Computer Architecture
-----A Quantitative Approach

陈文智

浙江大学计算机学院

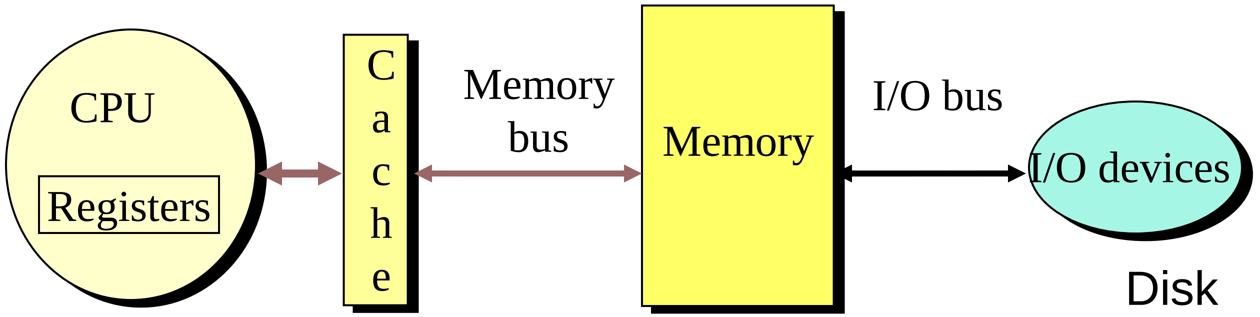
chenwz@zju. edu. cn

Computer Architecture

Chapter 5 Memory - Hierarchy Design

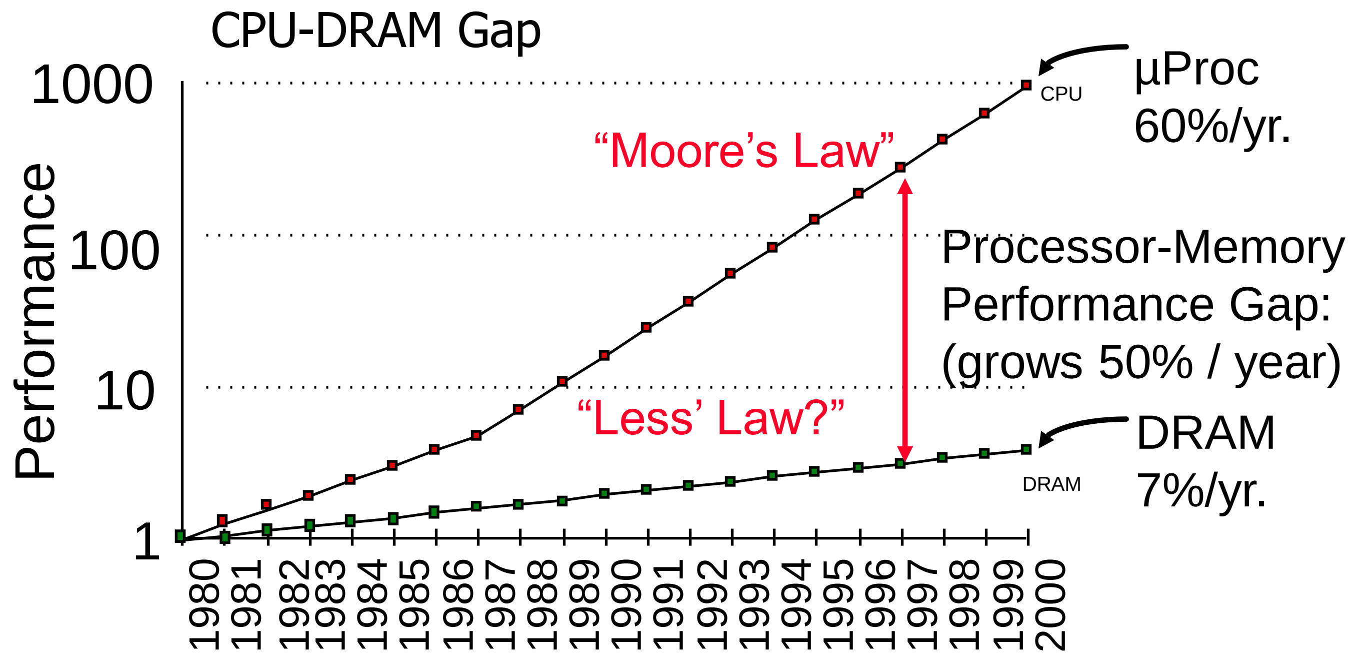
5.1 Introduction

- Processor-Memory Performance Gap



	Register reference	Cache reference	Memory reference	Disk Memory reference
Size:	500B	64KB	512MB	100GB
Speed:	0.25ns	1ns	100ns	5ms

Who Cares About the Memory Hierarchy?



- 1980: no cache in µproc; 1995 2-level cache on chip (1989 first Intel µproc with a cache on chip)

Three classes of computers With different concerns in memory hierarchy

- Desktop computers:
  - Are primarily running one application for single user
  - Are concerned more with average latency from the memory hierarchy.
- Server computers:
  - May typically have hundreds of users running potentially dozens of applications simultaneously.
  - Are concerned about memory bandwidth.
- Embedded computers:
  - Real-time applications.
    - Worst-case performance vs Best case performance
  - Are concerned more about power and battery life.
    - Hardware vs software
  - Running single app & use simple OS
    - The protection role of the memory hierarchy is often diminished.
  - Main memory is very small
    - often no disk storage

Enhance speed of memory

Component character of hardware:

- Smaller hardware is faster and more expensive
- Bigger memories are lower and cheaper

- The goal:
  - There are speed of smallest memory and capacity of biggest memory
  - To provide cost almost as low as the cheapest level of memory and speed almost as fast as the fastest level.

The method enhance speed of memory

By taking advantage of the principle of locality:

- **most programs do not access all code or data uniformly**
- Temporal Locality (Locality in Time):
- If an item is referenced, the same item will tend to be referenced again soon
  - Keep most recently accessed data items closer to the processor
- Spatial Locality (Locality in Space):
- If an item is referenced, nearby items will tend to be referenced soon
  - Move recently accessed groups of contiguous words(block) closer to processor.

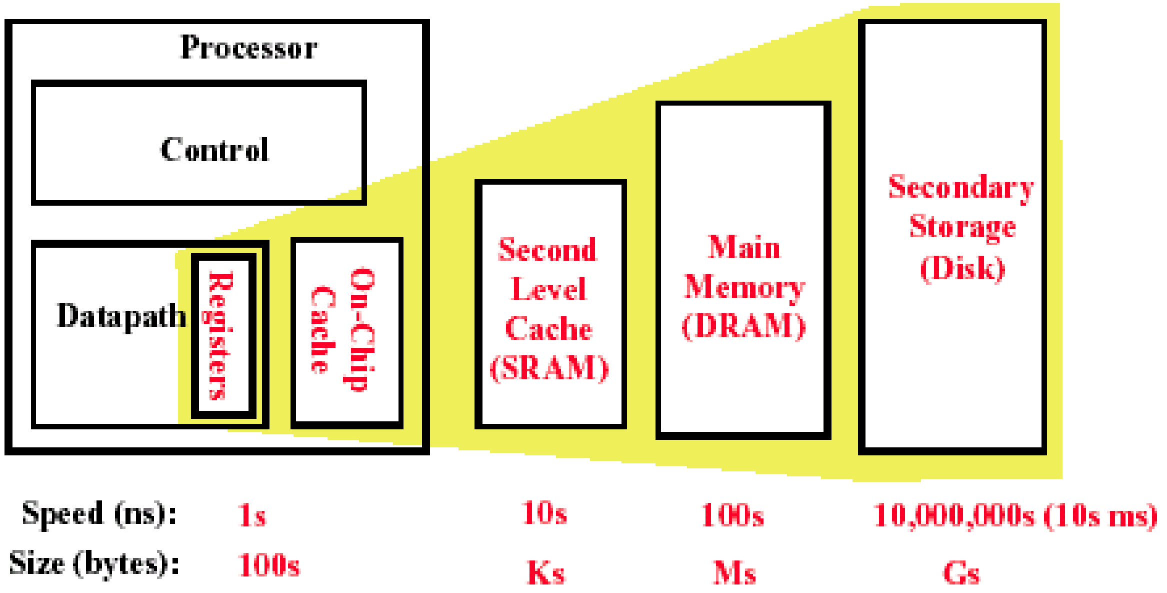
- The method
  - Hierarchies bases on memories of different speeds and size
  - The more closely CPU the level is, the faster the one is.
  - The more closely CPU the level is, the smaller the one is.
  - The more closely CPU the level is, the more expensive one is.

Memory Hierarchy of a Modern Computer System

By taking advantage of the principle of locality:

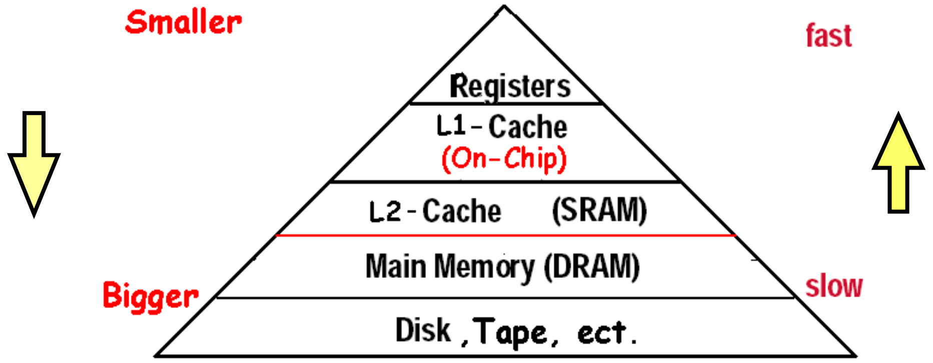
. Present the user with as much memory as is available in the cheapest technology.

. Provide access at the speed offered by the fastest technology.



What is a cache?

- Small, fast storage used to improve average access time to slow memory.
- In computer architecture, almost everything is a cache!
  - Registers “a cache” on variables – software managed
  - First-level cache a cache on second-level cache
  - Second-level cache a cache on memory
  - Memory a cache on disk (virtual memory)
  - TLB a cache on page table
  - Branch-prediction a cache on prediction information?



5.2 Review of the ABCs of Caches

36 terms of Cache

Cache	full associative	write allocate
Virtual memory	dirty bit	unified cache
Memory stall cycles	block	block offset
misses per instruction	direct mapped	write back
Valid bit	data cache	locality
Block address	hit time	address trace
Write through	cache miss	set
Instruction cache	page fault	miss rate
random replacememt	index field	cache hit
Average memory access time	page	tag field
n-way set associative	no-write allocate	miss penalty
Least-recently used	write buffer	write stall

Four Questions for Memory Hierarchy Designers

Caching is a general concept used in processors, operating systems, file systems, and applications.

There are **Four Questions** for Memory Hierarchy Designers

- Q1: Where can a block be placed in the upper level? *(Block placement)*
  - Fully Associative, Set Associative, Direct Mapped
- Q2: How is a block found if it is in the upper level? *(Block identification)*
  - Tag/Block
- Q3: Which block should be replaced on a miss? *(Block replacement)*
  - Random, LRU,FIFO
- Q4: What happens on a write? *(Write strategy)*
  - Write Back or Write Through (with Write Buffer)

Q1: Block Placement

- Direct mapped
  - Block can only go in one place in the cache

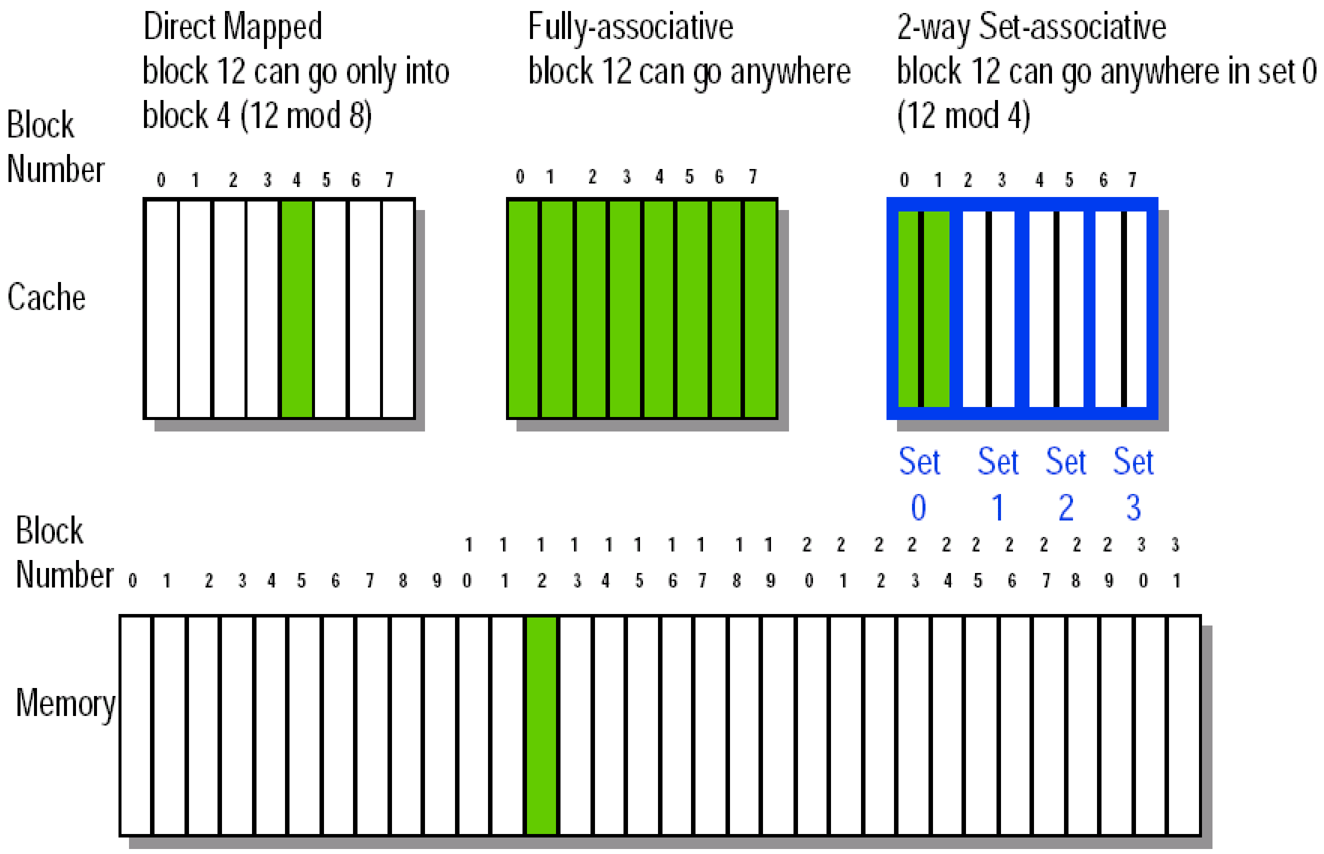
    Usually address MOD Number of blocks in cache
- Fully associative

  Block can go anywhere in cache.
- Set associative
  - Block can go in one of a set of places in the cache.
  - A set is a group of blocks in the cache.

    Block address MOD Number of *sets* in the cache
  - If sets have n blocks, the cache is said to be n-way set associative.

•*Note that direct mapped is the same as 1-way set associative, and fully associative is m-way set-associative (for a cache with m blocks).*

Figure5.4 8-32 Block Placement



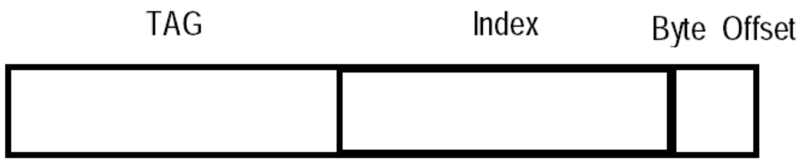
Q2: Block Identification

- Every block has an **address tag** that stores the main memory address of the data stored in the block.

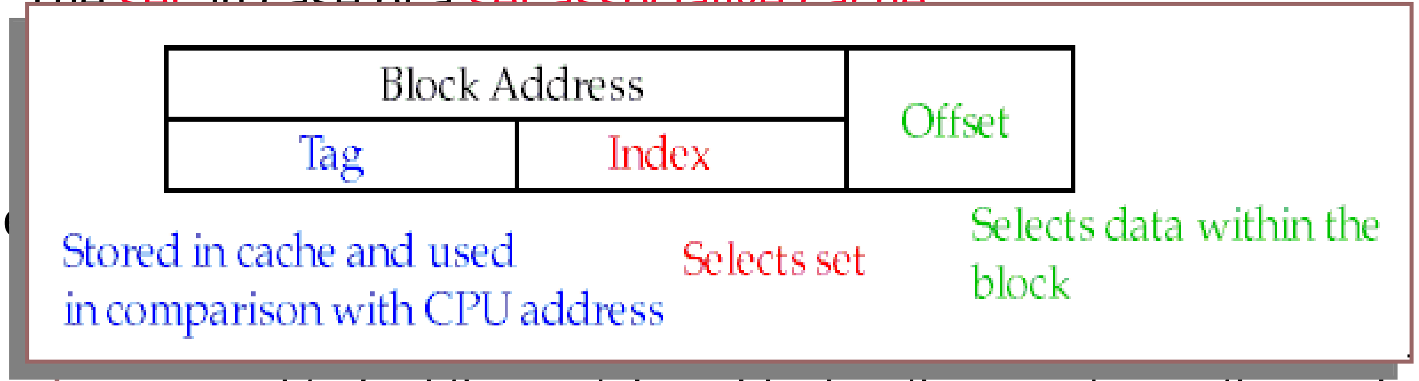
- When checking the cache, the processor will **compare** the requested **memory address to the cache tag** -- if the two are equal, then there is a cache hit and the data is present in the cache

- Often, each cache block also has a **valid bit** that tells if the contents of the cache block are valid

The Format of the Physical Address



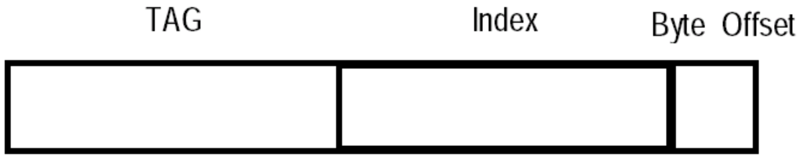
- The Index field selects
  - The set, in case of a set associative cache
  -
  -
- Th
  -
  -

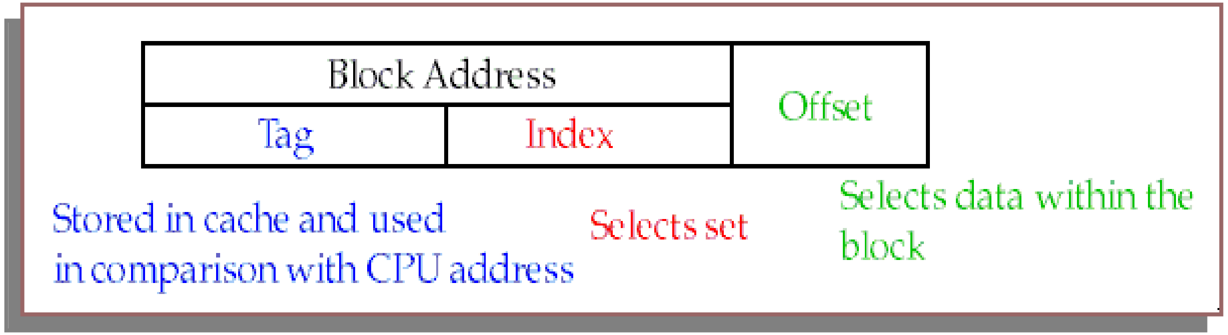

Block Address
Tag
Index
Offset
Stored in cache and used in comparison with CPU address
Selects set
Selects data within the block

- The Tag is used to find the matching block within a set or in the cache
  - Has as many bits as Address_size – Index_size – Byte_Offset_Size

The Format of the Physical Address

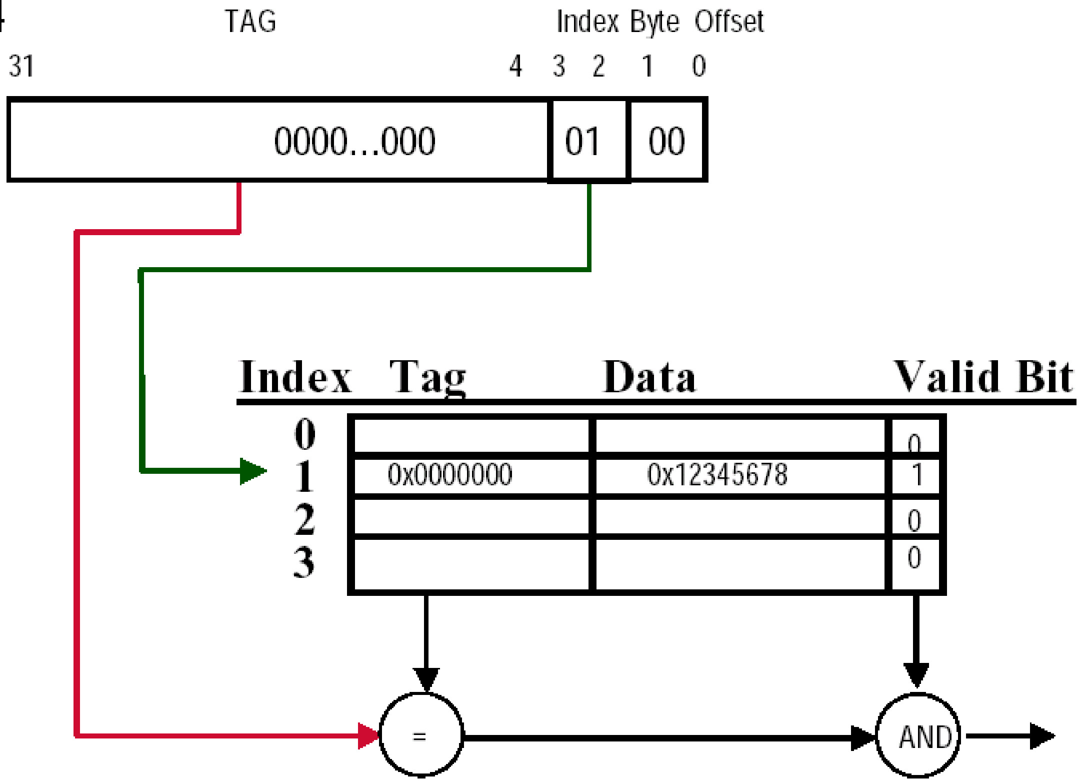
- The Index field selects
  - The set, in case of a set-associative cache
  - The block, in case of a direct-mapped cache
  - Has as many bits as log2(#sets) for set-associative caches, or log2(#blocks) for direct-mapped caches
- The Byte Offset field selects
  - The byte within the block
  - Has as many bits as log2(size of block)
- The Tag is used to find the matching block within a set or in the cache
  - Has as many bits as Address_size – Index_size – Byte_Offset_Size




Block Address
Offset
Tag
Index
Stored in cache and used in comparison with CPU address
Selects set
Selects data within the block

Direct-mapped Cache Example (1-word Blocks)

LOAD R1, 0x04

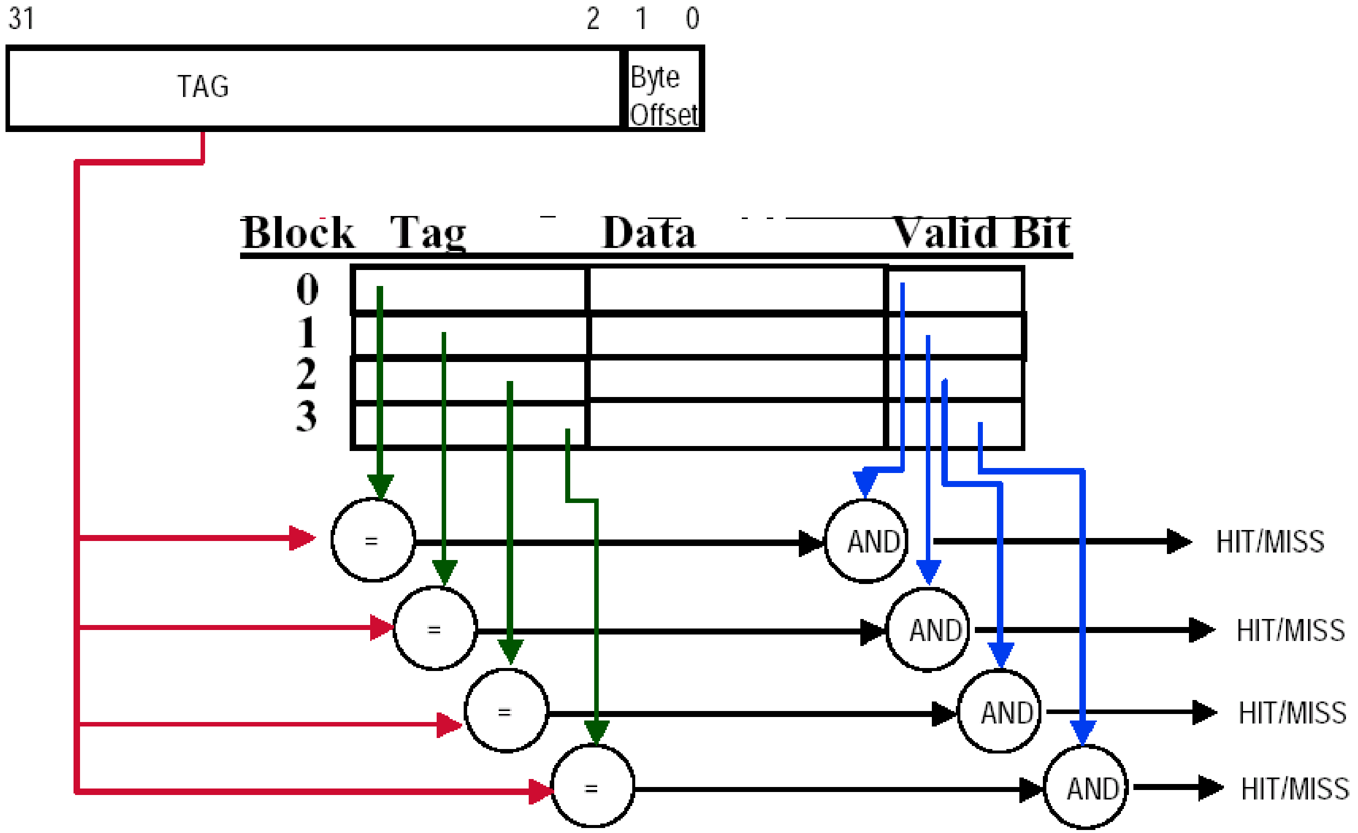


MEMORY

Address	Data
0x00	0x00000000
0x04	0x12345678
0x08	0x87654321
0x0C	0x11111111
0x10	0x22222222
0x14	0x33333333
0x18	0x44444444
0x1C	0x55555555
0x20	0x10101010

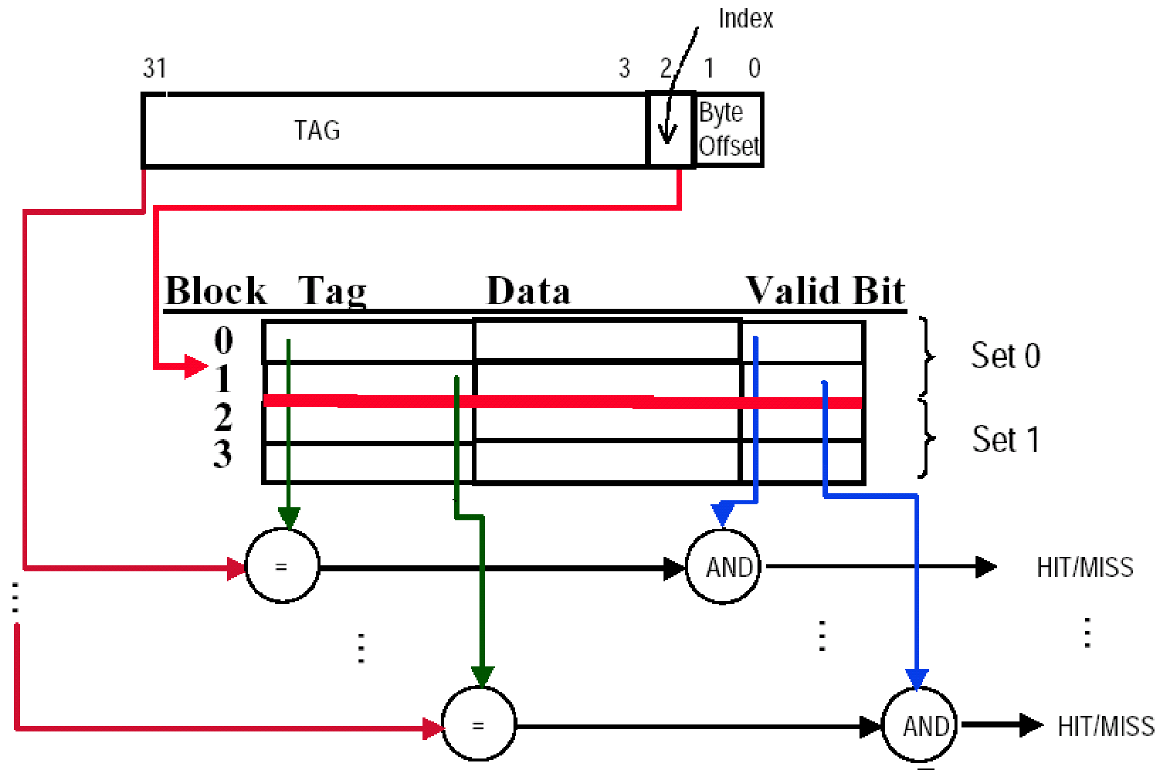
Fully-Associative Cache example (1-word Blocks)

- Assume cache has 4 blocks



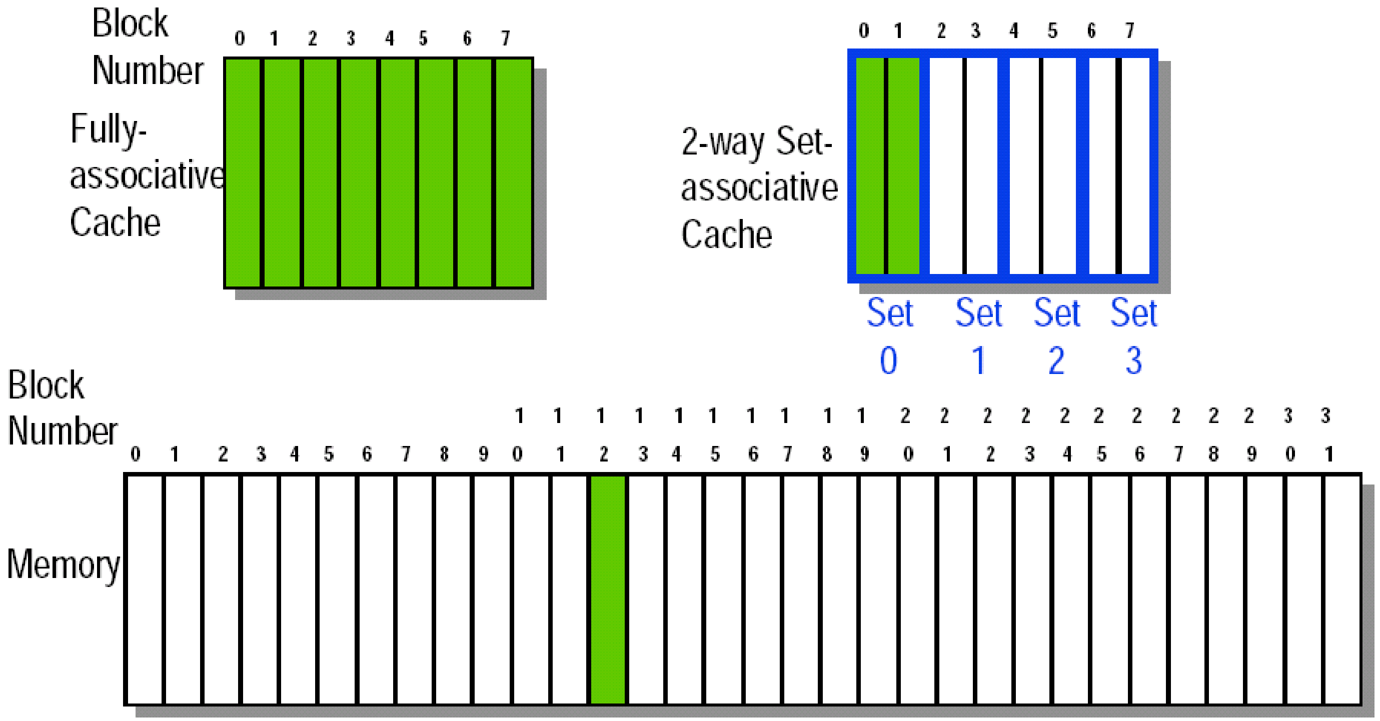
2-Way Set-Associative Cache

- Assume cache has 4 blocks and each block is 1 word
- 2 blocks per set, hence 2 sets per cache



Q3: Block Replacement

- In a direct-mapped cache, there is only one block that can be replaced
- In set-associative and fully-associative caches, there are N blocks (where N is the degree of associativity



Strategy of block Replacement

- Several different replacement policies can be used
  - Random replacement - *randomly pick any block*
    - Easy to implement in hardware, just requires a random number generator
    - Spreads allocation uniformly across cache
    - May evict a block that is about to be accessed
  - Least-recently used (LRU) - *pick the block in the set which was least recently accessed*
    - Assumed more recently accessed blocks more likely to be referenced again
    - This requires extra bits in the cache to keep track of accesses.
  - First in,first out(FIFO)-*Choose a block from the set which was first came into the cache*

Q4: Write Strategy

- When data is written into the cache (on a store), is the data also written to main memory?
  - If the data is written to memory, the cache is called a *write-through cache*
    - Can always discard cached data - most up-to-date data is in memory
    - Cache control bit: only a *valid* bit
    - memory (or other processors) always have latest data
  - If the data is NOT written to memory, the cache is called a *write-back cache*
    - Can't just discard cached data - may have to write it back to memory
    - Cache control bits: both *valid* and *dirty* bits
    - much lower bandwidth, since data often overwritten multiple times

- Write-through adv: Read misses don't result in writes, memory hierarchy is **consistent** and it is simple to implement.
- Write back adv: Writes occur at speed of cache and main memory bandwidth is smaller when multiple writes occur to the same block.

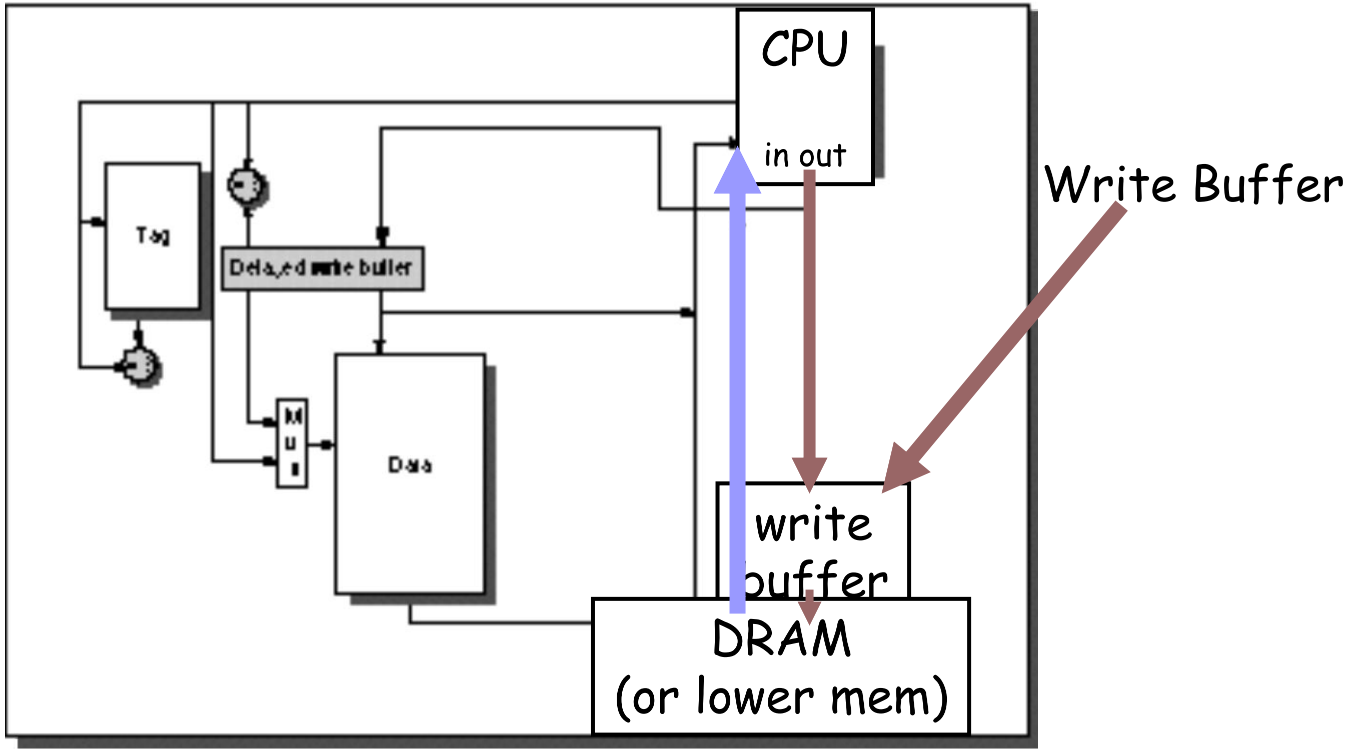
Write stall

- **Write stall** ---When the CPU must wait for writes to complete during write through

- Write buffers
  - A small cache that can hold a few values waiting to go to main memory.
  - To avoid stalling on writes, many CPUs use a write buffer.

  - This buffer helps when writes are clustered.
  - It does not entirely eliminate stalls since it is possible for the buffer to fill if the burst is larger than the buffer.

Write buffers



Write misses

- Write misses
  - If a miss occurs on a write (the block is not present), there are two options.
  - Write allocate
    - The block is loaded into the cache on a miss before anything else occurs.
  - Write around (no write allocate)
    - The block is only written to main memory
    - It is not stored in the cache.

  - In general, write-back caches use write-allocate , and write-through caches use write-around .

Example

- Assume a fully associative write-back cache with many cache entries that starts empty. Below is a sequence of five memory operations(the address is in square brackets):

1	write Mem[100];
2	write Mem[100];
3	Read Mem[200];
4	write Mem[200];
5	write Mem[100];

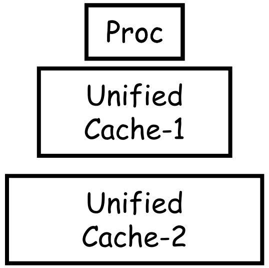
What are the number of hits and misses when using no-write allocate versus write allocate?

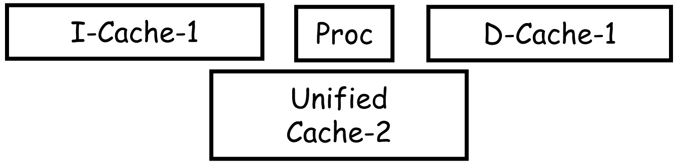
Answer :

for no-write allocate	misses: 1,2,3,5
	hit: 4
for write allocate	misses: 1,3
	hit: 2,4,5

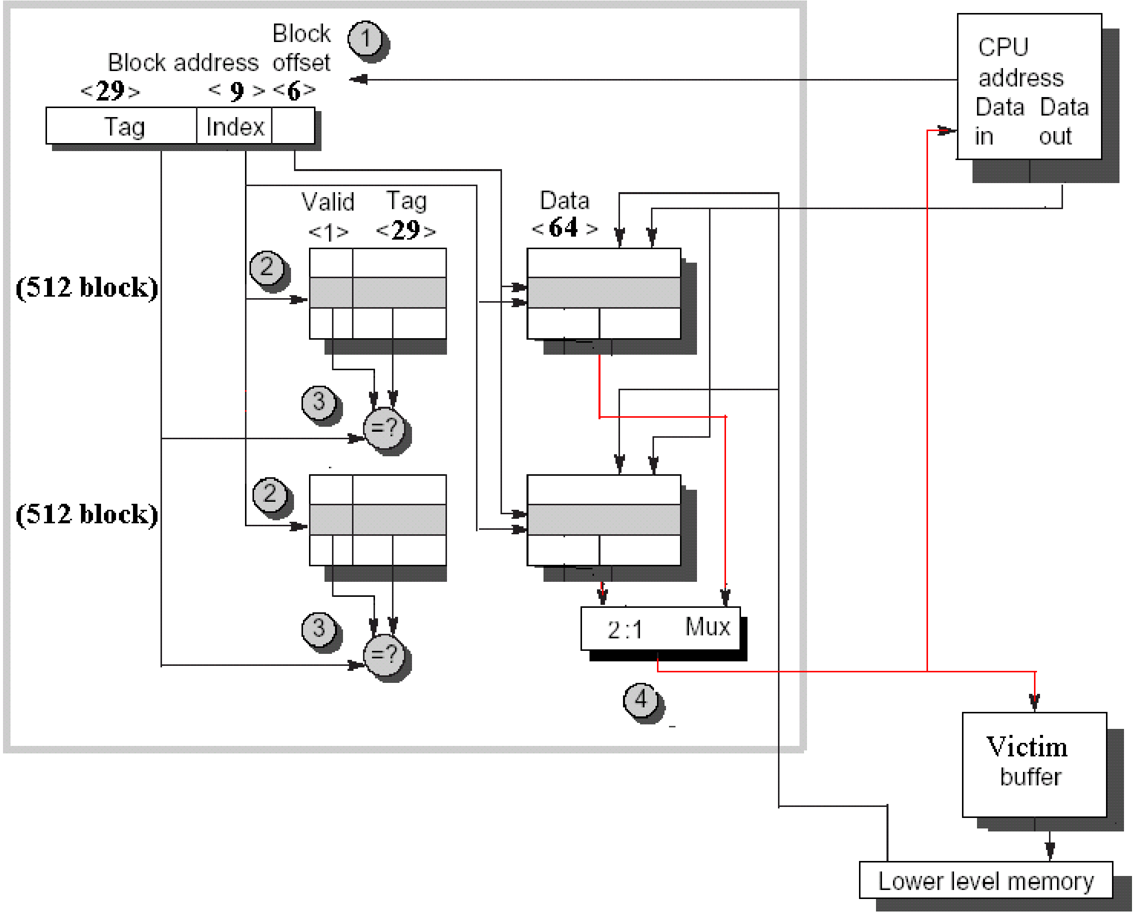
Split vs. unified caches

- Unified cache
  - All memory requests go through a single cache.
  - This requires less hardware, but also has lower performance
- Split I & D cache
  - A separate cache is used for instructions and data.
  - This uses additional hardware, though there are some simplifications (the I cache is read-only).





An example :the Alpha 21264 data cache



An example :the Alpha 21264 data cache

Step1 Cache is divided into 2 fields: the 38 bit block address and the 6-bit block offset(64=$2^6$and 38+6=44).

$$2^{\text{index}} = \frac{\text{Cache size}}{\text{Block size} \times \text{Set associativity}}$$

$$= \frac{65{,}536}{64 \times 2} = 512 = 2^9$$

Step2 *Index selection ,Be reading the two tags from cache.*

Step3 the two tags are compared and the winner is selected. Tag contains valid bit, else the results of the comparison are ignored.

Step4 If one tag does match, CPU loads the proper data from the cache, else from main memory.

The 21264 allows 3 clock cycles for these four steps, so the instructions in the following 2 clock cycles would wait if they tried to use the result of the load.

5.3 Cache performance

Memory System Performance

- CPU Execution time

CPU Execution time = (CPU clock cycles + Memory stall cycles) × Clock cycle time

$$\text{Memory stall cycles} = \text{IC} \times \text{Mem refs per instruction} \times \text{Miss rate} \times \text{Miss penalty}$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemAccess}{Inst} \times MissRate \times MissPenalty \right) \times CycleTime$$

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{MemMisses}{Inst} \times MissPenalty \right) \times CycleTime$$

$\text{CPI}_{\text{Execution}}$ includes ALU and Memory instructions

Average Memory Access Time

- Average Memory Access Time

$$\text{Average Memory Access Time} = \frac{\text{Whole accesses time}}{\text{All memory accesses in program}}$$

$$= \frac{\text{Accesses time on hitting} + \text{Accesses time on miss}}{\text{All memory accesses in program}}$$

$$= \text{Hit time} + (\text{Miss Rate} \times \text{Miss Penalty})$$

$$= \left(HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst}\right) \times Inst\%$$

$$\left(HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data}\right) \times Data\%$$

$$\boldsymbol{CPUtime = IC \times \left(\frac{AluOps}{Inst} \times CPI_{AluOps} + \frac{MemAccess}{Inst} \times AMAT\right) \times CycleTime}$$

Example1: Impact on Performance

- Suppose a processor executes at
  - Clock Rate = 200 MHz (5 ns per cycle), Ideal (no misses) CPI = 1.1
  - 50% arith/logic, 30% ld/st, 20% control
- Suppose that 10% of memory operations get 50 cycle miss penalty
- Suppose that 1% of instructions get same miss penalty
- What is the CPUtime and the AMAT ?

- Answer: CPI = ideal CPI + average stalls per instruction
  = 1.1(cycles/ins) +
  [0.30 (DataMops/ins)
  x 0.10 (miss/DataMop) x 50 (cycle/miss)]
  + [1 (InstMop/ins)
  x 0.01 (miss/InstMop) x 50 (cycle/miss)]
  = (1.1 + 1.5 + .5) cycle/ins = 3.1
- AMAT=(1/1.3)x[1+0.01x50]+(0.3/1.3)x[1+0.1x50]=2.54

Example2: Impact on Performance

Assume (p395): Ideal CPI=1 (no misses)

- L/S's structure . 50% of instructions are data accesses
- Miss penalty is 25 clock cycles
- Miss rate is 2%
- How faster would the computer be if all instructions were cache hits?

- Answer: first compute the performance for always hits:

$\text{CPU}_{\text{execution time}}$ =(CPU clock cycles+memory stall cycles) × clock cycle

=(IC × CPI+0) × Clock cycle

=IC × 1.0 × clock cycle

Now for the computer with the real cache,first compute memory stall cycles:

$$Memory\ stall\ \ cycles = IC \times \frac{Memory\ accesses}{Instruction} \times Missrate \times Miss\ penalty$$

$$= IC \times (1+0.5) \times 0.02 \times 25 = IC \times 0.75$$

Example2: Impact on Performance

The total performance is thus:

$$\text{CPU execution time cache} = (IC \times 1.0 + IC \times 0.75) \times \text{Clock cycle}$$

$$= 1.75 \times IC \times \text{Clock cycle}$$

The performance ratio is the inverse of the execution times

$$\frac{\text{CPU execution time}_{\text{cache}}}{\text{CPU execution time}} = \frac{1.75 \times IC \times \text{Clock cycle}}{1.0 \times IC \times \text{clock cycle}}$$

$$= 1.75$$

The computer with no cache misses is 1.75 time faster.

Example3-1: Impact on Performance

Assume(406) : unified caches: 32K unified cache

- Split cache: 16K D-cache and 16K I-cache
- 36% of the instructions are data transfer instructions
- A hit takes 1 colck cycle
- The miss penalty is 100 clock cycles
- A load/store take 1 extra clock cycle on a unified cache
- Write-through with a write-buffer
  and ignore stalls due to the write buffer
- What is the miss rate in each case?
- What is the average memory access time in each case?

From Figure 5.8 The unified miss rate needs to account for instruction and data accesses:

$$\text{Miss rate}_{\text{32KB unified}} = \frac{43.3/1000}{1.00+0.36} = 0.0318$$

instructions

Example3-1: Impact on Performance

Since every instruction access has exactly one memory access to fetch the instruction, according to Figure 5.8 the instruction cache miss rate is

$$\text{Miss rate}_{\text{16KB instruction}} = \frac{3.82/1000}{1.0} = 0.004$$

Since 36% of the instructions are data transfers, according to Figure 5.8 the data miss rate is

$$\text{Miss rate}_{\text{16KB data}} = \frac{40.9/1000}{0.36} = 0.114$$

Basing on Figure 2.32 on page 138 there is 74% instruction references in split cache. The average miss rate for the split cache is:

$$(74\% \times 0.004)+(26\% \times 0.114)=0.0324$$

Thus ,a 32KB unified cache has a slightly lower effective miss rate than two 16KB caches.

Example3-2: Impact on Performance

- The average memory access time can be divided into instruction and data accesses:

$$\textit{Average memory access time}$$
$$= \%instructions \times (HitTime_{Inst} + MissRate_{Inst} \times MissPenalty_{Inst})$$
$$+ \ \%\text{data} \times (HitTime_{Data} + MissRate_{Data} \times MissPenalty_{Data})$$

- Therefore,the time for each organization is

Average memory access time$_{split}$
$=74\% \times (1+0.004 \times 100)+ 26\% \times (1+0.114 \times 100)$
$=(74\% \times 1.38)+(26\% \times 12.36)=1.023+3.214=4.24$

Average memory access time$_{unified}$
$=74\% \times (1+0.0318 \times 100)+$
$26\% \times (1+1+0.0318 \times 100)$
$=(74\% \times 4.18)+(26\% \times 5.18)=3.096+1.348=4.44$

Hence, this split cache in this example—which offer two memory ports per clock cycle, thereby avoiding the structural hazard—have a better average memory access time than the single-ported unified cache despite having a worse effective miss rate.

Example4: Impact on Performance

Assume(408): in-order execution computer, such as the Ultra SPARC Ⅲ.

Miss penalty: 100 clock cycles

Miss rate : 2%

Memory references Per instruction: 1.5

Average cache misses per 1000 instructions: 30

CPI ＝1.0(ignoring memory stalls)

What is the impact on performance when behavior of the cache is included (Calclate the impact using both misses per instruction and miss rate.)?

Answer : The performance, including cache misses, is

$$CPU_{time} = IC \times \left(CPI_{exexution} + \frac{Mem\,stall\,clock\,cycles}{Instruction} \right) \times Clock\,cycle\,time$$

Example4: Impact on Performance

CPU time $_{\text{with cache}}$ =

$= \text{IC} \times (1.0 + (30/1000 \times 100)) \times$ Clock cycle time

$= \text{IC} \times 4.00 \times$ Clock cycle time

Now caculating performance using miss rate:

$$CPU_{time} = IC \times \left(CPI_{exexution} + \ Missrate \times \frac{Mem\ accesses}{Instruction} \times Misspenalty \right) \times Clock\,cycle\,time$$

CPU time $_{\text{with cache}}$ =

$= \text{IC} \times (1.0 + (1.5 \times 2\% \times 100)) \times$ Clock cycle time

$= \text{IC} \times 4.00 \times$ Clock cycle time

- The clock cycles time and instruction count are the same, with or without a cache. Thus, CPU time increases fourfold, with CPI from 1.00 a “perfect cache” to 4.00 with a cache that can miss.
- Without any memory hierarchy at all the CPI would increase again to $1.0+100 \times 1.5$ or 151—factor of almost 40 time longer than a system with a cache.

Example5: Impact on Performance

Assume(p409): CPI=2(perfect cache) clock cycle time＝1.0 ns

- MPI(memory reference per instruction)＝1.5
- Size of both caches is 64K and size of both block is 64 bytes
- One cache is direct mapped and other is two-way set associative. the former has miss rate of 1.4%, the latter has miss rate 1.0%
- The selection multiplexor forces CPU clock cycle time to be stretched 1.25 times
- Miss penalty is 75ns,and hit time is 1 clock cycle

● What is the impact of two diffect cache organizations on performance of CPU (first,calculate the average memory access time and then CPU performance.)?

Answer : Average memory access time is

Average memory access time＝Hit time+Miss rate×miss penalty

Thus, the time for each organization is

$\text{Average memory access time}_{1\text{-way}} = 1.0 + (0.014 \times 75) = 2.05$ ns

$\text{Average memory access time}_{2\text{-way}} = 1.0 \times 1.25 + (0.01 \times 75) = 2.00$ ns

Example5: Impact on Performance

The average memory access time is better for the 2-way set-associative cache.

CPU performance is

$$CPUtime = IC \times \left(CPI_{execution} + \frac{Misses}{Instruction} \times Misspenalty\right) \times Clock\,cycle\,time$$

$$= IC \times \left[\left(CPI_{execution} \times Clock\,cycle\,time\right) + \left(Miss\,rate \times \frac{Memory\ accesses}{Instruction} \times Misspenalty \times Clock\,cycle\,time\right)\right]$$

Substituting 75 ns for (miss penalty × Clock cycle time), the performance of each cache organization is

CPU $\text{time}_{\text{1-way}} = IC \times (2 \times 1.0 + (1.5 \times 0.014 \times 75)) = 3.58 \times IC$

Relative performance is

$$\frac{CPUtime_{2-way}}{CPUtime_{1-way}} = \frac{3.63 \times Instruction\,count}{3.58 \times Instruction\,count} = \frac{3.63}{3.58} = 1.01$$

In contrast to the results of average memory access time, the direct-mapped lesds to slighly better average performance. Since CPU time is our bottom-line evaluation.

How to Improve

$$AMAT = HitTime + MissRate \times MissPenalty$$

Hence, we organize 17 cache optimizations into four categories:

1.Reduce the miss penalty--5

——multilevel caches, critical word first, read miss before write miss, merging write buffers, and victim caches

2. Reduce the miss rate--5

——larger block size, large cache size, higher associativity, way prediction and pseudo associativity, and compiler optimizations

3. Reduce the miss penalty and miss rate via parallelism

——non-blocking caches, hardware prefetching, and compiler prefetching

4. Reduce the time to hit in the cache.--4

——small and simple caches, avoiding address translation, pipelined cache access, and trace caches

5.4 Reducing Cache miss penalty

Be continued

1.Reduce the miss penalty ——5

2. Reduce the miss rate

3. Reduce the miss penalty and miss rate via parallelism

4. Reduce the time to hit in the cache.

THANK YOU