
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

- Static RAM (SRAM)
 - 0.5ns – 2.5ns, \$2000 – \$5000 per GB
- Dynamic RAM (DRAM) capaciter에서 전기가 빠져나옴 30msec마다
 - 50ns – 70ns, \$20 – \$75 per GB
- Magnetic disk
 - 5ms – 20ms, \$0.20 – \$2 per GB
- Ideal memory
 - Access time of SRAM
 - Capacity and cost/GB of disk



Principle of Locality

- Programs access a small proportion of their address space at any time
- Temporal locality program
 - Items accessed recently are likely to be accessed again soon 최근 사용한 게 곧 또 쓰인다고 예상
 - e.g., instructions in a loop, induction variables
- Spatial locality hardware
 - Items near those accessed recently are likely to be accessed soon 사용한 곳의 주변 메모리가 나중에 쓰인다고 예상
 - E.g., sequential instruction access, array data

한 클럭에 한 명령어를 수행하는 게 이상적인데, 데이터를 dram이나 디스크에서 가져오면 느리기 때문에 Locality를 사용



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

최근에 사용된 건 cache에 저장



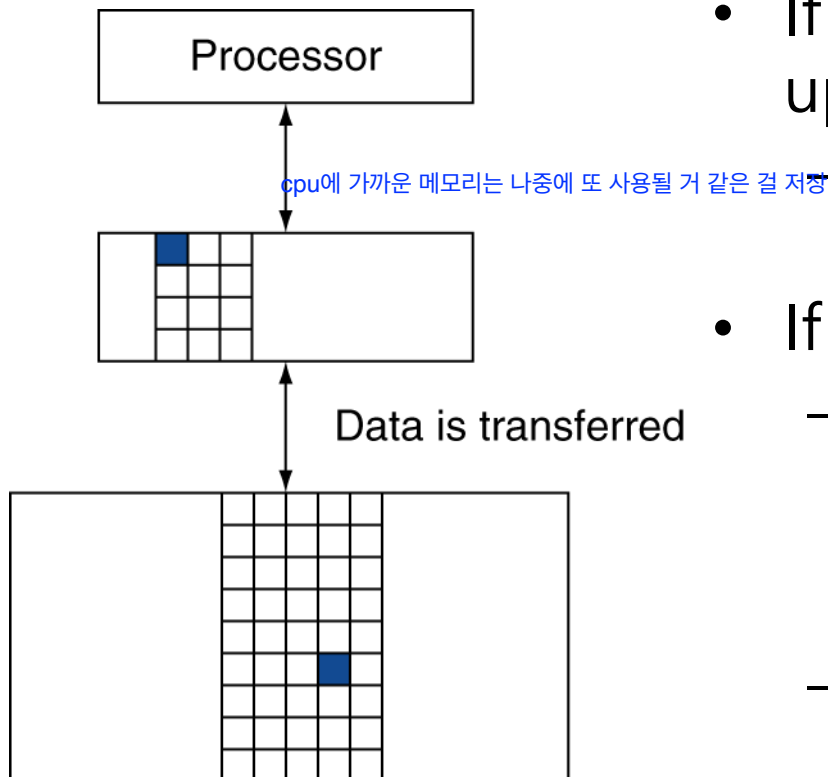
Memory Hierarchy Levels

옮길 때 block 단위로 옮김

- 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

- If accessed data is absent
 - Miss: block copied from lower level
 - Time taken: miss penalty
 - Miss ratio: misses/accesses
 $= 1 - \text{hit ratio}$
 - Then accessed data supplied from upper level



miss penalty - cache에 원하는 데이터가 없어서 main memory에서 데이터를 끌어오는데 걸리는 시간
miss ratio(rate) - miss # / total #. cache를 접근한 총 횟수에서 miss된 개수



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

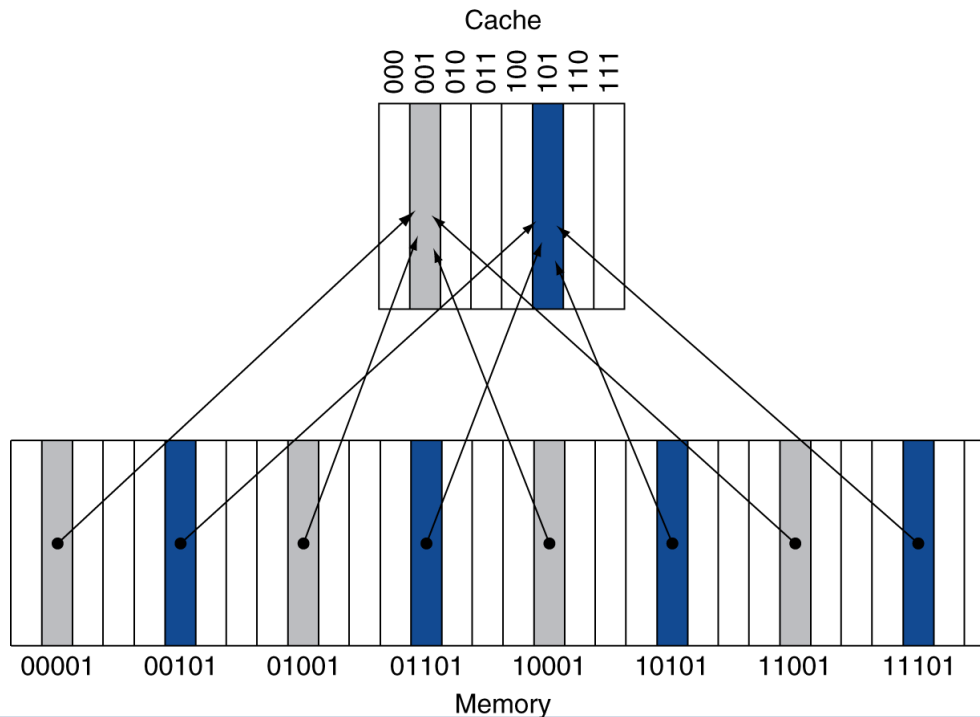
b. After the reference to X_n

- 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

Tags and Valid Bits

- How do we know which particular block is stored in a cache location?
 - Store block address as well as the data
 - Actually, only need the high-order bits
 - Called the tag
- What if there is no data in a location?
 - Valid bit: 1 = present, 0 = not present
 - Initially 0



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

앞2개 tag bit

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10	Mem[10110] 처음은 cache empty라서 miss
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		



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		



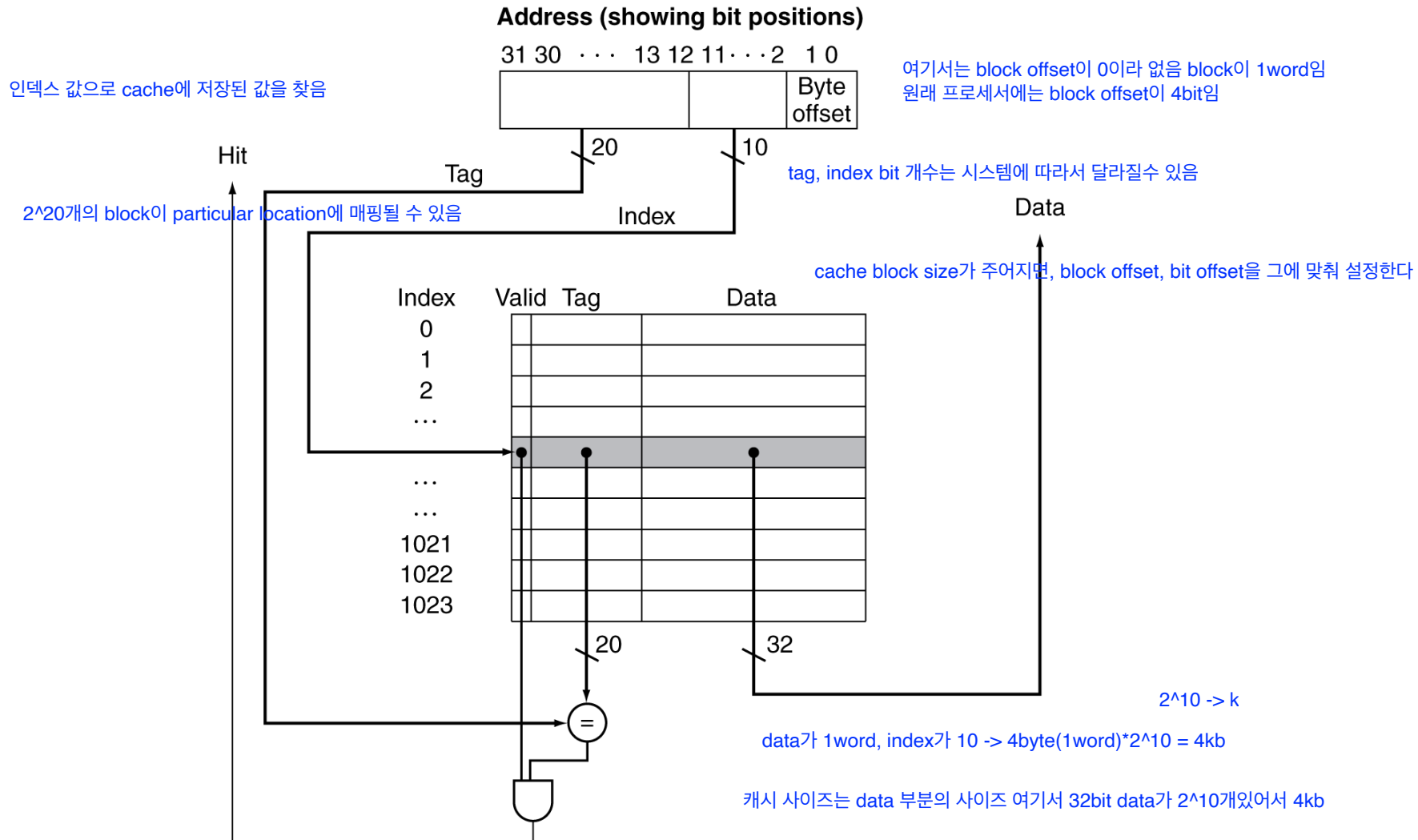
Cache Example

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

Index	V	Tag	Data
000	Y	10	Mem[10000]
001	N	tag가 다름	
010	Y	10	Mem[10010] replace with new value
011	Y	00	Mem[00011]
100	N		
101	N		
110	Y	10	Mem[10110]
111	N		



Address Subdivision

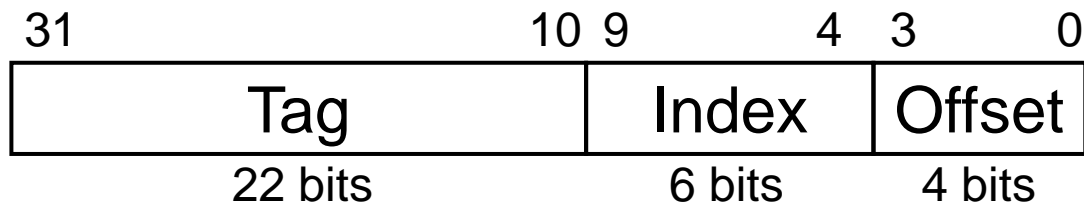


size of cache(size of data)= 2^(index+blockoffset+byteoffset)bytes



Example: Larger Block Size

- 64 blocks, 16 bytes/block
 - To what block number does address 1200 map?
- Block address = $\lfloor 1200/16 \rfloor = 75$
- Block number = $75 \text{ modulo } 64 = 11$



Block Size Considerations

- Larger blocks should reduce miss rate
 - Due to spatial locality
- But in a fixed-sized cache
 - Larger blocks \Rightarrow fewer of them
 - More competition \Rightarrow increased miss rate
 - Larger blocks \Rightarrow pollution
- Larger miss penalty
 - Can override benefit of reduced miss rate
 - Early restart and critical-word-first can help



Cache Misses

every instruction, access instruction cache
access data cache when load, store instruction

- 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



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
- 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
 - CPU continues immediately
 - Only stalls on write if write buffer is already full

cache에 바뀔게 buffer에 써지고 buffer가 main memory에 write
write buffer handles write to main memory

연속적으로 request가 들어올 때 buffer가 제어해줌
buffer 크기가 넘게 store하려 하면 그 때 stall



Write-Back

cache block이 replace될 때 memory update를 해야함
그 때 write buffer에 replace되는 걸 집어넣고 memory update함

- 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
non-allocate scheme -> write around (directly to main memory)
 - Since programs often write a whole block before reading it (e.g., initialization)
- For write-back
 - Usually fetch the block

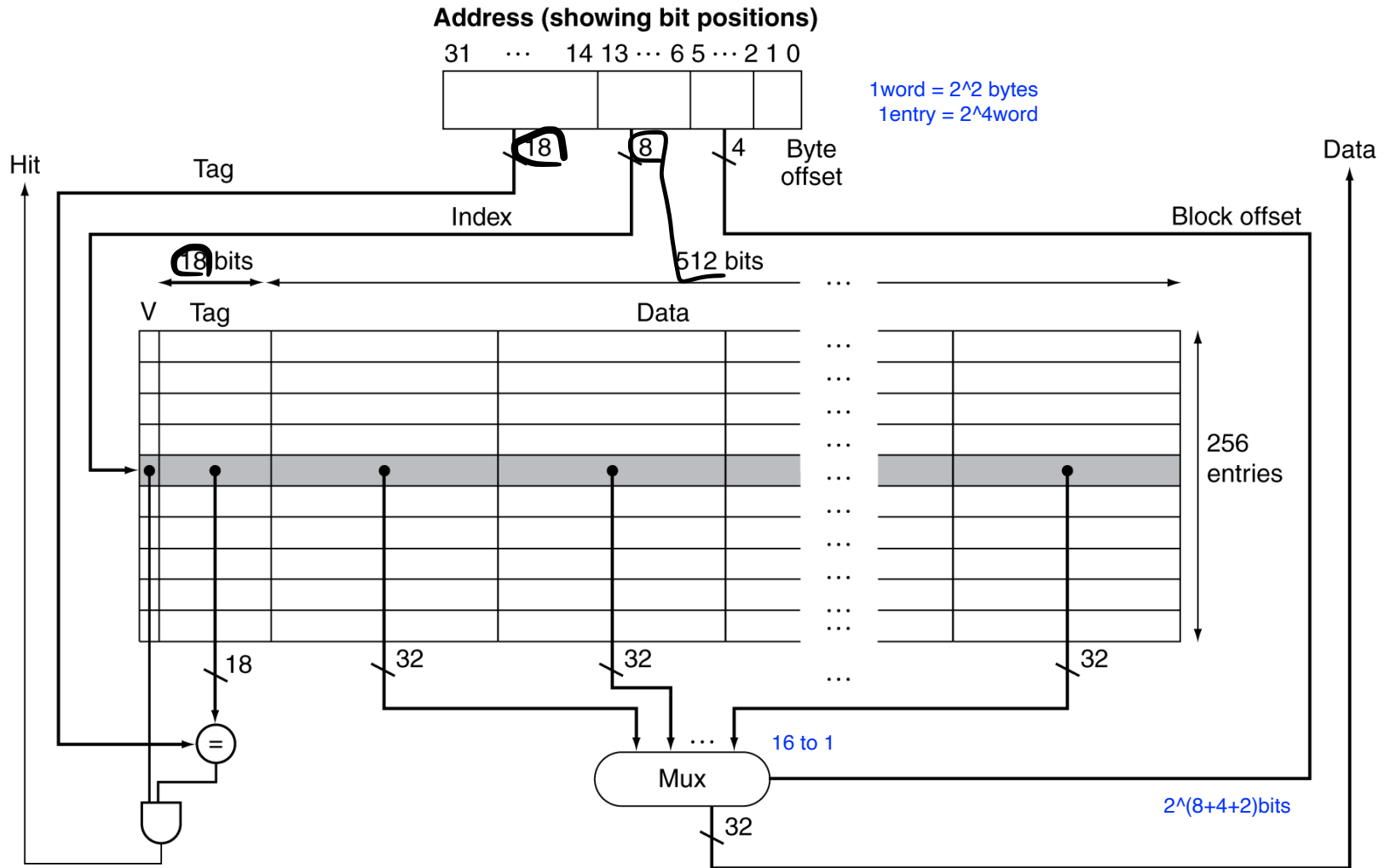


Example: Intrinsity FastMATH

- Embedded MIPS processor
 - 12-stage pipeline
 - Instruction and data access on each cycle
- Split cache: separate I-cache and D-cache
 - Each 16KB: 256 blocks \times 16 words/block
 - D-cache: write-through or write-back
- SPEC2000 miss rates
 - I-cache: 0.4% 명령어는 보통 다음 줄을 읽지만 데이터는 불확실함
 - D-cache: 11.4%
 - Weighted average: 3.2%



Example: Intrinsity FastMATH



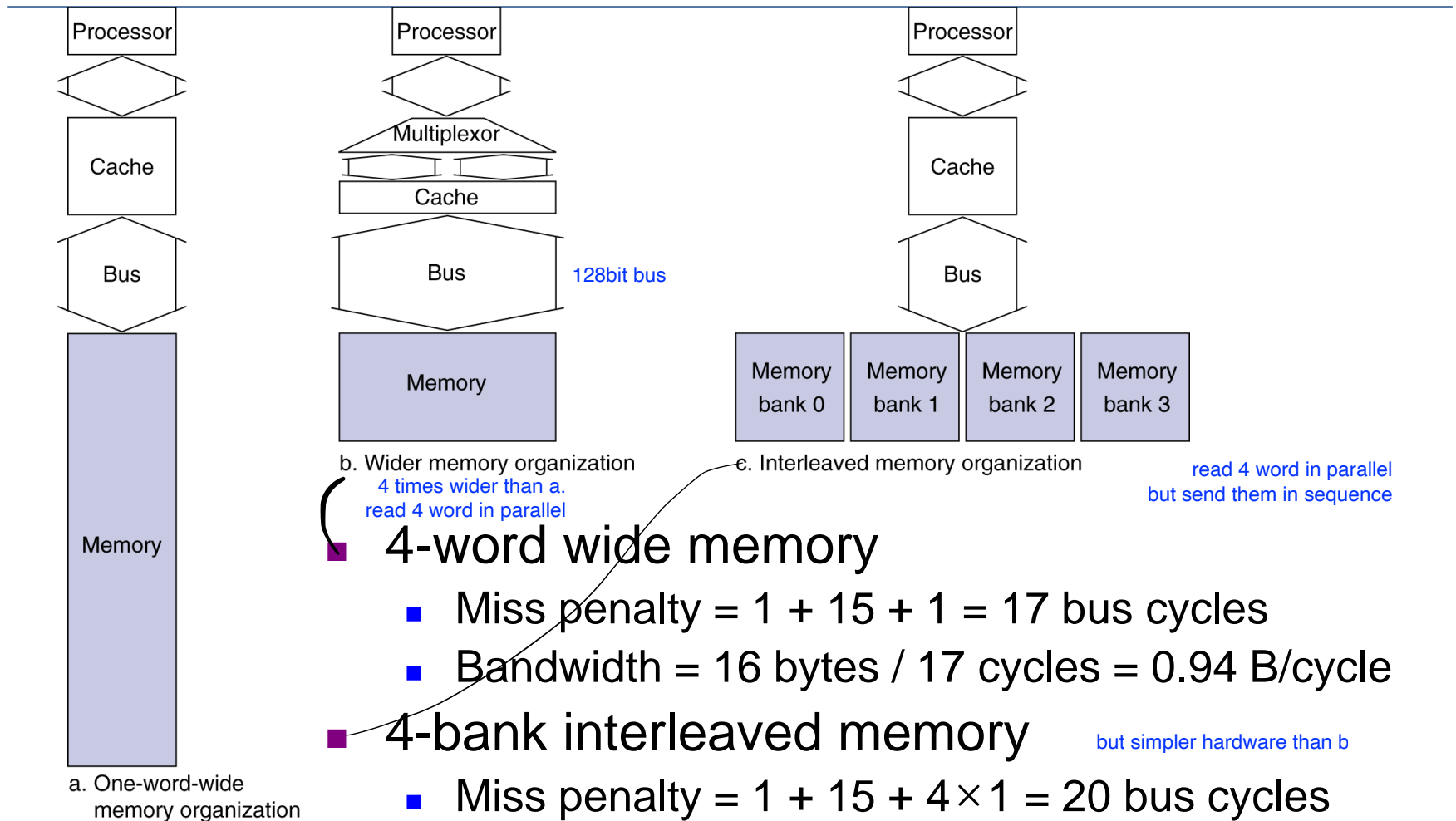
Main Memory Supporting Caches

- Use DRAMs for main memory
 - Fixed width (e.g., 1 word)
 - 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 to read one main memory block
 - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
 - Miss penalty = $1 + 4 \times 15 + 4 \times 1 = 65$ bus cycles
 - Bandwidth = $16 \text{ bytes} / 65 \text{ cycles} = 0.25 \text{ B/cycle}$

bus between cache and main memory
command, address, data bus
memory clock(bus clock)



Increasing Memory Bandwidth



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

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

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



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: $0.02 \times 100 = 2$
 - D-cache: $0.36 \times 0.04 \times 100 = 1.44$
- 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

hit time is smaller than clock time(included in base CPI)
when miss happens, need more clock cycle



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

