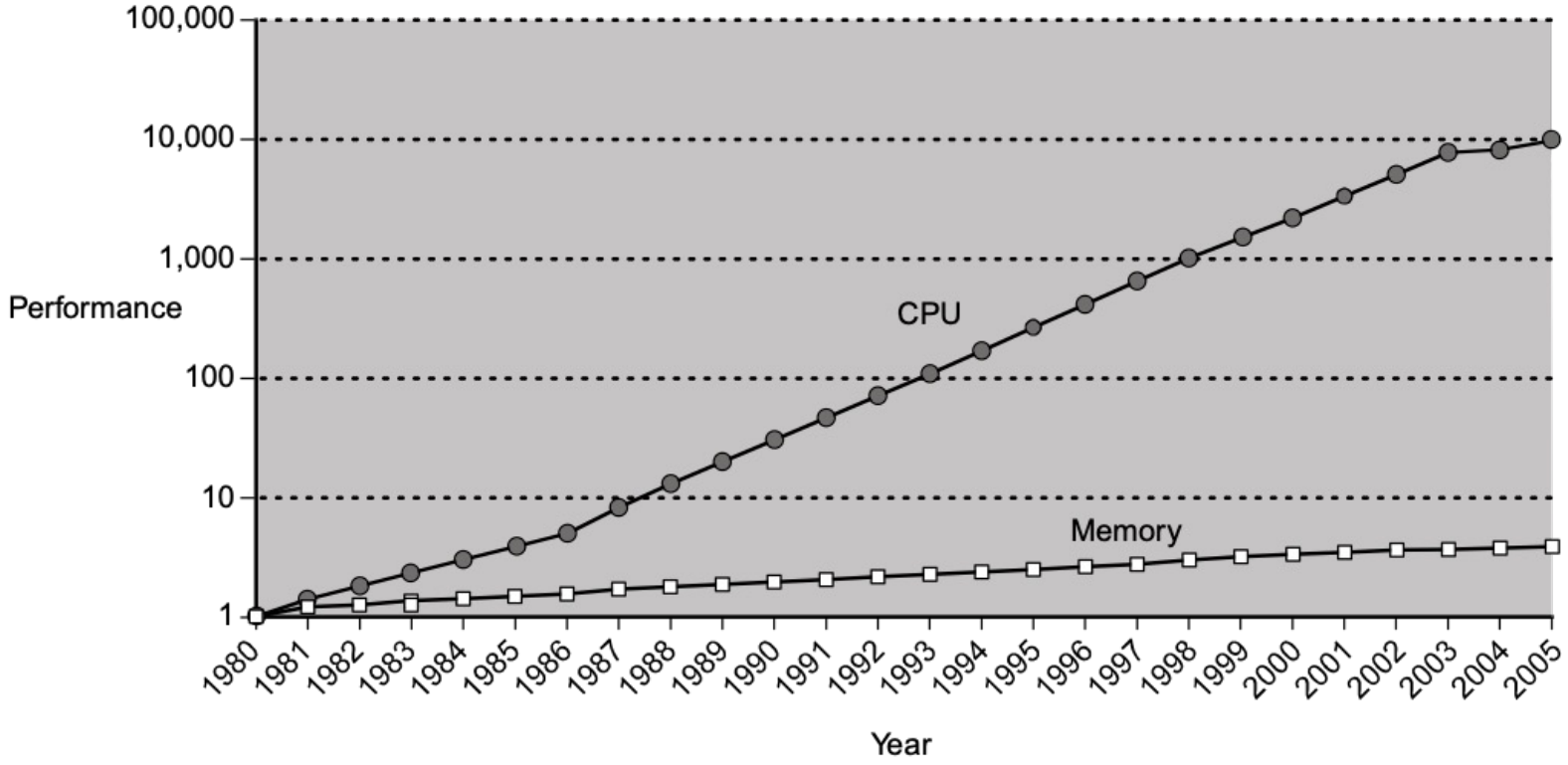
# Memory Hierarchy (Chapter 5)

# Memory Bottleneck

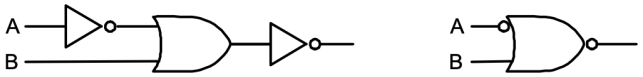
- **Processor speeds continue to increase very fast — much faster than either DRAM or disk access times**



# Different Memory Technologies

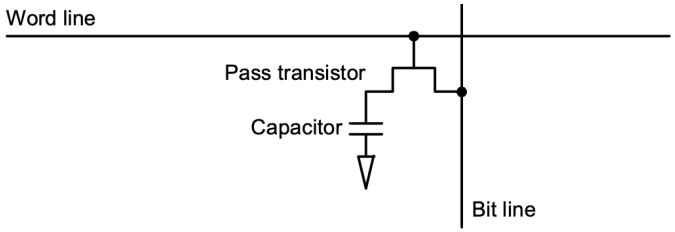
- **SRAM:**
  - **value is stored  on a pair of inverting gates**
  - **very fast but takes up more space than DRAM (4 to 6 transistors)**

- **DRAM:**
  - **value is stored as a charge on capacitor (must be refreshed)**
  - **very small but slower than SRAM (factor of 5 to 10)**



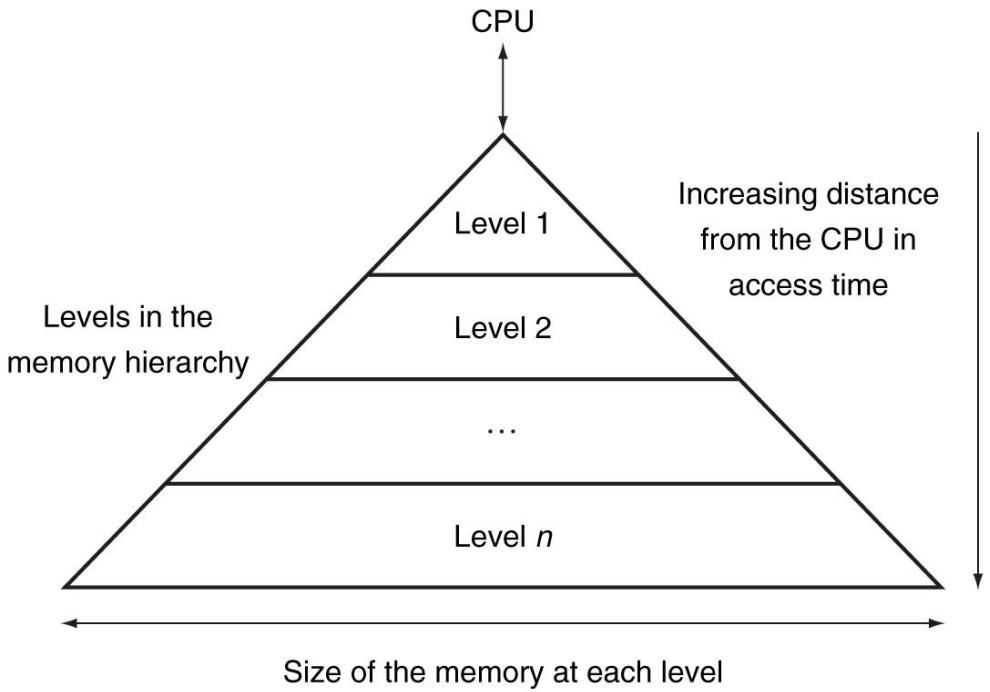






# Exploiting Memory Hierarchy

| Memory technology | Typical access time | $ per GiB in 2020 |
|---|---|---|
| SRAM semiconductor memory | 0.5–2.5 ns | $500–$1000 |
| DRAM semiconductor memory | 50–70 ns | $3–$6 |
| Flash semiconductor memory | 5,000–50,000 ns | $0.06–$0.12 |
| Magnetic disk | 5,000,000–20,000,000 ns | $0.01–$0.02 |



# SRAM (Static random access memory) 기술

- 읽기 접근 시간과 쓰기 접근 시간이 다를 수는 있지만, 데이터 위치에 상관 없이 접근 시간은 같다.
- 리프레시(refresh)가 필요 없다.
  - **접근 시간 ≈ 사이클 시간**
- 비트 당 6 ~ 8개의 트랜지스터
- 오늘날에는 모든 계층의 캐시가 프로세서 칩에 집적되므로 분리된 SRAM 칩 시장은 거의 사라졌다.

# DRAM (Dynamic random access memory) 기술

- 데이터를 전하 형태로 커패시터에 저장
  - 비트 당 트랜지스터 1개
    - SRAM에 비하여 훨씬 더 집적도가 높고 값도 싸다.
  - 전하는 수 밀리초 동안만 유지된다.
    - 주기적인 리프레시 필요
    - 셀 단위가 아니라 행 단위 리프레시
      - 읽기 사이클에서 전체 행을 한꺼번에 읽은 후 바로 쓰기 실행

# 플래시 메모리

- ROM (read only memory)
  - Mask ROM
  - PROM(Programmable ROM)
  - EPROM(Erasable Programmable ROM)
  - EEPROM(Electrically Erasable Programmable ROM)
- Flash memory
  - 비휘발성(nonvolatile)이면서 읽고 쓰기가 가능
  - 전기적으로 지울 수 있고 프로그래밍이 가능한 EEPROM의 한 종류
  - 마모 균등화(wear leveling)
    - 플래시 메모리의 쓰기는 비트를 마모시킨다.
    - 여러 번 쓰기가 수행된 블록을 덜 사용된 블록에 재사상(remapping)해서 쓰기를 분산시키는 방법

# 디스크 메모리

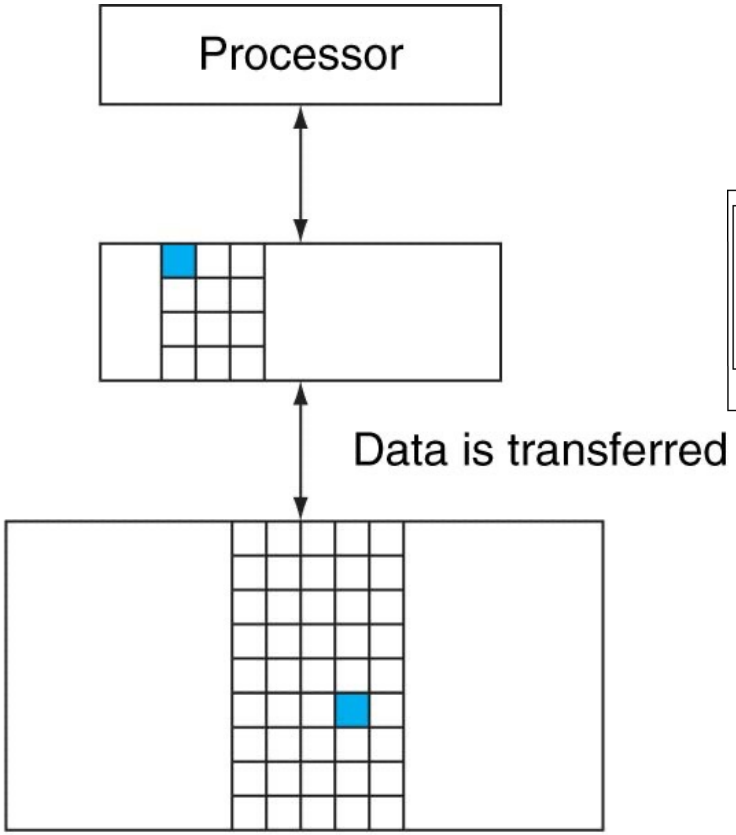
- 하드 디스크는 원판(platter)의 집합으로 구성
  - 원판은 5,400 ~ 15,000 rpm의 속도로 회전
  - 양면이 자성체로 코팅
- 트랙(track)
  - 자기 디스크의 표면을 이루고 있는 동심원
- 섹터 (sector)
  - 자기 디스크의 트랙을 구성하는 여러 세그먼트 중 하나
  - 디스크에 정보를 쓰거나 읽는 작업의 최소 단위
  - 512 ~ 4096 바이트
- 디스크 접근시간
  - 탐색 시간(seek time)

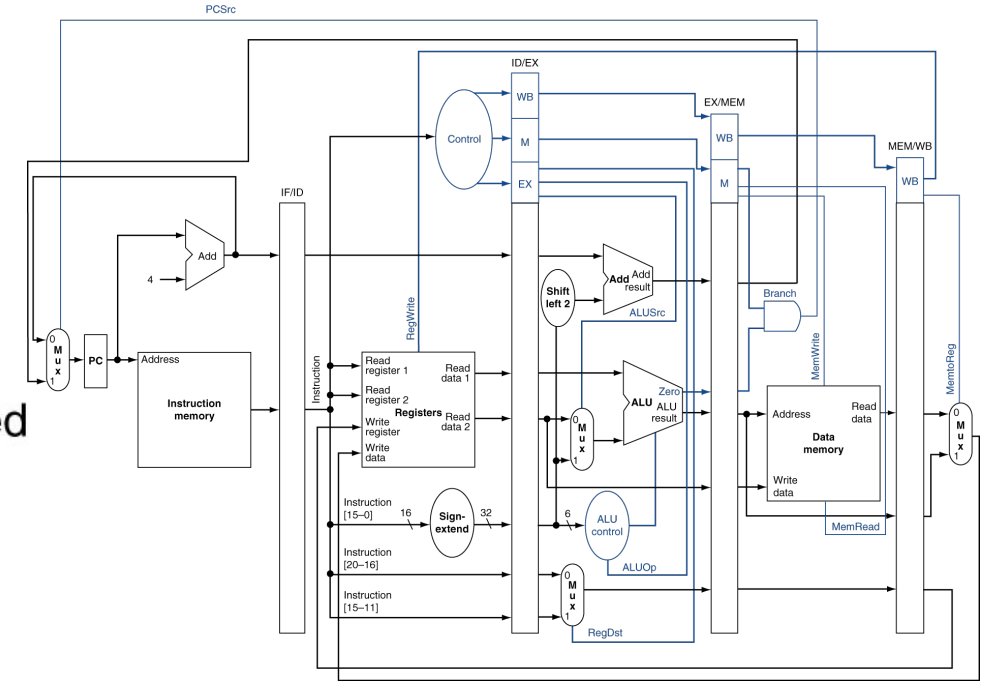
    + 회전 지연 시간 (rotational latency 또는 rotational delay)

    + 전송 시간(transfer time)

# Two levels of memory

- **Our initial focus: two levels (upper, lower)**
  - **block(line): minimum unit of data transfer**
  - **hit: data requested is in the upper level memory**
  - **miss: data requested is not in the upper level memory**



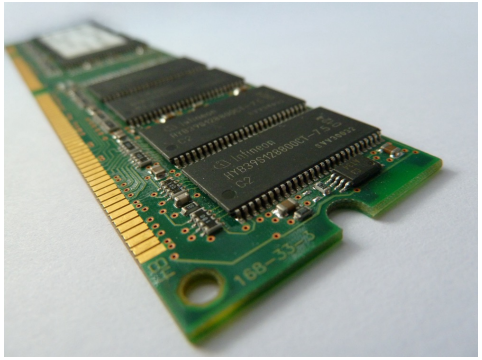



# 두 가지 지역성

1. 시간적 지역성(temporal locality)
   - 최근에 참조된 항목은 곧 다시 참조될 확률이 높다.
   - 메모리 접근이 시간적으로 집중되는 현상
   - 예: 순환문

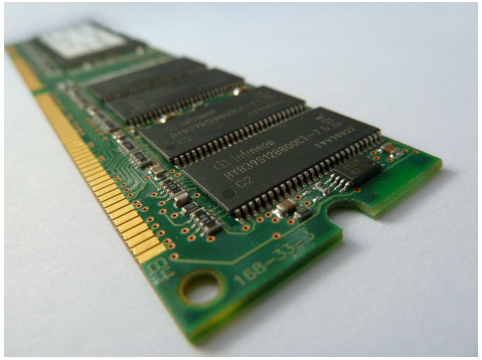
2. 공간적 지역성(spatial locality)
   - 어떤 항목이 참조되면 그 근처에 있는 다른 항목들이 곧바로 참조될 가능성이 높다.
   - 메모리 접근이 공간적으로 집중되는 현상
   - 예: 명령어의 순차적 실행
     배열의 접근

# Memory



```
0000 0000 0000 0101 0001 0000 1000 0000
0000 0000 1000 0010 0001 0000 0010 0000
1000 1100 0100 1111 0000 0000 0000 0000
1000 1100 0101 0000 0000 0000 0000 0100
1010 1100 0101 0000 0000 0000 0000 0000
1010 1100 0100 1111 0000 0000 0000 0100
0000 0011 1110 0000 0000 0000 0000 1000
```

# Memory



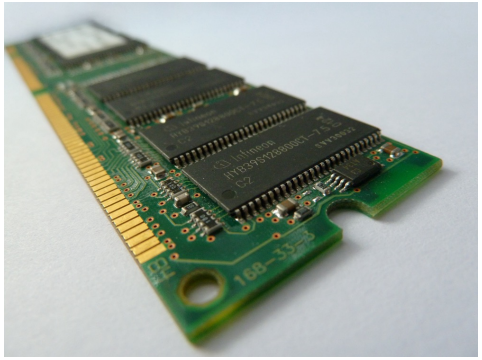
= 0x00400024 (hexadecimal representation)

[0000 0000 0100 0000 0000 0000 0010 0100]
[0000 0000 0100 0000 0000 0000 0010 1000]
[0000 0000 0100 0000 0000 0000 0010 1100]
[0000 0000 0100 0000 0000 0000 0011 0000]
[0000 0000 0100 0000 0000 0000 0011 0100]
[0000 0000 0100 0000 0000 0000 0011 1000]
[0000 0000 0100 0000 0000 0000 0011 1100]

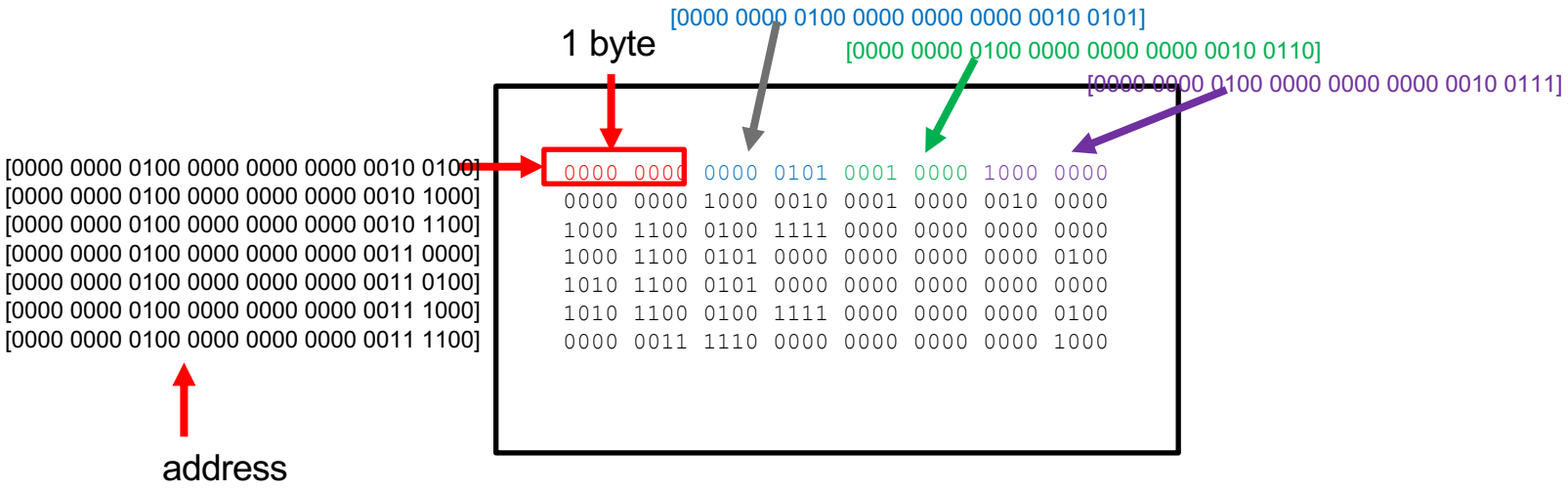
address

```
0000 0000 0000 0101 0001 0000 1000 0000
0000 0000 1000 0010 0001 0000 0010 0000
1000 1100 0100 1111 0000 0000 0000 0000
1000 1100 0101 0000 0000 0000 0000 0100
1010 1100 0101 0000 0000 0000 0000 0000
1010 1100 0100 1111 0000 0000 0000 0100
0000 0011 1110 0000 0000 0000 0000 1000
```

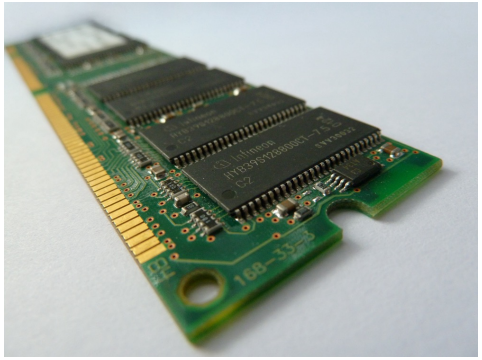
# Memory

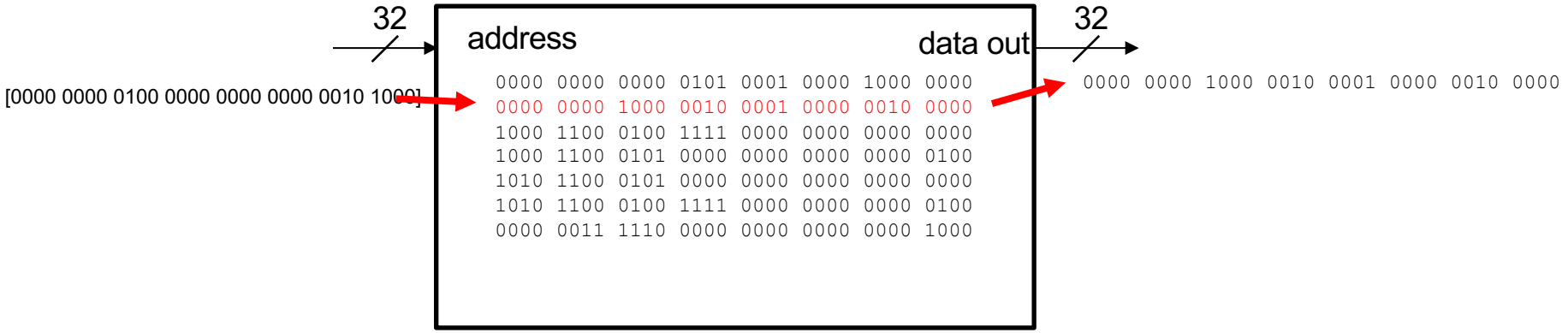


byte addressing

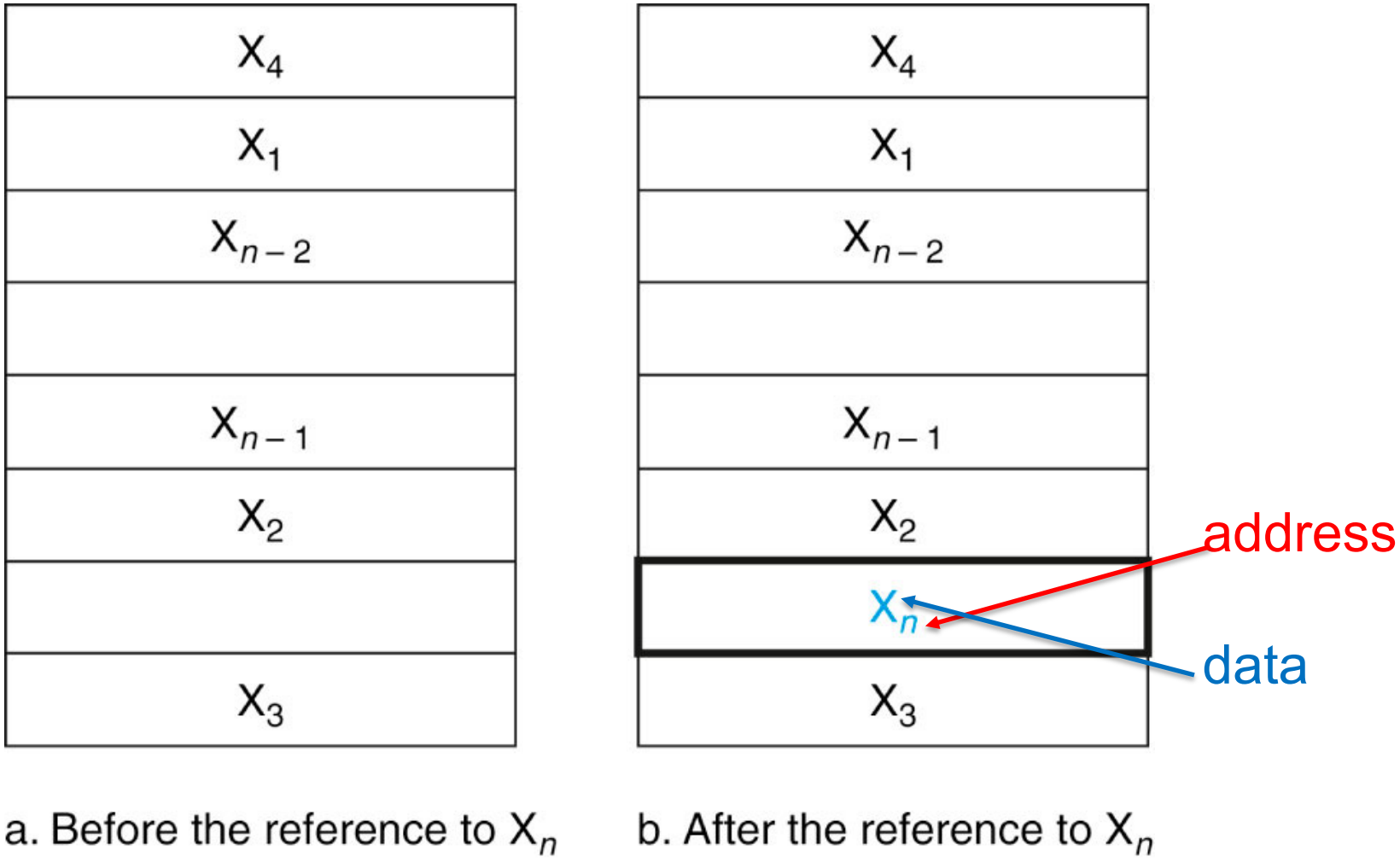


# Memory Read




32
address
data out
32
[0000 0000 0100 0000 0000 0000 0010 1000]
0000 0000 0000 0101 0001 0000 1000 0000
0000 0000 1000 0010 0001 0000 0010 0000
1000 1100 0100 1111 0000 0000 0000 0000
1000 1100 0101 0000 0000 0000 0000 0100
1010 1100 0101 0000 0000 0000 0000 0000
1010 1100 0100 1111 0000 0000 0000 0100
0000 0011 1110 0000 0000 0000 0000 1000
0000 0000 1000 0010 0001 0000 0010 0000

# Basics of Cache




a. Before the reference to $X_n$

b. After the reference to $X_n$

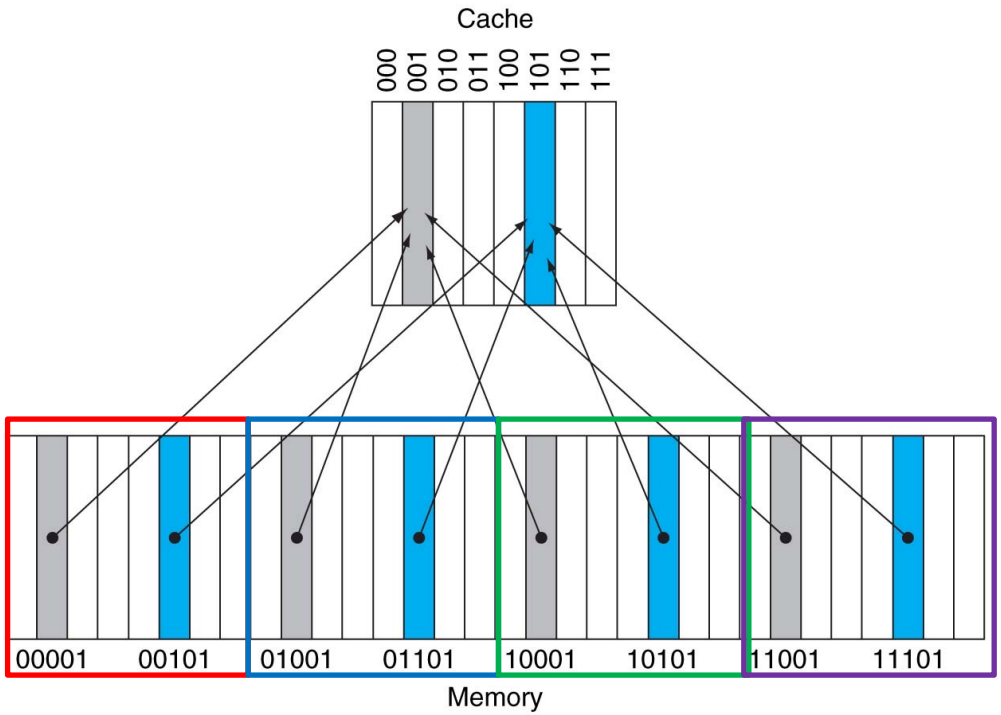
# Cache Organizations

1. Direct Mapped Cache
2. Set Associative Cache

Two issues:

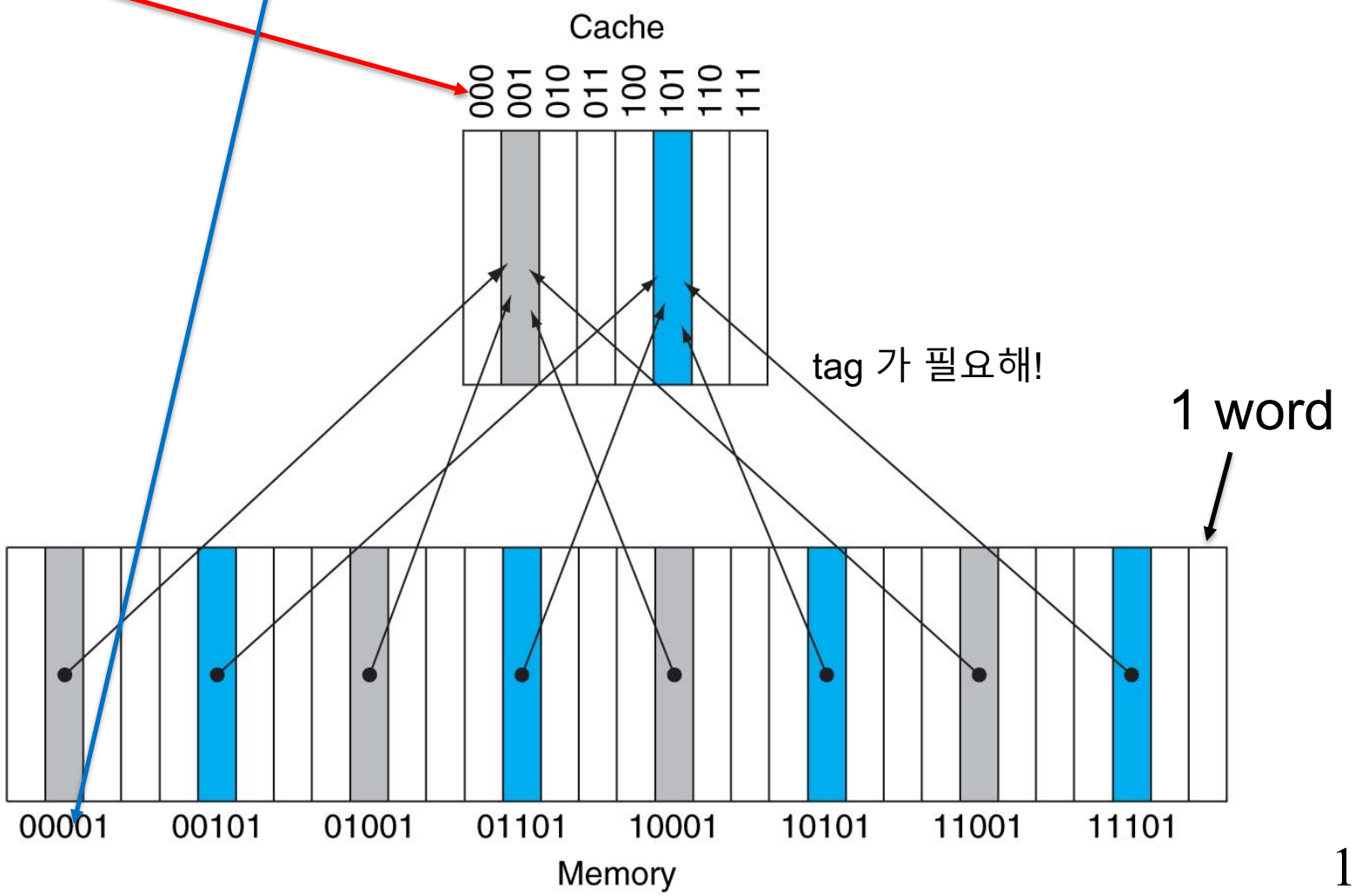
- How do we know if a data item is in the cache?
- If it is, how do we find it?

memory is referenced by address.



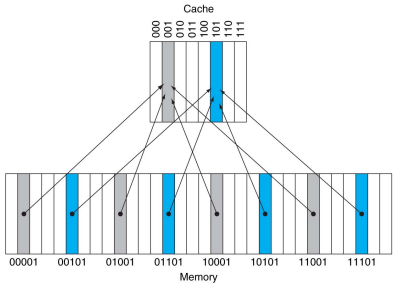
# Direct Mapped Cache (word addressing, 1 word/block)

**cache index** = memory address % **the number of blocks in the cache**



# Direct Mapped Cache : An example (word addressing)

memory reference sequence




| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

cache contents

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | N | | |
| 111 | N | | |

저장된 부분

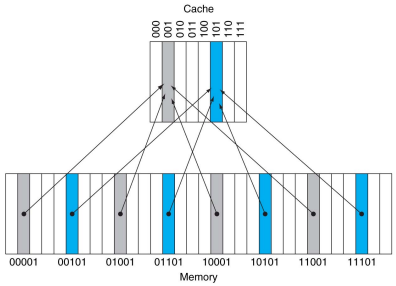
1 block

a. The initial state of the cache after power-on

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | N | | |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

b. After handling a miss of address ($10110_{two}$)

# Direct Mapped Cache : An example (word addressing)




| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

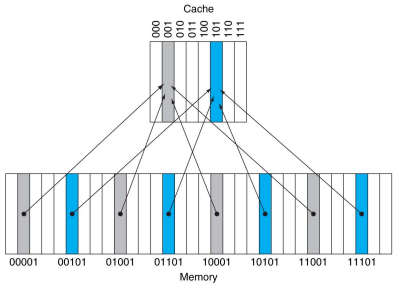
| Index | V | Tag | Data |
|---|---|---|---|
| 000 | N | | |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

c. After handling a miss of address ($11010_{two}$)

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | N | | |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

d. After handling a miss of address ($10000_{two}$)

# Direct Mapped Cache : An example (word addressing)




| Decimal address of reference | Binary address of reference | Hit or miss in cache | Assigned cache block (where found or placed) |
|---|---|---|---|
| 22 | $10110_{two}$ | miss (5.6b) | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | miss (5.6c) | $(11010_{two} \bmod 8) = 010_{two}$ |
| 22 | $10110_{two}$ | hit | $(10110_{two} \bmod 8) = 110_{two}$ |
| 26 | $11010_{two}$ | hit | $(11010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | miss (5.6d) | $(10000_{two} \bmod 8) = 000_{two}$ |
| 3 | $00011_{two}$ | miss (5.6e) | $(00011_{two} \bmod 8) = 011_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |
| 18 | $10010_{two}$ | miss (5.6f) | $(10010_{two} \bmod 8) = 010_{two}$ |
| 16 | $10000_{two}$ | hit | $(10000_{two} \bmod 8) = 000_{two}$ |

26 11010

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $11_{two}$ | Memory ($11010_{two}$) |
| 011 | Y | $00_{two}$ | Memory ($00011_{two}$) |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

e. After handling a miss of address ($00011_{two}$)

| Index | V | Tag | Data |
|---|---|---|---|
| 000 | Y | $10_{two}$ | Memory ($10000_{two}$) |
| 001 | N | | |
| 010 | Y | $10_{two}$ | Memory ($10010_{two}$) |
| 011 | Y | $00_{two}$ | Memory ($00011_{two}$) |
| 100 | N | | |
| 101 | N | | |
| 110 | Y | $10_{two}$ | Memory ($10110_{two}$) |
| 111 | N | | |

f. After handling a miss of address ($10010_{two}$)

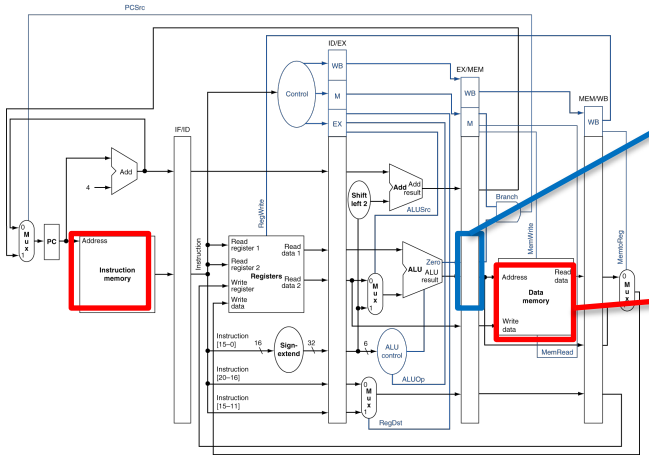
**Cache conflict at 18 (f)**

**Compulsory misses at 22,26,16,3 and 18**

**Conflict miss at 26**

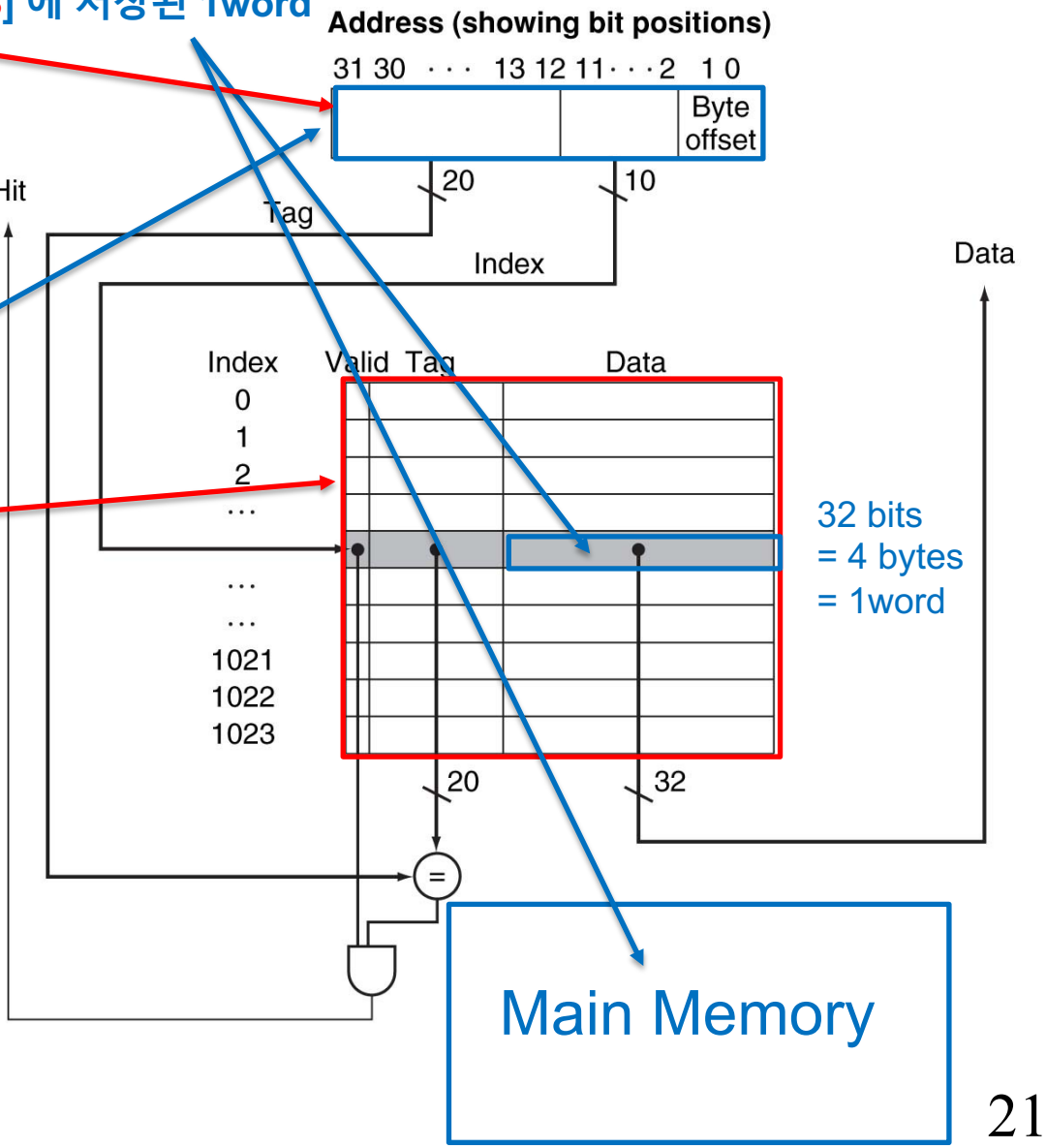
# Direct Mapped Cache (byte addressing, 1 word/block)

[메모리 주소 0x10000010~0x10000013] 에 저장된 1word



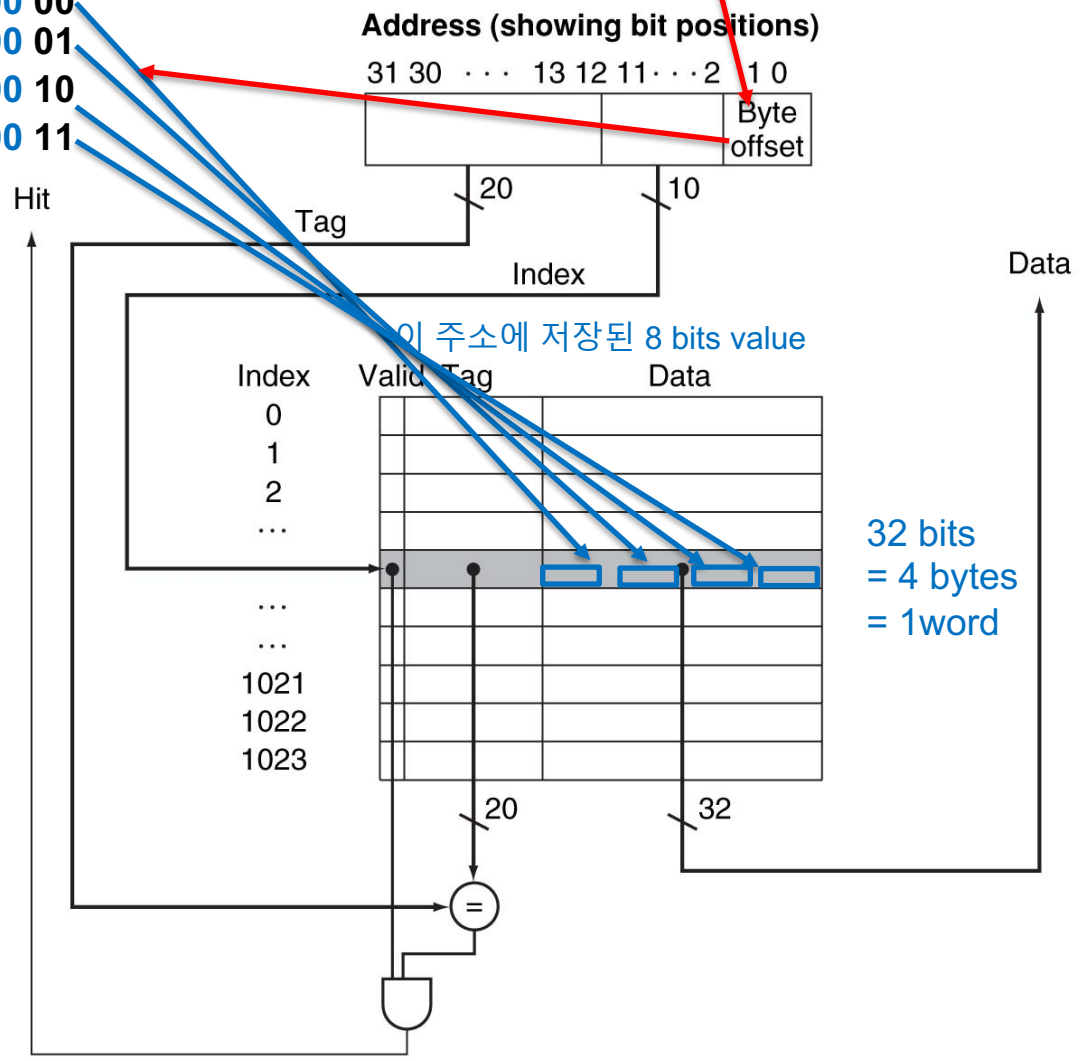
lw $4, 0($3)

$3 : 0x10000010



# Direct Mapped Cache (byte addressing, 1 word/block)

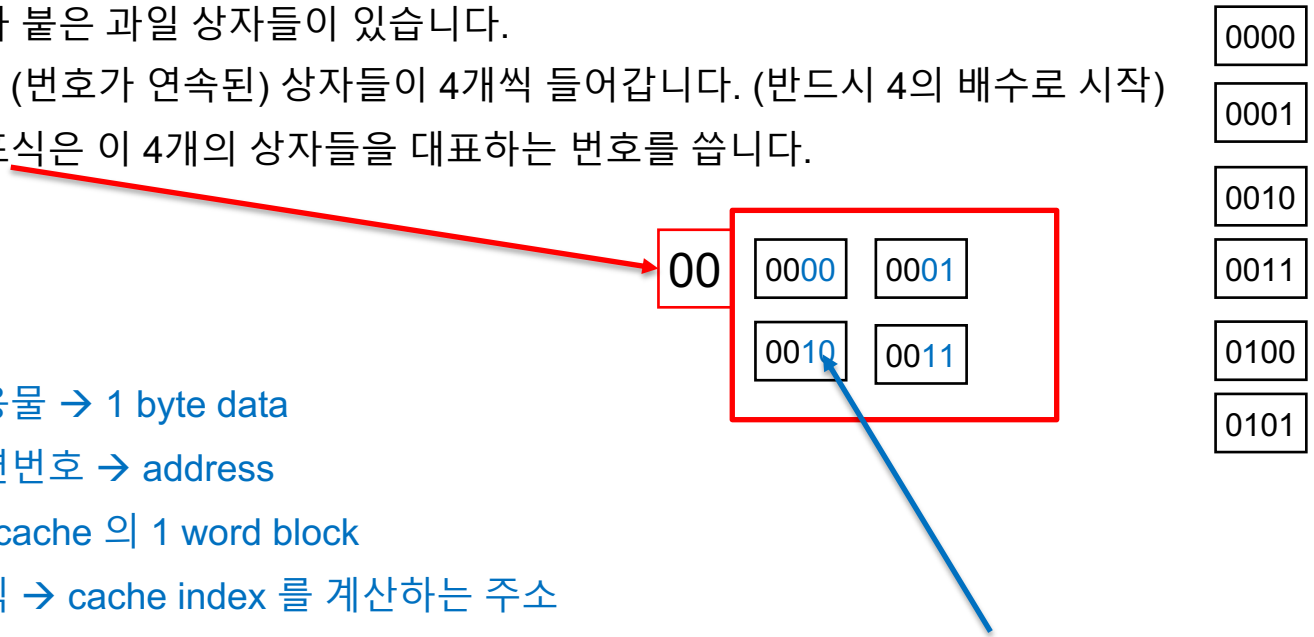
0001 0000 0000 0000 0000 0000 0001 00 00
0001 0000 0000 0000 0000 0000 0001 00 01
0001 0000 0000 0000 0000 0000 0001 00 10
0001 0000 0000 0000 0000 0000 0001 00 11
[메모리 주소 0x10000010~0x10000013]




*What kind of locality are we taking advantage of?*

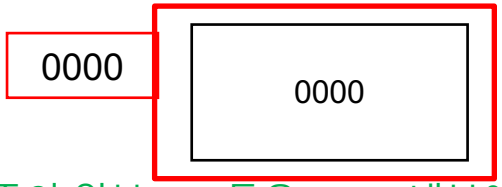
# byte offset

- 일련 번호가 붙은 과일 상자들이 있습니다.
- 바구니에는 (번호가 연속된) 상자들이 4개씩 들어갑니다. (반드시 4의 배수로 시작)
- 바구니의 표식은 이 4개의 상자들을 대표하는 번호를 씁니다.



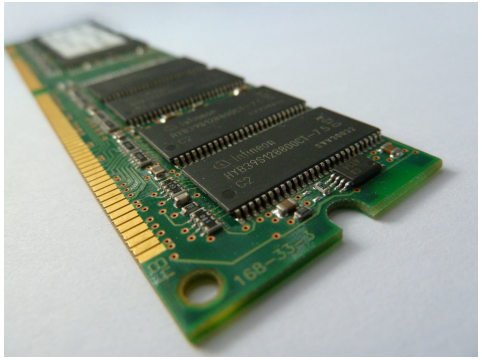

- 상자의 내용물 → 1 byte data
- 상자의 일련번호 → address
- 바구니 → cache 의 1 word block
- 바구니 표식 → cache index 를 계산하는 주소
- 바구니 안에서 4개의 상자들을 구분하기 위해서 사용하는 bits 가 byte offset

word addresing 을 하고 block 의 크기가 1 word 일 때는 1 block 에 저장된 data 의 주소가 1개이니까 block 내부에서 주소를 구분하는 offset 부분이 필요가 없었다.

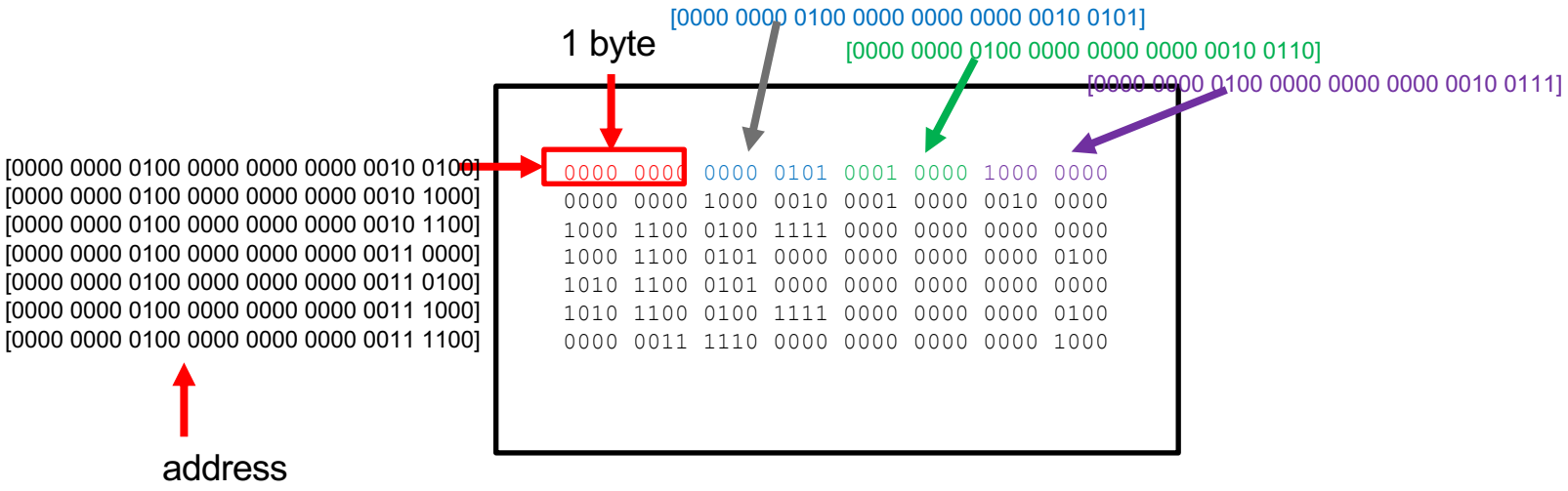



실제로는 block 내에 저장된 data 의 주소가 여러 개이므로 주소 중의 일부 bits 들은 block 내부에서 주소를 구분하는 offset 으로 사용된다.

# Memory

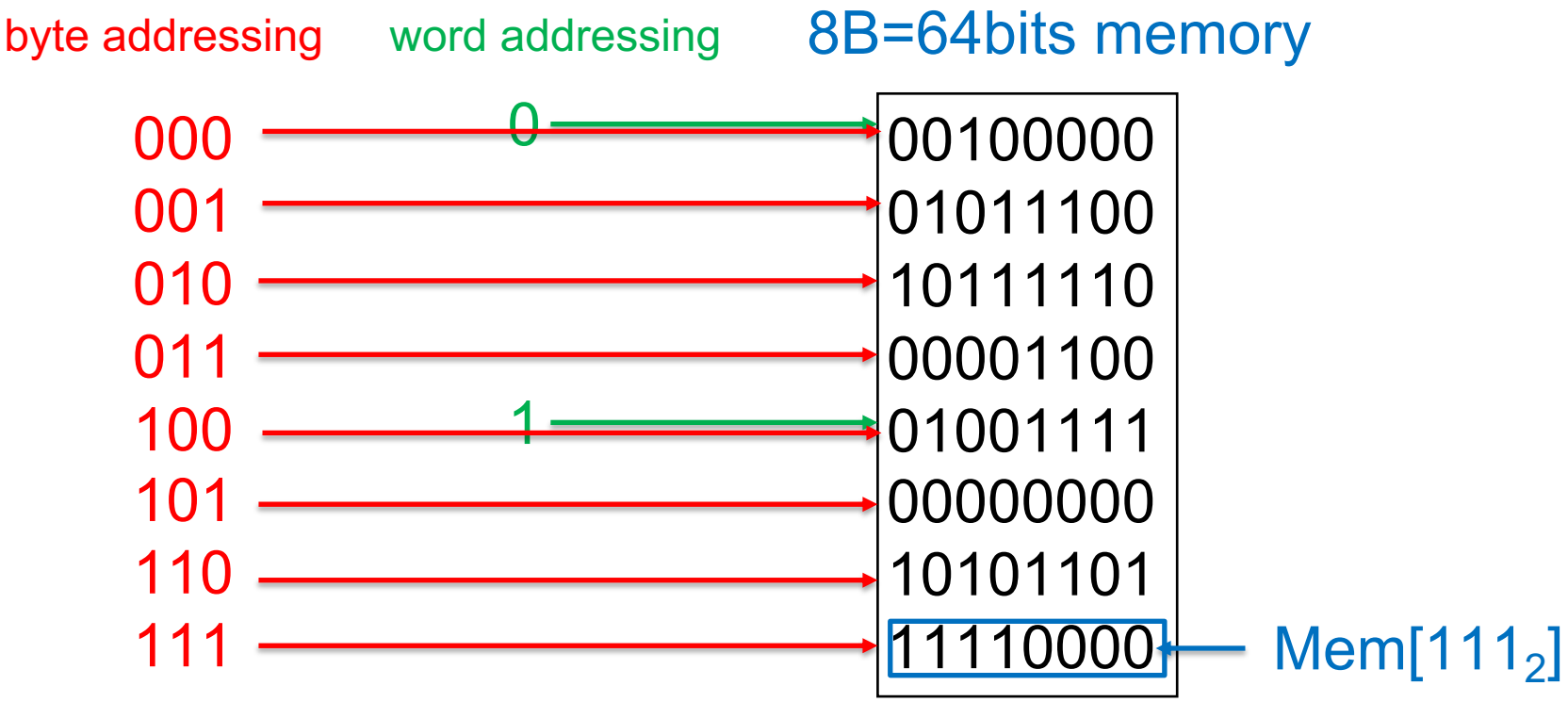


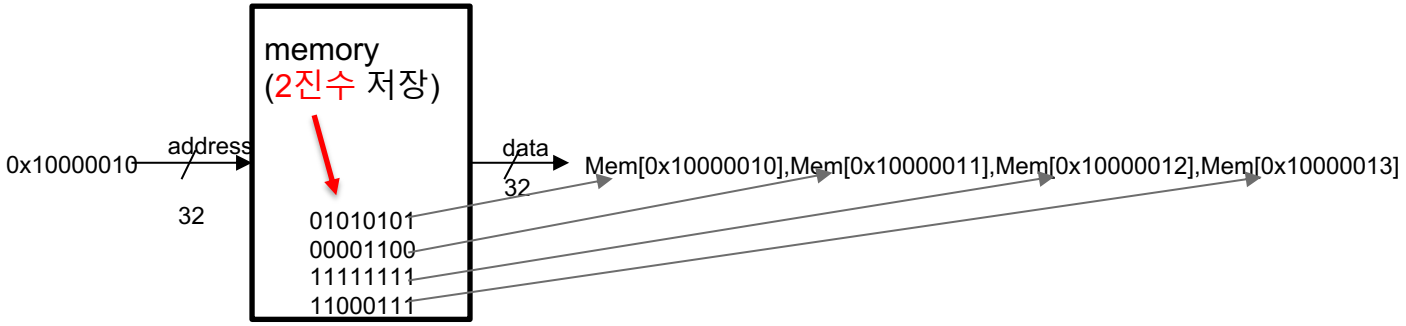
byte addressing



# What is memory address?

- 메모리에 저장된 것은 2진수 형태의 data 들 (bit 들의 모임) 이고,
- 이 수많은 bits 중 특정한 bits 를 지칭하기 위한 일련번호가 주소이며
- 다시 말하지만, 주소(index)는 메모리(cache)에 저장된 내용이 아닙니다!!!




memory
(2진수 저장)
0x10000010
address
32
01010101
00001100
11111111
11000111
data
32
Mem[0x10000010],Mem[0x10000011],Mem[0x10000012],Mem[0x10000013]

# Direct Mapped Cache (byte addressing, 1 word/block)

0001 0000 0000 0000 0000 0000 0001 0000
0001 0000 0000 0000 0000 0000 0001 0001
0001 0000 0000 0000 0000 0000 0001 0010
0001 0000 0000 0000 0000 0000 0001 0011
[메모리 주소 0x10000010~0x10000013]