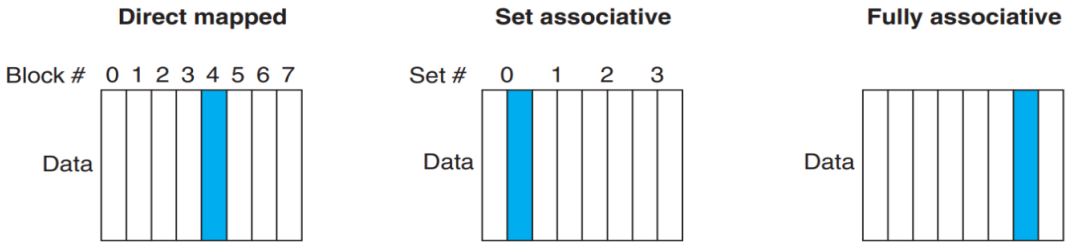
Computer Architecture (ENE1004)

▼ Lec 20

Lec 20: Large and Fast : Exploting Memory Hierarchy 4

More Flexible Placement of Blocks (1) (보다 유연한 블록 배치 1)

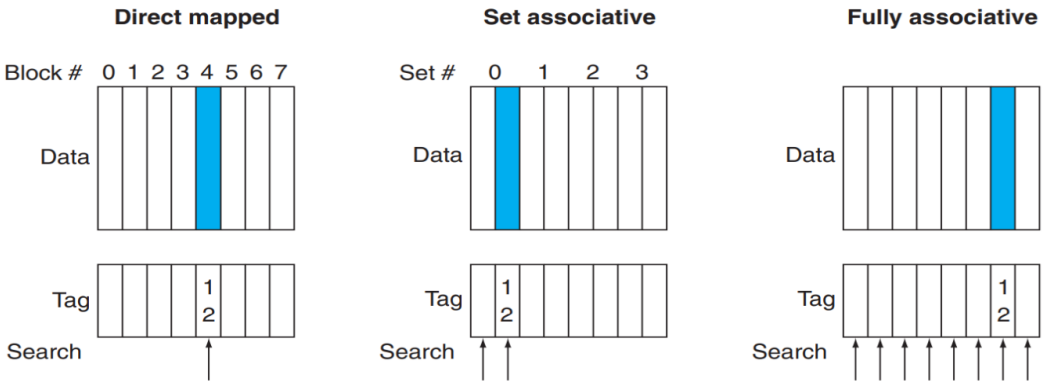


- In a direct mapped cache, a block can go in “exactly one place” in the cache (직접 매핑된 캐시에서 블록은 캐시에서 “정확히 한 곳”에만 저장될 수 있다)
  - Given Data 12, it can be stored into only cache block 4 (12 module 8 = 4) (데이터 12가 주어지면, 캐시 블록 4에만 저장할 수 있다 (12 module 8 = 4))
- There are other schemes that enable more flexible placement of blocks (블록을 보다 유연하게 배치할 수 있는 다른 방식도 있다)
- In a fully associative cache, a block can go in “any place” in the cache (완전 연관 캐시에서 블록은 캐시의 “어느 위치”에 저장될 수 있다)
  - Given Data 12, it can be stored into any cache block (데이터 12가 주어지면 모든 캐시 블록에 저장할 수 있다)
  - It does not need any “index” that indicates a block number in direct mapped cache (직접 매핑된 캐시에서는 블록 번호를 나타내는 “인덱스”가 필요하지 않다)
- In a set associative cache, a block can go in “a fixed number of places” in the cache (집합 연관 캐시에서 블록은 캐시에서 “고정된 수의 위치”에 들어갈

수 있다)

- A set associative cache with *n* locations for a data is called an *n* way set-associative cache (데이터에 대해 n개의 위치가 있는 집합 연관 캐시를 n 방향 집합 연관 캐시라고 한다)
- In 2-way set-associative cache, 8 cache blocks can be grouped into 4 sets (양방향 집합 연관 캐시에서는 8개의 캐시 블록을 4개의 집합으로 그룹화 할 수 있다)
- Given Data 12, it can be stored into one of the two blocks in Set 0 (12 module 4 = 0) (데이터 12가 주어지면, 집합 0의 두 블록 중 하나에 저장할 수 있다 (12 모듈 4 = 0))
- In a set associative cache, set # = (Address) module (Number of sets in the cache) (세트 연관 캐시에서, 세트 # = (주소) 모듈 (캐시 내 세트 수))

More Flexible Placement of Blocks (2) (보다 유연한 블록 배치 2)



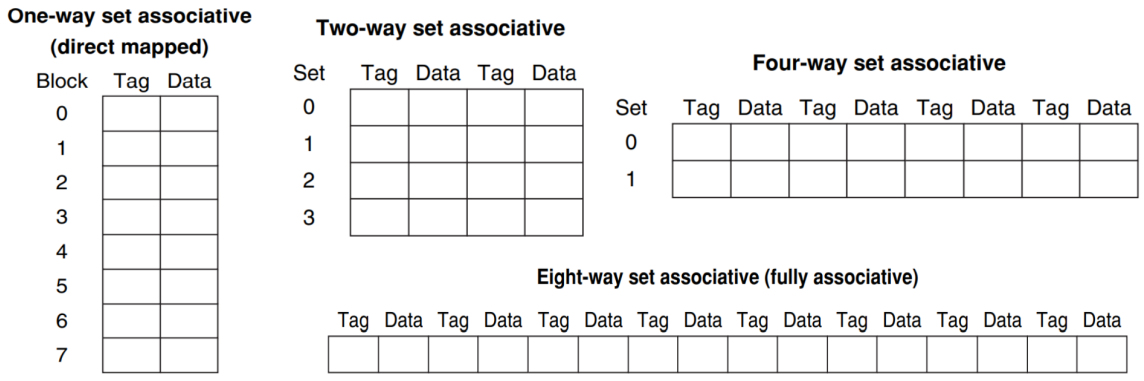
- In a set associative cache, a block can go in “a fixed number of places” in the cache (집합 연관 캐시에서 블록은 캐시에서 “고정된 수의 위치”에 저장될 수 있다.)
  - In 2-way set-associative cache, Data 12 can be stored into one of the two blocks in Set 0 (양방향 집합 연관 캐시에서 데이터 12는 집합 0의 두 블록 중 하나에 저장될 수 있다)
  - Here, we do not know whether Data 12 is in one of the two blocks in Set 0 (여기서는 데이터 12가 세트 0의 두 블록 중 하나에 있는지 여부를 알 수 없다)
  - If Data 12 is in the cache, we do not know which of the two blocks includes Data 12 (데이터 12가 캐시에 있는 경우, 두 블록 중 어느 블록에 데

이터 12가 포함되어 있는지 알 수 없다)

  - So, it is required to examine both the tags in Set 0 (따라서 세트 0의 두 태그를 모두 검사해야 한다)
  - For generalization, it is required to examine *n* tags in a *n* way associative cache (일반화를 위해서는 n 방향 연관 캐시에서 n개의 태그를 검사해야 한다)

- A fully associative cache can be considered as a 8-way set associative cache with one set (완전 연관 캐시는 하나의 세트가 있는 8방향 세트 연관 캐시로 간주할 수 있다)
  - It is required to examine all the tags in the single set (단일 세트의 모든 태그를 검사해야 한다)

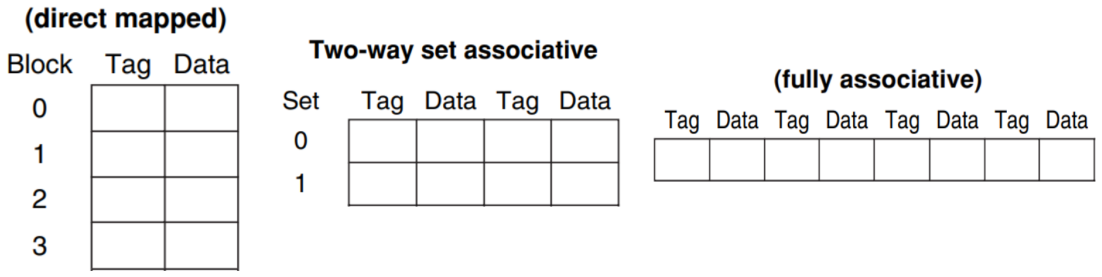
Set Associative Cache (집합 연관 캐시)



- Using the same amount of cache blocks (8 blocks), there are different schemes (동일한 양의 캐시 블록(8 블록)을 사용하는 경우, 다른 방식이 있다)
  - Direct mapped cache can be considered as 1-way set associative cache (1 block in each of 8 sets) (직접 매핑된 캐시는 단방향 세트 연관 캐시(8개의 세트에 각각 1개의 블록)로 간주할 수 있다)
  - 2-way set associative cache (2 blocks in each of 4 sets) (2방향 세트 연관 캐시(4개 세트 각각에 2개 블록))
  - 4-way set associative cache (4 blocks in each of 2 sets) (4방향 세트 연관 캐시(2개 세트에 각각 4개 블록))
  - Fully associative cache is 8-way set associative cache (8 blocks in a single set) (완전 연관 캐시는 8방향 세트 연관 캐시(단일 세트의 8블록)이다)

Misses and Associativity in Caches (1) (캐시의 누락 및 연관성 1)

- Assume there are three small caches (세 개의 작은 캐시가 있다고 가정한다)
  - Direct mapped cache vs 2-way set associative cache vs fully associative cache (직접 매핑 캐시 vs 양방향 세트 연관 캐시 vs 완전 연관 캐시)
  - Each consists of four blocks (a total of 4 blocks in each cache) (각각 4개의 블록으로 구성 (각 캐시에는 총 4개의 블록이 있음))
  - Each block is a single byte (so, there is no byte offset) (각 블록은 단일 바이트이다 (따라서 바이트 오프셋이 없음))
- Assume the following five cache accesses are given (다음과 같은 캐시 엑세스가 주어진다고 가정한다)
  - 0, 8, 0, 6, and 8



Misses and Associativity in Caches (2) - Direct Mapped (캐시의 누락 및 연관성 2 - 직접 매핑)

- Assume the following five cache accesses are given (다음과 같은 5개의 캐시 엑세스가 주어졌다고 가정한다)
  - 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			
		0	1	2 (Index/block#)	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 Associative Cache (캐시의 누락 및 연관성 3 - 양방향 집합 연관 캐시)

- Assume the following five cache accesses are given (다음과 같은 5개의 캐시 엑세스가 주어졌다고 가정한다)
  - 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 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]		

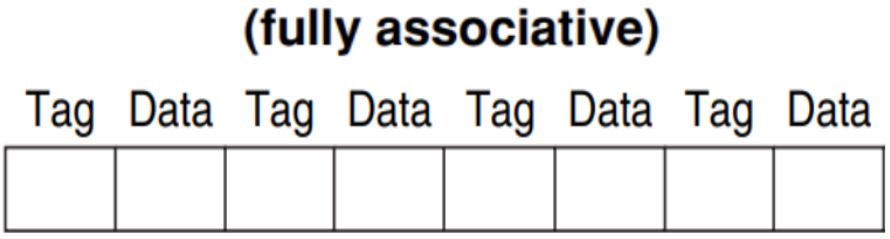
Set#

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

> 💡 **왜 "6 (0110) – Set#: 6 module 2 = 0 (0); tag: 011"에서 2번째 블록(오른쪽 블록)이 바뀌었을까? 첫번째 블록이 바뀌었다면 hit가 1 늘어나는 것 아닌가?**
> 일반적인 교체 정책에서는 FIFO이기 때문에 먼저 들어온 2번째 블록(오른쪽 블록)이 바뀐 것이다.

Misses and Associativity in Caches (4) - Fully Associative Cache (캐시의 누락 및 연관성 4 - 완전 연관 캐시)

- Assume the following five cache accesses are given (다음과 같은 5개의 캐시 엑세스가 주어졌다고 가정한다)
  - 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)