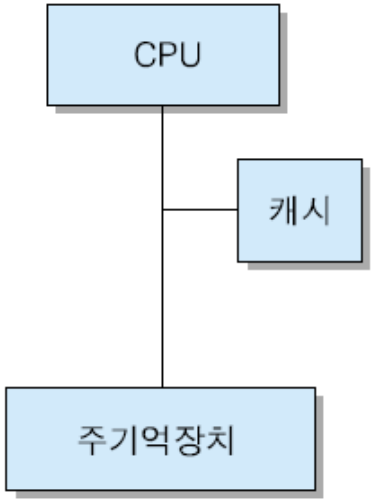
캐시 기억장치(Cache memory)

- 캐시용량
- 인출방식
- 사상방식
- 교체알고리즘
- 쓰기정책
- 다중캐시

이 자료는 김종현저-컴퓨터구조론(생능출판사)의 내용을 편집한 것입니다.

5.5 캐시 기억장치(Cache memory)

- 사용 목적 : CPU와 주기억장치의 속도 차이로 인한 CPU 대기 시간을 최소화 시키기 위하여 CPU와 주기억장치 사이에 설치하는 고속 반도체 기억장치
- 특징
  - 주기억장치보다 액세스 속도가 높은 칩 사용
  - 가격 및 제한된 공간 때문에 용량이 적다



캐시 기억장치

- 캐시 적중(cache hit) : CPU가 원하는 데이터가 캐시에 있는 상태
- 캐시 미스(cache miss) : CPU가 원하는 데이터가 캐시에 없는 상태. 이 경우에는 주기억장치로부터 데이터를 읽어온다.
- 적중률(hit ratio) : 캐시에 적중되는 정도(H)

$$H = \frac{\text{캐시에 적중되는 횟수}}{\text{전체기억장치 액세스 횟수}}$$

- 캐시의 미스율(miss ratio) = $(1 - H)$
- 평균 기억장치 액세스 시간(Ta) :

$$Ta = H \times Tc + (1 - H) \times Tm$$

단, Tc는 캐시 액세스 시간, Tm은 주기억장치 액세스 시간

- 캐시의 적중률이 높아질수록 평균 기억장치 액세스시간은 캐시 액세스 시간에 접근
- 캐시 적중률은 프로그램과 데이터의 지역성(locality)에 따라 달라짐

- 지역성(locality)
  - 시간적 지역성(temporal locality) : 최근에 액세스된 프로그램이나 데이터가 가까운 미래에 다시 액세스 될 가능성이 높다
  - 공간적 지역성(spatial locality) : 기억장치내에 인접하여 저장되어 있는 데이터들이 연속적으로 액세스 될 가능성이 높다
  - 순차적 지역성(sequential locality) : 분기(branch)가 발생하지 않는 한, 명령어들은 기억장치에 저장된 순서대로 인출되어 실행된다
- 캐시 설계 목표
  - 캐시 적중률의 극대화
  - 캐시 액세스 시간의 최소화
  - 캐시 미스에 따른 지연 시간의 최소화
  - 주기억장치와 캐시간의 데이터 일관성 유지 및 그에 따른 오버헤드의 최소화

캐시의 크기 / 인출 방식

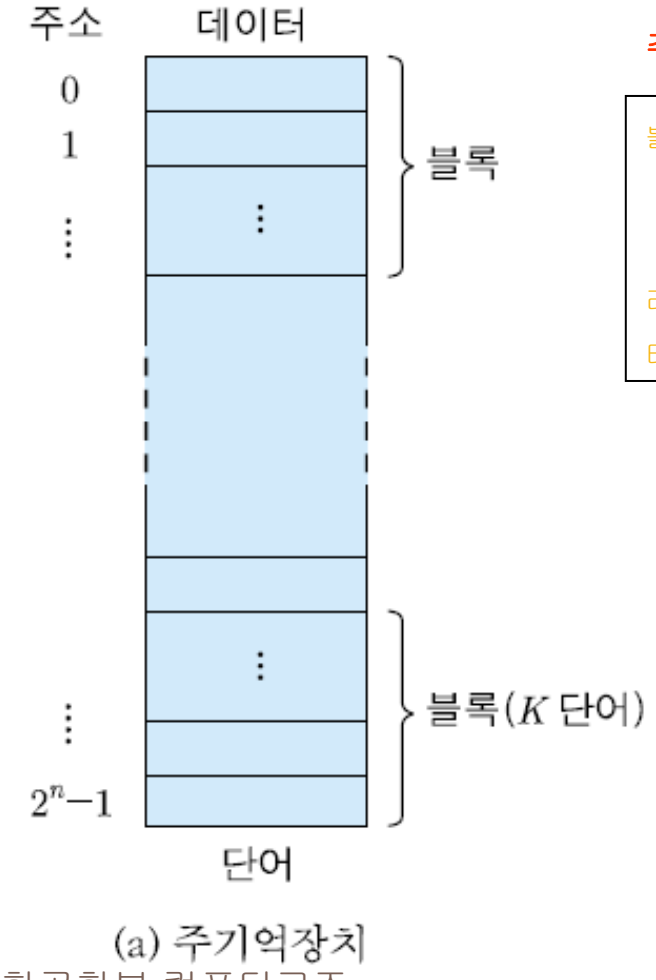
5.1 캐시의 크기(용량)

- 용량이 커질수록 적중률이 높아지지만, 비용이 증가
- 용량이 커질수록 주소 해독 및 정보 인출을 위한 주변 회로가 더 복잡해지기 때문에 액세스 시간이 다소 더 길어진다

5.2 인출 방식

- 요구 인출(demand fetch) 방식 : 필요한 정보만 인출해 오는 방법
- 선인출(prefetch) 방식
  - 필요한 정보 외에 앞으로 필요할 것으로 예측되는 정보도 미리 인출
  - 지역성이 높은 경우에 효과가 높다.

주기억장치와 캐시의 조직



(a) 주기억장치

주기억장치 블록 → 캐시의 라인(line)에 적재

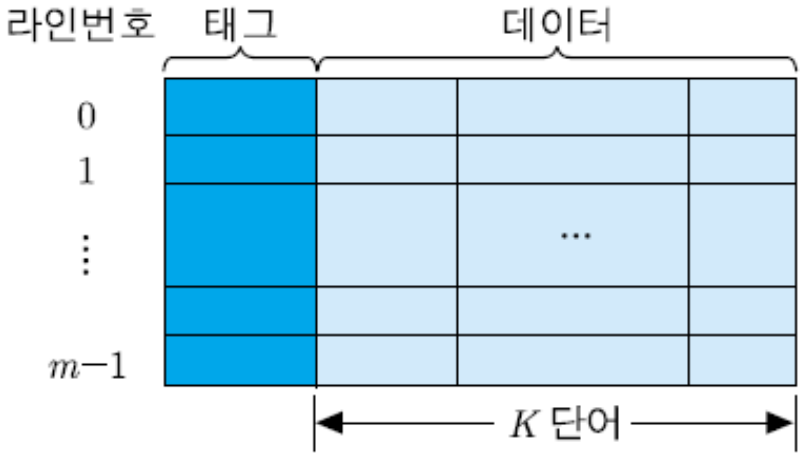
블록(block) : 주기억장치로부터 동시에 인출되는 정보들의 그룹

주기억장치 용량 = 2^n 단어, 블록 = K 단어

→ 블록의 수 = $2^n/K$ 개

라인(line) : 캐시에서 각 블록이 저장되는 장소

태그(tag) : 라인에 적재된 블록을 구분해주는 정보



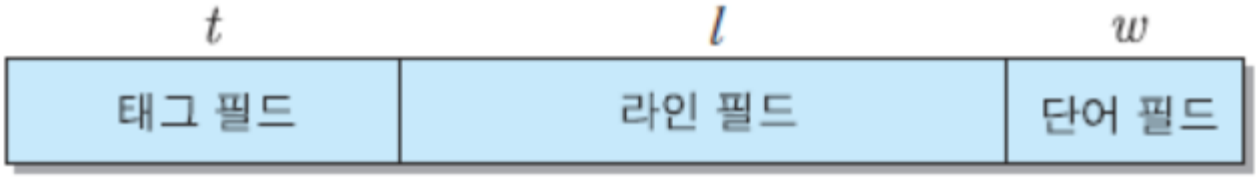
(b) 캐 시

5.5.3 사상 방식

- 어떤 주기억장치 블록들이 어느 캐시 라인을 공유할 것인 지를 결정해 주는 방법
  - 직접 사상(direct mapping)
  - 완전-연관 사상(fully-associative mapping)
  - 세트-연관 사상(set-associative mapping)

1) 직접 사상

- 주기억장치의 블록들이 지정된 하나의 캐시 라인으로만 적재됨
- 주기억장치 주소 형식



  - 태그 필드(t 비트) : 태그 번호
  - 라인 번호(l 비트) : 캐시의 $m = 2^l$ 개의 라인들 중의 하나를 지정
  - 단어 필드(w 비트) : 각 블록 내 2^w 개 단어들 중의 하나를 구분
- 주기억장치의 블록 j 가 적재될 수 있는 캐시 라인의 번호 i :

$$i = j \bmod m$$

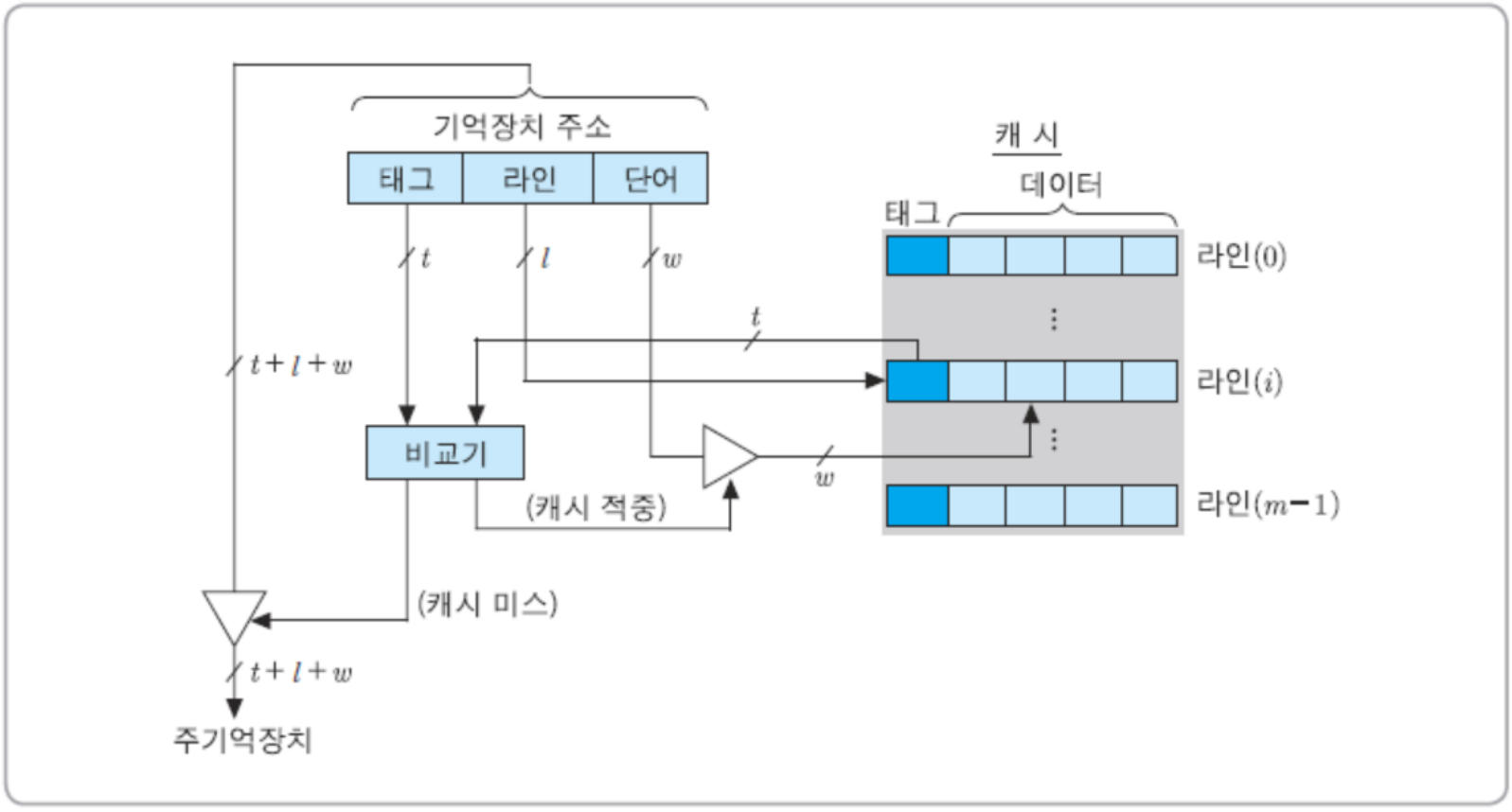
단, j : 주기억장치 블록 번호, m : 캐시 라인의 전체 수

라인을 공유하는 주기억장치 블록들

- 각 캐시 라인은 2^t 개의 블록들에 의하여 공유
- 같은 라인을 공유하는 블록들은 서로 다른 태그를 가짐
- 장점
  - 하드웨어가 간단하고, 구현 비용이 적게 든다
- 단점
  - 각 주기억장치 블록이 적재될 수 있는 캐시 라인이 한 개뿐이기 때문에, 그 라인을 공유하는 다른 볼록이 적재되는 경우에는 swap-out 됨

직접 사상 캐시의 동작 원리

- 캐시로 기억장치 주소가 보내지면, 그 중 l 비트의 라인 번호를 이용하여 캐시의 라인을 선택
- 선택된 라인의 태그 비트들을 읽어서 주소의 태그 비트들과 비교
  - 두 태그값이 일치하면 (캐시 적중) ➔ 주소의 w 비트들을 이용하여 라인내의 단어들 중에서 하나를 인출하여 CPU로 전송
  - 태그값이 일치하지 않는다면 (캐시 미스)
    - 주소를 주기억장치로 보내어 한 블록을 액세스
    - 인출된 블록을 지정된 캐시 라인에 적재하고, 주소의 태그 비트들을 그 라인의 태그 필드에 기록
    - 만약 그 라인에 다른 블록이 이미 적재되어 있다면, 그 내용은 지워지고 새로이 인출된 블록을 적재

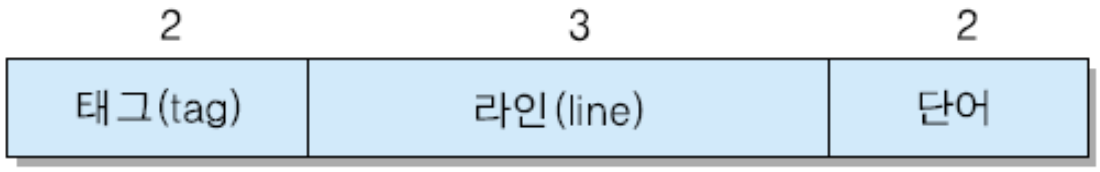

기억장치 주소
태그
라인
단어
t
l
w
t+l+w
비교기
(캐시 적중)
(캐시 미스)
t+l+w
주기억장치
캐 시
데이터
태그
라인(0)
라인(i)
라인(m−1)
t
w

직접 사상 캐시의 예

- 주기억장치 용량 = 128(2^7) 바이트
- 주기억장치 주소 = 7 비트 (바이트 단위 주소 지정)
- 블록 크기 = 4 바이트

  → 주기억장치는 128/4 = 32 개의 블록들로 구성
- 캐시 크기 = 32 바이트
- 캐시 라인 크기 = 4 바이트 (블록 크기와 동일)
- 전체 캐시 라인의 수, m = 32/4 = 8 개

➔기억장치 주소 형식



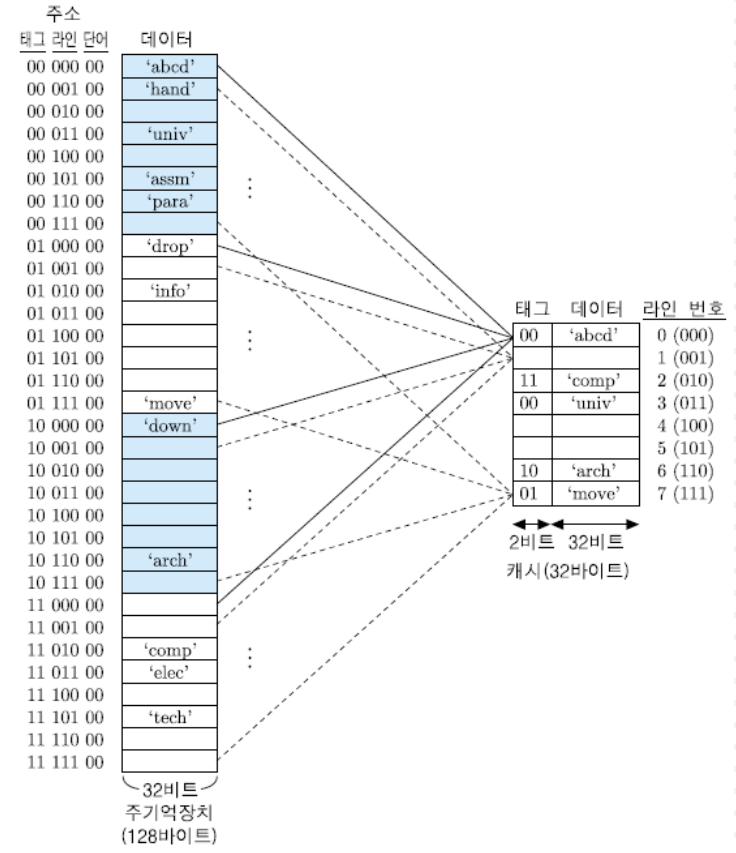
직접 사상 캐시의 예 (계속)

- 각 기억장치 블록이 공유하게 될 캐시 라인 번호

$$i = j \bmod 8$$

[표 5-5] 각 캐시 라인을 공유하는 주기억장치 블록들

캐시 라인	주기억장치 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



2) 완전-연관 사상

- 주기억장치 블록이 캐시의 어떤 라인으로든 적재 가능
- 태그 필드 = 주기억장치 블록 번호
- 기억장치 주소 형식

- 직접 사상 캐시의 예에 완전-연관 사상 방식을 적용하면,

[장점]

- 새로운 블록이 캐시로 적재될 때 라인의 선택이 매우 자유롭다
- 지역성이 높다면, 적중률이 매우 높아진다

[단점]

- 캐시 라인들의 태그들을 병렬로 검사하기 위하여 가격이 높은 연관 기억장치 (associative memory) 및 복잡한 주변 회로가 필요

완전-연관 사상 캐시의 장단점

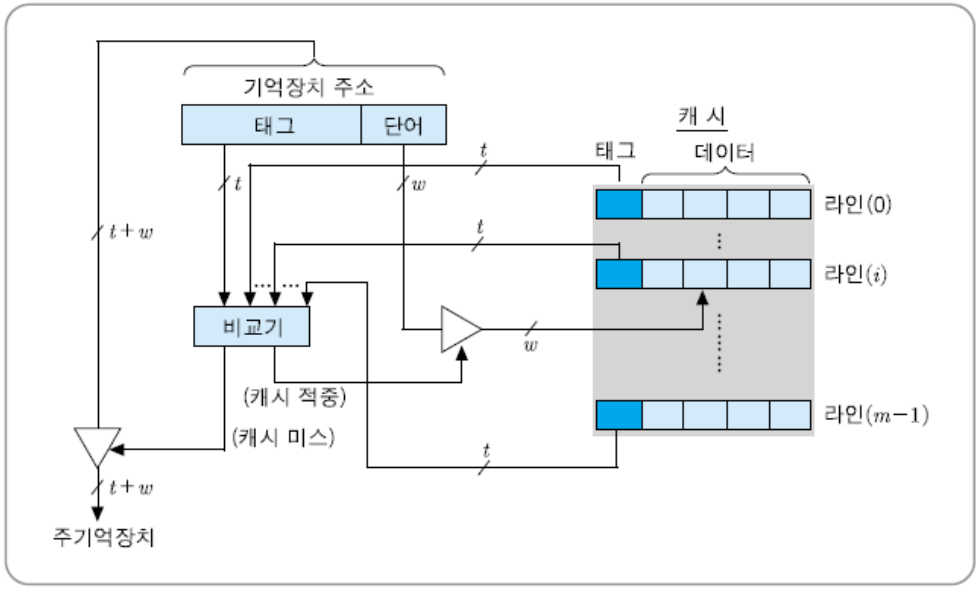
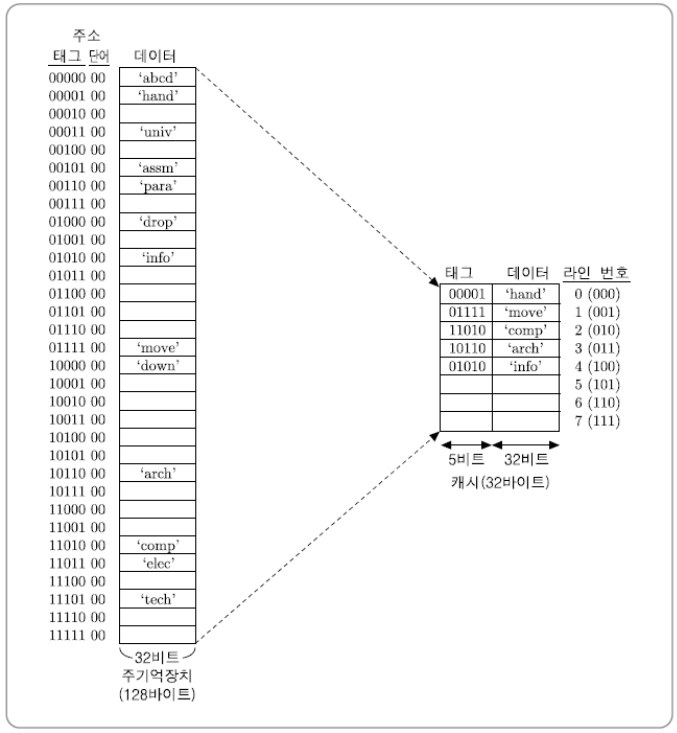


그림 5-20 완전-연관 사상 캐시의 조직



3) 세트-연관 사상

- 직접 사상과 완전-연관 사상의 조합
- 주기억장치 블록 그룹이 하나의 캐시 세트를 공유하며, 그 세트에는 두 개 이상의 라인들이 적재될 수 있음
- 캐시는 v 개의 세트(set)들로 나누어지며, 각 세트들은 k 개의 라인들로 구성 (*k-way* 세트-연관 사상 이라고 부름)
- 주기억장치 블록이 적재될 수 있는 캐시 세트의 번호 i :

$$i = j \bmod v$$

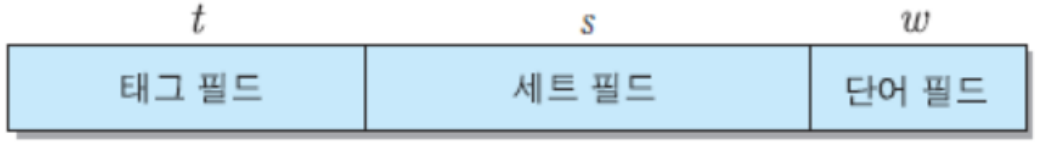
단, i : 캐시 세트의 번호

j : 주기억장치 블록 번호

v : 캐시 세트들의 수

세트-연관 사상 (계속)

- 기억장치 주소 형식
  - 태그 필드와 세트 필드를 합한 $(t+d)$비트가 주기억장치의 $2^{(t+d)}$블록들 중의 하나를 지정
  - 그 블록이 적재될 수 있는 세트 번호: 세트 필드에 의해 지정



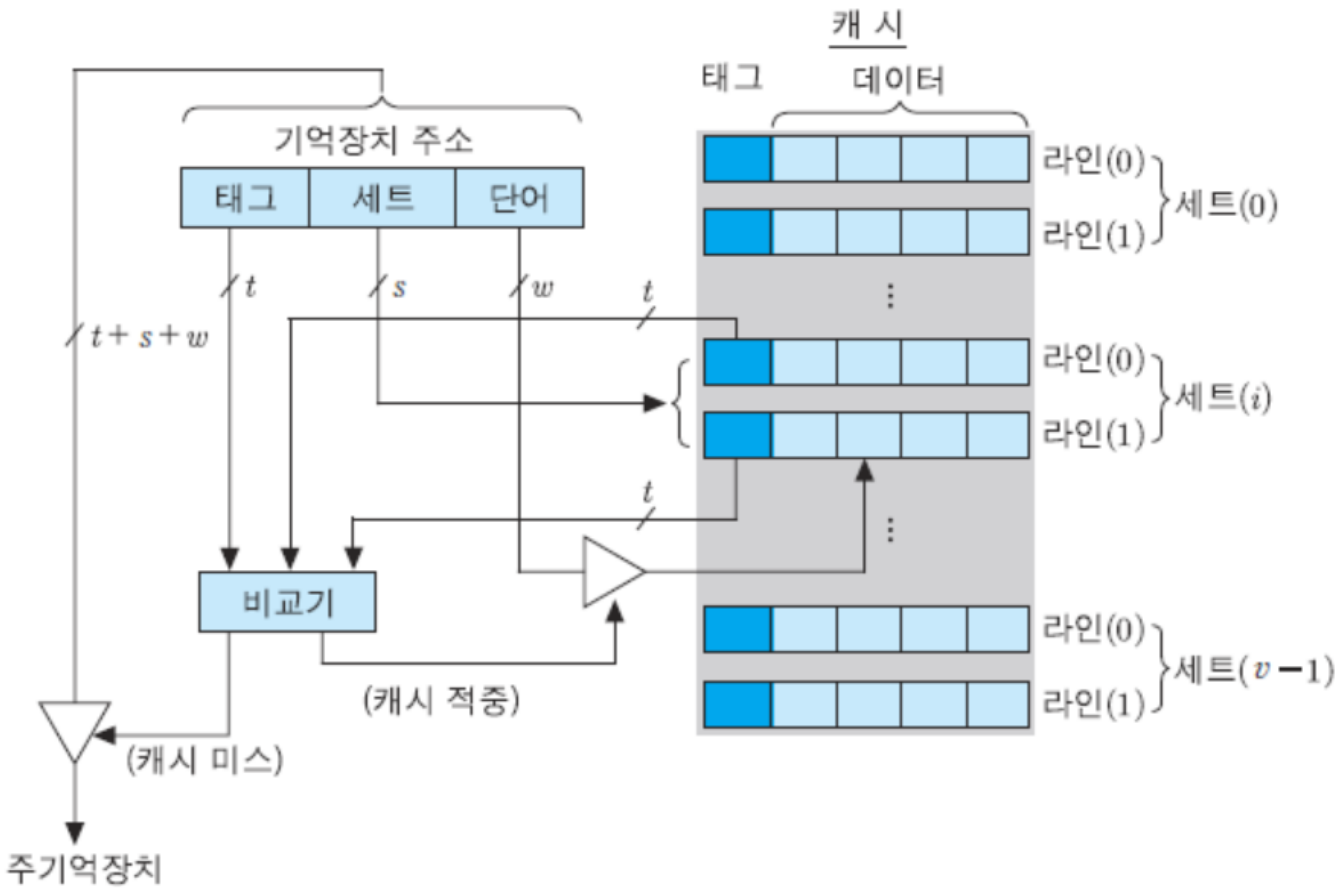
  - 앞의 [예]에 2-way 세트-연관 사상 방식을 적용하는 경우의 주소 형식: (세트 수 = 8/2 = 4개)

- 세트 수 = 캐시 라인 수 $(v = m)$ & 세트 내 라인의 수 k = 1 ➔ 직접 사상
- 세트 수 = 1 & 세트 내 라인의 수 = 캐시의 전체 라인 수 $(k = m)$ ➔ 완전-연관 사상

세트-연관 사상의 동작 원리

- 기억장치 주소의 세트 필드의 s 비트들을 이용하여 캐시 세트들 중의 하나를 선택
- 주소의 태그 필드 내용과 그 세트내의 태그들을 비교
  - 일치하는 것이 있으면 (캐시 적중)
    - ➔ 그 라인내의 한 단어를 w 비트에 의해 선택하여 인출
  - 일치하는 것이 없다면 (캐시 미스) ➔
    - 주기억장치를 액세스
    - 세트 내 라인들 중의 한 라인에 새로운 블록을 적재 (교체 알고리즘 필요)

2-*way* 세트-연관 사상 캐시의 조직



[큰 용량의 세트-연관 사상 캐시 조직의 예]

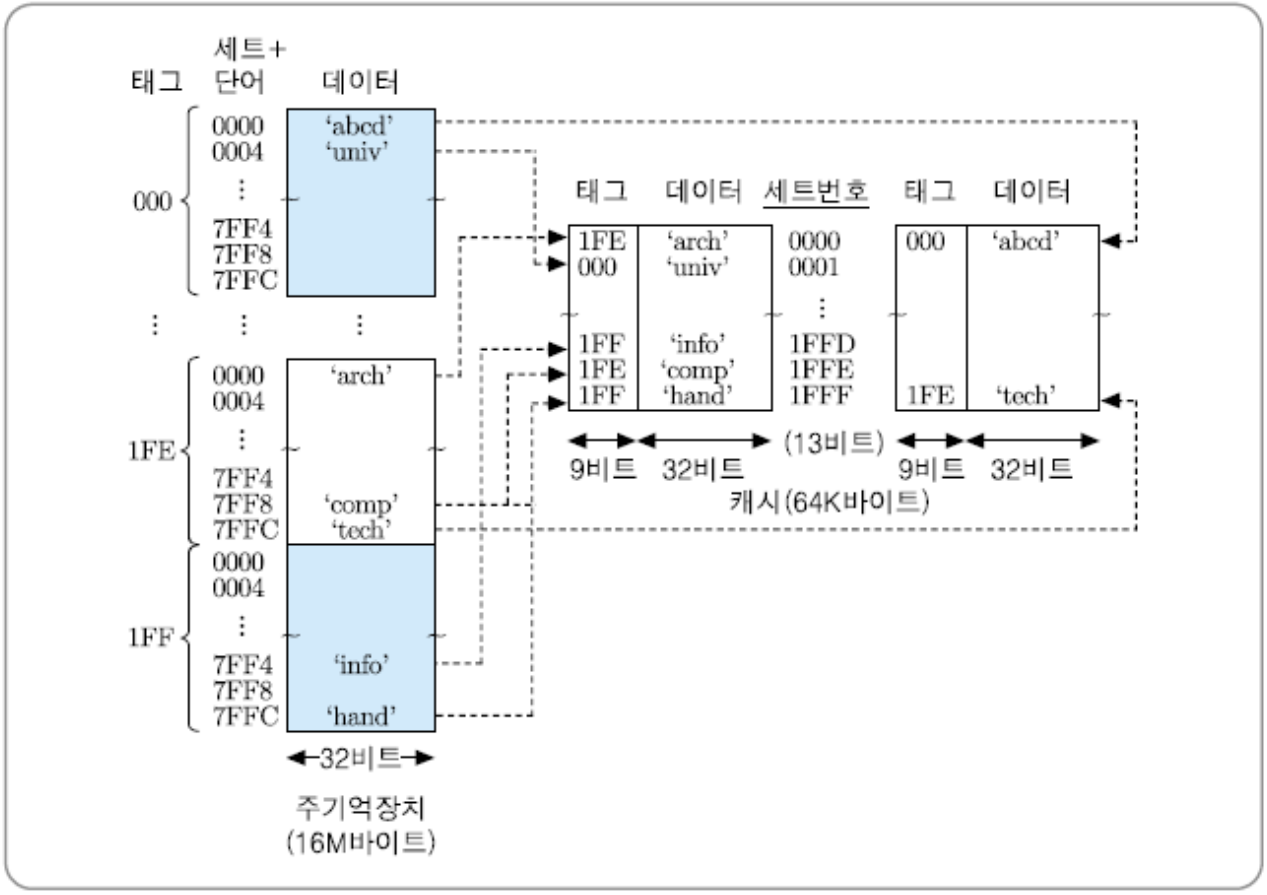
- 주기억장치의 용량은 16M (2^{24})바이트이다. 따라서 주기억장치의 주소는 24비트이고, 바이트 단위로 주소가 지정된다.
- 주기억장치는 4-바이트 크기의 블록들 4M (2^{22})개로 구성되어 있다.그리고 단어의 길이는 한 바이트이다.
- 캐시의 용량은 64K (2^{16})바이트이다.
- 주기억장치의 블록 크기가 4바이트이므로, 캐시 라인의 크기도 4바이트가 되며, 결과적으로 라인 수 m = 16K (2^{14})개가 된다.
- 2-way 세트-연관 사상 조직으로 가정한다. 따라서 세트의 수 v = 8K (2^{13}) 개 이다.

- 기억장치 주소 형식

- 각 세트

세트 번호	주기억장치 블록 번호
0000	000000, 008000, ···, FF8000
0001	000004, 008004, ···, FF8004
⋮	⋮ ⋮ ⋮
1FFF	007FFC, 00FFFC, ···, FFFFFC

64KByte 크기의 2-way 세트-연관 캐시의 예



5.5.4 교체 알고리즘

- 세트-연관 사상에서 주기억장치로부터 새로운 블록이 캐시로 적재될 때, 만약 세트 내 모든 라인들이 다른 블록들로 채워져 있다면, 그들 중의 하나를 선택하여 새로운 블록으로 교체

- 교체 알고리즘 : 캐시 적중률을 극대화할 수 있도록 교체할 블록을 선택하기 위한 알고리즘
  - 최소 최근 사용(Least Recently Used: LRU) 알고리즘 : 사용되지 않은 채로 가장 오래 있었던 블록을 교체하는 방식
  - FIFO(First-In-First-Out: FIFO) 알고리즘 : 캐시에 적재된 지 가장 오래된 블록을 교체하는 방식
  - 최소 사용 빈도(Least Frequently Used: LFU) 알고리즘 : 참조되었던 횟수가 가장 적은 블록을 교체하는 방식

5.5.5 쓰기 정책 (write policy)

- 캐시의 블록이 변경되었을 때 그 내용을 주기억장치에 갱신하는 시기와 방법의 결정
- 종류

① *Write-through* : 모든 쓰기 동작들이 캐시로 뿐만 아니라 주기억장치로도 동시에 수행되는 방식

[장점] 캐시에 적재된 블록의 내용과 주기억장치에 있는 그 블록의 내용이 항상 같다

[단점] 모든 쓰기 동작이 주기억장치 쓰기를 포함하므로, 쓰기 시간이 길어진다

쓰기 정책의 종류 (계속)

② *Write-back* : 캐시에서 데이터가 변경되어도 주기억장치에는 갱신되지 않는 방식

[장점] 기억장치에 대한 쓰기 동작의 횟수가 최소화되고, 쓰기 시간이 짧아진다

[단점] 캐시의 내용과 주기억장치의 해당 내용이 서로 다르다

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

쓰기 정책 (계속)

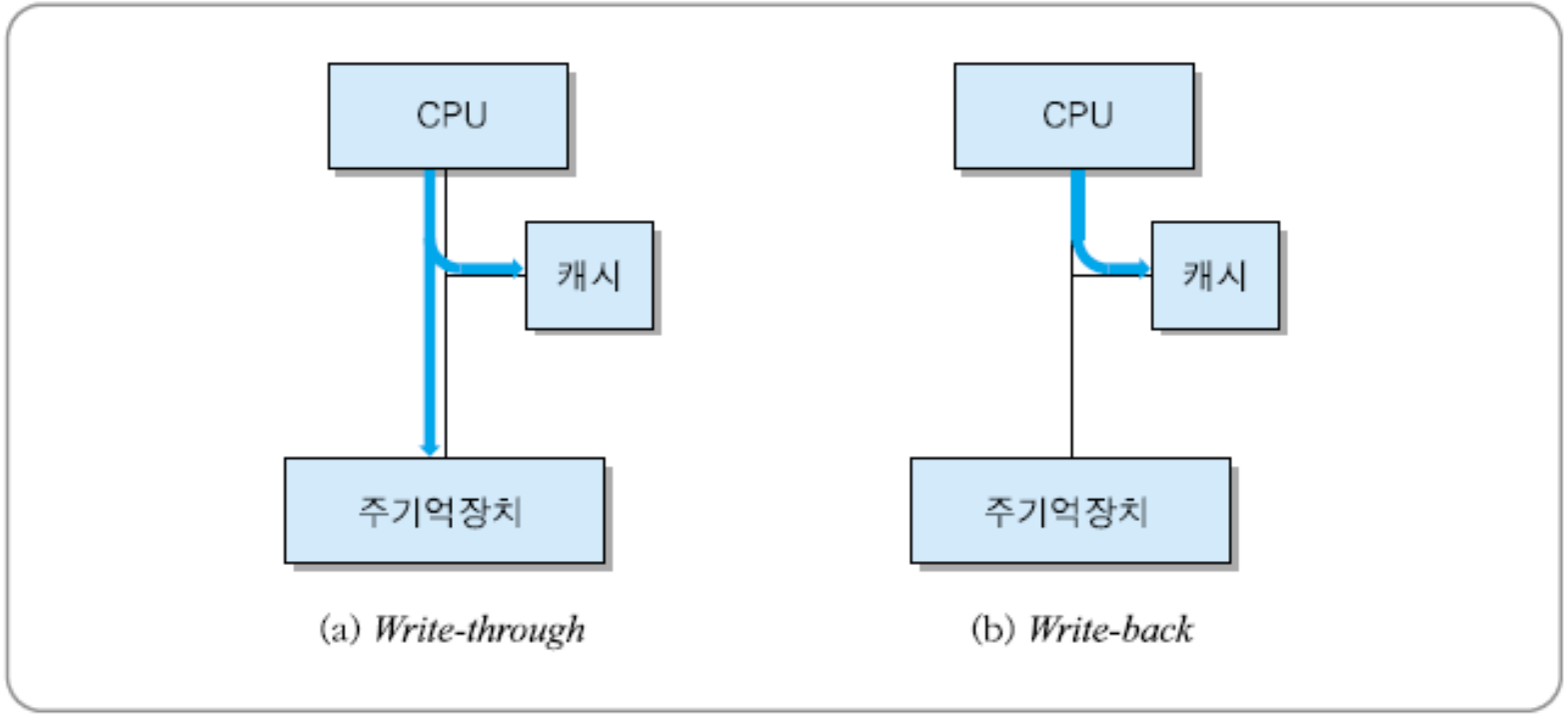
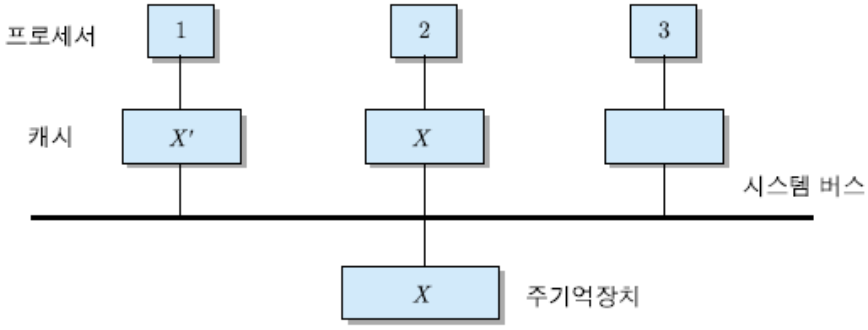


그림 5-26 쓰기 정책에 따른 쓰기 동작의 비교

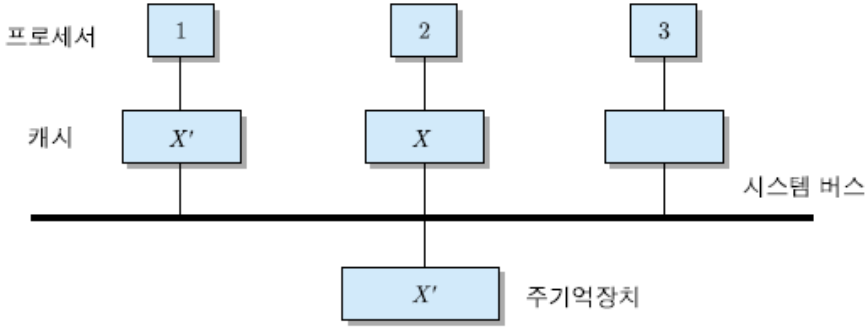
다중프로세서시스템에서의 데이터 불일치 문제

- 다중프로세서시스템에서의 데이터 불일치 문제(data inconsistency problem) : 주기억장치에 있는 블록의 내용과 캐시 라인에 적재된 블록의 내용이 서로 달라지는 문제
- 캐시 일관성 프로토콜(cache coherence protocol) 필요
  [예] MESI 프로토콜 (세부적 사항은 이 책에서는 다루지 않음)

다중프로세서시스템에서의 데이터 불일치 [예]



(a) *Write-back* 방식이 사용된 경우

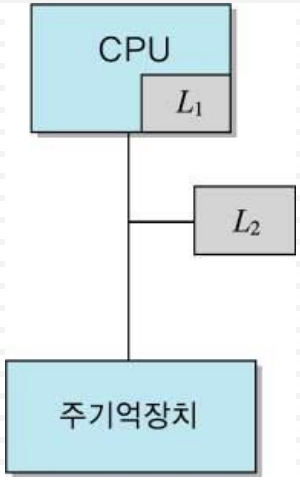


(b) *Write-through* 방식이 사용된 경우

5.5.6 다중 캐시(multiple cache)

- 온-칩 캐시(on-chip cache) : 캐시 액세스 시간을 단축시키기 위하여 CPU 칩 내에 포함시킨 캐시 (아래 그림의 L1)

1) 계층적 캐시(hierarchical cache)
   - 온-칩 캐시를 1차(L1) 캐시로 사용하고, 칩 외부에 더 큰 용량의 2차(L2) 캐시를 설치하는 방식



계층적 캐시 (계속)

- L_2는 L_1의 슈퍼-세트(super-set): L_2 의 용량이 L_1보다 크며, L_1 의 모든 내용이 L_2에도 존재
- 먼저 L_1을 검사하고, 만약 원하는 정보가 L_1에 없다면 L_2를 검사하며, L_2에도 없는 경우에는 주기억장치를 액세스
- L_1은 속도가 빠르지만, 용량이 작기 때문에 L_2 보다 적중률은 더 낮다
- 2-단계 캐시 시스템의 평균 기억장치 액세스 시간 :

$$Ta = H_1 \times T_{L1} + (H_2 - H_1) \times T_{L2} + (1 - H_2) \times Tm$$

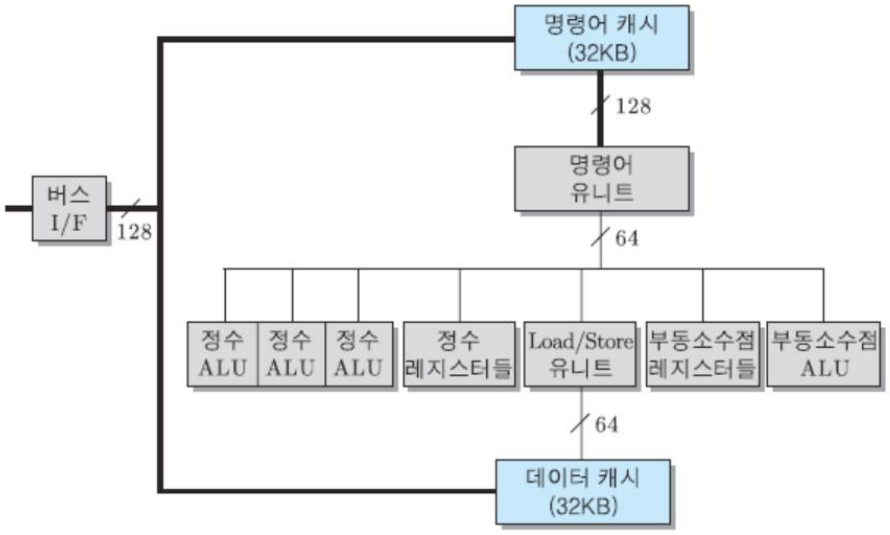
만약 H_2가 L_1에서 미스된 액세스들에 대한 L_2의 적중률이라면,

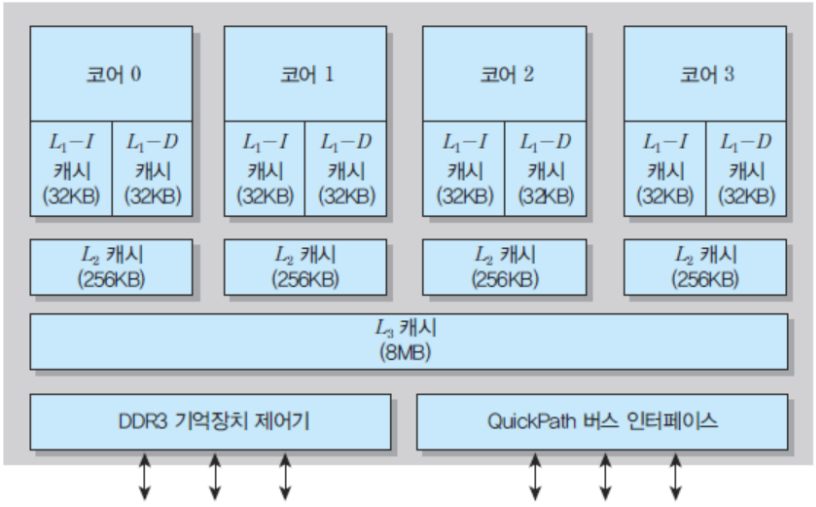
$$Ta = H_1 \times T_{L1} + (1- H_1) H_2 \times T_{L2} + \{1- H_1-(1 - H_1)H_2\} \times Tm$$

2) 분리 캐시 (split cache)

- 캐시를 명령어 캐시와 데이터 캐시로 분리
- 명령어 인출 유니트와 실행 유니트 간의 캐시 액세스 충돌 제거
- 대부분의 고속 프로세서들에서 사용

분리 캐시의 사례


명령어 캐시
(32KB)
128
명령어
유니트
64
버스
I/F
128
정수
ALU
정수
ALU
정수
ALU
정수
레지스터들
Load/Store
유니트
부동소수점
레지스터들
부동소수점
ALU
64
데이터 캐시
(32KB)


코어 0
코어 1
코어 2
코어 3
L_1-I 캐시 (32KB)
L_1-D 캐시 (32KB)
L_2 캐시 (256KB)
L_3 캐시 (8MB)
DDR3 기억장치 제어기
QuickPath 버스 인터페이스