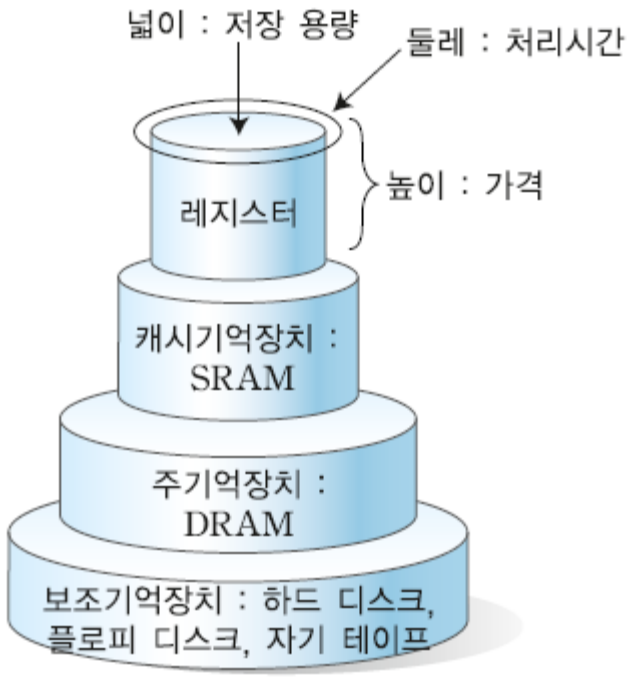
8장 캐시기억장치

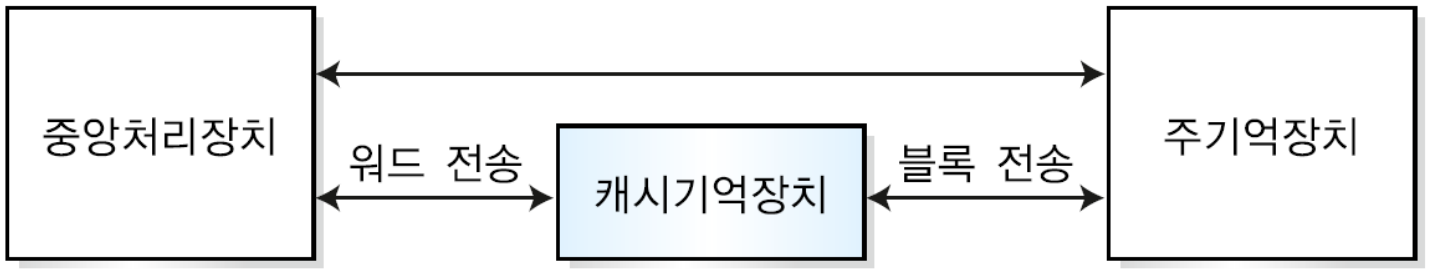
컴퓨터의 기억장치 구성

- 기억장치의 계층적 분류
- 캐시(Cache)기억장치
  - 주기억장치에 비해 5~10배 정도 접근속도가 빠르다.
  - 자주 사용되는 명령들을 저장하고 있다가 중앙처리장치에 빠른 속도로 제공한다.

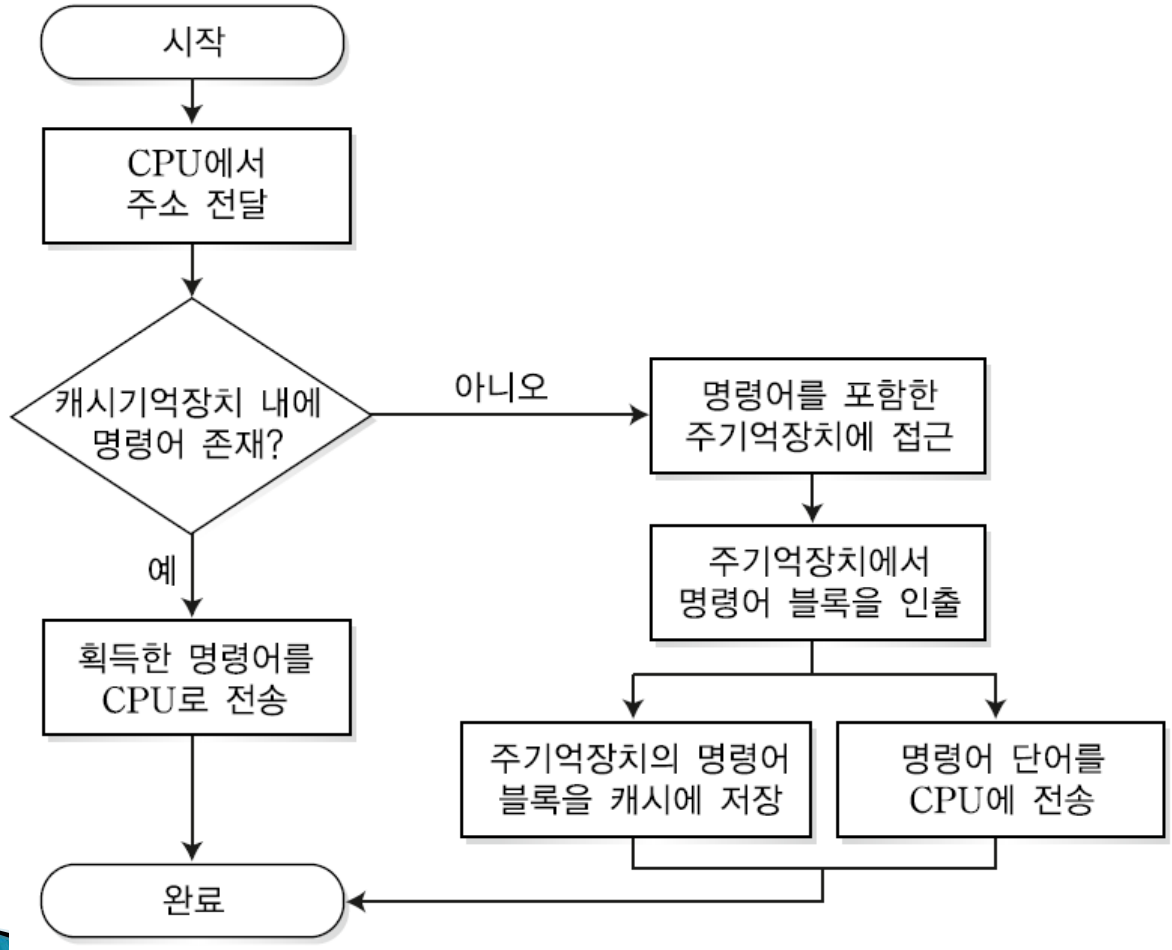


캐시기억장치의 원리

- 참조의 지역성 (locality of reference)
  - 공간적 지역성(spatial locality)
  - 시간적 지역성(temporal locality)
  - 순차적 지역성(Sequential Locality)



캐시기억장치의 동작


시작
CPU에서
주소 전달
캐시기억장치 내에
명령어 존재?
아니오
예
명령어를 포함한
주기억장치에 접근
주기억장치에서
명령어 블록을 인출
획득한 명령어를
CPU로 전송
주기억장치의 명령어
블록을 캐시에 저장
명령어 단어를
CPU에 전송
완료

기억장치 평균 접근시간

- 적중률(Hit Ratio)
  - 캐시기억장치를 가진 컴퓨터의 성능을 나타내는 척도

$$\text{적중률} = \frac{\text{적중 수}}{\text{전체 메모리 참조 횟수}}$$

- 소요되는 평균 기억장치 접근시간 $T_{average}$

$$T_{average} = H_{hit\_ratio} \times T_{cache} + (1 - H_{hit\_ratio}) \times T_{main}$$

$T_{average}$ = 평균 기억장치 접근 시간　　T_{main} = 주기억장치 접근 시간

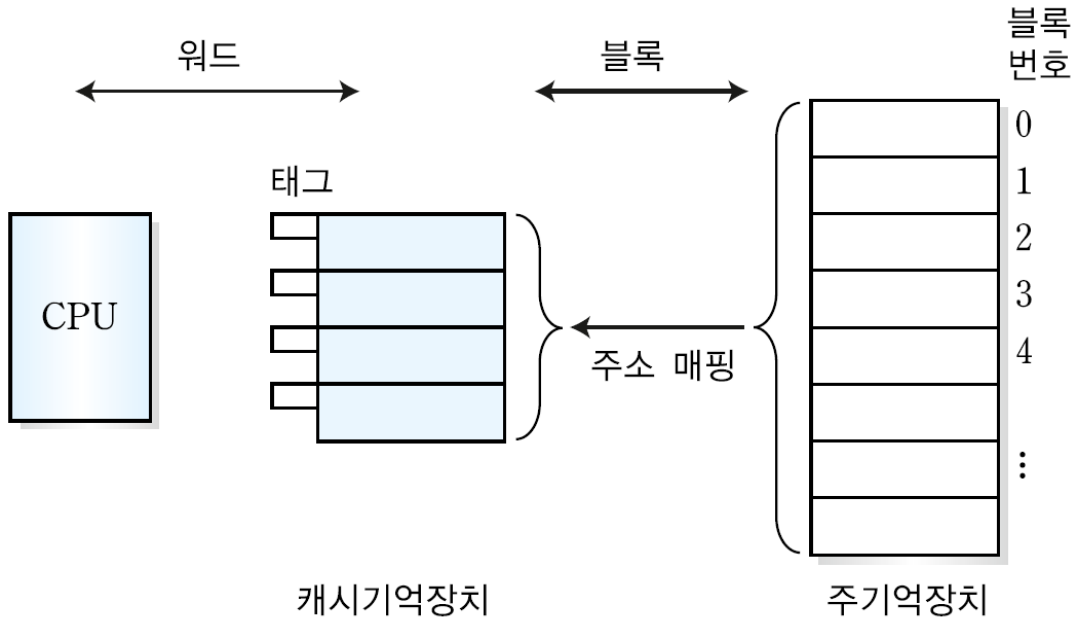
T_{cache} = 캐시기억장치 접근 시간　　H_{hit_ratio} = 적중률

평균 기억장치 접근시간 $T_{average}$ 예

- T_{cache} = 50ns, T_{main} = 400ns일 때, 적중률을 증가시키면서 기억장치 접근시간을 계산하면
  - 적중률 70%: $T_{average}$ = 0.7 x 50ns + 0.3 x 400ns = 155ns
  - 적중률 80%: $T_{average}$ = 0.8 x 50ns + 0.2 x 400ns = 120ns
  - 적중률 90%: $T_{average}$ = 0.9 x 50ns + 0.1 x 400ns = 85ns
  - 적중률 95%: $T_{average}$ = 0.95 x 50ns + 0.05 x 400ns = 67.5ns
  - 적중률 99%: $T_{average}$ = 0.99 x 50ns + 0.01 x 400ns = 53.5ns

캐시기억장치의 설계

- 주기억장치와 캐시기억장치 간의 정보 공유



캐시기억장치 설계 시 고려할 요소

- 캐시기억장치의 크기(Size)
- 인출방식(fetch algorithm)
- 사상함수(Mapping function)
- 교체 알고리즘(Replacement algorithm)
- 쓰기 정책(Write policy)
- 블록 크기(Block size)
- 캐시기억장치의 수(Number of caches)

주메모리와 캐시메모리의 사상 (mapping)

- 어떤 주메모리 블록들이 어떤 캐시 슬롯을 공유할 것인가 결정해주는 방법
- 세가지 사상 방식
  - 직접 사상(direct mapping)
  - 연관 사상(associative mapping)
  - 집합 연관 사상(set-associative mapping)

직접 사상(Direct mapping)

- 주기억장치의 블록 j가 적재될 수 있는 캐시 슬롯의 번호 i는 모듈로(modulo) 연산에 의해서 결정. m은 캐시 슬롯의 전체 수

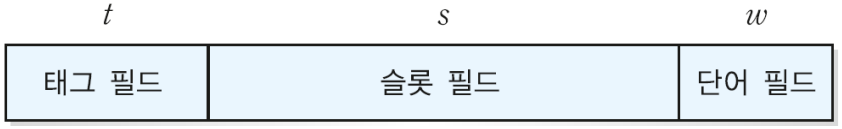
$$i = j \bmod m$$

- 캐시슬롯에 올수 있는 블록 번호

캐시 슬롯	주기억장치 블록 번호
0	$0, m, \cdots, 2^{t+s}-m$
1	$1, m+1, \cdots, 2^{t+s}-m+1$
...	...
m−1	$m-1, 2m-1, \cdots, 2^{t+s}-1$

직접 사상(Direct mapping)

- 같은 슬롯을 공유하는 주기억장치 블록들은 서로 다른 태그를 가짐
- 직접 사상에서 주기억장치의 주소 형식



- CPU가 요구하는 주기억장치의 주소 중 캐시의 해당 슬롯을 확인, 태그 필드와 비교하여 동일하면 적중
- 동일 슬롯 번호를 가지고 있지만 태그가 다른 두 개 이상의 단어가 반복하여 참조되면 적중률이 상당히 떨어지는 단점이 발생함
- 예제) 초기에는 캐시가 비어있고 CPU가 00001-00010-10001-00010 번지를 차례로 참조하는 경우 ※ 블록의 크기를 1단어로 가정하자.

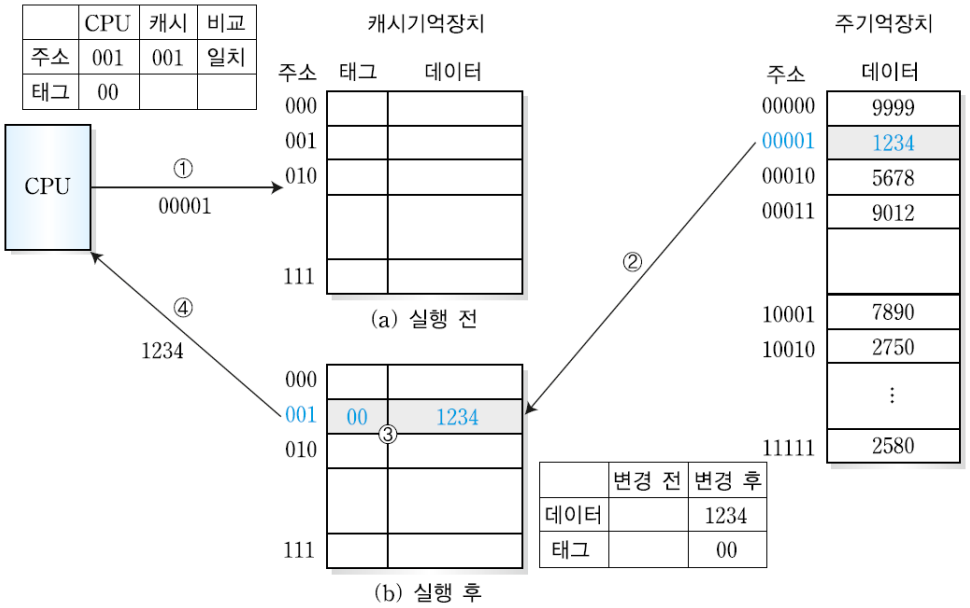
수행 절차 1-CPU가 00001번지 단어를 참조

① CPU는 캐시의 슬롯 번호 001에 가서 2비트의 태그 정보 00 를 비교 ➔해당 번지가 비어 있으므로 실패(miss)

② 캐시 실패를 확인하고 주기억장치에서 00001번지를 참조하여서 단어를 획득

③ 캐시기억장치의 해당 슬롯에 획득한 데이터 1234와 태그 00을 저장

④ 단어 1234를 CPU에 전달



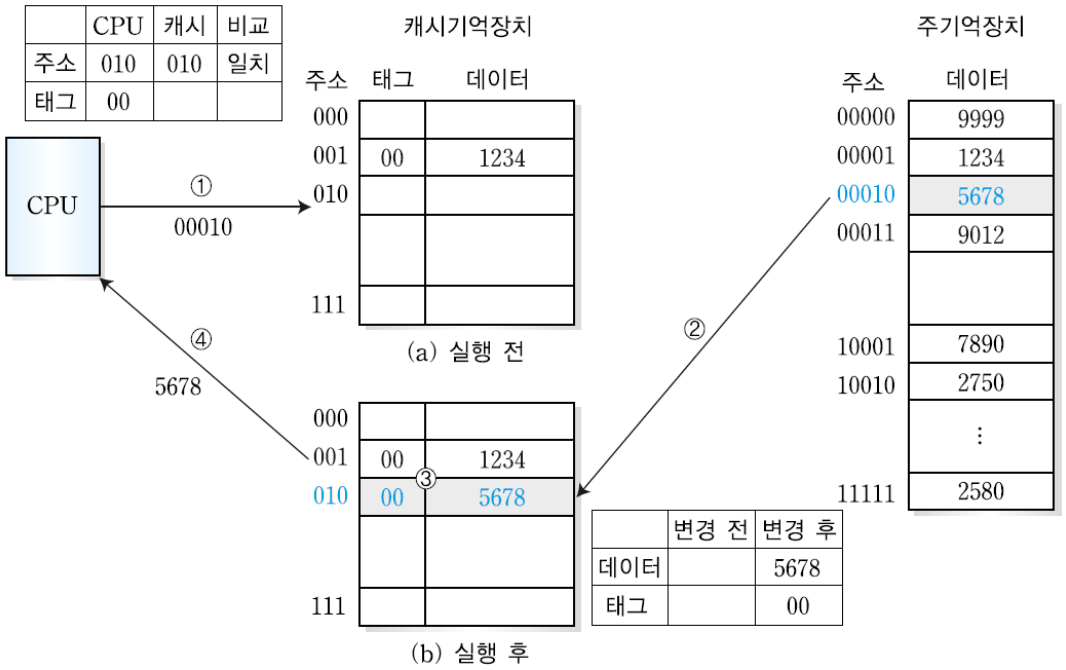
수행 절차 2 –CPU가 00010번지 단어를 참조

① 캐시기억장치의 010번지 주소에 접근, 해당 번지가 비어 있으므로 실패(miss)한 것으로 판단하고 주기억장치를 참조

② 주기억장치의 주소 00010에서 필요한 단어를 획득

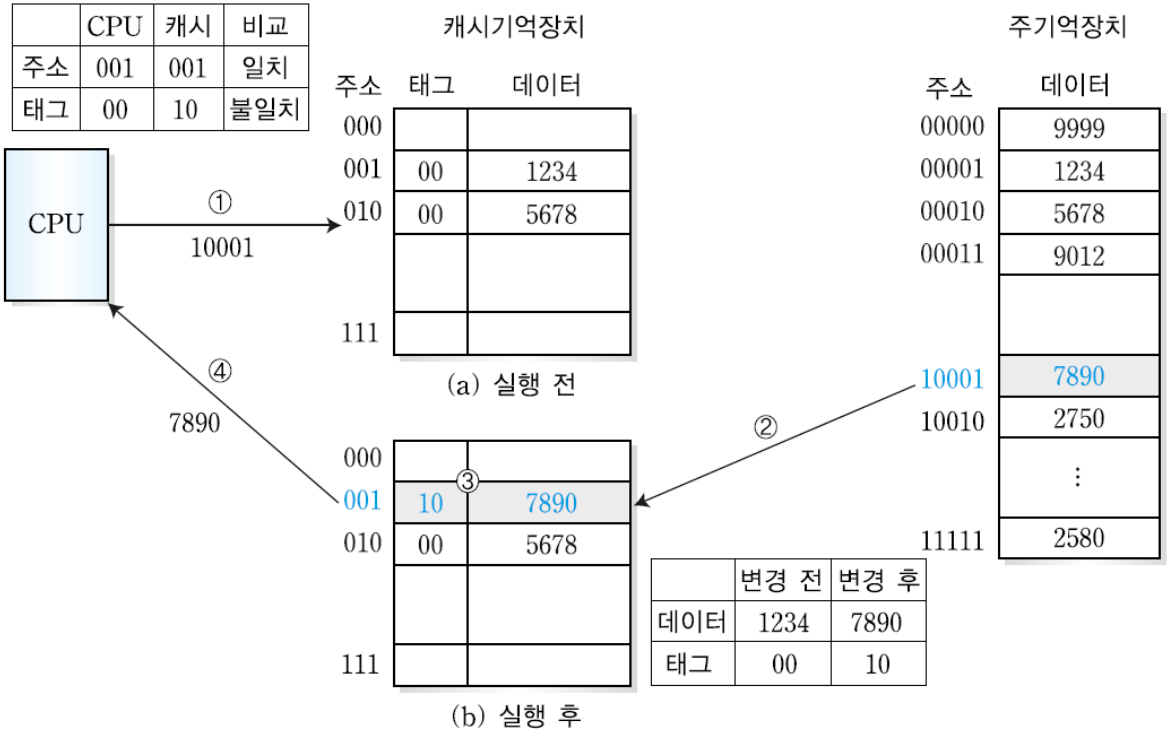
③ 캐시기억장치의 해당 주소 010에 단어 데이터 5678과 태그 00을 저장

④ 단어 데이터 5678를 CPU로 전달



수행 절차 3 –CPU가 10001번지 단어를 참조

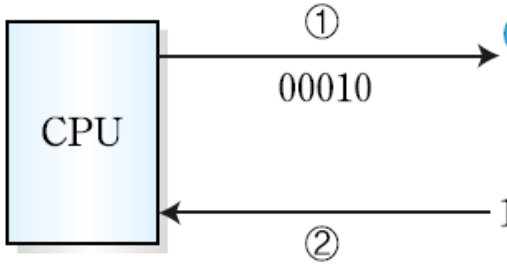
① 캐시기억장치의 주소 캐시기억장치의 001번지에 접근, 태그가 00으로 불일치하므로 실패
② 주기억장치 10001번지에서 단어를 획득
③ 캐시기억장치의 001번지에 단어 데이터 7890과 태그 10을 저장.
④ 태그를 포함해서 00 1234가 10 7890으로 변경됨
⑤ 단어 데이터 7890를 CPU로 전달



수행 절차 4 –CPU가 00010번지 단어를 참조

① 캐시기억장치의 주소 010번지에 접근. 태그가 00으로 일치하므로 적중(hit)
② 캐시기억장치 010번지에서 중앙처리장치가 요구하는 단어를 획득함

	CPU	캐시	비교
주소	010	010	일치
태그	00	00	일치



캐시기억장치

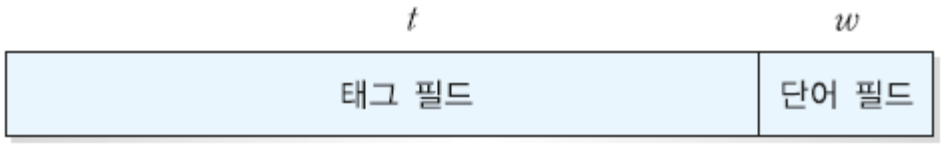
주소	태그	데이터
000		
001	00	7890
010	00	5678
111		

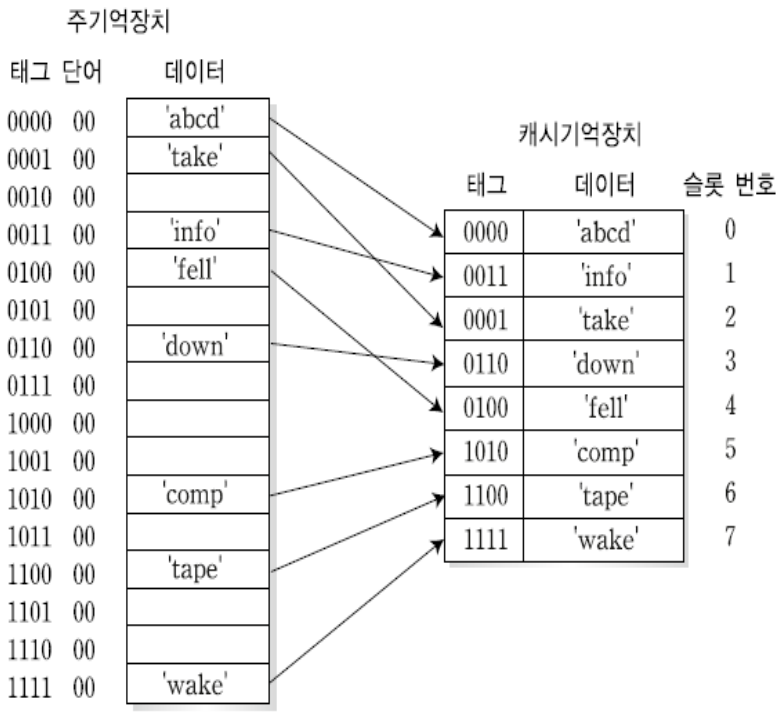
주기억장치

주소	데이터
00000	9999
00001	1234
00010	5678
00011	9012
10001	7890
10010	2750
	⋮
11111	2580

연관 사상(Associative mapping)

- 주기억장치의 블록이 캐시의 임의의 슬롯에 적재할 수 있다.
- 융통성이 있으나 모든 슬롯에 대해 태그가 일치하는지 검사해야 하므로 검사 시간이 오래 걸린다.
- 주기억장치의 주소 형식



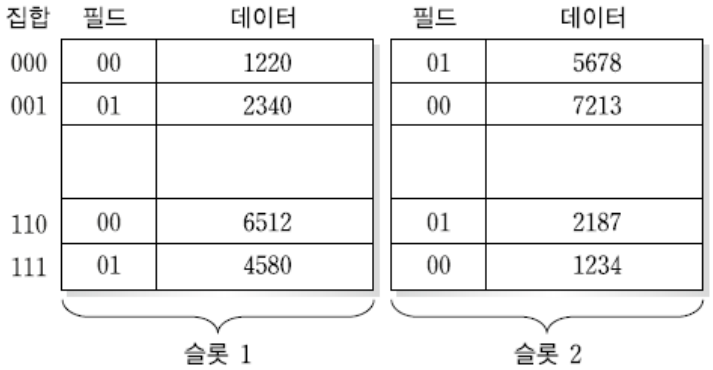


연관사상의 동작

- CPU가 주기억장치에 저장된 단어들을 인출하려면 우선, 캐시기억장치에 대하여 태그를 비교하여 일일이 검색을 수행
- 캐시 적중(hit)이 발생하면 인출
- 모든 지역을 검색하여도 원하는 데이터가 존재하지 않을 때는 캐시 실패(miss)
- 이 경우 주기억장치에서 원하는 데이터 인출 ➔ 캐시 교체 알고리즘 필요
- 빠른 검색을 위해 associative-mapped memory 사용

집합 연관 사상(Set-associative mapping)

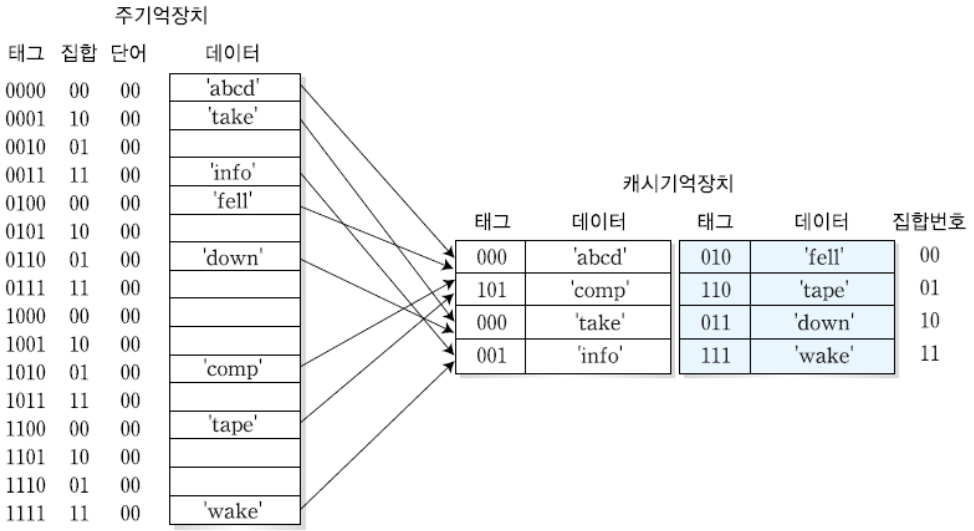
- 직접 사상과 연관 사상 방식을 조합한 방식
- 두 개의 집합을 갖는 집합 연관 캐시기억장치의 구조



- 같은 슬롯에 여러 개의 블록이 상주하는 것이 가능하여 직접 사상 방식의 단점을 보완 (위 예에서는 2개의 블록)
- 같은 집합 내에서는 연관사상 방식을 적용

집합 연관 캐시 기억장치의 동작

- 주기억장치의 용량이 128바이트, 블록의 크기가 4바이트, 캐시의 크기는 32바이트, 캐시 슬롯의 크기가 약 8바이트일 때 집합 연관 사상 방법의 예



교체 알고리즘(Replacement Algorithms)

- 연관 사상 및 집합 연관 사상 방식의 경우 교체 알고리즘이 필요함.
  - 교체 알고리즘- 캐시기억장치의 어느 슬롯 데이터를 제거하는가를 결정하는 방식
  - 직접 사상 방식에서는 교체 알고리즘을 사용할 필요 없음.
- 교체 알고리즘에는 대표적인 종류
  - LRU : 최소 최근 사용 알고리즘
  - LFU : 최소 사용 빈도 알고리즘
  - FIFO : 선입력 선출력 알고리즘
  - RANDOM : 랜덤

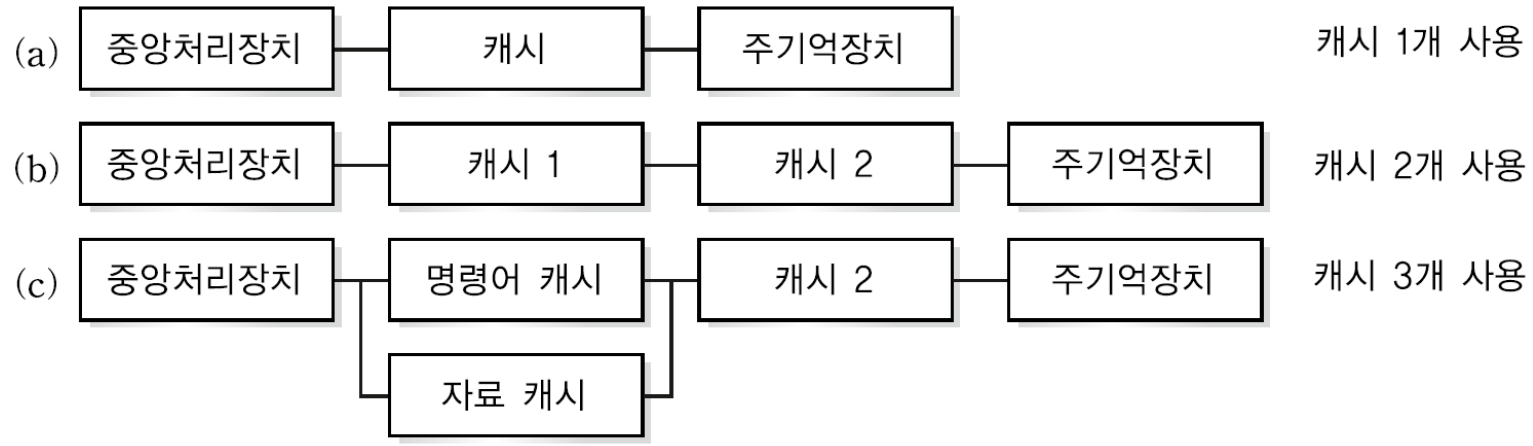
교체 알고리즘의 종류

- 최소 최근 사용(LRU, Least Recently Used) 알고리즘
  - 현재까지 알려진 교체 알고리즘 중에서 가장 효과적인 교체 알고리즘
  - CPU로의 인출이 없는 가장 오래 저장되어 있던 블록을 교체하는 방식
- 최소 사용 빈도(LFU, Least Frequently Used) 알고리즘
  - 적재된 블록들 중에서 인출 횟수가 가장 적은 블록을 교체하는 방식
- 선입력 선출력(FIFO, First In First Out) 알고리즘
  - 가장 먼저 적재된 블록을 우선적으로 캐시기억장치에서 삭제하는 교체 방식
  - 구현이 용이, 시간적으로 오래된 블록을 교체하여 효율성을 보장하지 못한다.
- 랜덤(Random)
  - 캐시기억장치에서 임의의 블록을 선택하여 삭제하고 새로운 블록으로 교체하는 방식.
  - 효율성을 보장하기가 어렵다.

쓰기 정책(Write Policy)

- CPU가 메모리에 쓰기를 실행할 때, 주메모리를 갱신하는 절차에는 다음과 같은 2가지 정책이 있다.
  - 즉시 쓰기(Write-though) 방식
    - CPU의 연산 결과가 기억장치에 저장하는 쓰기 동작은 캐시기억장치뿐만 아니라 주기억장치에서도 동시에 발생
    - 데이터의 일관성을 쉽게 보장
    - 매번 쓰기 동작이 발생할 때마다 캐시기억장치와 주기억장치간 접근➔ 쓰기 시간이 길어지게 된다.
  - 나중 쓰기(Write-back) 방식
    - 새롭게 생성된 중앙처리장치의 데이터를 캐시기억장치에만 기록하고 주기억장치는 나중에 기록하는 방식
    - 새로운 블록에 의해서 캐시기억장치에서 삭제되는 교체가 이뤄지기 전에 주기억장치로 복사됨
    - 주기억장치에 기록하는 동작을 최소할 수 있다.

다양한 캐시기억장치의 구조



온-칩(On-chip) 캐시기억장치

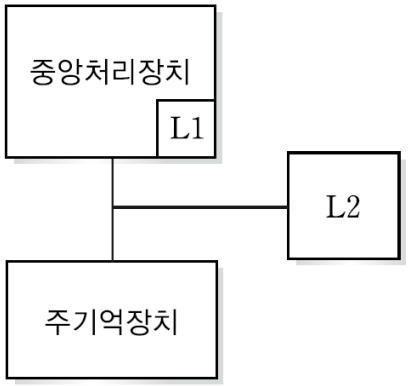
- 집적회로(Integrate Circuit)의 기술 발달로 캐시기억장치를 CPU의 내부에 포함시킨 것

◦블록 크기(Block size)

- 블록이 커지면?
  - 한꺼번에 많은 정보를 읽어 올 수 있다.
  - 블록 인출시간이 길어진다.
  - 불필요한 자료도 함께 적재된다.

단일 프로세서에서 캐시기억장치의 구조

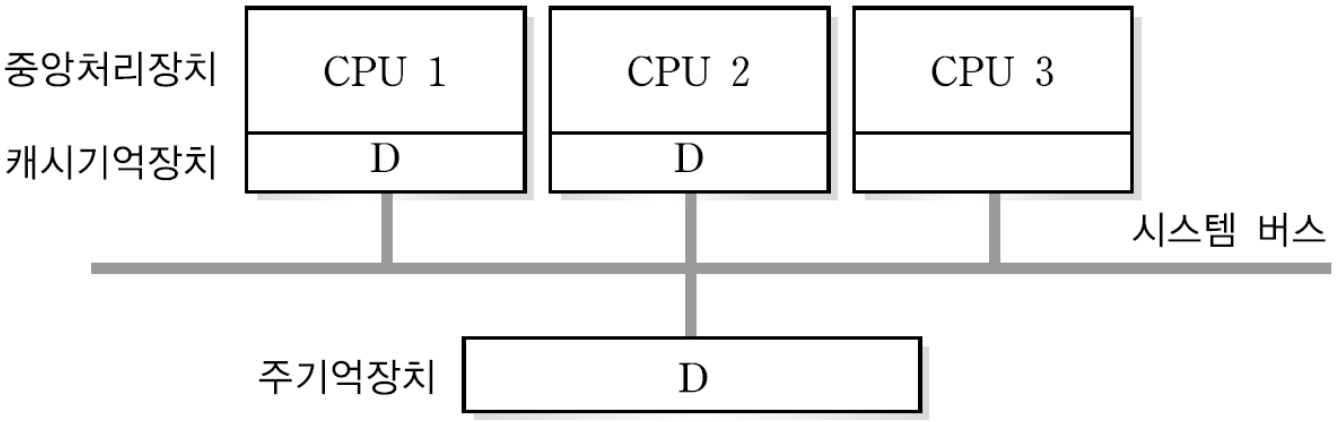
- 계층적 캐시(Hierarchical Cache)기억장치
  - 온-칩 캐시를 1차 캐시(L1)로 사용하고 칩 외부에 더 큰 용량의 오프-칩 캐시를 2차 캐시(L2)로 설치하는 방식
- 계층적 캐시의 구조에서 평균 기억장치 접근시간



$$T_{average} = H_{L1} \times T_{L1} + (H_{L2} - H_{L1}) \times T_{L2} + (1 - H_{L2}) \times T_{main}$$

- $T_{average}$ = 평균 기억장치 접근시간
- T_{main} = 주기억장치 접근시간
- T_{L1} = L1 캐시기억장치 접근시간
- T_{L2} = L2 캐시기억장치 접근시간
- H_{L1} = L1 캐시 적중률
- H_{L2} = L2 캐시 적중률

멀티 프로세서의 캐시기억장치 구조



- 즉시 쓰기방식에서의 데이터 불일치 상태
- 나중 쓰기 방식에서의 데이터 불일치 상태
- 캐시기억장치의 데이터 일관성 유지 방법
  - 공유 캐시기억장치를 사용하는 방법
  - 공유 변수는 캐시기억장치에 저장하지 않는 방법
  - 버스 감시 시스템을 사용하는 방법