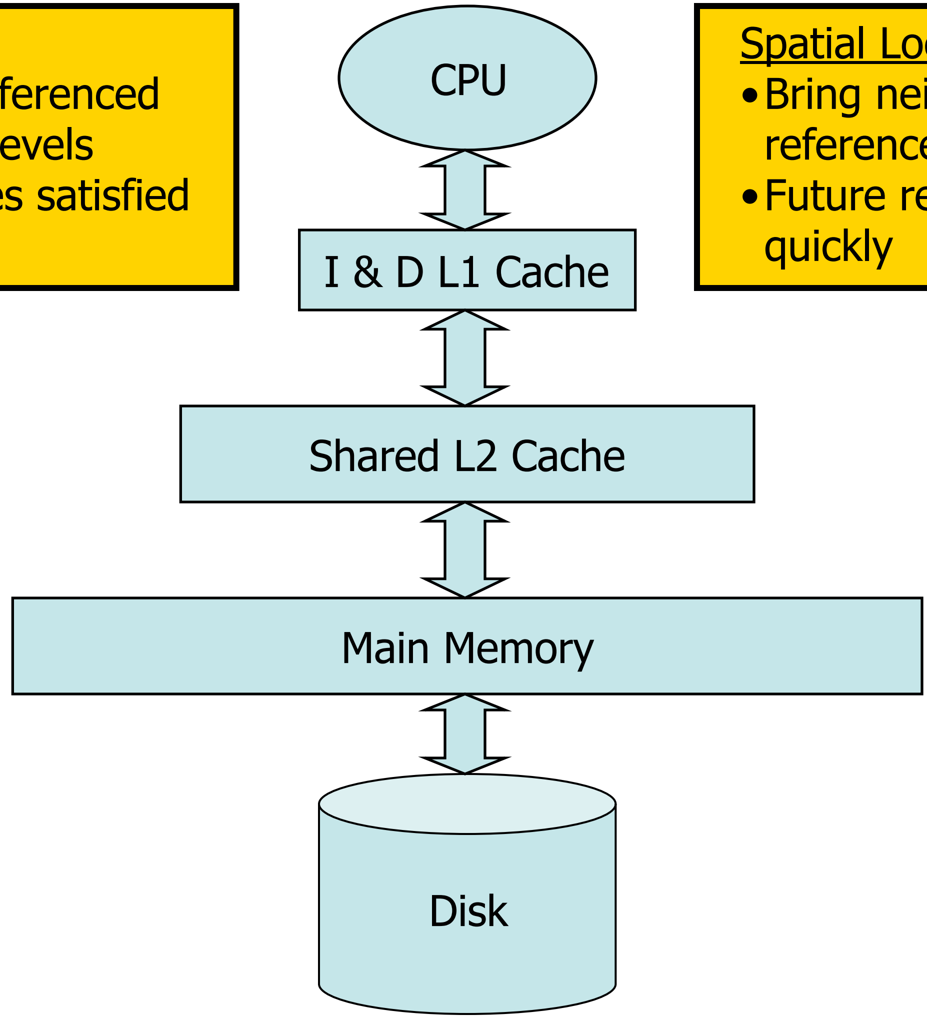
Virendra Singh
Associate Professor
Computer Architecture and Dependable Systems Lab
Department of Electrical Engineering
Indian Institute of Technology Bombay
http://www.ee.iitb.ac.in/~viren/
E-mail: viren@ee.iitb.ac.in

CS-683: Advanced Computer Architecture

Lecture 7 (16 Aug 2013)

Memory Hierarchy

Temporal Locality
- Keep recently referenced items at higher levels
- Future references satisfied quickly



Spatial Locality
- Bring neighbors of recently referenced to higher levels
- Future references satisfied quickly

Four Burning Questions

- These are:
  - Placement
    - Where can a block of memory go?
  - Identification
    - How do I find a block of memory?
  - Replacement
    - How do I make space for new blocks?
  - Write Policy
    - How do I propagate changes?
- Consider these for caches
  - Usually SRAM

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
		0
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
		0

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss

Tag Array

Tag0	Tag1	LRU
		0
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss

Tag Array

Tag0	Tag1	LRU
10		1
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss

Tag Array

Tag0	Tag1	LRU
10	11	0
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict

Tag Array

Tag0	Tag1	LRU
01	11	1
		0
10		1
11		1

Cache Example

- 32B Cache: <BS=4,S=4,B=8>
  - o=2, i=2, t=2; 2-way set-associative
  - Initially empty
  - Only tag array shown on right
- Trace execution of:

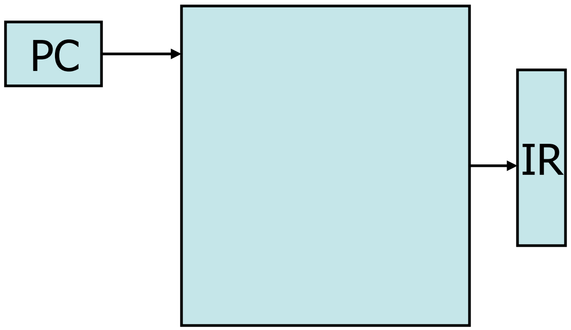
Reference	Binary	Set/Way	Hit/Miss
Load 0x2A	101010	2/0	Miss
Load 0x2B	101011	2/0	Hit
Load 0x3C	111100	3/0	Miss
Load 0x20	100000	0/0	Miss
Load 0x33	110011	0/1	Miss
Load 0x11	010001	0/0 (lru)	Miss/Evict
Store 0x29	101001	2/0	Hit/Dirty

Tag Array

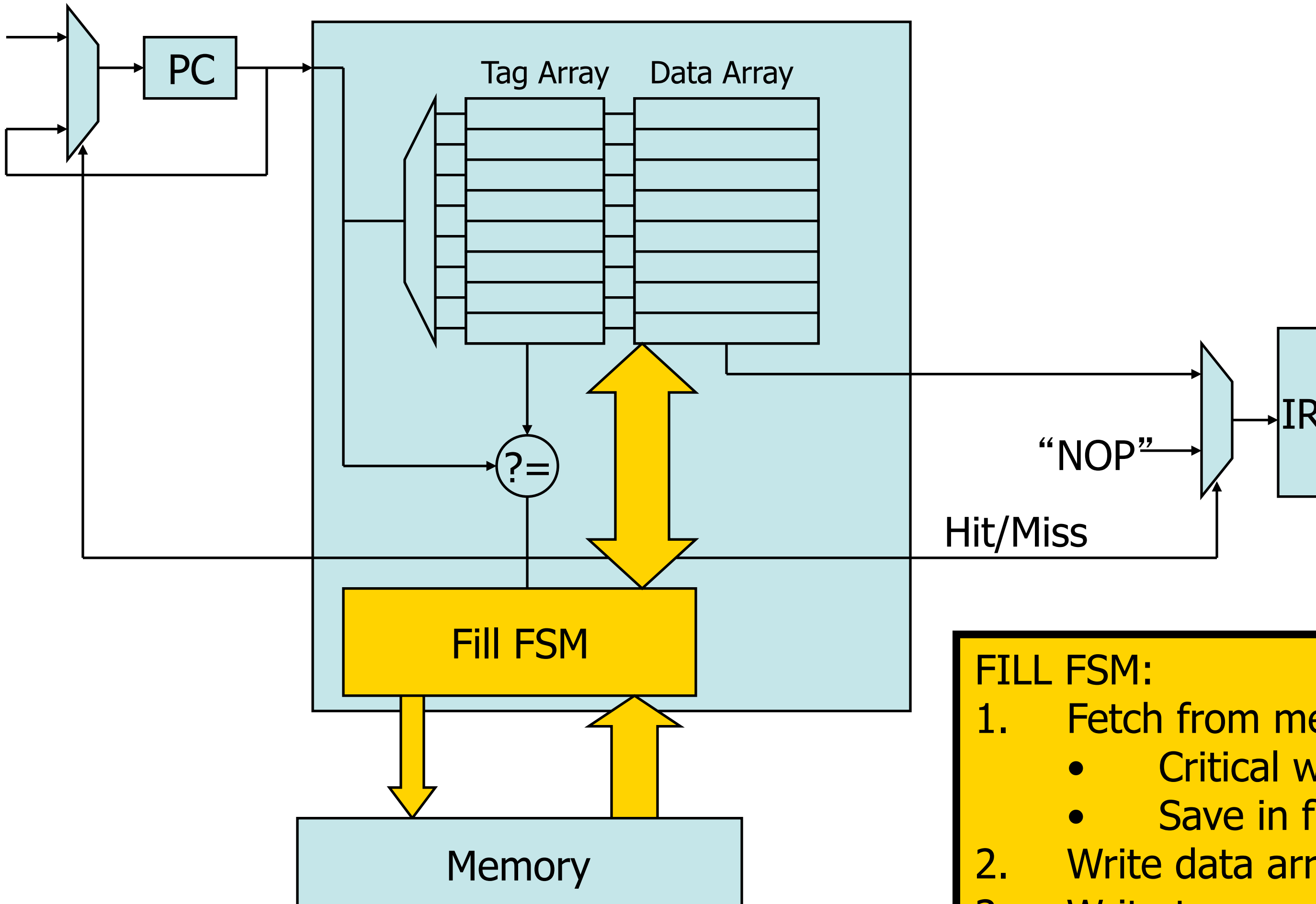
Tag0	Tag1	LRU
01	11	1
		0
10 d		1
11		1

Caches and Pipelining

- Instruction cache
  - No writes, so simpler
- Interface to pipeline:
  - Fetch address (from PC)
  - Supply instruction (to IR)
- What happens on a miss?
  - Stall pipeline; inject nop
  - Initiate cache fill from memory
  - Supply requested instruction, end stall condition



I-Caches and Pipelining



FILL FSM:
1. Fetch from memory
   - Critical word first
   - Save in fill buffer
2. Write data array
3. Write tag array
4. Miss condition ends

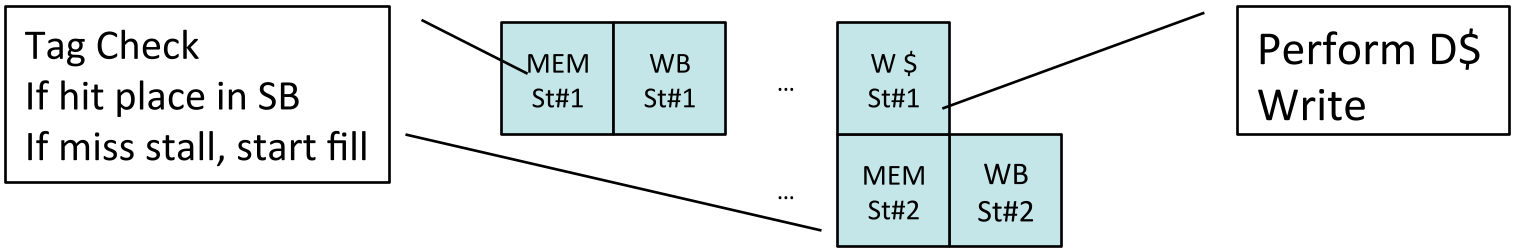
D-Caches and Pipelining

- Pipelining loads from cache
  - Hit/Miss signal from cache
  - Stalls pipeline or inject NOPs?
    - Hard to do in current real designs, since wires are too slow for global stall signals
  - Instead, treat more like branch misprediction
    - Cancel/flush pipeline
    - Restart when cache fill logic is done

D-Caches and Pipelining

- Stores more difficult
  - MEM stage:
    - Perform tag check
    - Only enable write on a hit
    - On a miss, must not write (data corruption)
  - Problem:
    - Must do tag check and data array access sequentially
    - This will hurt cycle time or force extra pipeline stage
    - Extra pipeline stage delays loads as well: IPC hit!

Solution: Pipelining Writes



- Store #1 performs tag check only in MEM stage
  - <value, address, cache way> placed in store buffer (SB)
- When store #2 reaches MEM stage
  - Store #1 writes to data cache
- In the meantime, must handle RAW to store buffer
  - Pending write in SB to address A
  - Newer loads must check SB for conflict
  - Stall/flush SB, or forward directly from SB
- Any load miss must also flush SB first
  - Otherwise SB D$ write may be to wrong line
- Can expand to >1 entry to overlap store misses

Performance

CPU execution time = (CPU clock cycles + memory stall cycles) x Clock Cycle time

Memory Stall cycles = Number of misses x miss penalty

= IC x misses/Instruction x miss penalty

=IC x memory access/instruction x miss rate x miss penalty

Performance: Miss

- Miss rate
  - Fraction of cache access that result in a miss
- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Optimization

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

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

- Reducing miss rate
  - Larger block size, larger cache size, higher associativity
- Reducing miss penalty
  - Multi-level caches, read priority over write
- Reducing time to hit in the cache
  - Avoid address translation when indexing caches

Memory Hierarchy Basics

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption

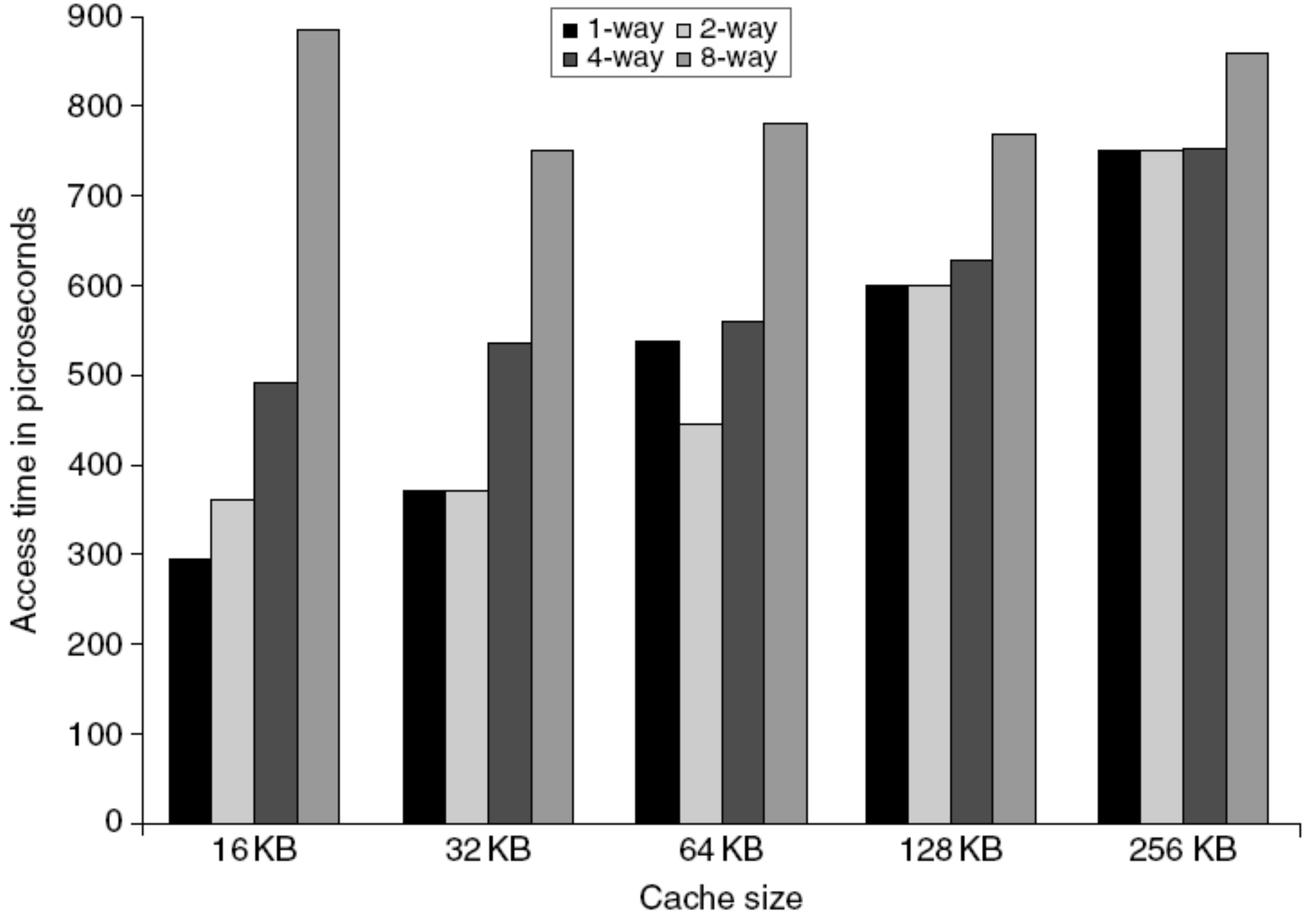
Memory Hierarchy Basics

- Six basic cache optimizations:
  - ➢ Higher number of cache levels
    - Reduces overall memory access time
  - ➢ Giving priority to read misses over writes
    - Reduces miss penalty
  - ➢ Avoiding address translation in cache indexing
    - Reduces hit time

Cache: Advanced Optimizations

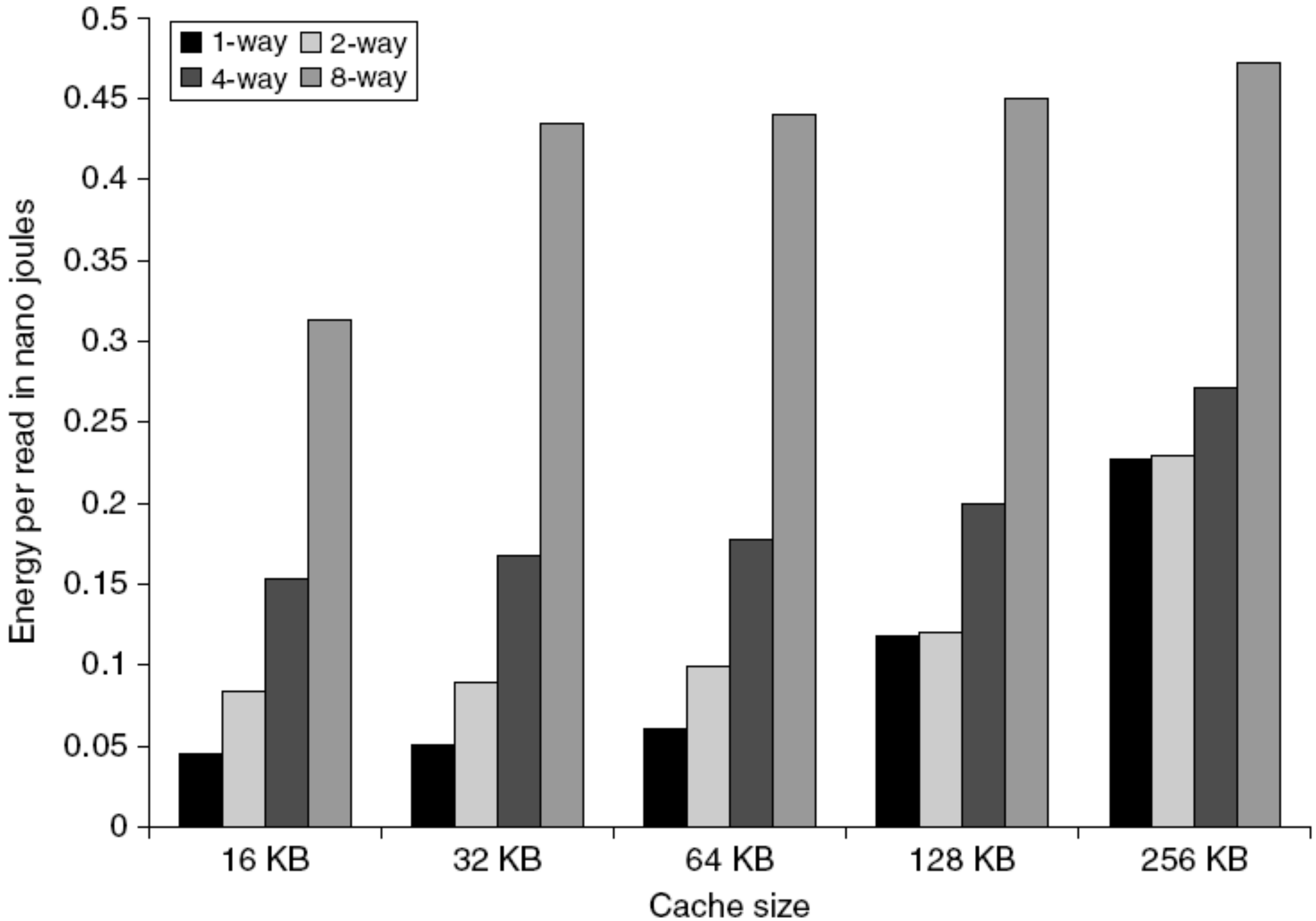
- Small and simple first level caches
  - ➢ Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - ➢ Direct-mapped caches can overlap tag compare and transmission of data
  - ➢ Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

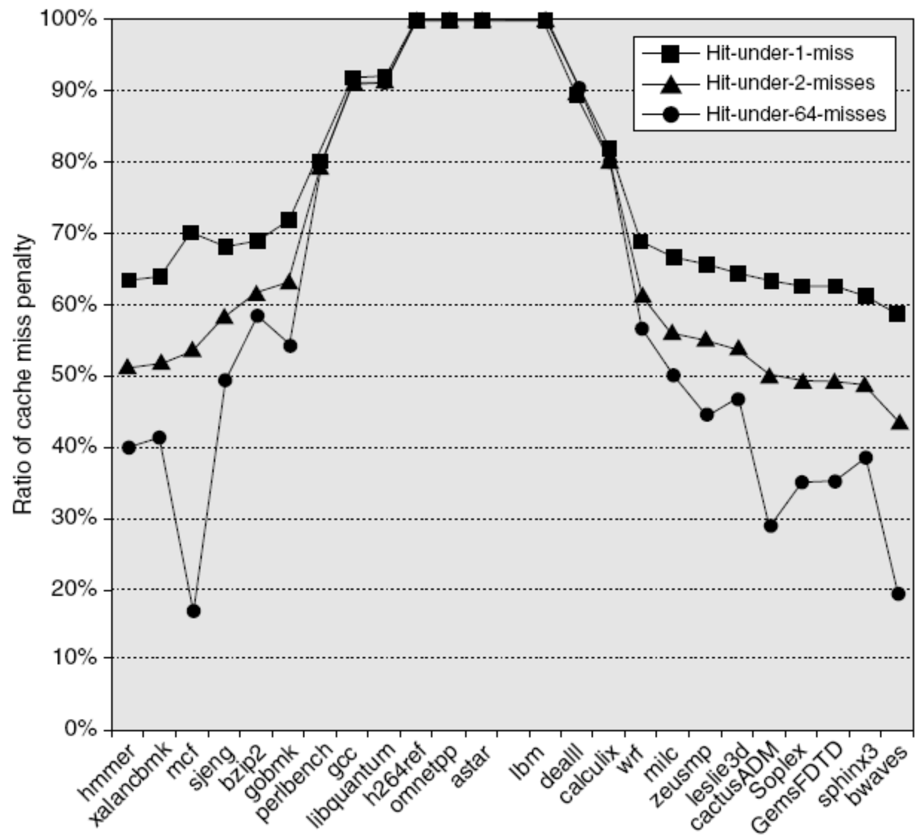
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - “Way selection”
  - Increases mis-prediction penalty

Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Nonblocking Caches

- Allow hits before previous misses complete
  - ✧ "Hit under miss"
  - ✧ "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2

- Interleave banks according to block address

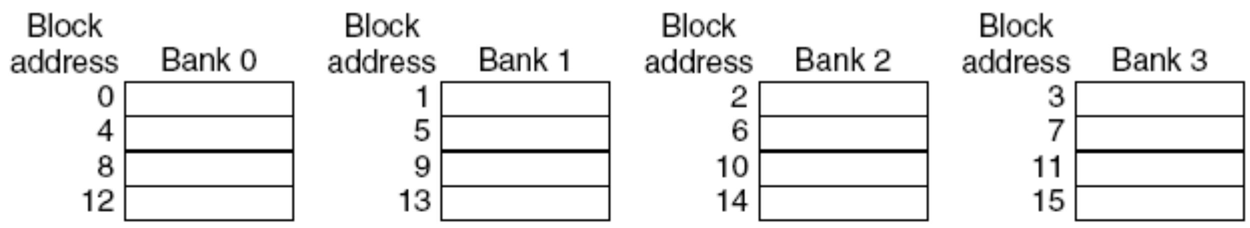


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Summary

- Memory technology
- Memory hierarchy
  - Temporal and spatial locality
- Caches
  - Placement
  - Identification
  - Replacement
  - Write Policy
- Pipeline integration of caches

Thank You