캐시기억장치 사상

❑ 완전-연관 사상(fully-associative mapping)

- 주기억 장치 블록이 캐시의 어떤 라인으로든 적재 가능
- 태그 필드 = 주기억 장치 블록 번호

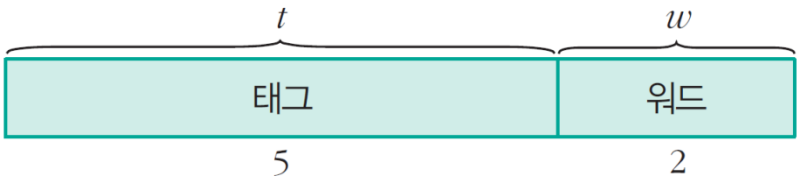
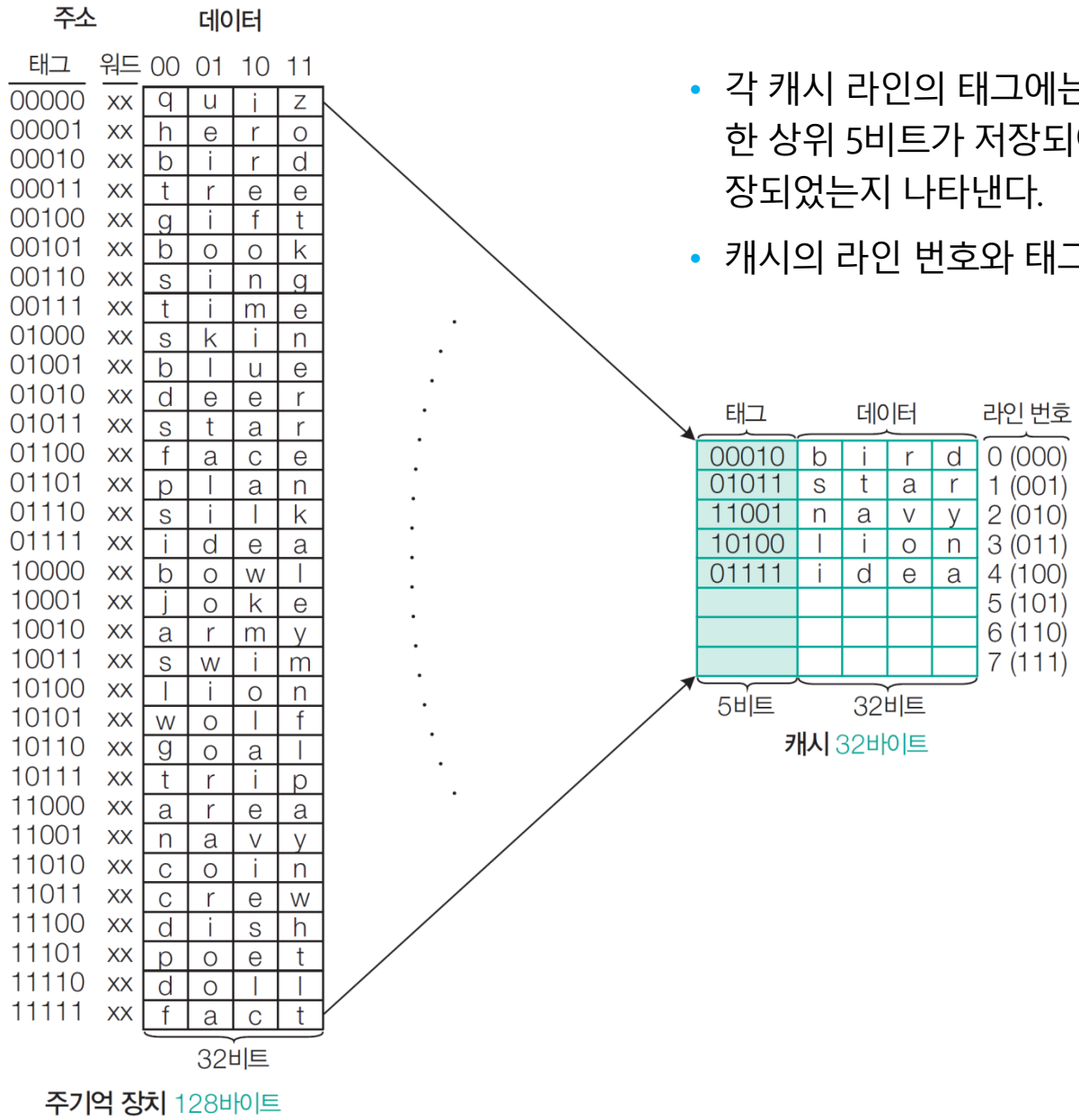


그림 6-28 완전-연관 사상에서 주소 필드의 구성

03 캐시 기억 장치



- 각 캐시 라인의 태그에는 주소의 워드 필드(2비트)를 제외한 상위 5비트가 저장되어 주기억 장치의 어느 블록이 저장되었는지 나타낸다.
- 캐시의 라인 번호와 태그 필드는 관련이 없다.

그림 6-29 완전-연관 사상의 예

❖ 완전-연관 사상에서 캐시 내부 구성 및 읽기 동작

❶ 캐시로 주기억 장치 주소 11001 11이 보내진다. 태그 필드는 11001이고, 워드 필드는 11이다.

❷ 캐시의 모든 라인의 태그들과 주기억 장치 주소의 태그 필드 내용을 비교 한다.

❸ 2번 캐시 라인의 11001과 주소의 태그 필드 11001이 일치하므로 캐시가 히트된 것이다.

❹ 다음에는 주소의 워드 필드가 11이므로 navy 중에서 y가 인출되어 CPU로 전송된다.

❺ 그러나 태그 비트가 일치하지 않으면 캐시가 미스된 것이므로 주소 전체가 주기억 장치로 보내져서 해당 블록을 인출해 온다. 인출된 블록은 5번 캐시 라인에 저장된다.

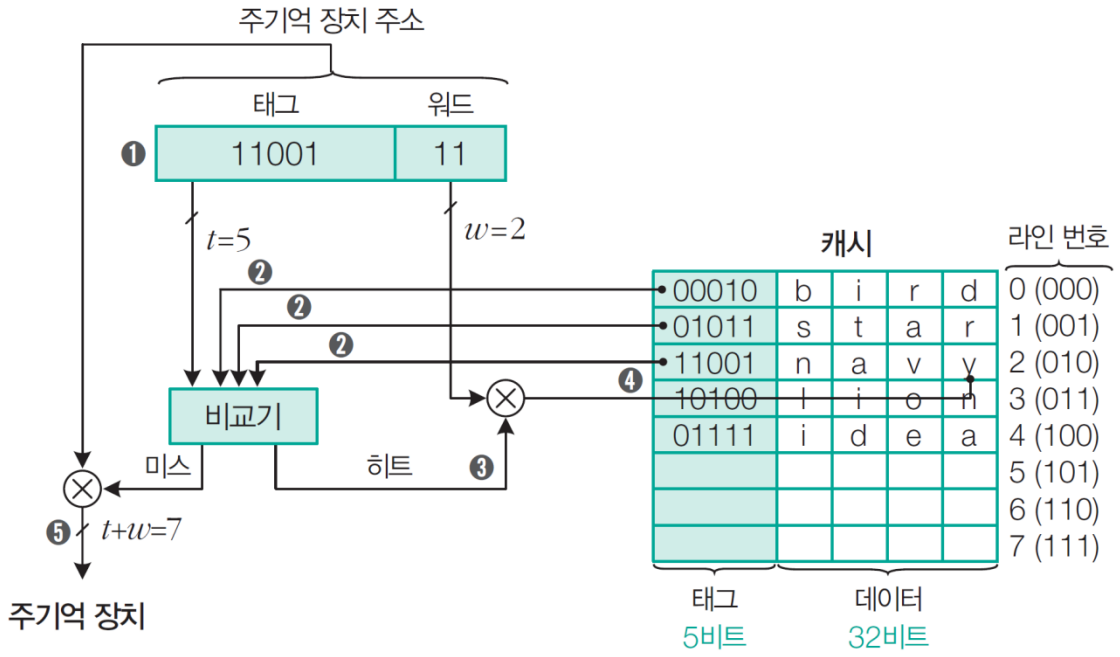


그림 6-30 완전-연관 사상에서 캐시 내부 구성 및 동작

예제 6-7

완전-연관 사상 캐시의 라인들이 [그림 6-29]와 같이 저장되어 있다고 가정하자. 이때 CPU에서 발생한 주소가 다음과 같은 경우 히트인지 미스인지 구별하여라. 미스인 경우 주기억 장치에서 데이터를 인출하여 해당 라인에 적재된 후의 결과에 대해 설명하라.

(a) 01011 01 (b) 10010 11
(c) 00010 10 (d) 11100 00

풀이

(a) **히트** : 1번 라인에 적재되어 있으며, 인출 데이터는 star에서 t다.

(b) **미스** : 비어 있는 첫 번째 라인인 5번 라인에 적재되므로 데이터 필드에는 army가 적재되고, 태그는 10010으로 변경된다. 최종 인출 데이터는 army에서 y다.

(c) **히트** : 0번 라인에 적재되며, 인출 데이터는 bird에서 r이다.

(d) **미스** : 라인 순으로 6번 라인에 적재되므로 데이터 필드에는 dish가 적재되고, 태그는 11100으로 변경된다. 최종 인출 데이터는 dish에서 d다.

End of Example

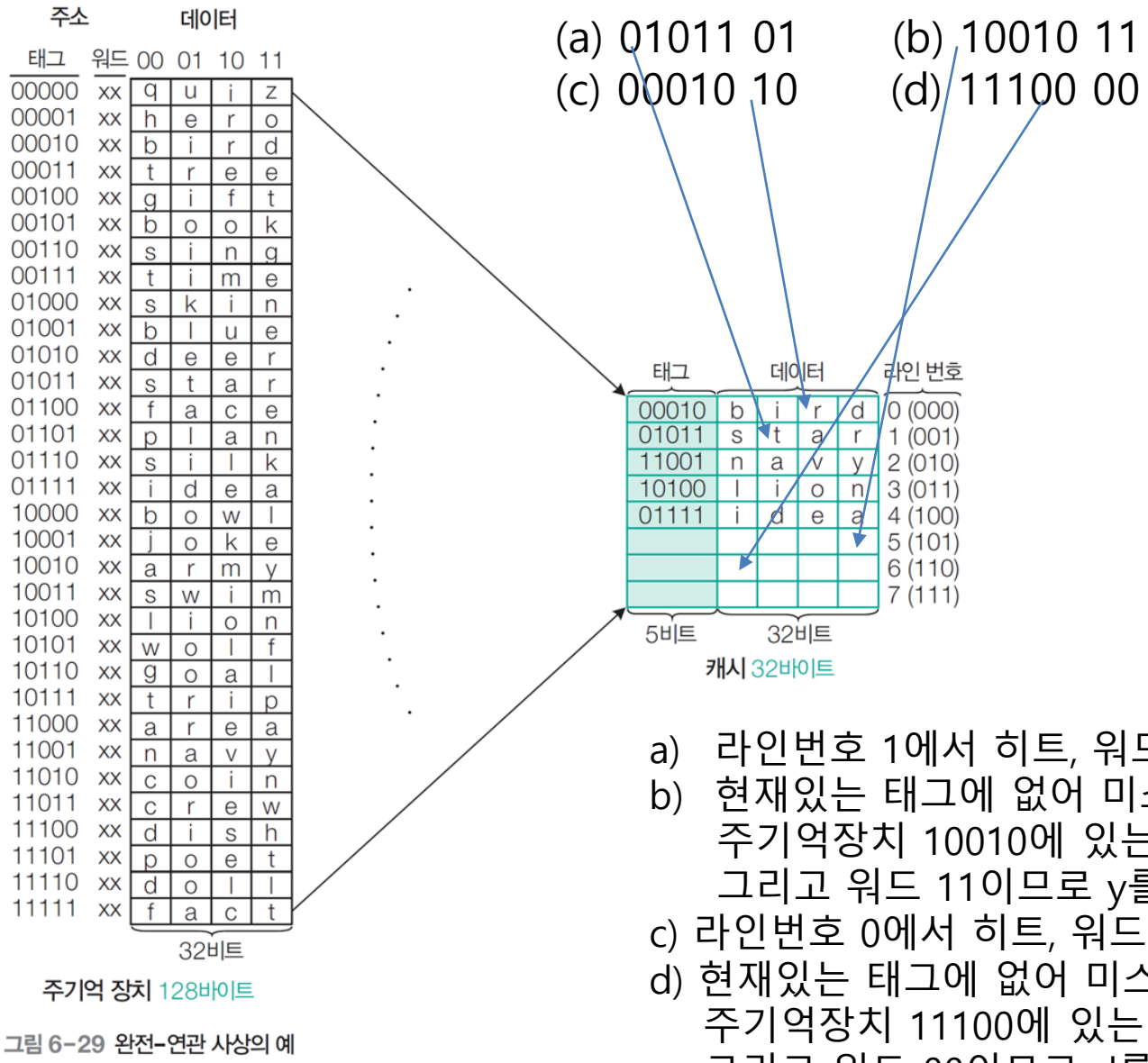


그림 6-29 완전-연관 사상의 예

a) 라인번호 1에서 히트, 워드01이므로 t인출
b) 현재있는 태그에 없어 미스, 5번 라인에 주기억장치 10010에 있는 army 가져옴 그리고 워드 11이므로 y를 인출
c) 라인번호 0에서 히트, 워드10이므로 r인출
d) 현재있는 태그에 없어 미스, 6번 라인에 주기억장치 11100에 있는 dish 가져옴 그리고 워드 00이므로 d를 인출

❑ 세트-연관 사상(set-associative mapping)

- 직접 사상과 완전-연관 사상의 조합
- 주기억 장치 블록 그룹이 하나의 캐시 세트를 공유하며, 그 세트에는 두 개 이상의 라인들이 적재될 수 있음
- 전체 캐시 라인(m)은 v개의 세트들로 나누어지며, 각 세트들은 k개의 라인들로 구성(k-way)
- 주기억 장치 블록(j)이 적재될 수 있는 캐시 세트의 번호 i :

$$m = v \times k$$
$$i = j \bmod v$$

- 앞의 예에서 캐시 라인의 수는 m=8이다. 세트당 캐시 라인의 수가 k=2라면 세트 수는 v=8/2=4개다. 따라서 캐시는 세트 4개로 구성되고, 각 세트에 캐시 라인이 2개 있다.

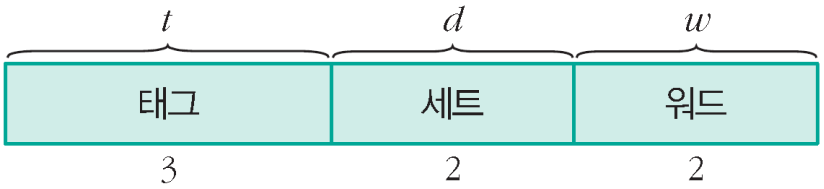


그림 6-31 세트-연관 사상에서 주소 필드의 구성

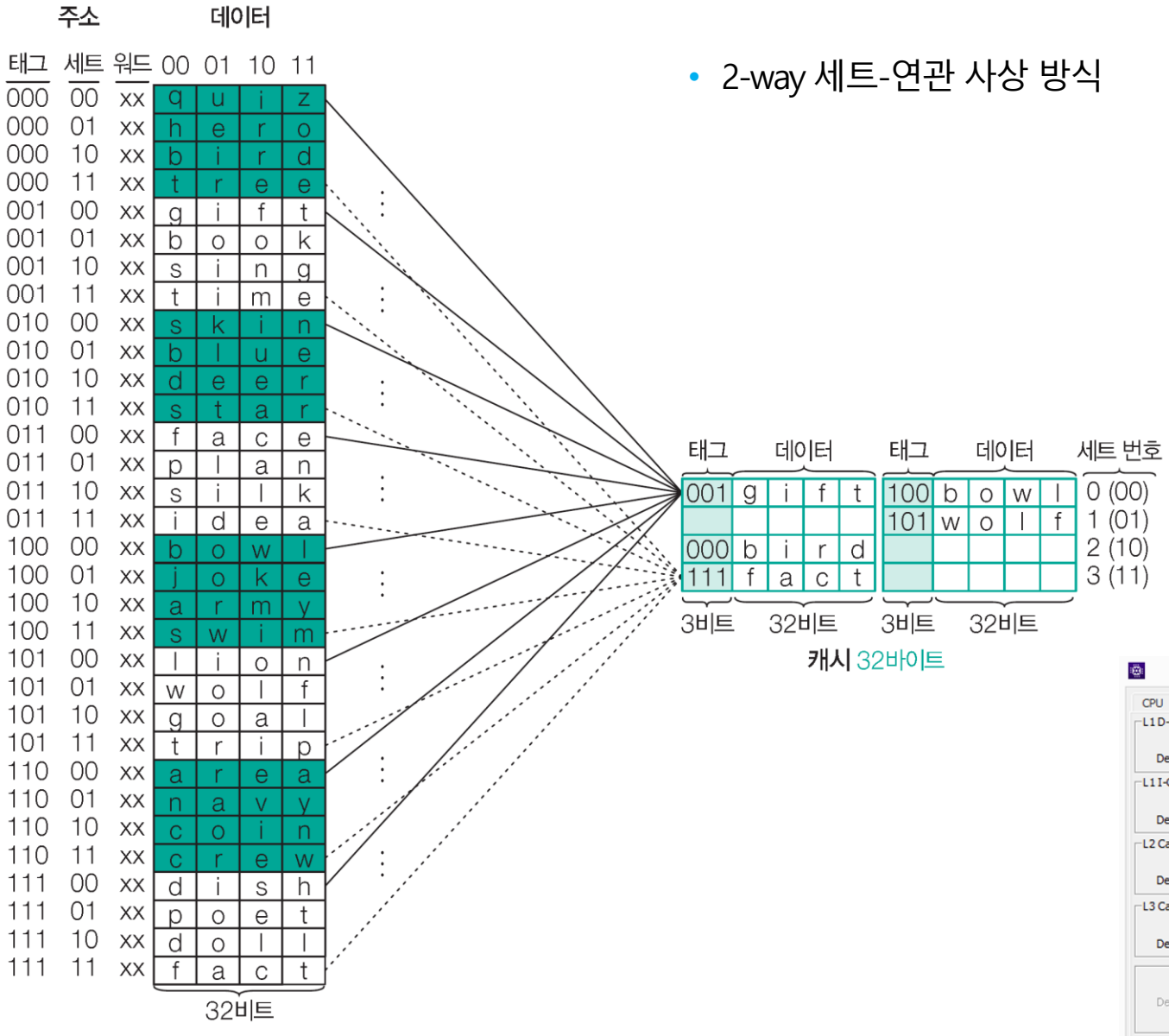
03 캐시 기억 장치

- 주기억 장치의 블록 j(=0, 1, ..., 31)가 적재될 수 있는 세트 번호는 j mod 4로 결정된다.
- 세트-연관 사상 방식에서 세트 4개, 세트당 캐시 라인 2개에 들어갈 수 있는 주기억 장치 블록 32개는 다음과 같다.

표 6-8 세트-연관 사상에서 각 세트를 공유하는 주기억 장치 블록

세트 번호	주기억 장치 블록 번호							
0(00)	00000	00100	01000	01100	10000	10100	11000	11100
1(01)	00001	00101	01001	01101	10001	10101	11001	11101
2(10)	00010	00110	01010	01110	10010	10110	11010	11110
3(11)	00011	00111	01011	01111	10011	10111	11011	11111

03 캐시 기억 장치



- 2-way 세트-연관 사상 방식

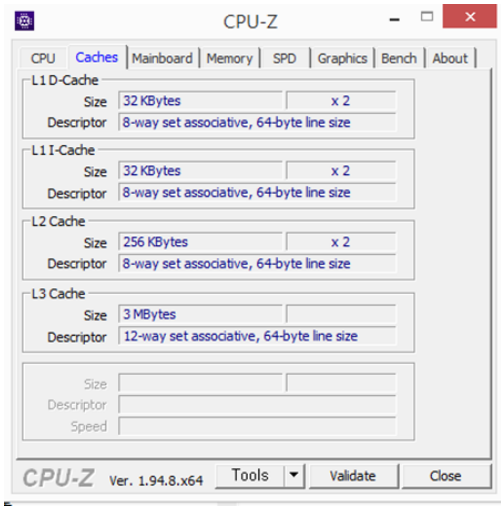


그림 6-32 세트-연관 사상의 예

03 캐시 기억 장치

❖ 세트-연관 사상에서 캐시 내부 구성 및 읽기 동작

❶ 주기억 장치 주소 001 00 10이 캐시로 보내진다(태그 필드=001, 세트 필드=00, 워드 필드= 10).

❷ 00 세트 번호를 이용해 캐시의 0번 세트를 선택한다.

❸ 0 번 세트 라인의 태그에 주기억 장치 주소의 태그 비트0 01과 일치하는 것이 있는지 검사한다.

❹ 0번 세트 내 첫 번째 라인의 태그 비트가 001이므로 히트되었다.

❺ 다음에는 주소의 워드 필드가 10이므로 gift 중에서 f가 인출되어 CPU로 전송된다.

❻ 그러나 태그 비트가 일치하지 않으면 캐시가 미스된 것이므로 주소 전체가 주기억 장치로 보내져서 해당 블록을 인출해 온다. 적절한 교체 알고리즘에 의하여 2개 중 하나를 선택하여 그 라인에 새로운 블록을 적재해야 한다.

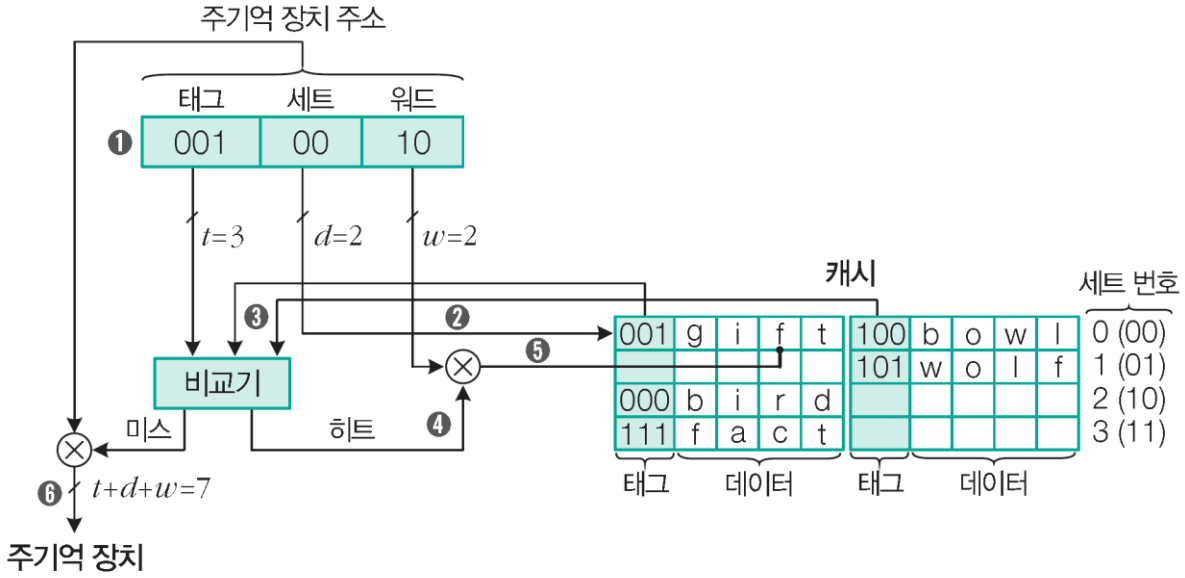


그림 6-33 세트-연관 사상에서 캐시 내부 구성 및 동작

예제 6-8

세트-연관 사상 캐시의 라인들이 [그림 6-32]와 같이 저장되어 있다고 가정하자. 이때 CPU에서 발생한 주소들이 다음과 같은 경우 히트인지 또는 미스인지 구별하여라. 그리고 미스인 경우 주기억 장치에서 데이터를 인출하여 해당 라인에 적재된 후 결과를 설명하여라.

(a) 100 00 01 (b) 011 10 11

(c) 111 11 10 (d) 010 00 00

풀이

(a) **히트** : 0번 세트의 두 번째 라인에 적재되어 있으며, 인출 데이터는bowl에서 o이다.

(b) **미스** : 적재될 수 있는 2번 세트의 첫 번째 라인의 태그 000과 주소의 태그 011이 일치하지 않는다. 따라서 빈 두 번째 라인에 silk가 적재되고, 태그는 011로 변경된다. 최종적으로 인출 데이터는 silk에서 k다.

(c) **히트** : 3번 세트의 첫 번째 라인에 적재되어 있으며, 인출 데이터는fact에서 c다.

(d) **미스** : 이 블록이 적재될 수 있는 0번 세트의 두 라인의 태그(001, 100)가 주소의 태그 010과 일치하지 않는다. 편의상 새로운 블록이 두 번째 라인에 적재된다고 가정하면 skin이 적재되고, 태그는 010으로 변경된다. 최종 인출 데이터는 skin에서 s다.

End of Example

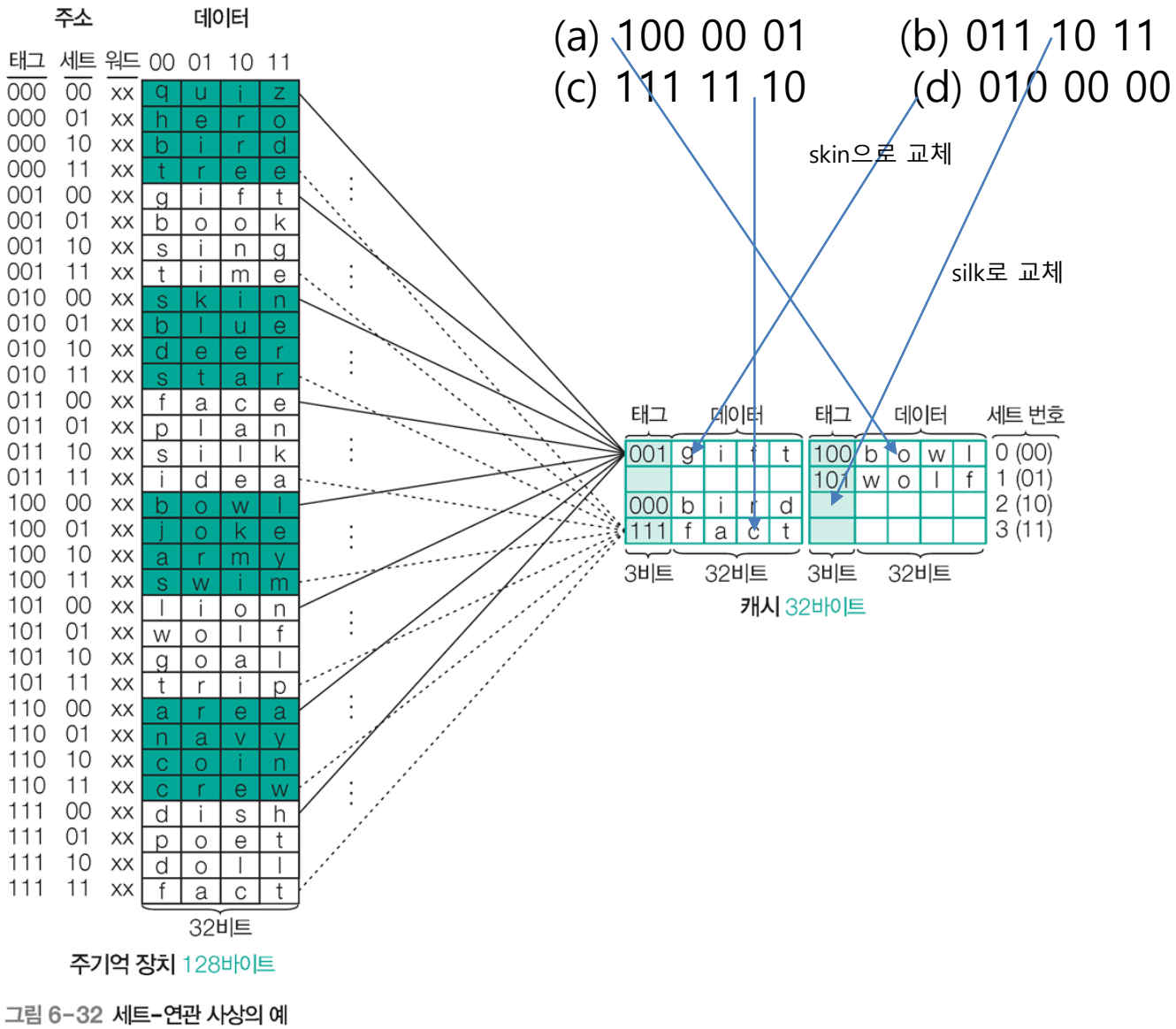


그림 6-32 세트-연관 사상의 예

❖ 사상 방식의 비교

표 6-9 사상 방식의 비교

사상 기법	단순성	태그 연관 검색	캐시 효율	교체 기법
직접	단순	없음	낮음	불필요
완전-연관	복잡	연관	높음	필요
세트-연관	중간	중간	중간	필요

라인 번호에 의해
자리가 고정되있어
미스시에는 라인번호에 따라
교체됨

3 교체 알고리즘

- 세트-연관 사상에서 주기억 장치로부터 새로운 블록이 캐시로 적재될 때, 만약 세트 내 모든 라인들이 다른 블록들로 채워져 있다면, 그들 중의 하나를 선택하여 새로운 블록으로 교체
- 교체 알고리즘 : 캐시 히트율을 극대화할 수 있도록 교체할 블록을 선택하기 위한 알고리즘
  - **LRU**(Least Recently Used) : 사용되지 않은 채로 가장 오래 있었던 블록을 교체하는 방식
  - **FIFO**(First-In-First-Out) 알고리즘 : 캐시에 적재된 지 가장 오래된 블록을 교체하는 방식
  - **LFU**(Least Frequently Used) 알고리즘 : 참조되었던 횟수가 가장 적은 블록을 교체하는 방식
  - **Random** : 사용 횟수를 고려하지 않고 후보 캐시 라인 중 임의로 선택하여 교체하는 방식

예제 6-9

FIFO 교체 알고리즘을 사용하는 세트-연관 사상 캐시로 다음 블록을 연속으로 액세스하는 경우 각 캐시 라인에 적재되는 블록을 표시하고 히트율을 구하여라. 단, 각 세트의 라인 수는 (a) 2개, (b) 4개라고 가정한다.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

풀이

(a) 캐시 라인 수가 2개인 경우 : 히트율=6/20

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	1	1	0	0	0	4	4	3	3	3	3	1	1	1	1	7	7	7
		0	0	2	2	3	3	3	2	0	0	0	2	2	2	0	0	0	0	1
							hit				hit	hit			hit		hit		hit	

(b) 캐시 라인 수가 4개인 경우 : 히트율=10/20

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	7	7	3	3	3	3	3	3	3	3	3	2	2	2	2	2	2
		0	0	0	0	0	0	4	4	4	4	4	4	4	4	4	4	7	7	7
			1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0
				2	2	2	2	2	2	2	2	2	2	1	1	1	1	1	1	1
					hit		hit		hit	hit		hit	hit			hit	hit		hit	hit

End of Example

예제 6-10

LRU 교체 알고리즘을 사용하는 세트-연관 사상 캐시로 다음 블록을 연속으로 액세스하는 경우 각 캐시 라인에 적재되는 블록을 표시하고 히트율을 구하여라. 단, 각 세트의 라인 수는 (a) 2개, (b) 4개라고 가정한다.

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

풀이

(a) 캐시 라인 수가 2개인 경우 : 히트율=3/20

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	1	1	0	0	0	0	2	2	0	0	2	2	2	2	1	1	0	0
		0	0	2	2	3	3	4	4	3	3	3	3	1	1	0	0	7	7	1
							hit					hit			hit					

(b) 캐시 라인 수가 4개인 경우 : 히트율=12/20

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
	7	7	7	7	7	3	3	3	3	3	3	3	3	3	3	3	3	7	7	7
		0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
			1	1	1	1	1	4	4	4	4	4	4	1	1	1	1	1	1	1
				2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
					hit		hit		hit	hit	hit	hit	hit		hit	hit	hit		hit	hit

End of Example

4 쓰기 정책

- 캐시의 블록이 변경되었을 때 그 내용을 주기억 장치에 갱신하는 시기와 방법의 결정

❖ Write-through

- 모든 쓰기 동작들이 캐시 뿐만 아니라 주기억 장치로도 동시에 수행되는 방식

❖ Write-back

- 캐시에서 데이터가 변경되어도 주기억 장치에는 갱신되지 않는 방식

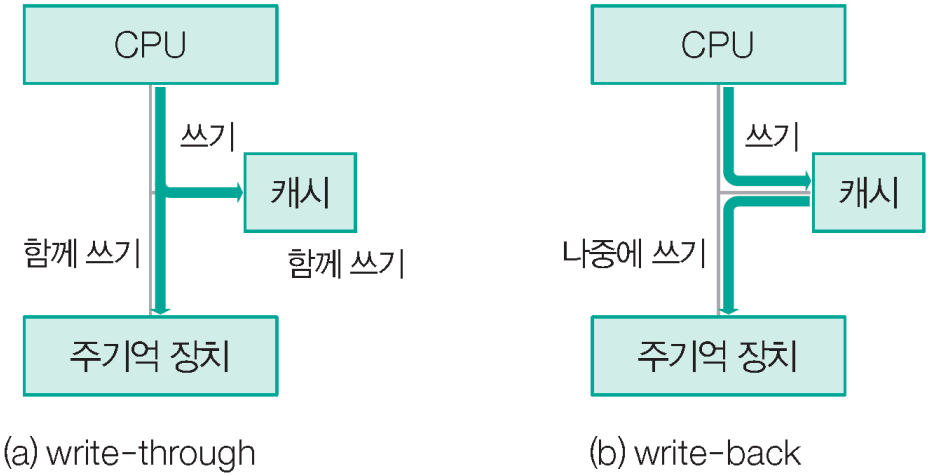


그림 6-34 쓰기 정책

03 캐시 기억 장치

❖ 각 방식의 장단점

Write-through

장점	• 캐시에 적재된 블록의 내용과 주기억 장치에 있는 그 블록의 내용이 항상 같다.
단점	• 모든 쓰기 동작이 주기억 장치 쓰기를 포함하므로, 쓰기 시간이 길어진다.

Write-back

장점	• 기억장치에 대한 쓰기 동작의 횟수가 최소화되고, 쓰기 시간이 짧아진다.
단점	• 캐시의 내용과 주기억 장치의 해당 내용이 서로 다르다.

⇨ 블록을 교체할 때는 캐시의 상태를 확인하여 주기억 장치에 갱신하는 동작이 선행되어야 하며, 그를 위하여 각 캐시 라인이 상태 비트(status bit)를 가지고 있어야 한다.

예제 6-11

주기억 장치 액세스 시간이 50ns, 캐시 액세스 시간이 5ns인 시스템이 있다. 전체 액세스 요구 중에서 70%는 읽기 동작이고, 30%는 쓰기 동작이었으며, 캐시 히트율은 90%였다. 캐시 쓰기 정책이 (a) write-through일 때와 (b) write-back일 때 평균 기억 장치 액세스 시간을 각각 구하여라. 단, write-back에서 미스가 발생한 경우 새로운 블록을 적재하기 위해 교체할 라인들의 20%는 이미 수정된 상태에 있었다고 가정한다.

풀이

(a) Write-through 정책인 경우

- 읽기동작의 평균시간 = 0.9×5ns+0.1×50ns = 9.5ns
- 쓰기동작의 평균시간 = 50ns (모든 쓰기 동작은 주기억 장치를 액세스)
- 평균 기억장치 액세스 시간 = 0.7×9.5ns+0.3×50ns = 21.65ns

(b) Write-back 정책인 경우 읽기동작과 쓰기동작에 같은 시간이 걸리므로,
평균 기억장치 액세스 시간 = 0.9×5ns+0.1×(50ns+0.2×50ns) = 10.5ns

End of Example

수고하셨습니다!