COMP4611: Design and Analysis of Computer Architectures

Memory System

Cache

Lin Gu
CSE, HKUST

Memory Hierarchy: Motivation
The Principle Of Locality

- Programs usually access a relatively small portion of their address space (instructions/data) at any instant of time (loops, data arrays).
- Two Types of locality:
  - **Temporal Locality: If an item is referenced, it will tend to be referenced again soon.**
  - **Spatial locality: If an item is referenced, items whose addresses are close by will tend to be referenced soon .**
- The presence of locality in program behavior (e.g., loops, data arrays), makes it possible to satisfy a large percentage of program access needs (both instructions and operands) using memory levels with much less capacity than the program address space.

Locality Example

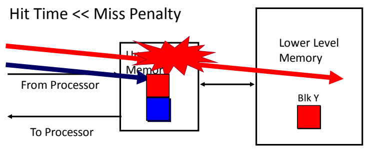
Locality Example:

```
sum = 0;
for (i = 0; i < n; i++)
    sum += a[i];
return sum;
```

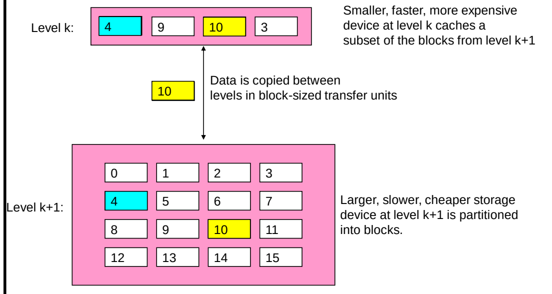
- Data
  - Reference array elements in succession (stride-1 reference pattern): Spatial locality
  - Reference `sum` each iteration: Temporal locality
- Instructions
  - Reference instructions in sequence: Spatial locality
  - Cycle through loop repeatedly: Temporal locality

Memory Hierarchy: Terminology

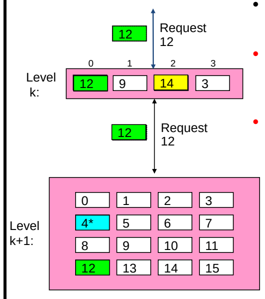
- **A Block:** The smallest unit of information transferred between two levels.
- **Hit:** Item is found in some block in the upper level (example: Block X)
  - **Hit Rate: The fraction of memory accesses found in the upper level.**
  - **Hit Time: Time to access the upper level which consists of**
    memory access time + time to determine hit/miss
- Miss: Item needs to be retrieved from a block in the lower level (Block Y)
  - **Miss Rate = 1 - (Hit Rate)**
  - **Miss Penalty: Time to replace a block in the upper level +**
    Time to deliver the block to the processor
- Hit Time << Miss Penalty



Caching in a Memory Hierarchy


Level k:
4
9
10
3
Smaller, faster, more expensive device at level k caches a subset of the blocks from level k+1
10
Data is copied between levels in block-sized transfer units
Level k+1:
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
Larger, slower, cheaper storage device at level k+1 is partitioned into blocks.

General Caching Concepts



- Program needs object d, which is stored in some block b.
- Cache hit
  - Program finds b in the cache at level k. E.g., block 14.
- Cache miss
  - b is not at level k, so level k cache must fetch it from level k+1. E.g., block 12.
  - If level k cache is full, then some current block must be replaced (evicted). Which one is the “victim”?
    - Placement policy: where can the new block go? E.g., b mod 4
    - Replacement policy: which block should be evicted? E.g., LRU

Cache Design & Operation Issues

- Q1: Where can a block be placed in cache?
  (Block placement strategy & Cache organization)
  - Fully Associative, Set Associative, Direct Mapped.
- Q2: How is a block found if it is in cache?
  (Block identification)
  - Tag/Block.
- Q3: Which block should be replaced on a miss?
  (Block replacement)
  - Random, LRU.
- Q4: What happens on a write?
  (Cache write policy)
  - Write through, write back.

7

Types of Caches: Organization

Type of cache	Mapping of data from memory to cache	Complexity of searching the cache
Direct mapped (DM)		Easy search mechanism
Set-associative (SA)	A memory value can be placed in **any of a set of locations** in the cache	Slightly more involved search mechanism
Fully-associative (FA)	A memory value can be placed in **any location** in the cache	Extensive hardware resources required to search (CAM)

•DM and FA can be thought as special cases of SA
•DM → 1-way SA
•FA → All-way SA

8

Cache Organization & Placement Strategies

Placement strategies or mapping of a main memory data block onto cache block frame addresses divide cache into three organizations:

- **Direct mapped cache:** A block can be placed in one location only, given by:

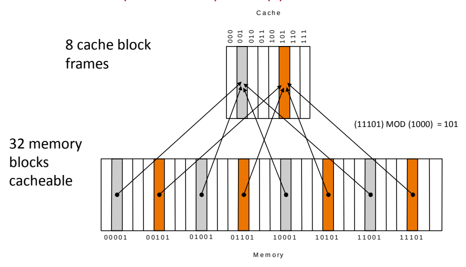
  (Block address) MOD (Number of blocks in cache)

  - Advantage: It is easy to locate blocks in the cache (only one possibility)
  - Disadvantage: Certain blocks cannot be simultaneously present in the cache (they can only have the same location)

9

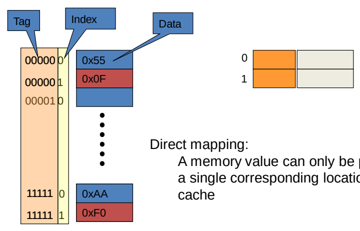
Cache Organization: Direct Mapped Cache

A block can be placed in one location only, given by:
(Block address) MOD (Number of blocks in cache)
In this case: (Block address) MOD (8)



10

Direct Mapping



Direct mapping:
A memory value can only be placed at a single corresponding location in the cache

11

Cache Organization & Placement Strategies

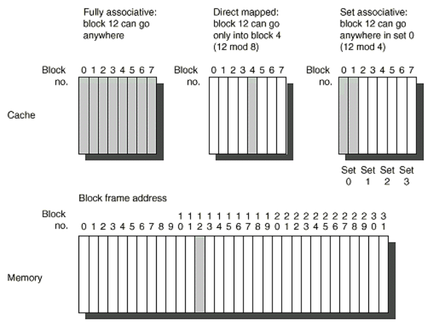
- **Fully associative cache:** A block can be placed anywhere in cache.
  - Advantage: No restriction on the placement of blocks. Any combination of blocks can be simultaneously present in the cache.
  - Disadvantage: Costly (hardware and time) to search for a block in the cache
- **Set associative cache:** A block can be placed in a restricted set of places, or cache block frames. A set is a group of block frames in the cache. A block is first mapped onto the set and then it can be placed anywhere within the set. The set in this case is chosen by:

  (Block address) MOD (Number of sets in cache)

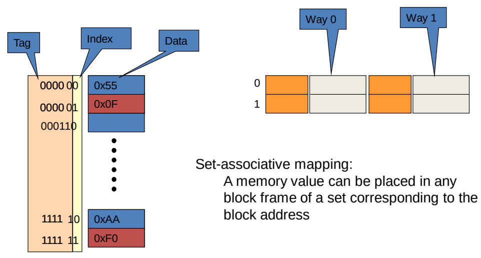
  - If there are *n* blocks in a set the cache placement is called *n*-way set-associative, or n-associative.
  - A good compromise between direct mapped and fully associative caches (most processors use this method).

12

Cache Organization Example

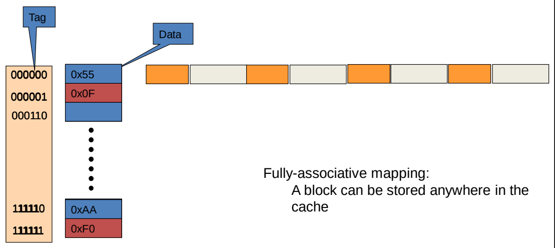

Fully associative: block 12 can go anywhere
Direct mapped: block 12 can go only into block 4 (12 mod 8)
Set associative: block 12 can go anywhere in set 0 (12 mod 4)
Block no. 0 1 2 3 4 5 6 7
Block no. 0 1 2 3 4 5 6 7
Block no. 0 1 2 3 4 5 6 7
Cache
Set 0 Set 1 Set 2 Set 3
Block frame address
Block no.
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31
Memory

Set Associative Mapping (2-Way)



Set-associative mapping:
A memory value can be placed in any block frame of a set corresponding to the block address

Fully Associative Mapping



Fully-associative mapping:
A block can be stored anywhere in the cache

Cache Organization Tradeoff

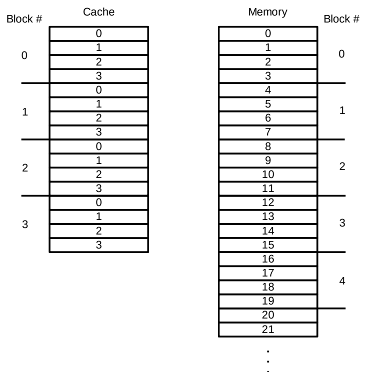
- For a given cache size, there is a tradeoff between hit rate and complexity
- If L = number of lines (blocks) in the cache,
  L = Cache Size / Block Size

How many places for a block to go	Name of cache type	Number of Sets
1	Direct Mapped	L
n	n-way associative	L/n
L	Fully Associative	1

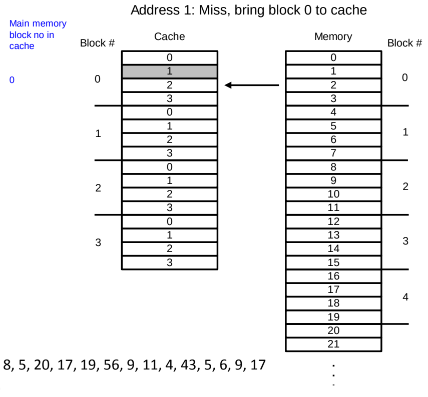
↑
Number of comparators needed to compare tags

An Example

- Assume a direct mapped cache with 4-word blocks and a total size of 16 words.
- Consider the following string of address references given as word addresses:
  – 1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17
- Show the hits and misses and final cache contents.

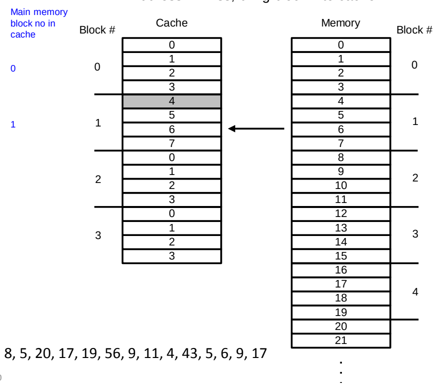


Address 1: Miss, bring block 0 to cache



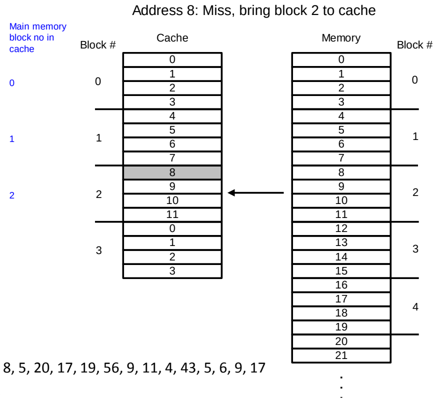
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 4: Miss, bring block 1 to cache



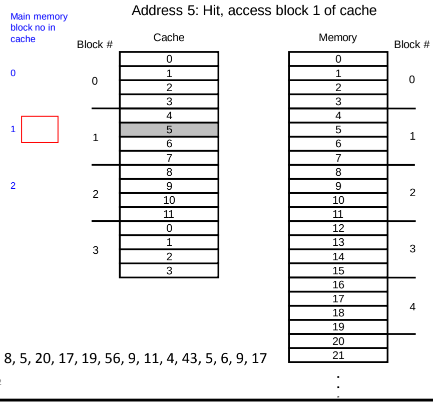
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 8: Miss, bring block 2 to cache



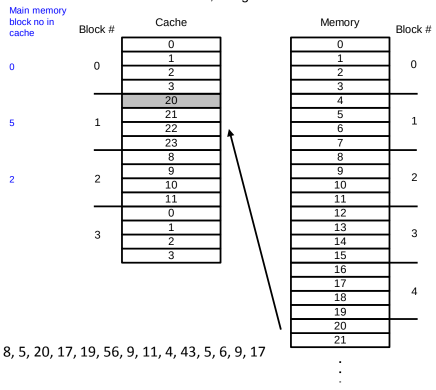
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 5: Hit, access block 1 of cache



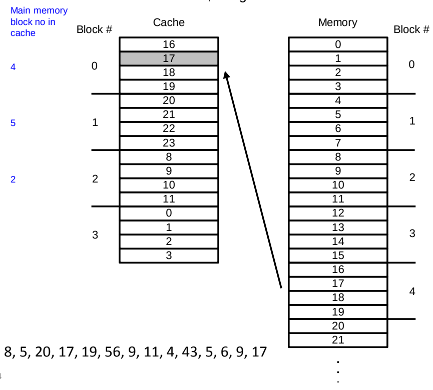
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 20: Miss, bring block 5 to cache block 1



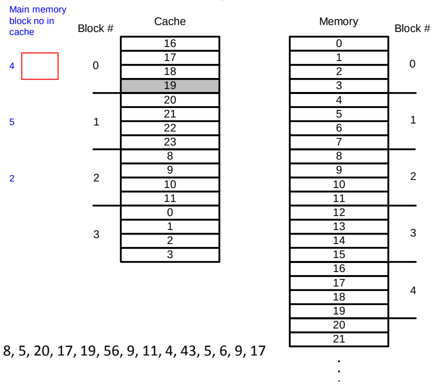
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 17: Miss, bring block 4 to cache block 0



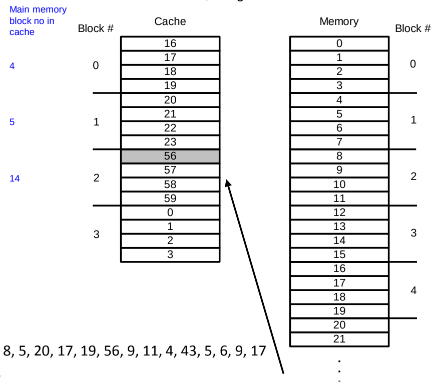
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 19: Hit, access block 0 of cache



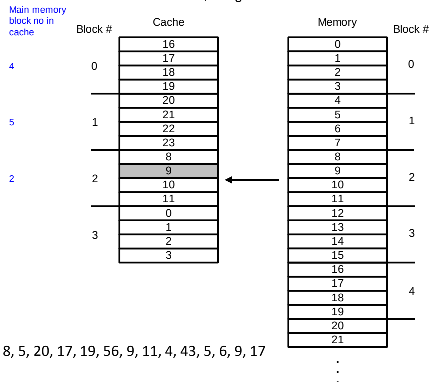
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 56: Miss, bring block 14 to cache block 2



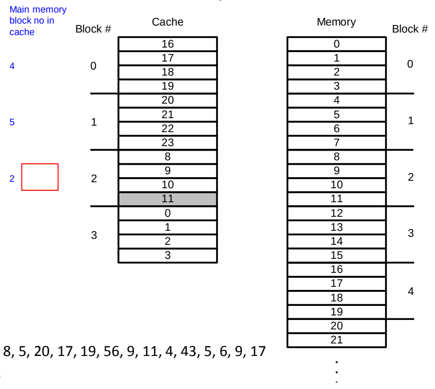
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 9: Miss, bring block 2 to cache block 2



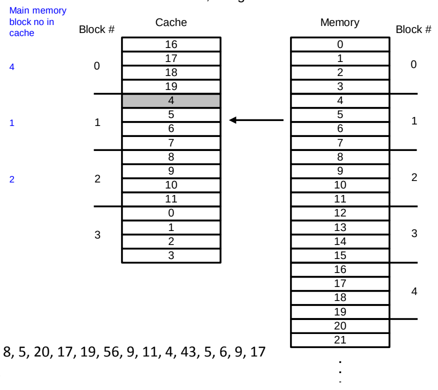
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 11: Hit, access block 2 of cache



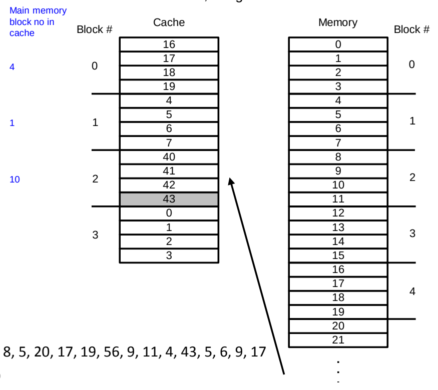
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 4: Miss, bring block 1 to cache block 1



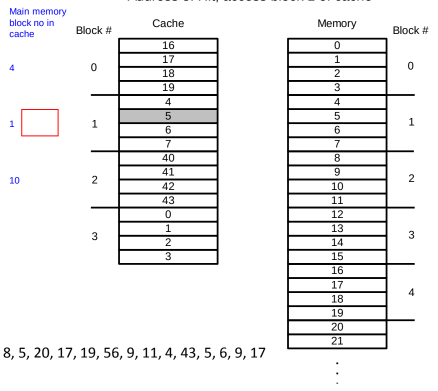
1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

Address 43: Miss, bring block 10 to cache block 2



1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

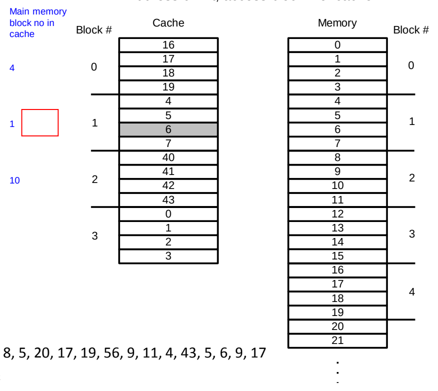
Address 5: Hit, access block 1 of cache



1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

31

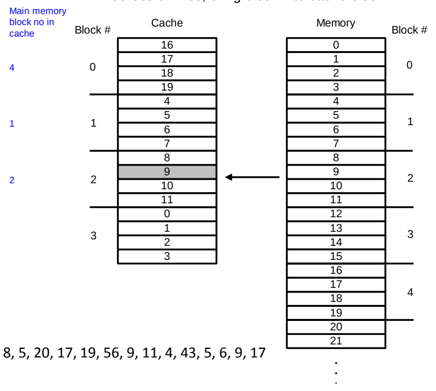
Address 6: Hit, access block 1 of cache



1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

32

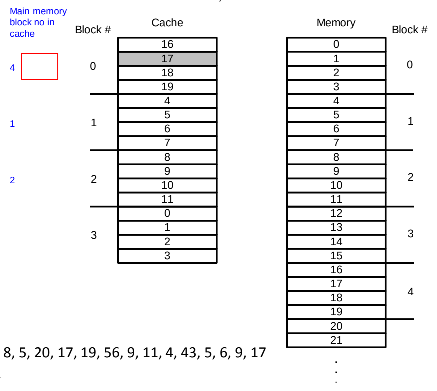
Address 9: Miss, bring block 2 to cache block 2



1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

33

Address 17: Hit, access block 0 of cache



1, 4, 8, 5, 20, 17, 19, 56, 9, 11, 4, 43, 5, 6, 9, 17

34

Summary

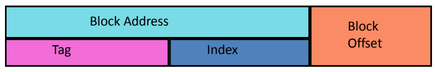
- Number of Hits = 6
- Number of Misses = 10
- Hit Ratio: 6/16 = 37.5% Unacceptable
- Typical Hit ratio: > 90%

Address	Miss/Hit
1	Miss
4	Miss
8	Miss
5	Hit
20	Miss
17	Miss
19	Hit
56	Miss
9	Miss
11	Hit
4	Miss
43	Miss
5	Hit
6	Hit
9	Miss
17	Hit

35

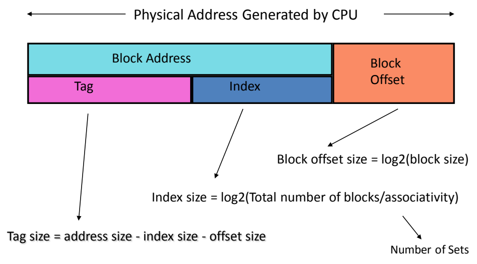
Locating A Data Block in Cache

- Each block in the cache has an address tag.
- The tags of every cache block that might contain the required data are checked in parallel.
- A valid bit is added to the tag to indicate whether this cache entry is valid or not.
- The address from the CPU to the cache is divided into:
  - **A block address, further divided into:**
    - An index field to choose a block set in the cache. (no index field when fully associative).
    - A tag field to search and match addresses in the selected set.
  - **A block offset to select the data from the block.**



36

Address Field Sizes


Physical Address Generated by CPU
Block Address
Tag
Index
Block Offset
Block offset size = log2(block size)
Index size = log2(Total number of blocks/associativity)
Tag size = address size - index size - offset size
Number of Sets

Mapping function:

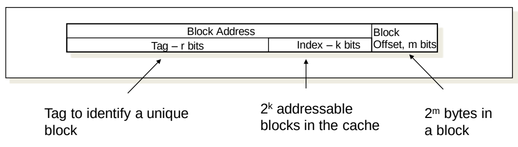
Cache set or block frame number = Index =

= (Block Address) MOD (Number of Sets)

37

Locating A Data Block in Cache

- Increasing associativity shrinks index, expands tag
  - Block index not needed for fully associative cache



38

Direct-Mapped Cache Example

- Suppose we have a 16KB of data in a direct-mapped cache with 4 word blocks
- Determine the size of the tag, index and offset fields if we're using a 32-bit architecture
- Offset
  - need to specify correct byte within a block
  - block contains 4 words = 16 bytes = 2^4 bytes
  - need 4 bits to specify correct byte

39

Direct-Mapped Cache Example

- Index: (~index into an "array of blocks")
  - need to specify correct row in cache
  - cache contains 16 KB = 2^{14} bytes
  - block contains 2^4 bytes (4 words)

# rows/cache =# blocks/cache (since there's one block/row)

$= \frac{\text{bytes/cache}}{\text{bytes/row}}$

$= \frac{2^{14}\ \text{bytes/cache}}{2^4\ \text{bytes/row}}$

$= 2^{10}$ rows/cache

need 10 bits to specify this many rows

40

Direct-Mapped Cache Example

- Tag: use remaining bits as tag
  - tag length = mem addr length
    - offset
    - index

    = 32 - 4 - 10 bits

    = 18 bits
  - so tag is leftmost 18 bits of memory address

41

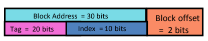
4KB Direct Mapped Cache Example

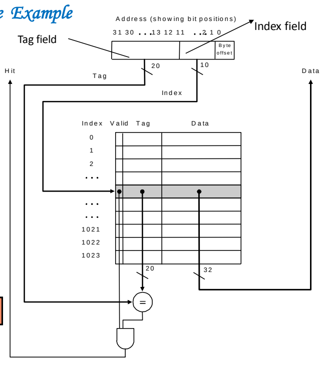
1K = 1024 Blocks
Each block = one word

Can cache up to 2^{32} bytes = 4 GB of memory

Mapping function:

Cache Block frame number = (Block address) MOD (1024)



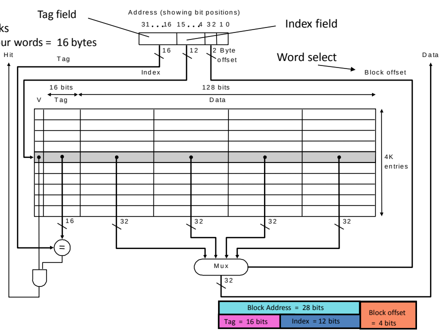


42

64KB Direct Mapped Cache Example

4K= 4096 blocks
Each block = four words = 16 bytes

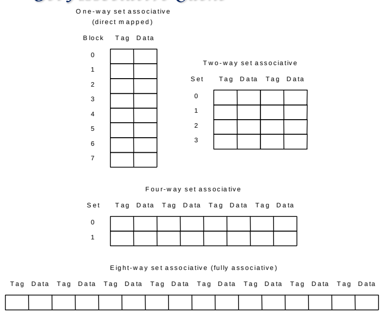
Can cache up to 2^{32} bytes = 4 GB of memory



Mapping Function: Cache Block frame number = (Block address) MOD (4096)
Larger blocks take better advantage of spatial locality

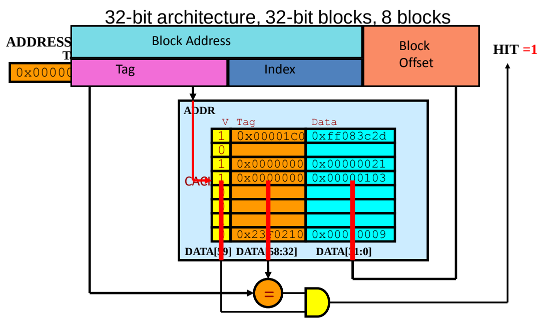
43

Cache Organization: Set Associative Cache


One-way set associative (direct mapped)
Block Tag Data
0
1
2
3
4
5
6
7
Two-way set associative
Set Tag Data Tag Data
0
1
2
3
Four-way set associative
Set Tag Data Tag Data Tag Data Tag Data
0
1
Eight-way set associative (fully associative)
Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data Tag Data

44

Direct-Mapped Cache Design

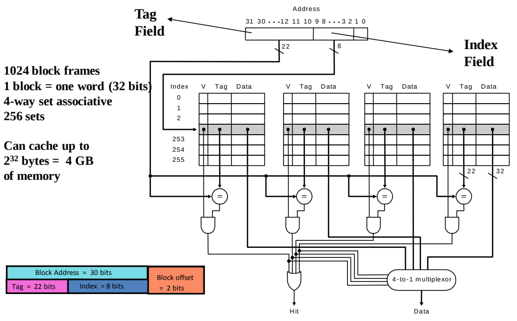


45

4K Four-Way Set Associative Cache: MIPS Implementation Example

1024 block frames
1 block = one word (32 bits)
4-way set associative
256 sets

Can cache up to 2^{32} bytes = 4 GB of memory

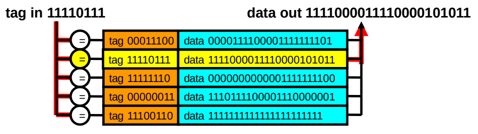


Mapping Function: Cache Set Number = (Block address) MOD (256)

46

Fully Associative Cache Design

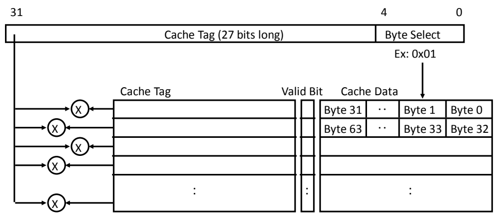
- Key idea: set size of one block
  - 1 comparator required for each block
  - No address decoding
  - Practical only for small caches due to hardware demands


tag in 11110111
data out 1111000011110000101011
=
tag 00011100 data 000011110000111111101
tag 11110111 data 1111000011110000101011
tag 11111110 data 000000000000111111100
tag 00000011 data 111011110000111000001
tag 11100110 data 111111111111111111111

47

Fully Associative

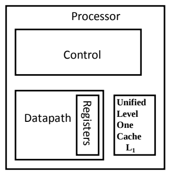
- Fully Associative Cache
  - 0 bit for cache index
  - Compare the Cache Tags of all cache entries in parallel
  - Example: Block Size = 32 B blocks, we need N 27-bit comparators



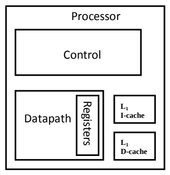
48

Unified vs.Separate Level 1 Cache

- Unified Level 1 Cache
  A single level 1 cache is used for both instructions and data.
- Separate instruction/data Level 1 caches (Harvard Memory Architecture):
  The level 1 (L_1) cache is split into two caches, one for instructions (instruction cache, L_1 I-cache) and the other for data (data cache, L_1 D-cache).



Unified Level 1 Cache



Separate Level 1 Caches
(Harvard Memory Architecture)

49

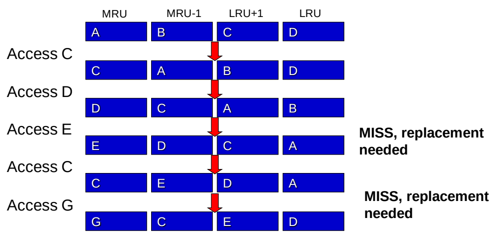
Cache Replacement Policy

When a cache miss occurs the cache controller may have to select a block of cache data to be removed from a cache block frame and replaced with the requested data, such a block is selected by one of two methods (for direct mapped cache, there is only one choice):

- **Random:**
  - Any block is randomly selected for replacement providing uniform allocation.
  - Simple to build in hardware.
  - The most widely used cache replacement strategy.
- **Least-recently used (LRU):**
  - Accesses to blocks are recorded and the block replaced is the one that was not used for the longest period of time.
  - LRU is *expensive* to implement, as the number of blocks to be tracked increases, and is usually approximated.

50

LRU Policy



51

Miss Rates for Caches with Different Size, Associativity & Replacement Algorithm

Sample Data

Associativity:	**2-way**		**4-way**		**8-way**	
Size	LRU	Random	LRU	Random	LRU	Random
16 KB	5.18%	5.69%	4.67%	5.29%	4.39%	4.96%
64 KB	1.88%	2.01%	1.54%	1.66%	1.39%	1.53%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

52

Cache and Memory Performance

Average Memory Access Time (AMAT), Memory Stall cycles

- ***The Average Memory Access Time*** (**AMAT**): The average number of cycles required to complete a memory access request by the CPU.
- Memory stall cycles per memory access: The number of stall cycles added to CPU execution cycles for one memory access.
- For an ideal memory: AMAT = 1 cycle, this results in zero memory stall cycles.
- Memory stall cycles per memory access = AMAT -1
- Memory stall cycles per instruction =

  Memory stall cycles per memory access

  x Number of memory accesses per instruction

  = (AMAT -1) x (1 + fraction of loads/stores)

Instruction Fetch

53

Cache Performance

Unified Memory Architecture

- For a CPU with a single level (L1) of cache for both instructions and data and no stalls for cache hits:

With ideal memory

Total CPU time = (CPU execution clock cycles +
Memory stall clock cycles) x clock cycle time

Memory stall clock cycles =
(Reads x Read miss rate x Read miss penalty) +
(Writes x Write miss rate x Write miss penalty)

If write and read miss penalties are the same:

Memory stall clock cycles = Memory accesses x Miss rate x Miss penalty

54

Cache Performance

Unified Memory Architecture

- **CPUtime = Instruction count x CPI x Clock cycle time**
- **$CPI_{execution}$ = CPI with ideal memory**
- **CPI = $CPI_{execution}$ + MEM Stall cycles per instruction**
- **CPUtime = Instruction Count x ($CPI_{execution}$ +**
  MEM Stall cycles per instruction) x Clock cycle time
- **MEM Stall cycles per instruction =**
  MEM accesses per instruction x Miss rate x Miss penalty
- **CPUtime = IC x ($CPI_{execution}$ + MEM accesses per instruction x**
  Miss rate x Miss penalty) x Clock cycle time
- **Misses per instruction = Memory accesses per instruction x Miss rate**
- **CPUtime = IC x ($CPI_{execution}$ + Misses per instruction x Miss penalty) x**
  Clock cycle time

55

Memory Access Tree
For Unified Level 1 Cache

CPU Memory Access

L_1

L1 Hit:
% = Hit Rate = H1
Access Time = 1
Stalls= H1 x 0 = 0
(No Stall)

L1 Miss:
% = (1- Hit rate) = (1-H1)
Access time = M + 1
Stall cycles per access = M x (1-H1)

AMAT = H1 x 1 + (1 -H1) x (M+ 1) = 1 + M x (1 -H1)

Stall Cycles Per Access = AMAT - 1 = M x (1 -H1)

M = Miss Penalty
H1 = Level 1 Hit Rate
1- H1 = Level 1 Miss Rate

56

Cache Impact On Performance: An Example

Assuming the following execution and cache parameters:

- **Cache miss penalty = 50 cycles**
- **Normal instruction execution CPI ignoring memory stalls = 2.0 cycles**
- **Miss rate = 2%**
- **Average memory references/instruction = 1.33**

CPU time = IC x [CPI $_{execution}$ + Memory accesses/instruction x Miss rate x
Miss penalty] x Clock cycle time

CPUtime $_{with\ cache}$ = IC x (2.0 + (1.33 x 2% x 50)) x clock cycle time
= IC x 3.33 x Clock cycle time

→ *Lower CPI $_{execution}$ increases the impact of cache miss clock cycles*

57

Cache Performance Example

- Suppose a CPU executes at Clock Rate = 200 MHz (5 ns per cycle) with a single level of cache.
- $CPI_{execution}$ = 1.1
- Instruction mix: 50% arith/logic, 30% load/store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

CPI = $CPI_{execution}$ + mem stalls per instruction
Mem Stalls per instruction = Mem accesses per instruction x Miss rate x Miss penalty
Mem accesses per instruction = 1 + .3 = 1.3

Mem Stalls per instruction = 1.3 x .015 x 50 = 0.975
Instruction fetch **Load/store**
CPI = 1.1 + .975 = 2.075

The ideal memory CPU with no misses is 2.075/1.1 = 1.88 times faster

58

Cache Performance Example

- Suppose for the previous example we double the clock rate to 400 MHZ, how much faster is this machine, assuming similar miss rate, instruction mix?
- Since memory speed is not changed, the miss penalty takes more CPU cycles:

Miss penalty = 50 x 2 = 100 cycles.
CPI = 1.1 + 1.3 x .015 x 100 = 1.1 + 1.95 = 3.05
Speedup = (CPI_{old} x C_{old})/ (CPI_{new} x C_{new})
= 2.075 x 2 / 3.05 = 1.36

The new machine is only 1.36 times faster rather than 2 times faster due to the increased effect of cache misses.

→ *CPUs with higher clock rate, have more cycles per cache miss and more memory impact on CPI.*

59

Cache Performance

Harvard Memory Architecture

For a CPU with separate or split level one (L1) caches for instructions and data (Harvard memory architecture) and no stalls for cache hits:

CPUtime = Instruction count x CPI x Clock cycle time

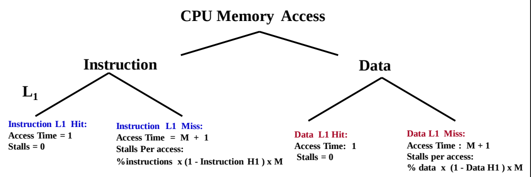
CPI = $CPI_{execution}$ + Mem Stall cycles per instruction

CPUtime = Instruction Count x ($CPI_{execution}$ +
Mem Stall cycles per instruction) x Clock cycle time

Mem Stall cycles per instruction = Instruction Fetch Miss rate x Miss Penalty + Data Memory Accesses Per Instruction x Data Miss Rate x Miss Penalty

60

Memory Access Tree For Separate Level 1 Caches



Stall Cycles Per Access = % Instructions x (1 - Instruction H1) x M + % data x (1 - Data H1) x M

AMAT = 1 + Stall Cycles per access

61

Typical Cache Performance Data Using SPEC92

Size	Instruction cache	Data cache	Unified cache
1 KB	3.06%	24.61%	13.34%
2 KB	2.26%	20.57%	9.78%
4 KB	1.78%	15.94%	7.24%
8 KB	1.10%	10.19%	4.57%
16 KB	0.64%	6.47%	2.87%
32 KB	0.39%	4.82%	1.99%
64 KB	0.15%	3.77%	1.35%
128 KB	0.02%	2.88%	0.95%

62

Cache Performance Example

- To compare the performance of either using a 16-KB instruction cache and a 16-KB data cache as opposed to using a unified 32-KB cache, we assume a hit to take one clock cycle and a miss to take 50 clock cycles, and a load or store to take one extra clock cycle on a unified cache, and that 75% of memory accesses are instruction references. Using the miss rates for SPEC92 we get:

Overall miss rate for a split cache = (75% x 0.64%) + (25% x 6.47%) = 2.1%

- From SPEC92 data a unified cache would have a miss rate of 1.99%

Average memory access time = 1 + stall cycles per access

= 1 + % instructions x (Instruction miss rate x Miss penalty)

+ % data x (Data miss rate x Miss penalty)

For split cache:

Average memory access time$_{split}$

= 1 + 75% x (0.64% x 50) + 25% x (6.47%x50) = 2.05 cycles

For unified cache:

Average memory access time$_{unified}$

= 1 + 75% x (1.99%) x 50) + 25% x (1 + 1.99% x 50) = 2.24 cycles

63

Cache Write Strategies

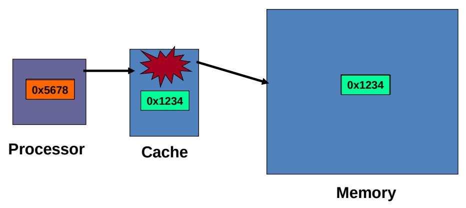
64

Cache Read/Write Operations

- Statistical data suggest that reads (*including instruction fetches)* dominate processor cache accesses (writes account for 25% of data cache traffic).
- In cache reads, a block is read at the same time while the tag is being compared with the block address (*searching*). If the read is a hit the data is passed to the CPU, if a miss it ignores it.
- In cache writes, modifying the block cannot begin until the tag is checked to see if the address is a hit.
- Thus for cache writes, tag checking cannot take place in parallel, and only the specific data requested by the CPU can be modified.
- Cache is classified according to the write and memory update strategy in place: write through, or write back.

65

Write-through Policy

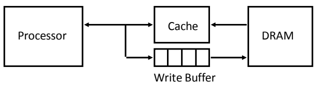


66

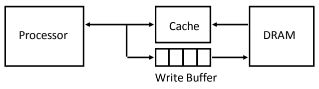
Cache Write Strategies

1 Write Though: Data is written to both the cache block and the main memory.

- The lower level always has the most updated data; an important feature for I/O and multiprocessing.
- Easier to implement than write back.
- A write buffer is often used to reduce CPU write stall while data is written to memory.

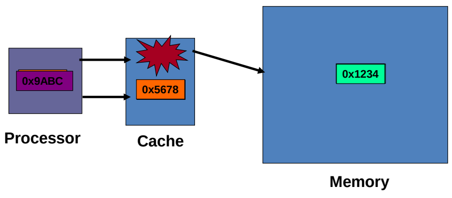


Write Buffer for Write Through



- A Write Buffer is needed between the Cache and Memory
  - Processor: writes data into the cache and the write buffer
  - Memory controller: write contents of the buffer to memory
- Write buffer is just a FIFO queue:
  - Typical number of entries: 4
  - Works fine if: Store frequency (w.r.t. time) << 1 / DRAM write cycle

Write-back Policy



Cache Write Strategies

2 Write back: Data is written or updated only to the cache block.

- **Writes occur at the speed of cache**
- **The modified or dirty cache block is written to main memory later (e.g., when it's being replaced from cache)**
- **A status bit called a dirty bit, is used to indicate whether the block was modified while in cache; if not the block is not written to main memory.**
- **Uses less memory bandwidth than write through.**

Write misses

- If we try to write to an address that is not already contained in the cache; this is called a write miss.
- Let's say we want to store 21763 into Mem[1101 0110] but we find that address is not currently in the cache.

Index	V	Tag	Data
...			
110	1	00010	123456
...			

Address	Data
...	
1101 0110	6378
...	

- When we update Mem[1101 0110], should we *also* load it into the cache?

No write-allocate

- With a no-write allocate policy, the write operation goes directly to main memory *without* affecting the cache.

Mem[1101 0110] = 21763

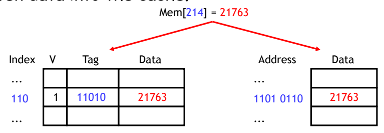
Index	V	Tag	Data
...			
110	1	00010	123456
...			

Address	Data
...	
1101 0110	21763
...	

- This is good when data is written but not immediately used again, in which case there's no point to load it into the cache yet.

Write Allocate

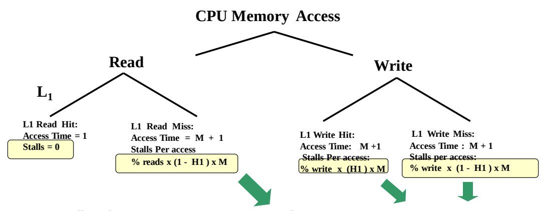
- A write allocate strategy would instead load the newly written data into the cache.



- If that data is needed again soon, it will be available in the cache.

Memory Access Tree, Unified L_1

Write Through, No Write Allocate, No Write Buffer

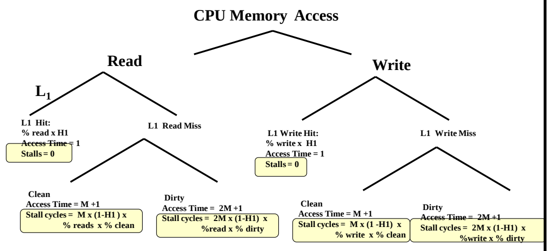


Stall Cycles Per Memory Access = % reads x (1 - H1) x M + % write x M

AMAT = 1 + % reads x (1 - H1) x M + % write x M

Memory Access Tree Unified L_1

Write Back, With Write Allocate



Stall Cycles Per Memory Access = (1-H1) x (M x % clean + 2M x % dirty)

AMAT = 1 + Stall Cycles Per Memory Access

Write Through Cache Performance Example

- A CPU with $CPI_{execution}$ = 1.1 uses a unified L1 Write Through, No Write Allocate and no write buffer.
- Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

CPI = $CPI_{execution}$ + MEM stalls per instruction

MEM Stalls per instruction = MEM accesses per instruction x Stalls per access

MEM accesses per instruction = 1 + .3 = 1.3

Stalls per access = % reads x miss rate x Miss penalty + % write x Miss penalty

% reads = 1.15/1.3 = 88.5% % writes = .15/1.3 = 11.5%

Stalls per access = 50 x (88.5% x 1.5% + 11.5%) = 6.4 cycles

Mem Stalls per instruction = 1.3 x 6.4 = 8.33 cycles

AMAT = 1 + 6.4 = 7.4 cycles

CPI = 1.1 + 8.33 = 9.43

The ideal memory CPU with no misses is 9.43/1.1 = 8.57 times faster

Write Back Cache Performance Example

- A CPU with $CPI_{execution}$ = 1.1 uses a unified L1 with write back , write allocate, and the probability a cache block is dirty = 10%
- Instruction mix: 50% arith/logic, 15% load, 15% store, 20% control
- Assume a cache miss rate of 1.5% and a miss penalty of 50 cycles.

CPI = $CPI_{execution}$ + mem stalls per instruction

MEM Stalls per instruction =

MEM accesses per instruction x Stalls per access

MEM accesses per instruction = 1 + .3 = 1.3

Stalls per access = (1-H1) x (M x % clean + 2M x % dirty)

Stalls per access = 1.5% x (50 x 90% + 100 x 10%) = .825 cycles

Mem Stalls per instruction = 1.3 x .825 = 1.07 cycles

AMAT = 1 + .825 = 1.825 cycles

CPI = 1.1 + 1.07 = 2.17

The ideal CPU with no misses is 2.17/1.1 = 1.97 times faster

Impact of Cache Organization: An Example

Given:

- A CPI with ideal memory = 2.0 Clock cycle = 2 ns
- 1.3 memory references/instruction Cache size = 64 KB with
- Cache miss penalty = 70 ns, no stall on a cache hit

Compare two caches

- One cache is direct mapped with miss rate = 1.4%
- The other cache is two-way set-associative, where:
  - **CPU clock cycle time increases 1.1 times to account for the cache selection multiplexor**
  - **Miss rate = 1.0%**

Impact of Cache Organization: An Example

Average memory access time = Hit time + Miss rate x Miss penalty

Average memory access time $_{1\text{-way}}$ = 2.0 + (.014 x 70) = 2.98 ns

Average memory access time $_{2\text{-way}}$ = 2.0 x 1.1 + (.010 x 70) = 2.90 ns

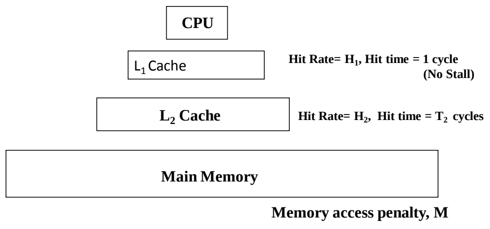
CPU time = IC x [CPI $_{execution}$ + Memory accesses/instruction x Miss rate x Miss penalty] x Clock cycle time

CPUtime $_{1\text{-way}}$ = IC x (2.0 x 2 + (1.3 x .014 x 70) = 5.27 x IC

CPUtime $_{2\text{-way}}$ = IC x (2.0 x 2 x 1.10 + (1.3 x 0.01 x 70)) = 5.31 x IC

- *In this example, 1-way cache offers slightly better performance with less complex hardware.*

2 Levels of Cache: L_1, L_2

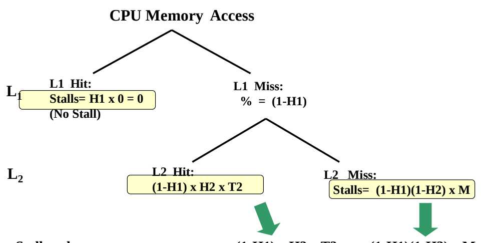

CPU
L_1 Cache
Hit Rate= H_1, Hit time = 1 cycle (No Stall)
L_2 Cache
Hit Rate= H_2, Hit time = T_2 cycles
Main Memory
Memory access penalty, M

Miss Rates For Multi-Level Caches

- Local Miss Rate: This rate is the number of misses in a cache level divided by the number of memory accesses to this level. Local Hit Rate = 1 - Local Miss Rate
- Global Miss Rate: The number of misses in a cache level divided by the total number of memory accesses generated by the CPU.
- Since level 1 receives all CPU memory accesses, for level 1:
  - **Local Miss Rate = Global Miss Rate = 1 - H1**
- For level 2 since it only receives those accesses missed in level 1:
  - Local Miss Rate = Miss rate$_{L2}$= 1- H2
  - Global Miss Rate = Miss rate$_{L1}$ x Miss rate$_{L2}$

    = (1- H1) x (1 - H2)

2-Level Cache Performance Memory Access Tree



Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M
AMAT = 1 + (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

T2: L2 cache hit time in cycle

2-Level Cache Performance

CPUtime = IC x (CPI$_{execution}$ + Mem Stall cycles per instruction) x C

Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

- For a system with 2 levels of cache, assuming no penalty when found in L_1 cache:
- Stall cycles per memory access =

  [miss rate L_1] x [Hit rate L_2 x Hit time L_2

  + Miss rate L_2 x Memory access penalty)] =

  (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

L1 Miss, L2 Hit

L1 Miss, L2 Miss: Must Access Main Memory

Two-Level Cache Example

- CPU with CPI$_{execution}$ = 1.1 running at clock rate = 500 MHZ
- 1.3 memory accesses per instruction.
- L_1 cache operates at 500 MHZ with a miss rate of 5%
- L_2 cache operates at 250 MHZ with local miss rate 40%, (T_2 = 2 cycles)
- Memory access penalty, M = 100 cycles. Find CPI.

CPI = CPI$_{execution}$ + MEM Stall cycles per instruction

With No Cache, CPI = 1.1 + 1.3 x 100 = 131.1

With single L_1, CPI = 1.1 + 1.3 x .05 x 100 = 7.6

With L1 and L2 caches:

MEM Stall cycles per instruction =

MEM accesses per instruction x Stall cycles per access

Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1)(1-H2) x M

= .05 x .6 x 2 + .05 x .4 x 100

= .06 + 2 = 2.06

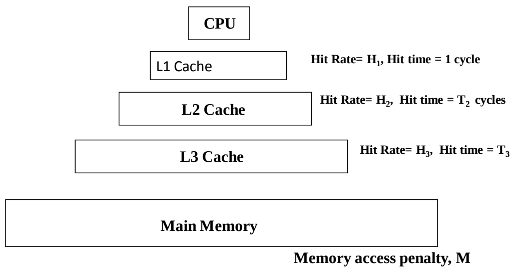
MEM Stall cycles per instruction =

MEM accesses per instruction x Stall cycles per access

= 2.06 x 1.3 = 2.678

CPI = 1.1 + 2.678 = 3.778 Speedup = 7.6/3.778 = 2

3 Levels of Cache



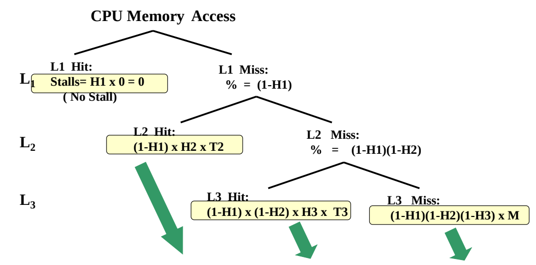
CPU
L1 Cache
Hit Rate= H_1, Hit time = 1 cycle
L2 Cache
Hit Rate= H_2, Hit time = T_2 cycles
L3 Cache
Hit Rate= H_3, Hit time = T_3
Main Memory
Memory access penalty, M

85

3-Level Cache Performance

Memory Access Tree

CPU Stall Cycles Per Memory Access



Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3 + (1-H1)(1-H2) (1-H3)x M
AMAT = 1 + Stall cycles per memory access

86

3-Level Cache Performance

CPUtime = IC x ($CPI_{execution}$ + Mem Stall cycles per instruction) x C
Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

- For a system with 3 levels of cache, assuming no penalty when found in L_1 cache:

Stall cycles per memory access =
[miss rate L_1] x [Hit rate L_2 x Hit time L_2
+ Miss rate L_2 x (Hit rate L3 x Hit time L_3
+ Miss rate L_3 x Memory access penalty)] =

(1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3

+ (1-H1)(1-H2) (1-H3)x M

L1 Miss, L2 Hit **L2 Miss, L3 Hit**

L1 Miss, L2 Miss:
Must Access Main Memory

87

Three-Level Cache Example

- CPU with $CPI_{execution}$ = 1.1 running at clock rate = 500 MHZ
- 1.3 memory accesses per instruction.
- L_1 cache operates at 500 MHZ with a miss rate of 5%
- L_2 cache operates at 250 MHZ with a local miss rate 40%, (T_2 = 2 cycles)
- L_3 cache operates at 100 MHZ with a local miss rate 50%, (T_3 = 5 cycles)
- Memory access penalty, M= 100 cycles. Find CPI.

88

Three-Level Cache Example

- Memory access penalty, M= 100 cycles. Find CPI.

With No Cache, CPI = 1.1 + 1.3 x 100 = 131.1
With single L_1, CPI = 1.1 + 1.3 x .05 x 100 = 7.6
With L1, L2 CPI = 1.1 + 1.3 x (.05 x .6 x 2 + .05 x .4 x 100) = 3.778

CPI = $CPI_{execution}$ + Mem Stall cycles per instruction
Mem Stall cycles per instruction = Mem accesses per instruction x Stall cycles per access

Stall cycles per memory access = (1-H1) x H2 x T2 + (1-H1) x (1-H2) x H3 x T3 + (1-H1)(1-H2) (1-H3)x M
= .05 x .6 x 2 + .05 x .4 x .5 x 5 + .05 x .4 x .5 x 100
= .06 + .05 + 1 = 1.11

CPI = 1.1 + 1.3 x 1.11 = 2.54

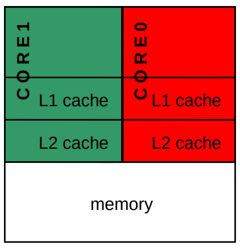
Speedup compared to L1 only = 7.6/2.54 = 3
Speedup compared to L1, L2 = 3.778/2.54 = 1.49

89

Cache on Multicore

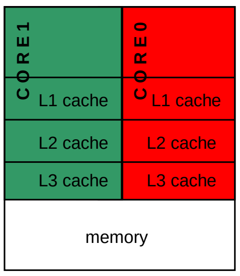
90 90

Multi-Core and caches coherence



Both L1 and L2 are private

Examples: AMD Opteron, AMD Athlon, Intel Pentium D



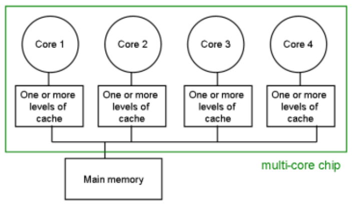
A design with L3 caches

Example: Intel Itanium 2

91

The cache coherence problem

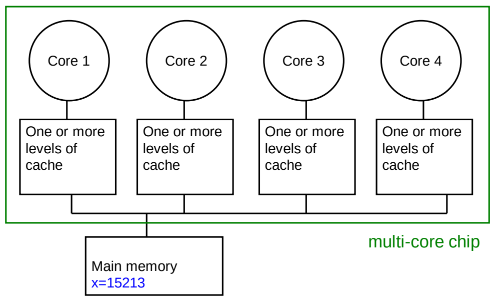
- Since we have private caches: How to keep the data consistent across caches?
- Each core should perceive the memory as a monolithic array, shared by all the cores


Core 1
Core 2
Core 3
Core 4
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
multi-core chip

92

The cache coherence problem

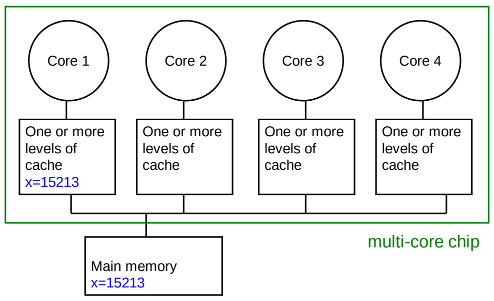
Suppose variable x initially contains 15213


Core 1
Core 2
Core 3
Core 4
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
x=15213
multi-core chip

93

The cache coherence problem

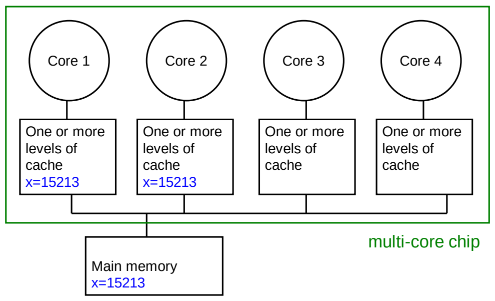
Core 1 reads x


Core 1
Core 2
Core 3
Core 4
One or more levels of cache
x=15213
One or more levels of cache
One or more levels of cache
One or more levels of cache
Main memory
x=15213
multi-core chip

94

The cache coherence problem

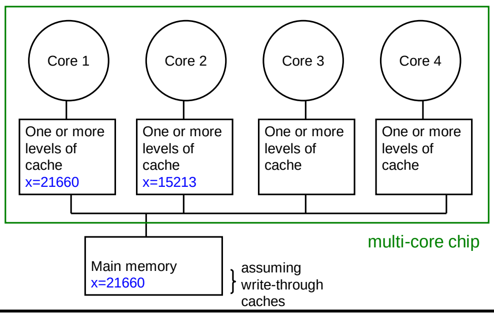
Core 2 reads x


Core 1
Core 2
Core 3
Core 4
One or more levels of cache
x=15213
One or more levels of cache
x=15213
One or more levels of cache
One or more levels of cache
Main memory
x=15213
multi-core chip

95

The cache coherence problem

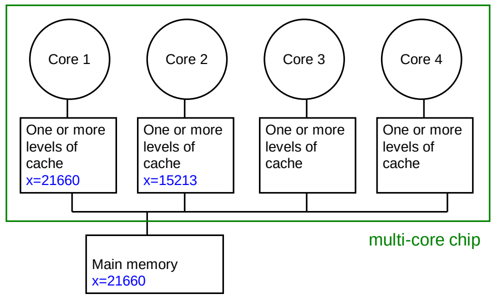
Core 1 writes to x, setting it to 21660


Core 1
Core 2
Core 3
Core 4
One or more levels of cache
x=21660
One or more levels of cache
x=15213
One or more levels of cache
One or more levels of cache
Main memory
x=21660
assuming write-through caches
multi-core chip

96

The cache coherence problem

Core 2 attempts to read x… gets a stale copy

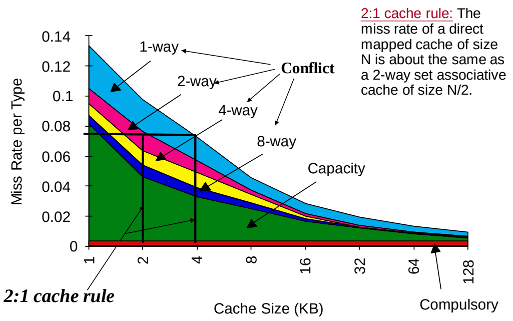

Core 1
Core 2
Core 3
Core 4
One or more levels of cache x=21660
One or more levels of cache x=15213
One or more levels of cache
One or more levels of cache
multi-core chip
Main memory x=21660

Reduce Miss Rate

Reducing Misses (3 Cs)

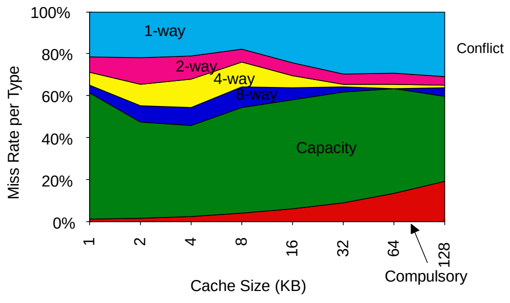
- Classifying Misses: 3 Cs
  - **—*Compulsory***—The first access to a block is not in the cache, so the block must be brought into the cache. These are also called *cold start misses* or *first reference misses*. *(Misses even in infinite size cache)*
  - **—*Capacity***—If the cache cannot contain all the blocks needed during the execution of a program, capacity misses will occur due to blocks being discarded and later retrieved. *(Misses due to size of cache)*
  - **—*Conflict***—If the block-placement strategy is not fully associative, conflict misses (in addition to compulsory and capacity misses) will occur because a block can be discarded and later retrieved if too many blocks map to its set. These are also called *collision misses* or *interference misses*. *(Misses due to associativity and size of cache)*

3Cs Absolute Miss Rates



2:1 cache rule: The miss rate of a direct mapped cache of size N is about the same as a 2-way set associative cache of size N/2.

3Cs Relative Miss Rate



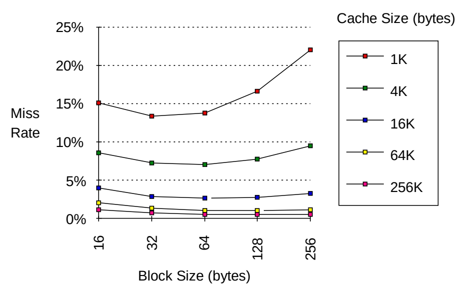
How to Reduce the 3 Cs Cache Misses?

- Increase Block Size
- Increase Associativity
- Use a Victim Cache
- Use a Pseudo Associative Cache
- Hardware Prefetching

1. Increase Block Size

- One way to reduce the miss rate is to increase the block size
  - Take advantage of spatial locality
  - Reduce compulsory misses
- However, larger blocks have disadvantages
  - May increase the miss penalty (need to get more data)
  - May increase hit time
  - May increase conflict misses (smaller number of block frames)
- Increasing the block size can help, but don't overdo it.

1. Reduce Misses via Larger Block Size



2. Reduce Misses via Higher Associativity

- Increasing associativity helps reduce conflict misses (8-way should be good enough)
- 2:1 Cache Rule:
  - The miss rate of a direct mapped cache of size N is about equal to the miss rate of a 2-way set associative cache of size N/2
- Disadvantages of higher associativity
  - Need to do large number of comparisons
  - Need n-to-1 multiplexor for n-way set associative
  - Could increase hit time
    - Hit time for 2-way vs. 1-way external cache +10%, internal + 2%

Example: Avg. Memory Access Time vs. Associativity

- Example: assume CCT = 1.10 for 2-way, 1.12 for 4-way, 1.14 for 8-way vs. CCT=1 of direct mapped.

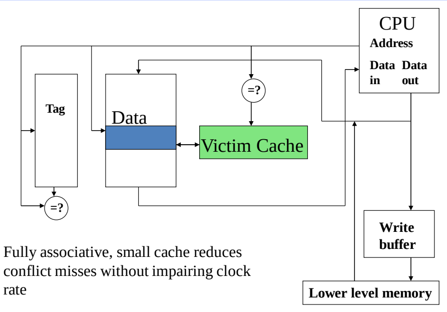
Cache Size	Associativity			
(KB)	1-way	2-way	4-way	8-way
1	7.65	6.60	6.22	5.44
2	5.90	4.90	4.62	4.09
4	4.60	3.95	3.57	3.19
8	3.30	3.00	2.87	2.59
16	2.45	2.20	2.12	2.04
32	2.00	1.80	1.77	1.79
64	1.70	1.60	1.57	1.59
128	1.50	1.45	1.42	1.44

(**Red** means memory access time not improved by higher associativity)
Does not take into account effect of slower clock on rest of program

3. Reducing Misses via Victim Cache

- Add a small fully associative victim cache to hold data discarded from the regular cache
- When data not found in cache, check victim cache
- 4-entry victim cache removed 20% to 95% of conflicts for a 4 KB direct mapped data cache
- Get access time of direct mapped with reduced miss rate

3. Victim Cache



Fully associative, small cache reduces conflict misses without impairing clock rate

4. Reducing Misses via Pseudo-Associativity

- How to combine fast hit time of direct mapped cache and the lower conflict misses of 2-way SA cache?
- Divide cache: on a miss, check other half of cache to see if there, if so have a pseudo-hit (slow hit).
- Usually check other half of cache by flipping the MSB of the index.

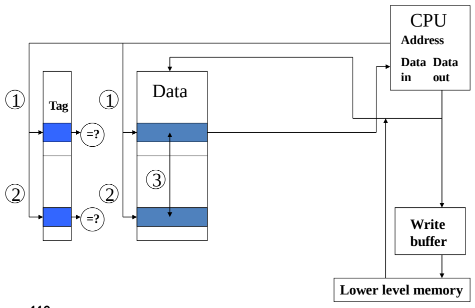
Drawbacks
- CPU pipeline is hard if hit takes 1 or 2 cycles
- Slightly more complex design

Hit Time

Pseudo Hit Time   Miss Penalty

109 109

Pseudo Associative Cache



110 110

5. Hardware Prefetching

- Instruction Prefetching
  - Alpha 21064 fetches 2 blocks on a miss
  - Extra block placed in ***stream buffer***
  - On miss check stream buffer
- Works with data blocks too:
  - 1 data stream buffer gets 25% misses from 4KB DM cache; 4 streams get 43%
  - For scientific programs: 8 streams got 50% to 70% of misses from 2 64KB, 4-way set associative caches
- Prefetching relies on having extra memory bandwidth that can be used without penalty

111 111

Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- 3 Cs: Compulsory, Capacity, Conflict Misses
- Reducing Miss Rate
  1. Larger Block Size
  2. Higher Associativity
  3. Victim Cache
  4. Pseudo-Associativity
  5. HW Prefetching Instr, Data

112 112

Pros and cons – Re-visit cache design choices

Larger cache block size

- Pros
  - Reduces miss rate
- Cons
  - Increases miss penalty

Important factors deciding cache performance: hit time, miss rate, miss penalty

113 113

Pros and cons – Re-visit cache design choices

Bigger cache

- Pros
  - Reduces miss rate
- Cons
  - May increases hit time
  - My increase cost and power consumption

114 114

Pros and cons – Re-visit cache design choices

Higher associativity

- Pros
  - Reduces miss rate
- Cons
  - Increases hit time

115 115

Pros and cons – Re-visit cache design choices

Multiple levels of caches

- Pros
  - Reduces miss penalty
- Cons
  - Increases cost and power consumption

116 116

Multilevel Cache Design Considerations

- Design considerations for L1 and L2 caches are very different
  - Primary cache should focus on minimizing hit time in support of a shorter clock cycle
    - Smaller cache with smaller block sizes
  - Secondary cache (s) should focus on reducing miss rate to reduce the penalty of long main memory access times
    - Larger cache with larger block sizes and/or higher associativity

117

Key Cache Design Parameters

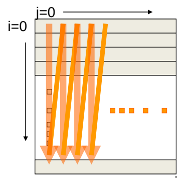
	L1 typical	L2 typical
Total size (blocks)	250 to 2000	4000 to 250,000
Total size (KB)	16 to 64	500 to 8000
Block size (B)	32 to 64	32 to 128
Miss penalty (clocks)	10 to 25	100 to 1000
Miss rates (global for L2)	2% to 5%	0.1% to 2%

118

Reducing Miss rate with programming

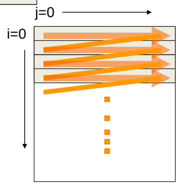
Examples:
cold cache, 4-byte words, 4-word cache blocks

```
int sumarraycols(int a[M][N])
{
    int i, j, sum = 0;

    for (j = 0; j < N; j++)
        for (i = 0; i < M; i++)
            sum += a[i][j];
    return sum;
}
```



Miss rate = ~ 100%

```
int sumarrayrows(int a[M][N])
{
    int i, j, sum = 0;

    for (i = 0; i < M; i++)
        for (j = 0; j < N; j++)
            sum += a[i][j];
    return sum;
}
```



119 **Miss rate = ~1/N**

Reducing Miss Penalty

120 120

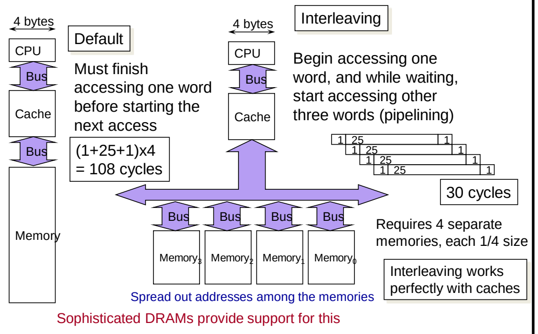
The cost of a cache miss

- For a memory access, assume:
  - 1 clock cycle to send address to memory
  - 25 clock cycles for each DRAM access (clock cycle 2ns, 50 ns access time)
  - 1 clock cycle to send each resulting data word

This actually depends on the bus speed

- Miss access time (4-word block)
  - 4 x (Address + access + sending data word)
  - 4 x (1 + 25 + 1) = 108
    = 108 cycles for each miss

Memory Interleaving


4 bytes
CPU
Bus
Cache
Bus
Memory
Default
Must finish accessing one word before starting the next access
(1+25+1)x4 = 108 cycles
4 bytes
CPU
Bus
Cache
Interleaving
Begin accessing one word, and while waiting, start accessing other three words (pipelining)
1 25 1
30 cycles
Bus
Memory3
Memory2
Memory1
Memory0
Requires 4 separate memories, each 1/4 size
Interleaving works perfectly with caches
Spread out addresses among the memories

Sophisticated DRAMs provide support for this

Memory Interleaving: An Example

Given the following system parameters with single cache level L_1:

Block size=1 word Memory bus width=1 word Miss rate =3% Miss penalty=27 cycles
(1 cycle to send address 25 cycles access time/word, 1 cycle to send a word)
Memory access/instruction = 1.2 Ideal CPI (ignoring cache misses) = 2
Miss rate (block size=2 words) = 2% Miss rate (block size=4 words) =1%

- The CPI of the base machine with 1-word blocks = 2+(1.2 x 0.03 x 27) = 2.97
- Increasing the block size to two words gives the following CPI:
  - **32-bit bus and memory, no interleaving = 2 + (1.2 x .02 x 2 x 27) = 3.29**
  - **32-bit bus and memory, interleaved = 2 + (1.2 x .02 x (28)) = 2.67**
- Increasing the block size to four words; resulting CPI:
  - **32-bit bus and memory, no interleaving = 2 + (1.2 x 0.01 x 4 x 27) = 3.29**
  - **32-bit bus and memory, interleaved = 2 + (1.2 x 0.01 x (30)) = 2.36**

Summary

$$CPUtime = IC \times \left(CPI_{Execution} + \frac{Memory\ \ accesses}{Instruction} \times \textbf{Miss rate} \times Miss\ \ penalty \right) \times Clock\ \ cycle\ \ time$$

- Interleaving
- Multiple levels of caches
- Other advanced techniques…

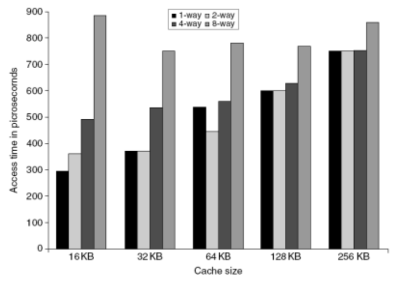
Cache Optimization

- Six basic cache optimizations:
  - Larger block size
    - Reduces compulsory misses
    - Increases capacity and conflict misses, increases miss penalty
  - Larger total cache capacity to reduce miss rate
    - Increases hit time, increases power consumption
  - Higher associativity
    - Reduces conflict misses
    - Increases hit time, increases power consumption
  - Higher number of cache levels
    - Reduces overall memory access time
  - Giving priority to read misses over writes
    - Reduces miss penalty
  - Avoiding address translation in cache indexing
    - Reduces hit time

Ten Advanced Optimizations

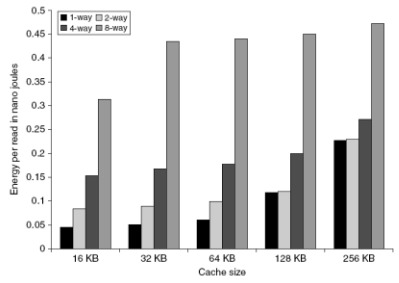
- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

L1 Size and Associativity



Access time vs. size and associativity

L1 Size and Associativity



Energy per read vs. size and associativity

Way Prediction

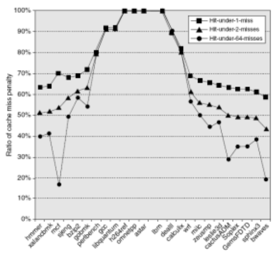
- To improve hit time, predict the way to pre-set mux
  - Mis-prediction gives longer hit time
  - Prediction accuracy
    - > 90% for two-way
    - > 80% for four-way
    - I-cache has better accuracy than D-cache
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8
- Extend to predict block as well
  - "Way selection"
  - Increases mis-prediction penalty

Pipelining Cache

- Pipeline cache access to improve bandwidth
  - Examples:
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
- Increases branch mis-prediction penalty
- Makes it easier to increase associativity

Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- L2 must support this
- In general, processors can hide L1 miss penalty but not L2 miss penalty



Multibanked Caches

- Organize cache as independent banks to support simultaneous access
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Interleave banks according to block address

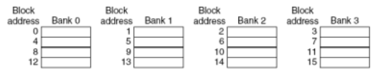


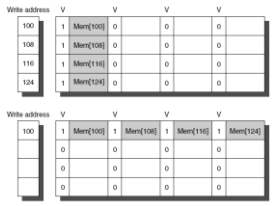
Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

Critical Word First, Early Restart

- Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- Early restart
  - Request words in normal order
  - Send missed work to the processor as soon as it arrives
- Effectiveness of these strategies depends on block size and likelihood of another access to the portion of the block that has not yet been fetched

Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Do not apply to I/O addresses



No write buffering

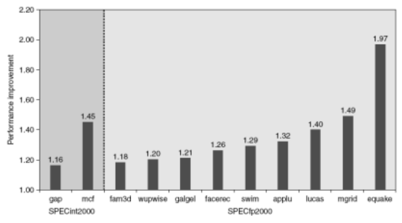
Write buffering

Compiler Optimizations

- Loop Interchange
  - Swap nested loops to access memory in sequential order
- Blocking
  - Instead of accessing entire rows or columns, subdivide matrices into blocks
  - Requires more memory accesses but improves locality of accesses

Hardware Prefetching

- Fetch two blocks on miss (include next sequential block)

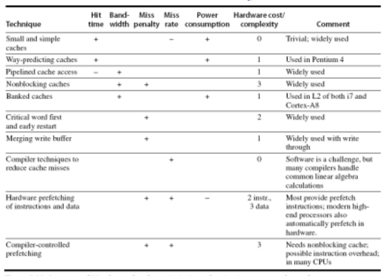


Pentium 4 Pre-fetching

Compiler Prefetching

- Insert prefetch instructions before data is needed
- Non-faulting: prefetch doesn't cause exceptions
- Register prefetch
  - Loads data into register
- Cache prefetch
  - Loads data into cache
- Combine with loop unrolling and software pipelining

Summary



Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity	Comment
Small and simple caches	+			–	+	0	Trivial; widely used
Way-predicting caches	+				+	1	Used in Pentium 4
Pipelined cache access	–	+				1	Widely used
Nonblocking caches		+	+			3	Widely used
Banked caches		+			+	1	Used in L2 of both i7 and Cortex-A8
Critical word first and early restart			+			2	Widely used
Merging write buffer			+			1	Widely used with write through
Compiler techniques to reduce cache misses				+		0	Software is a challenge, but many compilers handle common linear algebra calculations
Hardware prefetching of instructions and data			+	+	–	2 instr., 3 data	Most provide prefetch instructions; modern high-end processors also automatically prefetch in hardware.
Compiler-controlled prefetching			+	+		3	Needs nonblocking cache; possible instruction overhead; in many CPUs

Figure 2.11 Summary of 10 advanced cache optimizations showing impact on cache performance, power consumption, and complexity. Although generally a technique helps only one factor, prefetching can reduce misses if done sufficiently early; if not, it can reduce miss penalty. + means that the technique improves the factor, – means it hurts that factor, and blank means it has no impact. The complexity measure is subjective, with 0 being the easiest and 3 being a challenge.