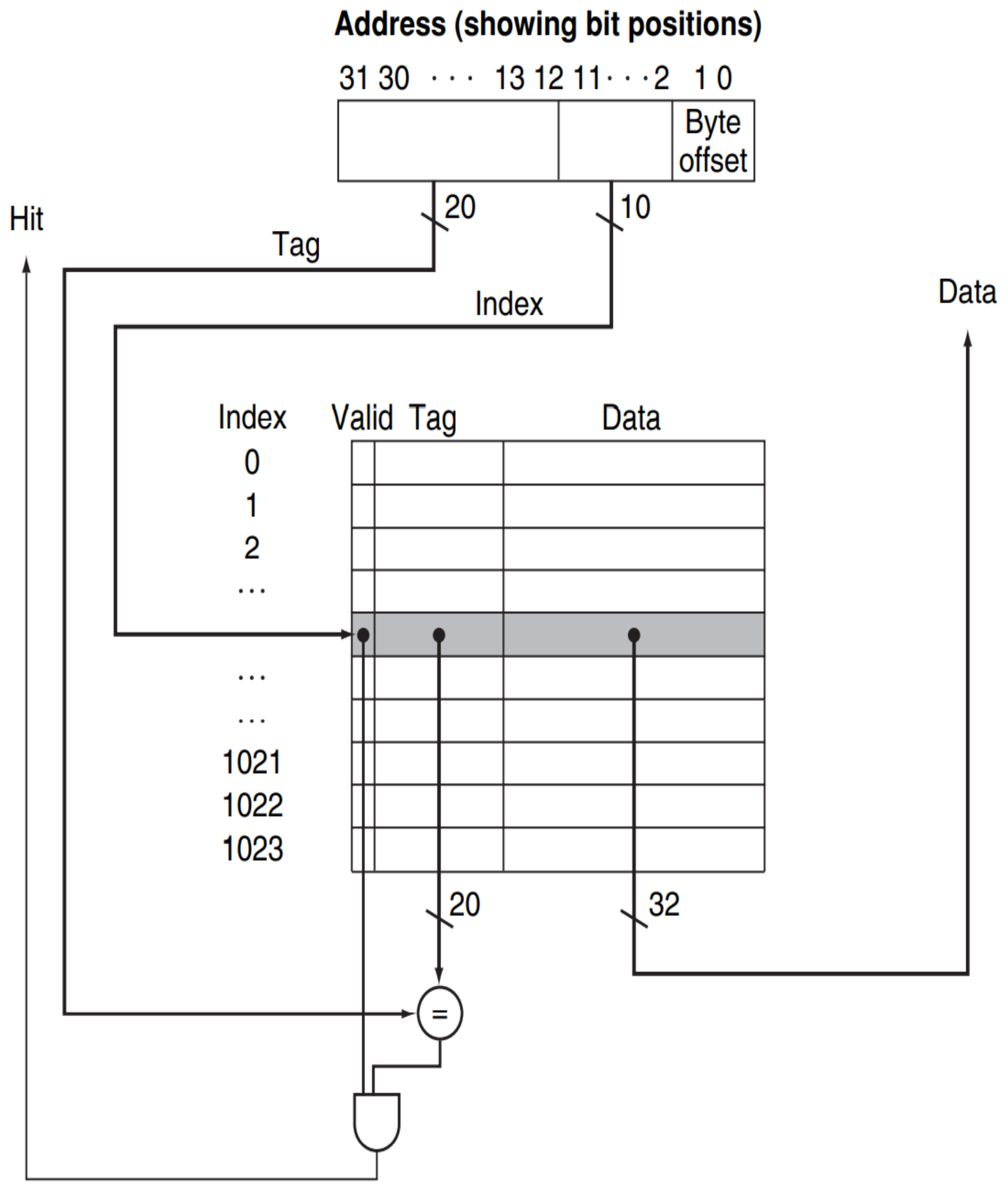
Computer Architecture
(ENE1004)

Lec – 22: Large and Fast: Exploiting Memory Hierarchy (Chapter 5) – 4

Schedule

- Final exam: Jun. 19, Monday
  - (24334) 1:25~2:25pm
  - (23978) 2:30~3:30pm
  - Sample questions will be provided by Jun. 17 (Saturday)
- Assignment #2: Jun. 20, Tue. by midnight
  - It is put back as much as possible
- Remaining class days
  - 8(Thur), 12(Mon), 15 (Thur)

Generalization of Direct Mapped Cache

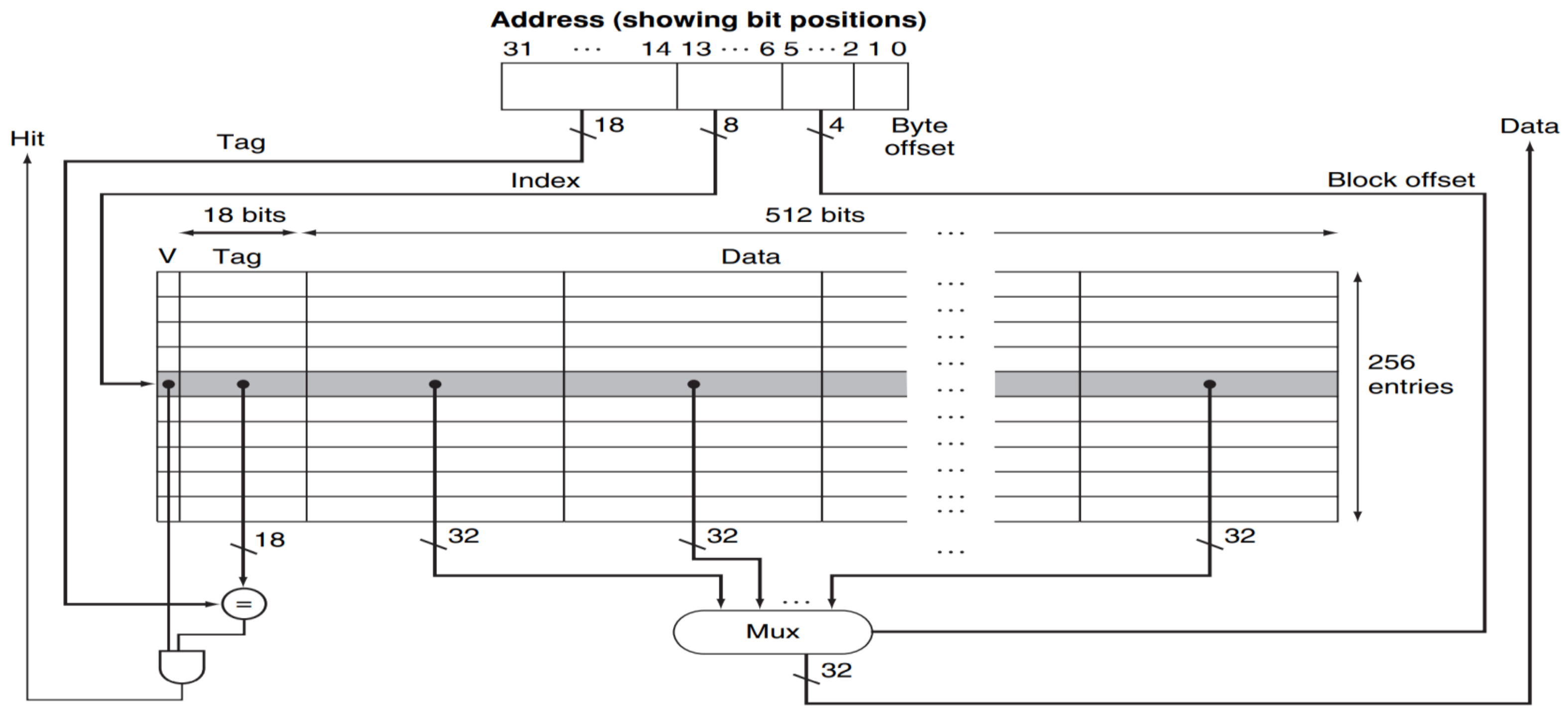


- Assumption
  - 32-bit address
  - The cache size is 2^n blocks
  - The block size is 2^m words (2^{m+2} bytes)
- Address fields
  - Byte offset: $m+2$ bits
  - Index: n bits
  - Tag: $32 - (n + m + 2)$ bits
- The total number of bits in a cache
  - # of blocks x (block size + tag size + valid bit size)
  - 2^n x (2^{m+2} bytes + (32-n-m-2) bits + 1 bit)
  - 2^n x (2^{m+5} + (32-n-m-2) + 1) bits
- Due to the tag and valid bit fields, the actual size of the cache is larger than the total amount of data (2^n x 2^{m+2} bytes)

Bits in Cache

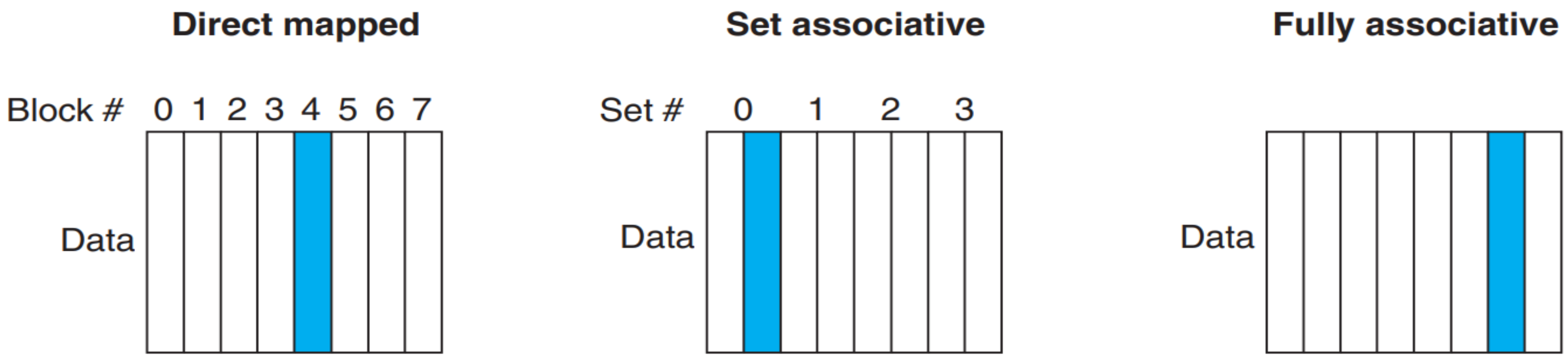
- Question: How many total bits are required for a direct mapped cache with 16 KiB of data and 4-word blocks, assuming a 32-bit address?
- Hint: We need to calculate "total # of blocks (indices)" and "# bits for tag"
  - Cache size = total # blocks x (block size + tag size + valid bit size) for each block
- The total number of cache blocks
  - The entire size of blocks = 16 KiB (2^{14} Bytes = 2^{12} words)
  - The size of each block = 4 words (2^4 Bytes = 2^2 words)
  - The total number of blocks = entire size / a block size = 2^{10} = 1024
  - The number of bits for "index" is 10 bits
- The number of bits for "tag"
  - # bits for tag = total # address bits – (# bits for index + # bits for byte offset)
  - # bits for byte offset = 4, because the size of a cache block is of 4-word (2^4 Bytes)
  - # bits for tag = 32 – (10 + 4) = 18
- Cache size = total # blocks x (block size + tag bit size + valid bit size) for each block
  - Cache size = 1024 x (4 words + 18 bits + 1 bit) = 1024 x (2^7 bits + 18 bits + 1 bit) = 2^{10} x 147 bits
  - 147 Kbits

Bits in Cache (2)



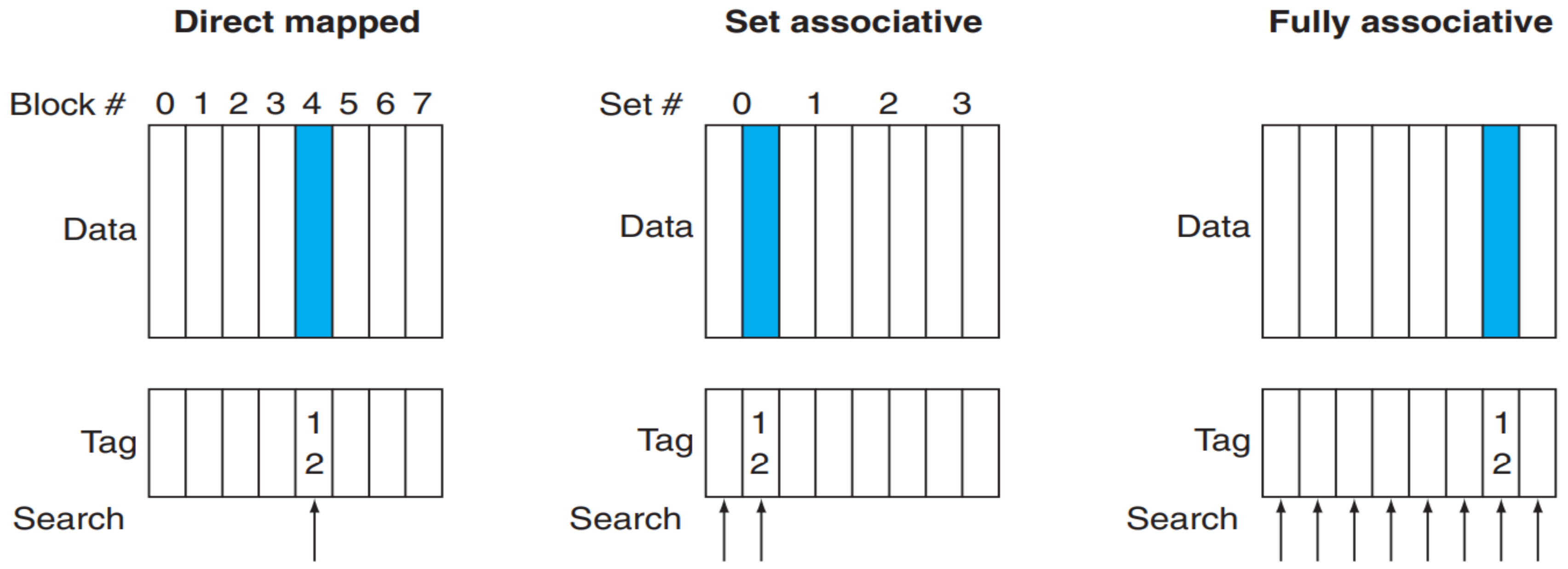
- Cache block size: 512 bits = 64 (2^6) Bytes = 16 words
  - Byte offset = 6 bits (b5-b0)
  - You may want to access the data in the unit of “word”; then, you can use upper bits in the byte offset bits; in this example, upper 4 bits (b5-b2) is used for word offset (called “block offset”)
- Total number of cache blocks is 256 (2^8); # bits for index is 8 (b13-b6)
- # bits for tag = 32 – (8 + 6) = 18 bits
- The cache size = 256 x (512 bits for data + 18 bits for tag + 1 bit for valid bit)

More Flexible Placement of Blocks (1)



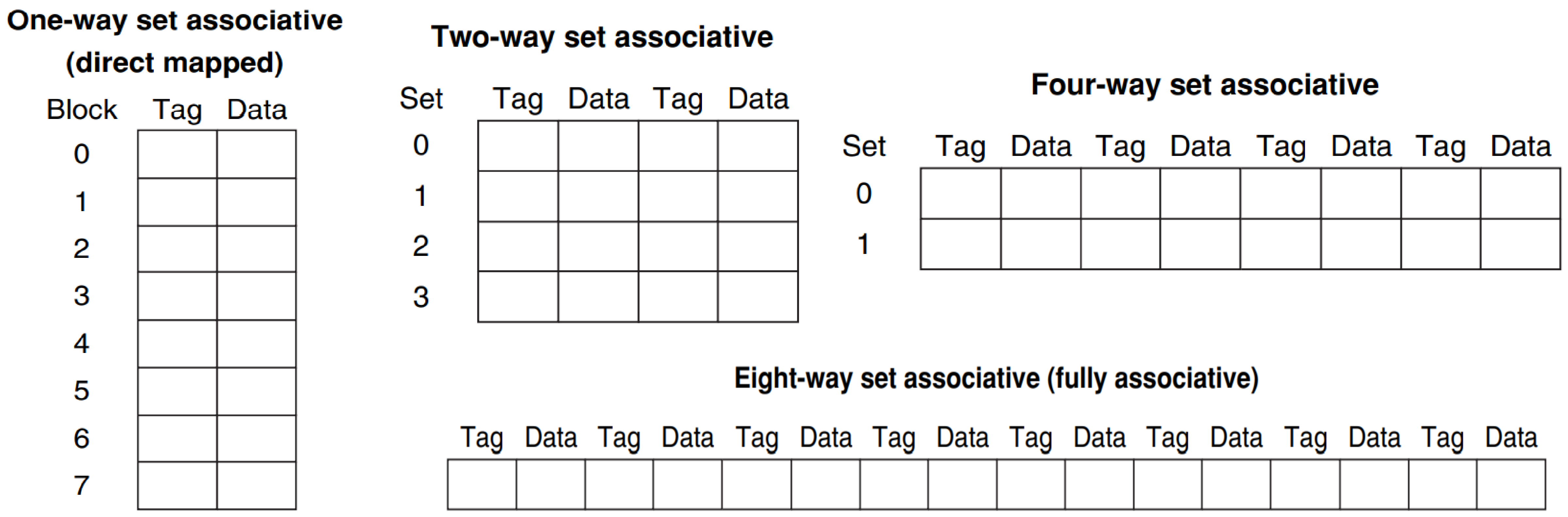
- In a direct mapped cache, a block can go in "exactly one place" in the cache
    - Given Data 12, it can be stored into only cache block 4 (12 module 8 = 4)
- There are other schemes that enable more flexible placement of blocks
- In a fully associative cache, a block can go in "any place" in the cache
    - Given Data 12, it can be stored into any cache block
    - It does not need "index" that indicates a block number in direct mapped cache
- In a set associative cache, a block can go in "a fixed number of places" in the cache
    - A set associative cache with *n* locations for a data is called an *n*-way set-associative cache
    - In 2-way set-associative cache, 8 cache blocks can be grouped into 4 sets
    - Given Data 12, it can be stored into one of the two blocks in Set 0 (12 module 4 = 0)
    - In a set associative cache, set # = (Address) module (Number of sets in the cache)

More Flexible Placement of Blocks (2)



- In a set associative cache, a block can go in “a fixed number of places” in the cache
  - In 2-way set-associative cache, Data 12 can be stored into one of the two blocks in Set 0
  - Here, we do not know whether Data 12 is in one of the two blocks in Set 0
  - If Data 12 is in the cache, we do not know which of the two blocks includes Data 12
  - So, it is required to examine both the tags in Set 0
  - For generalization, it is required to examine *n* tags in a *n*-way associative cache
- A fully associative cache can be considered as a 8-way set associative cache with one set
  - It is required to examine all the tags in the single set

Set Associative Cache



- Using the same amount of cache blocks (8 blocks), there are different schemes
  - Direct mapped cache can be considered as 1-way set associative cache (1 block in each of 8 sets)
  - 2-way set associative cache (2 blocks in each of 4 sets)
  - 4-way set associative cache (4 blocks in each of 2 sets)
  - Fully associative cache is 8-way set associative cache (8 blocks in a single set)

Misses and Associativity in Caches (1)

- Assume there are three small caches
  - Direct mapped cache vs 2-way set associative cache vs fully associative cache
  - Each consists of four blocks (a total of 4 blocks in each cache)
  - Each block is a single byte (so, there is no byte offset)
- Assume the following five cache accesses are given
  - 0, 8, 0, 6, and 8

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		

Two-way set associative

Set	Tag	Data	Tag	Data
0				
1				

(fully associative)

Tag	Data	Tag	Data	Tag	Data	Tag	Data

Misses and Associativity in Caches (2) – Direct Mapped

- Assume the following five cache accesses are given
  - 0 (0000) – index/block#: 0 module 4 = 0 (00); tag: 00
  - 8 (1000) – index/block#: 8 module 4 = 0 (00); tag: 10
  - 0 (0000) – index/block#: 8 module 4 = 0 (00); tag: 00
  - 6 (0110) – index/block#: 6 module 4 = 2 (10); tag: 01
  - 8 (1000) – index/block#: 8 module 4 = 0 (00); tag: 10

(direct mapped)

Block	Tag	Data
0		
1		
2		
3		

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Index/block#			
		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]	

- All the five accesses are misses! (hit ratio = 0; miss ratio = 1)

Misses and Associativity in Caches (3) – 2-way Set Asso.

- Assume the following five cache accesses are given
  - 0 (0000) – Set#: 0 module 2 = 0 (0); tag: 000
  - 8 (1000) – Set#: 8 module 2 = 0 (0); tag: 100
  - 0 (0000) – Set#: 8 module 2 = 0 (0); tag: 000
  - 6 (0110) – Set#: 6 module 2 = 0 (0); tag: 011
  - 8 (1000) – Set#: 8 module 2 = 0 (0); tag: 100

Two-way set associative

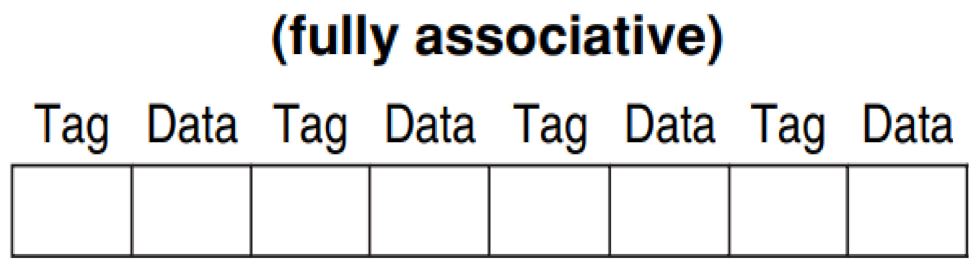
Set	Tag	Data	Tag	Data
0				
1				

Address of memory block accessed	Hit or miss	Contents of cache blocks after reference			
		Set 0	Set 0	Set# 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]		

- A total of 4 accesses are misses! (hit ratio = 0.2; miss ratio = 0.8)

Misses and Associativity in Caches (4) – Fully Asso.

- Assume the following five cache accesses are given
  - 0 (0000) – Tag: 0000
  - 8 (1000) – Tag: 1000
  - 0 (0000) – Tag: 0000
  - 6 (0110) – Tag: 0110
  - 8 (1000) – Tag: 1000



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]	

- A total of 3 accesses are misses! (hit ratio = 0.4; miss ratio = 0.6)