MEMORY

Memory Hierarchy: Caches

Introduction

- An ideal computing system has unlimited fast memory
  - Obviously, this is not possible
  - A memory hierarchy gives the illusion of large amounts of fast memory

Principle of Locality

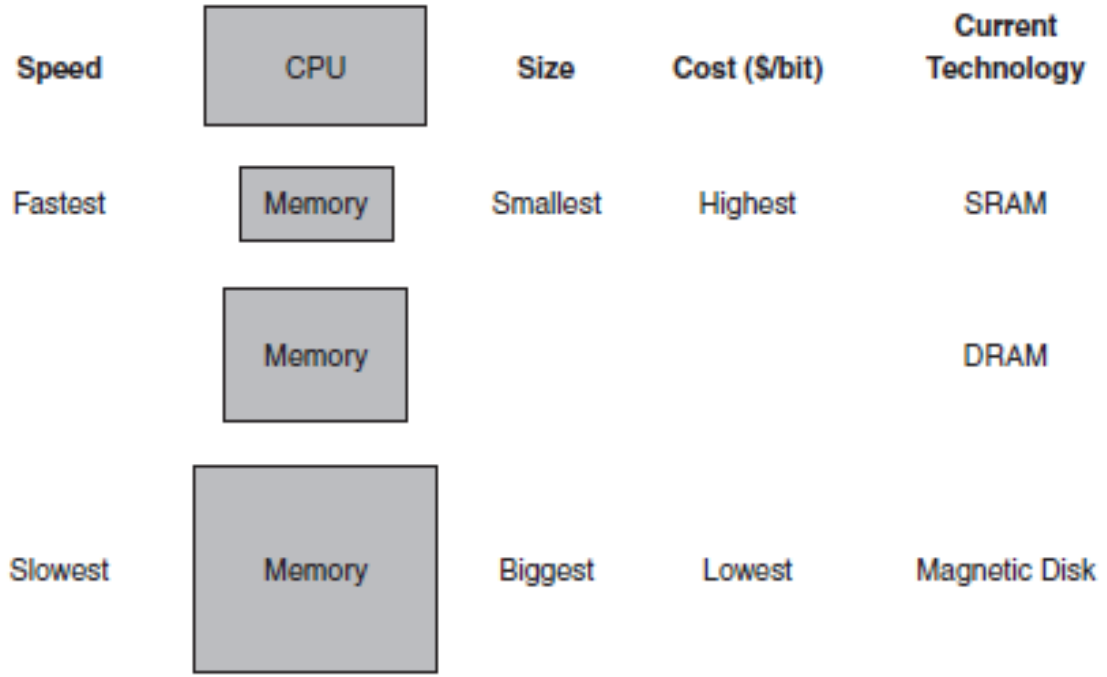
- Programs access only a small proportion of their address space at any time

- 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

Memory Hierarchy

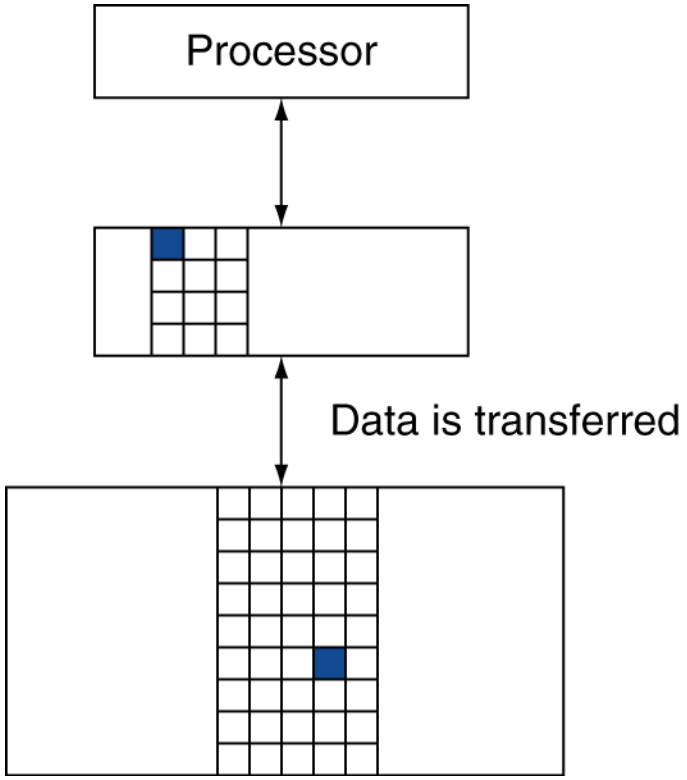
- Smaller, faster memory is closer to the processor



Caches

- Permanent storage on disk
- Copy recently accessed and nearby items from disk to smaller DRAM memory
  - Main memory
- Copy recently accessed and nearby items from DRAM to smaller SRAM memory
  - Cache memory attached to CPU
  - Cache may also refer to any storage that takes advantage of locality

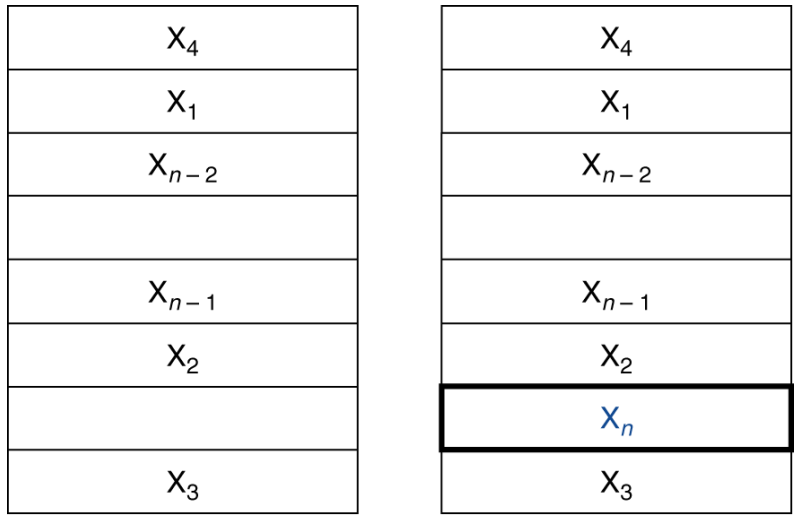
Memory Hierarchy Levels



- Block: 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 – hit ratio
  - Then accessed data supplied from upper level

Cache Memory

- Cache memory
  - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, …, $X_{n–1}$, X_n



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

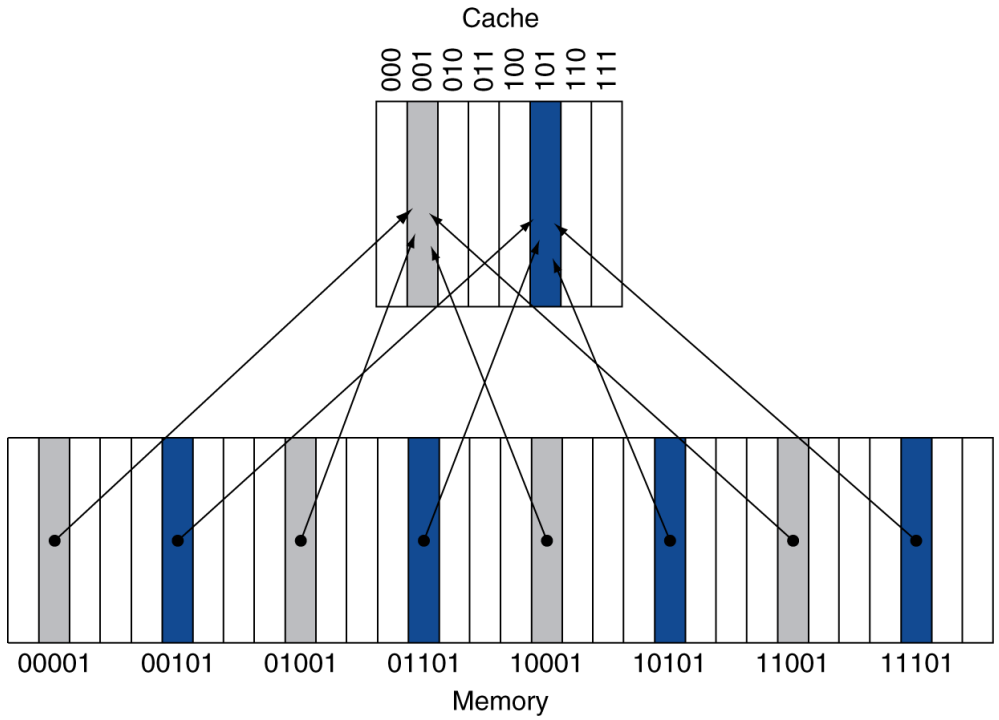
- How do we know if the data is present?
- Where do we look?

Direct Mapped Cache

- Each word can go in exactly one place in the cache
  - Assign a location based on the address of the word
  - (Block address) modulo (#Blocks in cache)

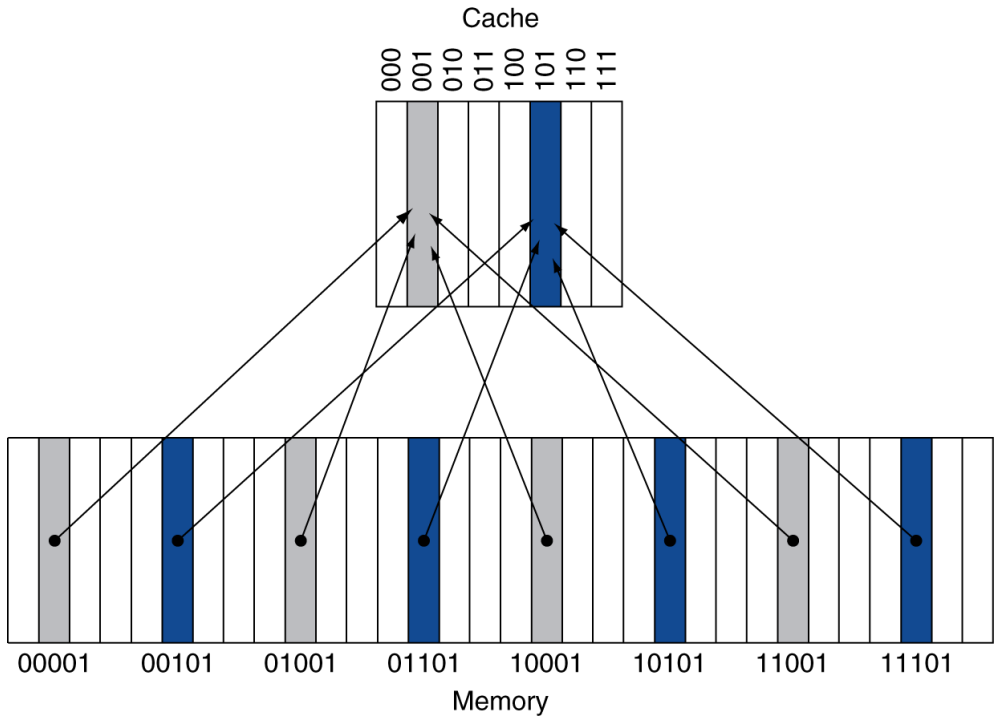
Direct Mapped Cache

- Each word can go in exactly one place in the cache
  - Assign a location based on the address of the word
  - (Block address) modulo (#Blocks in cache)



Direct Mapped Cache

- Each word can go in exactly one place in the cache
  - Assign a location based on the address of the word
  - (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
  - 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 Read

- Reads are simpler because reads do not change the contents of the cache

- Example:
  - 8-Block, Direct Mapped Cache
  - Addresses:

Binary	Decimal	Cache Block (address mod 8)
10110	22	110
11010	26	010
10110	22	110
11010	26	010
10000	16	000
00011	3	011
10000	16	000
10010	18	010
10000	16	000

Cache Example

- 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		
110	**Y**	**10**	**Mem[10110]**
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		
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
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

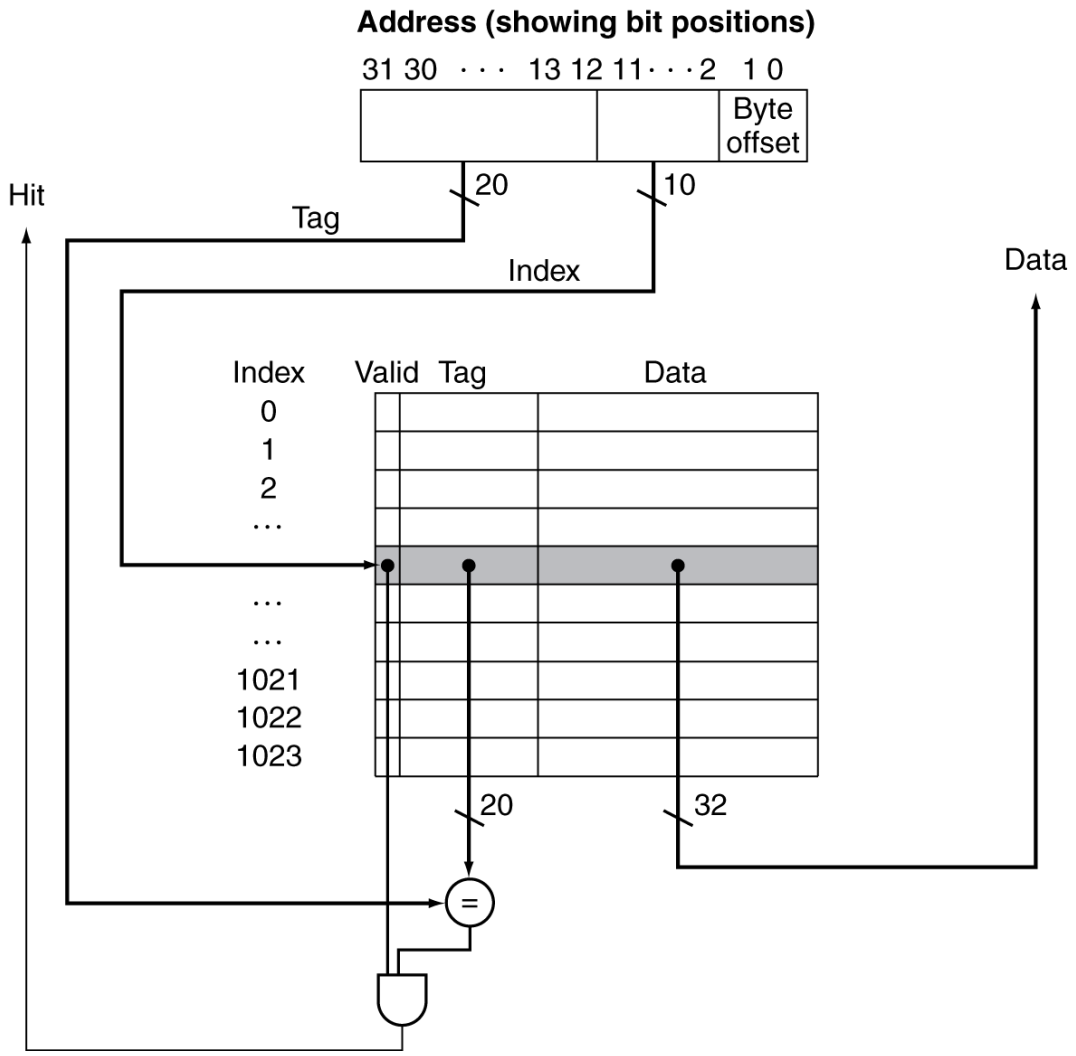
Index	V	Tag	Data
000	**Y**	**10**	**Mem[10000]**
001	N		
010	Y	11	Mem[11010]
011	**Y**	**00**	**Mem[00011]**
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

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		
010	**Y**	**10**	**Mem[10010]**
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Address Subdivision



Cache Hit

- Index of the address specifies cache block
- If the valid bit indicates data is present, check the tags
- If the tags match, it is a hit
- We can use the data stored in the cache
- CPU proceeds normally

Cache Misses

- If the valid bit is not set or if the tags do not match, it is a miss
- Control unit must detect and process a miss
  - Creates a stall in the pipeline

Cache Miss (Instruction Memory)

1. Send the original PC value (current PC – 4) to the memory.
2. Instruct main memory to perform a read and wait for the memory to complete its access.
3. Write the cache entry, putting the data from memory in the data portion of the entry, writing the upper bits of the address (from the ALU) into the tag field, and turning the valid bit on.
4. Restart the instruction execution at the first step, which will refetch the instruction, this time finding it in the cache.

Cache Miss

- Stall until the data is fetched from memory
  - Instruction cache miss
    - Restart instruction fetch
  - Data cache miss
    - Complete data access

Cache Miss

- Misses are classified into three different types

- Compulsory (cold start)
  - First access to a piece of data
  - Cannot be avoided
- Capacity
  - Working set of program is larger than cache size
  - Not enough room to cache a locality
- Conflict
  - collisions, multiple blocks competing for the same space

- Misses may be reduced by changing the associativity of the cache

Cache Organizations

- Direct mapped
- Set associative
- Fully associative

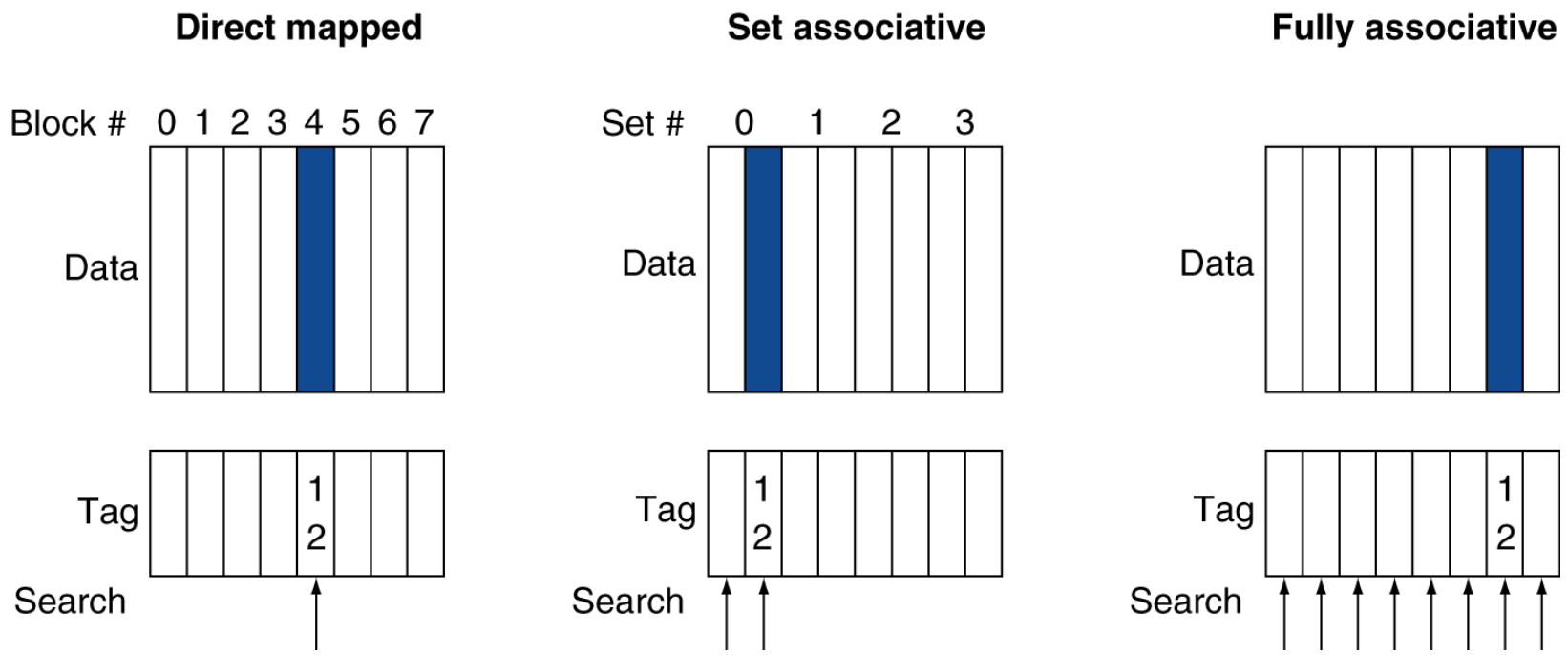
Fully Associative

- A block in memory may be associated with any entry in the cache
- Requires all entries to be searched at once
- Comparator per entry (expensive)

N-way Set Associative

- Each block may go to n locations
- 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



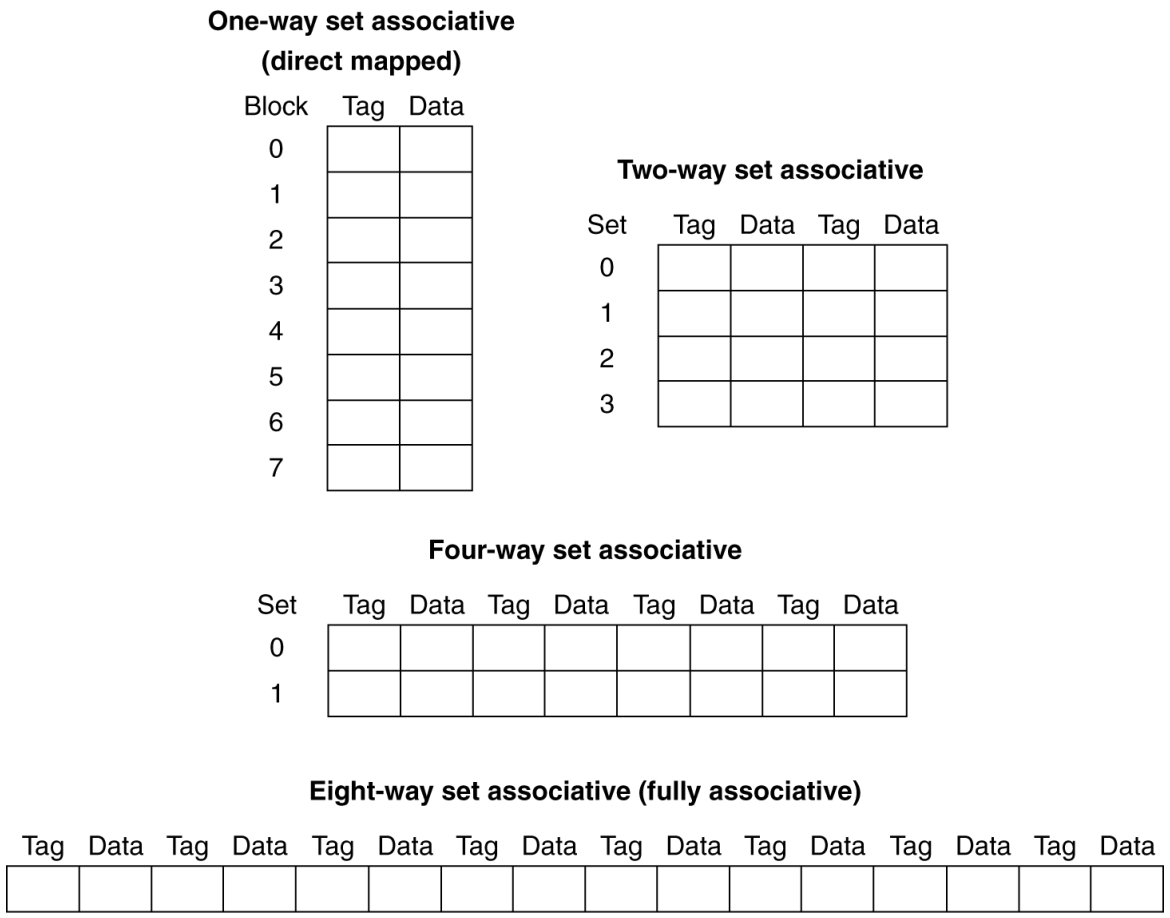
Associativity

- All cache organizations are a variation of set associativity.

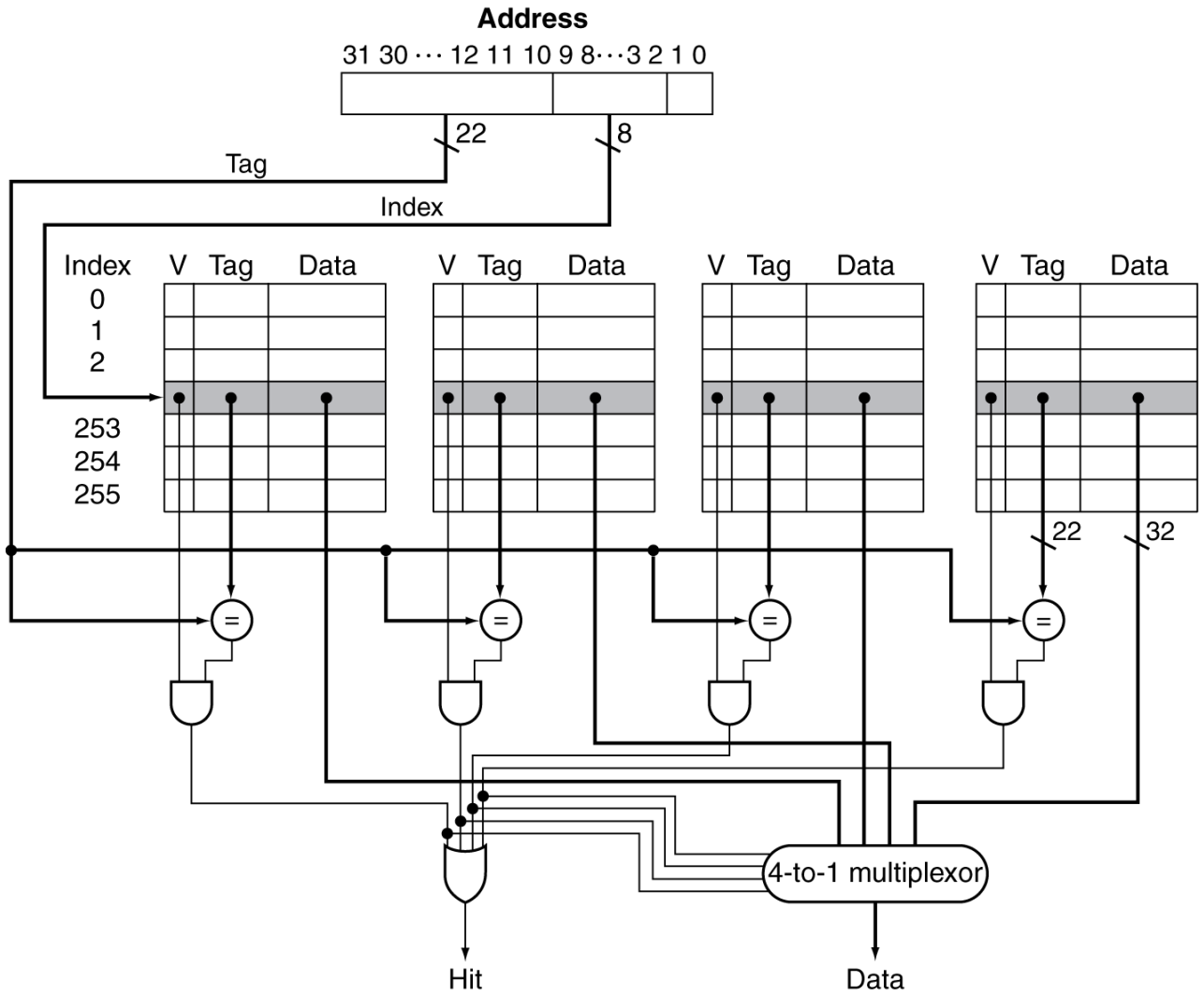
- Direct mapped: 1-way set associative
- Fully associative: m-way associative where m is the number of blocks in the cache

Spectrum of Associativity

- For a cache with 8 entries



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

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

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]**			

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]**			

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]**			

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]**	

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

- 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]	

- 5 misses

Associativity Example

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

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	

Associativity Example

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

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

Associativity Example

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

Associativity Example

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

Associativity Example

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

Associativity Example

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

Associativity Example

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

- 4 misses

Associativity Example

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

- 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]	

- 3 misses

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns

- Simulation of a system with 64KB cache, 16-word blocks
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Cache Write

- If we replace the data memory in our datapath with a cache, what happens on a store word instruction?

- If we wrote the data into only the data cache without changing main memory; then main memory would have a different value from that in the cache.
  - In such a case, the cache and memory are said to be inconsistent.

Write-Through

- Whenever changes are made to the data in the cache, also change the data in main memory.

Write-Through

- Whenever changes are made to the data in the cache, also change the data in main memory.

- Writes take longer
  - Suppose 10% of instructions are stores and write to memory takes 100 cycles
  - If base CPI = 1, then effective CPI = 1 + 0.1×100 = 11

Write-Through

- Whenever changes are made to the data in the cache, also change the data in main memory.
- Writes take longer
- 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

- On data-write hit, just update the block in cache
  - Keep track of whether each block is “clean” or “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

Performance

- The time to service hits and misses affects the CPU time.
  - Hit time
    - time required to access the cache
    - includes time to determine if it’s a hit or a miss
  - Miss penalty
    - time required to fetch a block from the next lowest level
  - Hit time < Miss penalty

Measuring Cache Performance

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

- CPU time = (CPU cycles + Memory-stall cycles) * Clock cycle time

- Memory-stall cycles = Read-stall cycles + Write-stall cycles

- Read-stall cycles = Reads/Program * Read miss rate * Read miss penalty

- Write-stall cycles = (Writes/Program * write miss rate * write miss penalty) + Write buffer stalls

Measuring Cache Performance

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

$$\text{Memory stall cycles}$$

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

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

Cache Performance Example

- Assume an instruction cache miss rate for a program is 2% and a data cache miss rate is 4%. If a processor has a CPI of 2 without any memory stalls and the miss penalty is 100 cycles for all misses, determine how much faster a processor would run with a perfect cache that never missed. Assume the frequency of all loads and stores is 36%.

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Base CPI (ideal cache) = 2
  - Miss penalty = 100 cycles
  - Load & stores are 36% of instructions

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Base CPI (ideal cache) = 2
  - Miss penalty = 100 cycles
  - Load & stores are 36% of instructions
- Determine miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
  - Memory miss cycles = 2 + 1.44 = 3.44

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Base CPI (ideal cache) = 2
  - Miss penalty = 100 cycles
  - Load & stores are 36% of instructions
- Determine miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
  - Memory miss cycles = 2 + 1.44 = 3.44
- Actual CPI = 2 + 3.44 = 5.44

Cache Performance Example

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Base CPI (ideal cache) = 2
  - Miss penalty = 100 cycles
  - Load & stores are 36% of instructions
- Determine miss cycles per instruction
  - I-cache: $0.02 \times 100 = 2$
  - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
  - Memory miss cycles = 2 + 1.44 = 3.44
- Actual CPI = 2 + 3.44 = 5.44

- Ideal CPU is 5.44/2 =2.72 times faster

Performance Considerations

- What happens if the processor is made faster, but the memory system is not?
  - The amount of time spent on memory stalls will take up an increasing fraction of the execution time
  - Amdahl's law

Cache Performance Example Revisit

- Suppose we speed up the system by reducing its CPI from 2 to 1 without changing the clock rate.
  - Actual CPI = 1 + 3.44 = 4.44
  - Ideal CPU is 4.44/1 = 4.44 times faster

- Amount of time spent on memory stalls increases:
  - Before: 3.44/5.44 = 63%
  - Now: 3.44/4.44 = 77%

Performance Considerations

- Increasing clock rate without changing the memory system also increases the performance lost due to cache misses.
  - Memory stalls account for more CPU cycles

Performance Considerations

- Hit Access Time
  - If the hit time increases, the total time to access a word from the memory system will increase.
  - Affect by:
    - Cache size
    - Number of pipeline stages
    - Cache organization

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

- Suppose we have a processor with a base CPI of 1.0, assuming all references hit in the primary cache, and a clock rate of 5 GHz. Assume a main memory access time of 100 ns, including all the miss handling. Suppose the miss rate per instruction at the primary cache is 2%.
- How much faster will the processor be if we add a secondary cache that has a 5 ns access time for either a hit or a miss and is large enough to reduce the miss rate to main memory to 0.5%?

Multilevel Cache Example

- Given
  - CPU base CPI = 1
  - CPU clock rate = 5 GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns

Multilevel Cache Example

- Given
  - CPU base CPI = 1
  - CPU clock rate = 5 GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.2ns = 500 cycles
  - Effective CPI = 1 + 0.02 × 500 = 11

Multilevel Cache Example

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.2ns = 25 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- CPI = 1 + 0.02 × 25 + 0.005 × 500 =4
- Performance ratio = 11/4 = 2.8

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

Summary

- Cost/Performance tradeoff in memory technologies
  - Create an illusion of large amounts of fast memory with a hierarchy
  - Uses caches close to the processor
- Cache Details
  - Organization: direct mapped, set associative, fully associative
  - Write policy: write-through vs. write-back
  - Replacement policy: direct, LRU, random

Summary

- Tradeoffs to Improve Cache Performance:
  - Use better technology
    - Use faster RAMs
    - Cost and availability are limitations
  - Decrease Hit Time
    - Make cache smaller, but miss rate increases
    - Use direct mapped, but miss rate increases
  - Decrease Miss Rate
    - Make cache larger, but can increases hit time
    - Add associativity, but can increase hit time
    - Increase block size, but increases miss penalty
  - Decrease Miss Penalty
    - Reduce transfer time component of miss penalty
    - Add another level of cache