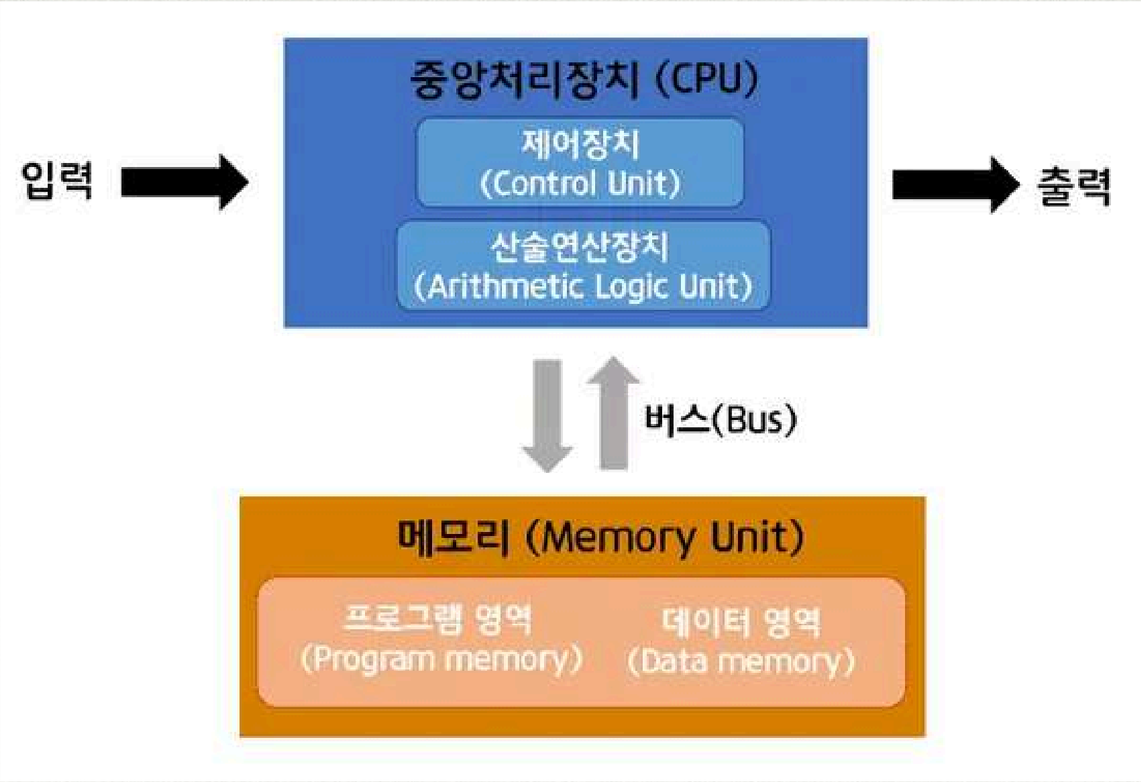
Cache

서울 7반 강병호

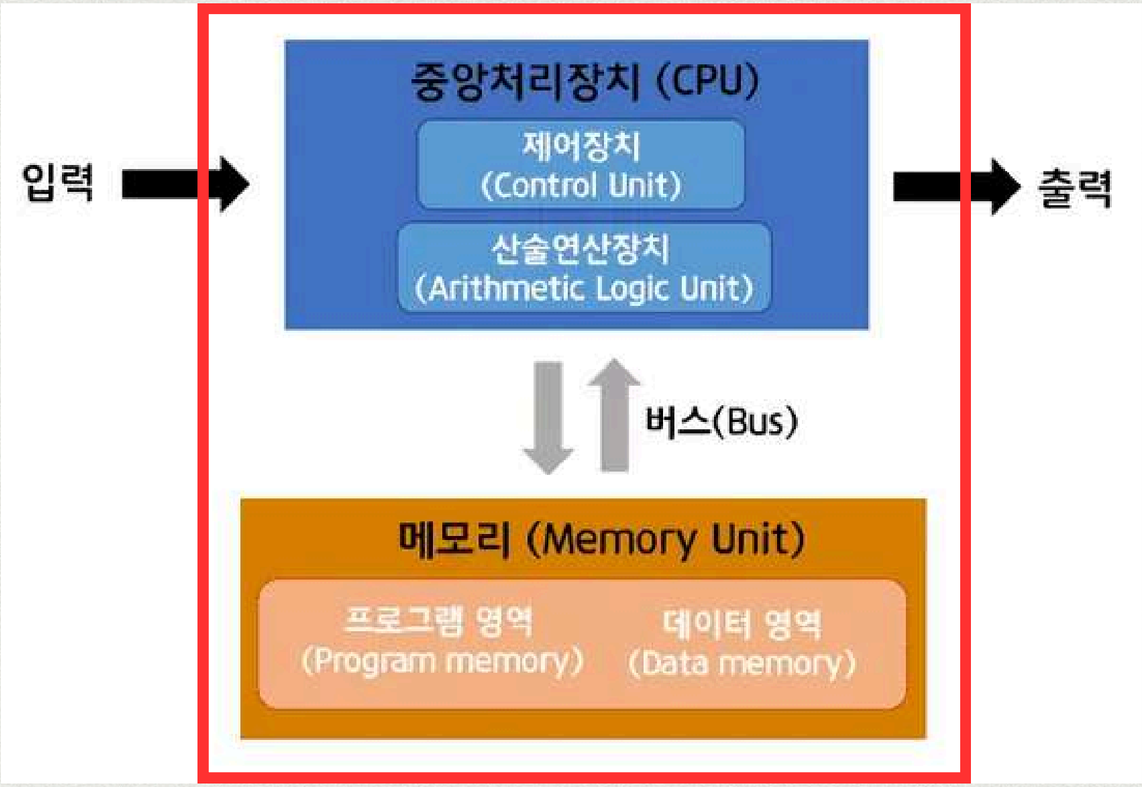
Contents

폰 노이만 아키텍처



- 중앙 처리 장치
- 메모리
- 입출력 장치
- 버스

폰 노이만 병목현상

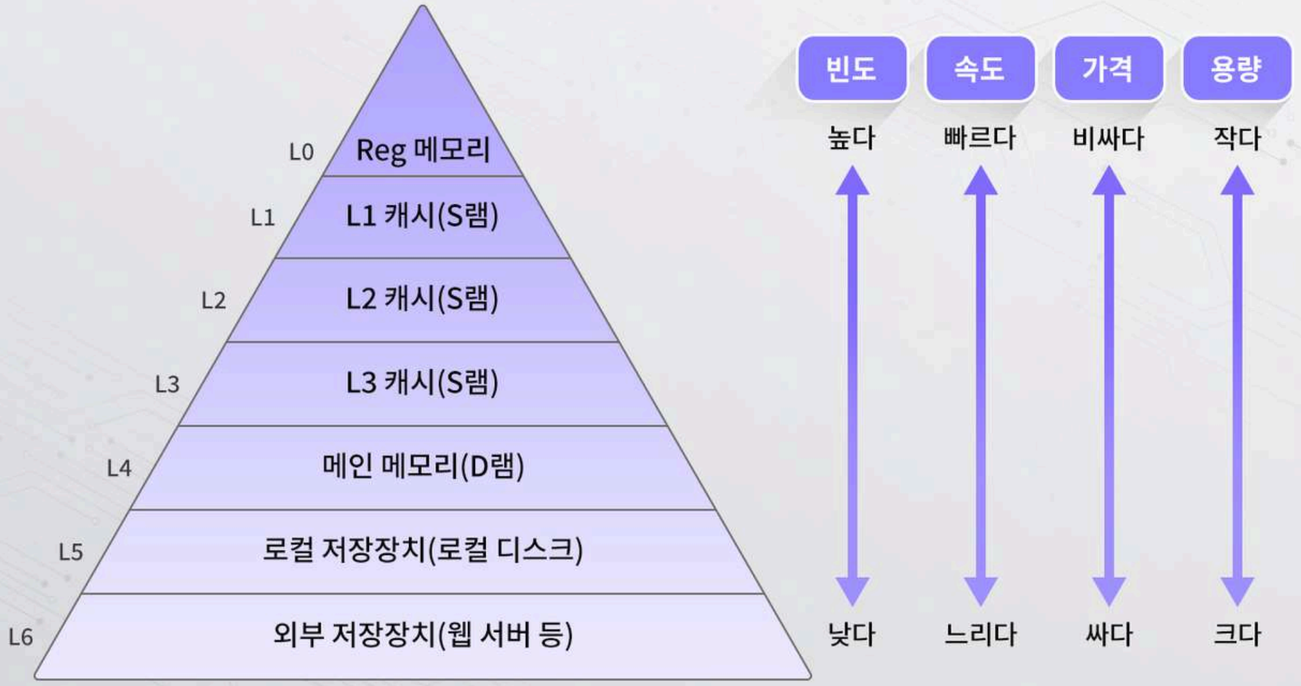


- 공통의 버스로 연결
- CPU, 메모리 속도의 불균형 심화

CPU-메모리의 속도 격차

SK hynix NEWSROOM

메모리 계층 구조(Memory Hierarchy)



SK hynix NEWSROOM

참조 지역성

**컴퓨터 프로그램이 메모리에 무작위로 접근하지 않고
최근에 사용한 데이터, 명령어를 참조하는 경향**

시간적 지역성

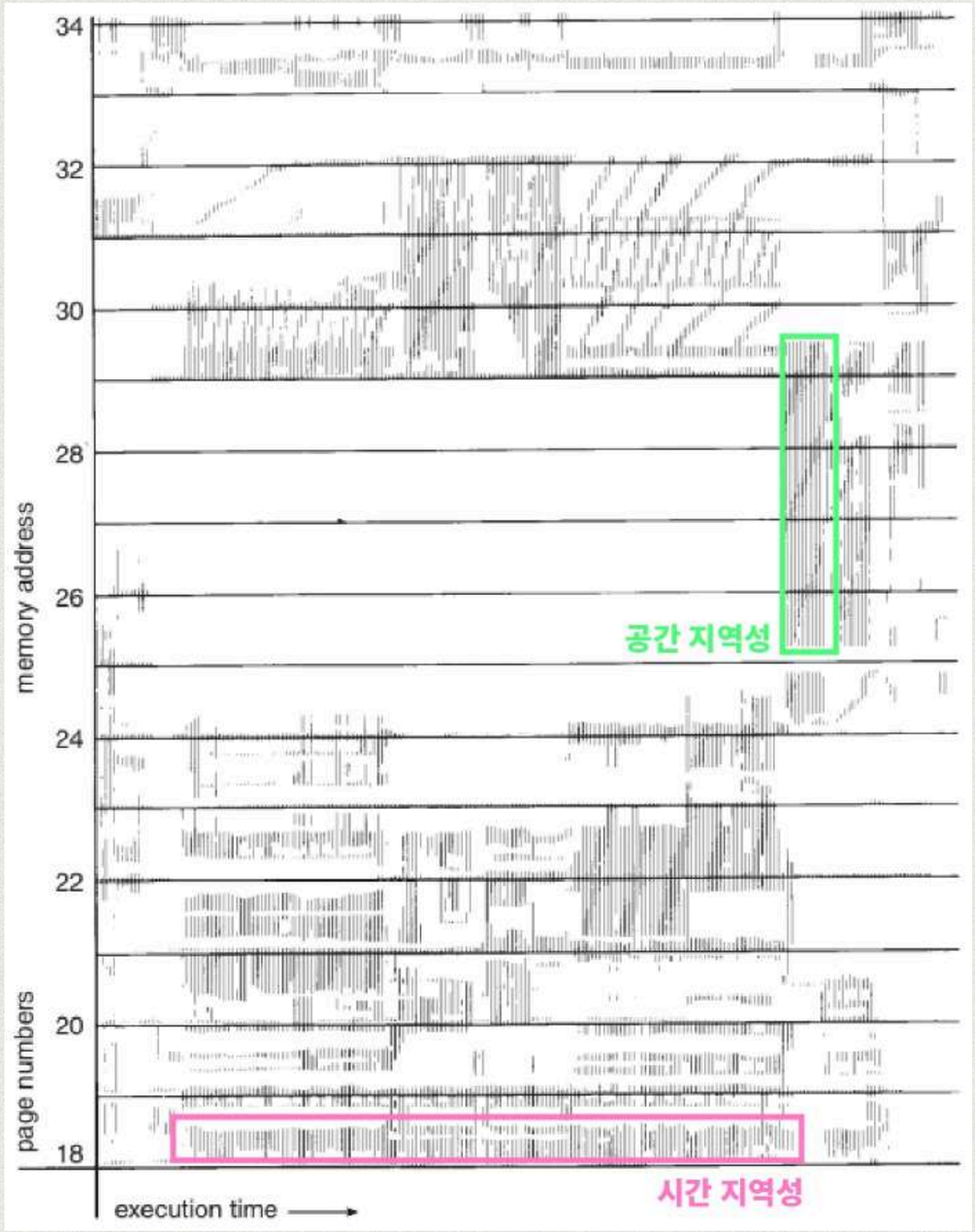
한번 참조된 메모리 위치가 가까운 미래에 다시 참조
될 가능성이 높은 경향

- 루프
- 서브루틴
- 스택택

공간적 지역성

프로그램이 특정 메모리 위치를 참조하면, 그 주변의 인접한 메
모리 위치들을 참조할 가능성이 높은 경향

- 배열 순회
- 명령어 실행


34
32
30
28
26
24
22
20
18
memory address
page numbers
execution time
공간 지역성
시간 지역성

캐시

속도가 빠른 장치와 느린 장치간의 속도차에 따른 병목 현상을 줄이기 위한 범용 메모리

전송 단위

- Word : CPU의 처리단위로 블록을 구성하는 기본 단위
- Block : 메모리를 기준으로 하였을 때 캐시와의 전송 단위

데이터 존재 여부

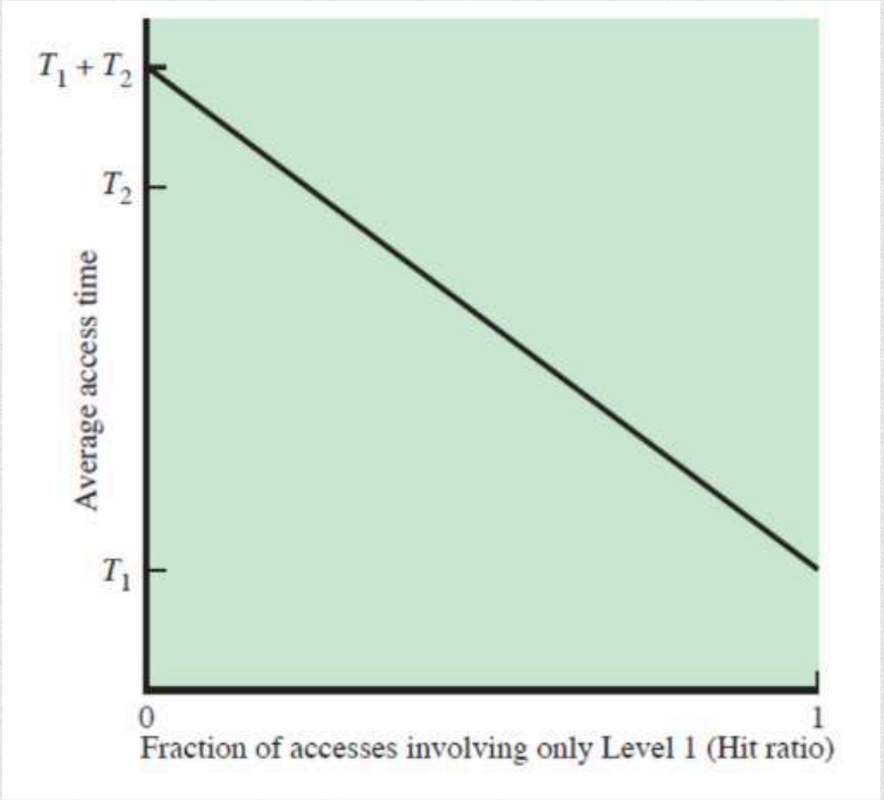
- Cache Hit : 필요한 데이터가 Cache에 있는 경우
- Cache Miss : 필요한 데이터가 넓는 경우

T_1 - 캐시 메모리의 액세스 시간

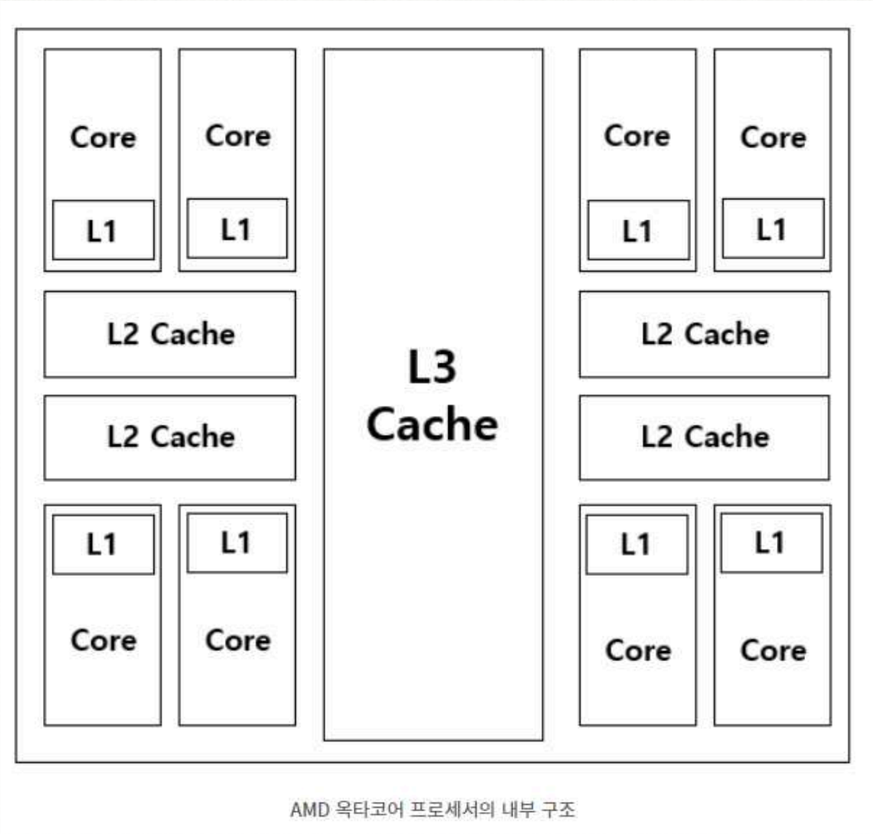
T_2 - 주기억장치의 액세스 시간

H - Hit rate(참조가 캐시 메모리에서 발견되는 비율)

$$H = \frac{\text{캐시 메모리에 적중되는 횟수}}{\text{전체 기억장치 액세스 횟수}}$$



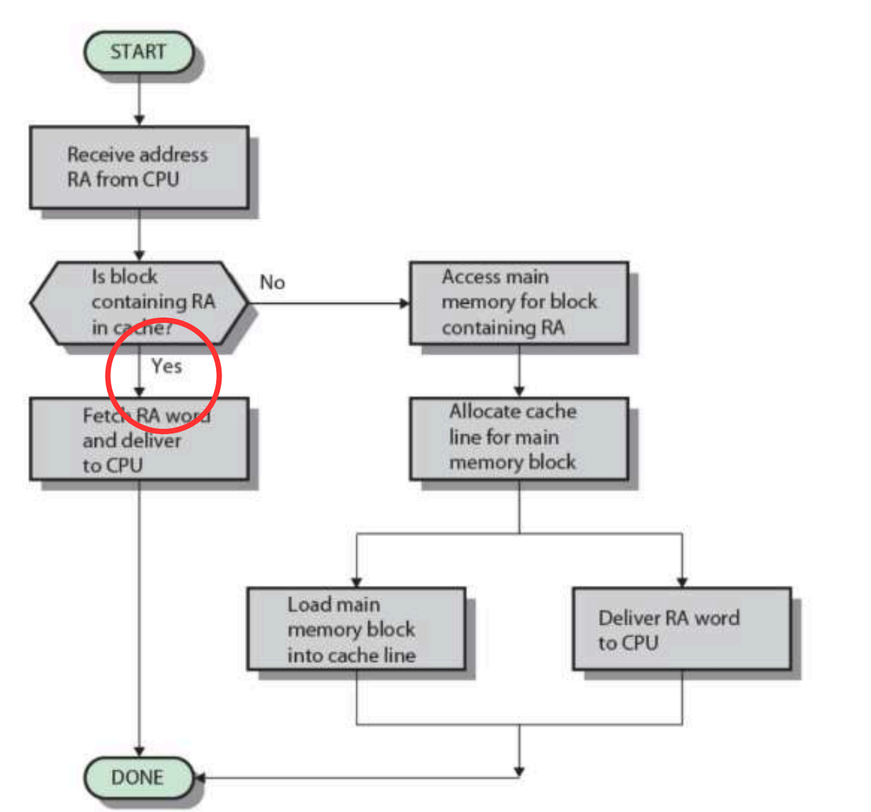
캐시의 종류



AMD 옥타코어 프로세서의 내부 구조

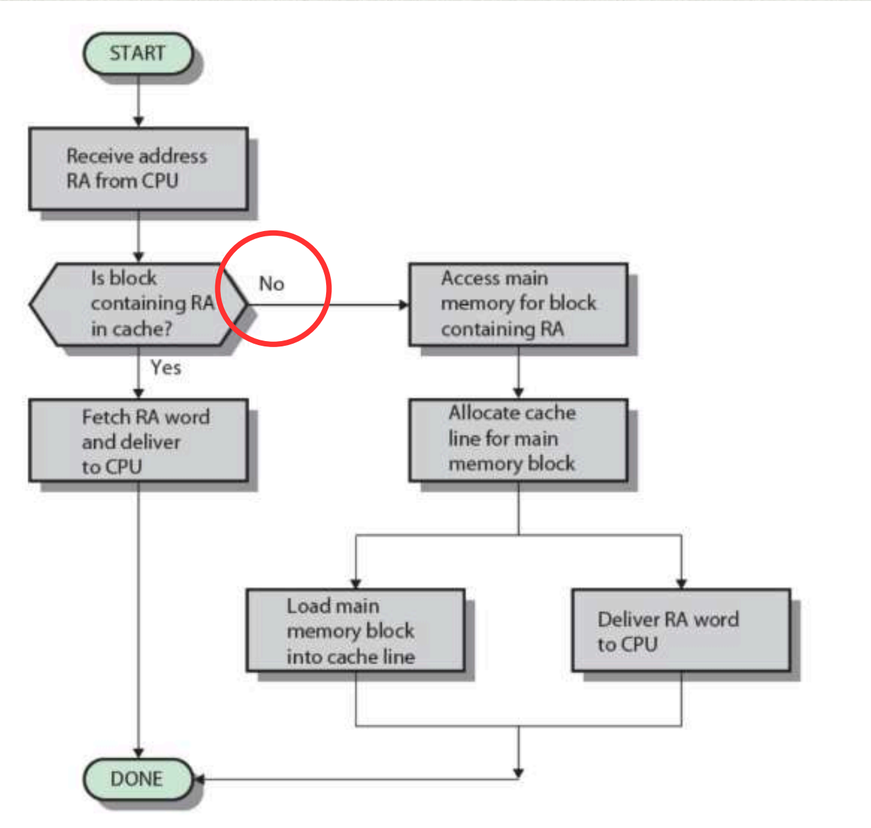
- L1 : 프로세서와 가장 가까운 캐시
- L2 : L1도 용량이 큰 캐시
- L3 : 멀티 코어 시스템에서 여러 코어가 공유하는 캐시

02 캐시 읽기 과정 - Cache Hit



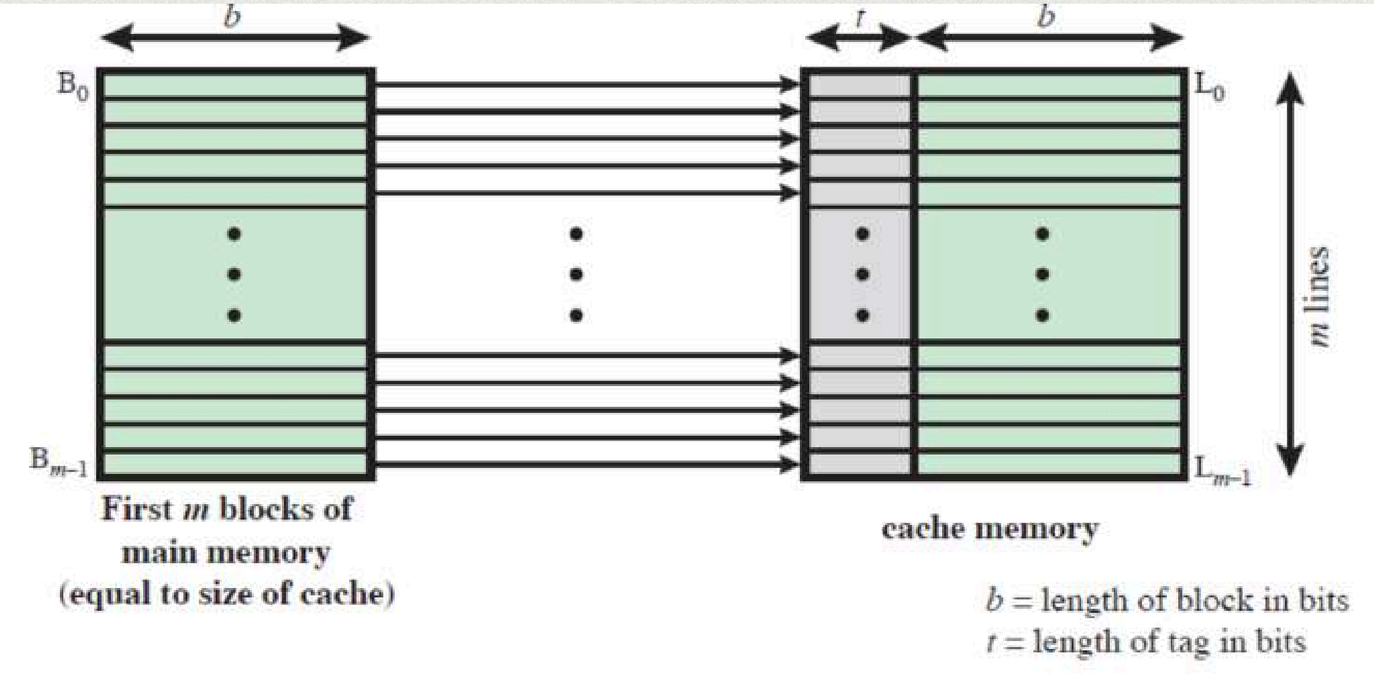
1. CPU로 부터 RA 주소 획득
2. 캐시에 RA를 포함하는 블록의 존재 확인
3. RA에 존재하는 단어를 CPU로 전달

02 캐시 읽기 과정 - Cache Miss



1. CPU로 부터 RA 주소 획득
2. 캐시에 RA를 포함하는 블록의 존재 확인
3. RA를 포함하는 block을 가진 메모리에 접근
4. 해당 블록 정보를 캐시에 저장
5. 요청한 정보를 CPU로 전달

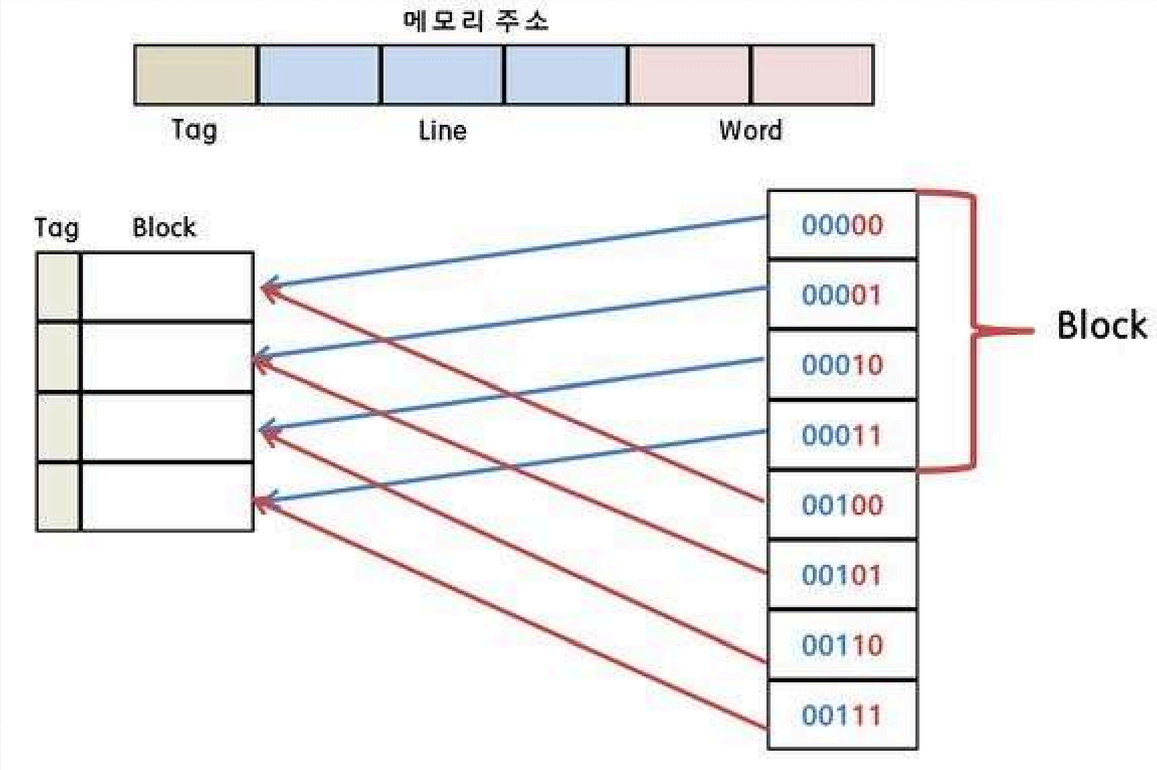
03 Mapping Function - Direct Mapping



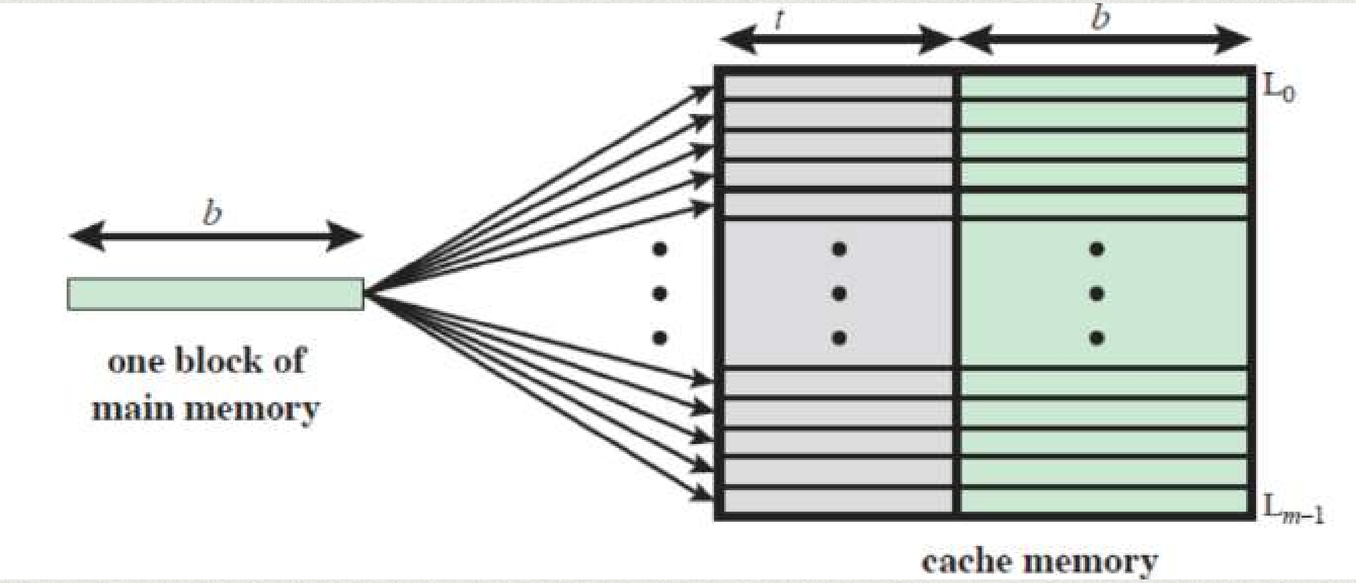
- 구현의 간단함
- Cache Miss 비율이 커짐

이미 정해져 있는 자리에 매핑

03 Mapping Function - Direct Mapping



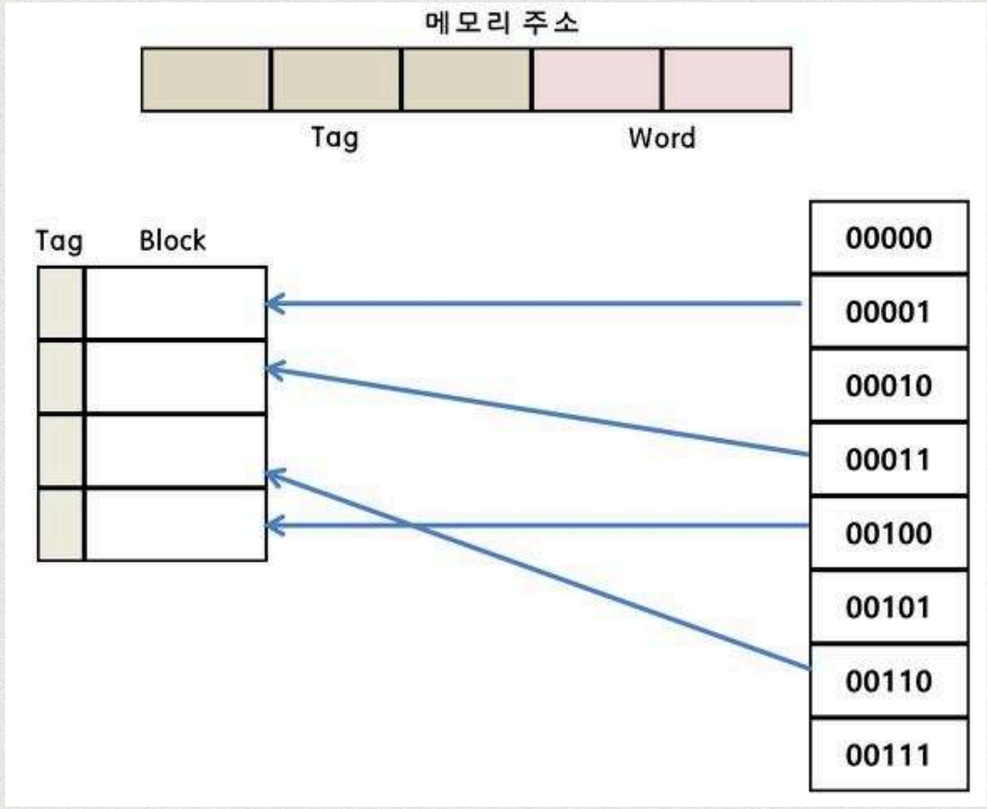
03 Mapping Function - Associative Mapping



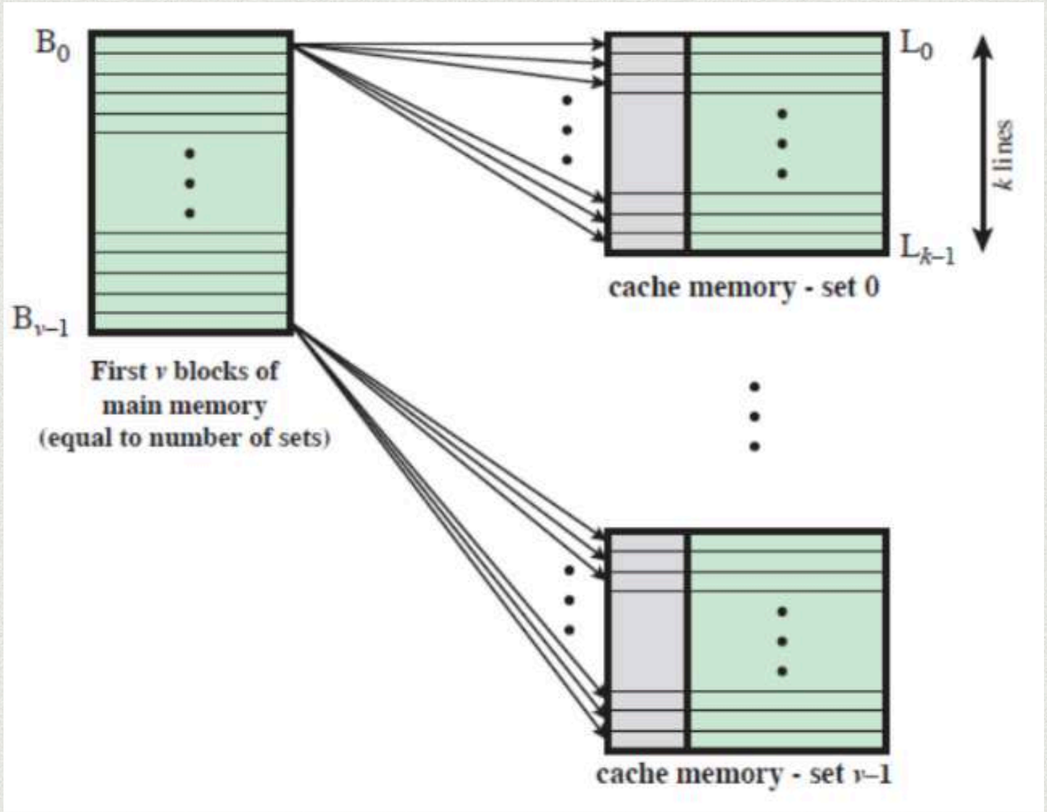
- Cache Miss가 발생하지 않음
- 캐시 라인과 tag의 일치 여부를 확인하기 위해 모든 캐시 라인 비교

주기억장치의 Block이 어떤 캐시 라인에도 적재되는 것

03 Mapping Function - Associative Mapping

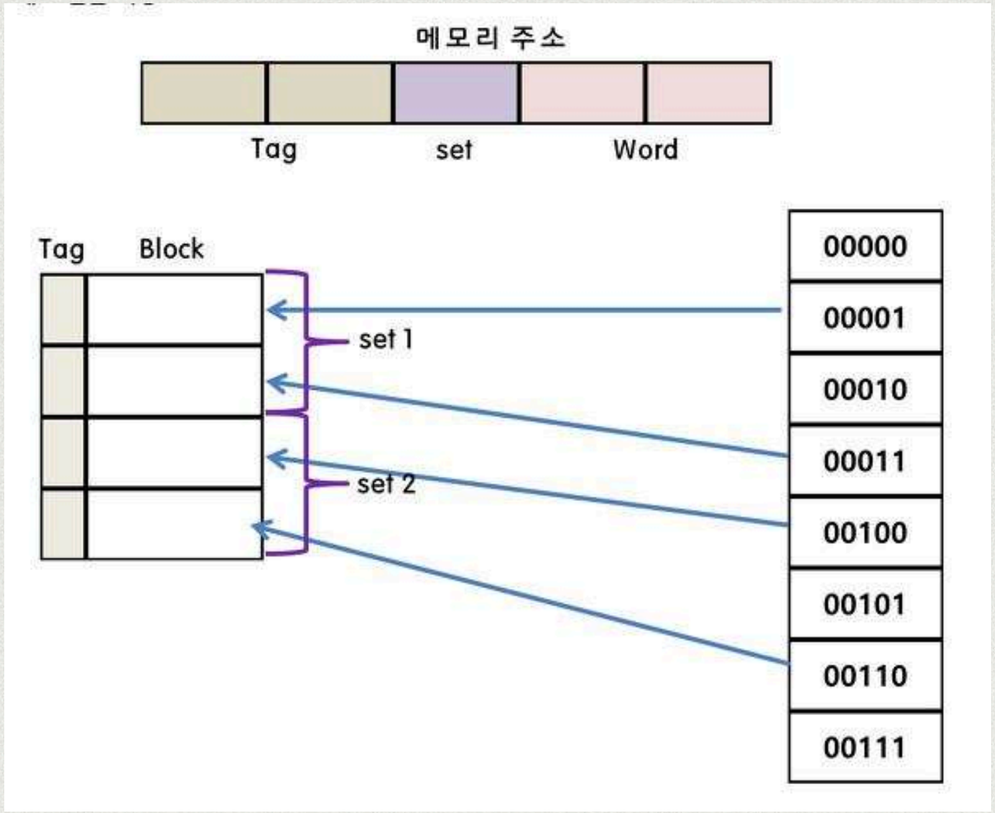


03 Mapping Function - Set Associative Mapping



캐시를 여러 세트로 나눈 후 각 메모리 블록은 정해진 세트 내에서 어디든 저장

03 Mapping Function - Set Associative Mapping



03 Mapping Function

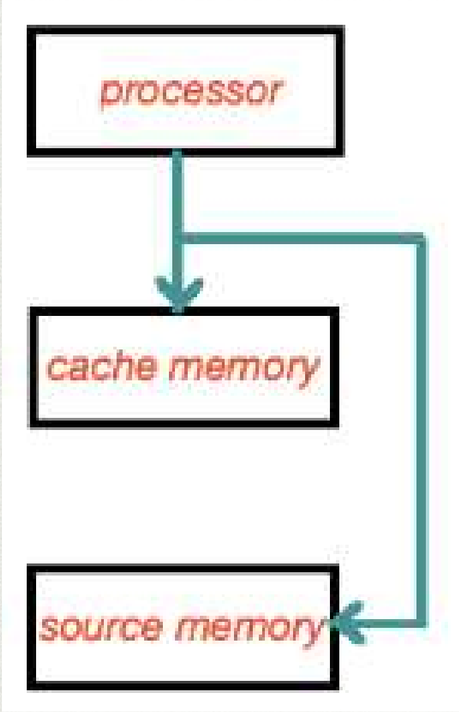
구분	직접 매핑 (Direct Mapping)	연관 매핑 (Associative Mapping)	집합 연관 매핑 (Set-Associative Mapping)
저장 방식	메인 메모리의 각 블록은 캐시의 정해진 1개의 라인에만 저장됨	메인 메모리의 블록이 캐시의 아무 라인에나 저장 가능	메인 메모리의 블록이 특정 집합(Set) 내의 라인들 중 하나에 저장 가능
주소 구조	`Tag + Line + Word`	`Tag + Word`	`Tag + Set(Index) + Word`
장점	- 구현이 간단 - 하드웨어 비용이 낮음	- 충돌이 거의 없음(겹쳐도 다른 라인에 저장 가능) - 적중률 높음	- 직접 매핑보다 충돌 적음 - 연관 매핑보다 하드웨어 비용 적음
단점	- 같은 라인에 매핑되는 블록이 많으면 충돌 심함 - 적중률 낮아질 수 있음	- 원하는 블록이 캐시에 있는지 확인하려면 전체 라인 탐색 필요 → 하드웨어 복잡 & 비용 높음	- 집합 크기에 따라 성능이 달라짐 - 여전히 교체 알고리즘 필요
교체 방식	필요 없음 (자리 고정)	교체 알고리즘 필요 (LRU, FIFO 등)	교체 알고리즘 필요 (집합 내에서만)
적중률	낮음 (특히 충돌이 많은 경우)	가장 높음	중간 (적중률과 비용 균형)
하드웨어 복잡도	가장 단순	가장 복잡	중간

04 Cache Write

- 데이터를 읽는 동작이 아닌 입력하는 동작에서 발생
- 데이터를 변경할 주소가 캐싱되어 있으면 캐시 블록의 데이터가 업데이트

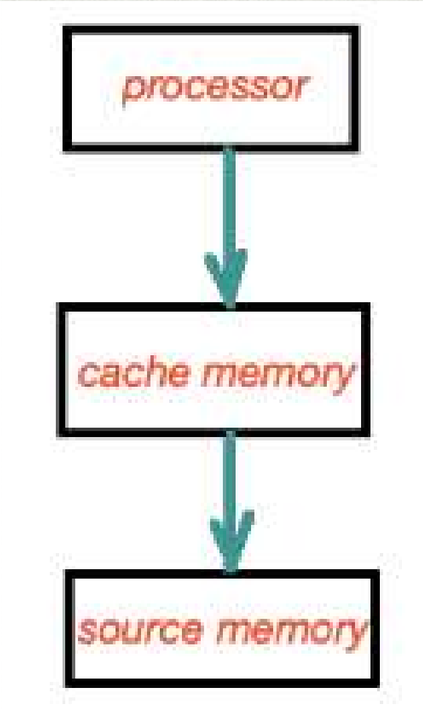
언제 메모리에 업데이트?

04 Write Through



- 캐시에 데이터 작성될 때마다 메모리에 업데이트
- 항상 메모리와 캐시의 데이터 동일하게 유지
- 많은 트래픽 발생

04 Write Back



- 블록 교체시에만 메모리의 데이터 업데이트
- dirty 비트를 사용해 데이터의 변경 확인 (1이면 변경으로 메모리의 데이터 변경)
- 캐시 크기의 증가
- 메모리 엑세스의 감소

THANK YOU

감사합니다.