
Computer Architecture

Chapter 5a. Memory System

Hyuk-Jun Lee, PhD

Dept. of Computer Science and Engineering
Sogang University
Seoul, Korea

Email: hyukjunl@sogang.ac.kr



Sogang University

Memory Technology

특징

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM) → 주기적으로 refresh.
 - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk (HDD)
 - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost/GB of disk

cell
SRAM > DRAM
speed
SRAM > DRAM
capacity
DRAM > SRAM



Principle of Locality (지역성)

- Programs access a small proportion of their address space at any time 전체 페이지가 리드 ~~해당~~ ~~부분~~ 일부분만 읽은
- Temporal locality 시간적 지역성
 - Items accessed recently are likely to be accessed again soon
 - e.g., instructions in a loop, induction variables
- Spatial locality 공간적 지역성
 - Items near those accessed recently are likely to be accessed soon
 - E.g., sequential instruction access, array data

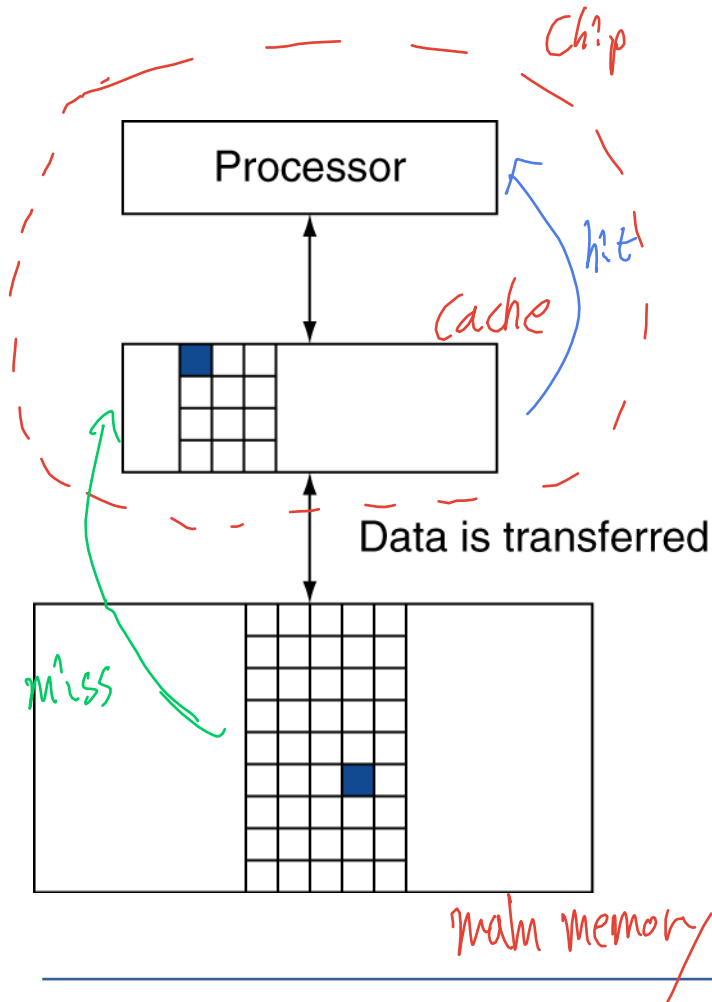


Taking Advantage of Locality

- Memory hierarchy
- Store everything on disk
- Copy recently accessed (and nearby) items from disk to smaller DRAM memory
 - Main memory
- Copy more recently accessed (and nearby) items from DRAM to smaller SRAM memory
 - Cache memory attached to CPU



Memory Hierarchy Levels



- Block (aka line): unit of copying
 - May be multiple words
- If accessed data is present in upper level
 - Hit: access satisfied by upper level
 - Hit ratio: hits/accesses ex) 95/100
- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses
= 1 – hit ratio ex) 5/100
 - Then accessed data supplied from upper level

Cache Memory

- Cache memory
 - The level of the memory hierarchy closest to the CPU
- Given accesses X_1, \dots, X_{n-1}, X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_3

a. Before the reference to X_n

X_4
X_1
X_{n-2}
X_{n-1}
X_2
X_n
X_3

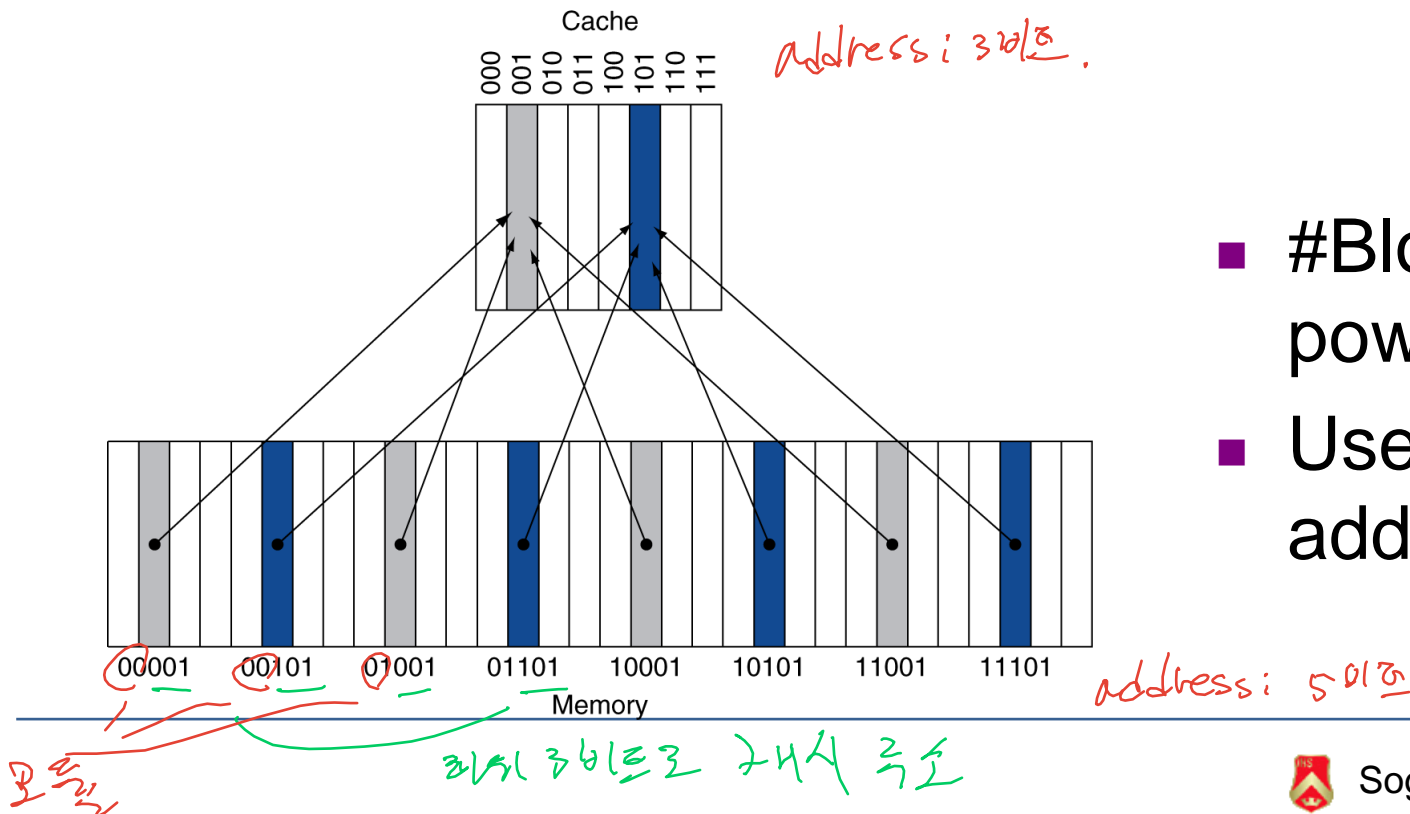
b. After the reference to X_n

- How do we know if the data is present?
- Where do we look?



Direct Mapped Cache

- Location determined by address
- Direct mapped: only one choice
 - (Block address) modulo (#Blocks in cache)



- #Blocks is a power of 2
- Use low-order address bits



Cache block: V tag Data

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
data or instruction
↓
 - Store block address as well as the data
 - Actually, only need the high-order bits of address
 - Called the tag
*→ 앞의 여백어는
상위 2비트 (tag)*
- What if there is no data in a location?
 - Valid bit: 1 = present, 0 = not present
 - Initially 0
→ tag: garbage.



Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state
- Order of memory references

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

Address
22
26
22
26
16
3
16
18



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Miss	110

value bit *tag* *index*

(copied from lower level)

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

✓



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
26	11 010	Miss	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
22	10 110	Hit	110
26	11 010	Hit	010

Index	V	Tag	Data
000	N		
001	N		
010	Y	11	Mem[11010]
011	N		
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

Cache에 들어간 것



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
16	10 000	Miss	000
3	00 011	Miss	011
16	10 000	Hit	000

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	11	Mem[11010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Cache Example

Word addr	Binary addr	Hit/miss	Cache block
18	10 010	Miss	010

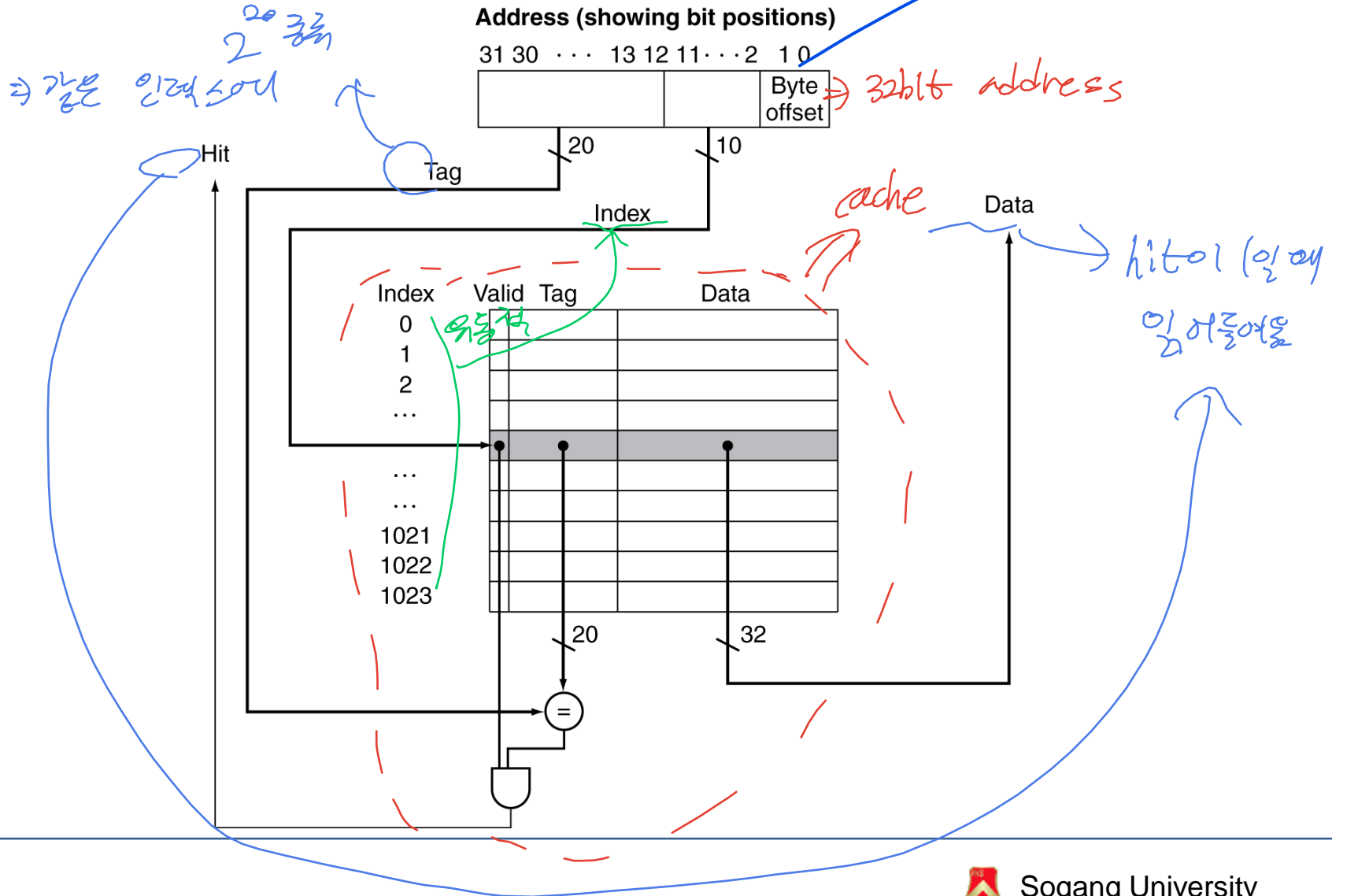
replacement

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N		
010	Y	10	Mem[10010]
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		

→ 2 for 1011!
1010 → 10010



Address Subdivision

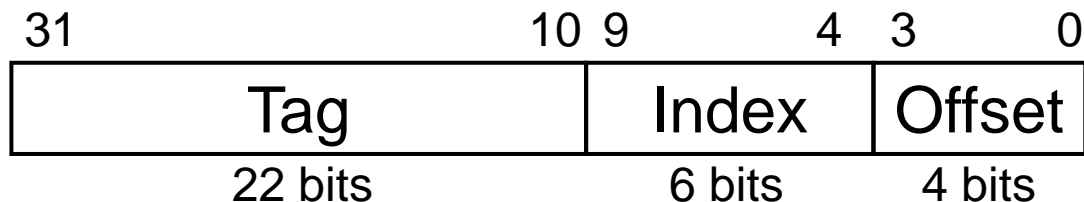


Example: Larger Block Size

size of cache $\Rightarrow 64 \times 16 \text{ B} = 1024 \text{ B}$

- 64 blocks, 16 bytes/block $\approx 4 \text{ words/block}$
 - To what block number does address 1200 map?


- Block address = $\lfloor 1200/16 \rfloor = 75 \Rightarrow$ 4 bits (offset 2 bits)
- Block number = $75 \text{ modulo } 64 = 11$



0100 1011
tag Index

block offset



MR  \Rightarrow pollute
때문에 계속 줄이는 것 못함.

Block Size Considerations

- Larger blocks should reduce miss rate

- Due to spatial locality

\Rightarrow miss
A[0], A[1], A[2], A[3], A[4]...
(한번
다녀오
같은
공간
다녀오)

- But in a fixed-sized cache

- Larger blocks \Rightarrow fewer of them

- More competition \Rightarrow increased miss rate

- Larger blocks \Rightarrow pollution overhead \uparrow

- Larger miss penalty

- Can override benefit of reduced miss rate problem

- Early restart and critical-word-first can help

solution.



Cache Misses

- On cache hit, CPU proceeds normally
 - On cache miss
 - Stall the CPU pipeline
 - Fetch block from next level of hierarchy
 - Instruction cache miss
 - Restart instruction fetch
 - Data cache miss
 - Complete data access
- Handwritten notes:
- IF ID EX MEM WB
 ↓
 stall

Write-Through

- On data-write hit, could just update the block in cache
 - But then cache and memory would be inconsistent
- Write through: also update memory *물 다 업데이트*
- But makes writes take longer
 - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
 - Effective CPI = $1 + 0.1 \times 100 = 11$
- Solution: write buffer
 - Holds data waiting to be written to memory *받은 메모리 업데이트*
 - CPU continues immediately *write buffer의 데이터 사용,*
 - Only stalls on write if write buffer is already full



Write-Back

- Alternative: On data-write hit, just update the block in cache
 - Keep track of whether each block is dirty
- When a dirty block is replaced
 - Write it back to memory
 - Can use a write buffer to allow replacing block to be read first



Write Allocation

- What should happen on a write miss?
- Alternatives for write-through
 - Allocate on miss: fetch the block
 - Write around: don't fetch the block
 - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
 - Usually fetch the block



Example: Intrinsity FastMATH

- Embedded MIPS processor
 - 12-stage pipeline
 - Instruction and data access on each cycle

- Split cache: separate I-cache and D-cache

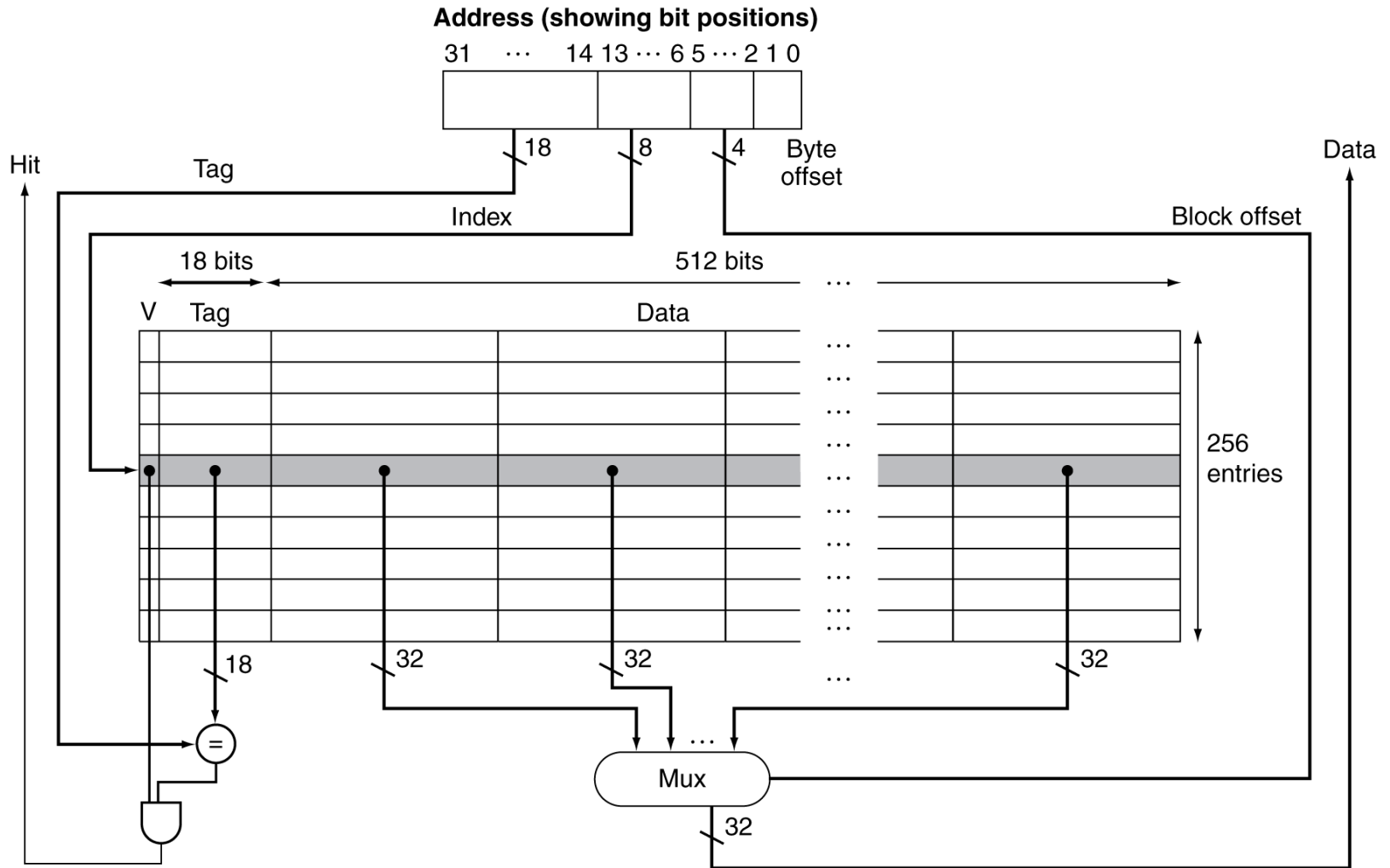
I-cache – Each 16KB: 256 blocks \times 16 words/block \rightarrow 4KB!

D-cache – D-cache: write-through or write-back

- SPEC2000 miss rates
 - I-cache: 0.4%
 - D-cache: 11.4%
 - Weighted average: 3.2%



Example: Intrinsity FastMATH

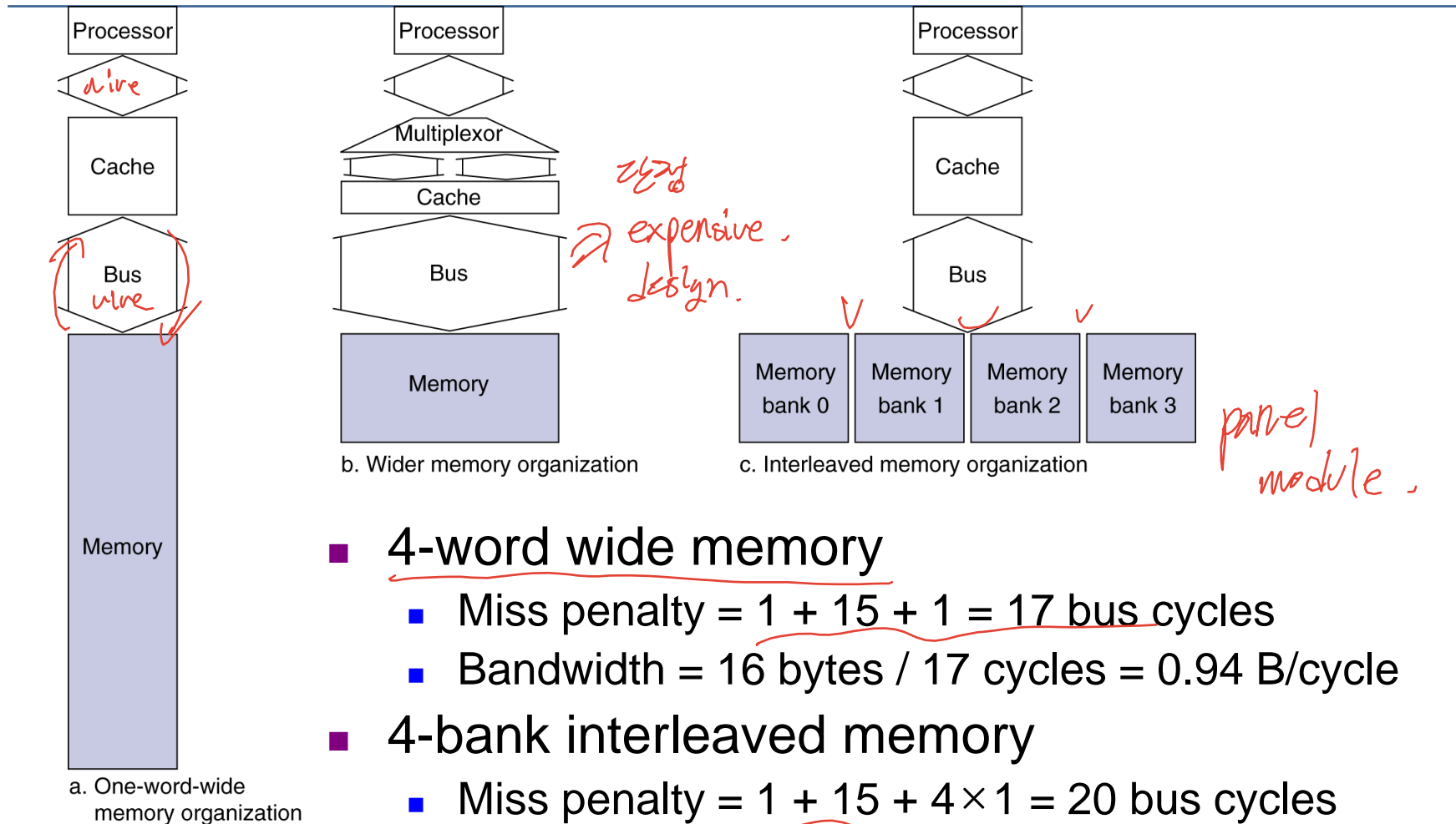


Main Memory Supporting Caches

- Use DRAMs for main memory
 - Fixed width (e.g., 1 word) *32 bit or 64 bit ...*
 - Connected by fixed-width clocked bus
 - Bus clock is typically slower than CPU clock
- Example cache block read
 - ✓ – 1 bus cycle for address transfer
 - ✓ – 15 bus cycles per DRAM access *read block.*
 - ✓ – 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
 - *→ 4-word 블록을 1-word-wide DRAM으로 읽는 것*
Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$



Increasing Memory Bandwidth



■ 4-word wide memory

- Miss penalty = $1 + 15 + 1 = 17$ bus cycles
- Bandwidth = $16 \text{ bytes} / 17 \text{ cycles} = 0.94 \text{ B/cycle}$

■ 4-bank interleaved memory

- Miss penalty = $1 + 15 + 4 \times 1 = 20$ bus cycles
- Bandwidth = $16 \text{ bytes} / 20 \text{ cycles} = 0.8 \text{ B/cycle}$

Measuring Cache Performance

- Components of CPU time

- Program execution cycles

- Includes cache hit time

- Memory stall cycles

- Mainly from cache misses

- With simplifying assumptions:

Memory stall cycles $\overset{(*)}{10^6} \times 0.5 \times 100 = 5 \times 10^8$

$$= \frac{\text{Memory accesses}}{\text{Program}} \times \text{Miss rate} \times \text{Miss penalty}$$

$$= \frac{\text{Instructions}}{\text{Program}} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$



Cache Performance Example

- Given
 - I-cache miss rate = 2%
 - D-cache miss rate = 4%
 - Miss penalty = 100 cycles
 - Base CPI (ideal cache) = 2
 - Load & stores are 36% of instructions
- Miss cycles per instruction
 - I-cache: $0.02 \times 100 = 2$
3% of instructions are cache miss
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
Base CPI
- Actual CPI = 2 + 2 + 1.44 = 5.44
 - Ideal CPU is $5.44/2 = 2.72$ times faster



Average Access Time

- Hit time is also important for performance
- Average memory access time (AMAT)
 - $\text{AMAT} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$
- Example
 - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, l-cache miss rate = 5%
 - ✓ – $\text{AMAT} = 1 + 0.05 \times 20 = 2\text{ns}$
 - 2 cycles per instruction

Performance Summary

- When CPU performance increased
 - Miss penalty becomes more significant
- Decreasing base CPI
 - Greater proportion of time spent on memory stalls
- Increasing clock rate
 - Memory stalls account for more CPU cycles
- (Can't neglect cache behavior when evaluating system performance)

$$CPI = CPI_{base} + CPI_{stall} + CPI_{cache}$$

+ CPI memory

