Computer Organization/ Architecture (COMP2825)

Instructor: Maryam Tanha

Fall 2021

Improving Cache Performance

- **Two techniques for improving cache performance**
  1) Flexible placement of blocks in cache: reduces the miss rate by reducing the probability that two different memory blocks will contend for the same cache location.
  2) Multilevel caching: reduces the miss penalty by adding an additional level to the hierarchy.

Placement of Blocks in Cache (Block Placement Strategies)

- Direct-mapped Cache
- Fully Associative Cache
- n-way Set Associative Cache

Placement of Blocks in Cache (Block Placement Strategies)

- **Why should we care about the placement of blocks in cache?**

Placement of Blocks in Cache (Block Placement Strategies)

- **Why should we care about the placement of blocks in cache?**
  - it affects cache performance (impact on cache misses)

Placement of Blocks in Cache (Block Placement Strategies)

- **Why should we care about the placement of blocks in cache?**
  - it affects cache performance (impact on cache misses).
- **The simplest placement scheme**
  - direct-mapped cache
    - a block can go **exactly in one place** in cache.
    - the position of a memory block in cache: (Block address) modulo (Number of *blocks* in the cache).
    - this is one extreme.

Placement of Blocks in Cache (Block Placement Strategies)

- **Why should we care about the placement of blocks in cache?**
  - it affects cache performance (impact on cache misses)
- **The simplest placement scheme**
  - direct-mapped cache
    - a block can go **exactly in one place** in cache.
    - the position of a memory block in cache: (Block address) modulo (Number of *blocks* in the cache).
    - this is one extreme.
- **A scheme at the other extreme → fully associative**

Associative Caches

- **Fully associative**
  - allow a given block in memory to **go in (associate with) any cache entry.**
  - to find a given block, all entries in the cache must be searched.
  - practical for caches with small number of blocks.

Associative Caches

- **Fully associative**
  - allow a given block in memory to **go in (associate with) any cache entry.**
  - to find a given block, all entries in the cache must be searched .
  - practical for caches with small number of blocks.
- ***n*-way set associative**
  - there are a fixed number of locations where each block can be placed.
  - a number of sets, each set contains *n* entries.
  - each block in the memory maps to a unique *set* in the cache given by the index field, and a block can be placed in *any* element of that set.
  - block number determines which set: (Block address) modulo (#Sets in cache).
  - search all entries in a given set at once.

Spectrum of Associativity

For a cache with 8 entries

(we have ignored the valid bit in this example):

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

Spectrum of Associativity

For a cache with 8 entries

(we have ignored the valid bit in this example):

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

Advantage of increasing the degree of associativity:
decrease in the miss rate

Main disadvantage of increasing the degree of associativity:
potential increase in the hit time

Direct-Mapped Cache

- A cache structure in which each memory location is mapped to exactly one location in the cache.
- The mapping that is used to find a block in cache:
  - **(Block address) modulo (Number of Blocks in cache)**
- Each block may contain one or multiple words.
- Given a memory address, there is only one place to look for it in the cache. If it is not there, then it is not in the cache.
- Multiple blocks from main memory can be mapped to the same block in cache.

Direct-Mapped Cache – Tags and Valid Bits

- **Each cache location can contain the contents of a number of different memory locations (but one at a time).**
- **How do we know which particular block is stored in a cache location?**

Direct-Mapped Cache – Tags and Valid Bits

- **Each cache location can contain the contents of a number of different memory locations (but one at a time).**
- **How do we know which particular block is stored in a cache location?**
  - add a set of tags to the cache
    - ✓contain the address information required to identify whether a word in the cache corresponds to the requested word by the processor.
    - ✓the tag needs only to include the upper portion of the address, corresponding to the bits that are not used as an index into the cache.

Direct-Mapped Cache – Tags and Valid Bits

- **Each cache location can contain the contents of a number of different memory locations (but one at a time).**
- **How do we know which particular block is stored in a cache location?**
  - add a set of tags to the cache
    - ✓contain the address information required to identify whether a word in the cache corresponds to the requested word by the processor.
    - ✓the tag needs only to include the upper portion of the address, corresponding to the bits that are not used as an index into the cache.
- **What if there is no data in a location?**
  - tag should be ignored.
  - valid bit: to indicate whether a cache entry contains a valid address.
    - ✓initially 0 , 1 = present, 0 = not present

Accessing Cache – Summary

- A referenced address is divided into two parts:
  - a t*ag field*, which is used to compare with the value of the tag field of the cache.
  - a *cache index*, which is used to select the block.
- The index of a cache block (cache entry), together with the tag contents of that block, uniquely specifies the memory address of the word contained in the cache block.

Accessing Cache – Summary – Continued

- **When there is a request for accessing a word in memory:**
  - If that cache entry is valid and the TAG field of the memory address and the Tag field in cache entry are the same, the cache entry holds the word being requested → **cache hit**
  - If the cache entry is invalid or it is valid but the tags do not match, the needed entry is not present in the cache → **cache miss**
    - the block is fetched from main memory and stored in the cache entry, replacing what was there. However, if the existing cache entry has been modified since being loaded, it must be written back to main memory before being overwritten.

Total Number of Bits in a Direct-Mapped Cache

- The total number of bits needed for a cache is a function of the cache size and the address size.
  - ✓because the cache includes both the storage for the data and the tags.
- Assumptions
  - ➢32-bit addresses
  - ➢cache size = 2^n blocks → cache index is *n* bits
- The **total number of bits in the cache** = 2^n *x* (block size + tag size + valid field size)

Address Subdivision in MIPS – Example

Assume you are given a cache such that each block in cache is one word. This cache holds 4KiB (i.e., the cache size is 4KiB). What can you conclude about the number of bits for the index and the tag?

Address Subdivision in MIPS – Example – Continued

Assume you are given a cache such that each block in cache is one word. This cache holds 4KiB (i.e., the cache size is 4KiB). What can you conclude about the number of bits for the index and the tag?

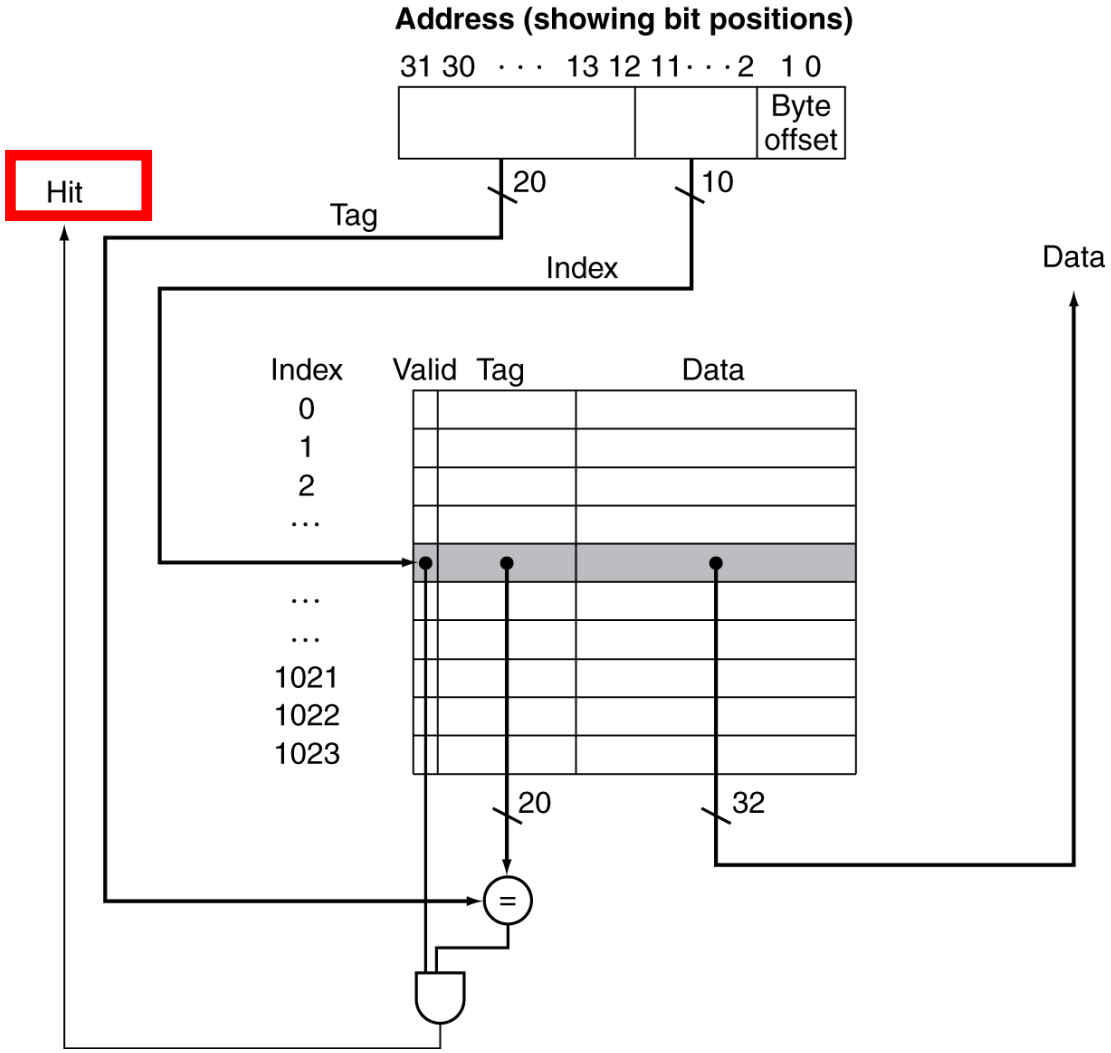
4 KiB = 2^2 x 2^{10} bytes → each block is one word (4 bytes) → 1024 (2^{10}) blocks in the cache → number of index bits = 10
→ offset = 2 bits
→ Number of bits for tag = 32 – 10 - 2 = 20

Address Subdivision in MIPS – Example – Continued

Assume you are given a cache such that each block in cache is one word. This cache holds 4KiB (i.e., the cache size is 4KiB). What can you conclude about the number of bits for the index and the tag?

4 KiB = 2^2 x 2^{10} bytes → each block is one word (4 bytes) → 1024 (2^{10}) blocks in the cache → number of index bits = 10
→ offset = 2 bits
→ Number of bits for tag = 32 – 10 - 2 = 20

Tag	Index	Block offset



Example – Number of Bits in a Direct-Mapped Cache

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Example – Number of Bits in a Direct-Mapped Cache

How many total bits are required for a direct-mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?

Answer:

valid field size = 1 bit, block size = 4 words = 16 bytes

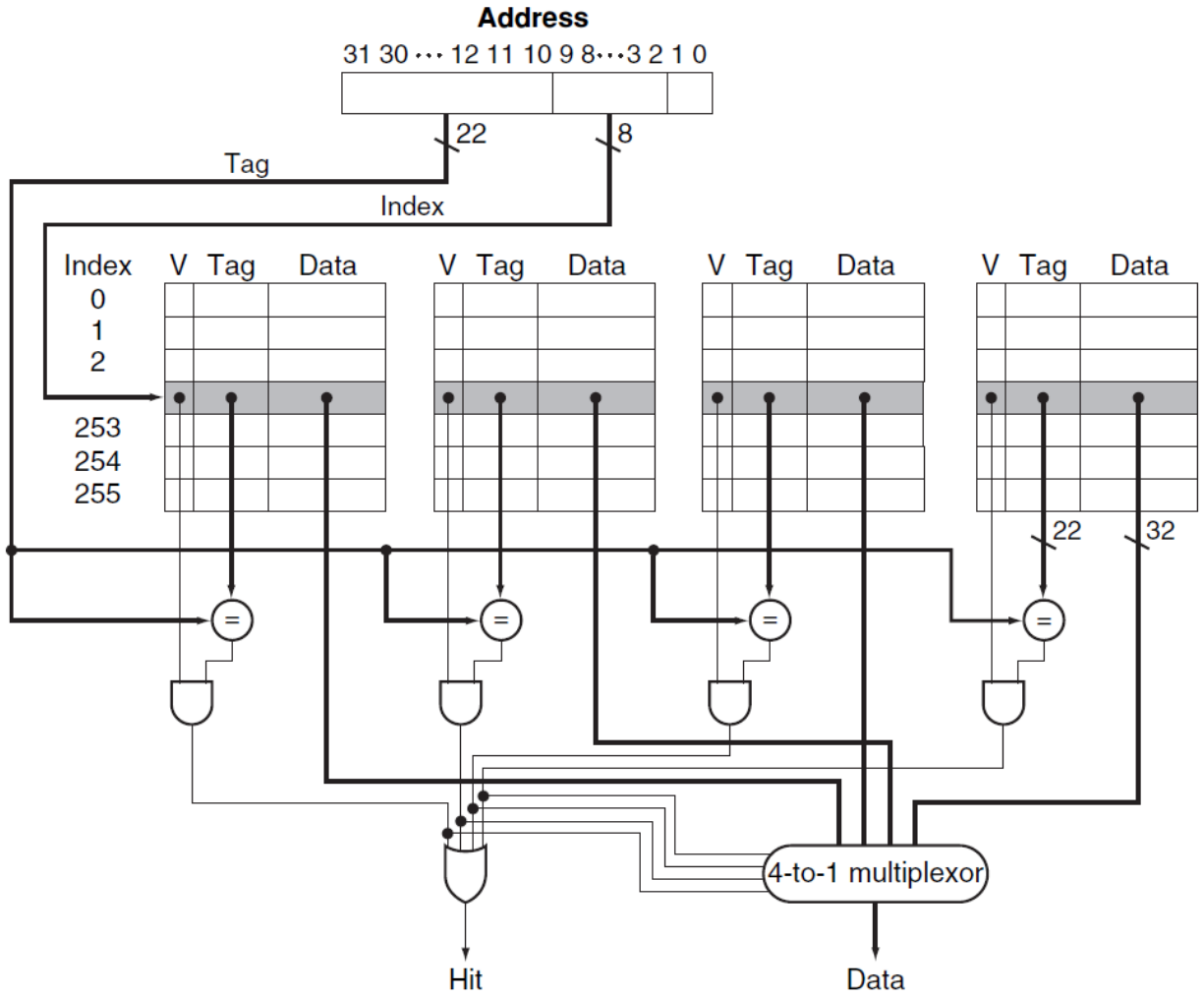
Number of bits for offset = $\log_2 16 = 4$, 16 KiB = $2^4 \times 2^{10}$ *bytes*

number of blocks = cache size/*block size* = $(2^4 \times 2^{10})/16 = 2^{10}$ → number of bits for index = $\log_2 2^{10} = 10$

tag size = 32 – index – offset = 32 – 10 – 4 = 18 bits

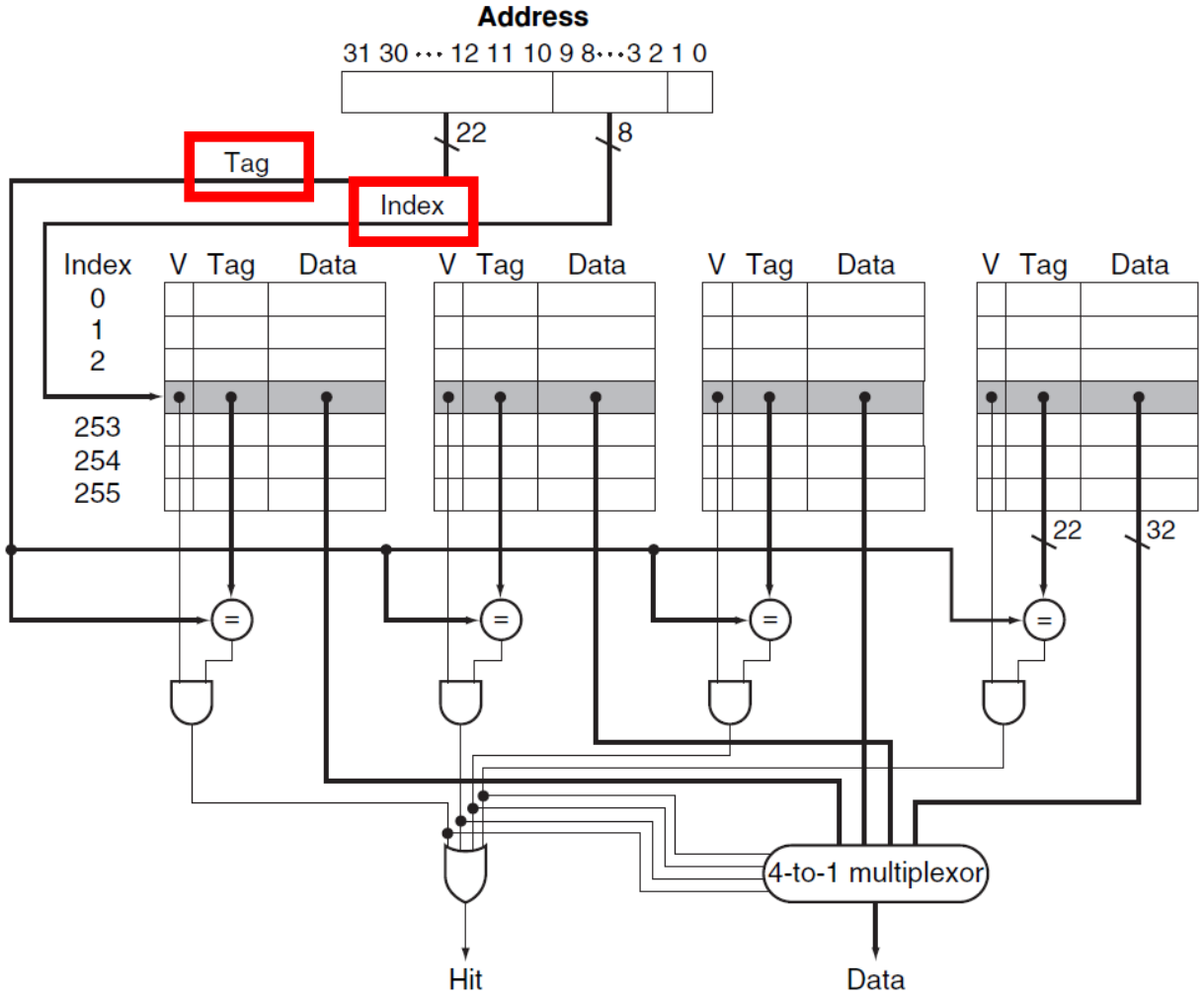
total number of bits in the cache = $2^{10} \times$ *(block size + tag + valid bit)* = $2^{10} \times (16 \times 8 + 18 + 1) = 2^{10} \times 147$

Set Associative Cache Organization (Example: A Four-Way Set-Associative Cache)



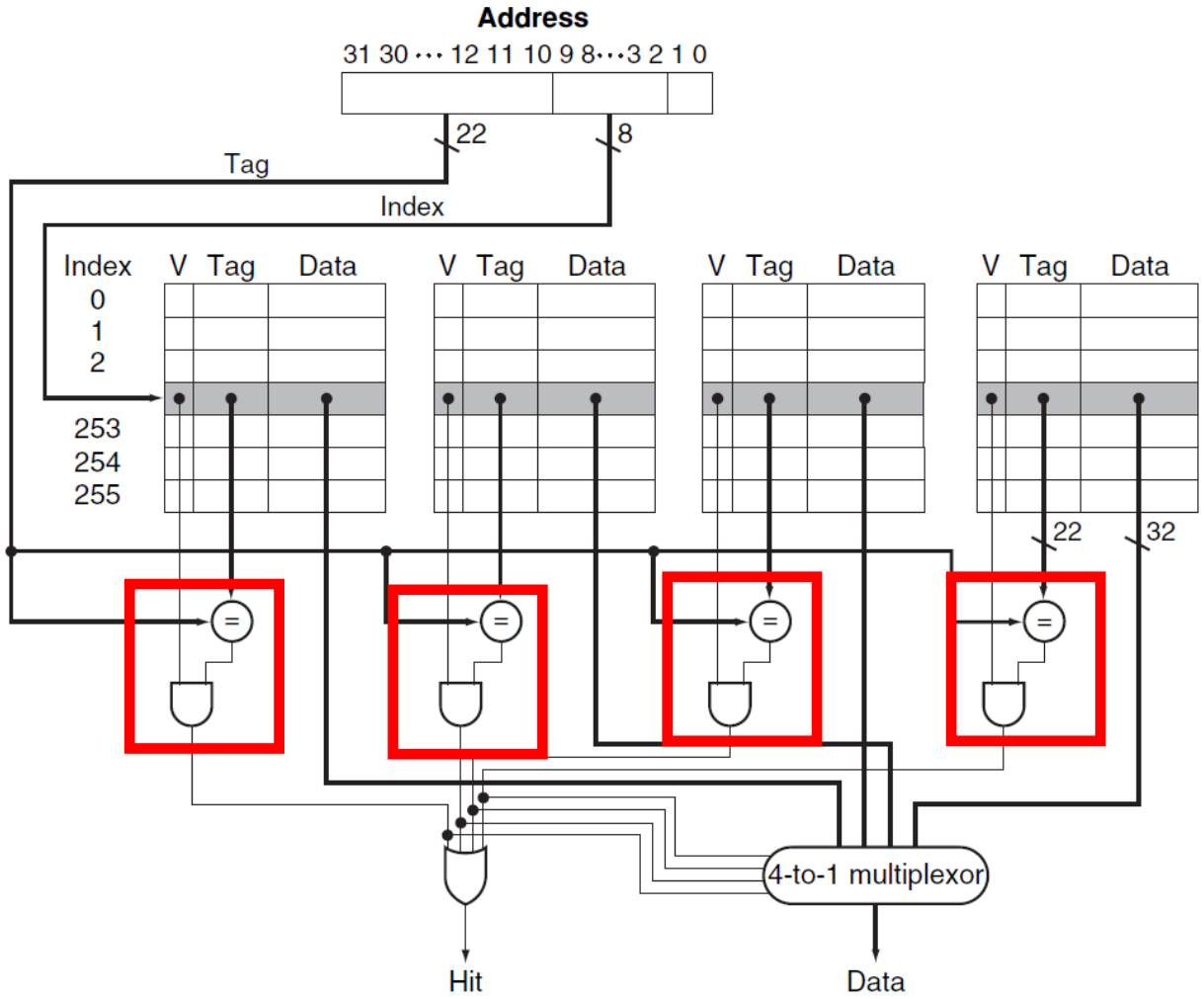
Set Associative Cache Organization (Example: A Four-Way Set-Associative Cache)

The cache access consists of indexing the appropriate set and then searching the tags of the set.



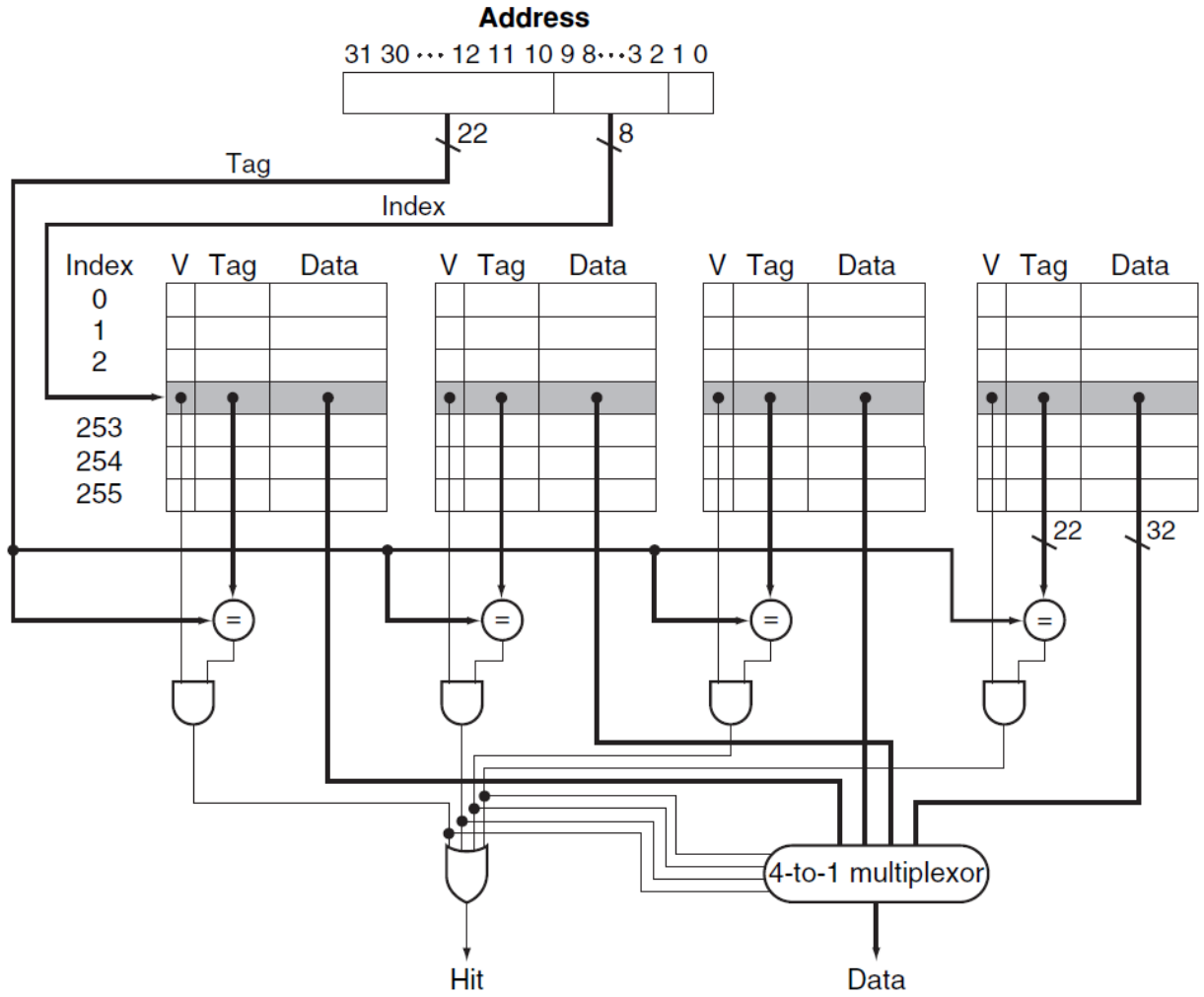
Set Associative Cache Organization (Example: A Four-Way Set-Associative Cache)

4 comparators are used. The outputs of comparators are used to indicate a hit as well as the input selector for a 4-to-1 MUX to select data from one of 4 entries of a set.



Set Associative Cache Organization (Example: A Four-Way Set-Associative Cache)

The costs of an associative cache are the extra comparators and any delay imposed by having to do the compare and select from among the elements of the set.



Choosing which Block to replace

- **When a miss occurs**,
  - **direct-mapped cache**: the requested block can go in exactly one position, and the block occupying that position must be replaced.
  - **fully associative cache**: all blocks are candidates for replacement.
  - **set-associative cache:** we must choose among the blocks in the selected set .

Block Replacement Strategies

- **Least Recently Used (LRU)**
  - the most commonly used replacement scheme.
  - the block replaced is the one that has been unused for the longest time.
  - simple for 2-way set associative, manageable for 4-way, too hard beyond that.
- **Random**
  - gives approximately the same performance as LRU for high associativity.

Associativity Example

Assume there are three small caches, each consisting of four one-word blocks. One cache is direct-mapped, the second one is two-way set-associative, and the third is fully associative. Find the number of misses for each cache organization given the following sequence of block addresses: 0, 8, 0, 6, and 8. Assume LRU replacement strategy is used.

Associativity Example – Answer

Direct-mapped cache

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Associativity Example – Answer – Continued

Direct-mapped cache

Block address	Cache block
0	(0 modulo 4) = 0
6	(6 modulo 4) = 2
8	(8 modulo 4) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		0	1	2	3
0	miss	Memory[0]			
8	miss	Memory[8]			
0	miss	Memory[0]			
6	miss	Memory[0]		Memory[6]	
8	miss	Memory[8]		Memory[6]	

Five misses for five accesses

Associativity Example – Answer – Continued

Set-associative cache (2 sets each one has 2 elements) with least recently used replacement strategy

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Associativity Example – Answer – Continued

Set-associative cache (2 sets each one has 2 elements) with least recently used replacement strategy

Block address	Cache set
0	(0 modulo 2) = 0
6	(6 modulo 2) = 0
8	(8 modulo 2) = 0

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set 1	Set 1
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[6]		
8	miss	Memory[8]	Memory[6]		

Four misses for five accesses

Associativity Example – Answer – Continued

Fully-associative cache

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Block 0	Block 1	Block 2	Block 3
0	miss	Memory[0]			
8	miss	Memory[0]	Memory[8]		
0	hit	Memory[0]	Memory[8]		
6	miss	Memory[0]	Memory[8]	Memory[6]	
8	hit	Memory[0]	Memory[8]	Memory[6]	

Three misses for five accesses → **best performance**

Three misses is the best we can do, because three unique block addresses are accessed (and the cache was empty at the beginning).

Conclusion from the Previous Example?

If we had 8 blocks in cache in the previous example, how many misses we would have for the two-way set associative cache?

Conclusion from the Previous Example?

If we had 8 blocks in cache in the previous example, how many misses we would have for the two-way set associative cache?
Three (no replacement needed)

Conclusion from the Previous Example?

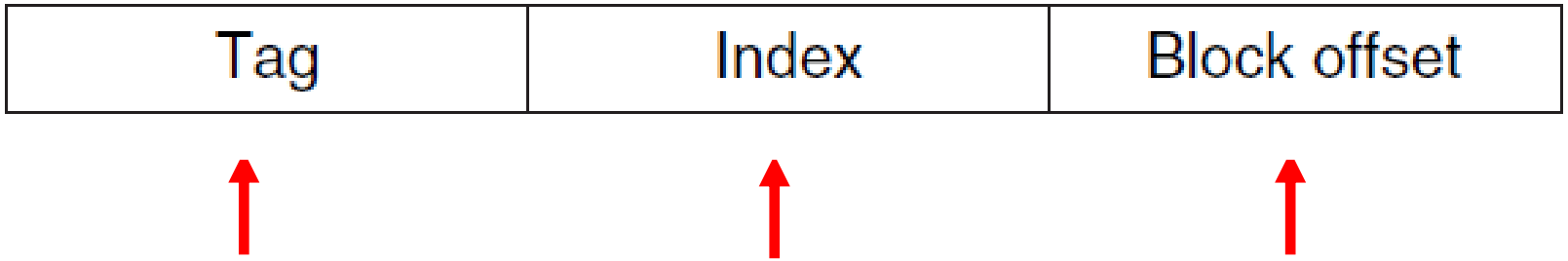
Cache size and **associativity** are **not independent** in determining **cache performance.**

How Much Associativity?

- Increased associativity decreases miss rate.
- Simulation of a system with 64KB data cache, 16-word blocks (miss rate are shown below).
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Locating a Block in the Cache

- **The three portions of an address in a set-associative cache**



to choose the block by comparison with the blocks in the selected set

to select the set

the address of the desired data within the block

How to Choose Associativity?

The choice among direct-mapped, set-associative, or fully associative mapping in any memory hierarchy will depend on the cost of a miss versus the cost of implementing associativity, both in time and in extra hardware.

Multilevel Caches

- Most microprocessors support an additional level of caching to reduce the miss penalty.
- The second-level cache is normally on the same chip and is accessed whenever a miss occurs in the primary cache.
- If the second-level cache contains the desired data, the miss penalty for the first-level cache will be essentially the access time of the second-level cache, which will be much less than the access time of main memory.
- If neither the primary nor the secondary cache contains the data, a main memory access is required, and a larger miss penalty is incurred.

Multilevel Caches – Continued

- The primary cache of a multilevel cache is often smaller than a single-level cache.
  - the primary cache may use a smaller block size, to go with the smaller cache size and also to reduce the miss penalty.
- The secondary cache will be much larger than a single-level cache, since the access time of the secondary cache is less critical.
  - larger total size→ a larger block size than a single-level cache
  - often uses higher associativity than the primary cache given the focus of reducing miss rates.

References

[1] David A. Patterson and John L. Hennessy, Computer Organization and Design:

The Hardware/Software Interface, 6th Ed, 2020, Morgan Kaufmann Publishers, Inc.