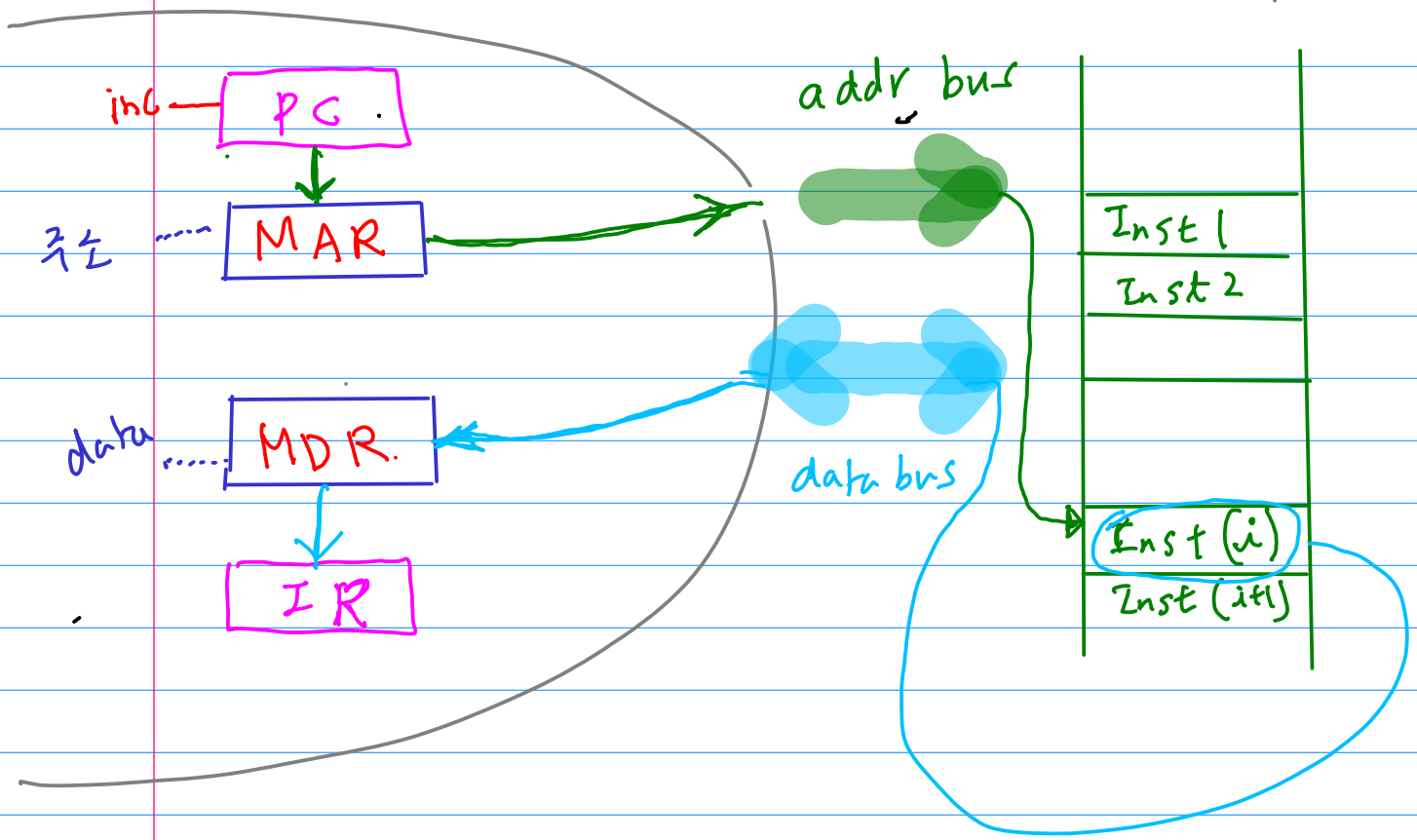
Cache Memory (H.1)

2015.06.11

www.wikiversity.org

The necessities in Computer Organization

명령어 읽기



pc가 가리키는 주소에는
다음에 읽어올 명령어가 있다

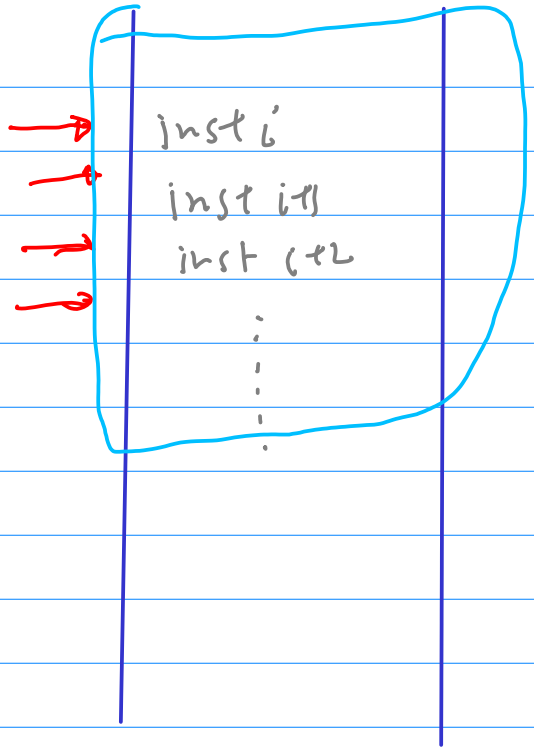
메모리에 있는 명령어를
순차적으로 수행하는 경우가 많다.

Instruction i ← memory
Instruction i+1 ← access는
Instruction i+2 ← 근처에서
Instruction i+3 ← 일어난다
Instruction i+4 ←

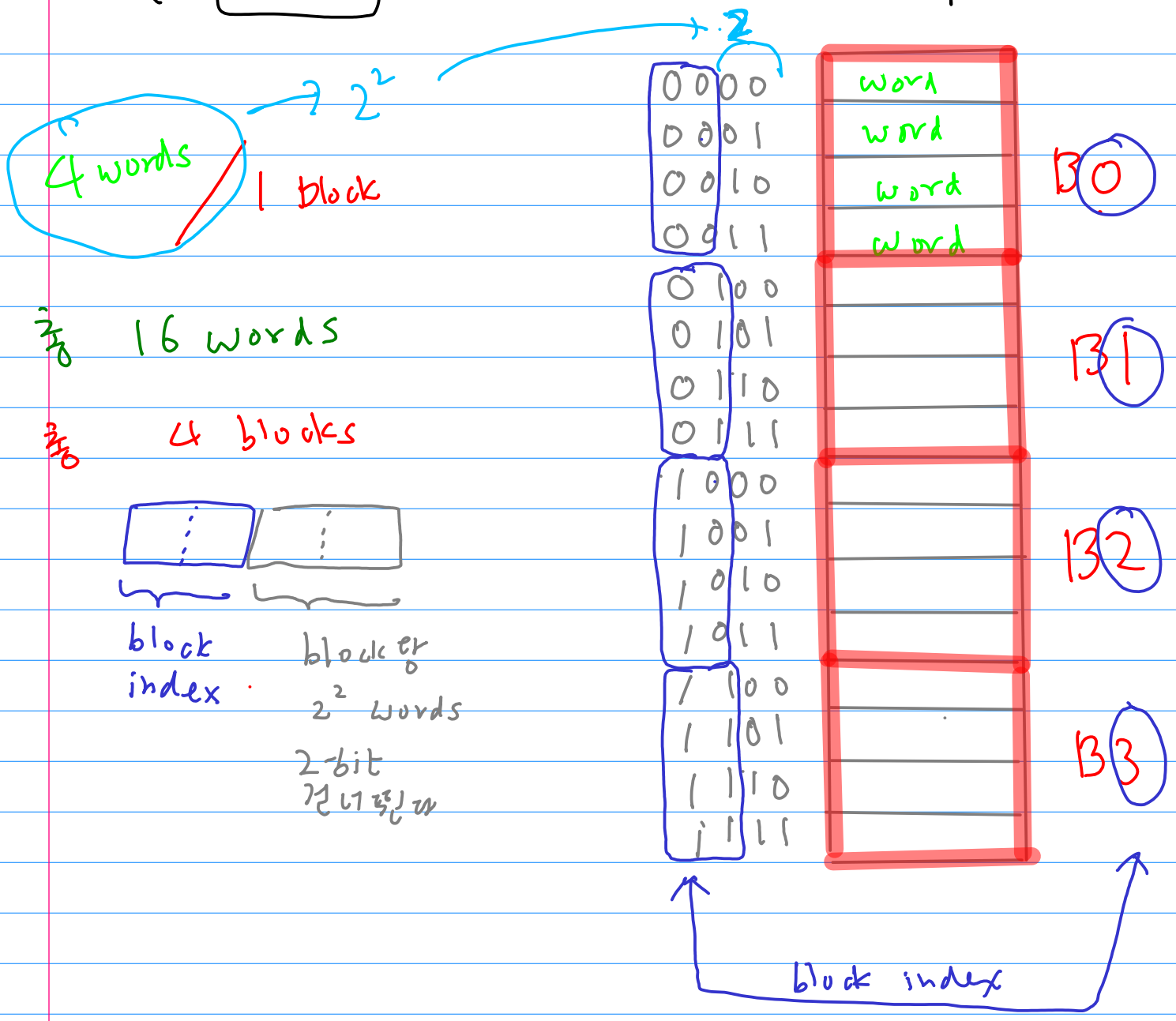
Locality of reference

지역성

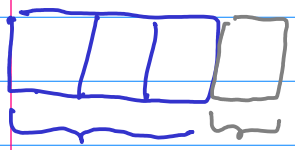
locality of reference



Fixed size **block** .... main memory partition



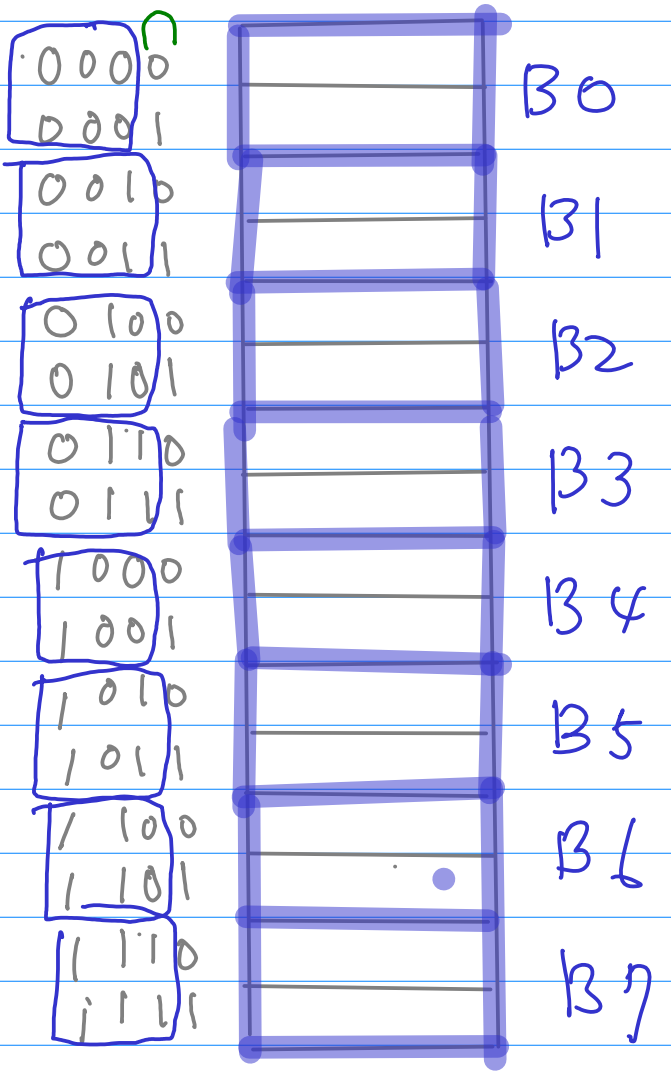
2 words / 1 block

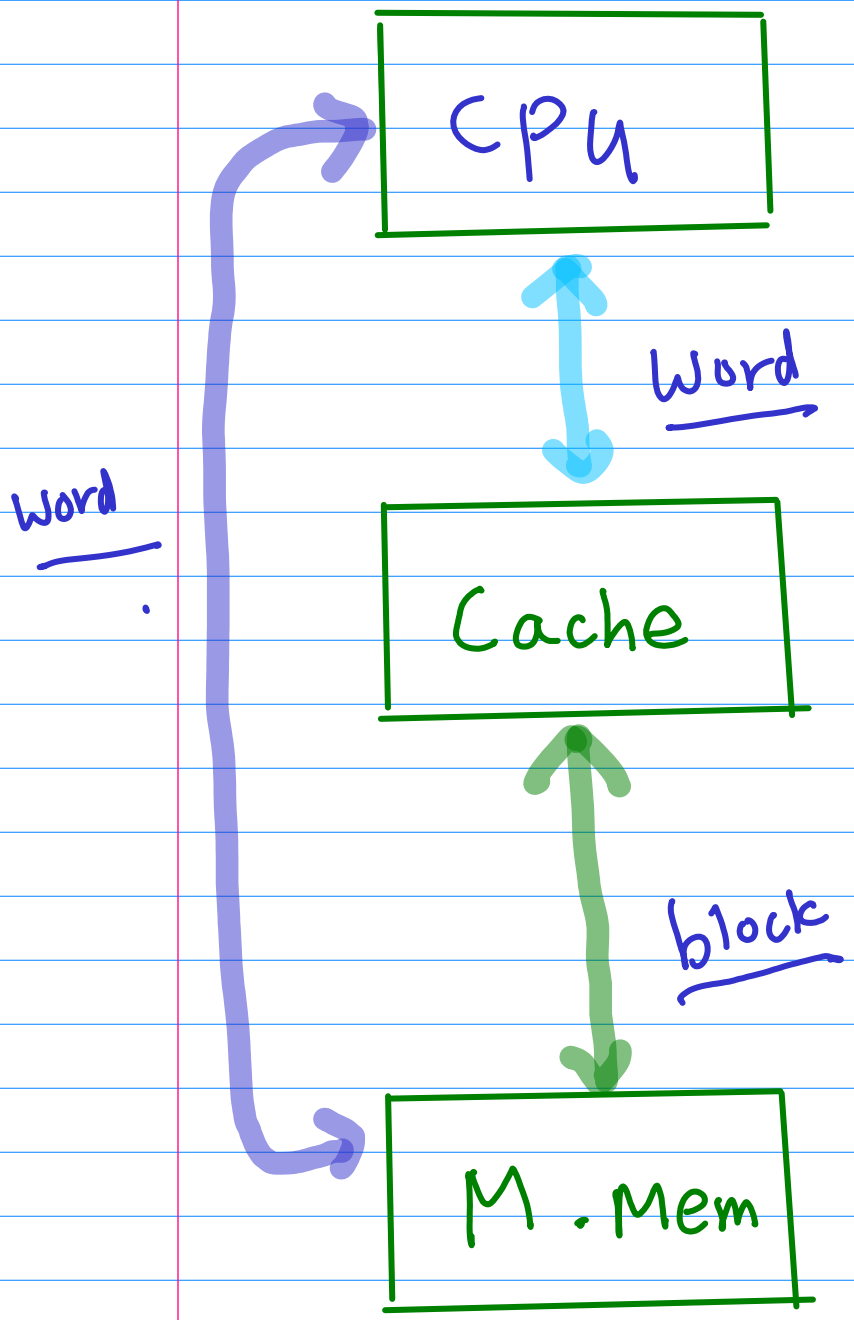


block index

1 block은 2^1 words 1-bit을 갖게 됨

2^1 words/block.



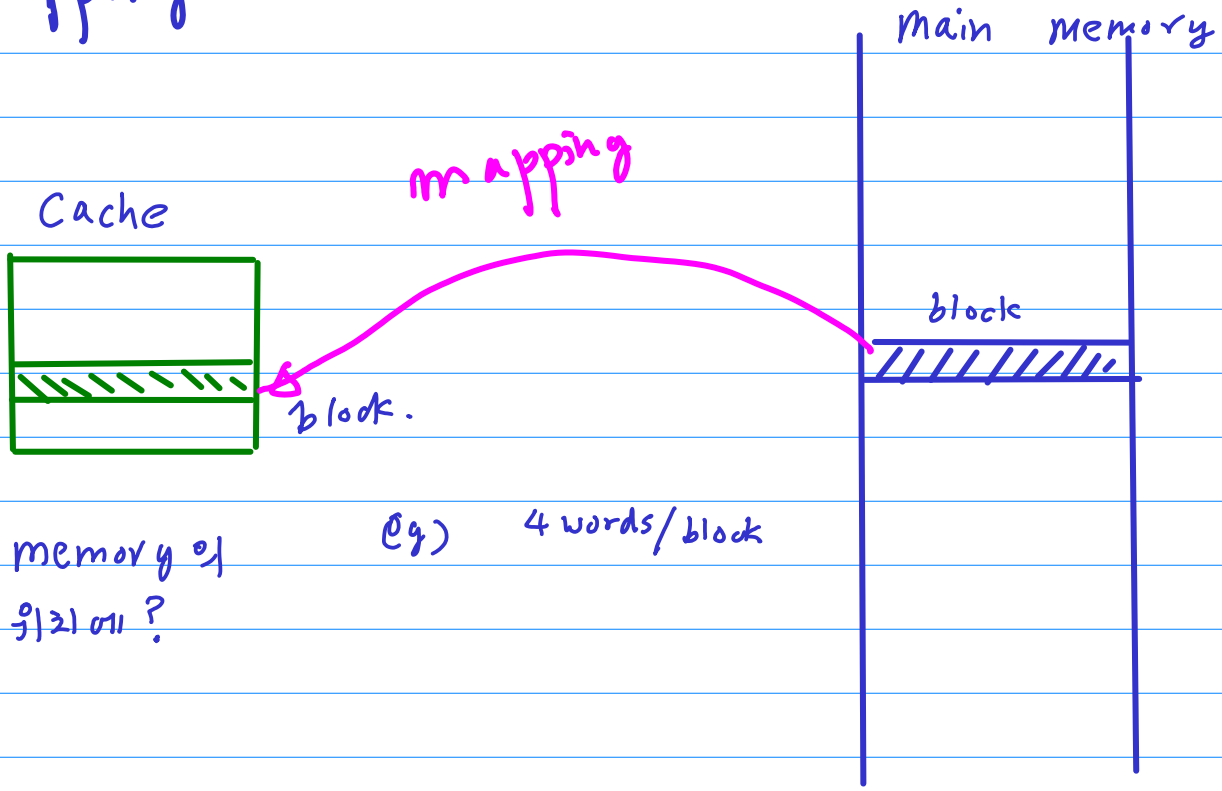

CPU
Word
Word
Cache
block
M. Mem

Cache Mapping Type

① Fully Associative Mapping
② Direct Mapping
③ k-way Set-Associative Mapping

① 완전 연관 사상
② 직접 사상
③ 세트 연관 사상

Mapping



Cache memory의 어느 위치에?

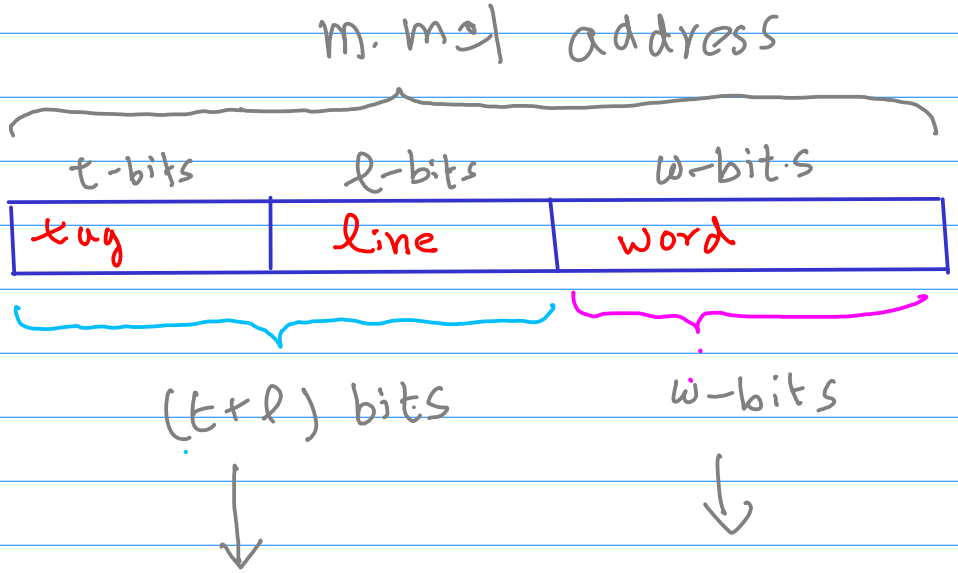
(eg) 4 words/block

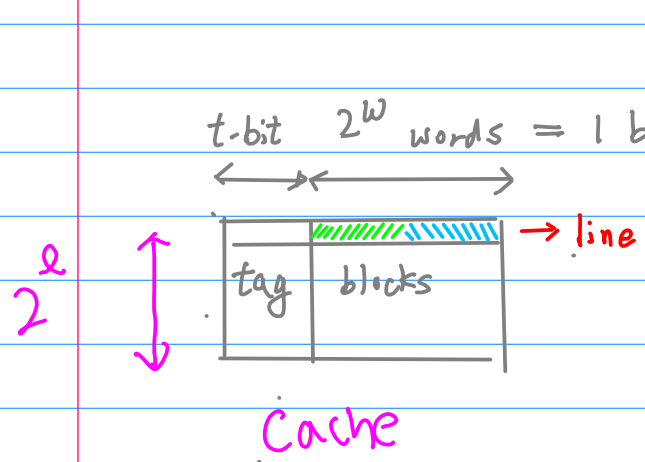
Direct Mapped Cache

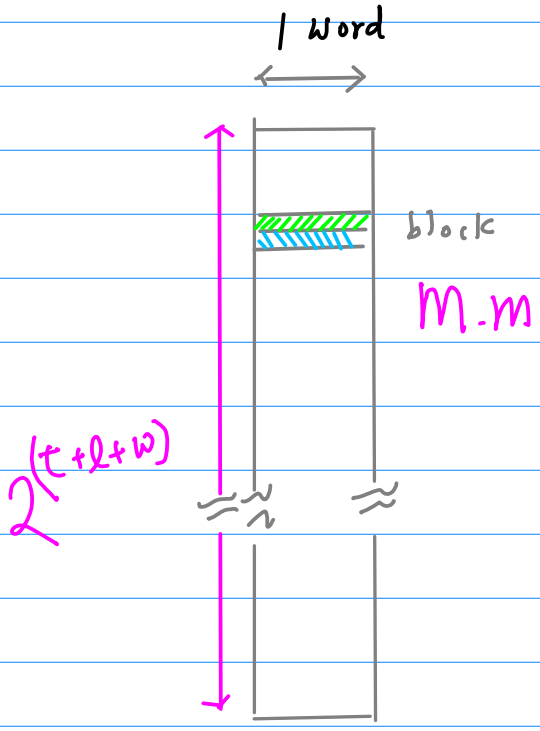
m.m 의 blocks들이

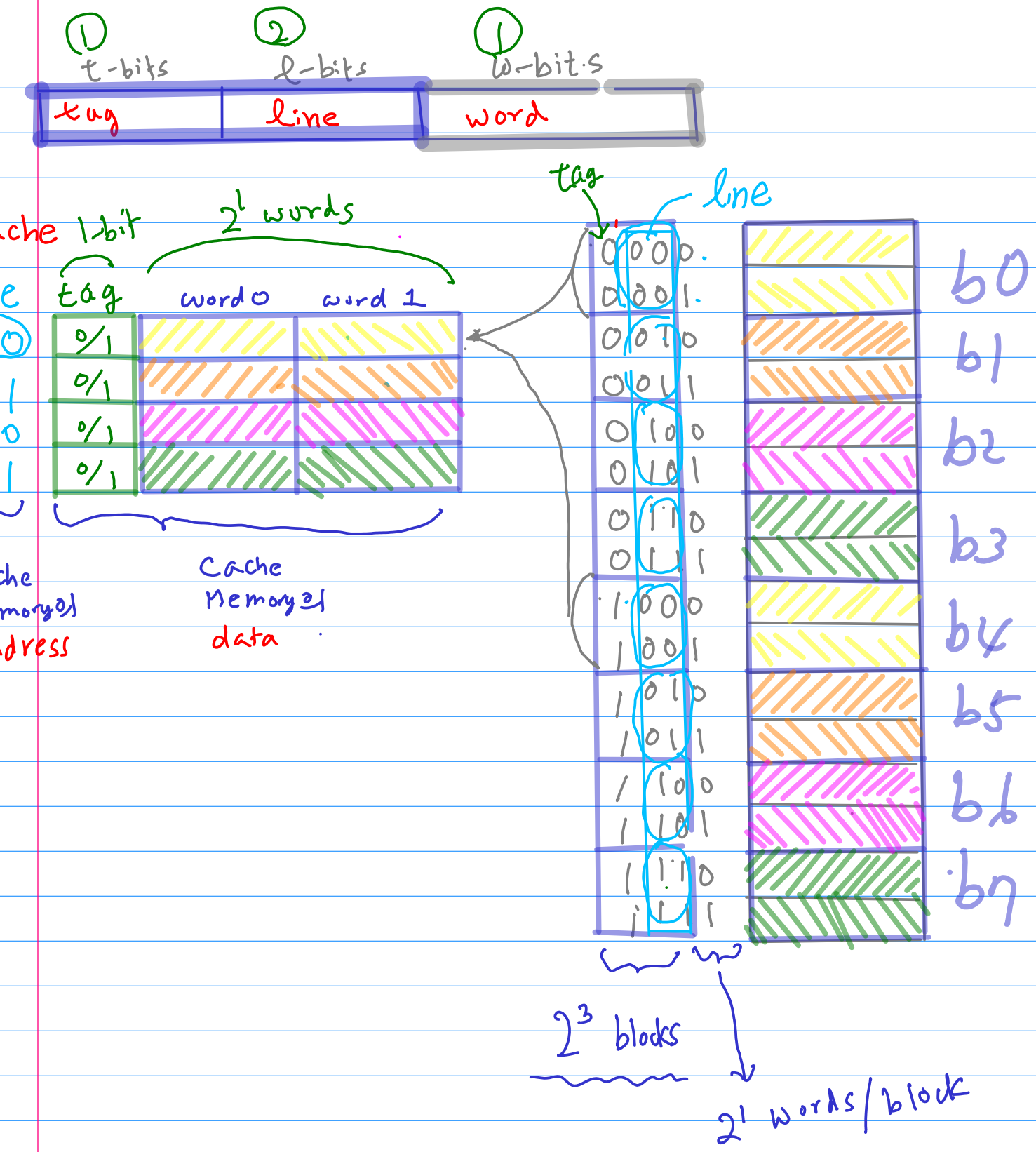
지정된 어느 한 cache line으로만 mapping

각 block들은 지정된 한개의 특정 line이 있다

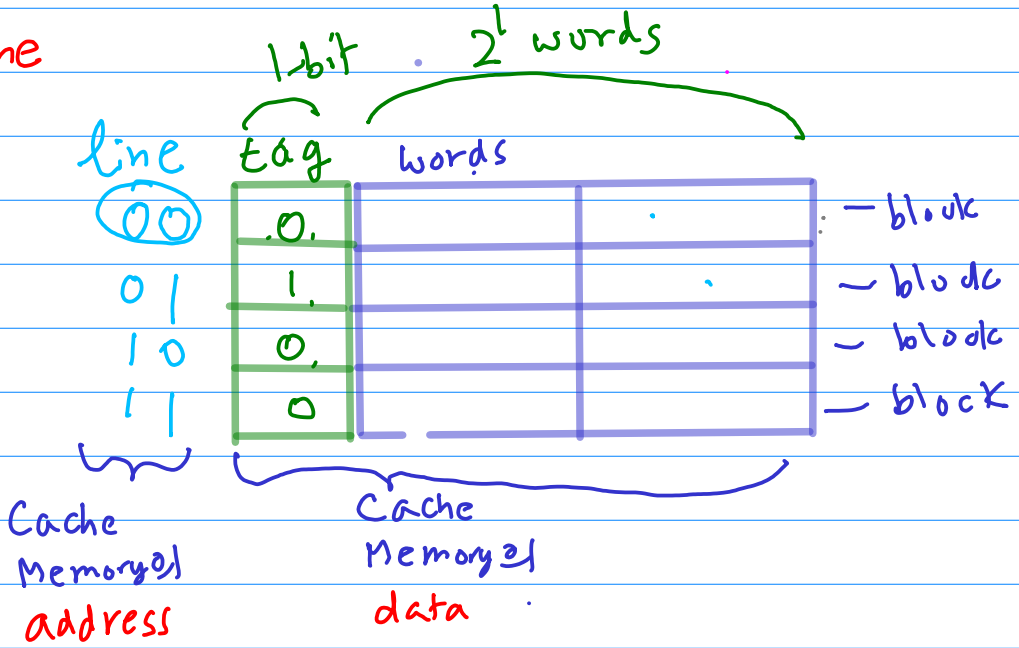


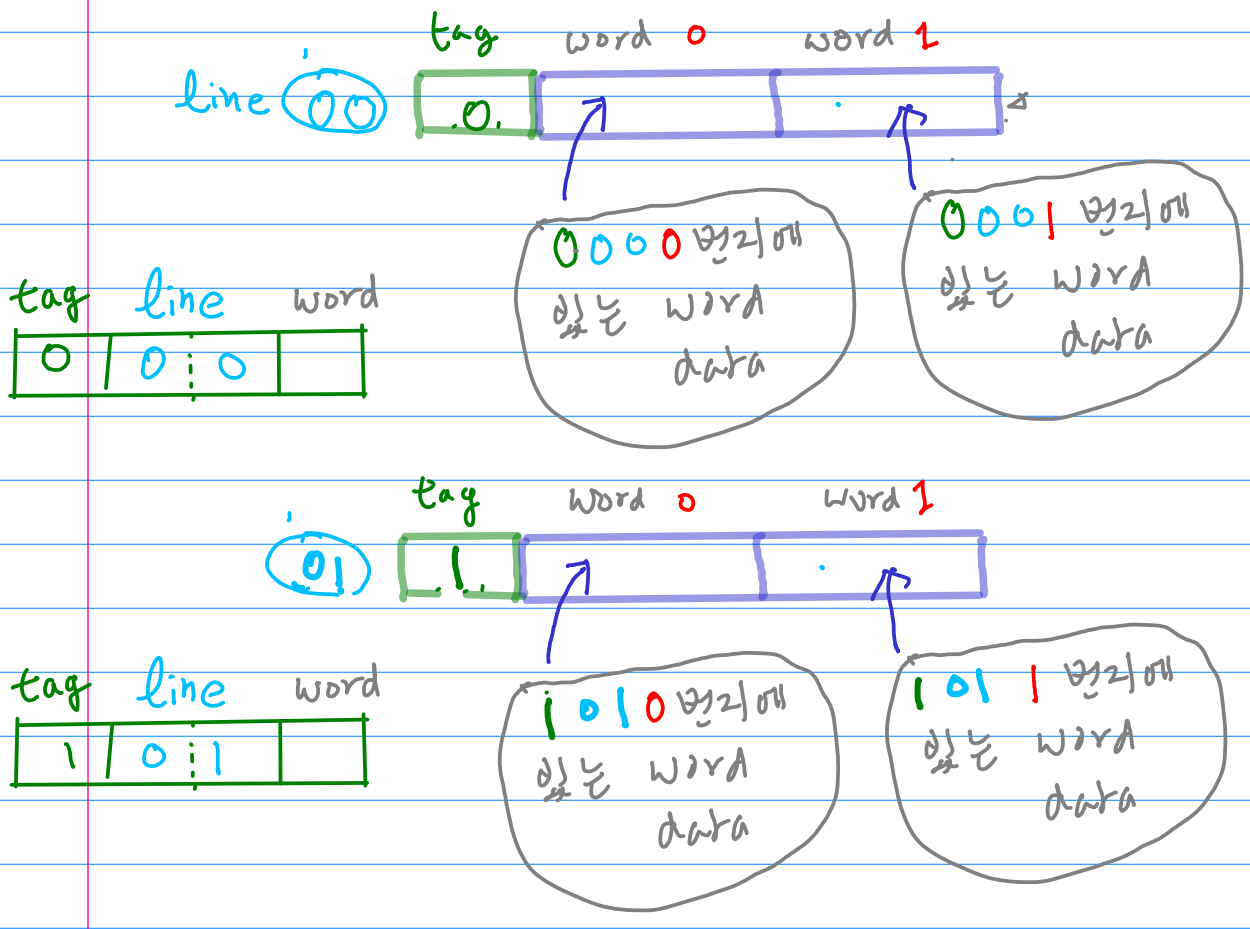


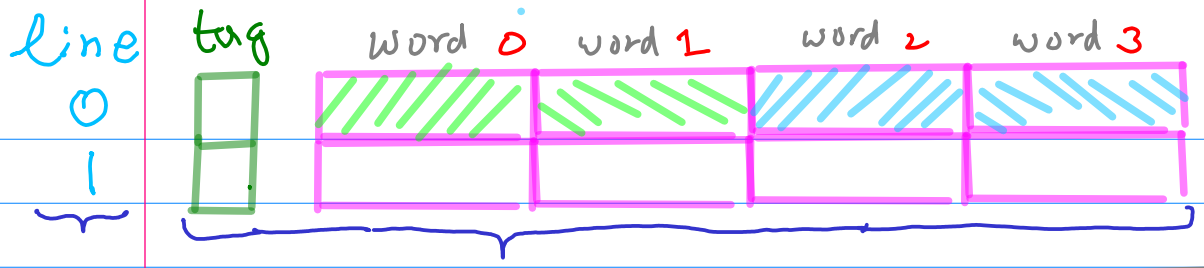



① t-bits
② l-bits
① w-bits
tag
line
word
* Cache 1 bit
2^1 words
line
tag
word 0
word 1
00
01
10
11
0/1
Cache Memory address
Cache Memory data
tag
line
b0
b1
b2
b3
b4
b5
b6
b7
2^3 blocks
2^1 words/block

* Cache

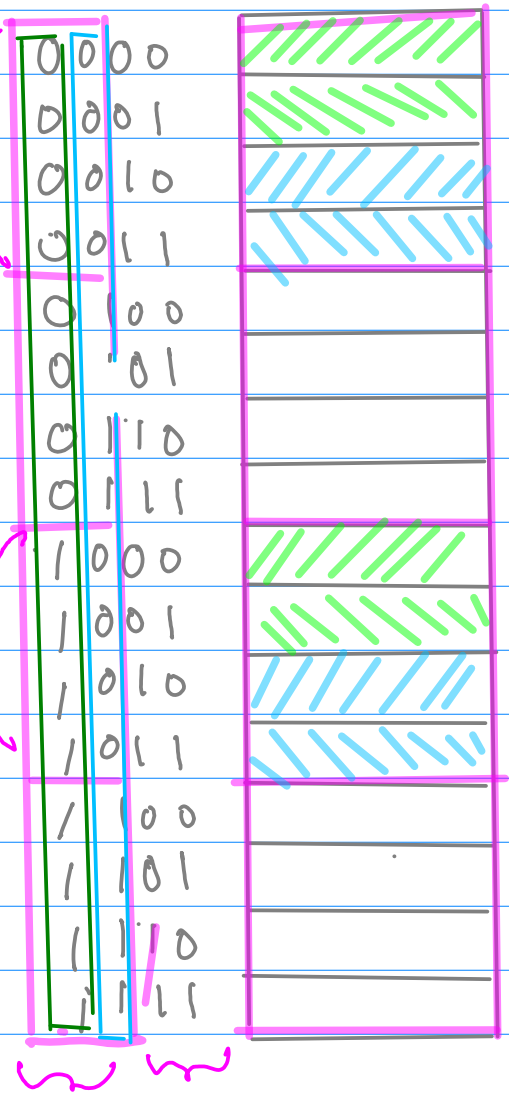

1bit
2^1 words
line
tag
words
00
01
10
11
0
1
0
0
block
block
block
block
Cache Memory의 address
Cache Memory의 data


tag
word 0
word 1
line
00
0
0000 번지에 있는 word data
0001 번지에 있는 word data
tag
line
word
0
0
0
01
1
1010 번지에 있는 word data
1011 번지에 있는 word data
tag
line
word
1
0
1



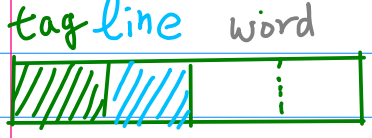
Cache Memory의 address

Cache Memory의 data



같은 line에 mapping

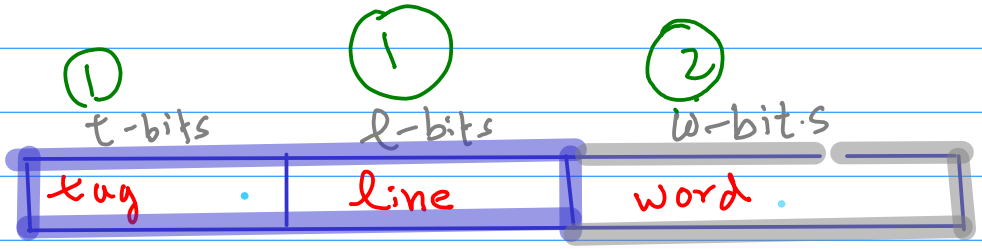
2개씩 1개만 load 가능



어떤 block이 load 되는지를 나타내기 위하여 메모리 주소의 tag 필드를 cache memory의 data의 일부분으로 저장

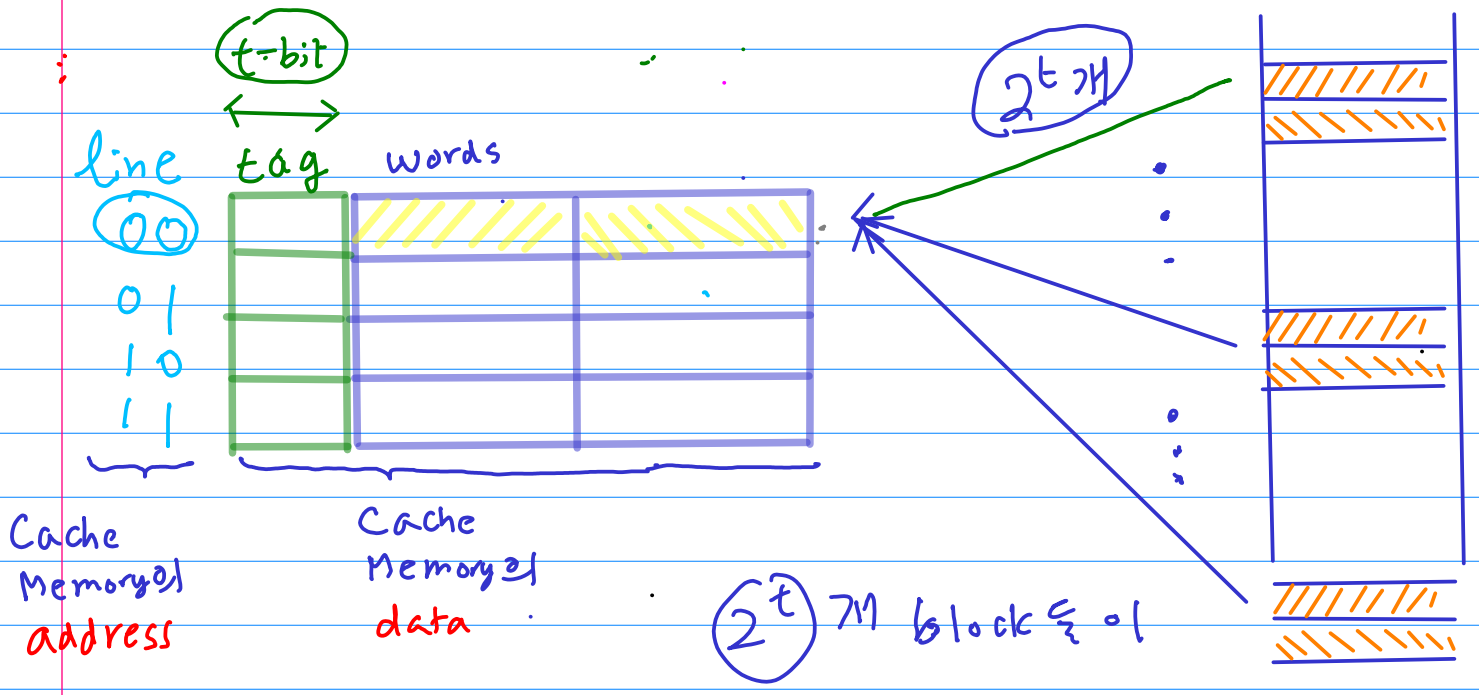
2^2 blocks

2^2 words / block

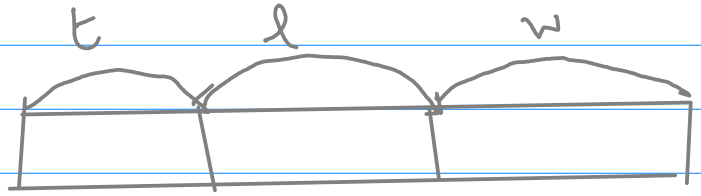


Direct Mapped Cache

각 cache line은 2^t block의 의해 공유된다

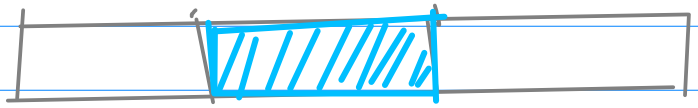


같은 line의 mapping



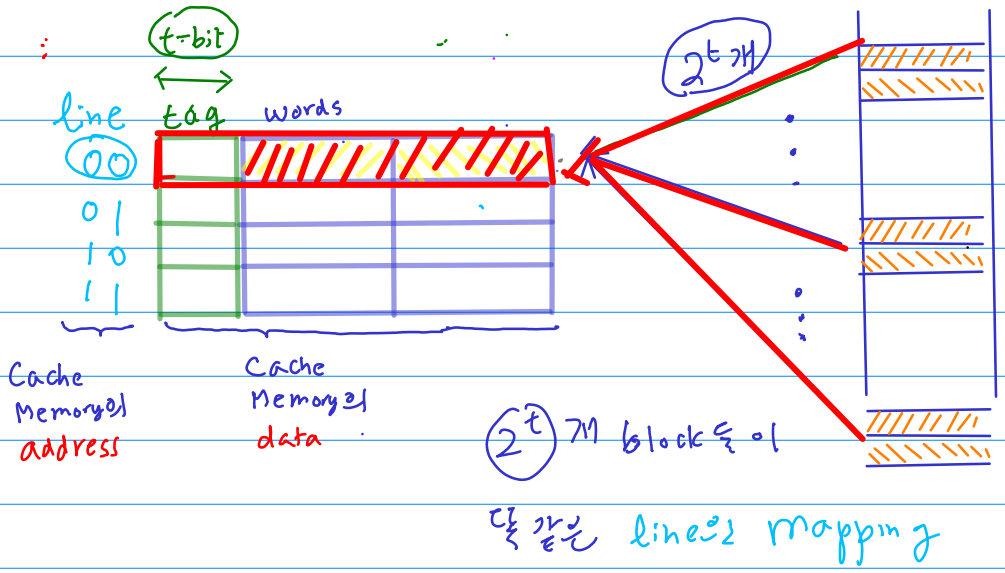
2^w words / block

2^l 개 line

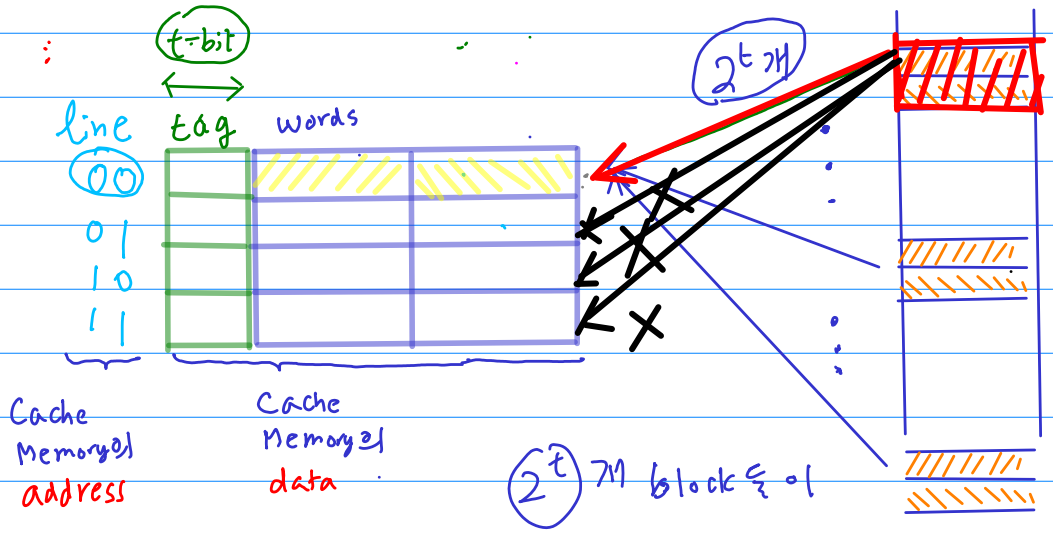


* 같은 line에는 2^t개 block이 올 수 있음.

* 1개의 block은 정해진 1개의 line에만 load 가능



* 1개의 line에는 서로 다른 2^t개 block이 올 수 있음.



* 1개의 block은 정해진 1개의 line에만 load 가능

어떤 block이 지정된 line에 load 되었을 때

그 line에 load될 수 있는 다른 block들과 구별하기 위해서

t-bit tag를 cache memory의 data로서 저장한다.

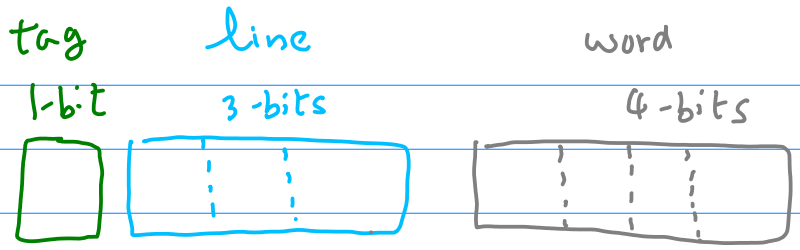
tag가 1-bit 이면 2^1개 block들이 같은 line에 ...

2-bit 2^2개 block들이 같은 line에 ...

즉 cache line은

{ t-bit tag
  2^w개 word로 구성된 block을

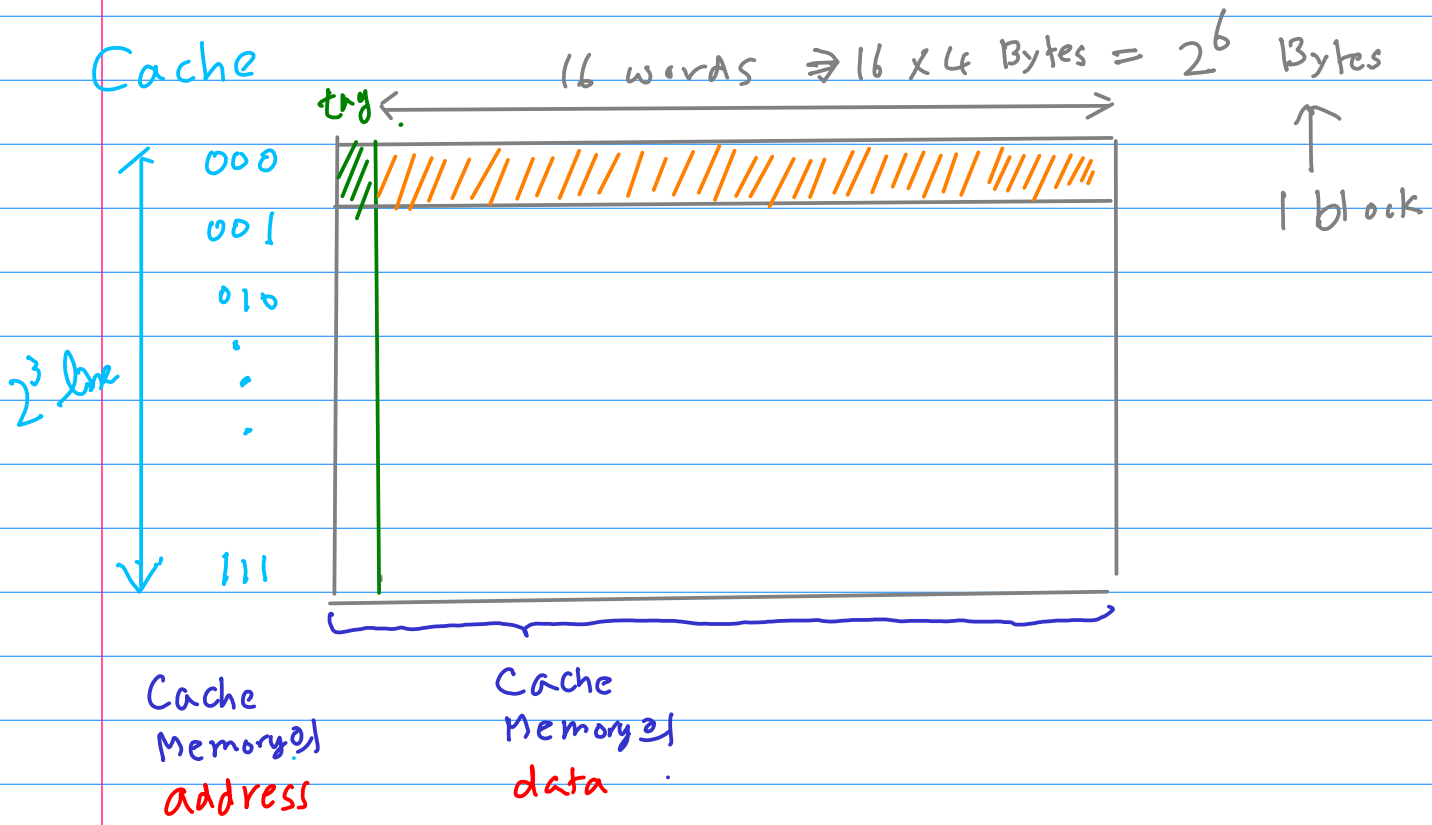
cache memory의 data로 저장한다

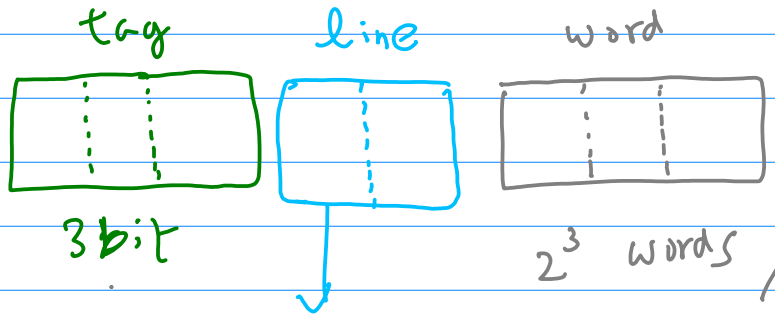


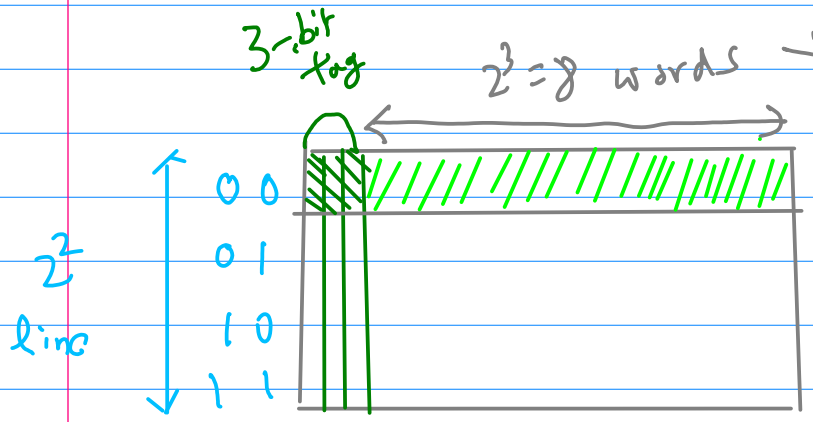
2^3 line

blocksize : 2^4 words / 1 block

4 bytes / 1 word

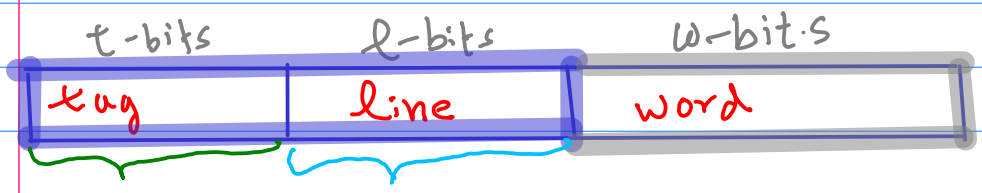






8×4 Bytes $= 2^5$ Bytes / 1 block

* M.M address



Cache Memory의 data를 찾을 때.

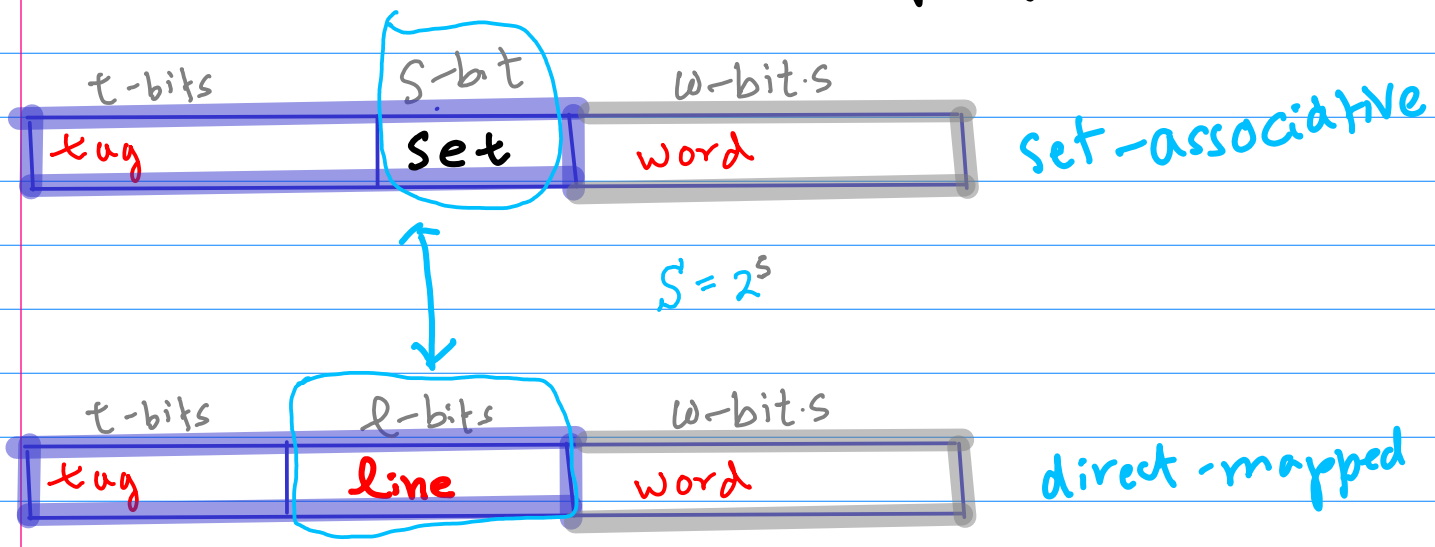
Cache Memory의 address를 사용.

* Cache

2^{ℓ} 개의 line이 있다.

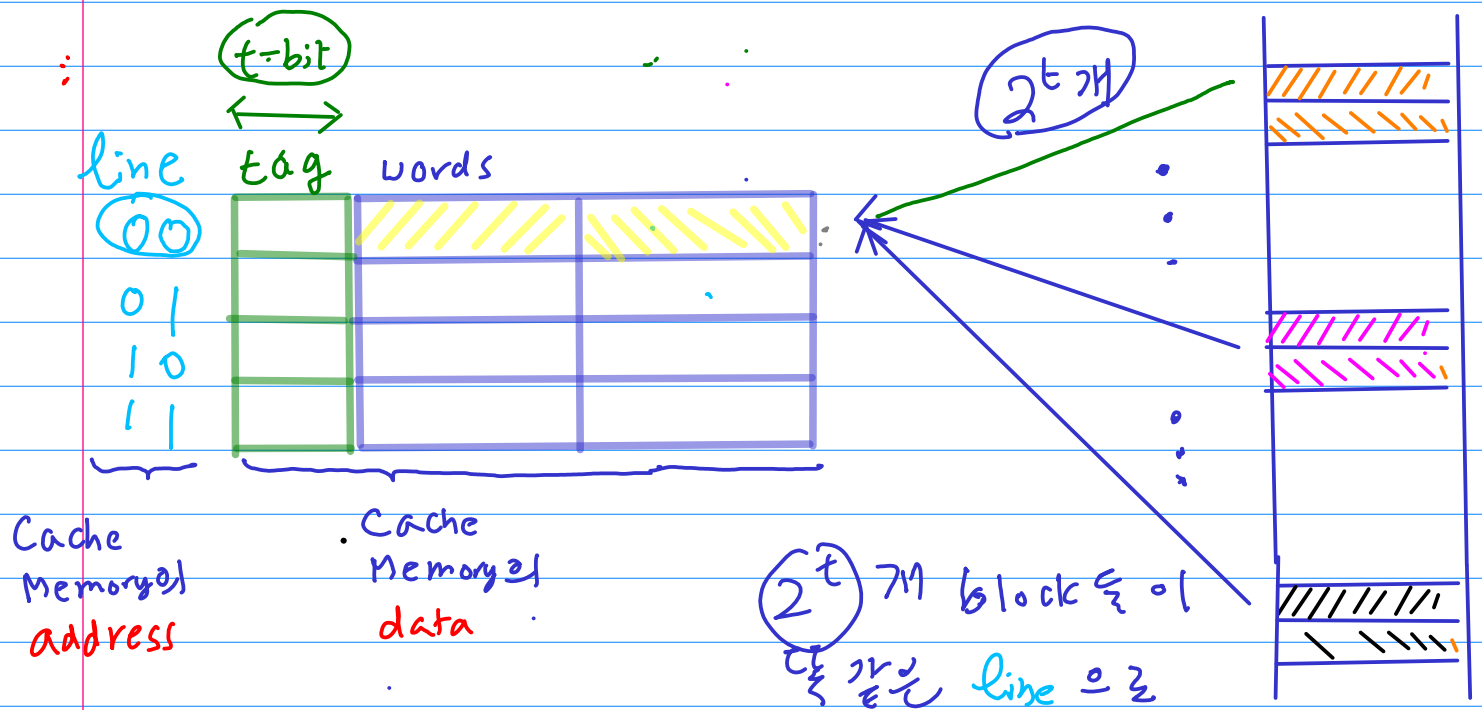
k-way

* Set - associative mapping

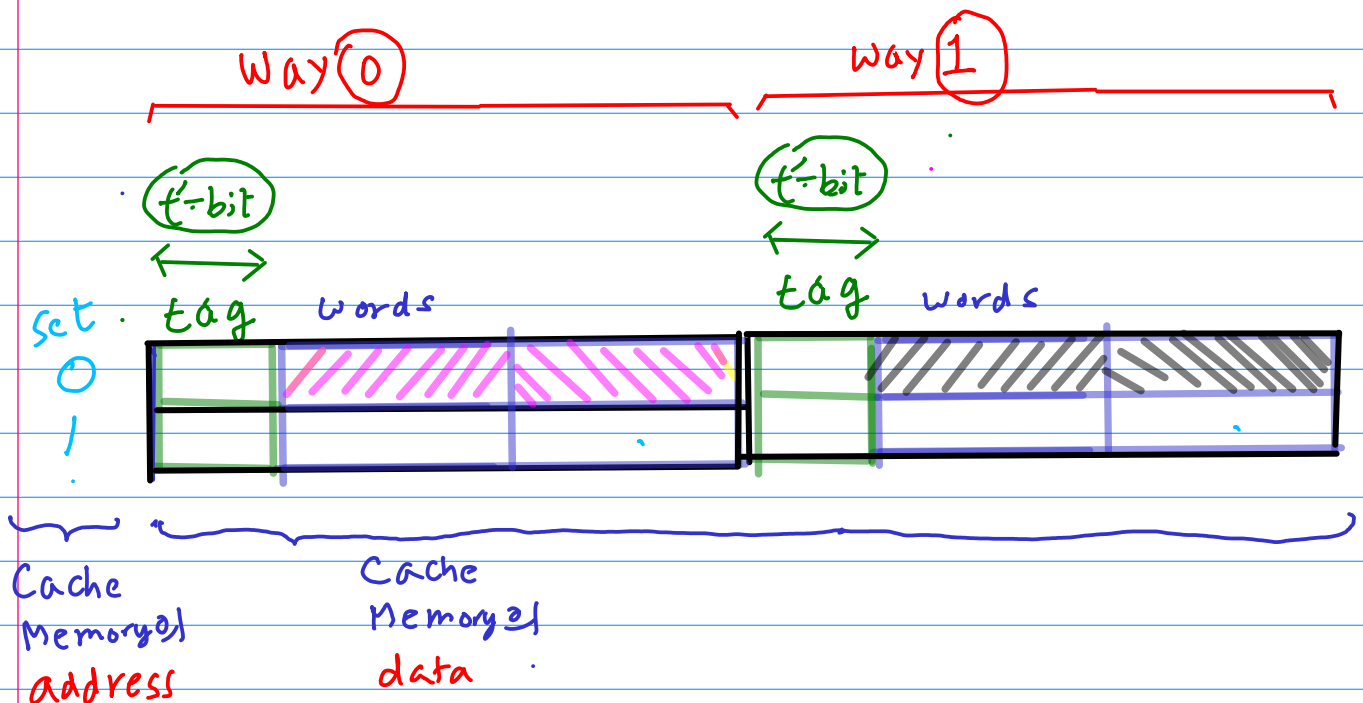


{ direct mapping
Fully-associative mapping } 2가지 방법 ⇒ Set-asrociative mapping

direct mapped 각 line당 1개의 block만 load

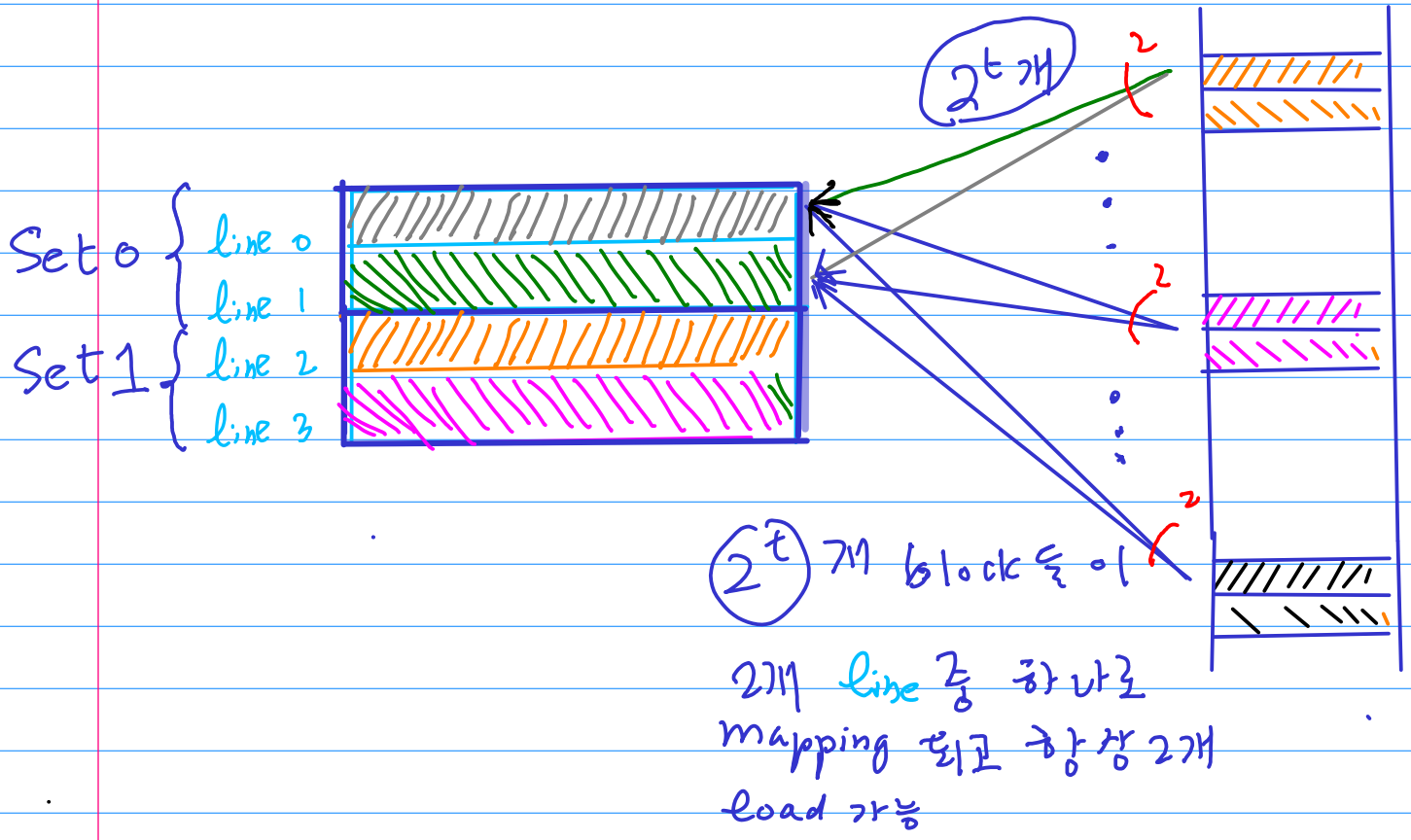


2-way set associative

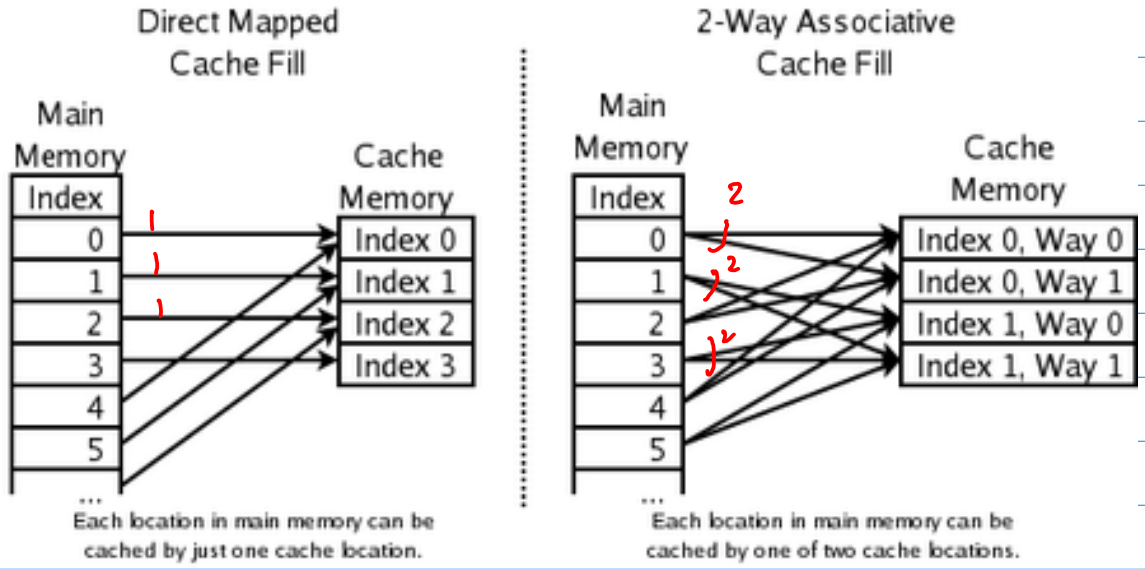


2-way set associative 에서는
각 set당 2개의 block이 load 될 수 있음

2-way set associative Set 당 2개의 block load 가능



2개 line 중 하나로
mapping 되고 한 set 2개
load 가능



http://en.wikipedia.org/wiki/CPU_cache#/media/File:Cache,associative-fill-both.png

원래 cache에

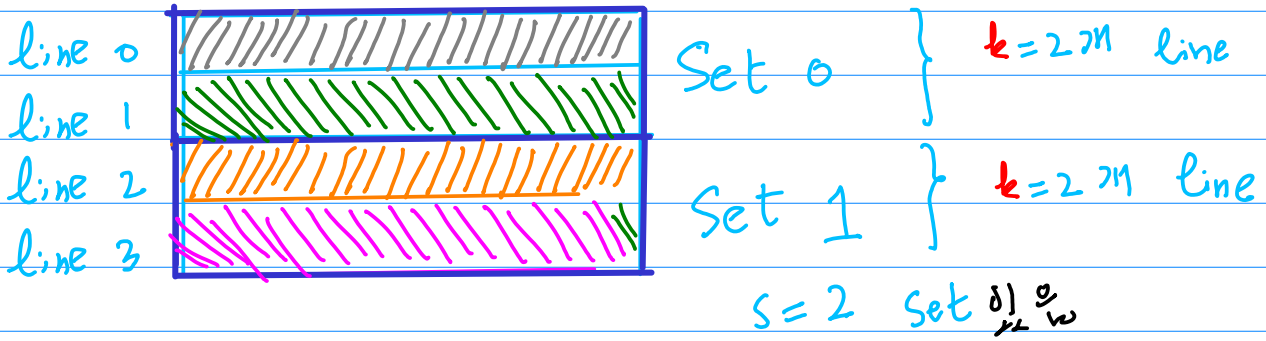
l 개의 line 있다고 가정.

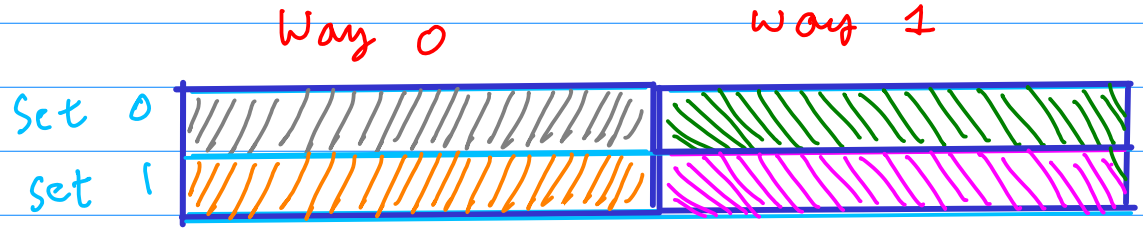
S 개의 set 있음

각 set는 k 개의 line을 저장할 수 있음.

총 $S \cdot k = l$ 개의 line들이 있다.

예제 cache에 $l = 4$ lines 있을 때





2-way set associative mapping

전체 Cache size가 32 bytes
line size는 4 bytes

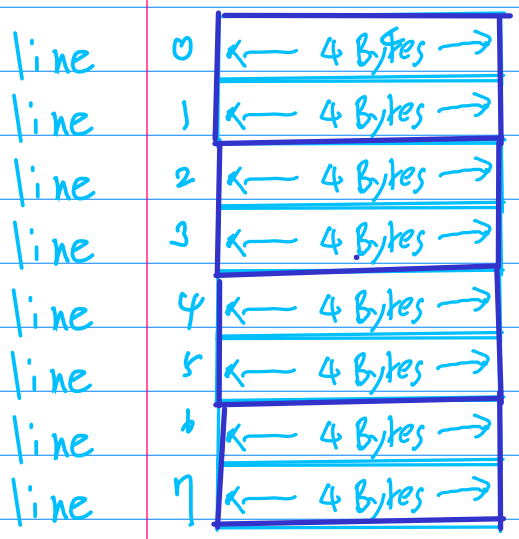
원래 총 32/4 = 8개 line이 있다.

만약 1 set당 2개의 line을 사용하면

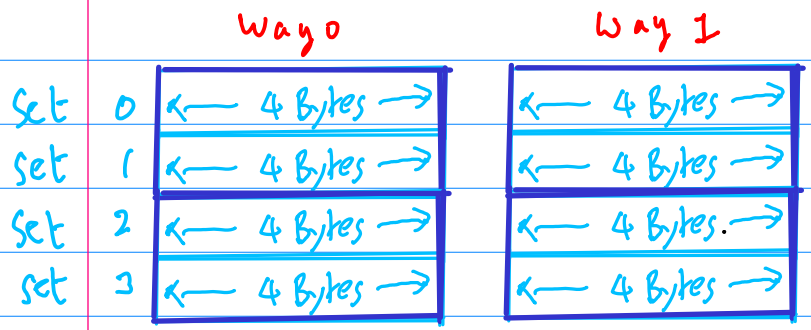
$S \cdot k = 4 \cdot 2 = 8 = \ell$

S = 4 개의 set가 존재

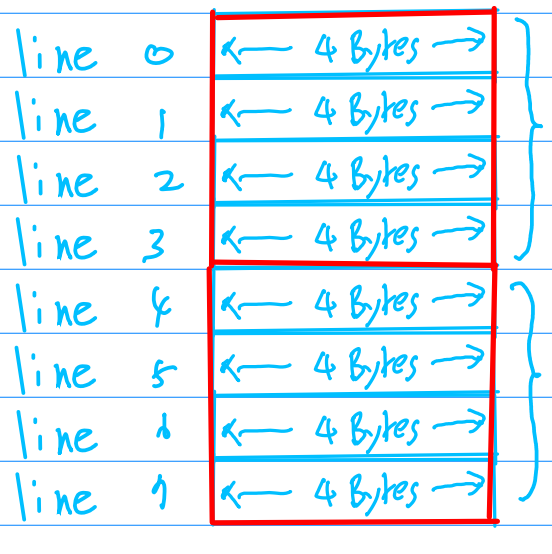
Cache는 S=4개의 set들로 구성되고
각 set는 k=2개의 line이 있다.



2-way set-associative mapping

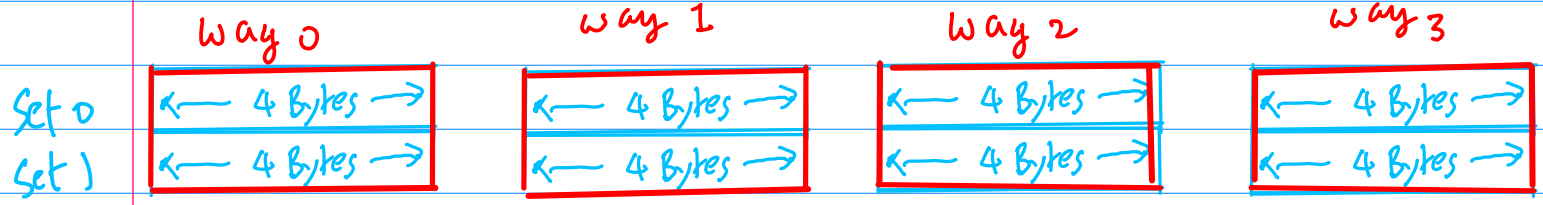


* tag도 block과 함께 cache memory의 data로 저장됨.
같은의 tag 생략



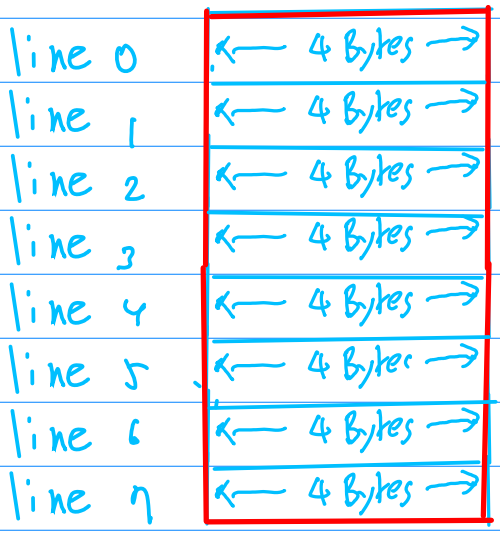
각 set에 k개씩 line이 있으면

k-way set-associative mapping



* tag도 block과 함께 cache memory의 data로 저장됨.
값에서 tag 생략

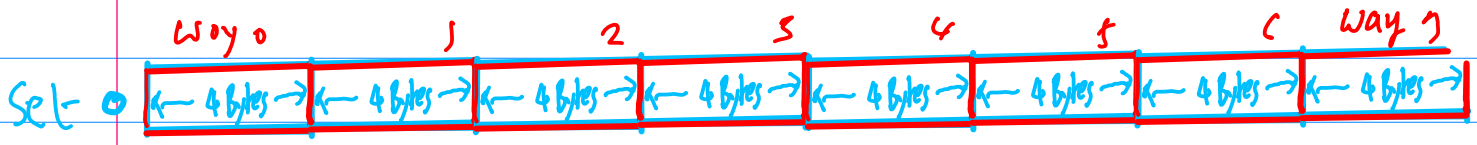
Fully Associative Mapping



set 0

only 1 set

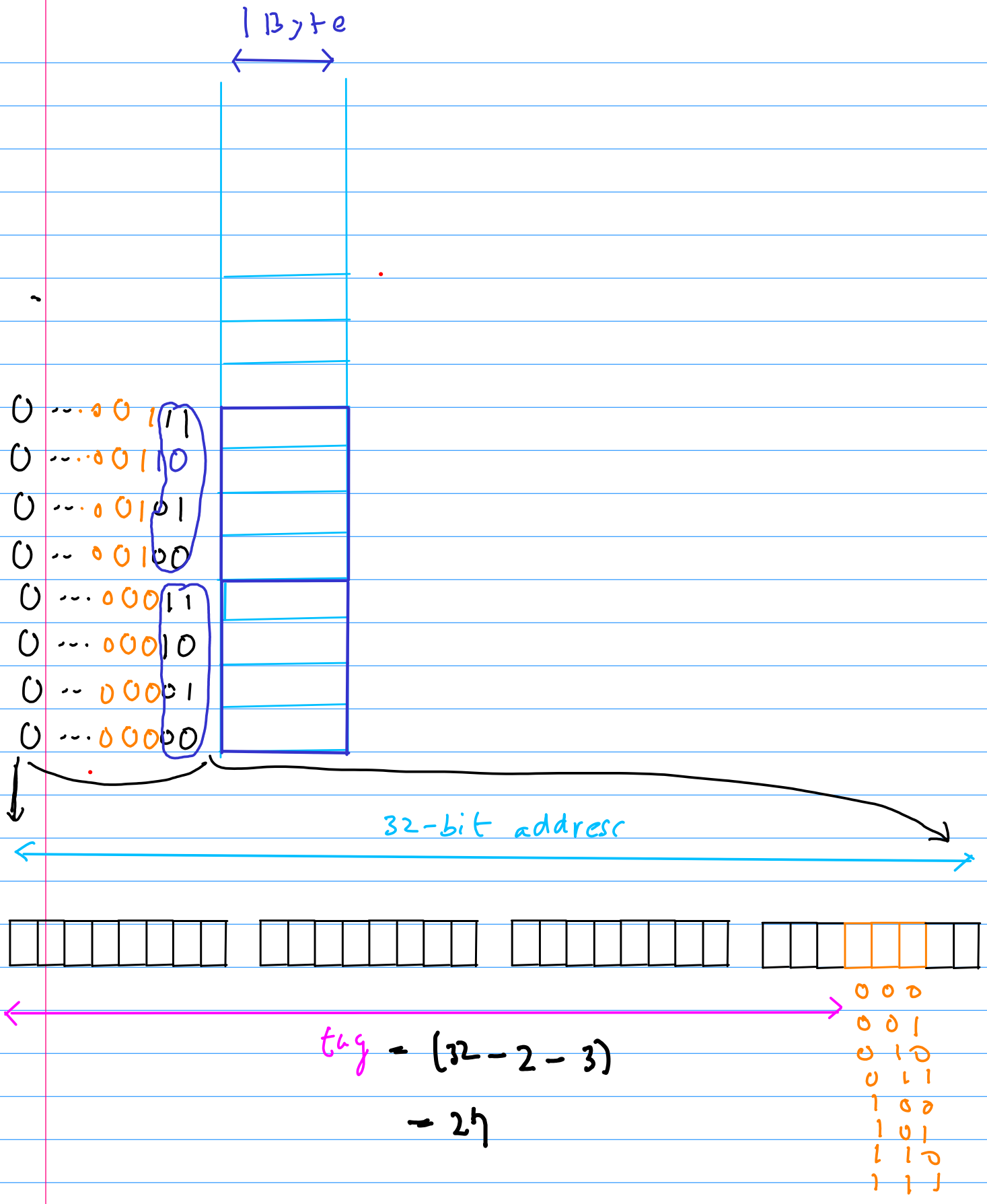
k = l lines



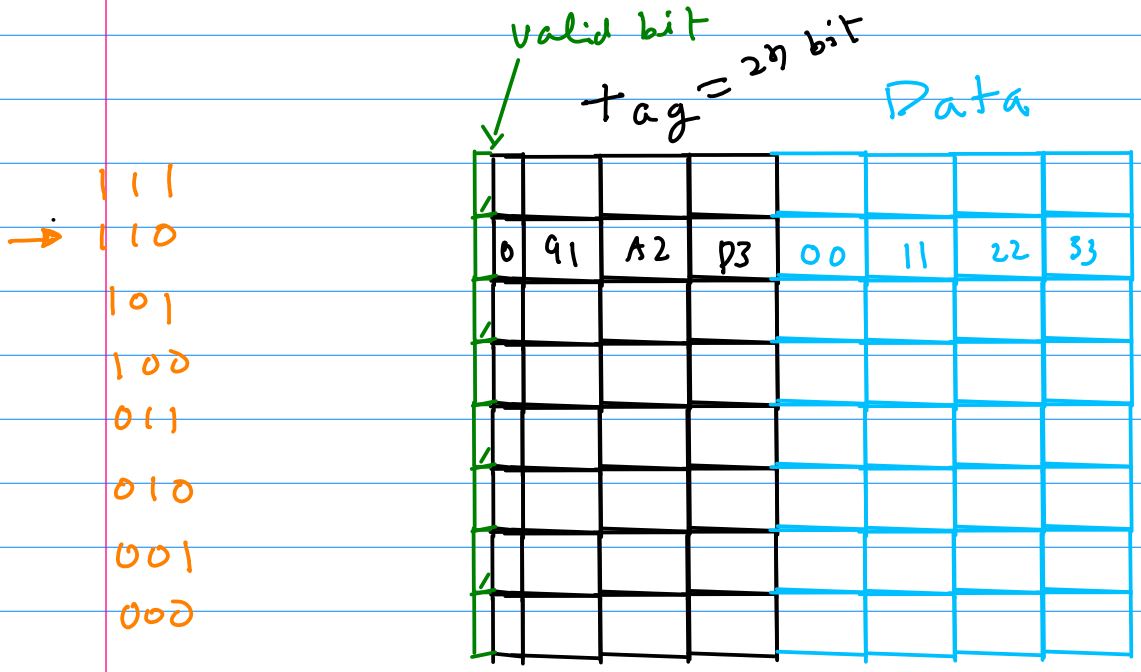
* Fully associative mapping에서는 tag과 address에서 하위 w-bit를 제외한 전체가 된

* tag도 block과 함께 cache memory의 data로 저장됨.

같음에서 tag 생략


1 Byte
0 ... 0 0 1 1 1
0 ... 0 0 1 1 0
0 ... 0 0 1 0 1
0 ... 0 0 1 0 0
0 ... 0 0 0 1 1
0 ... 0 0 0 1 0
0 ... 0 0 0 0 1
0 ... 0 0 0 0 0
32-bit address
tag - (32-2-3)
- 27
0 0 0
0 0 1
0 1 0
0 1 1
1 0 0
1 0 1
1 1 0
1 1 1

Direct Mapping



a 32-bit address

data

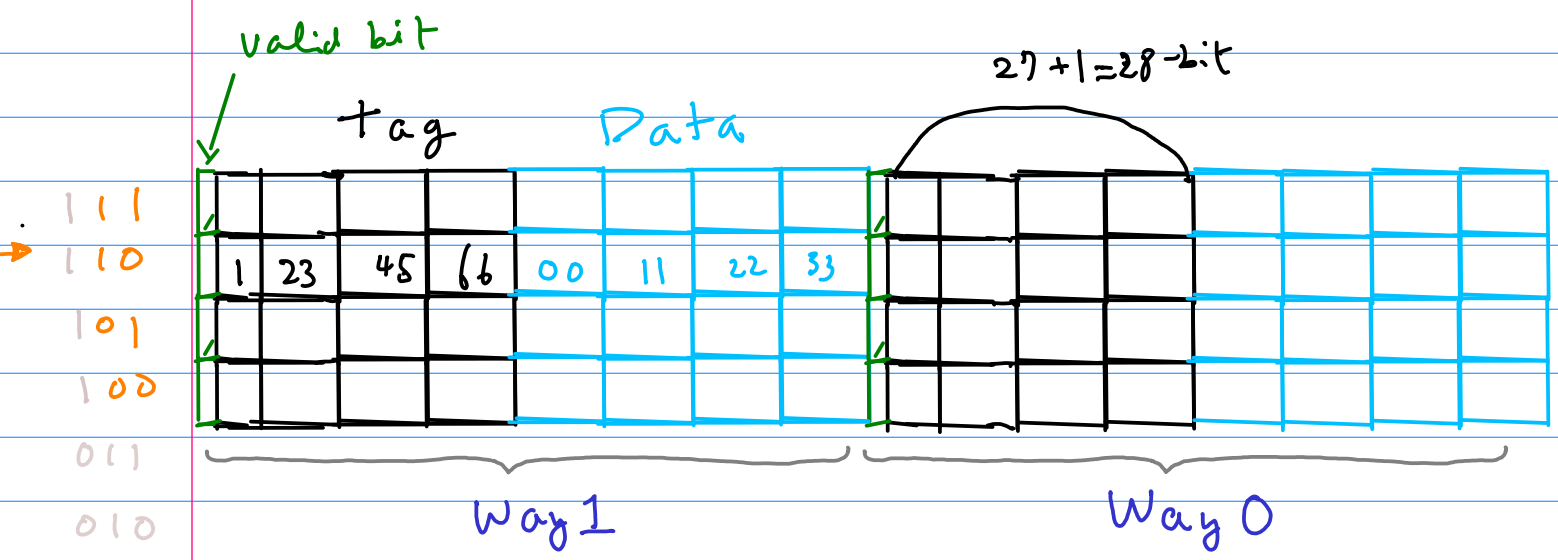
0X 12 34 56 78 00 11 22 33

0111 1000

0001 0010 0011 0100 0101 0111 0011

0 9 1 A 2 D 3

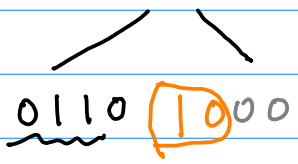
2-Way Set associative mapping

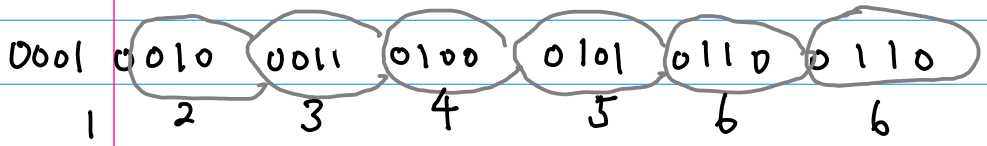


a 32-bit address

data

0x 12 34 56 78 00 11 22 33





Direct

tag 29-bit

line 3-bit

word 2-bit

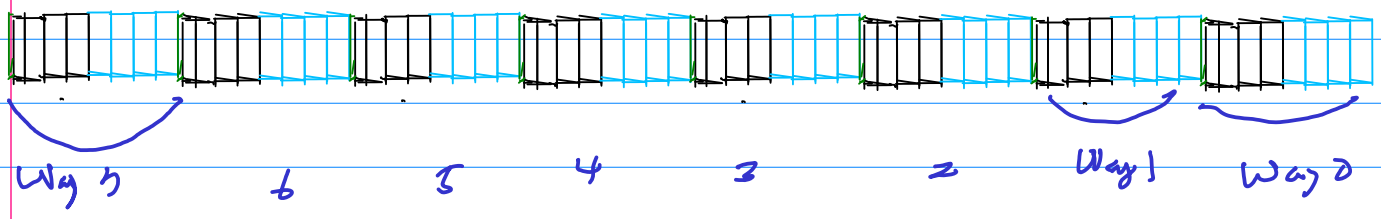
2-way

tag 28-bit

set 2-bit

word 2-bit

Fully Associative



a 32-bit address

data

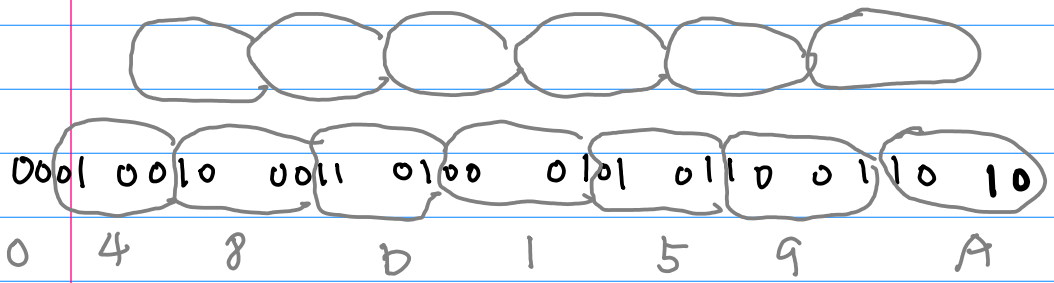
0x 12 34 56 78

00 11 22 33

27-bit tag 0110 1000 Direct Mapping

28-bit tag 0110 1000 2-Way Set-Assoc

32-bit tag 0110 1000 Fully Assoc

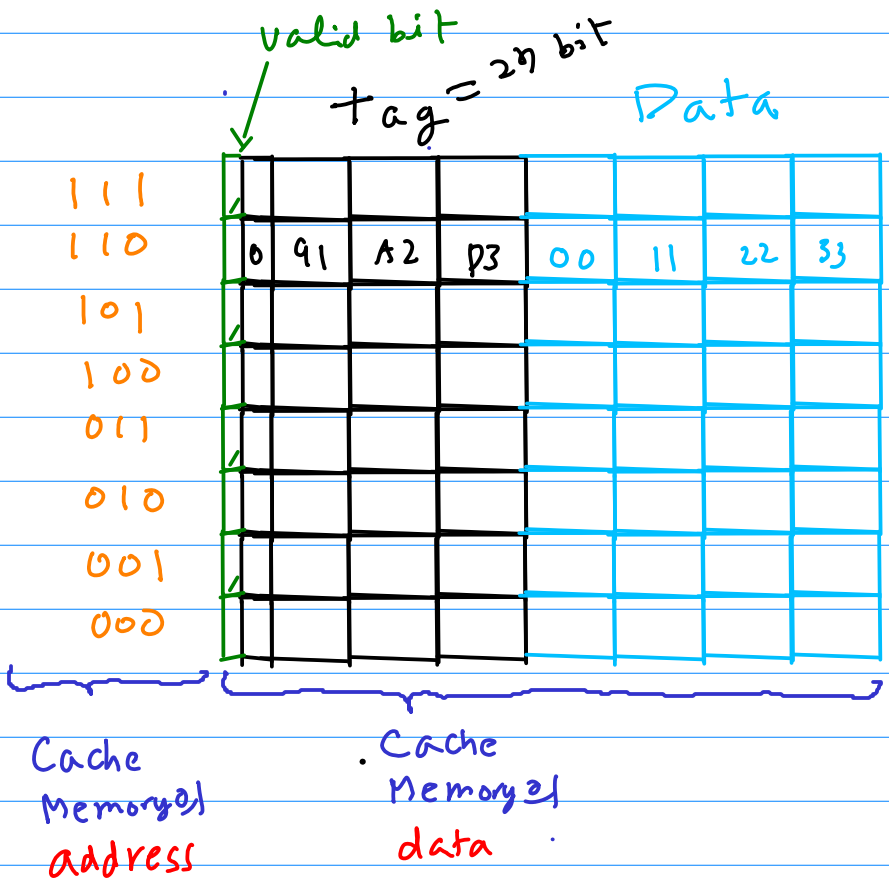


27+1+2 = (30)

Address Matching Logic

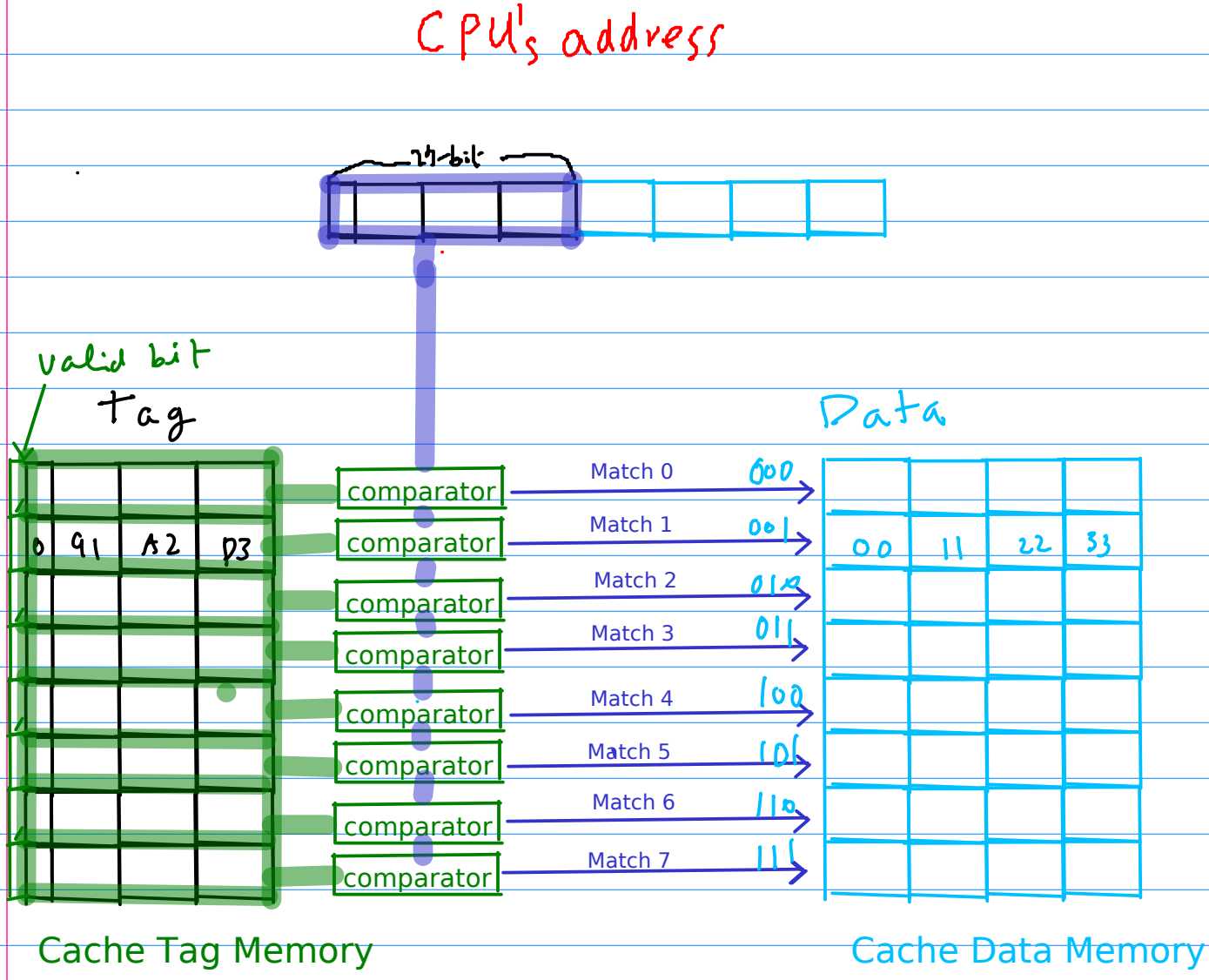
no address decoding ✗
instead, address matching

⇒ Content Addressable Memory (CAM)
Associative Memory



one of these are selected
not by address decoding logic
but by address matching logic

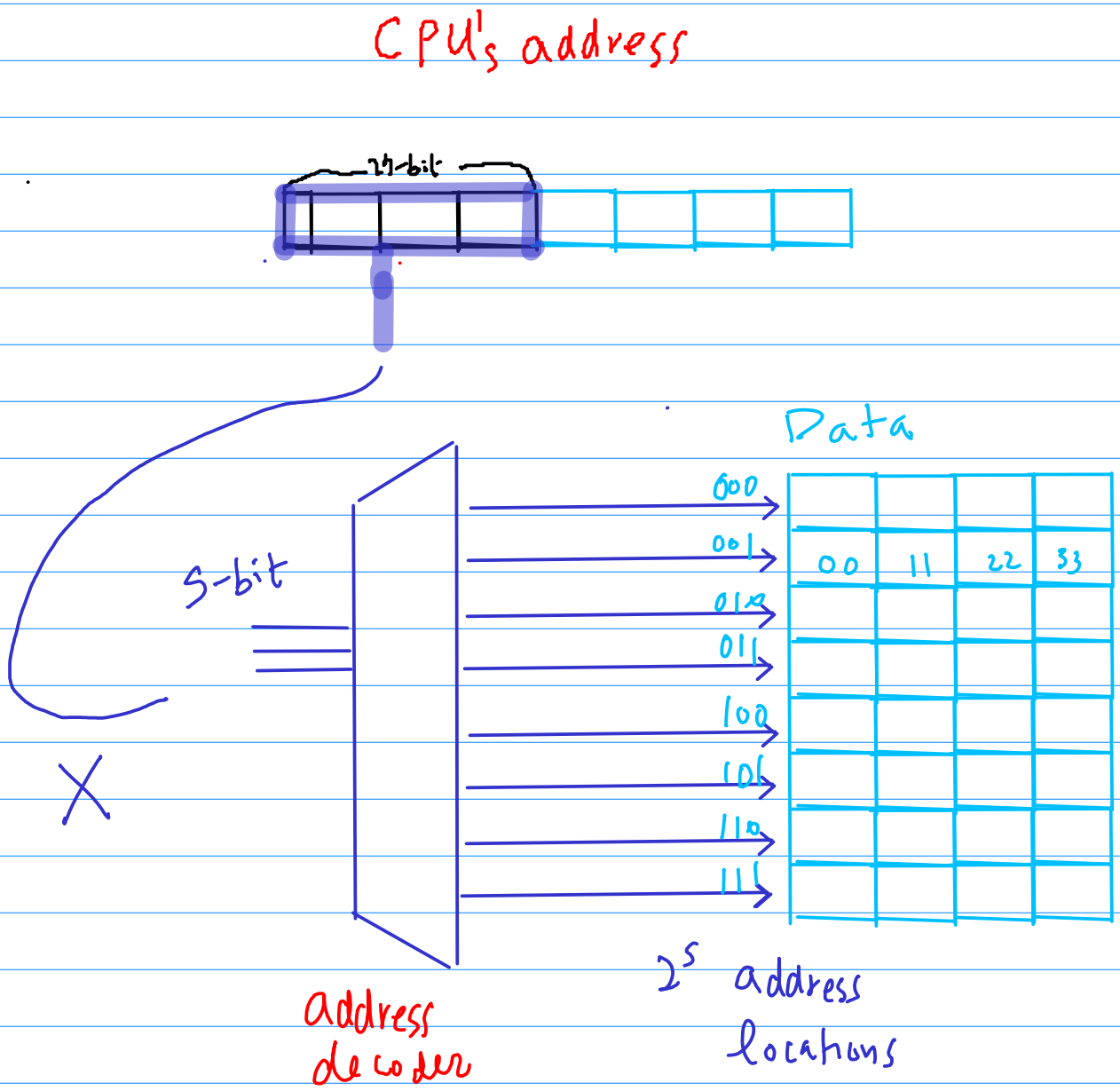
Address Matching



the tag field of a CPU address is matched with each content of tag memories.

Content addressable

Address Decoder



In cache memories, address decoding is not used, address matching is used.

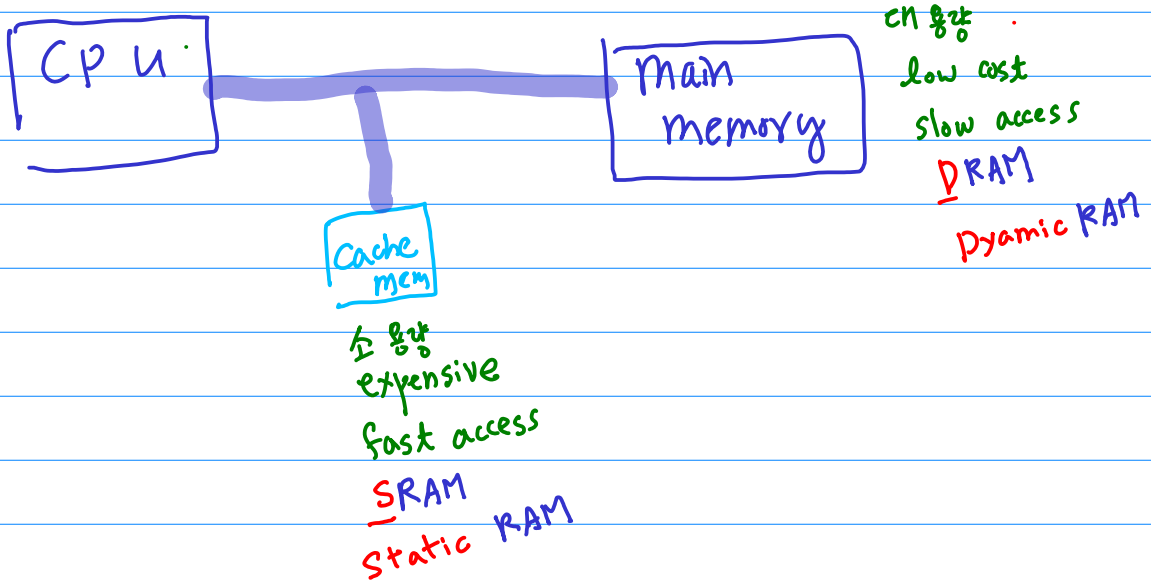
Cache Memory

Speed of CPU (빠르다) > Speed of Main Memory

instruction 수행시간 < memory access time 길다

CPU가 연산결과를 memory에 쓰기 위해서 CPU가 idle하면서 기다리는 상황이 있을수 있다 → performance 성능저하

CPU와 main memory 간의 속도차이 인한 성능저하를 방지하기 위해서 cache를 사용한다.



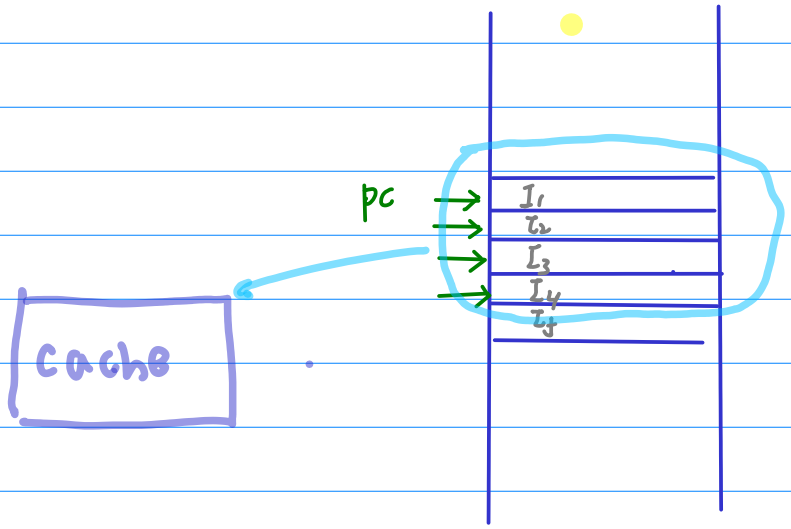
CPU가 Data를 Read 하는 경우

먼저 Cache를 찾아보고

- 원하는 data가 cache에 있으면 (Cache hit)
  cache에 있는 data를 읽어온다
  그럼 main memory access 없이
- 원하는 data가 cache에 없는 경우 (Cache miss)
  main memory를 access해서
  CPU가 읽어오려고 했던 data 뿐만 아니라
  그 주변 data block을 cache에 Load 한다

Locality (지역성)

CPU가 main memory의 특정 부분 (지역)에 위치한
program code나 data를 반복적/국지/집중적으로
access 하는 현상을 의미



temporal locality (시간적 지역성)
spatial locality (공간적 지역성)
sequential locality (순차적 지역성)

temporal locality (시간적 지역성)

최근에 access한 data가 가까운 미래에
다시 access될 가능성이 높다.

for loop, subroutine, common variables.

Spatial locality (공간적 지역성)

memory에서 서로 인접되어 있는 data들이
연속적으로 access될 가능성이 많다.

table, array

⭐ **Sequential locality** (순차적 지역성)

branch가 발생하지 않는 한
명령어들은 기억장치에 저장된 순서대로 인출되어 실행된다.

program이 수행되는 동안에
이런 지역성으로 인하여
cache에 이미 load된 data를
CPU가 access하는 경우가 많다.
(cache hit ratio 증가)

→ 평균 memory access time을 단축시킬 수 있다.

cache design 시 목표.

- **Cache hit ratio를 극대화.**
  CPU가 원하는 data가 cache에 존재할 확률을 높여야 한다.
- **cache access time을 최소화.**
  Cache hit인 경우 CPU가 cache로 부터 읽어오는 시간을 줄인다.
- Cache miss인 경우 **delay time을 최소화**
  main memory로 부터 cache로 data를 읽어오는 시간을 최소화.
- **main memory와 cache의 data consistency (일관성)** 유지 / **overhead 최소화**
  CPU가 cache의 내용을 변경하였을 때 main memory에 그 내용을 update하는 절차로 인한 delay를 최소화.
  - Write-Through
  - Write-Back

Fetch Policy Cache ← M.M

- demand fetch (Bring in)
- prefetch (Send in)

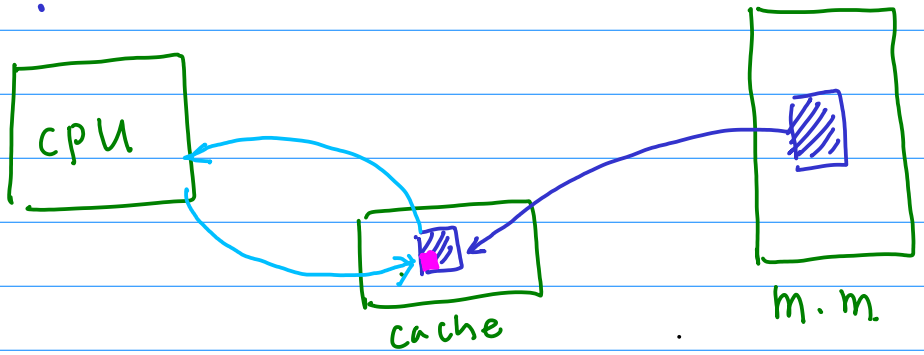
Write Policy Cache → M.M

- Write-Through
- Write-Back.

Replace Policy Cache ⇄

- Least Recently Used
- FIFO (First In First Out)
- Least Frequently Used
- Random

Write Policy (쓰기 정책)



cache에 load된 data ← 변경 ← program 수행

↓ 변경

main memory에 있는 상응하는 data

일관성 Consistency 유지 ..... cache와 m. mem과 같은 data는 같도록 유지.

cache write 시간 < main memory write 시간

Write-Through

- cache가 변경될때 마다 main memory의 내용도 변경하는 방법

Write-Back

- cache가 변경되더라도 main memory를 update하지 않고 변경된 data가 속한 라인, replace 될때 update한다

Modified Bit ..... Cache의 내용이 변경되어서 main memory로 update할 필요가 있음을 나타내는 bit

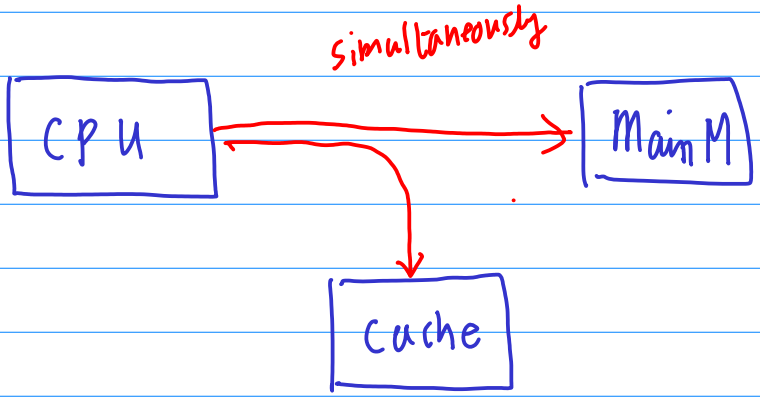
Write-through

Simple.

모든 write operation이
(cache와 main memory에)
동시에 수행된다 (simultaneously)

항상 main memory에 있는 data들은 valid하다 (유효)
consistent한 data.

평균 access time이 W-Back에 비해서 많이 길다.



Write-back policy

어떤 block이 cache에서 replace 되어 cache에서 나갈 때

- 변경된 내용이 없는 경우 (M=0)
  다른 block 으로 replace 하면 된다
- 변경된 내용이 있는 경우 (M=1)
  replace 되는 block을 main memory에 update 한후
  다른 block 으로 replace 하면 된다

Write-Back에서 Modified Bit이 set된 block을 교체할 때 (cache → main. memory)
block 전체를 main-memory write → update

장점: main memory write operation 횟수가 감소

단점: cache에서 수정된 내용이 main memory에서 update 될 때까지 main memory의 해당 block이 무효화 상태 (valid x) data inconsistency
→ control 회로가 복잡하다.

cache miss가 발생하여 원하는 data가 있는 block을 cache에 load 해야되는 replace 되는 block이 변경된 경우는 write-back을 하기 때문에 추가적인 시간이 소요된다.

(replace 되는 block을 m.m 쓰는 시간
+ 새로 block을 m.m에서 cache로 load하는 시간)
= Cache miss 일때 걸리는 시간

Cache hit 의 access time < cache miss access time

Replacement Policy (교체정책)

Cache miss 발생하여 (CPU가 원하는 data가 cache에 없을 때)
새로운 block을 M. mem 에 cache로 load 할 때
. cache에 빈 자리가 없으면 cache에 있는 block을
교체 (replace)해야 한다

LRU (Least Recently Used)
사용되지 않은 채로 가장 오랫동안 load 되어 있던 block을 교체
(가장 오랫동안 access 되지 않은 block)

FIFO (First In First Out)
cache에 load 된지 가장 오래된 block을 교체
(제일 먼저 load 된 block)

LFU (Least Frequently Used)
Cache에 load된 이후 access된 횟수가 가장 작은 block을 교체

Random
임의 block을 교체.

Fetch Policy (인출 방식)

main memory → cache

demand fetch (요구 인출)
필요한 정보만 인출.

prefetch (선 인출)
필요한 정보뿐만 아니라 앞으로 필요할 것으로 예측되는 정보까지 fetch

CPU가 원하는 data를 fetch할 때
그 data와 근방의 data까지 함께 (block)
fetch해서 cache에 load한다.