HY225 Lecture 11: Caches and Memory

Dimitrios S. Nikolopoulos

University of Crete and FORTH-ICS

May 16, 2011

Memory Technology

- Static RAM (SRAM)
  - Volatile memory using bistable latching circuitry to store each bit
  - No need to refresh to preserve content
  - Access time: 0.5ns–2.5ns, Cost: $5,000 per GB
- Dynamic RAM (DRAM)
  - Volatile memory using capacitors to store bits of data
  - Need to be refreshed due to leakage current
  - Access time: 50ns–70ns, Cost: $20–$75 per GB
- Solid State Disks (SSD)
  - Non-voliatile memory using solid state materials to store data
  - Access time: $\approx$0.1ms, Cost:$1.5–$12 per GB
- Magnetic disk
  - Non-volatile memory storing data magnetically
  - Access time: 5ms–10ms, Cost: $0.07–$0.16 per GB
- Ideal memory
  - Access time of SRAM
  - Cost of magnetic disk

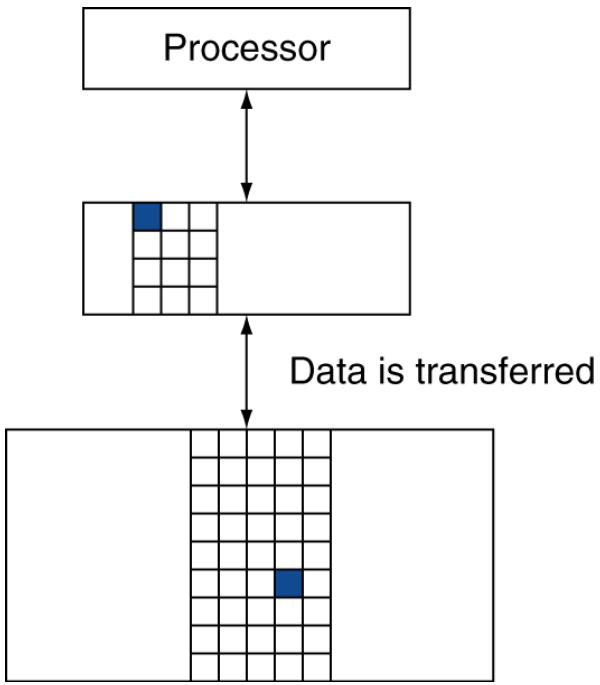
Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently in memory are likely to be accessed soon e.g., sequential instruction access, array data

Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
  - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU

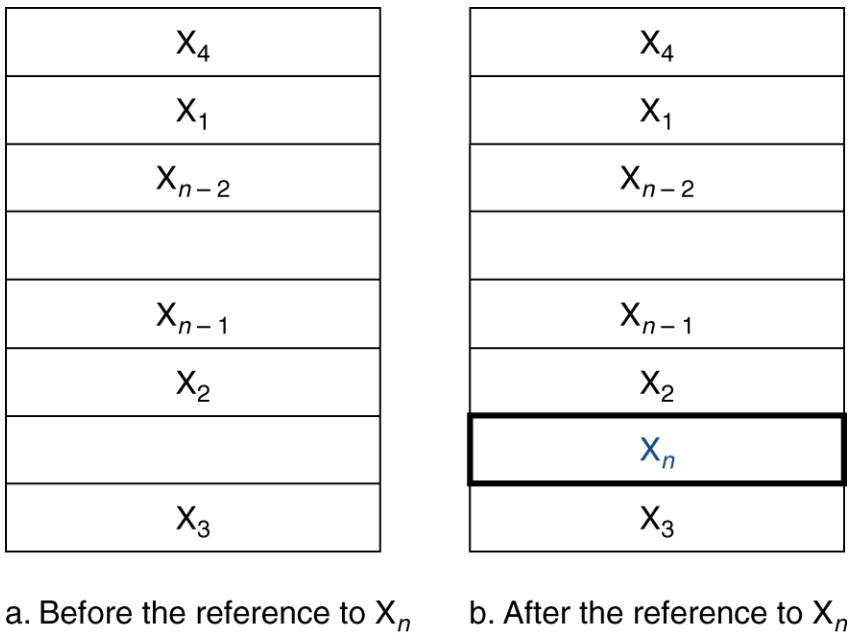
Memory Hierarchy Levels



- Data copied between levels of the memory hierarchy in blocks
  - Block size is typically many words to amortize cost of copying
- If data accessed by processor is present in a level, we have a hit at that level
  - Hit ratio = hits/accesses
- If accessed data is absent, we have a miss at that level
  - Miss requires copying data from lower level
  - Time to copy data is miss penalty
  - Miss ratio = misses/accesses = 1 - Hit ratio
  - Data is eventually accessed from the upper level

Cache Memory

- The level of the memory hierarchy closest to the CPU
- Typically organized in multiple levels, some or all of which are on the same chip with the processor core(s)
- Modern multicore processors have either private per-core cache memories, or shared cache memories, or both

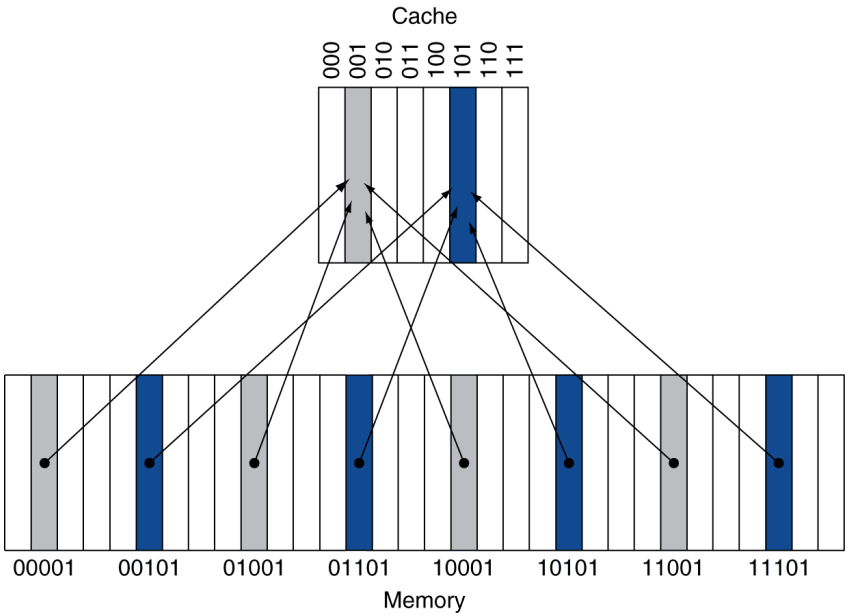


a. Before the reference to X_n b. After the reference to X_n

- Assume an address reference stream $X_0, X_1, \ldots, X_{n-1}, X_n$
- How do we know if data is present?
- Where do we look for the data

Direct-Mapped Cache

- Location determined by address
- Direct mapped: one choice
  - (Block address) modulo (# Blocks in the cache)



- # Blocks is a power of two
- Mapping scheme uses lower bits of addresses

Tag and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Could store block address as well as the data
  - Actually, we only need the high-order bits, called the tag
- What if there is no data in a location?
  - Valid bit: 1 = present, 0 = not present
  - Initially 0

Cache Example

- 8 blocks, 1 word/block, direct-mapped
- Initial state

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	N		
111	N		

Cache Example (Cont.)

Word addr	Binary addr	Hit/miss	Cache block
88	10 110 00	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[1011000]
111	N		

Cache Example (Cont.)

Word addr	Binary addr	Hit/miss	Cache block
104	11 010 00	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[1101000]
011	N		
100	N		
101	N		
110	Y	10	Mem[1011000]
111	N		

Cache Example (Cont.)

Word addr	Binary addr	Hit/miss	Cache block
88	10 110 00	Hit	110
104	11 010 00	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[1101000]
011	N		
100	N		
101	N		
110	Y	10	Mem[1011000]
111	N		

Cache Example (Cont.)

Word addr	Binary addr	Hit/miss	Cache block
64	10 000 00	Miss	000
12	00 011 00	Miss	011
64	10 000 00	Hit	000

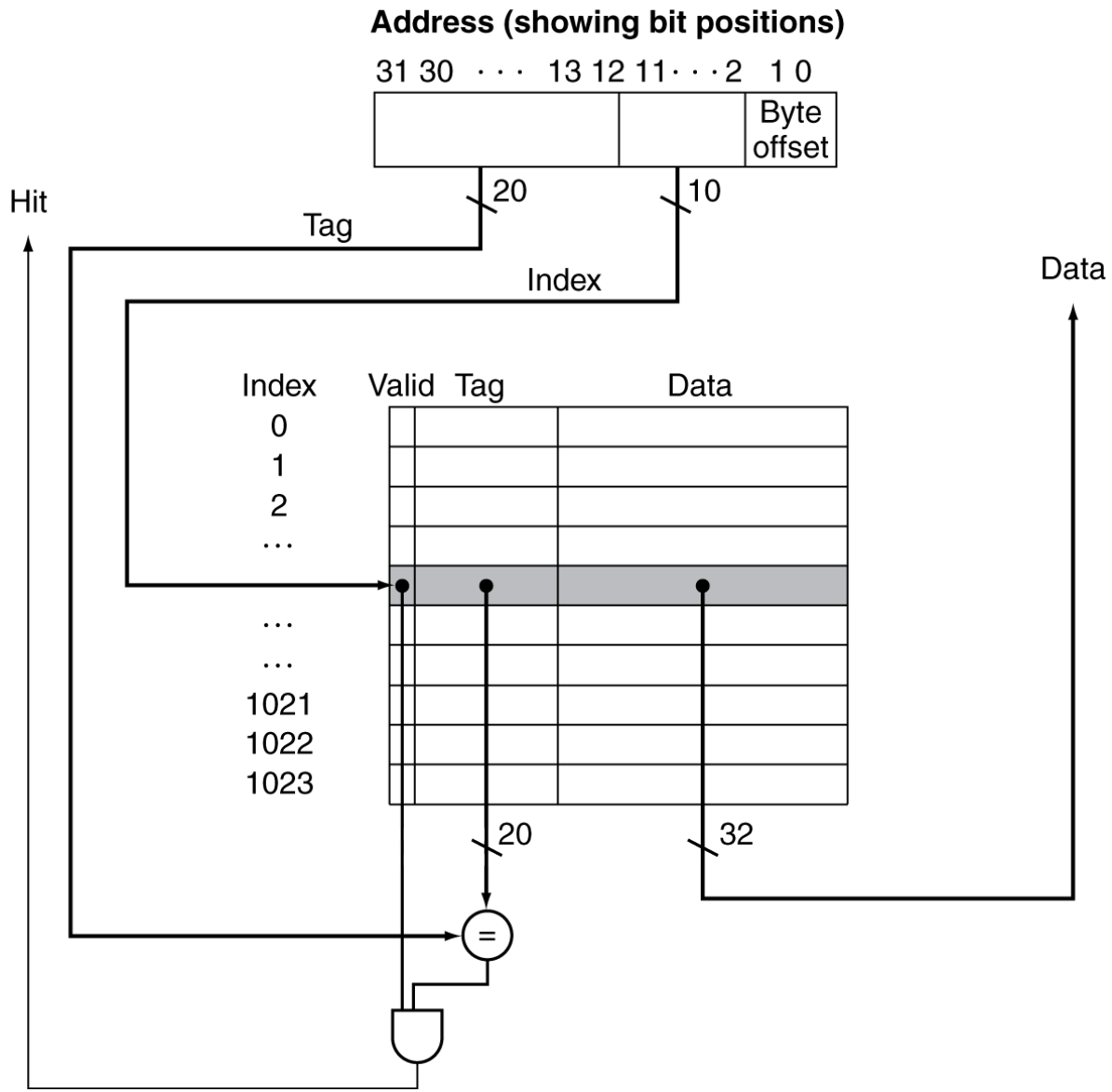
Index	V	Tag	Data
000	Y	10	Mem[1000000]
001	N		
010	Y	11	Mem[1101000]
011	Y	00	Mem[0001100]
100	N		
101	N		
110	Y	10	Mem[1011000]
111	N		

Index	V	Tag	Data
000	Y	10	Mem[1000000]
001	N		

Word addr	Binary addr	Hit/miss	Cache block
72	10 010 00	Miss	010

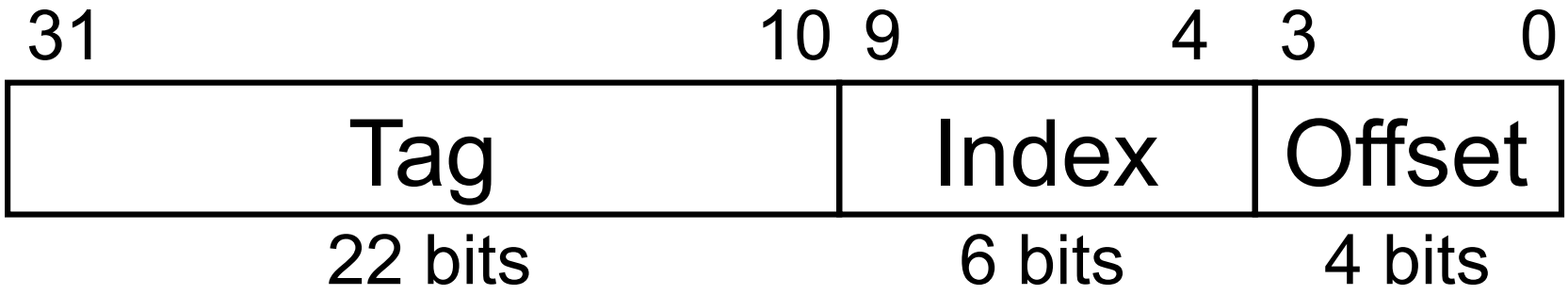
Index	V	Tag	Data
000	Y	10	Mem[1000000]
001	N		
010	Y	11	Mem[1101000]
011	Y	00	Mem[0001100]
100	N		
101	N		
110	Y	10	Mem[1011000]
111	N		

Address Subdivision



Larger Block Size

- 64 blocks, 16 bytes/block
  - Where is word address 4800 mapped?
- Block number = word address / 16 = 300
- Block number = 300 modulo 64 = 44



Block Size Performance Considerations

- Larger blocks should reduce miss rate
  - Due to spatial locality
- Assume fixed-sized cache
  - Larger blocks means fewer blocks
    - More competition between addresses for same block, increased miss rate
  - Larger blocks may introduce pollution
- Larger miss penalty
  - Can override benefit of reduced miss rate
  - Early restart and critical-word-first can help
    - Early restart: Processor continues as soon as it fetches the requested word in a block (even if theare are more words in the block to be fetched
    - Critical-word-first: Processor fetches first requested word and restarts then fetches the rest of the words in the block

Cache misses

- On cache hit, processor proceeds normally
- On cache miss
  - Stall processor pipeline
  - Fetch block from next level of memory hierarchy
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Write-Through

- On data-write hit, could just update block in the cache
  - This means that cache and memory (or lower level of the hierarchy) would be inconsistent
- Write through: update both cache and lower level memory
- Write through makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, and write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1 $\times$ 100 = 11
- Solution: write buffer
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls if write buffer is full

Write-Back

- Alternative: On data-write hit, just update the block in cache
  - Keep track of whether each block is dirty
  - Use a dirty bit per block in the cache
- When a dirty block is replaced
  - Write it back to memory
  - Can use a write buffer to allow replacing block to be read first

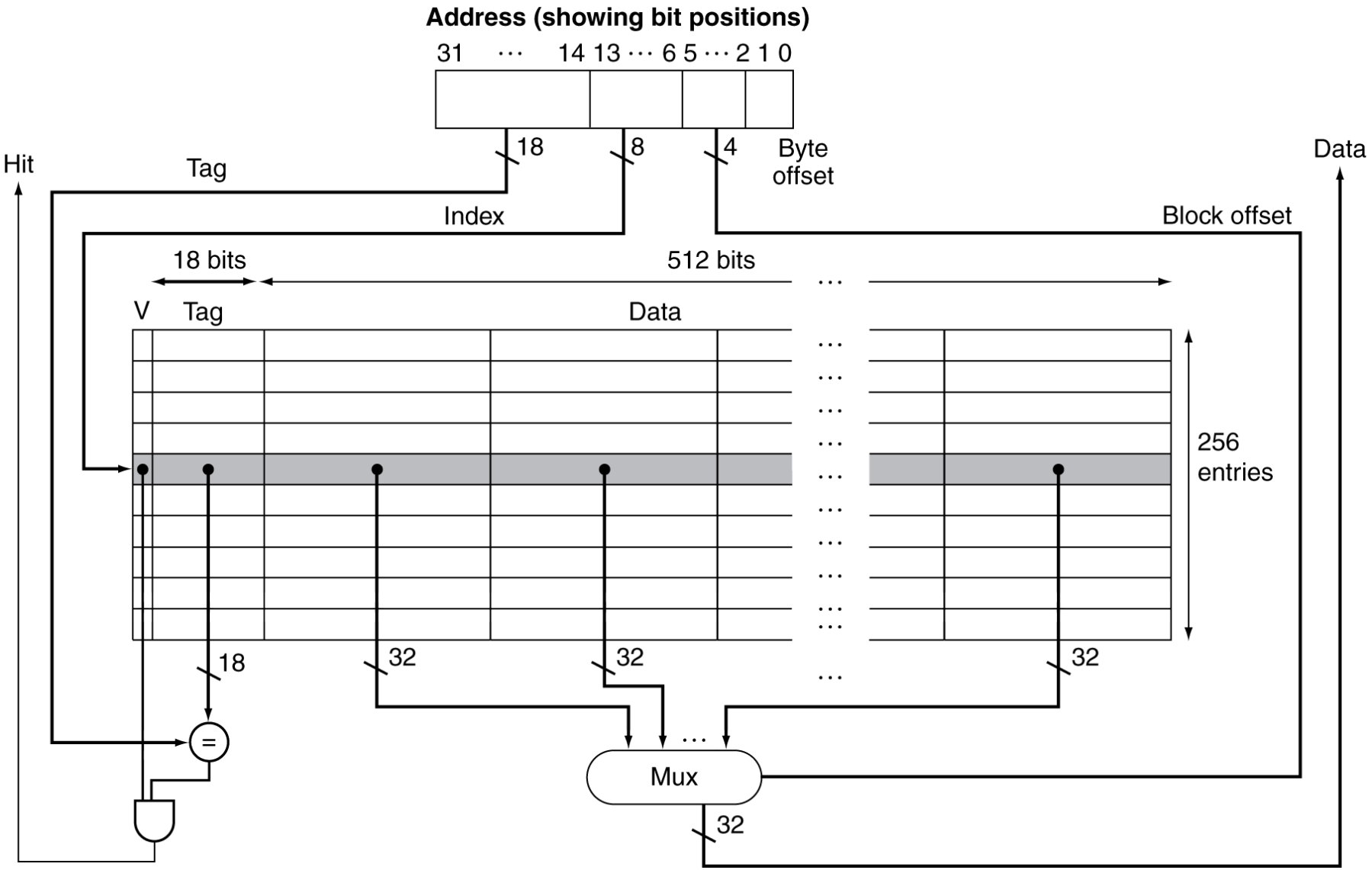
Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
  - Allocate on miss: fetch the block
  - Write around: don't fetch the block
    - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
  - Usually fetch the block

Real Example: Intrinsity FastMath Processor

- Embedded MIPS processor
  - 12-stage pipeline
  - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks $\times$ 16 words/block
  - D-cache: write-through or write-back
- SPEC2000 miss rates
  - I-cache: 0.4%
  - D-cache: 11.4%
  - Weighted average: 3.2%

Real Example: Intrinsity FastMath Processor



Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time, assumed one cycle so far
  - Memory stall cycles
    - Time wasted due to misses in one or more levels of the memory hierarchy
- Simplifying assumptions:

$$\text{Memory stall cycles} =$$

$$\frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} =$$

$$\frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

Cache Performance Example

- Given:
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Loads and stores are 36% of instructions
- Miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- Actual CPI = 2 + 1.44 + 2 = 5.44
  - Ideal CPU is 5.44/2 = 2.72 times faster

Average Access Time

- Average memory acess time affected also by hit time, not only miss penalty
- Average Memory Access Time (AMAT)
  - AMAT = Hit Time + Miss Rate $\times$ Miss Penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, overall cache miss rate = 5%
  - AMAT = 1 + 0.05 $\times$ 20 = 2ns = 2 cycles per instruction

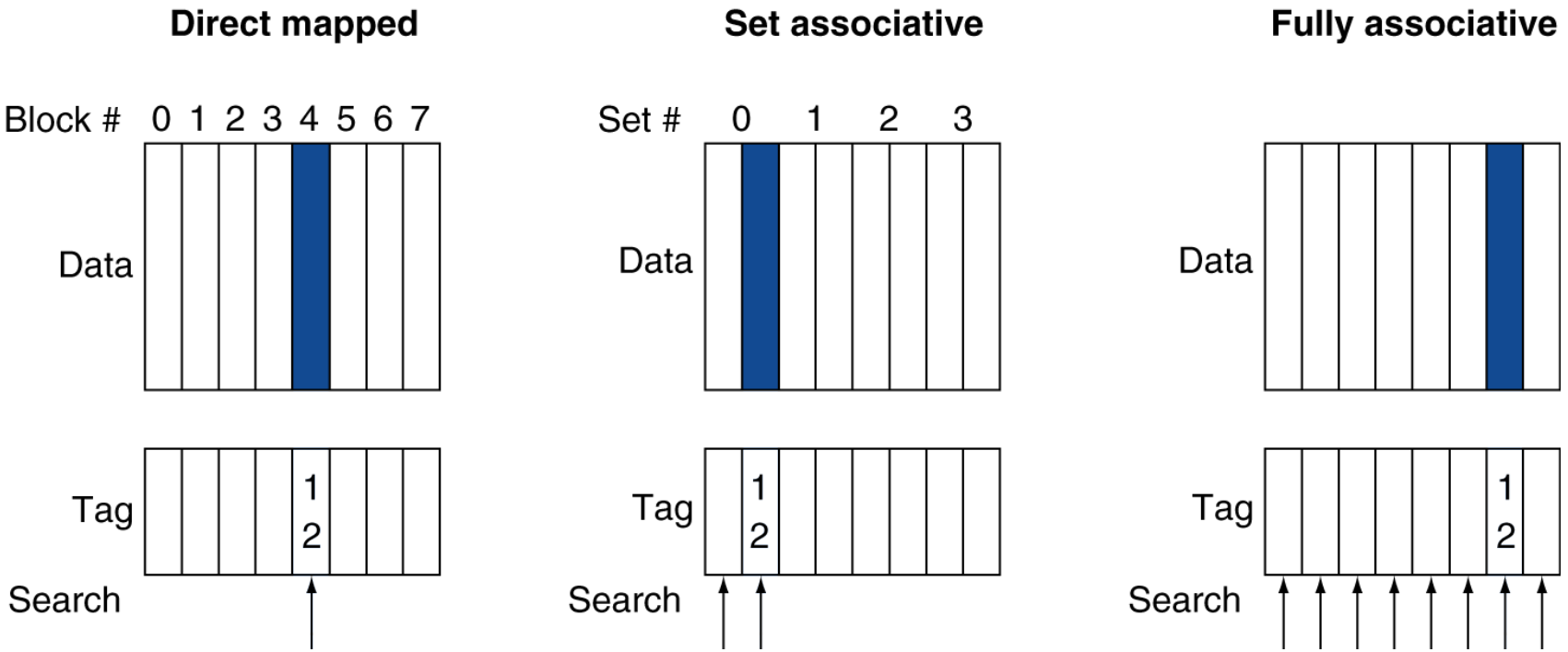
Cache Performance Summary

- Higher processor performance makes miss penalty more significant
- Decreasing processor base CPI
  - Greater portion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Cache performance is critical for system performance

Associative Caches

- Direct-mapped caches may incur frequent conflicts
  - Assume 2^{32} addresses, 2^{10} cache blocks, 2^3 words per cache block
  - Memory stores up to $2^{27} = 2^{32-2-3}$ blocks of 2^3 words each
  - $2^{27-10} = 2^{17}$ memory blocks map to the same cache block
- Fully associative cache
  - Allow a memory block to map to any cache entry
  - Requires all entries to be searched at once, to find at least one invalid
  - Comparator per entry (expensive)
- n-way associatve cache
  - Allow a memory block to map to any of n entries in a set
  - Organize cache in sets of blocks with n blocks each
  - Block number defines which set block is mapped to
    - (Block number) modulo (#sets in the cache)
  - Cache searches all entries in a given set at once
  - n comparators (less expensive)

Associative Cache Examples



Associativity Spectrum

One-way set associative
(direct mapped)

Block	Tag	Data
0		
1		
2		
3		
4		
5		
6		
7		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				
2				
3				

Four-way set associative

Set	Tag	Data	Tag	Data	Tag	Data	Tag	Data
0								
1								

Eight-way set associative (fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data	Tag	Data

Associativity Example

- Compare caches with 4 blocks
  - Direct mapped, 2-way associative, fully associative
- Block access sequence: 0, 8, 0, 6, 8

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example (Cont.)

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

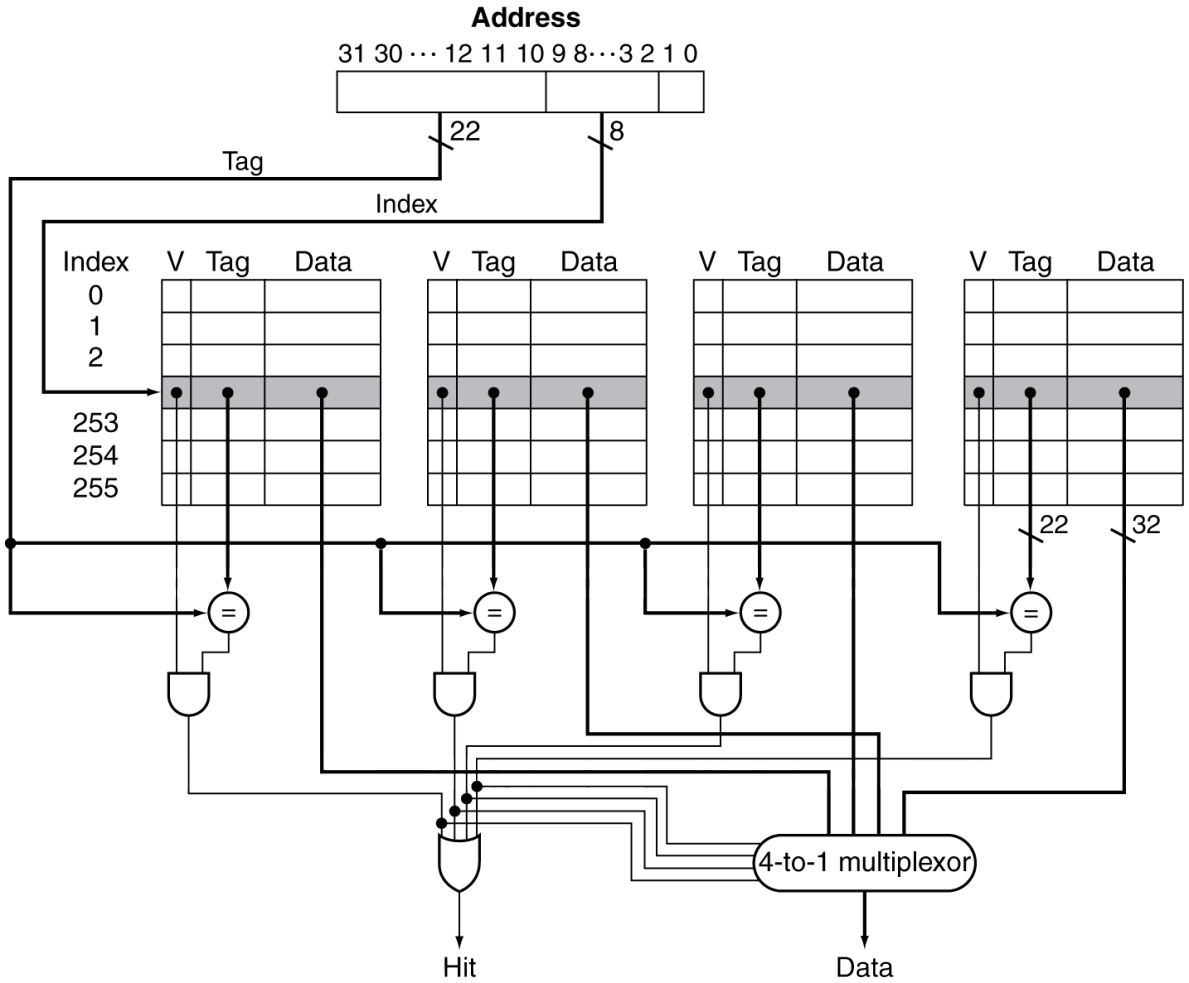
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity?

- Higher associativity may decrease miss rate
  - Associative caches reduce conflicts between blocks mapping to the same space in the cache
  - Can not decrease miss rate beyond a certain hard lower bound
  - Programs have compulsory cold-start misses and capacity misses
    - Compulsory misses happen at program start-up, when cache is empty
    - Capacity misses happen because program data do not fit in the cache
    - A block brought in the cache, then replaced, then rebrought causes a capacity miss
- Three C's model: compulsory, capacity, conflict misses

Set-Associative Cache Organization



- Set-associative caches have higher hit times than direct-mapped caches due to latency of multiplexer and block selection

Set Associative Cache Performance Example

Direct-mapped vs. set-associative cache

- $CPI_{execution}$ = 2.0
- 64 KB caches with 64-byte blocks
- 1.5 memory references per instruction
- Direct mapped cache miss rate = 1.4%
- Set associative cache stretches clock cycle by 1.25, miss rate = 1.0%
- 1 GHz processor
- 75 ns miss penalty
- 1 cycle hit time

$$AMAT_{direct-mapped} = 1.0 + (.014 \times 75) = 2.05ns$$

$$AMAT_{2-way} = 1.0 \times 1.25 + (.01 \times 75) = 2.00ns$$

Set Associative Cache Performance Example

Direct-mapped vs. set-associative cache

$$\text{CPU time} = IC \times \left(CPI_{execution} + \frac{Misses}{Instruction} \times \text{miss penalty} \right) \times \text{clock cycle time}$$

$$\text{CPU time}_{direct-mapped} = IC \times (2.0 \times 1.0 + 0.014 \times 1.5 \times 75) = 3.58 \times IC$$

$$\text{CPU time}_{two-way} = IC \times (2.0 \times 1.25 + 0.01 \times 1.5 \times 75) = 3.63 \times IC$$

- Associative cache achieves lower AMAT than direct-mapped cache
- Direct-mapped cache achieves higher performance than associative cache

Replacement Policy

- Direct-mapped caches have one choice for placing each memory block in the cache – unique mapping
- Associative caches offer a choice to place a memory block in one of several/many cache blocks in a set
- Set associative:
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-Recently Used (LRU)
  - Choose the block which is unused for the longest time
  - Simple to implement for 2-way or 4-way associative (one bit or two bits per block)
  - Complexity grows fast with number of ways – often infeasible for fully-associative caches
- Random
  - Simple to implement
  - Achieves similar performance as LRU for high associativity

Replacement Policy Performance

D-cache misses per 1000 instructions, SPECcpu 2000

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Multi-level Cache Organizations

- Primary (Level-1 or L1) cache
  - Small, simple, to minimize hit time
  - On-chip, attached to the processor
- Secondary (Level-2 or L2) cache
  - Larger than L1, attempting to avoid misses to main memory
  - Hit time has less overall impact on performance
  - On-chip, accessed upon L1 cache misses
- Some systems have also L3 caches, off-chip

Multi-level Cache Performance

- Each level of the cache has:
  - Local hit/miss rate: Hits/misses at that level divided by total accesses at that level
- The whole cache hierarchy has a global hit/miss rate
  - Global miss rate: Number of misses at all levels of the hierarchy (i.e. accesses served from main memory) divided by total number of memory accesses in program
  - Global hit rate: Number of hits at any level of the hierarchy divided by total number of memory accesses in program

Multi-level Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4 GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just L1 cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 $\times$ 400 = 9

Multi-level Cache Example (Cont.)

- Add L2 cache
  - Access time = 5ns
  - L2 local miss rate = 0.5%
- Primary miss with L2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L2 miss
  - Penalty = 20 cycles + 400 cycles = 420 cycles (access to main memory)
- CPI = $1 + 0.02 \times 0.995 \times 20 + 0.02 \times 0.005 \times 420 = 1.38$
- Performance ratio = 9/1.38 = 6.5

Multi-level Cache Alternative Example

- Two-level cache, same assumptions for L1
  - 2% miss rate
  - Miss penalty 20 cycles if L2 cache hit, 420 cycles if L2 cache miss
- Global miss rate = 0.5%
- CPI = $1 + 0.005 \times 420 + 0.02 \times 0.995 \times 20 = 3.48$
- Performance ratio = 9/3.48 = 2.58