Chapter 5

Large and Fast: Exploiting Memory Hierarchy

Memory Technology

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

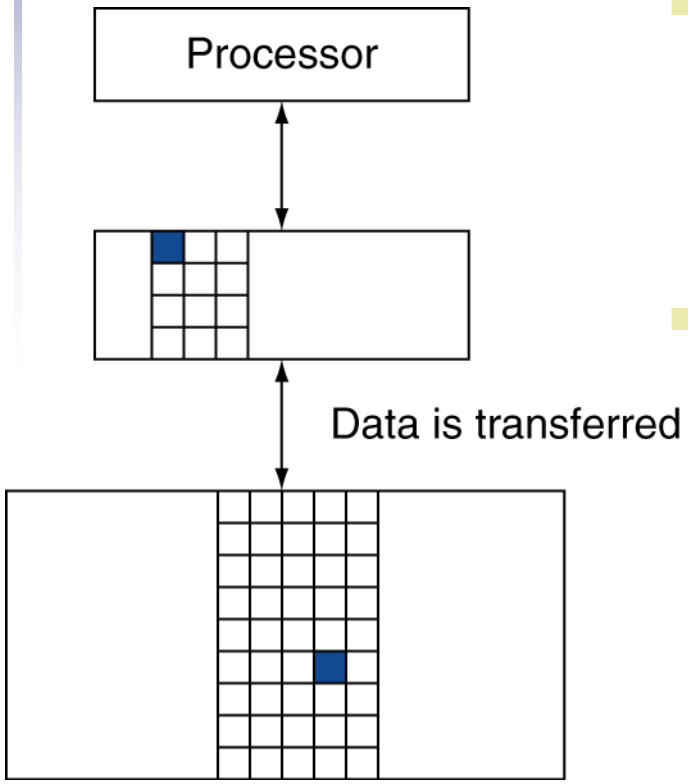
Principle of Locality

- 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 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

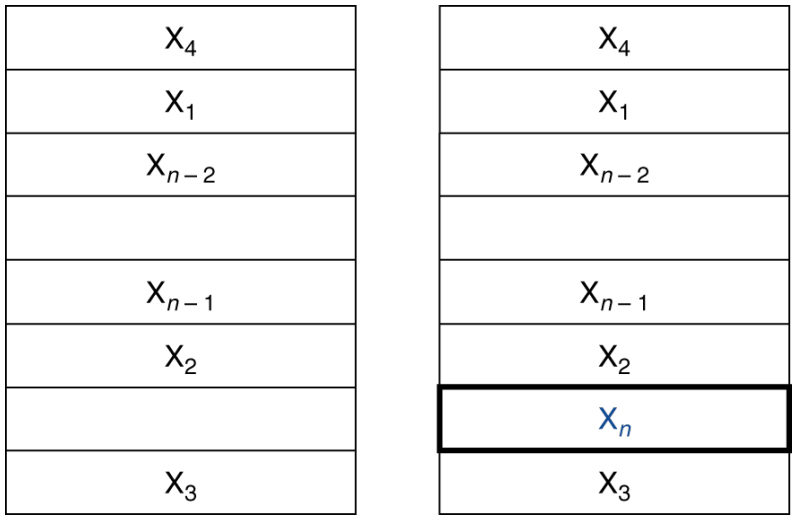
Memory Hierarchy Levels



- 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 – 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 $X_1, \ldots, X_{n-1}, X_n$



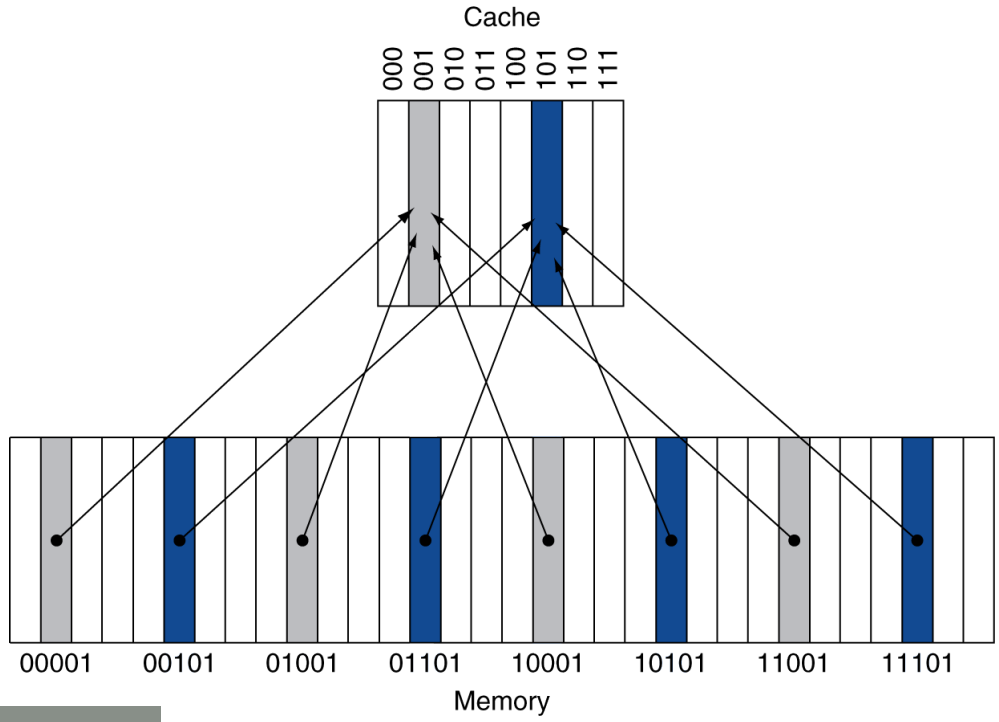
a. Before the reference to X_n

b. After the reference to X_n

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

Direct Mapped Cache

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

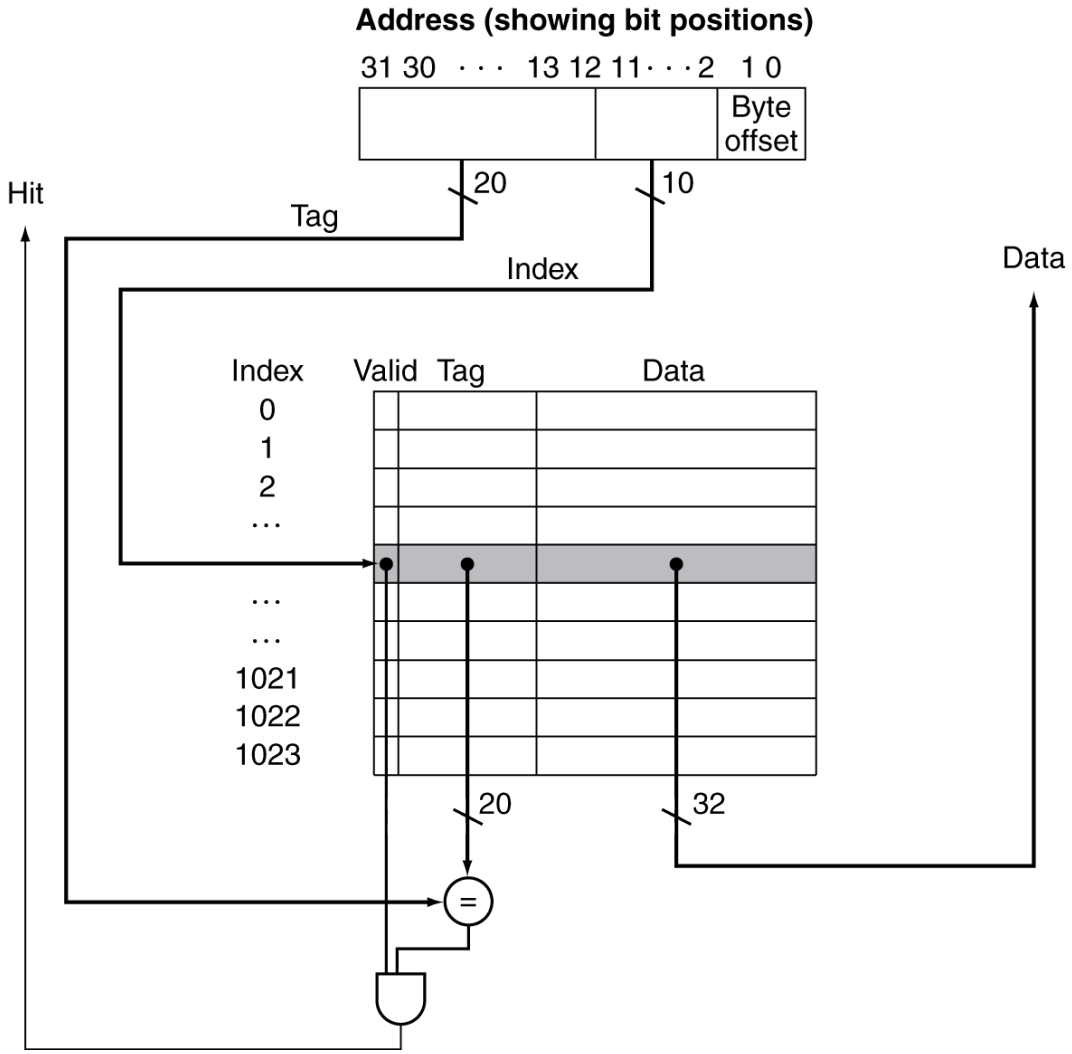


- #Blocks is a power of 2
- Use low-order address bits

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
  - 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

Address Subdivision



Address subdivision – example 1/2

- Assume:
  - The capacity of the main memory is 4GB.
  - The capacity of L1 cache is 16MB.
  - Cache block size is 64B.
  - Show how the address (X-bit wide address) is used to allow for the **direct mapping**. Make sure to clearly mark widths of the address and all fields (Tag, Index, and Offset, when applicable).
- Solution
  - Main memory capacity: 4GB = 2^{32}B => 32-bit address
  - L1 cache size: 4KB = 4*2^{10}B = 2^{12}B => 2^{12} different bytes-size entries can be stored inside a cache
  - Cache block size: 4B = 2^{2}B => 2 bits for the address with a cache block
  - 2^{index} = (cache size)/(cache block size * set associativity)

Address subdivision – example 2/2

- Main memory capacity: 4GB = 2^{32}B => 32-bit address
- L1 cache size: 4KB = 4*2^{10}B = 2^{12}B => 2^{12} different bytes-size entries can be stored inside a cache
- Cache block size: 4B = 2^{2}B => 2 bits for the address with a cache block
- 2^{index} = (cache size)/(cache block size * set associativity)

a. Direct mapping

- 2^{index} = 2^{12}/(2^{2} *1) => index = 12-2 = 10
- Index = 10 (set associativity = 1)
- Number of lines in the cache = 2^{10}
- !!! Cache block is placed into exact entry that “Index” points to !!!

31 … 12	11 … 2	1 … 0
Tag ((32-(10+2)) = 20 bits)	Index (10 bits)	Offset (2 bits)

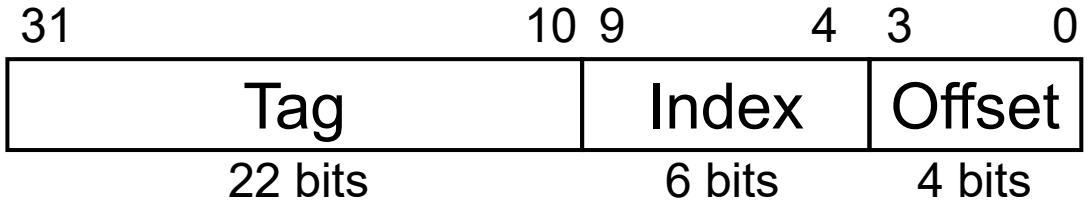
- Orthogonal approach:
- To compute BlockAddress

  BlockAddress = Floor [given address / # of Bytes per block)]
- Direct

  CacheIndex = (BlockAddress) modulo (# of blocks in the cache)

Example: Larger Block Size

- 64 blocks, 16 bytes/block
  - To what block number 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 ⇒ 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

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 + 0.1×100 = 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

Average Access Time

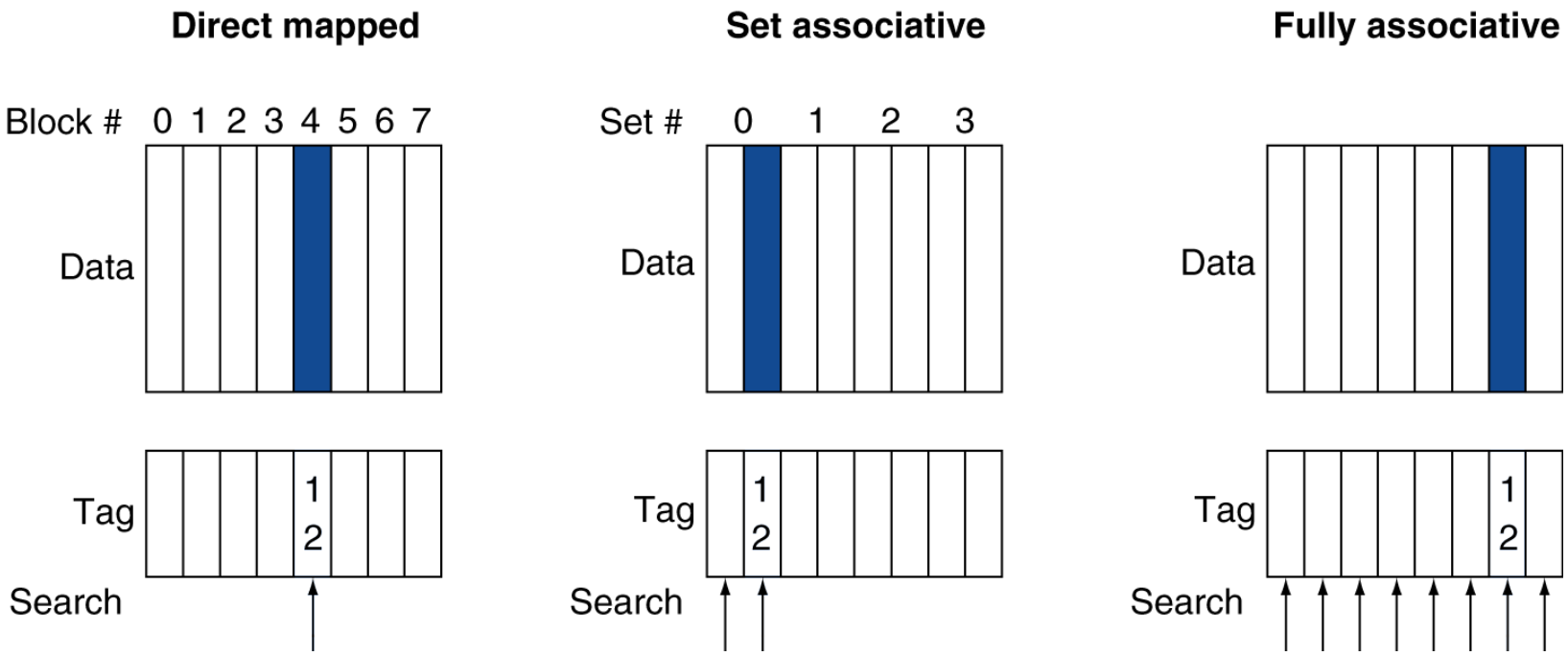
- Hit time is also important for performance
- Average memory access time (AMAT)
  - AMAT = Hit rate × Hit time + Miss rate × Miss penalty
- Example
  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 0.95 × 1 + 0.05 × 20 = 1.95ns
    - 1.95 cycles per instruction

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 each set at once
  - *n* comparators (less expensive)

Associative Cache Example



FYI: 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				

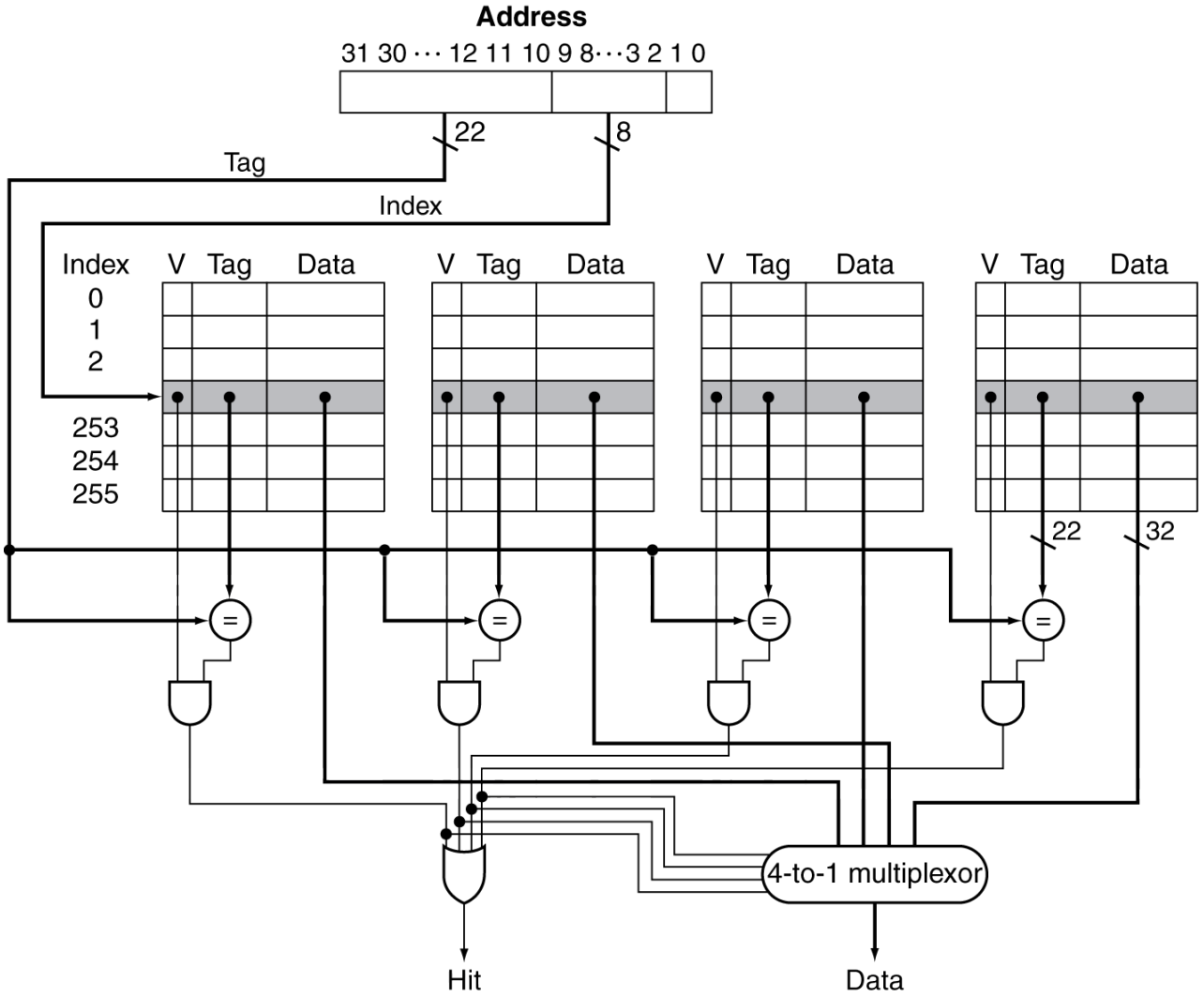
Four-way set associative

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

Set Associative Cache Organization



Address subdivision – example 1/3

- Assume:
  - The capacity of the main memory is 4GB.
  - The capacity of L1 cache is 16MB.
  - Cache block size is 64B.
  - Show how the address (X-bit wide address) is used to allow for the **fully associative** and **2-way set-associative cache**. Make sure to clearly mark widths of the address and all fields (Tag, Index, and Offset, when applicable).
- Solution (the same as in direct mapping example)
  - Main memory capacity: 4GB = 2^{32}B => 32-bit address
  - L1 cache size: 4KB = 4*2^{10}B = 2^{12}B => 2^{12} different bytes-size entries can be stored inside a cache
  - Cache block size: 4B = 2^{2}B => 2 bits for the address with a cache block
  - 2^{index} = (cache size)/(cache block size * set associativity)

Address subdivision – example 2/3

- Main memory capacity: 4GB = 2^{32}B => 32-bit address
- L1 cache size: 4KB = 4*2^{10}B = 2^{12}B => 2^{12} different bytes-size entries can be stored inside a cache
- Cache block size: 4B = 2^{2}B => 2 bits for the address with a cache block
- 2^{index} = (cache size)/(cache block size * set associativity)

- Fully associative mapping
- !!! Cache block is placed anywhere in the cache !!!

31 … 2	1 … 0
Tag ((32-2) = 30 bits)	Offset (2 bits)

- Orthogonal approach:
- To compute BlockAddress = Floor [given address / # of Bytes per block)]

- Fully associative
- Goes anywhere

Address subdivision – example 3/3

- Main memory capacity: 4GB = 2^{32}B => 32-bit address
- L1 cache size: 4KB = 4*2^{10}B = 2^{12}B => 2^{12} different bytes-size entries can be stored inside a cache
- Cache block size: 4B = 2^{2}B => 2 bits for the address with a cache block
- 2^{index} = (cache size)/(cache block size * set associativity)

- 2-way set associative mapping
- In 2-way set associative cache will be organized so that
  - $2^{10}/2^{1} = 2^{(10-1)} = 2^{9}$ => 2^{9} different entries (each containing 2 cache block) are in the cache

  .
- $2^{index} = 2^{12}/(2^{2} * 2^{1})$ => Index = 12-3 = 9
- !!! Cache block is placed in any of the entries that “Index” points to!!!

31 … 11	10 … 2	1 … 0
Tag ((32-(9+2)) = 21 bits)	Index (9 bits)	Offset (2 bits)

- Orthogonal approach:
- To compute BlockAddress = Floor [given address / # of Bytes per block)]
-
- Set-associative
- SetIndex = (BlockAddress) modulo (# of sets in the cache)

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

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 (Tclk = 0.25ns)
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles

Example (cont.)

- Assume L-1 hit is 1 cycle
- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 400 cycles

- Average memory access time (AMAT)

AMAT = Hit Rate L1 × Hit time L1

+ Miss rate L1 × [Hit Rate L2 × Hit time L2 + Miss rate L2 × Miss penalty L2]

- L1 hit rate = 0.98
- L1 miss rate = 0.02
- L2 hit rate = 0.95
- L2 miss rate = 0.05
-
- L1 hit cost = 1 cycle
- L1 miss cost = L2 hit cost = 20 cycles
- L2 miss cost = memory access = 400 cycles

- AMAT = 1.76

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

FYI

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		
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		

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]	