

# Computer Architecture

## Chapter 5: Memory Hierarchy

Phạm Quốc Cường

Adapted from Computer Organization the Hardware/Software Interface – 5th



# Principle of Locality

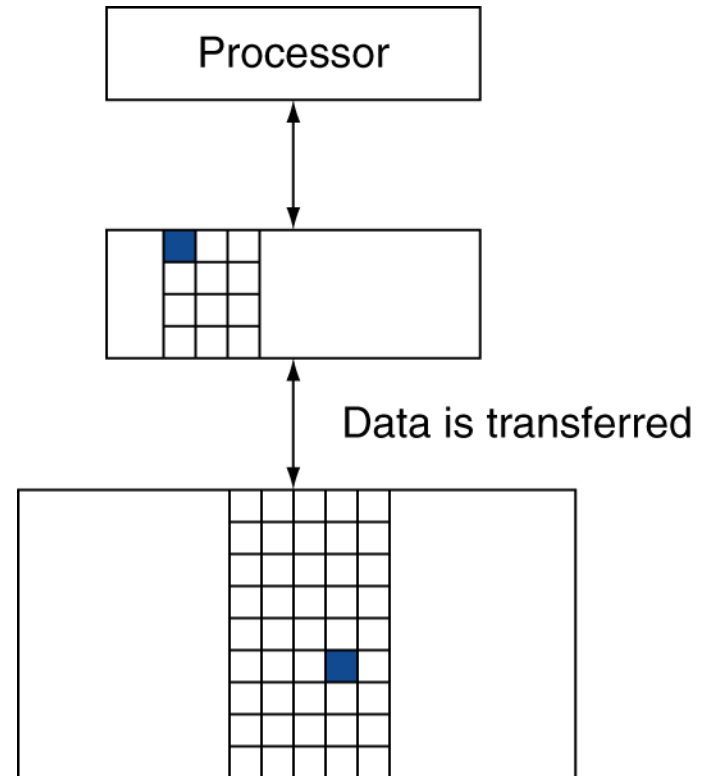
- Programs access a small proportion of their address space at any time
- Temporal locality
  - Items accessed recently are likely to be accessed again soon
  - e.g., instructions in a loop, induction variables
- Spatial locality
  - Items near those accessed recently are likely to be accessed soon
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

- Block (aka line): unit of copying
  - May be multiple words
- If accessed data is present in upper level
  - Hit: access satisfied by upper level
    - Hit ratio: hits/accesses
- If accessed data is absent
  - Miss: block copied from lower level
    - Time taken: miss penalty
    - Miss ratio: misses/accesses  
 $= 1 - \text{hit ratio}$
  - Then accessed data supplied from upper level



# Memory Technology

- Static RAM (SRAM)
  - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM)
  - 50ns – 70ns, \$20 – \$75 per GB
- Flash Memory
  - 5 $\mu$ s – 50 $\mu$ s, \$0.75 - \$1 per GB
- Magnetic disk
  - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
  - Access time of SRAM
  - Capacity and cost/GB of disk

# Cache Memory

- Cache memory
  - The level of the Mem. hierarchy closest to the CPU
- Given accesses  $X_1, \dots, X_{n-1}, X_n$

$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_3$

a. Before the reference to  $X_n$

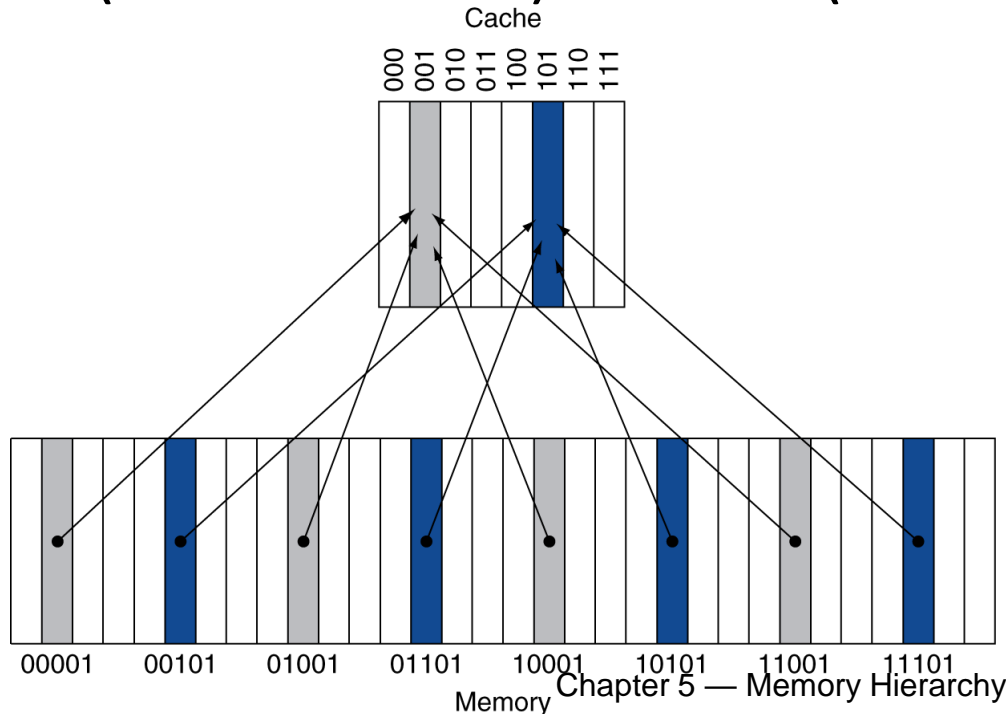
$X_4$
$X_1$
$X_{n-2}$
$X_{n-1}$
$X_2$
$X_n$
$X_3$

b. After the reference to  $X_n$

- How do we know if the data is present?
- Where do we look?

# Direct Mapped Cache

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



- #Blocks is a power of 2
- Use low-order address bits

# Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
  - Store block address as well as the data
  - Actually, 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

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
<b>110</b>	<b>Y</b>	<b>10</b>	<b>Mem[10110]</b>
111	N		

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
<b>010</b>	<b>Y</b>	<b>11</b>	<b>Mem[11010]</b>
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

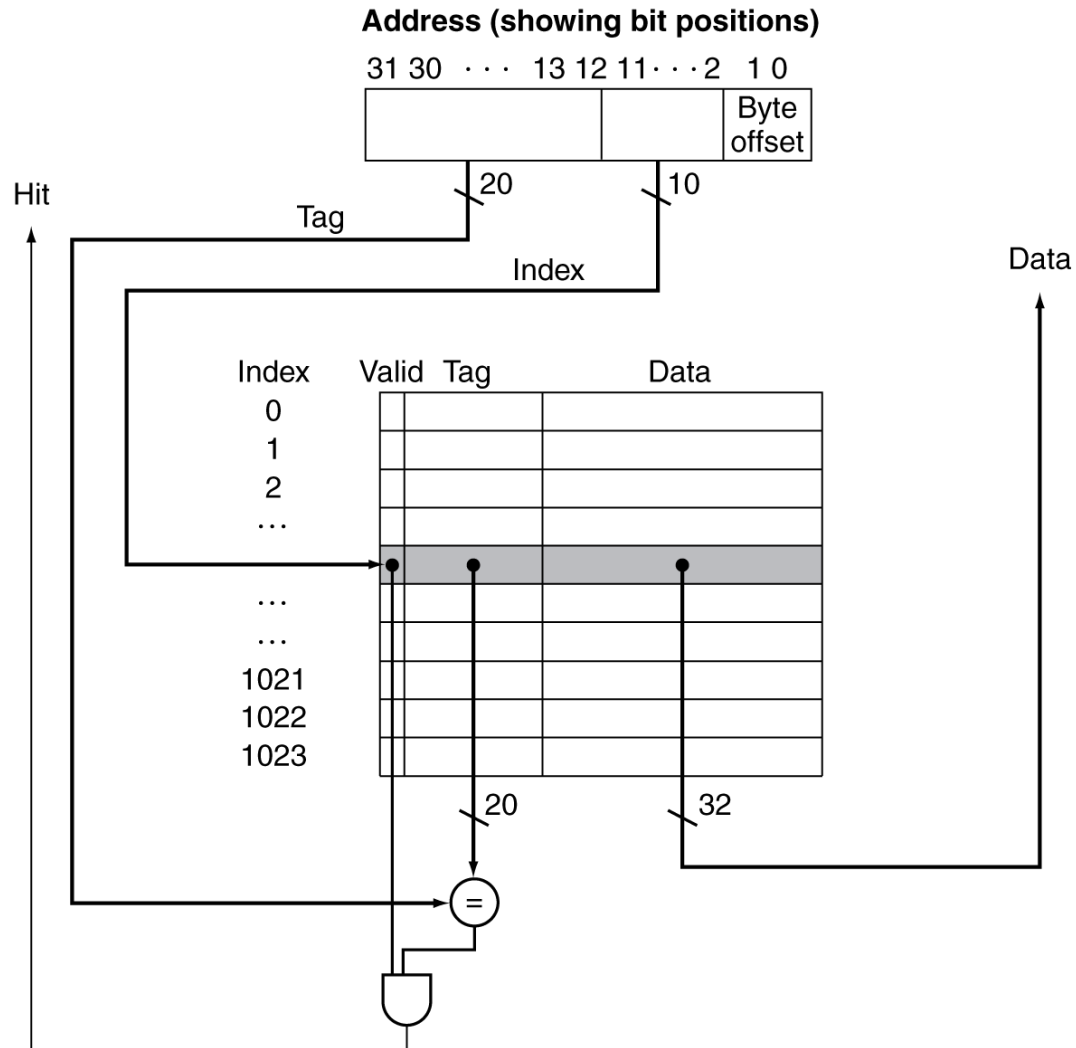
Index	V	Tag	Data
<b>000</b>	<b>Y</b>	<b>10</b>	<b>Mem[10000]</b>
001	N		
010	Y	11	Mem[11010]
<b>011</b>	<b>Y</b>	<b>00</b>	<b>Mem[00011]</b>
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

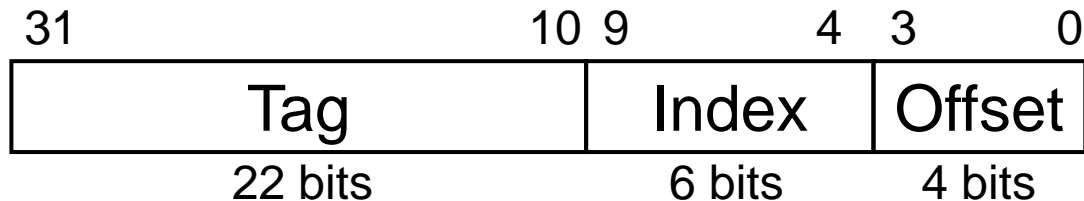
Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
<b>010</b>	<b>Y</b>	<b>10</b>	<b>Mem[10010]</b>
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

# Address Subdivision



# Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what bytenumber does address 1200 map?
- Block address =  $\lfloor 1200/16 \rfloor = 75$
- Block number = 75 modulo 64 = 11





# Block Size Considerations

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

# Cache Misses

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

# Write-Through

- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent
- Write through: also update memory
- But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, 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 on write if write buffer is already full

# Write-Back

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

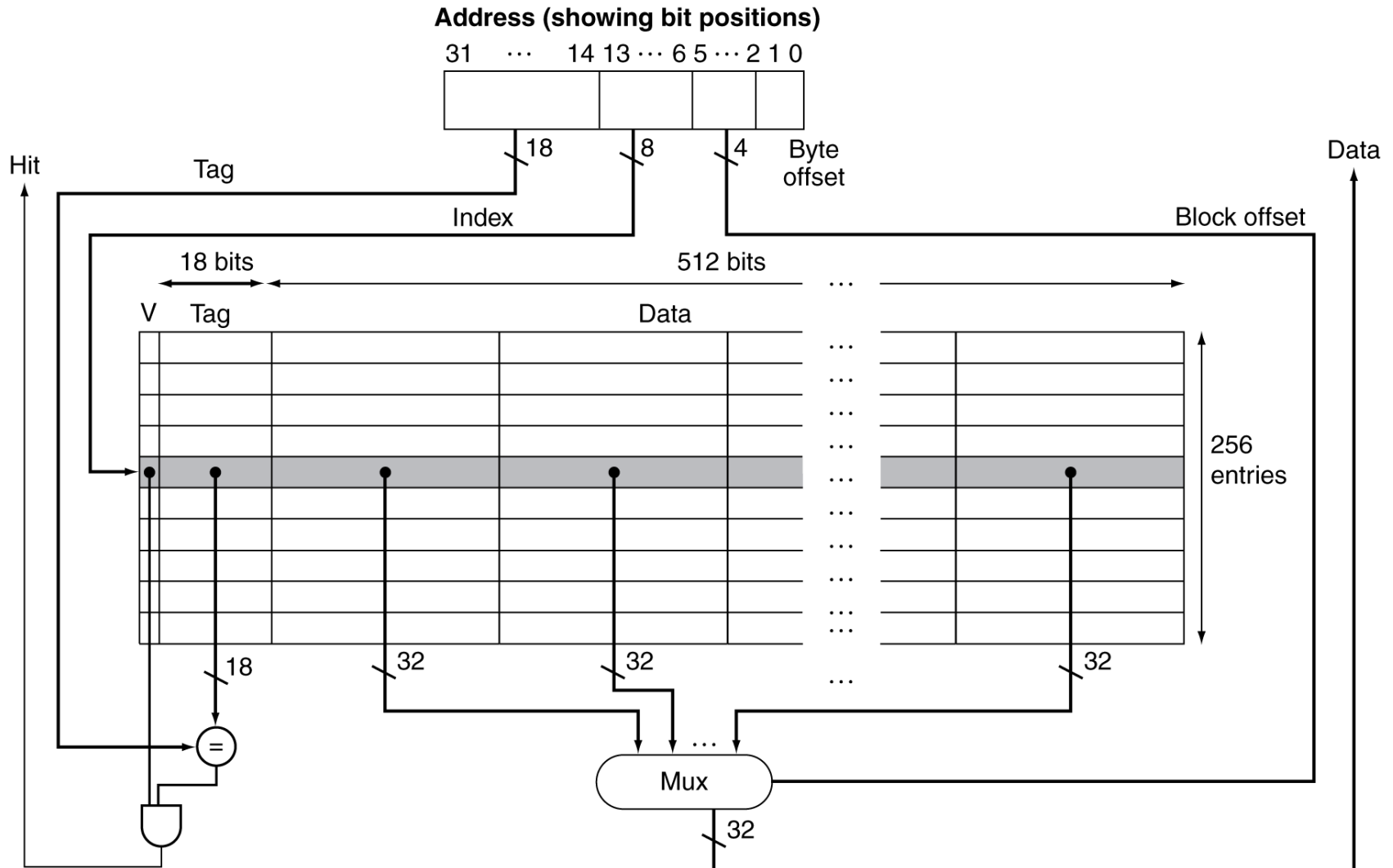
# Write Allocation

- 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

# Example: Intrinsity FastMATH

- 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%

# Example: Intrinsity FastMATH

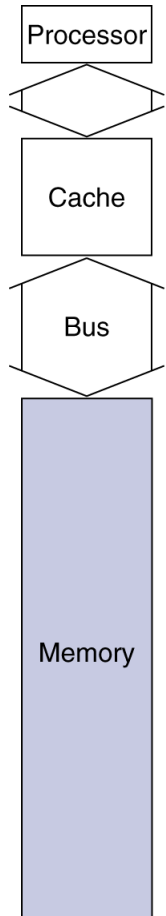


# Main Memory Supporting Caches

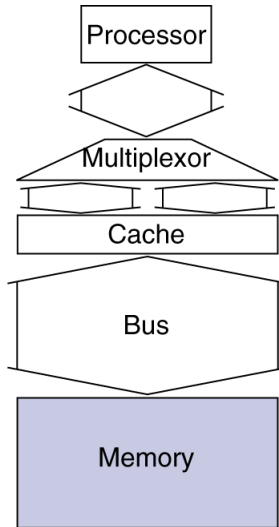
- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty =  $1 + 4 \times 15 + 4 \times 1 = 65$  bus cycles
  - Bandwidth =  $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$



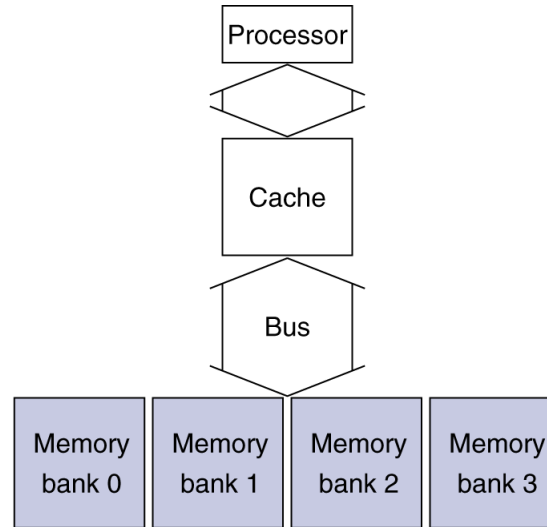
# Increasing Memory Bandwidth



a. One-word-wide memory organization



b. Wider memory organization



c. Interleaved memory organization

- 4-word wide memory
  - Miss penalty =  $1 + 15 + 1 = 17$  bus cycles
  - Bandwidth =  $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$
- 4-bank interleaved memory
  - Miss penalty =  $1 + 15 + 4 \times 1 = 20$  bus cycles
  - Bandwidth =  $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$

# Measuring Cache Performance

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - Memory stall cycles
    - Mainly from cache misses
- With simplifying assumptions:

$$\begin{aligned}\text{Memory stall cycles} &= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty} \\ &= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}\end{aligned}$$

# Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & 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 =  $2 + 2 + 1.44 = 5.44$ 
  - Ideal CPU is  $5.44/2 = 2.72$  times faster

# Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
  - $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
  - $\text{AMAT} = 1 + 0.05 \times 20 = 2\text{ns}$ 
    - 2 cycles per instruction

# Performance Summary

- When CPU performance increased
  - Miss penalty becomes more significant
- Decreasing base CPI
  - Greater proportion of time spent on memory stalls
- Increasing clock rate
  - Memory stalls account for more CPU cycles
- Can't neglect cache behavior when evaluating system performance

# Exercise

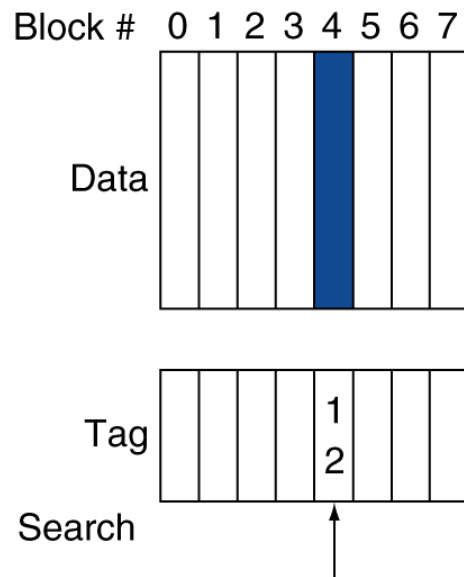
- Assume that, a memory has 20-bit byte address
  - a) How many bits does the 16 one-word blocks direct-mapped cache need?
  - b) How about the 8 2-word blocks direct-mapped cache?

# Associative Caches

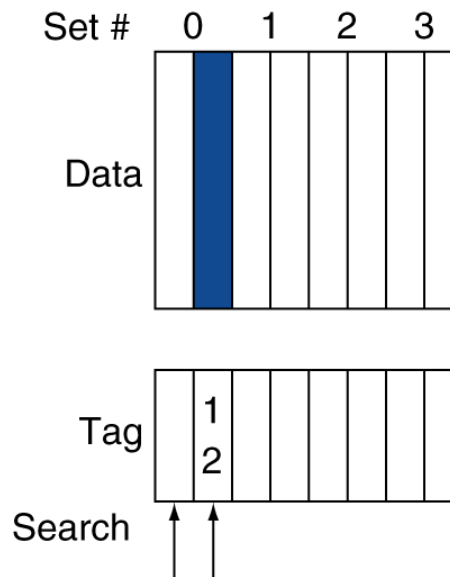
- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

# Associative Cache Example

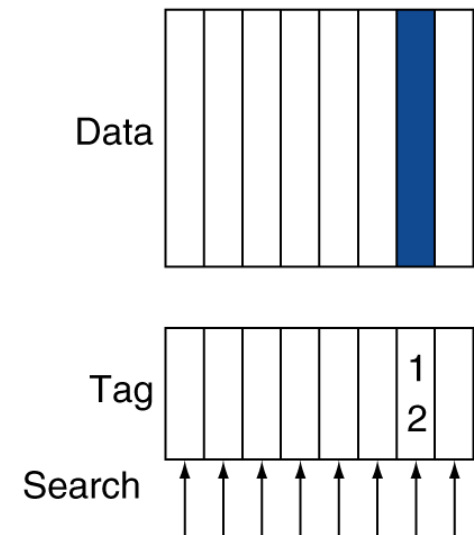
**Direct mapped**



**Set associative**



**Fully associative**





# Spectrum of Associativity

- For a cache with 8 entries

**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 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

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

- 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]		

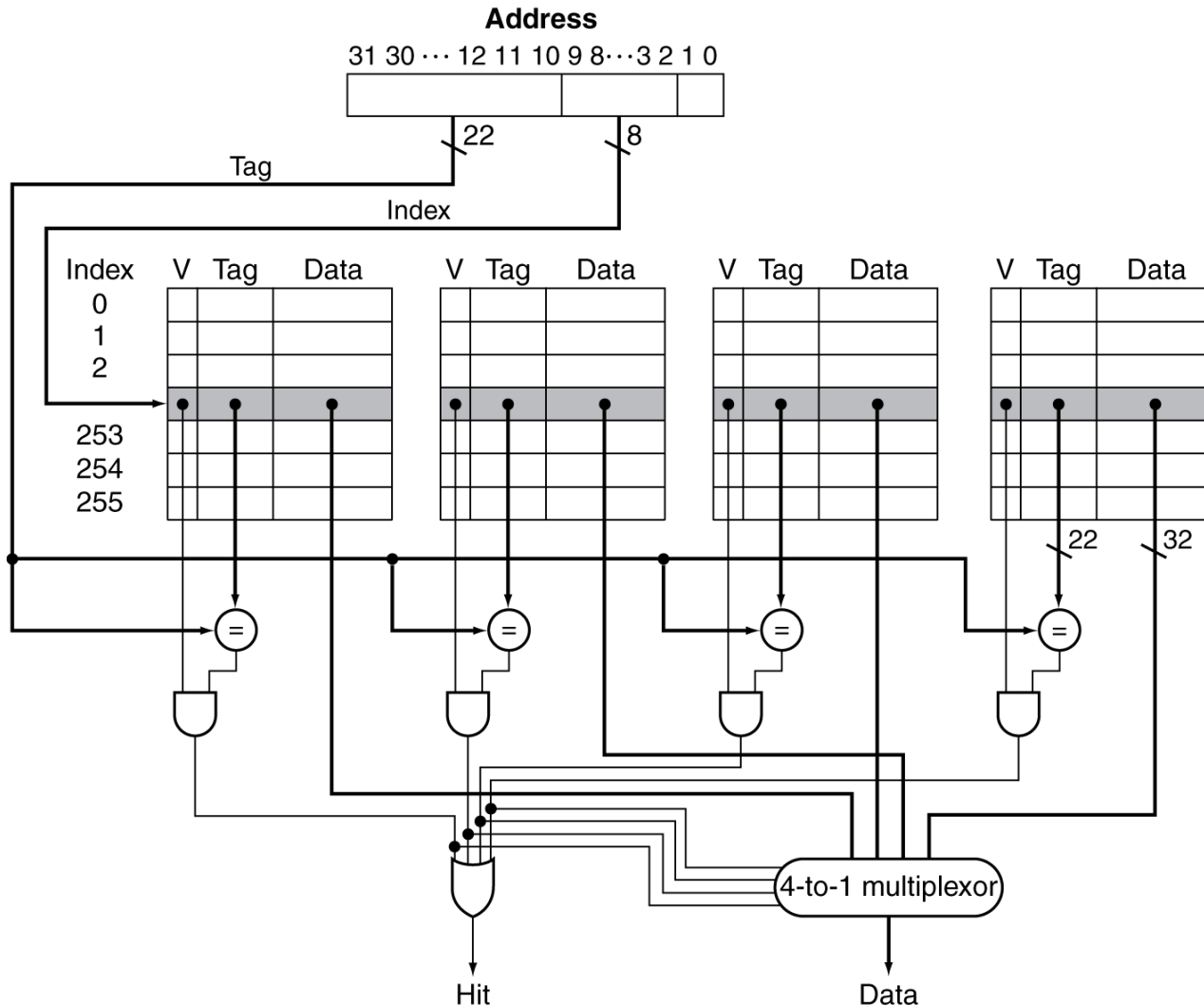
- Fully associative

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

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

# Set Associative Cache Organization



# Replacement Policy

- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

# Least Recently Used Algorithm

- Need to keep track of what was used when
  - Keep “age bits” for each cache line
  - Update all the “age bits” of all cache lines when a cache line is used
- Pseudo-LRU: use one bit per cache line

# Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



# Multilevel Cache Example

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

# Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty =  $5\text{ns}/0.25\text{ns} = 20$  cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles
- $\text{CPI} = 1 + 0.02 \times 20 + 0.005 \times 400 = 3.4$
- Performance ratio =  $9/3.4 = 2.6$

# Multilevel Cache Considerations

- Primary cache
  - Focus on minimal hit time
- L-2 cache
  - Focus on low miss rate to avoid main memory access
  - Hit time has less overall impact
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

# Interactions with Advanced CPUs

- Out-of-order CPUs can execute instructions during cache miss
  - Pending store stays in load/store unit
  - Dependent instructions wait in reservation stations
    - Independent instructions continue
- Effect of miss depends on program data flow
  - Much harder to analyse
  - Use system simulation

# Interactions with Software

- Misses depend on memory access patterns
  - Algorithm behavior
  - Compiler optimization for memory access

