캐시(cache)메모리

03 캐시 기억 장치

❖ 캐시의 사용 목적

- CPU와 주기억 장치의 속도 차이로 인한 CPU 대기 시간을 최소화 시키기 위하여 CPU와 주기억 장치 사이에 설치하는 고속 반도체 기억장치

❖ 캐시의 특징

- 주기억 장치보다 액세스 속도가 높은 칩 사용
- 가격 및 제한된 공간 때문에 용량이 적다.

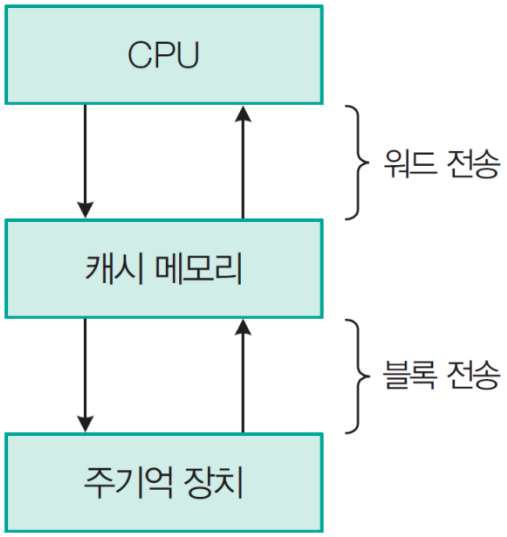


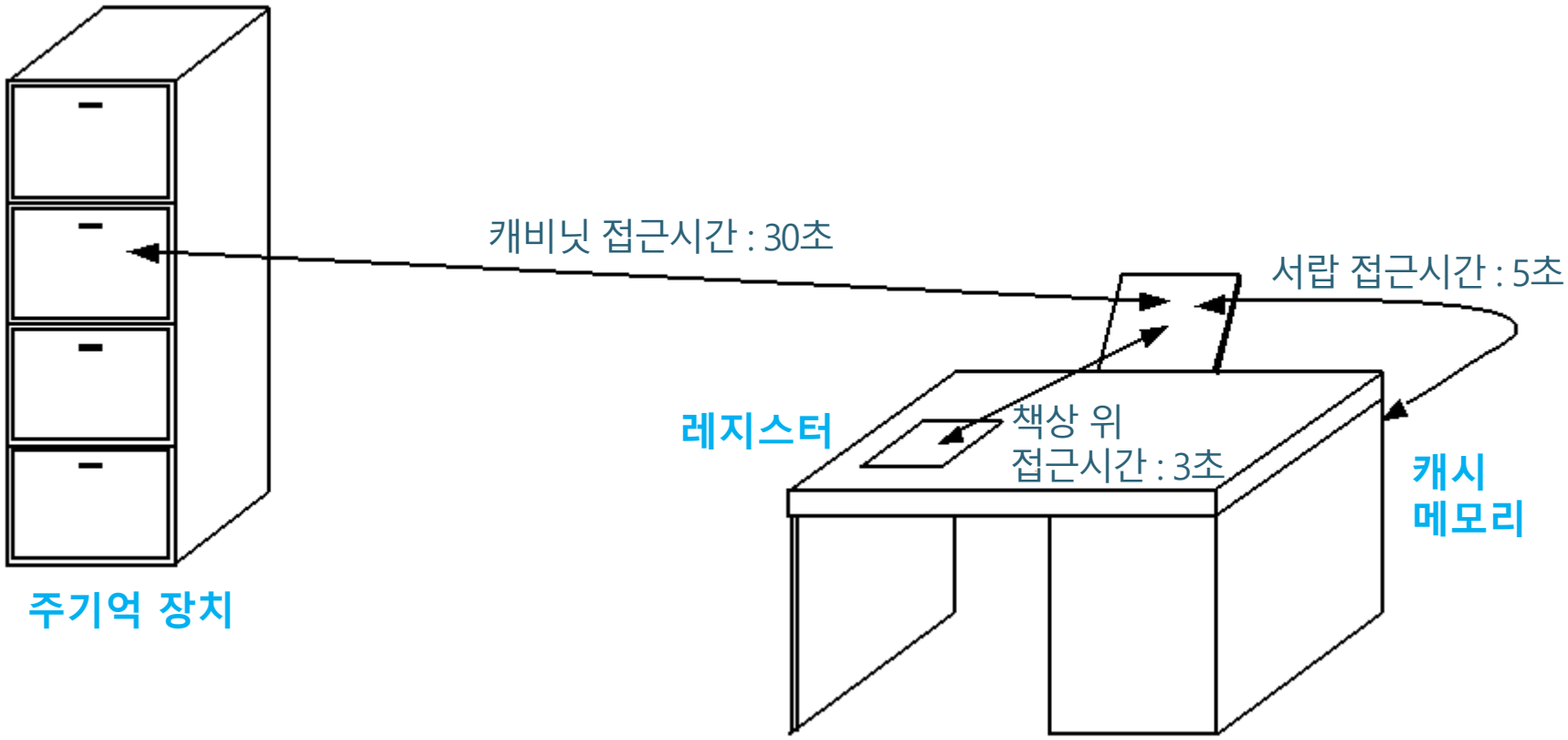
그림 6-20 캐시 위치

워드 : CPU가 한번에 처리 할 수 있는 데이터량

블록 : 워드가 일정 크기 만큼 모인량

03 캐시 기억 장치

❖ **책상 위**(레지스터), **서랍**(캐시), **파일 캐비닛**(주기억 장치)**의 비교**



03 캐시 기억 장치

❖ 캐시의 기본적인 동작 흐름도

- CPU가 기억 장치에서 어떤 정보(명령어 또는 데이터)를 읽으려는 경우 먼저 해당 정보가 캐시에 있는지 검사한다.
- 있다면 해당 정보를 즉시 읽어 들이고, 없다면 해당 정보를 주기억 장치에서 캐시로 적재한 후 읽어 들인다.

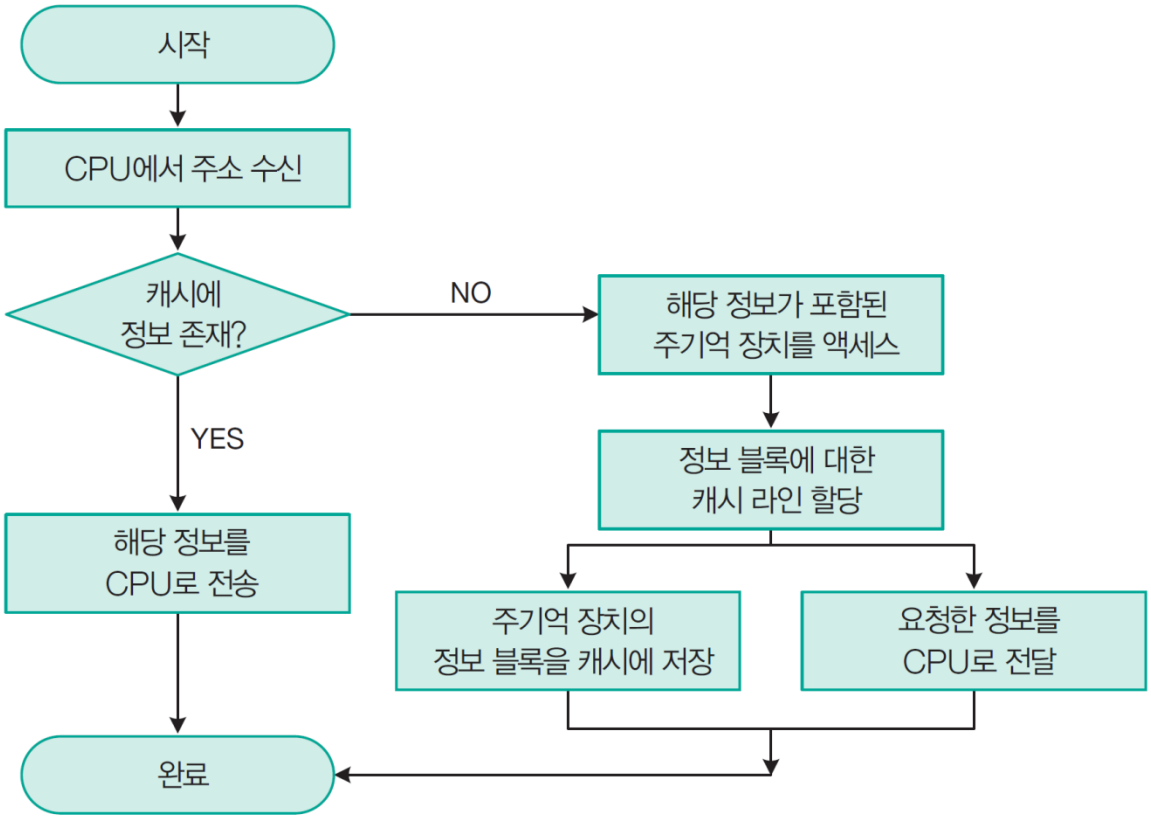


그림 6-21 캐시 읽기 동작 순서

03 캐시 기억 장치

❖ 캐시의 히트율 및 평균 액세스 시간

- 캐시 히트(cache hit) : CPU가 원하는 데이터가 캐시에 있는 상태
- 캐시 미스(cache miss) : CPU가 원하는 데이터가 캐시에 없는 상태. 이 경우에는 주기억 장치로부터 데이터를 읽어온다.
- 히트률(hit ratio) : 캐시에 히트되는 정도(H)

$$H = \frac{\text{캐시에 히트되는 횟수}}{\text{전체 기억장치 액세스 횟수}}$$

- 캐시의 미스율(miss ratio) = (1 - H)
- 평균 기억장치 액세스 시간(T_a)

$$T_a = H \times T_c + (1-H) \times (T_c + T_m)$$
$$\approx H \times T_c + (1-H) \times T_m$$

$T_c + T_m \approx T_m$

단, T_c 는 캐시 액세스 시간, T_m 은 주기억 장치 액세스 시간

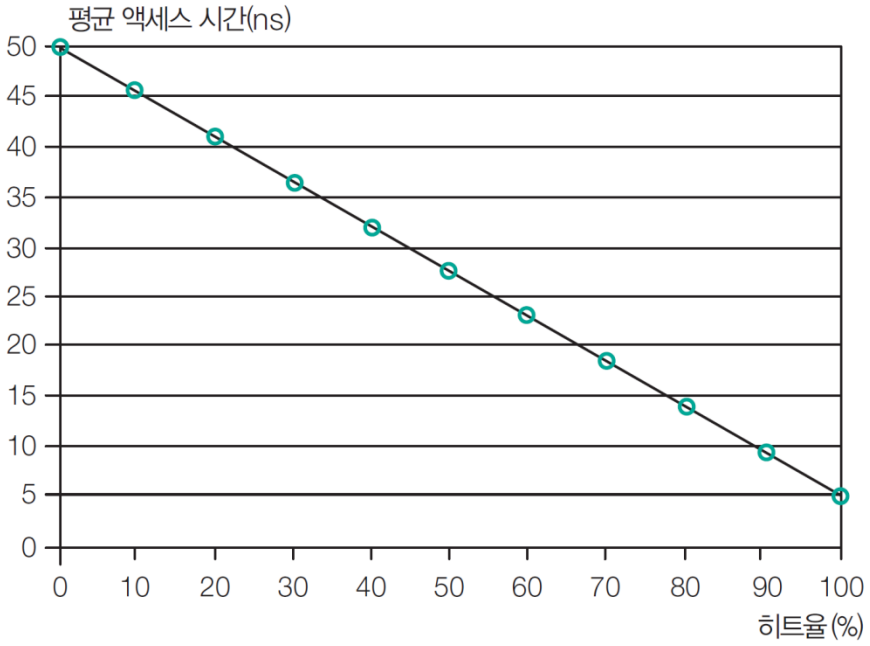
예제 6-5

캐시의 액세스 시간이 T_c=5ns이고, 주기억 장치의 액세스 시간이 T_m=50ns인 기억 장치 시스템에서 캐시 히트율이 0%부터 100%까지 10% 간격으로 변할 때 평균 액세스 시간 T_a를 구하고 이를 그래프로 나타내어라. 결과도 설명하여라. 단, 그래프의 x축은 히트율, y축은 평균 액세스 시간으로 한다.

풀이

히트율(%)	평균 액세스 시간(ns)
0	50.0
10	45.5
20	41.0
30	36.5
40	32.0
50	27.5
60	23.0
70	18.5
80	14.0
90	9.5
100	5.0

캐시의 히트률이 높아질수록 평균 기억장치 액세스시간은 캐시 액세스 시간에 접근



End of Example

❖ 참조 지역성(locality of reference)

- **공간적 지역성**(spatial locality) : 기억장치 내에 인접하여 저장되어 있는 데이터들이 연속적으로 액세스될 가능성이 높다. 예를 들어 표나 배열의 데이터들이 저장되어 있는 기억 장치 영역은 관련 연산이 수행되는 동안 자주 액세스된다.
- **시간적 지역성**(temporal locality) : 최근에 액세스된 프로그램이나 데이터가 가까운 미래에 다시 액세스될 가능성이 높다. 예를 들어 서브루틴이나 루프 프로그램들은 반복적으로 호출되며, 공통 변수들도 자주 액세스된다.

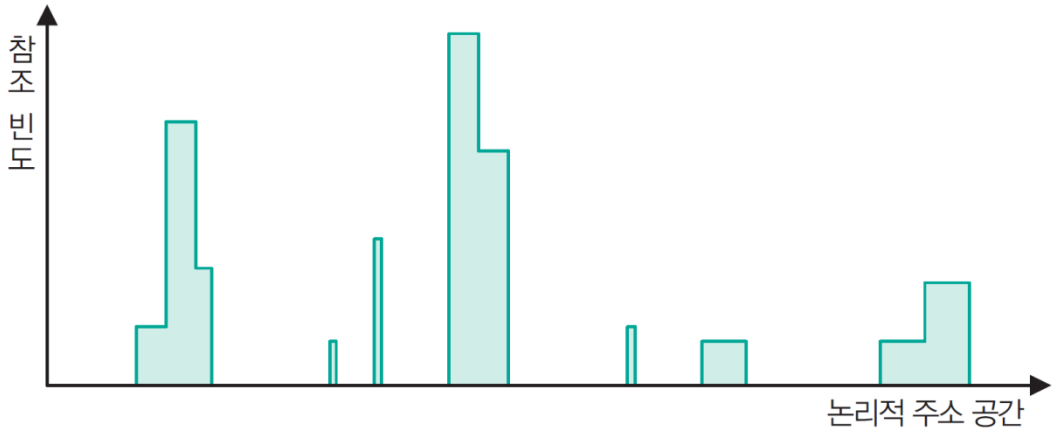


그림 6-22 주소 참조의 전형적인 비균등 분포

03 캐시 기억 장치

❖ 캐시 설계에 있어서의 공통적인 목표

- 캐시의 히트율을 극대화해야 한다.
- 캐시 히트인 경우 캐시에서 정보를 읽어 오는 시간을 최소화해야 한다.
- 캐시 미스인 경우 주기억 장치에서 캐시로 정보를 가져오는 데 걸리는 시간을 최소화해야 한다.
- 캐시의 내용이 변경되었을 경우 주기억 장치에 해당 내용을 갱신하는 데 소요되는 시간을 최소화해야 한다.

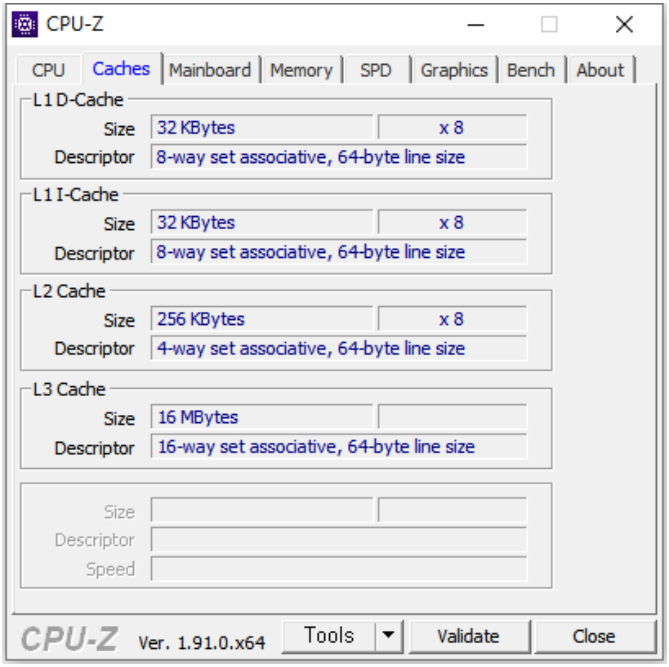
❖ 캐시 설계의 설계 요소

표 6-6 캐시의 설계 요소

캐시 용량	클수록 좋지만 가격과 절충 필요	쓰기 정책	write-through
사상 방식	직접 사상		write-back
	연관 사상 (주기억 장치의 블록이 어느 캐시 라인에 들어갈 것인지 결정하는 방법)	라인 크기	캐쉬와 주기억장치 사이 정보가 이동되는 블록의 단위
	세트-연관 사상	캐시 수	단일 레벨 또는 다중 레벨
교체 알고리즘	Least Recently Used LRU		단일 캐시 또는 분리 캐시
	First In First Out FIFO		
	Least Frequently Used LFU		
	Random		

1 캐시 용량

- 용량이 커질수록 히트율은 높아지지만, 비용이 증가
- 용량이 커질수록 주소 해독 및 정보 인출을 위한 주변 회로가 복잡해지므로 액세스 시간이 다소 더 길어진다.
- CPU 칩 또는 메인보드의 공간에도 제한을 받는다.



2 사상 방식 : 주기억 장치의 블록이 어느 캐시 라인에 들어갈 것인지 결정하는 방법

- **블록**(block) : 주기억 장치와 캐시 사이에 이동되는 정보 단위
- 주기억 장치 용량 = 2^n 워드, 블록 = K개 워드 → 블록의 수 = $2^n/K$ 개
- **라인**(line) : 캐시에서 각 블록이 저장되는 장소. 라인 수 m개, 각 라인에는 워드 K개가 저장된다.
- **태그**(tag) : 라인에 적재된 블록을 구분해주는 정보

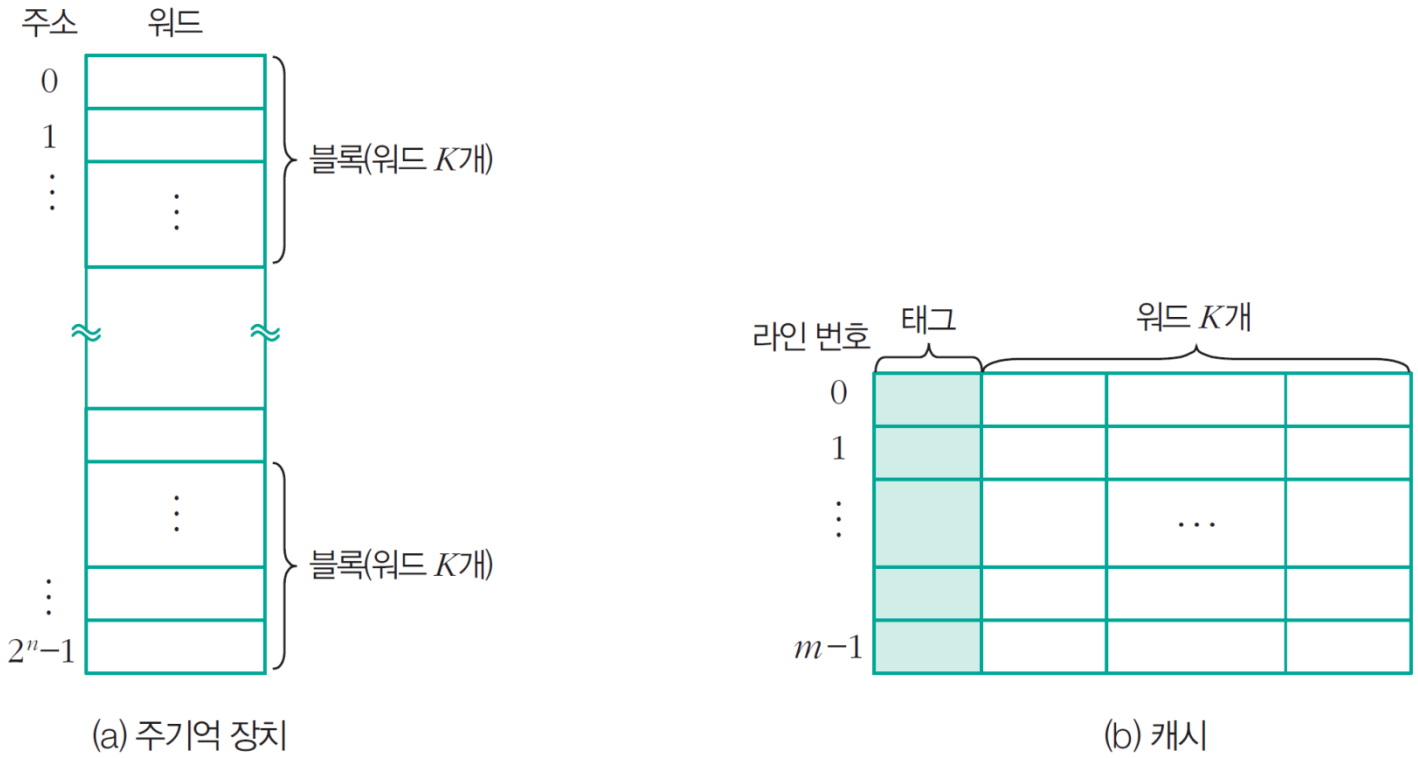


그림 6-23 주기억 장치와 캐시의 구성

03 캐시 기억 장치

❖ 사상 방식(mapping scheme)

사상 방식이란 주기억 장치의 블록이 어느 캐시 라인에 들어갈 것인지 결정하는 방법

- **직접 사상(direct mapping) : 주기억장치 블록이 지정된 하나의 캐쉬 라인에 들어감**
- **완전-연관 사상(fully-associative mapping) : 주기억장치 블록이 캐쉬의 어떤 라인이든 적재 가능**
- **세트-연관 사상(set-associative mapping) : 직접 사상과 완전 연관 사상의 조합 방식**

❖ 사상 방식을 설명하기 위한 모델 예

- 주기억 장치의 용량은 128(=2^7)바이트
- 주기억 장치 주소 = 7비트 (바이트 단위로 주소가 지정, 워드 길이는 1바이트)
- 블록 크기 = 4바이트 → 주기억 장치에는 128/4=32개의 블록이 있다.
- 캐시 크기 = 32 바이트, 캐시 라인의 크기 = 4바이트(블록 크기와 동일)
- 전체 캐시 라인의 수, m = 32/4 = 8개

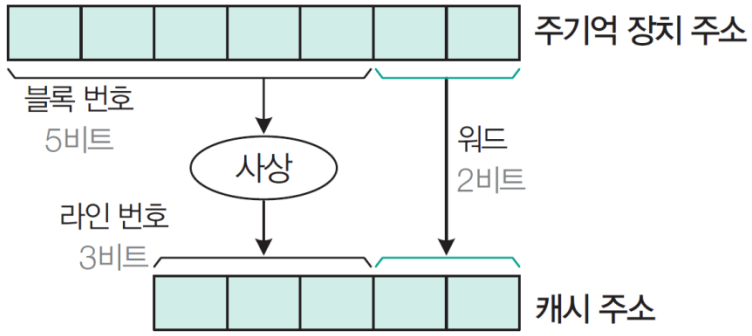


그림 6-24 주기억 장치와 캐시의 주소 사상

❑ 직접 사상 (direct mapping)

- 주기억 장치의 블록들이 지정된 하나의 캐시 라인으로만 적재됨
- 주기억 장치 주소는 필드 3개로 구성된다. 주기억 장치의 주소 지정에는 7비트가 필요하므로 $t+s+w=7$이다.
- 한 블록 내에는 워드 4개가 들어가므로 각 워드를 구별하기 위해 $w=2$비트가 필요하고, 캐시 라인의 수가 8개이므로 각각을 구별하기 위해 $s=3$비트, 그리고 태그 필드는 나머지 $t=2(=7-3-2)$비트가 된다.
- 태그 필드 값은 캐시 라인을 공유하는 주기억 장치 블록들을 서로 구분하는 데 사용된다.

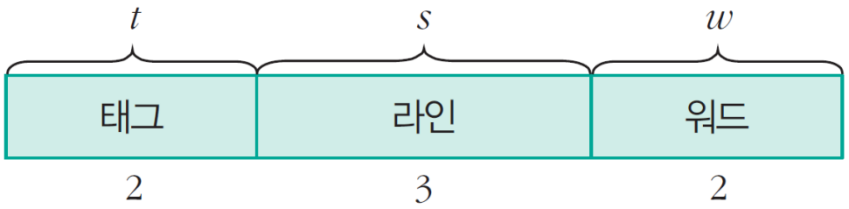


그림 6-25 직접 사상에서 주소 필드의 구성

03 캐시 기억 장치

- 주기억 장치의 블록 j(=0, 1, …, 31)가 적재될 수 있는 캐시 라인 번호는 다음 연산으로 결정된다.

$$i = j \bmod m$$

- [표 6-7]에는 각 캐시 라인을 공유하는 블록들의 번호 4(=2^t=2^2)개가 나열되어 있다. 여기서 블록 번호는 주기억 장치 주소의 상위 5비트에 해당한다.

표 6-7 직접 사상에서 각 캐시 라인을 공유하는 주기억 장치 블록들

캐시 라인	주기억 장치 블록 번호			
0(000)	00000	01000	10000	11000
1(001)	00001	01001	10001	11001
2(010)	00010	01010	10010	11010
3(011)	00011	01011	10011	11011
4(100)	00100	01100	10100	11100
5(101)	00101	01101	10101	11101
6(110)	00110	01110	10110	11110
7(111)	00111	01111	10111	11111

03 캐시 기억 장치

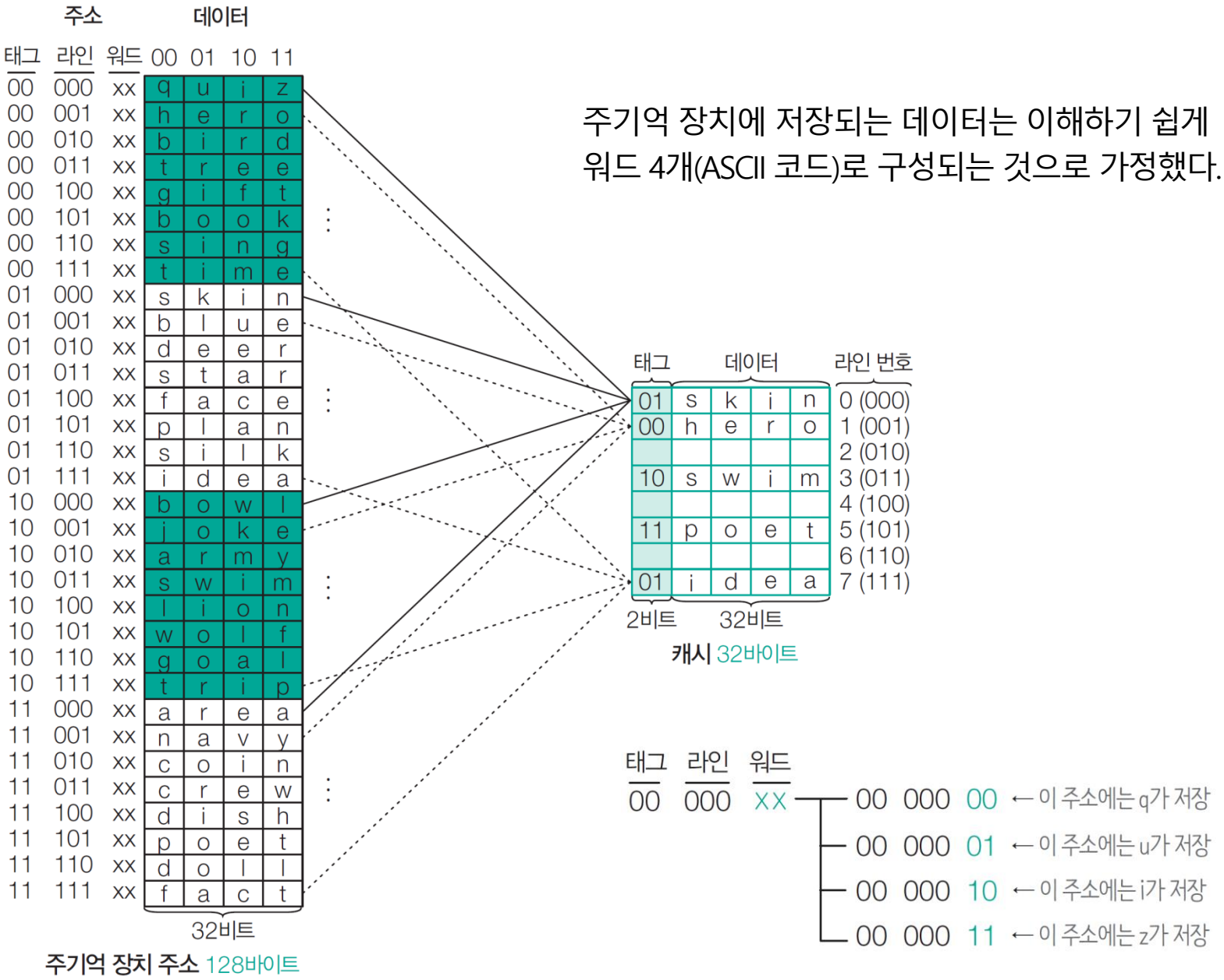


그림 6-26 직접 사상의 예

03 캐시 기억 장치

❖ 직접 사상에서 캐시 내부 구성 및 읽기 동작

❶ 캐시로 주기억 장치 주소 11 101 01이 보내진다(태그 필드=11, 라인 필드=101, 워드 필드=01).

❷ 라인 필드가 101이므로 5번 캐시 라인이 선택된다.

❸ 선택된 5번 라인의 태그 비트 11을 읽어서, ❹ 주기억 장치 주소의 태그 필드인 11과 비교한다.

❺ 두 태그 비트가 일치하므로 캐시가 히트된 것이다.

❻ 다음에는 주소의 워드 필드가 01이므로 poet 중에서 o(1110 1111)가 인출되어 CPU로 전송된다.

❼ 그러나 태그 비트가 일치하지 않으면 캐시가 미스된 것이므로 주소 전체가 주기억 장치로 보내져서 해당 블록을 인출해 온다. 인출된 블록은 지정된 캐시 라인에 저장된다. 그런데 그 라인을 공유하는 다른 블록이 이미 저장되어 있는 상태라면 원래 블록은 지워지고 새로 인출된 블록이 저장된다.

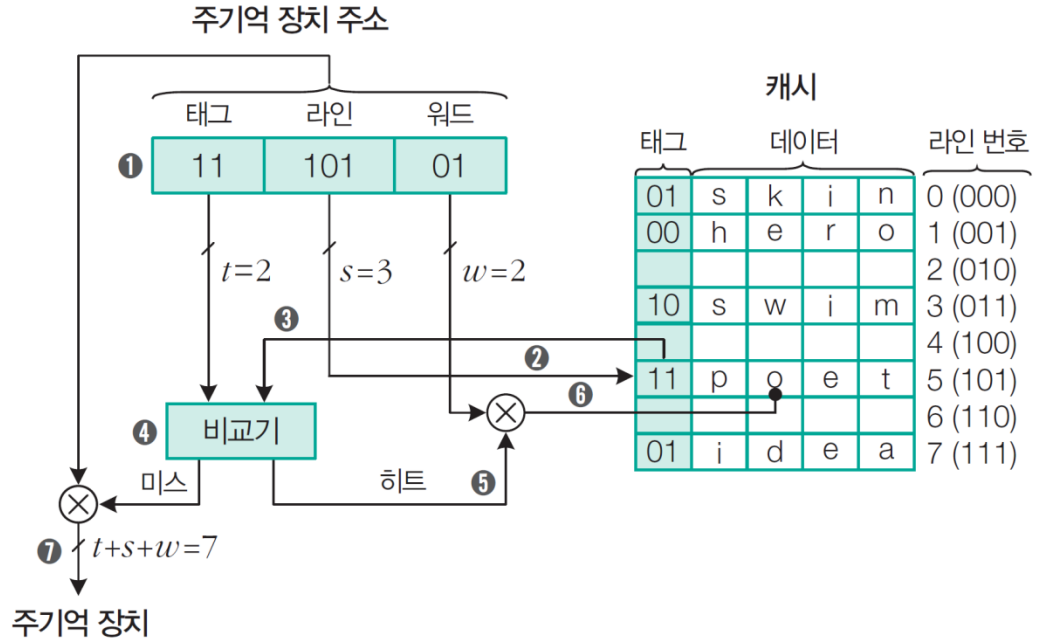


그림 6-27 직접 사상에서 캐시 내부 구성 및 동작

예제 6-6

직접 사상 캐시의 라인들이 [그림 6-26]과 같이 저장되어 있다고 가정하자. 이때 CPU에서 발생한 주소들이 다음과 같은 경우 히트인지 또는 미스인지 구별하여라. 미스인 경우 주기억 장치에서 데이터를 인출하여 해당 라인에 적재된 후의 결과를 설명하라.

(a) 00 001 01 (b) 01 010 11 (c) 10 011 10 (d) 11 111 00

풀이

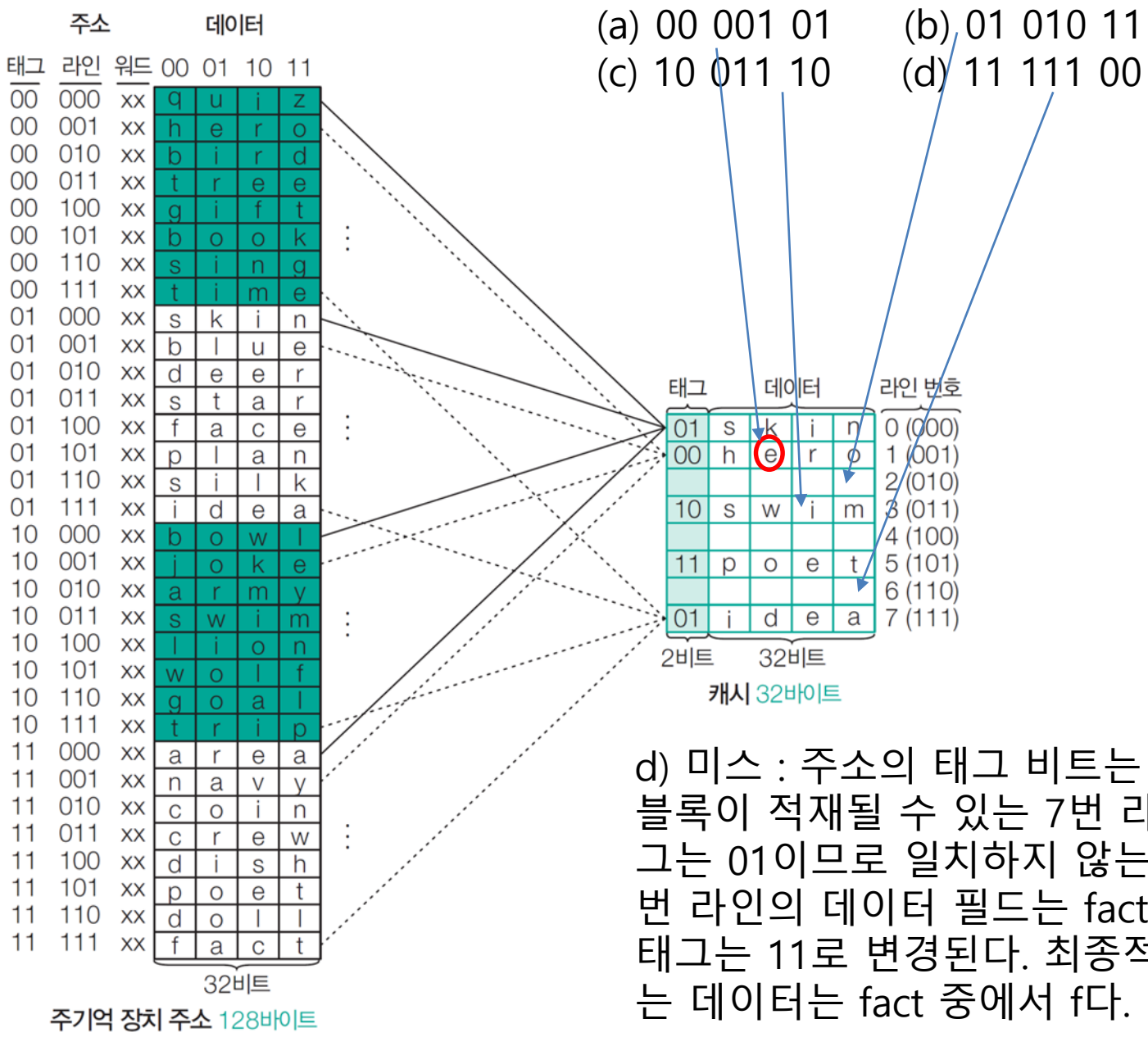
(a) **히트** : 이 블록은 1번 라인에 적재될 수 있는데, 현재 태그가 00이므로 일치한다. 인출되는 데이터는 hero 중에서 e다.

(b) **미스** : 이 블록이 적재될 수 있는 2번 라인이 비어 2번 라인의 데이터 필드에는 deer가 적재되고, 태그는 01로 세트된다. 최종적으로 인출되는 데이터는 deer 중에서 r이다.

(c) **히트** : 이 블록은 3번 라인에 적재될 수 있는데, 현재의 태그가 10이므로 일치한다. 인출되는 데이터는 swim 중에서 i다.

(d) **미스** : 주소의 태그 비트는 11이고, 이 블록이 적재될 수 있는 7번 라인의 현재 태그는 01이므로 일치하지 않는다. 따라서 7번 라인의 데이터 필드는 fact로 대체되고, 태그는 11로 변경된다. 최종적으로 인출되는 데이터는 fact 중에서 f다.

End of Example



d) 미스 : 주소의 태그 비트는 11이고, 이 블록이 적재될 수 있는 7번 라인의 현재 태그는 01이므로 일치하지 않는다. 따라서 7번 라인의 데이터 필드는 fact로 대체되고, 태그는 11로 변경된다. 최종적으로 인출되는 데이터는 fact 중에서 f다.

그림 6-26 직접 사상의 예

03 캐시 기억 장치

❖ 직접 사상 캐시의 장단점

장점	• 하드웨어가 간단하고, 구현 비용이 적게 든다.
단점	• 각 주기억 장치 블록이 적재될 수 있는 캐시 라인이 한 개뿐이기 때문에, 그 라인을 공유하는 다른 블록이 적재되는 경우에는 반복적으로 미스되어 히트율이 떨어짐(swap-out)

수고하셨습니다!