

① Memory hierarchy ② Cache Memory ③ Virtual Memory

Principle of Locality

가까이 있는 메모리는 빠르게 멀리 있는 메모리는 느리게 접근

- Temporal Locality : items accessed recently are likely to be accessed again soon
최근에 접근한 데이터에 다시 접근할 확률↑
ex) loop, variables
- Spatial Locality : items near those accessed recently are likely to be accessed soon
어떤 항목이 참조되면 그 근처의 다른 항목들이 참조될 확률↑

Memory Hierarchy

- 메모리 계층구조를 구현함으로써 principle of Locality를 이용할 수 있다
- 서로 다른 속도/크기를 갖는 여러 계층의 메모리로 구성

< 빠른 메모리 - 프로세서에 가깝게 - 가격↑ small space

느린 메모리 - 프로세서에서 멀게 - 가격↓ large space

↑ SRAM - DRAM - Flash - Magnetic Disk ↓

- By implementing the memory system as a hierarchy, the user has the illusion of a memory that is large as the largest level of hierarchy, but can be accessed as if it were all built from the fastest memory

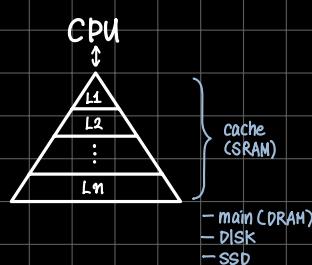
Memory Hierarchy Organization

- 프로세서에 가까운 계층의 메모리 → 멀리 있는 메모리의 subset
모든 데이터는 가장 낮은 레벨에 저장되어 있다
- 데이터는 한 번에 인접한 두 레벨 사이에서 복사된다
- The minimum unit of information that is present or not is called Block (aka line)

Terminologies

- Block : the minimum unit of information
- Hit : 프로세서가 요구한 데이터가 상위계층에 있다
- Miss : 데이터가 없다
- Hit rate, Miss Rate
- Hit Time : 메모리 계층구조의 상위계층에 접근하는 시간
접근이 hit인지 miss인지 판단하는 시간 포함
- Miss Penalty : 하위 계층에서 해당블록을 가져와 상위계층 블록과 교체하는 시간
+ 그 블록을 프로세서에 보내는데 걸리는 시간

The Basics of Caches



- Between CPU and main memory
- Takes advantage of locality of access
- Almost every general-purpose machine includes a cache

SRAM Technology - p395

DRAM Technology \rightsquigarrow volatile

- Data stored as a charge in a capacitor
전하의 형태로 capacitor에 저장
- 무한히 저장될 수 없으므로 주기적으로 리프레시 필요
 \hookrightarrow 데이터를 DRAM에 계속 유지하기 위해 계속 업데이트 해줘야

read - row 단위로 가져와서

column 단위로 읽기

write - column 단위로 쓰고

row 단위로 집어넣기

Flash Storage

- Nonvolatile semiconductor storage
 \hookrightarrow 파워를 끼우면 데이터 유지

Disk Storage

- Nonvolatile, rotating magnetic storage

Cache Memory (SRAM) - the level of the memory hierarchy closest to the CPU

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

Direct Mapped Cache 직접사상

- Location determined by Address
- 각 메모리 위치가 캐시 내의 딱 한 장소와 매치
 \hookrightarrow (Block 주소) modulo (# Blocks in Cache)

나머지 연산 ex) $11 \text{ mod } 4 = 3$

Block = 8이면 주소의 하위 3비트로 결정된다 ($2^3=8$)

Tags and Valid Bit

\rightarrow 각 캐시 엔트리는 여러 주소의 메모리 내용을 적재할 수 있다

- Tags : contain the address information required to identify whether a word in the cache corresponds to the requested word
캐시 내의 워드가 요청된 것인지 아닌지 식별하는 데 필요한 주소 정보
인덱스로 사용되지 않은 주소의 상위 비트로 구성
- Valid bit : Indicates whether an entry contains a valid address
캐시 블록이 유효한 정보를 갖고 있는지

ex)	Index	V	Tag	Data
	000	N		
	001	N		
	:	:		
	110	Y	10	Mem[10110]
	111	N		

8 blocks, (word / block, direct mapped)

- 메모리 주소 10110 (Binary Address)
Index

\rightarrow Index 해싱되는 곳 찾기

Valid bit : Yes로 바꾸고 tag 업데이트, Data 저장

- Conflict 발생할 경우? 메모리 주소 11110

\rightarrow Index 같은데 tag가 다르다

tag를 새로운 주소로 업데이트

miss ① valid bit = 0

② valid bit = 1 BUT tag가 일치하지 않을 경우

hit valid bit = 1, tag 일치

• Block size 고려하기

\rightarrow Block 단위의 원기이므로 \log_2 (Block size) 개의

하위 bit는 무시하고 읽는다

1 block = 4 byte 면 2 bit 무시

Index, Tag, and Cache Size

32 bit byte address, direct-mapped cache of size 2^n blocks with 2^m -word blocks

• block size : 2^m -word blocks $\rightarrow 2^m$ word = 2^{m+2} byte

• # Block : $2^n \rightarrow n$ bit index

\Rightarrow tag의 bit 수 : $32 - n - (m+2)$

\Rightarrow # of bits in such a cache

2^n (block size + tag size + valid field size)

$$= 2^n \times (2^m \times 32 + (32 - n - m - 2) + 1)$$

이 부분 뭐라는 건지 모르겠음

Cache bit 수 계산

제발 제발 질문 !!

ex) 64 blocks, 1b bytes / Block

address 1200

\Rightarrow offset (무시) : 4 bit

index : 6 bit

Tag : $32 - 6 - 4 = 22$ bit

Block address : $\text{floor}(1200/1b) = 75$ (offset)

Block number : $75 \bmod 64 = 11$ (index number)

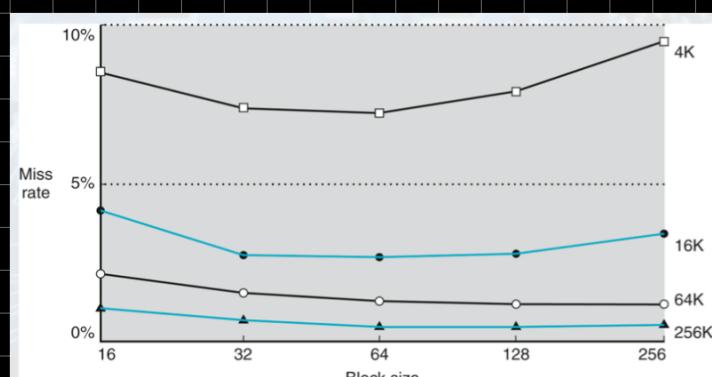
Large Blocks

• Larger blocks \rightarrow miss rate \downarrow -- spatial locality

• BUT 블록 크기가 커지면 miss penalty 증가

miss penalty = fetch a block from the memory + load it into the cache
 \hookrightarrow 첫번째 word를 찾는 지연시간 (latency) + 블록을 이동시키는 전송시간

한 블록 단위로 로딩하기 때문에
↑ 공간적 지역성이 혜택을 많이 받는다



The miss rate actually goes up because the number of blocks is too small & many memory blocks will compete for the limited number of cache blocks

Handling Cache Misses

→ Stall the entire pipeline

1. send the original PC value (PC-4) $\xrightarrow{\text{PC+4}} \text{연산이 자동 진행된 상태}$ to the memory

2. Instruct the main memory to perform a read
and wait for the memory to complete its access

메인 메모리에 읽기 동작을 지시하고 기다린다

3. Write the cache entry 메모리에서 인출한 레이어를 캐시의 레이어 부분에 준다

write the upper bits of the address (from ALU) into the tag field 태그 작성
turn the valid bit on 유효비트 1로 만들기

4. restart the instruction execution at the first step
(이제 캐시에서 명령어를 찾을 수 있다)

Handling Write

→ 레이어를 캐시에만 쓰고 메인 메모리에는 쓰지 않을 경우 : 캐시와 메인 메모리의 불일치

• write-through 즉시쓰기

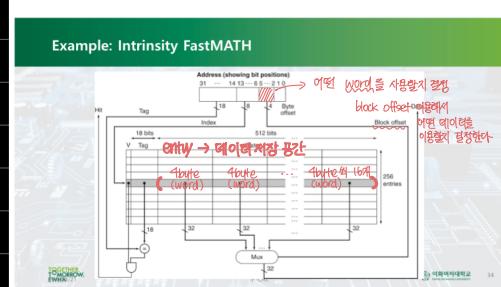
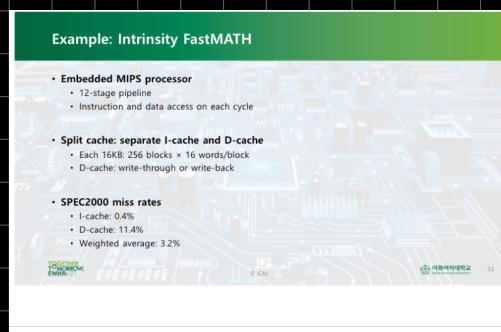
- always write data into both memory & cache

- with write buffer : 메인 메모리에 써지기 위해 기다리는 동안 레이어를 저장

• write-back 나중에 쓰기

- 일단은 캐시에만 저장

나중에 캐시에서 불록이 교체될 때 메모리에 저장된다



* I-cache : Instruction cache

D-cache : Data cache

Main Memory Supporting Caches

- Use DRAMs for main memory
 - 고정된 크기
 - Connected by fixed-width clocked bus
(Bus clock is typically slower than CPU clock)

Example Cache Block read

- 1 bus Cycle - address transfer
- 15 bus Cycle - DRAM Access *read data*
- 1 bus Cycle - data transfer *send data to cache*

send address
cache → main memory
(캐시 미스가 발생한 경우)

DRAM이 1word 크기로 고정 (fixed width)

cycles

⇒ For 4-word block, 1-word-wide DRAM (Miss Penalty = 1 + 4 × 15 + 4 × 1 = 65 bus cycles)
→ 4word in 1 cache block
즉, main memory에서 4word를 읽어야 한다

메모리에서 한번에 4word 읽어오므로 1×15

⇒ band width from memory to cache 증가하면 miss penalty 감소한다 대역폭↑ miss penalty ↓

• 4-word-wide Memory (Miss penalty : 1 + 15 + 1 = 17 bus cycles)

① 메모리폭 확장 4word

• 4-bank interleaved memory (Miss penalty : 1 + 15 + 4 × 1 = 20 bus cycles)
② interleaving

Band width : 16 bytes / 20 cycles = 0.8 B/cycle

Memory Interleaving

- memory chips - organized in banks to read/write multiple words in one access time
- Each bank could be one word wide
 - so that the width of bus and the cache need not change,
 - but sending an address to several banks permits them all to read simultaneously
- Each bank can write independently, quadrupling the write bandwidth in a write-through cache

Byte Addressable Memory

- address of a word
 - byte address의 4개 주소 (1word = 4byte) 중 하나와 매치
 - most architectures address individual bytes
- address of sequential words differ by 4

Measuring and Improving Cache Performance

캐시 성능을 향상시키는 법

① miss rate ↓

by reducing the probability that two different memory blocks will contend for the same location

② miss penalty ↓

by adding an additional level to hierarchy
⇒ Multilevel caching

Cache Performance

- CPU Time = (CPU execution clock cycle + memory stall clock cycle) × clock cycle time
- Memory stall clock cycles = read-stall clock cycles + write-stall clock cycles
→ read-stall clock cycles
 - : $(\text{read}/\text{program}) \times \text{read miss rate} \times \text{Read miss penalty}$ 프로그램 내 read 명령어의 수
 - write-stall cycles (for write-through)
 - : $(\text{writes}/\text{program}) \times \text{write miss rate} \times \text{write miss penalty}$ (write buffer stall) 제거하고 설명해주세요

Memory Stall Cycles

대부분의 캐시에서 read penalties = write penalties (메모리에서 블록을 가져오는데 걸리는 시간)

- 쓰기 비呸지연이 무시할 수 있을 정도로 작다고 가정 ... 읽기와 쓰기를 합쳐서 표현

$$\boxed{\text{Memory Stall Cycles} = \frac{\text{Memory Access}}{\text{Program}} \times \text{miss rate} \times \text{miss penalty}}$$

misses/instruction

Cache Performance Example

- Miss Rates (Instruction cache miss rate : 2%
Data cache miss rate : 4%)
- 메모리 지연 없을 때 CPI = 2
- Miss Penalty = 100 cycles for all misses
- 읽기 (loads) 와 쓰기 (stores)의 비율 : 36:1

⇒ 실패가 발생하지 않는 완벽한 캐시에서 시스템이 얼마나 빨라지는지 계산하라

- Instruction miss cycles : $I \times 2\% \times 100 = 2.00 \times I$
명령어에 대한 실패
miss penalty
Instruction cache miss rate
- Data miss cycles : $I \times 36\% \times 4\% \times 100 = 1.44 \times I$
데이터 참조에 대한 실패
load/store 비율

- Total # of memory-stall cycles : $3.44 \times I$

- Total CPI : 5.44

↳ 전체 메모리 - 지연 사이클 수 $3.44I$

메모리 지연 고려할 경우 $(2 + 3.44)I$

- $\frac{\text{지연 있는 경우 CPU 시간}}{\text{완벽한 캐시의 CPU 시간}} = \frac{I \times \text{CPI(stall)} \times \text{clock cycle}}{I \times \text{CPI(pfect)} \times \text{clock cycle}} = \frac{5.44}{2} = 2.72$

Cache Performance with Increased Clock Rate

프로세서가 더 빠르게 동작하면 ... relative cache penalties ↑

If a machine improves both clock rate and CPI ... suffers a double hit

→ lower the CPI → impact of stall cycles ↑

↳ CPI가 2에서 1로 감소하면 # of memory stall cycles = $4.44 \times I$ cycles

→ 두 기계의 메인 메모리가 same absolute access time을 가진다면

CPU clock Rate ↑ larger miss penalty CPU Time (밀초) = clock cycles × clock time

↳ Clock Time (1 sec → 0.5 sec) 이라 하면 Clock cycle (100 → 200)

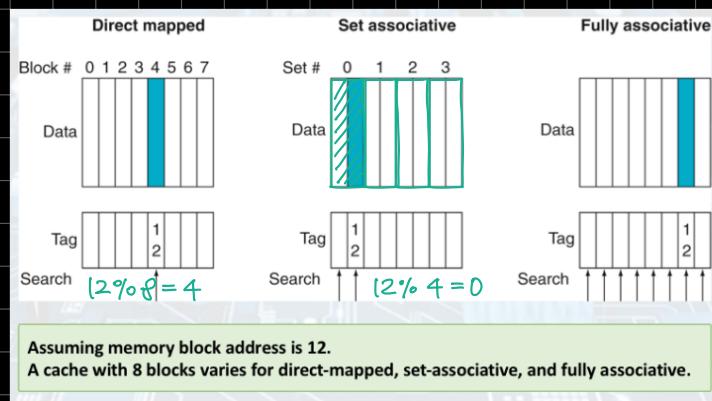
시비
애이익는지
모르겠음
clock rate

Fully Associative Cache

- : reducing cache misses by more flexible placement of blocks
 - Direct mapped cache — 블록이 캐시의 지정된 한 곳에만 들어갈 수 있다
 - Fully Associative cache — 블록이 캐시 내의 어느 곳에나 들어갈 수 있다
→ 주어진 블록을 찾기 위해서는 캐시내의 모든 entry를 검색해야 한다.

Set Associative Cache

- fully associative 방식에서는
 - 모든 entry 검색
 - 각 캐시 엔트리와 연결된 comparator를 이용하여 병렬로 검색
- set associative cache
 - : 한 블록이 들어갈 수 있는 자리의 수가 고정되어 있다



N-Way Set - Associative Cache

- : set associative cache with n locations for a block
- 각 블록은 index field가 지정하는 캐시 안의 a unique set로 mapped
- 직접사상과 완전연관의 combination
→ set와 직접 mapped, set 내에서 블록 검색

Position of a Memory Block

< 직접사상 : (블록번호) modulo (캐시 내 블록의 수)
< 집합연관 : (블록번호) modulo (캐시 내 set의 수)

Tag Searching

- < 완전연관 : 모든 blocks 검색
< 집합연관 : set 내의 blocks만 검색

associativity가 증가하면 set의 수는 감소, set 당 element 수는 증가

직접사상 8 blocks

집합사상 4set x 2blocks

2set x 4blocks

:

- 직접사상 - 1 way set associative
- 집합사상 (1 set on m blocks) - m way set associative
fully associative with m entries
- Increasing degree of Associativity (장: miss rate ↓
단: hit time ↑)

Example : Misses and Associativity in Cache

- three small caches : 직접사상, 2-way 집합연관, 완전연관
 - each consisting of four one-word blocks
- ⇒ find the number of misses : address sequence 0, 8, 0, 6, 8

① 직접사상 : $0 \rightarrow 8 \rightarrow 0 \rightarrow 6 \rightarrow 8$

$$0 \% 4 = 0$$

$$6 \% 4 = 2$$

$$8 \% 4 = 0$$

M - M - M - M - M

② 집합연관 : $0 \% 2 = 0$

$$6 \% 2 = 0$$

$$8 \% 2 = 0$$

M - M - H - M - M

③ 완전연관 : M - M - H - M - H

Locating a Block in Cache

- Each block in set-associative cache includes an address tag
→ 선택된 집합set 내의 모든 블록의 태그 - 프로세서에서 나온 태그와 일치하는지 검사
- associativity ↑ #of blocks in set ↑
그냥 associativity = 한 세트의 블록 수라고 보면 될듯

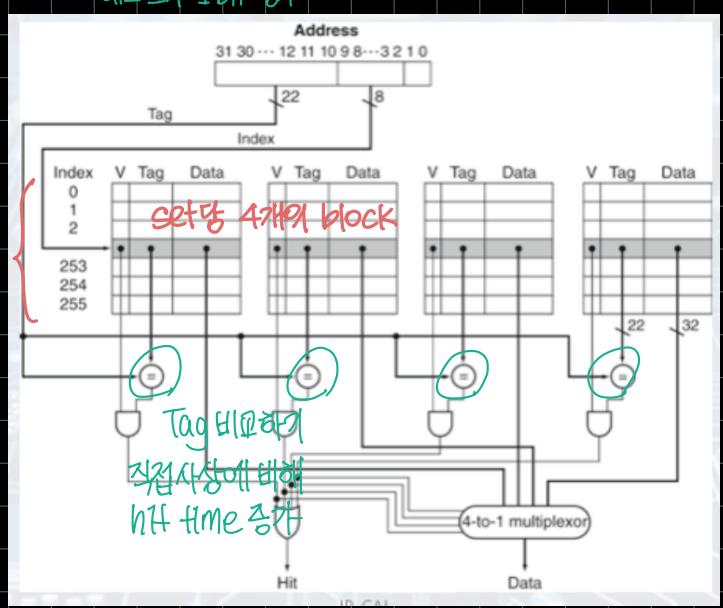
Three Portions of an Address for set Associative

Tag	Index	offset
31	0	

저장할 수 있는 Block 크기
association 2배 증가 ⇒ 인덱스크기 1bit 감소
태그크기 1bit 증가

- Index - select the set
- Tag - 선택된 set 내에서 특정 block을 검색
- Block Offset - Block 크기로 무시하는 bit

256
set
↓
인덱스 8bit



Choosing Which Block to Replace

- Direct Mapped Cache : 인덱스만을 사용해 블록에 접근 가능
- N-way Set Associative : 인덱스 \rightarrow set 지정
miss 일 경우 set 내의 아무 곳이나 들어갈 수 있다
- Fully Associative : Block can be written into any position

\Rightarrow Incoming block을 어디에 write 해야 하는지??

Block Replacement Policy

- Valid bit : off (= empty) 인 블록이 있다면 그곳에 작성
- 모든 블록이 valid : on 이라면 replace block을 결정해야 한다
"cached out" block의 선택

LRU : Least - Recently Used

- 사용된지 가장 오래된 블록부터 cached out
- set 안의 element의 사용시기를 추적하여 구현 가능

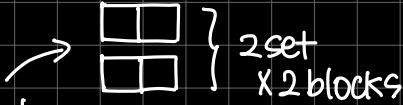
<장점> ① recent use \rightarrow 미래에도 사용 가능성↑ : Temporal locality
② effective

<단점> ① 2-way set associative \rightarrow easy to keep track ... one LRU bit
② $4 \leq m$, 연관정도 커질수록 원소추적 어렵다
requires complicated hardware, much time

Random replacement Policy - high associativity의 경우 LRU와 비슷하다

Block Replacement Example

- 2-way associative set
- four word total capacity, one word block
- (word) address : 0 2 0 1 4 0 2 3 5 4



\Rightarrow LRU 방식을 사용할 때 hit/miss?

0	2	0	1	4	0	2	3	5	4
0	0	0	1	0	0	0	1	1	0
M	M	H	M	M	H	M	M	M	M

mod 연산으로 인덱스 구하기 ... % 0 2

2지우고 4지우고 1지우고
4저장 2저장 5저장

Multilevel Caches

- Primary cache attached to CPU : small & fast
- Level 2 cache - services misses from primary cache
 - primary cache에서 miss 발생 시
: primary cache에 비해 크고 느리다
- Main memory - services L2 cache misses
- ⋮
- some high-end system include L3 cache

Multi Level Cache Example

- A base of CPI of 1.0 with perfect cache (모든 참조가 hit일 경우 CPI가 1.0이란 뜻)
- Clock rate of 4GHz
- Main memory access time 100ns (실패 처리까지 포함한 시간)
- Miss rate per instruction at primary cache : 2%
 - ⇒ How much faster will the processor be if we add a secondary cache that has a 5ns access time? - It is large enough to reduce the miss rate to main memory to 0.5%.

$$\text{clock time} = \frac{1}{\text{clock rate}} = \frac{1}{4\text{GHz}} = \frac{1}{4 \times 10^9 \text{Hz}} = 0.25 \text{ ns}$$

primary cache 만 있을 경우

- miss penalty : 100ns / clock time = 400 cycles
- effective CPI : $1 + 0.02 \times 400 = 1 + 8 = 9$
 - base CPI 1.0
with a perfect cache
(디플트로 가져가는 CC...?)
 - miss rate x miss penalty

with L1 & L2 cache

- miss penalty to L2 cache : 5ns / 0.25ns = 20 cycles
- Effective CPI = $1 + 0.02 \times 20 + 0.005 \times 400 = 1 + 0.4 + 2 = 3.4$
 - L1 → L2
 - L2 → main memory

$$\Rightarrow \text{performance ratio } 9/3.4 = 2.6$$

Multi Level Cache Considerations

- Primary cache - focus on minimal hit time
- L-2 cache - focus on low miss rate to avoid main memory access (hit time has less overall impact)
 - ⇒ L-1) usually smaller than a single cache ↗ 디만체 아닌 경우
 - L-1 block size smaller than L-2 cache
 - * L-1은 속도, L-2는 실패율 ↓에 집중

Virtual Memory

- Use main memory as a "cache" for secondary (disk) storage managed jointly by CPU hardware & OS
- programs share main memory
 - 각 프로그램은 private virtual address space 를 갖는다
↳ 자주 사용하는 code, data 저장
 - protected from other programs
- CPU and OS translate virtual addresses to physical address
 - VM "block" 를 page 라고 한다 ... page 단위
 - VM translation "miss" 를 a page fault 라 한다
virtual address 를 physical address 로 translate
대응되는 physical address 가 없으면 DISK address에 접근하여 읽어야 하는데
이것을 page fault 라고 한다