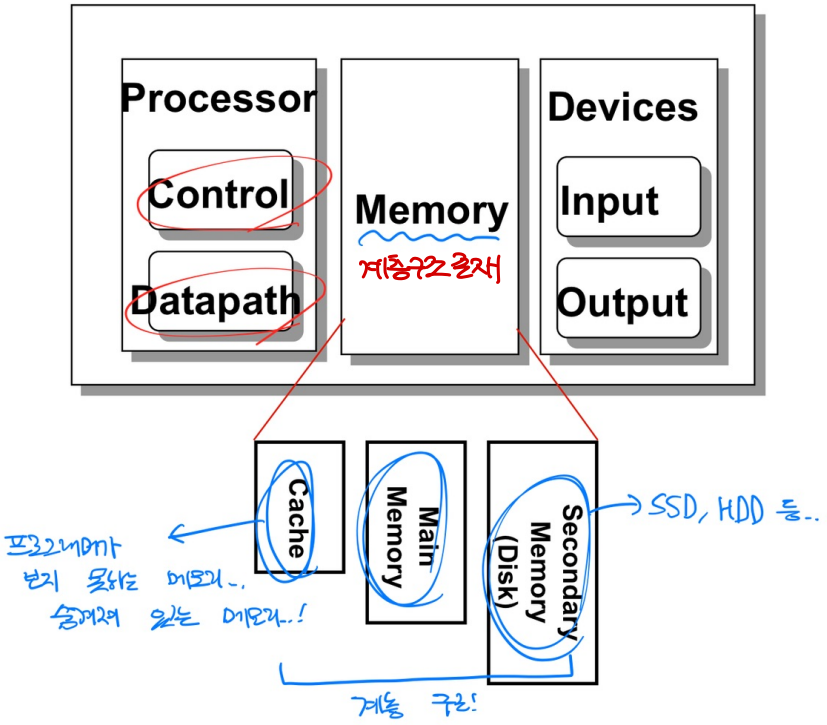
Ch.5-1 Large and Fast: Exploiting Memory Hierarchy Part 1 Cache Memory

* Computer의 주요 Component



- processor ⇒ 속도가 정말 빨라짐
  But, DRAM ⇒ 〃 느려진다면? → 전체적인 컴퓨터 성능 Bad
  ⇒ cache memory 개념 탄생
- Memory Hierarchy
  i) size ↑ → speed ↓
  size ↓ → speed ↑ → memory마다 특징 다름.
  ⇒ 적재적소에 알맞게 이용해서 memory 전체가 좀 더 싸고 빠른것처럼?
  ⇒ 계층적 병렬적으로 구현 가능

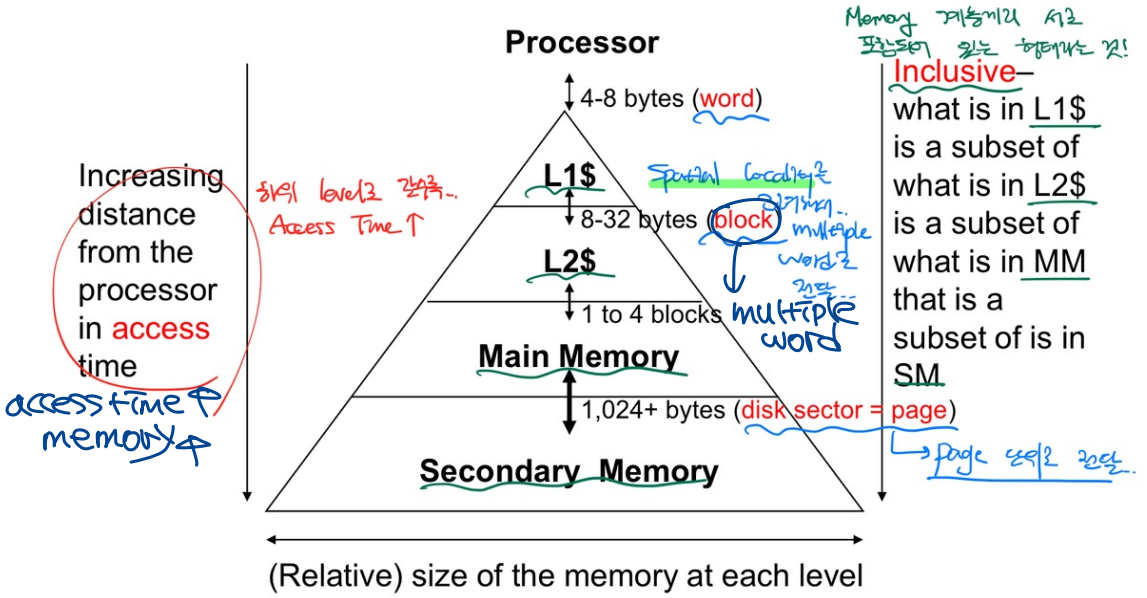
ii) Hierarchy가 유용한 이유 ⇒ Locality

- Locality : access하는 부분을 계속 access하는 경향 존재
  - Temporal Locality 최근에 참조된 item이 또 참조될 가능성 ↑
  - Spatial Locality " 근처에 있는 item "

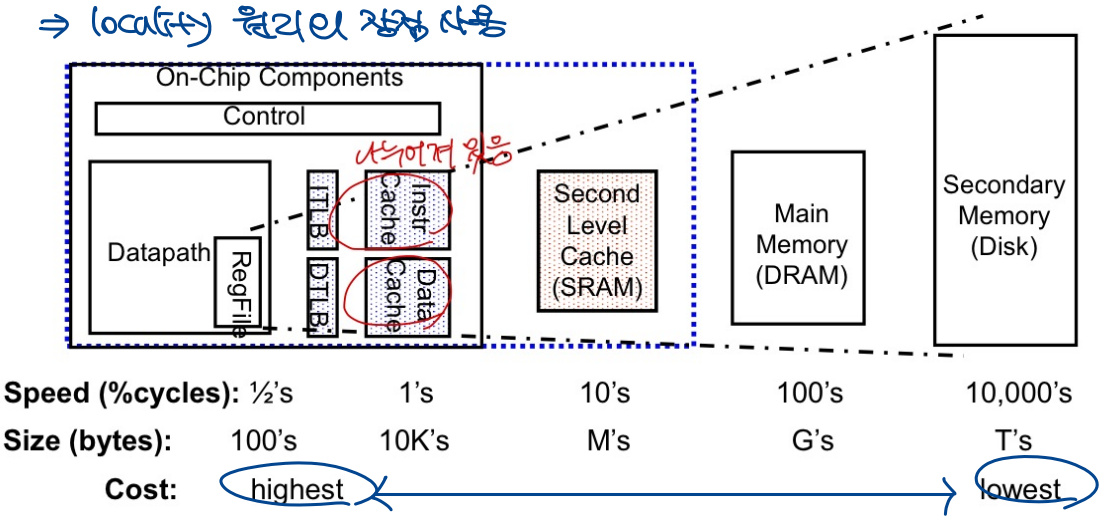
iii) Hierarchy 관련 terms

- Block : 정보의 최소 단위 (multiple words) → copy할 때 Block 단위로
- Hit Rate : memory access를 통해 원하는 Data 찾는 비율
  - ↳ Hit Time : access block + determine hit/miss
- Miss Rate : " 찾지 못하는 " (= 1 − Hit Rate)
  - ↳ Miss Penalty : 특정 level에서 해당하는 Block → 더 low level로 replace

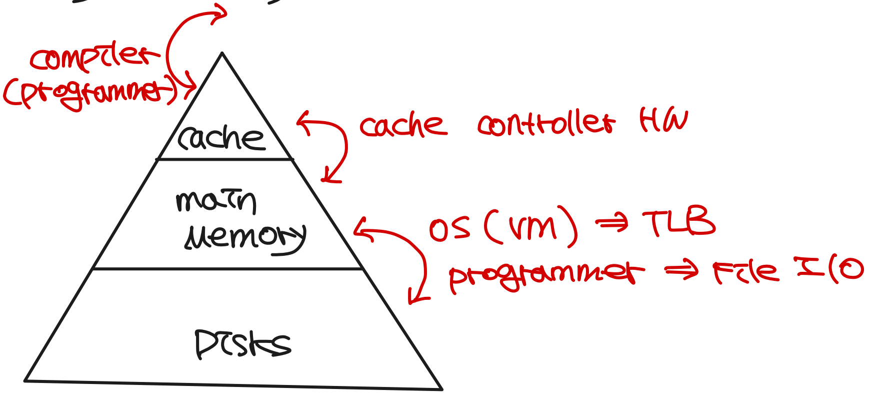
∴ Hit Time << Miss Penalty (∵ 하위 level로 갈수록 시간↑)



⇒ locality 원리의 실제 사용



IV) Memory Hierarchy Tech

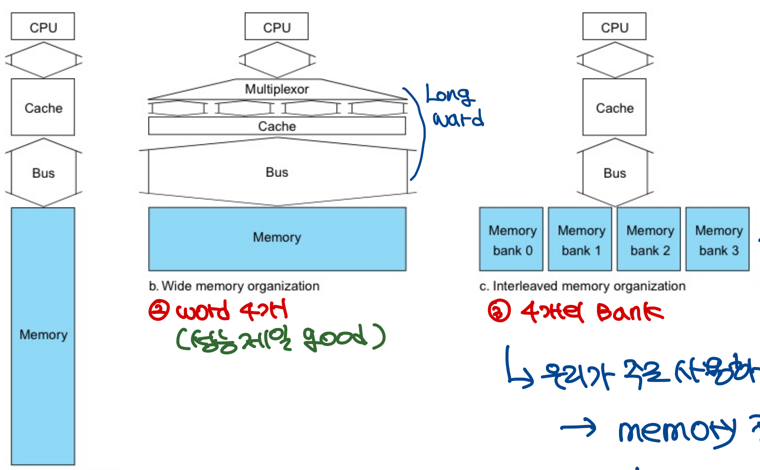


① Cache : SRAM
- 속도↑ (access time : 0.5 ~ 2.5 ns)
- 집적도↓ ⇒ power 많이 필요, 가격↑
- Static : 비휘발성!!

② Main memory : DRAM
- 속도↓
- 집적도↑ ⇒ power가 비교적 덜↓, 가격↓
- Dynamic : 주기적으로 refreshed (~every 8초)
  - ↳ DRAM의 1~2% 사용
- Address : row, column 나누어 할당
  - RAS : row detector triggering
  - CAS : column 〃

↳ Advanced DRAM organization
- DRAM에 존재하는 모든 bit ⇒ rectangular array처럼 조직 (→ 모든 row에 접근)
  - ⇒ Burst Mode : continuous한 정보를 Main Memory에 한 번에 read
    - → latency ↓
- Double Data Rate (DDR) : rising edge에서 전송
- Quad 〃 (QDR) : DDR의 input, output 분리

③ Increasing Memory Bandwidth → memory Bank 이용 ⇒ multiple word read 시용!



a. One-word-wide memory organization
① word 하나

b. Wide memory organization
② word 4개 (성능제일 good)

c. Interleaved memory organization
③ 4개의 Bank

Bank? → I/O가 독립적으로 이루어질 수 있는 단위

↳ 우리가 주로 사용하는 방식
→ memory 접근을 여러 곳으로 동시에 (4 Bank)
→ data transfer는 차례대로!

③ = ①의 H/w + ②의 성능

word알무기 ☆!

⇒ example addr + DRAM + Data ★

memory system
- cache block : 4 words per block
- 1 memory bus clock cycle : address 전송
- 15 " : DRAM 초기 상태 정근
- 1 " : data (word 단위) 전송

• One-word-wide
1번의 address 전송 + 4개의 DRAM 정근 ⇒ 1 + 4×15 + 4×1 = 65 bus cycles (addr DRAM data)

• Wide-memory bank
1번의 " + 2개의 " ⇒ 1 + 2×15 + 2×1 = 33 "

• Interleave ★
1번의 " + 1개의 " + 4개의 cycle ⇒ 1 + 1×15 + 4×1 = 20 "
↳ 효율적

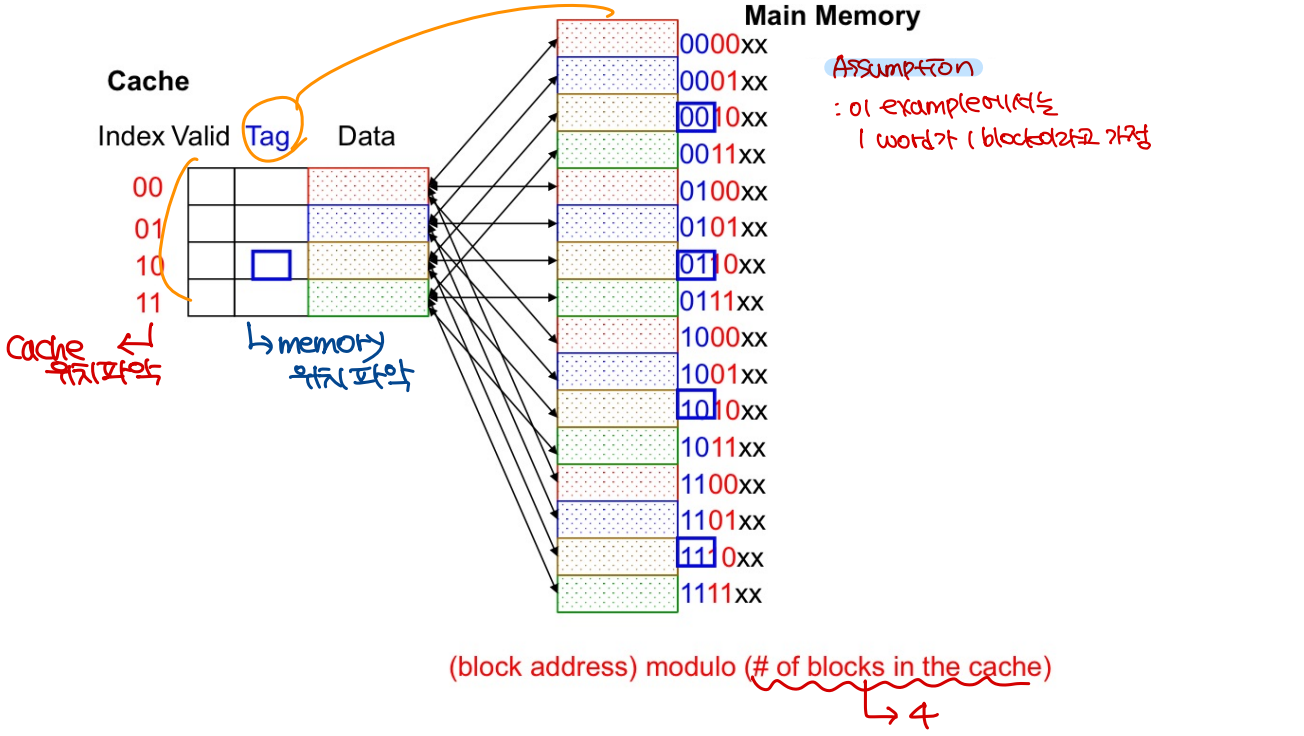
★ Caching

Q1 : cache 안에 data item이 있는지 어떻게?
Q2 : 안에 있다고 하면 어떤 방식?

① Direct mapped
memory의 각 word에 cache 내의 위치를 memory 주소에 기반을 두어 할당
↳ 각 word가 한 장소 내에서만 있을 수 있음

ㄱ) Address mapping : (Block address) modulo (Cache Block 개수)

ㄴ) tag : ~~Memory Block 구분함~~ ⇒ 상위 26개
★ index : ~~Cache Block~~ 구분함 ⇒ tag 다음 2bit (modulo 연산 값) ★



② Hit / Miss ⇒ Cache 조작 여부

- Read Hits : 읽었을 때 cache에 해당 Data 존재
- Read Misses : CPU 명령 → memory로부터 Data 가져옴 → 재시작

cache란 → 속도 많이 느림

- Write Hits
  - write-through : Cache, main memory 동시에 write ⇒ 불일치 해결하려고
  - write-back : 우선 Cache에 있는 것만 다 write → 이후에 memory update → 일관성 알고리즘 필요
- Write Misses : cache로 Block 전체 Read → word write
  - ↳ Read misses → data → write hits

Start with an empty cache - all blocks initially marked as not valid 0 1 2 3 4 3 4 15

idx tag

0 miss

00	00	Mem(0)

1 miss

00	00	Mem(0)
01	00	Mem(1)

2 miss

00	00	Mem(0)
01	00	Mem(1)
10	00	Mem(2)

3 miss

00	00	Mem(0)
01	00	Mem(1)
10	00	Mem(2)
11	00	Mem(3)

Assumption : 이 example에서는 1 word가 1 block이라고 가정

4 miss

00 (01)	00 (4)	Mem(0)
01	00	Mem(1)
10	00	Mem(2)
11	00	Mem(3)

3 hit

01	00	Mem(4)
00	00	Mem(1)
00	00	Mem(2)
00	00	Mem(3)

4 hit

01	00	Mem(4)
00	00	Mem(1)
00	00	Mem(2)
00	00	Mem(3)

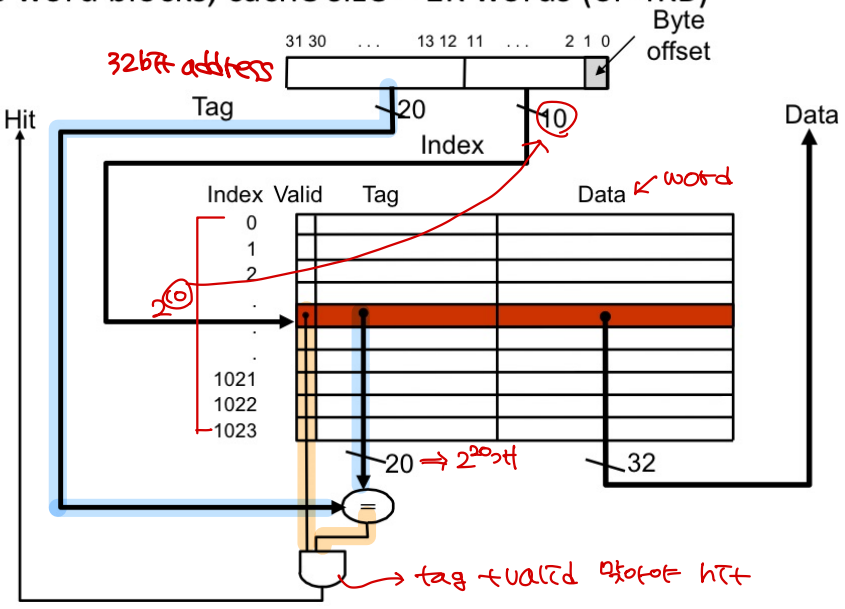
15 miss

01	00	Mem(4)
00	00	Mem(1)
00	00	Mem(2)
11 (00)	00 (15)	Mem(3)

8 requests, 6 misses

③ Direct Mapped Example

1) One word blocks, cache size = 1K words (or 4KB)



⇒ Spatial Locality 장점

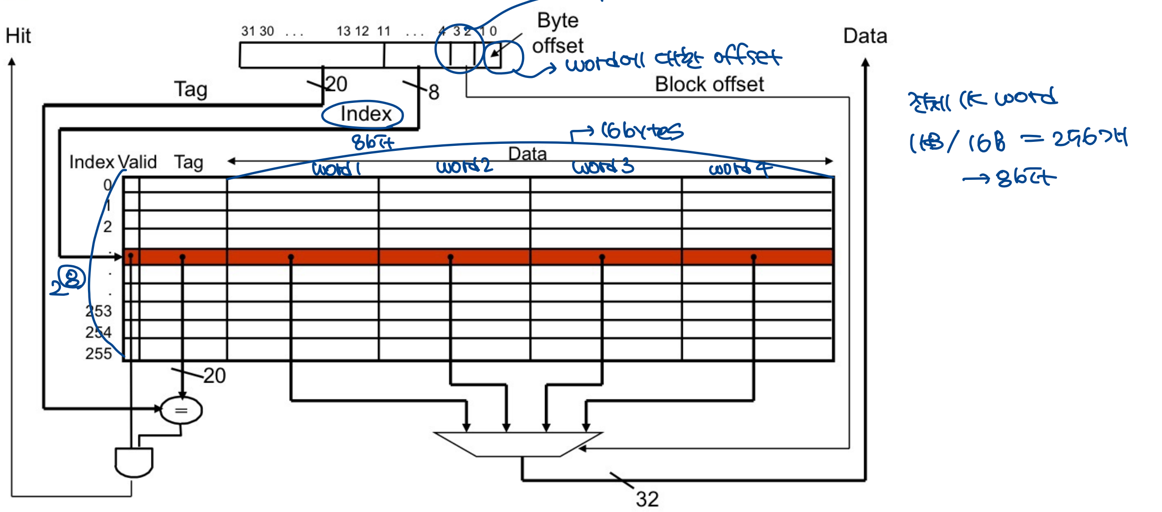
(한번 access된 data의 근처가 자주 옴 ⇒ 1 word block 단위로 저장)

ex) address 1200 → 각 64개의 16byte Block

- Block address = 1200 / 16 = 75 → 75번째서
- Block number = 75 modulo 64 = 11 (index)

- Cache size : 64×16 = 1KB
- Block당 4개의 word (16B)

ㄸ) ❑ Four words/block, cache size = 1K words → Block offset



전체 1K word
1KB / 16B = 256개
→ 8bit

⇒ Spatial Locality 장점 ⇒ Block이 더 넓은 범위 나타냄

❑ Let cache block hold more than one word

Cache 내에 존재하는 Cache Block 수 : 2

Start with an empty cache - all blocks initially marked as not valid

miss나면 sequential하게 다음 word도 가져옴

0 1 2 3 4 3 4 15 ⇒ address

idx: 0 0 1 1 2 1 2 7 → address / 2 → Cache Block

tag: 0 0 0 0 1 0 1 3 → address mod 4 → word 개수

0 miss

00	Mem(1)	Mem(0)

1 hit

00	Mem(1)	Mem(0)

2 miss

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

↳ 같이!!

3 hit

00	Mem(1)	Mem(0)
00	Mem(3)	Mem(2)

4 miss

~~00~~ 01	~~Mem(1)~~ 5	~~Mem(0)~~ 4
00	Mem(3)	Mem(2)

3 hit

01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

4 hit

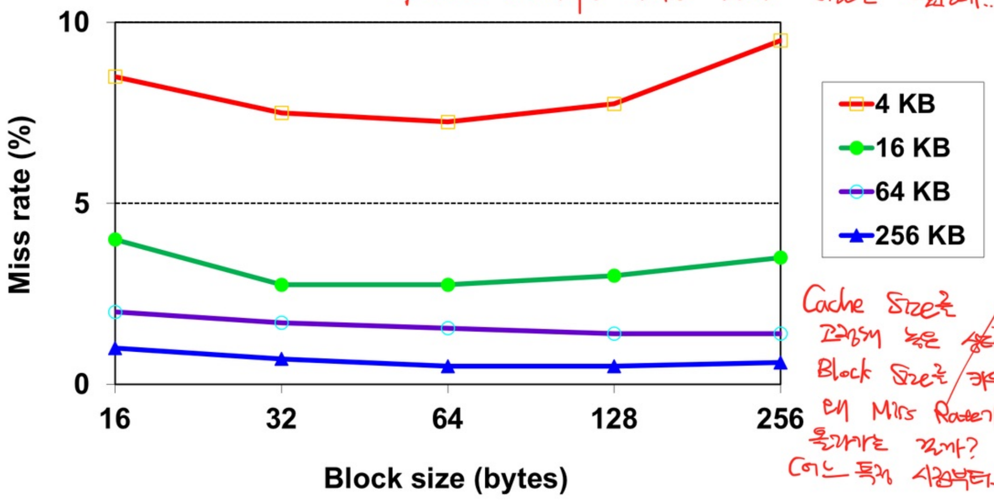
01	Mem(5)	Mem(4)
00	Mem(3)	Mem(2)

15 miss

01	Mem(5)	Mem(4)
~~00~~ 11	~~Mem(3)~~ 15	~~Mem(2)~~ 14

8 requests, 4 misses

Spatial Locality를 위해.. Block Size를 키웠는데..



Cache Size를 고정시켜 놓은 상태에서 Block Size를 키우면 왜 Miss Rate가 올라가는 걸까? (어느 특정 시점부터..)

어느 시점 이후로 다시 증가
↳ 동일한 size의 cache에 보관할 수 있는 block 수가 block size가 커지면 커질수록 더 ↓

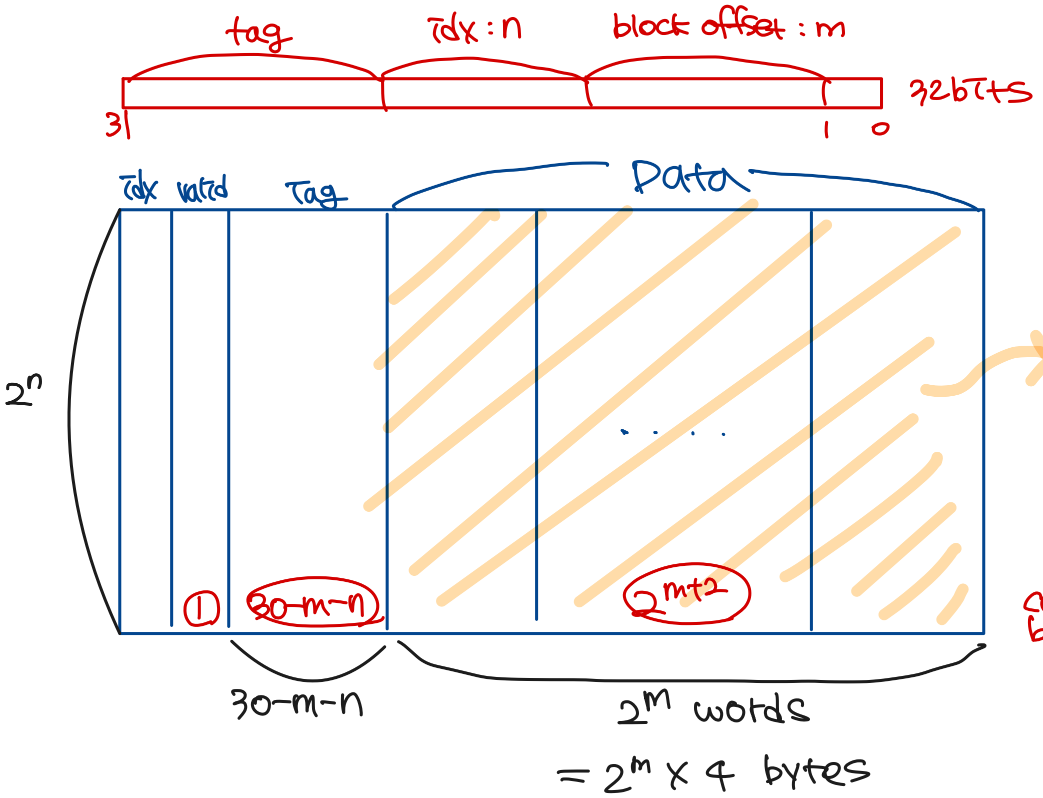
④ Block size Consideration

적절한 크기로 정하는 것이 중요

⇒ Block 크기 ↑ → miss rate ↓ But, pollution ↑

→ miss penalty ↑

⑤ Cache field size 계산해보기



* cache size

$2^n \times (2^m \times 4)$

$= 2^{n+m+2}$ bytes

* cache를 위해 필요한 memory size

$2^n \times (1 + 30-m-n + 2^{m+2})$ (valid, tag, Data)

2^n → cache block 개수

→ cache block 하나

⑥ Cache hits handling

i) Read hits : 원하는 결과 (I$, D$ 모두 !)

ii) write hits : (D$ 에서만)

- cache, memory ⇒ 일관성 → But, 성능 Bad
- 〃 ⇒ 일관성 X 허용 ⇒ write-back

⑦ Cache Miss handling

i) Read misses (I$, D$ 모두 !) ↳ main memory로부터 data 가져옴

⇒ Block size ↑ miss penalty ↑

- Early Restart
- Requested word first

ii) write misses (D$ 에서만)

- write allocate : 쓰기 실패만 덮어씌우기 (memory Read)
- No-write allocate : Block 자체를 갱신 (memory Read X)

→ memory에 block 가져온 후 word을 block에 씀

⑧ 성능측정 with Direct mapped cache

* simplified model

execution time (실행시간) = (execution cycles + memory stall cycles) x cycle time

memory stall cycles = # of inst x miss rate x miss penalty

⇒ miss rate 혹은 miss penalty 줄여보자

* example

- Cache miss rate → instruction : 2%
- " → Data : 4%
- CPI (stall 없이) → 2 (perfect cache ver)
- Miss penalty → 100 cycles
- Total memory reference → 40% (LW / SW)

⇒ miss cycles = I x 2% x 100 = 2I

Data " = I x 40% x 4% x 100 = 1.6I

CPI-stalls = 2 + 1.6 + (2) = (5.6) * ↝ 2.8배 speed up

↳ perfect

* average access time

AMAT = Hit Time x miss rate x miss penalty

⑨ Associative Caches

- Direct mapped
- 2-way set (덜비쌈)
- Fully associative (비쌈).

↑ tag ↓, index ↑

tag ↑ index ↓

⑩ Replacement Policy

- Direct mapped parallel하게 진행
- set associative Cache → 정해진 규칙 내에서 block 선정
- LRU → HW 필요
- Random

⇒ associativity 늘릴수록 miss rate 현저하게 감소

* Multi-Level Caches

① L1 Cache : 작고 빠름

② L2 Cache : 크지만 느림

example) CPU base CPI = 1

Clock rate = 4GHz ⇒ cycle time = 0.25ns

Instruction miss rate = 2%

memory access time = 100ns

1) L1 cache만 있을 때

Miss penalty = 100ns / 0.25ns = 400 cycles

Effective CPI = 1 + 0.02 × 400 = 9

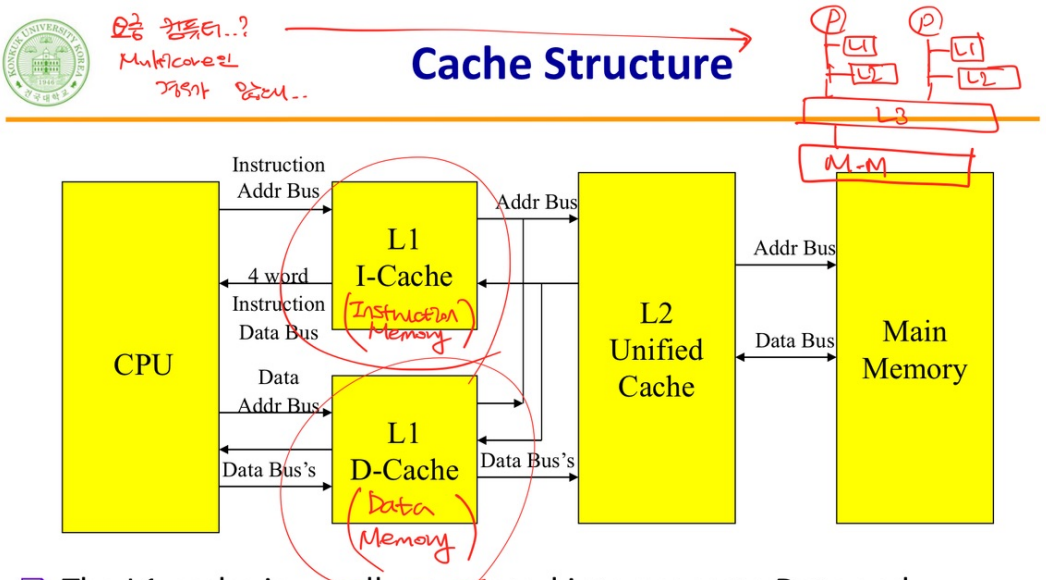
2) L2 cache 추가

access time = 5ns

L1 → L2 miss rate = 0.5%

⇒ L2 miss penalty = 5ns / 0.25ns = 20 cycles

⇒ 전체 CPI = 1 + 0.02 × 20 + 0.005 × 4000 = 3.4



- The L1 cache is usually separated into separate Data and Instruction caches (L1 cache는 주로.. Data/Instruction Cache로 나뉘어진다!)
- The L2 and higher caches are normally unified caches, i.e. both instructions and data are in the same cache