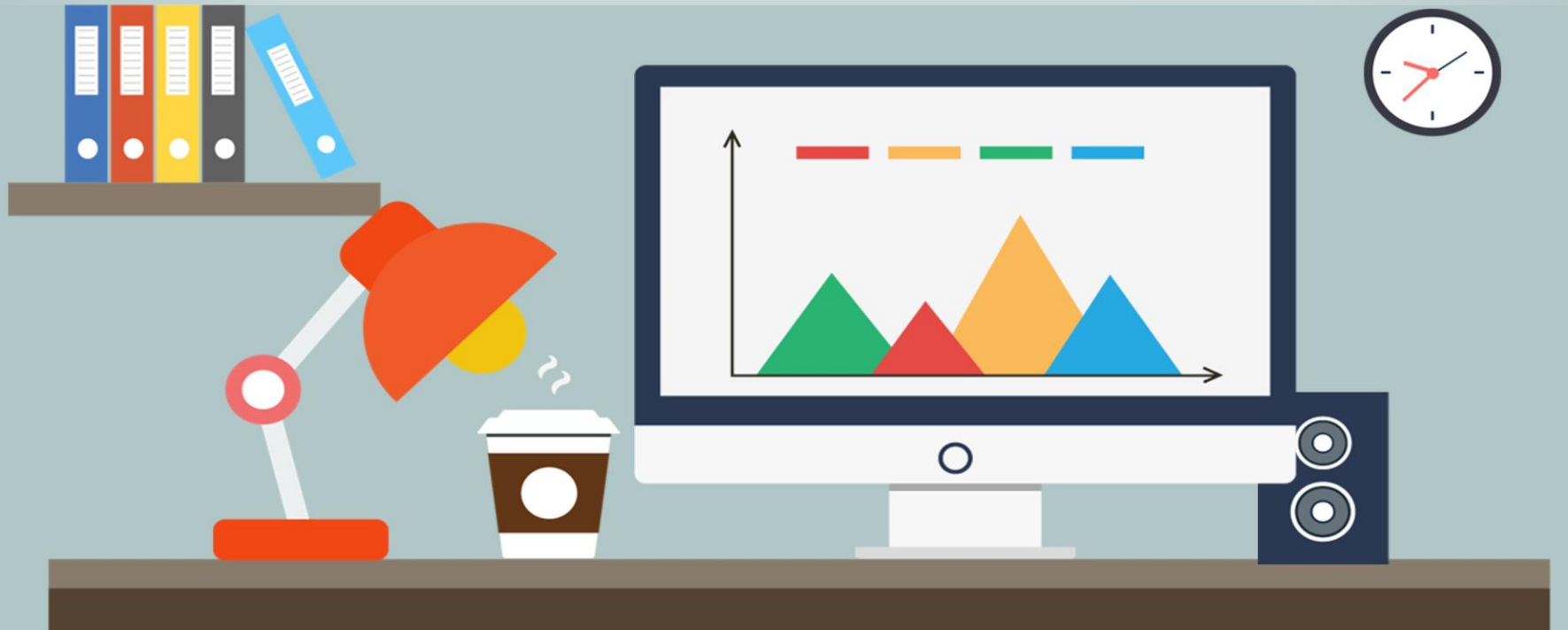
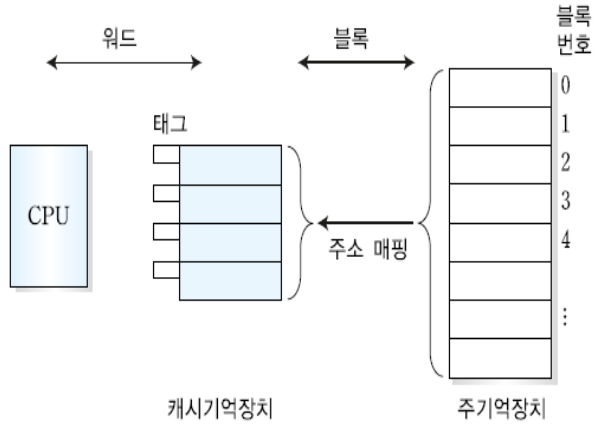
7주 2강

캐시기억장치의 설계



주기억장치와 캐시기억장치 간의 정보 공유



- 주기억장치에 저장된 데이터의 일부가 블록단위로 캐시기억장치에 복사
- 중앙처리장치가 적중된 데이터들을 워드 단위로 캐시기억장치에서 읽어온다.

캐시기억장치 설계시 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)

캐시기억장치의 크기

- 캐시기억장치의 용량이 주기억장치의 많은 블록을 복사해 저장할 수 있으므로 적중률이 높아진다.

- 용량이 클수록 주소 해독 및 정보 인출을 위한 주변 회로가 더 복잡해지기 때문에 접근시간이 더 길어진다.
  - 용량이 증가한 만큼의 속도 향상을 가져오지는 못한다.

- 캐시기억장치는 주기억장치보다는 비용이 고가이므로 용량의 증가는 비용의 증가로 이어진다.

- 캐시기억장치의 용량과 주변회로 간의 복잡도 관계를 최적화하여 적중률을 향상시키고 접근시간에 대한 저하를 막는 용량을 결정

- 캐시기억장치의 크기와 비용 간의 조정을 통해 적절한 용량과 비용을 결정해야 한다.
  - 연구 결과에 의하면 1k ~ 128k 단어(word)가 최적

인출방식

1 주기억장치에서 캐시기억장치로 명령이나 데이터 블록을 인출해 오는 방식에 따라서 캐시기억장치의 적중률의 변화가 크다.

2 요구인출(Demand Fetch) 방식

- CPU가 현재 필요한 정보만을 주기억장치에서 블록 단위로 인출해 오는 방식.
- 매번 주기억장치에서 인출을 수행하므로 실패율이 높아져서 캐시기억장치의 효과를 얻지 못할 때도 있다.

3 선인출(Prefetch) 방식

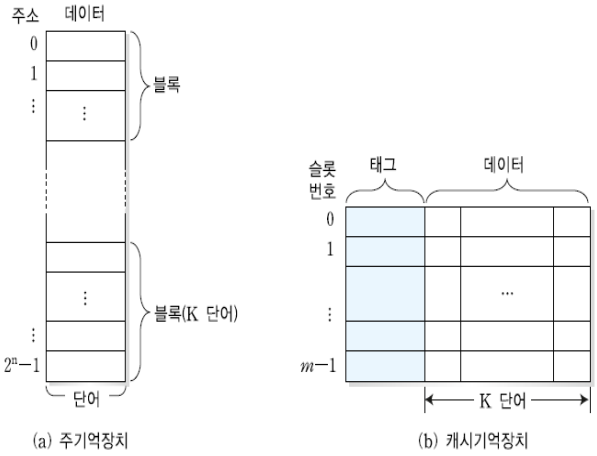
- CPU가 현재 필요한 정보 외에도 앞으로 필요할 것으로 예측되는 정보를 미리 인출하여 캐시기억장치에 저장하는 방식
- 주기억장치에서 명령이나 데이터를 인출할 때 필요한 정보와 이웃한 위치에 있는 정보들을 함께 인출하여 캐시에 적재하는 방식으로 지역성(locality)이 높은 경우에 효과가 높다.

블록과 슬롯

블록과 블록수

- 주기억장치에서 하나의 번지에 저장되는 데이터의 단위는 단어
- 단어가 모여서 하나의 블록(Block)이 되며, 캐시기억장치로 인출되는 정보의 그룹이다.
- 주기억장치의 용량이 2^n 개의 단어라고 하고 블록이 K개의 단어로 구성되면, 블록 개수는

$$\text{블록의 수} = 2^n/K \text{ 개}$$

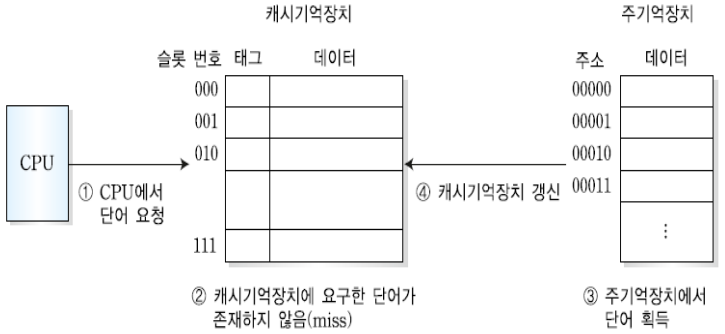


캐시기억장치의 구조

- 슬롯(Slot)
  - 캐시에서 한 블록이 저장되는 장소
  - 블록은 캐시의 각 슬롯에 저장되는 데이터의 길이가 된다.
- 태그(tag)
  - 슬롯에 적재된 블록을 구분해주는 정보

사상의 개념

- 주기억장치와 캐시기억장치 사이에서 정보를 옮기는 것을 사상(mapping)
  - 캐시에서 인출이 실패하면 캐시의 일부분은 주기억장치로 옮기고 주기억장치에서 필요한 정보를 캐시에 기억시키는 정보 교환



- 캐시기억장치의 사상방법에는 다음과 같이 대표적인 세 가지 방법이 존재
  - 직접 사상(direct mapping)
  - 연관 사상(associative mapping)
  - 집합 연관 사상(set-associative mapping)

교체 알고리즘(Replacement Algorithms)

- 캐시기억장치의 모든 슬롯이 데이터로 채워져 있는 상태에서 실패일 때

- 캐시의 특정 슬롯에서 데이터를 제거하고 주기억장치에서 새로운 데이터 블록을 가져와야 한다. 캐시의 어느 슬롯 데이터를 제거하는가를 결정하는 방식이 교체 알고리즘이다.

- 직접 사상 방식에서는 주기억장치의 데이터가 캐시기억장치의 동일 슬롯에 저장되기 때문에 교체 알고리즘을 사용할 필요가 없다.

- 연관 사상 및 집합 연관 사상 방식의 경우 교체 알고리즘이 필요하다.

교체 알고리즘의 분류

1 최소 최근 사용(LRU, Least Recently Used) 알고리즘

- 가장 효과적인 교체 알고리즘
- CPU로의 인출이 없는 가장 오래 저장되어 있던 블록을 교체하는 방식이다.

2 최소 사용 빈도(LFU, Least Frequently Used) 알고리즘

- 적재된 블록들 중에서 인출 횟수가 가장 적은 블록을 교체하는 방식이다.
- LRU 알고리즘이 시간적으로 오랫동안 사용되지 않은 블록을 교체,
  LFU알고리즘은 사용된 횟수가 적은 블록을 교체하는 방식이다

3 선입력 선출력 (FIFO, First In First Out) 알고리즘

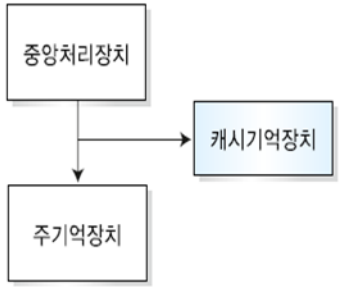
- 가장 먼저 적재된 블록을 우선적으로 캐시기억장치에서 삭제하는 교체 방식
- 구현이 용이, 시간적으로 오래된 블록을 교체하여 효율성을 보장하지 못한다.

4 랜덤(Random)

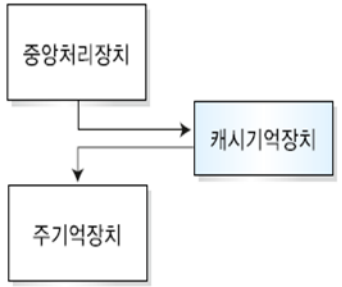
- 캐시기억장치에서 임의의 블록을 선택하여 삭제하고 새로운 블록으로
  교체하는 방식이다. 그러나 효율성을 보장하기가 어렵다

쓰기 정책(Write Policy)

- CPU가 연산 결과를 캐시에 기록하는 경우
  - 캐시기억장치와 주기억장치에 저장된 데이터가 상이하게 존재하므로 주기억장치의 데이터를 갱신하는 절차가 필요



(a) 즉시 쓰기



(b) 나중 쓰기

1 즉시 쓰기(Write-though) 방식

- 기억장치 쓰기 동작이 캐시기억장치뿐만 아니라 주기억장치에서도 동시에 발생, 데이터의 일관성을 쉽게 보장할 수 있다.
- 매번 쓰기 동작이 발생할 때마다 캐시기억장치와 주기억장치간 접근이 빈번하게 일어나고 쓰기 시간이 길어지게 된다.

2 나중 쓰기(Write-back) 방식

- 저장할 데이터를 캐시기억장치에만 기록하고 주기억장치는 나중에 기록하는 방식이다.
- 캐시기억장치에서 내용이 삭제되는 블록교체가 이뤄지기 전에 주기억장치로 복사된다.
- 두 기억장치의 데이터가 서로 일치하지 않아 주기억장치가 무효상태로 존재. 이 경우에는 주기억장치의 접근은 금지되고 캐시기억장치에만 접근
- 즉시 쓰기 방식과는 달리 주기억장치에 기록하는 동작을 최소할 수 있다.

블록크기와 캐시의 수

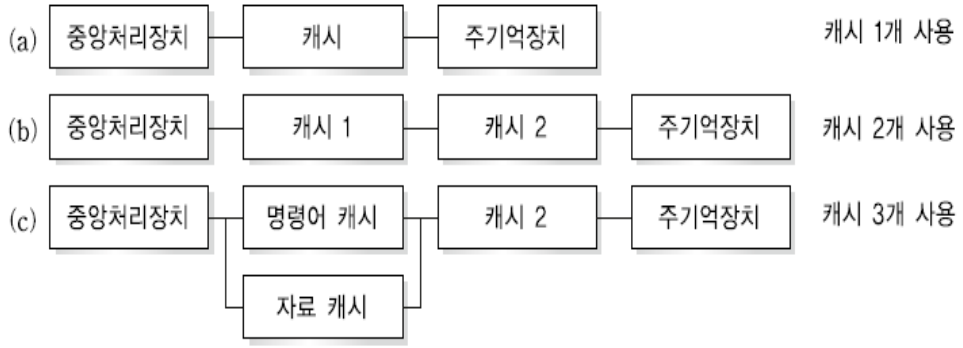
1 블록 크기(Block size)

- 블록이 커지면 한꺼번에 많은 정보를 읽어 올 수 있지만 인출시간이 길어지게 된다.
- 블록이 클수록 캐시에 적재할 수 있는 블록의 수가 감소하기 때문에 블록들이 더 빈번히 교체
- 블록이 커질수록 멀리 떨어진 단어들도 같이 읽혀오기 때문에 가까운 미래에 사용될 가능성이 낮다.
- 일반적인 블록의 크기는 4 ~ 8단어가 적당하다.

2 캐시의 수(Number of Caches)

- 일반적인 시스템은 오직 하나의 캐시기억장치를 가지고 있었다.
- 최근에는 캐시기억장치들이 계층적 구조로 설치되거나 기능별로 분리된 다수의 캐시기억장치를 사용하는 것이 보편화 되었다.
- 캐시기억장치를 설계할 때에는 몇 계층으로 할 것인지를 결정하여야 하며, 통합 형태와 분리 형태 중에서 어떤 형태로 구성할 것인지를 결정해야 한다.
- 따라서 설계과정에서 사용할 캐시의 수가 결정된다.

다양한 캐시기억장치의 구조



- (a) 가장 일반적인 구조로 1개의 캐시기억장치를 사용
- (b) 두 개의 캐시기억장치를 이용하여 계층적으로 구성한 구조다.
  - 캐시 1은 중앙처리장치에 내장되어 있는 경우가 일반적이다.
- (c) 캐시기억장치 3개를 이용하여 계층적 구조로 설계한 것
  - 캐시 1에 해당하는 부분이 분리된 형태의 두 개의 캐시로 발전
  - 각 기능에 따라서 명령어 캐시와 자료 캐시로 분리되었다.
  - 명령어 캐시와 자료 캐시는 CPU에 내장되어 있는 캐시기억장치다.

다음 시간

7주 3강. 캐시기억장치의 구조