Lecture 11: Memory Systems
-- Cache Organization and Performance

CSE 564 Computer Architecture Summer 2017

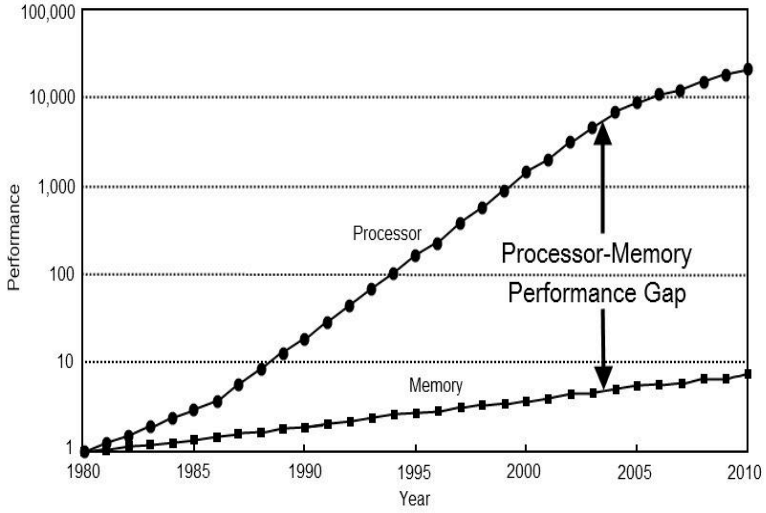
Department of Computer Science and Engineering
Yonghong Yan
yan@oakland.edu
www.secs.oakland.edu/~yan

Topics for Memory Systems

- Memory Technology and Metrics
  - SRAM, DRAM, Flash/SSD, 3-D Stack Memory
  - Locality
  - Memory access time and banking
- **Cache**
  - **Cache basics**
  - **Cache performance and optimization**
  - **Advanced optimization**
  - **Multiple-level cache, shared and private cache, prefetching**
- Virtual Memory
  - Protection, Virtualization, and Relocation
  - Page/segment, protection
  - Address Translation and TLB

Technology Challenge: Memory Wall

- Growing disparity of processor and memory speed
- DRAM: Slow cheap and dense:
  - Good choice for presenting the user with a BIG memory system
  - Used for Main memory



- SRAM: fast, expensive, and not very dense:
  - Good choice for providing the user FAST access time.
  - Used for Cache
- Speed:
  - Latency
  - Bandwidth
  - Memory interleaving

Program Behavior: Principle of Locality

- Programs tend to reuse data and instructions near those they have used recently, or that were recently referenced themselves
  - Spatial locality: Items with nearby addresses tend to be referenced close together in time
  - Temporal locality: Recently referenced items are likely to be referenced in the near future

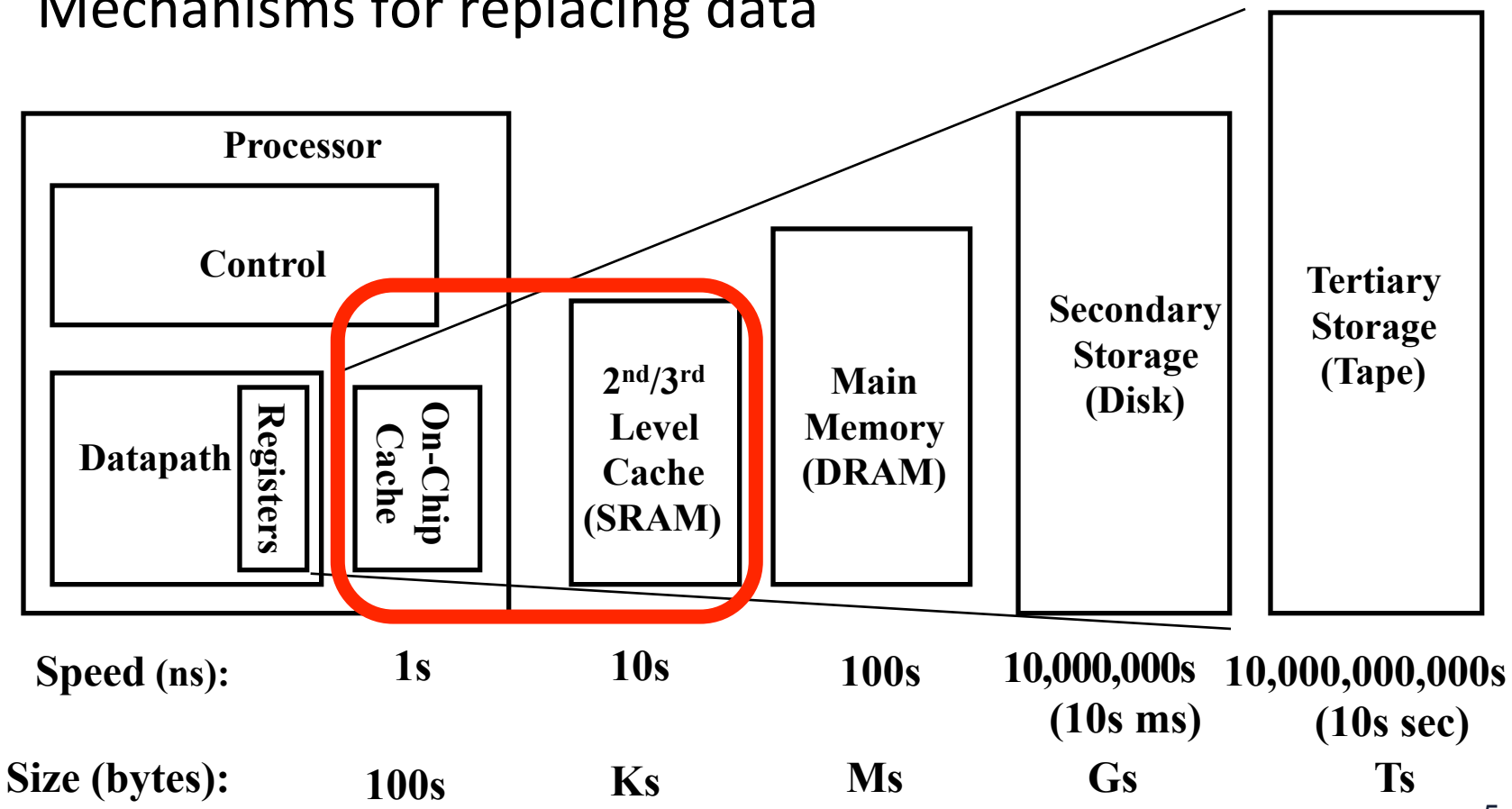
```
sum = 0;
for (i = 0; i < n; i++)
   sum += a[i];
return sum;
```

Locality Example:

- **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**

Architecture Approach: Memory Hierarchy

- Keep most recent accessed data and its adjacent data in the smaller/faster caches that are closer to processor
- Mechanisms for replacing data

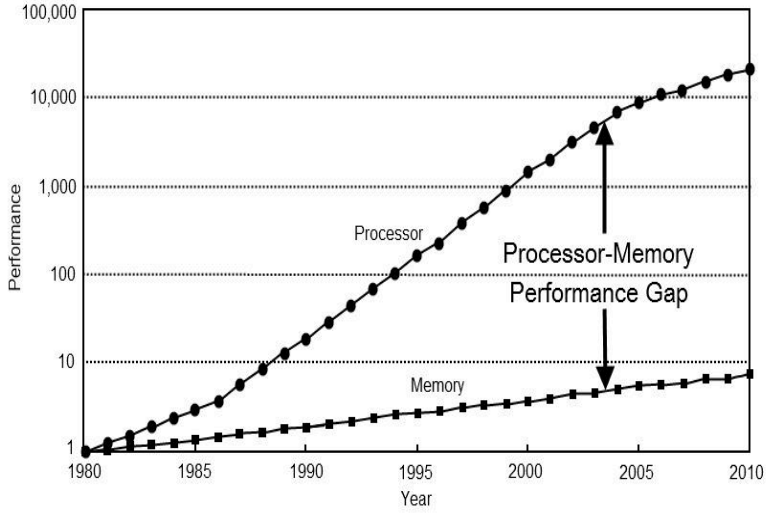


Review: Memory Technology and Hierarchy

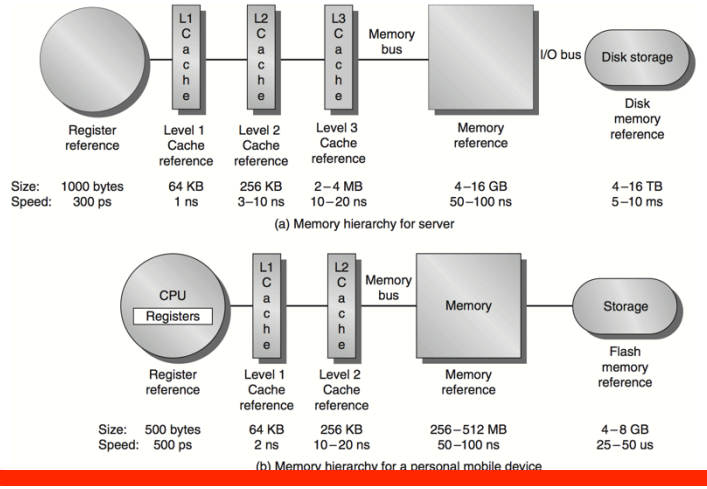
- Relationships

Program Behavior: *Principle of Locality*

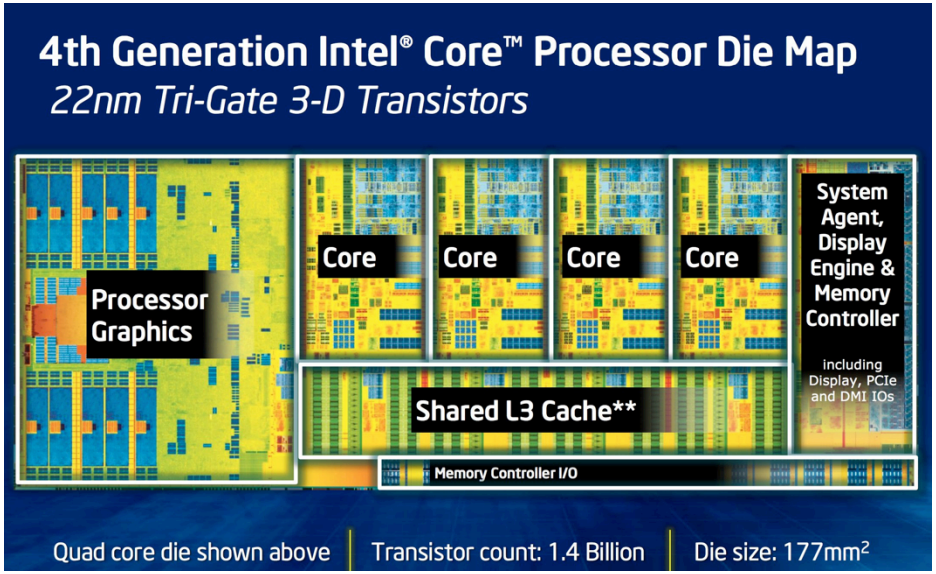
Technology challenge: *Memory Wall*

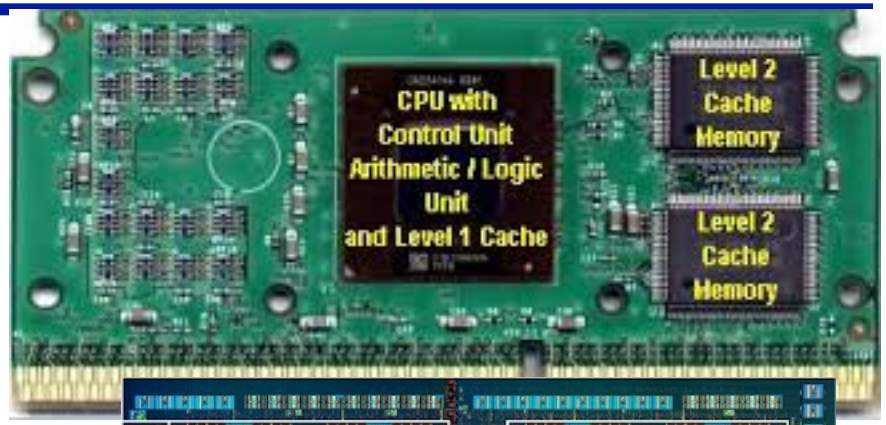


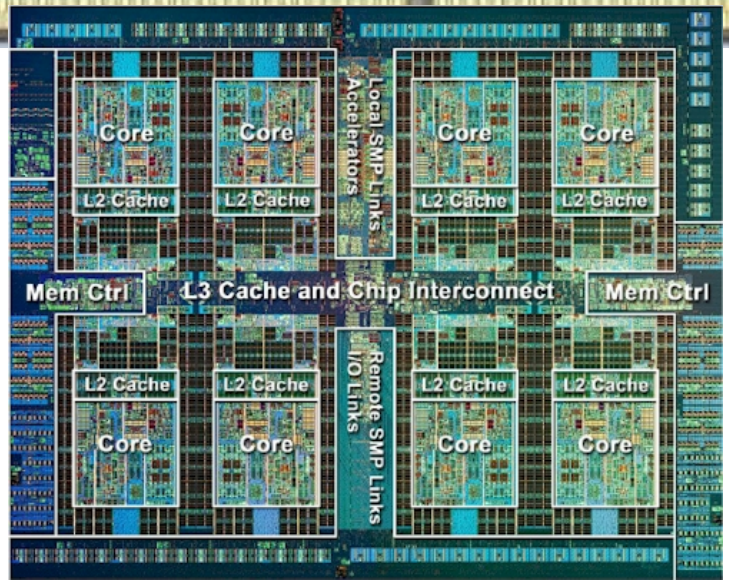
Architecture Approach: *Memory Hierarchy*



Memory Hierarchy







- *capacity*: Register << SRAM << DRAM
- *latency*: Register << SRAM << DRAM
- *bandwidth:* on-chip >> off-chip

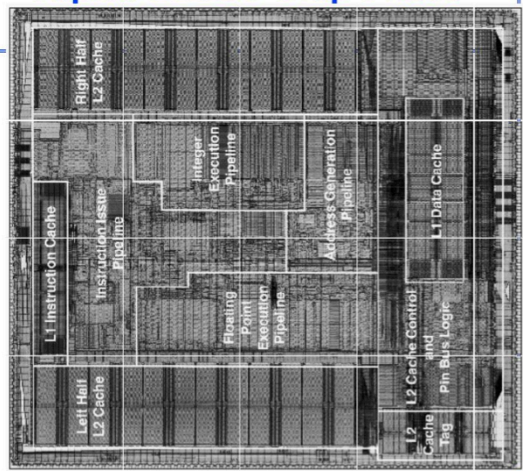
On a data access:

if data $\in$ fast memory $\Rightarrow$ low latency access *(SRAM)*

if data $\notin$ fast memory $\Rightarrow$ high latency access *(DRAM)*

History: Alpha 21164 (1994)

Alpha 21164 Chip Photo

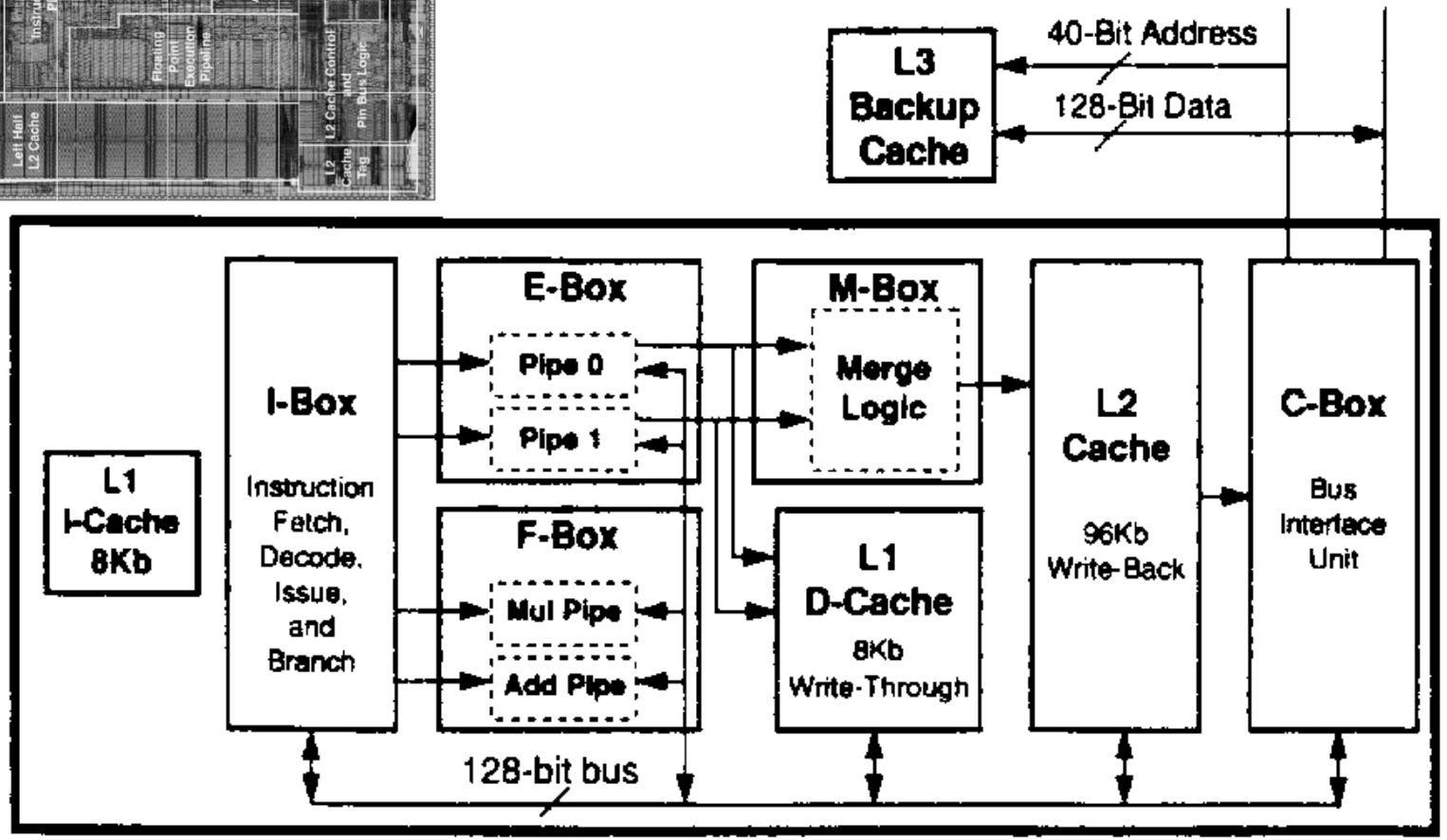


Microprocessor Report 9/12/94

Caches:

- L1 data
- L1 instruction
- L2 unified
- + L3 off-chip

https://en.wikipedia.org/wiki/Alpha_21164



History: Further Back

Ideally one would desire an indefinitely large memory capacity such that any particular ... word would be immediately available. ... We are ... forced to recognize the possibility of constructing a ***hierarchy of memories, each of which has greater capacity than the preceding but which is less quickly accessible.***

A. W. Burks, H. H. Goldstine, and J. von Neumann
***Preliminary Discussion of the Logical Design of an Electronic Computing Instrument,* 1946**

Management of Memory Hierarchy

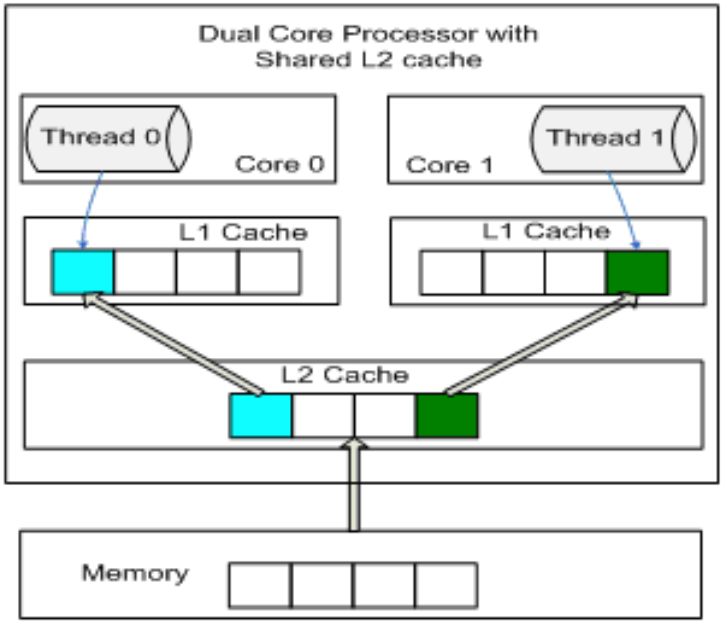
- Small/fast storage, e.g., registers
  - Address usually specified in instruction
  - Generally implemented directly as a register file
    - *but hardware might do things behind software's back, e.g., stack management, register renaming*

- Larger/slower storage, e.g., main memory
  - Address usually computed from values in register
  - Generally implemented as a hardware-managed cache hierarchy (hardware decides what is kept in fast memory)
    - *but software may provide "hints", e.g., don't cache or prefetch*

Caches exploit both types of predictability:

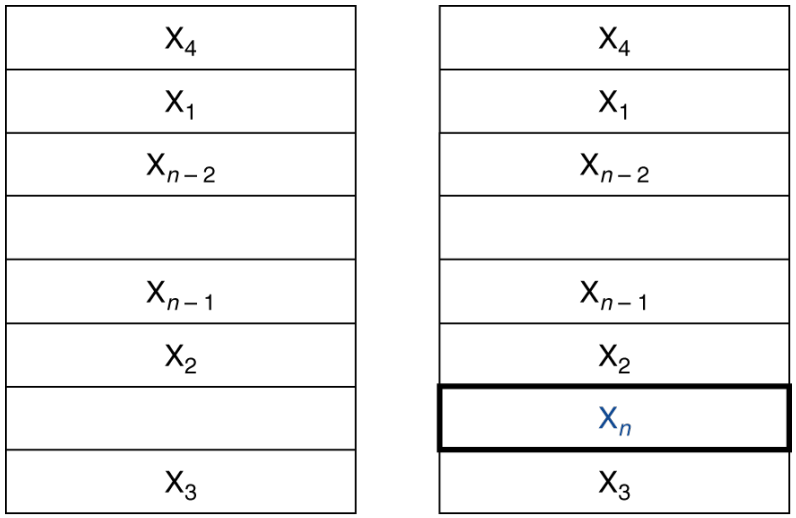
- Exploit temporal locality by remembering the contents of recently accessed locations.

- Exploit spatial locality by fetching blocks of data around recently accessed locations.



Cache Memory

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

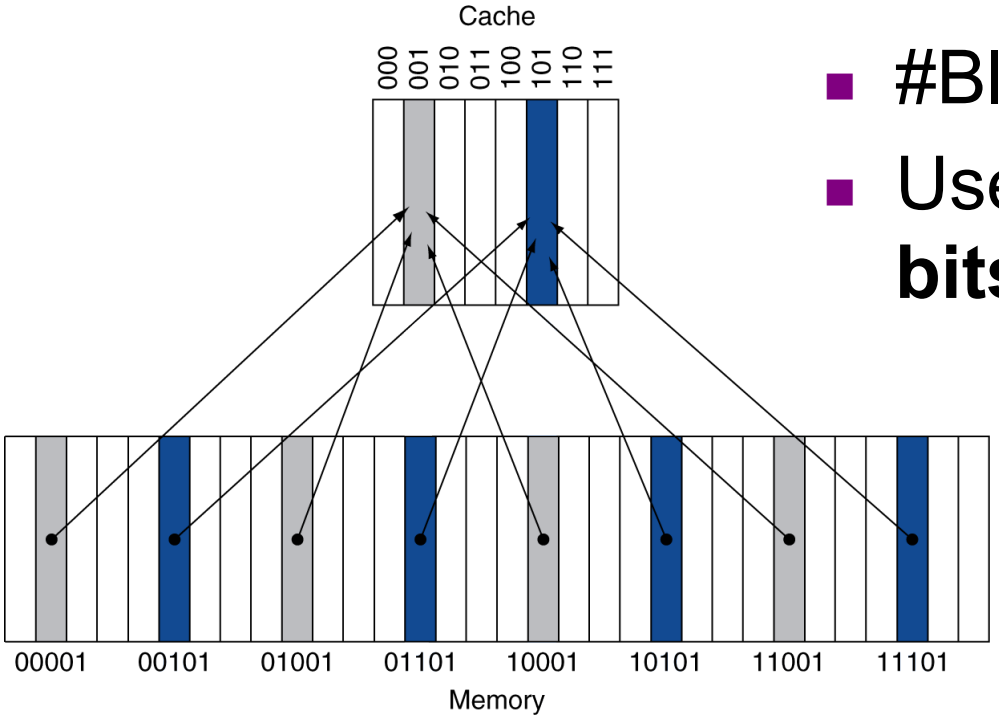


a. Before the reference to X_n b. After the reference to X_n

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

Direct Mapped Cache

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



- #Blocks is a power of 2
- Use **low-order address bits as index to the entry**

5-bit address space for total 32 bytes. This is simplified and a cache line normally contains more bytes, e.g. 64 or 128 bytes.

Tags and Valid Bits

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

Cache Example

- 8-blocks, 1 word/block, direct mapped
- Initial state

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

Cache Example

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

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

Cache Example

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

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

Cache Example

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

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

Cache Example

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

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

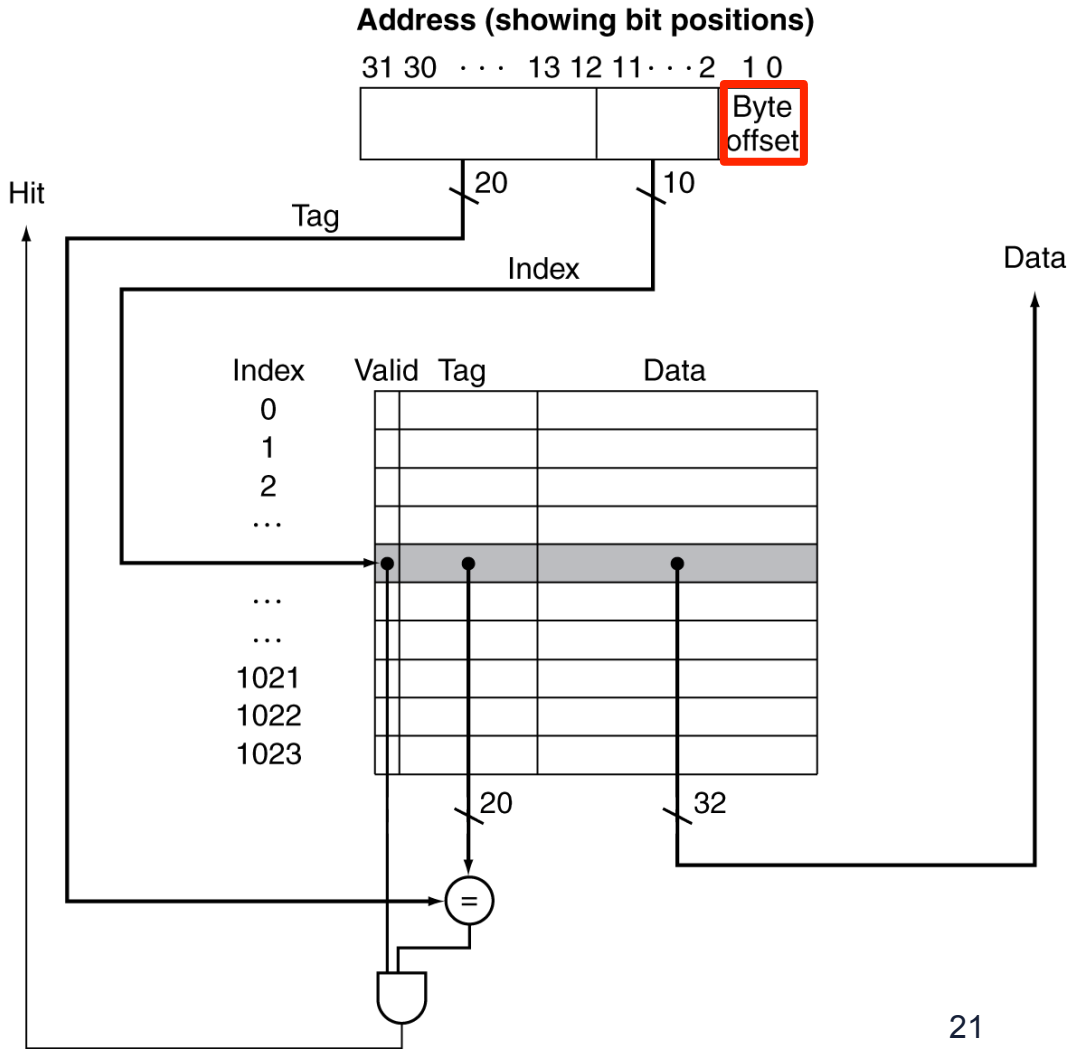
Cache Example

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

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

Address Subdivision

- 4-byte data per cache entry: block or line size
  - Cache line
- **Why do we keep multiple bytes in one cache line?**
  - **Spatial locality**

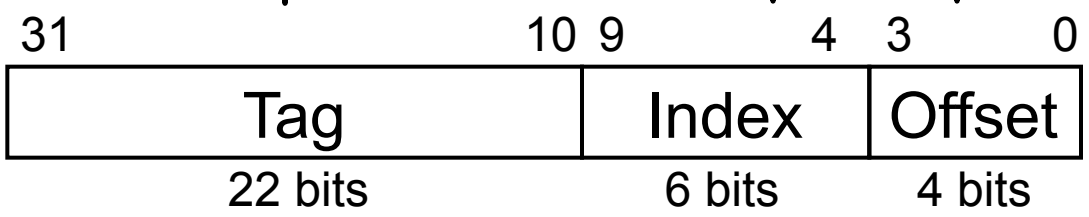


Example: Larger Block Size

- **64 blocks, 16 bytes/block**
  - To what block number does address 1200 (decimal) map?: 11

1200 = 0x4B0

0000 0000 0000 0000 0000 01**00 1011** **0000**



- Map all addresses between 1200 - 1215

Block Size Considerations

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

Cache Misses

- On cache **hit**, CPU proceeds normally
  - IF or MEM stage to access instruction or data memory
  - 1 cycle

100: LW X1 100(X2)

- On cache **miss**: → x10 or x100 cycles
  - Stall the CPU pipeline
  - Fetch block from next level of hierarchy

- **Instruction cache miss**
  - **Restart instruction fetch**
- **Data cache miss**
  - **Complete data access**

Write-Through

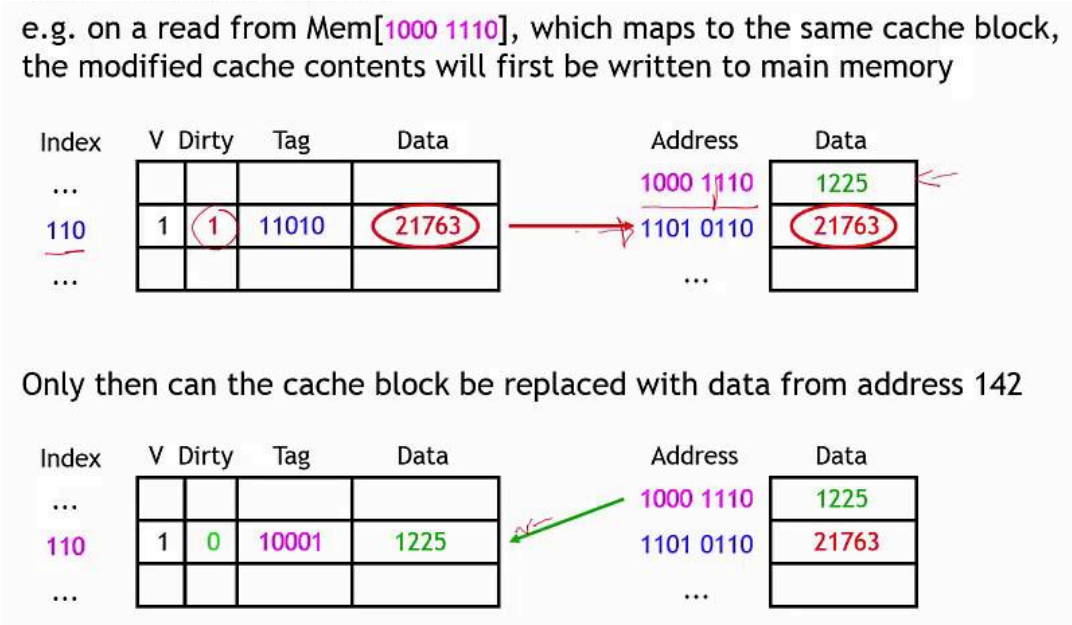
- On data-write hit, could just update the block in cache
  - But then cache and memory would be inconsistent

200: SW X1 100(X2)

- **Write through:** also update memory
  - But makes writes take longer
  - e.g., if base CPI = 1, 10% of instructions are stores, write to memory takes 100 cycles
    - Effective CPI = 1 + 0.1×100 = 11
- **Solution: write buffer**
  - Holds data waiting to be written to memory
  - CPU continues immediately
    - Only stalls on write if write buffer is already full and write multiple bytes a time

Write-Back

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



Write Allocation

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

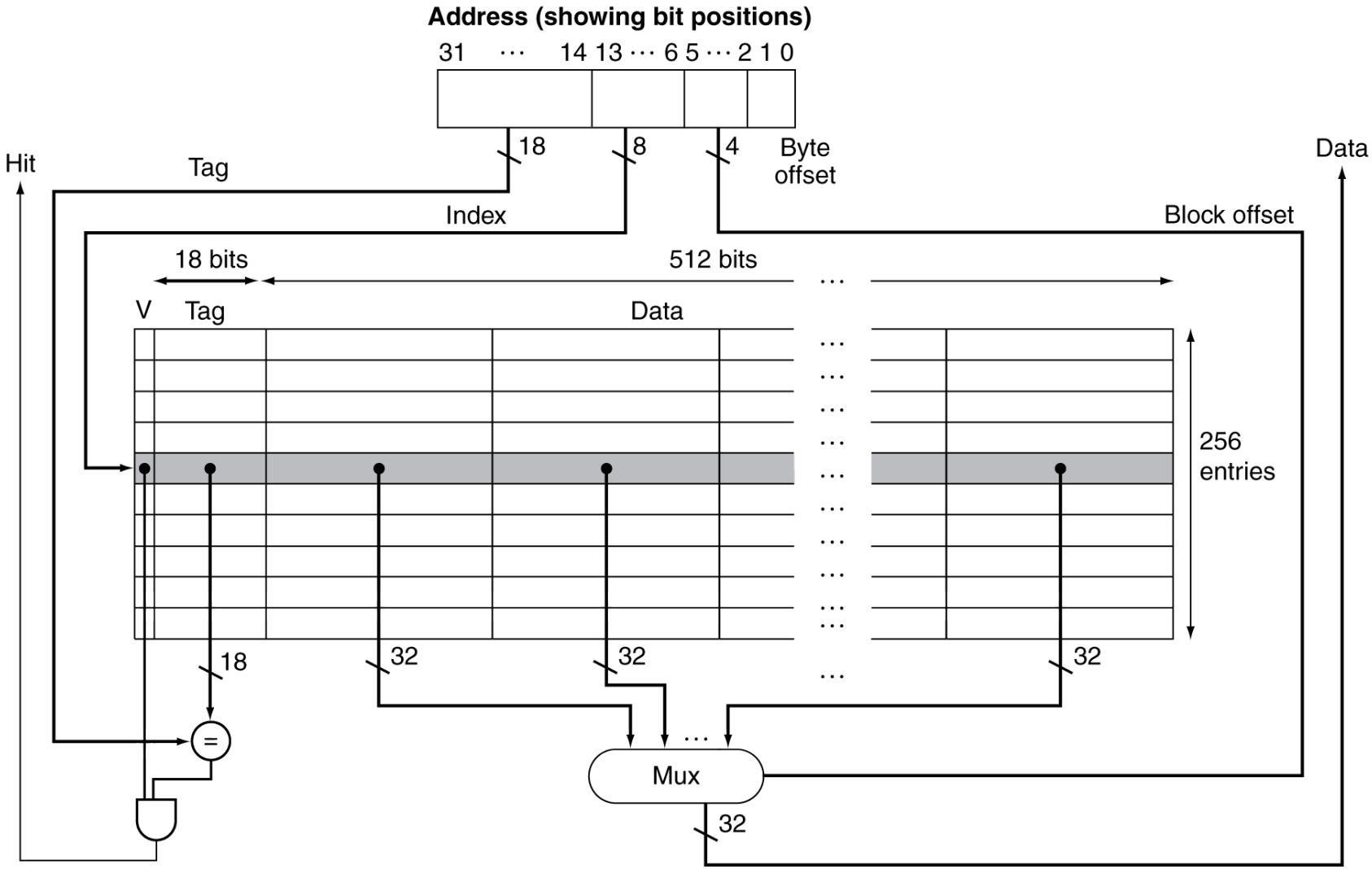
Example: Intrinsity FastMATH

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

- Split cache: separate I-cache and D-cache
  - Each 16KB: 256 blocks × 16 words/block
  - D-cache: write-through or write-back

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

Example: Intrinsity FastMATH



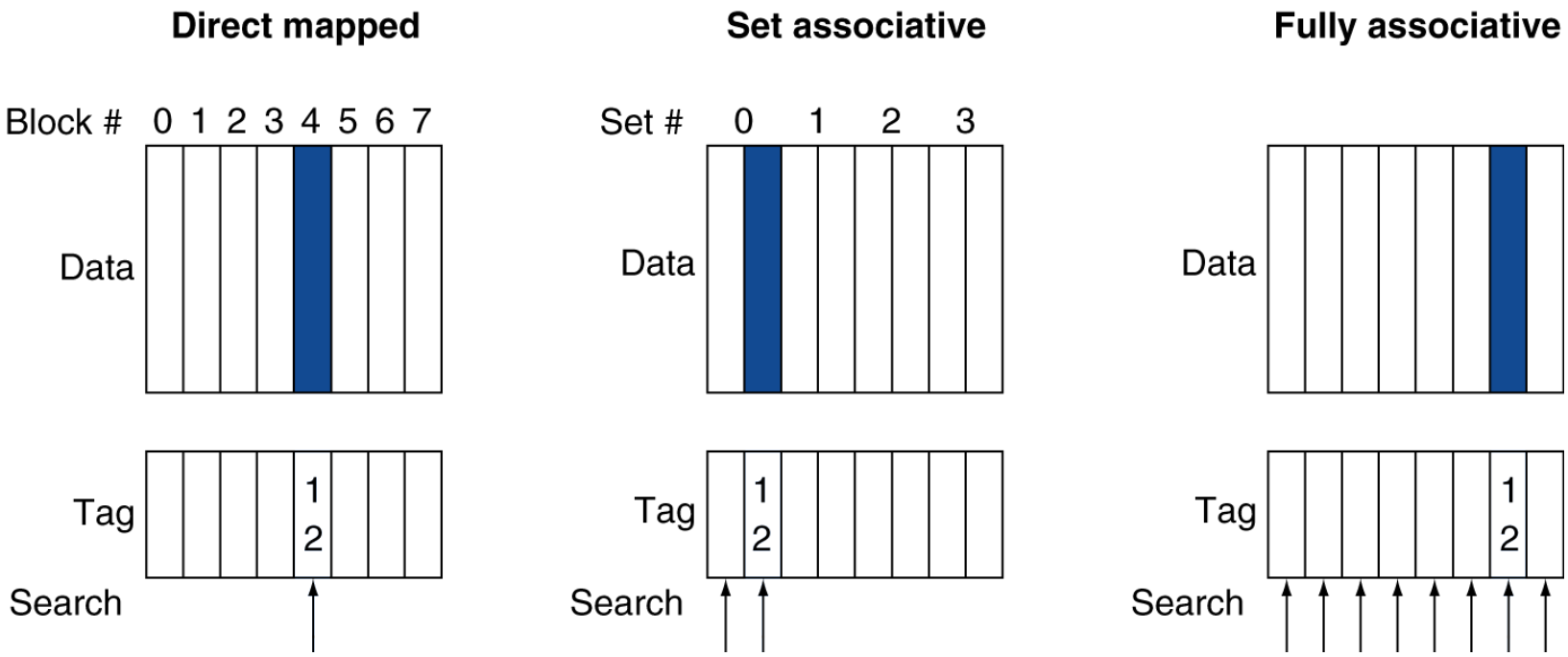
Main Memory Supporting Caches

- Use DRAMs for main memory
  - Fixed width (e.g., 1 word)
  - Connected by fixed-width clocked bus
    - Bus clock is typically slower than CPU clock
- Example cache block read
  - 1 bus cycle for address transfer
  - 15 bus cycles per DRAM access
  - 1 bus cycle per data transfer
- For 4-word block, 1-word-wide DRAM
  - Miss penalty = 1 + 4×15 + 4×1 = 65 bus cycles
  - Bandwidth = 16 bytes / 65 cycles = 0.25 B/cycle

Associative Caches

- Fully associative
  - Allow a given block to go in any cache entry
  - Requires all entries to be searched at once
  - Comparator per entry (expensive)
- *n*-way set associative
  - Each set contains *n* entries
  - Block number determines which set
    - (Block number) modulo (#Sets in cache)
  - Search all entries in a given set at once
  - *n* comparators (less expensive)

Associative Cache Example



Spectrum of Associativity

- For a cache with 8 entries

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

Associativity Example

- Compare 4-block caches
  - Direct mapped, 2-way set associative, fully associative
  - Block access sequence: 0, 8, 0, 6, 8
- Direct mapped

Access sequence

Block address	Cache index	Hit/miss	Cache content after access			
			0	1	2	3
0	0	miss	**Mem[0]**			
8	0	miss	**Mem[8]**			
0	0	miss	**Mem[0]**			
6	2	miss	Mem[0]		**Mem[6]**	
8	0	miss	**Mem[8]**		Mem[6]	

Associativity Example

- 2-way set associative

Block address	Cache index	Hit/miss	Cache content after access			
			Set 0		Set 1	
0	0	miss	**Mem[0]**			
8	0	miss	Mem[0]	**Mem[8]**		
0	0	hit	**Mem[0]**	Mem[8]		
6	0	miss	Mem[0]	**Mem[6]**		
8	0	miss	**Mem[8]**	Mem[6]		

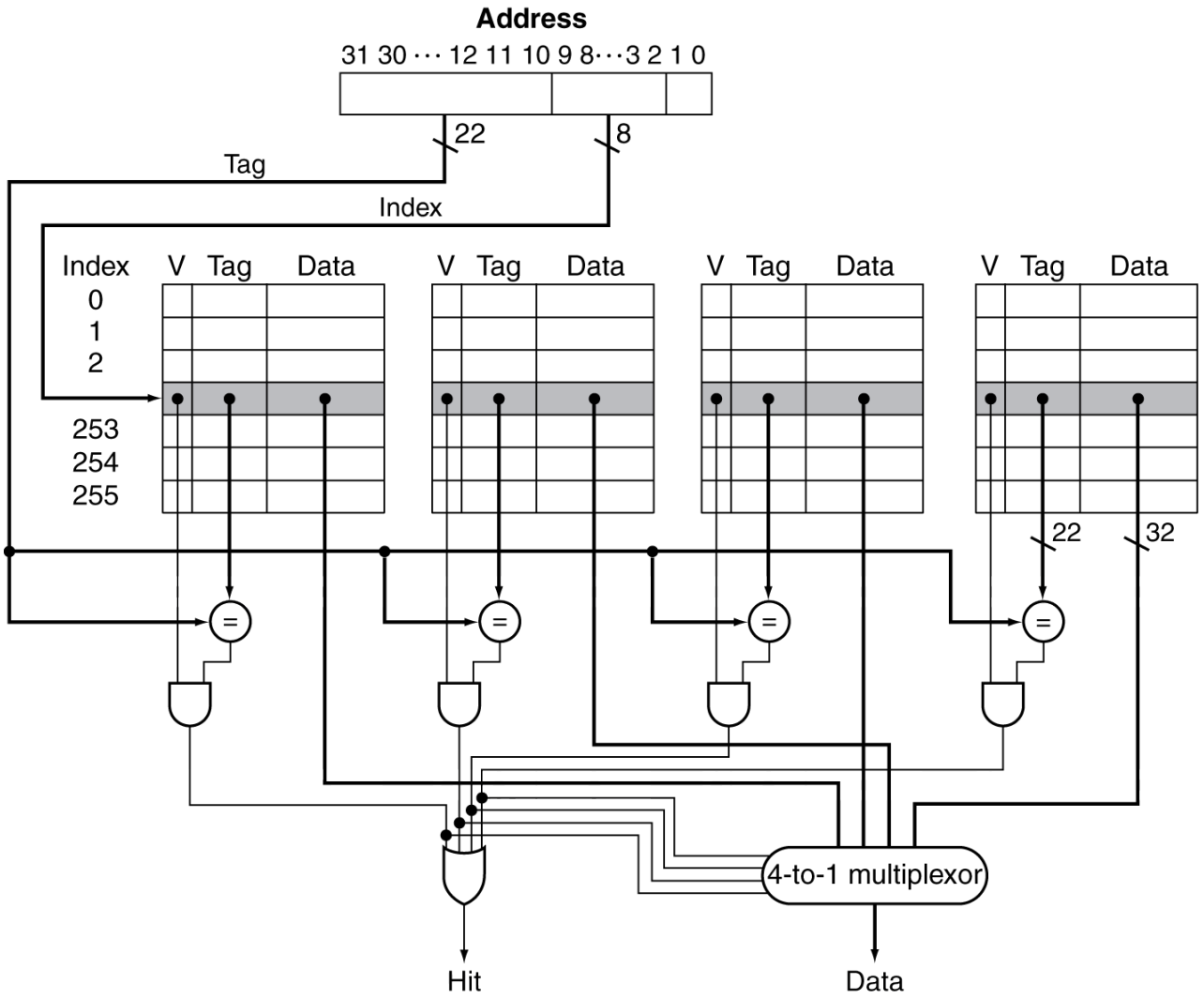
- Fully associative

Block address		Hit/miss	Cache content after access			
0		miss	**Mem[0]**			
8		miss	Mem[0]	**Mem[8]**		
0		hit	**Mem[0]**	Mem[8]		
6		miss	Mem[0]	Mem[8]	**Mem[6]**	
8		hit	Mem[0]	**Mem[8]**	Mem[6]	

How Much Associativity

- Increased associativity decreases miss rate
  - But with diminishing returns
- Simulation of a system with 64KB D-cache, 16-word blocks, SPEC2000
  - 1-way: 10.3%
  - 2-way: 8.6%
  - 4-way: 8.3%
  - 8-way: 8.1%

Set Associative Cache Organization



Replacement Policy

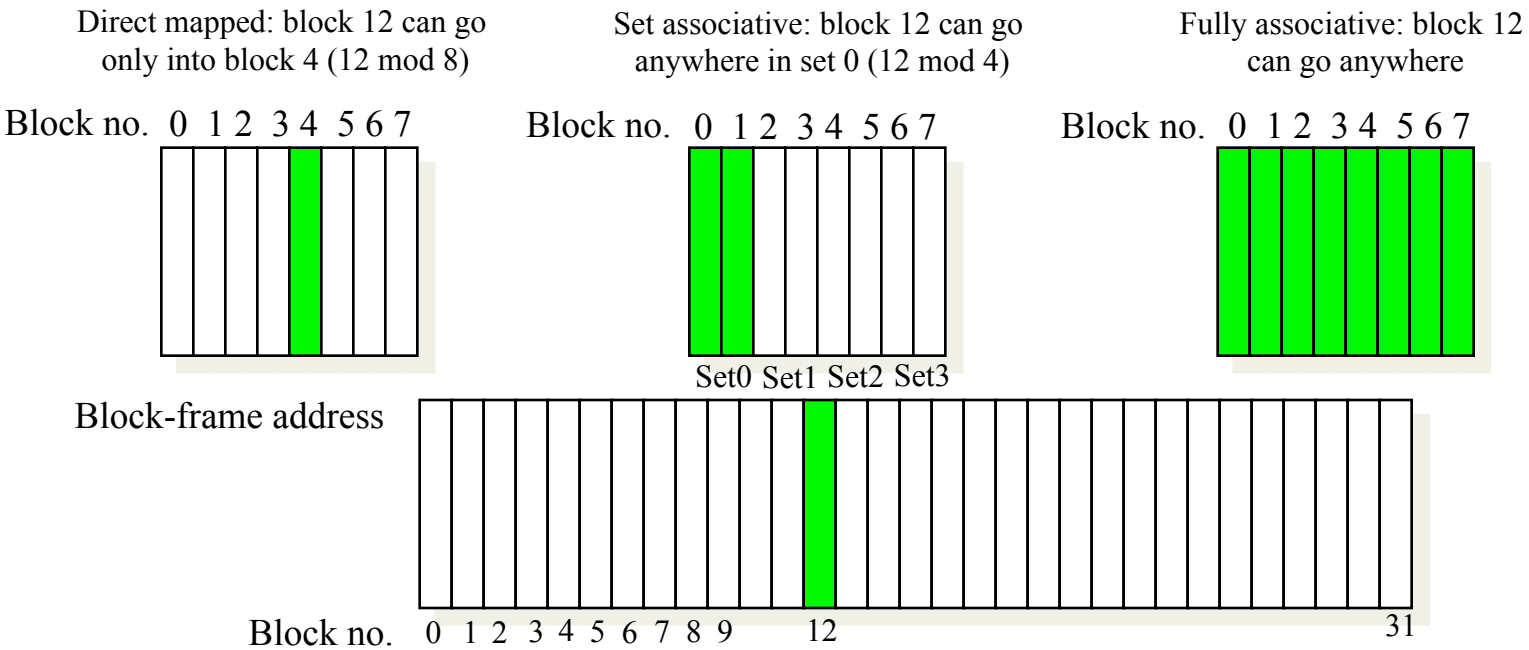
- Direct mapped: no choice
- Set associative
  - Prefer non-valid entry, if there is one
  - Otherwise, choose among entries in the set
- Least-recently used (LRU)
  - Choose the one unused for the longest time
    - Simple for 2-way, manageable for 4-way, too hard beyond that
- Random
  - Gives approximately the same performance as LRU for high associativity

4 Questions for Cache Organization Review

- Q1: Where can a block be placed in the upper level?
  - Block placement
- Q2: How is a block found if it is in the upper level?
  - Block identification
- Q3: Which block should be replaced on a miss?
  - Block replacement
- Q4: What happens on a write?
  - Write strategy

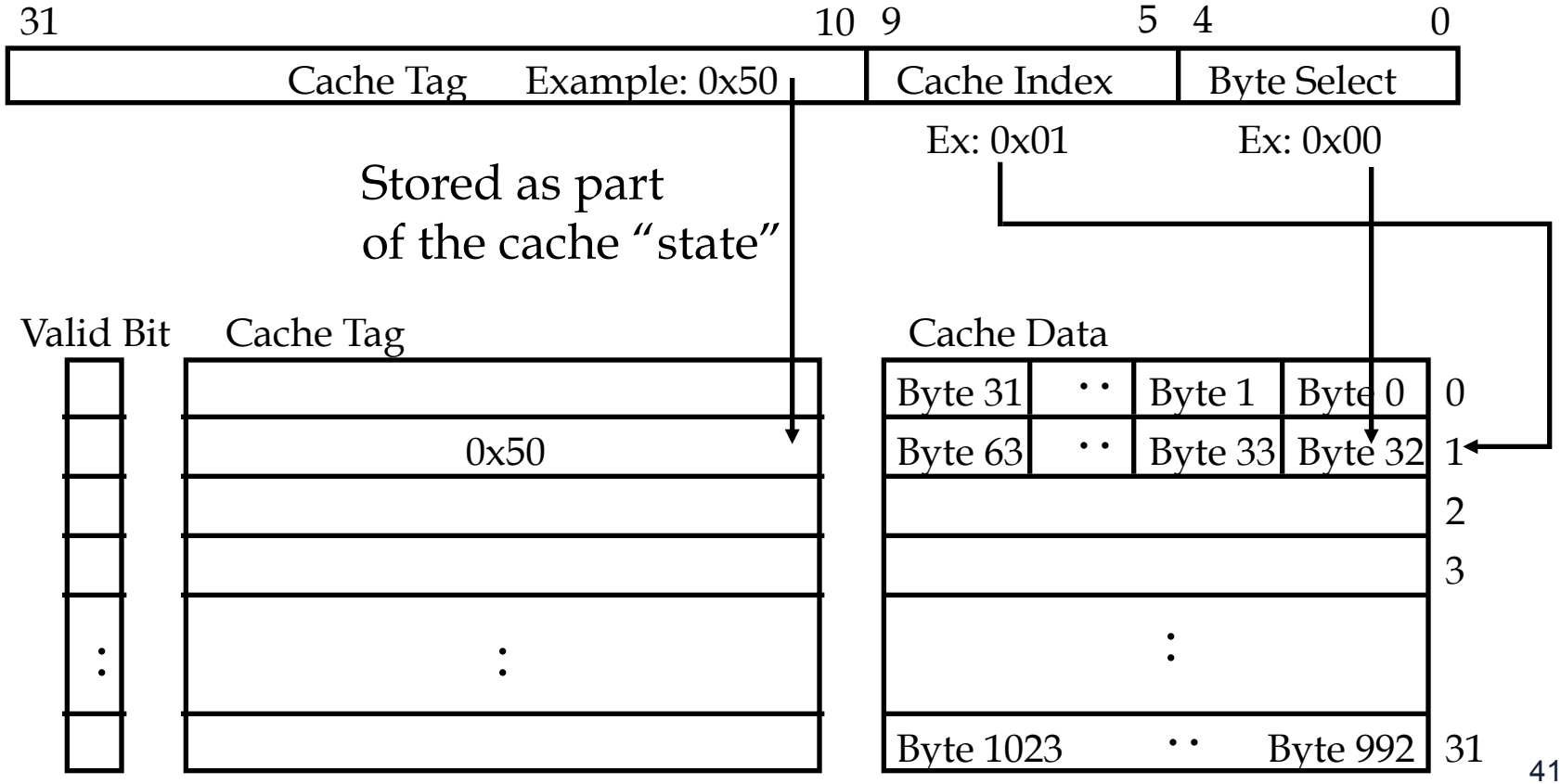
Q1: Where Can a Block be Placed in The Upper Level?

- Block Placement
  - Direct Mapped, Fully Associative, Set Associative
    - Direct mapped: (Block number) mod (Number of blocks in cache)
    - Set associative: (Block number) mod (Number of sets in cache)
      - # of set ≤ # of blocks
      - *n*-way: *n* blocks in a set
      - 1-way = direct mapped
    - Fully associative: # of set = 1



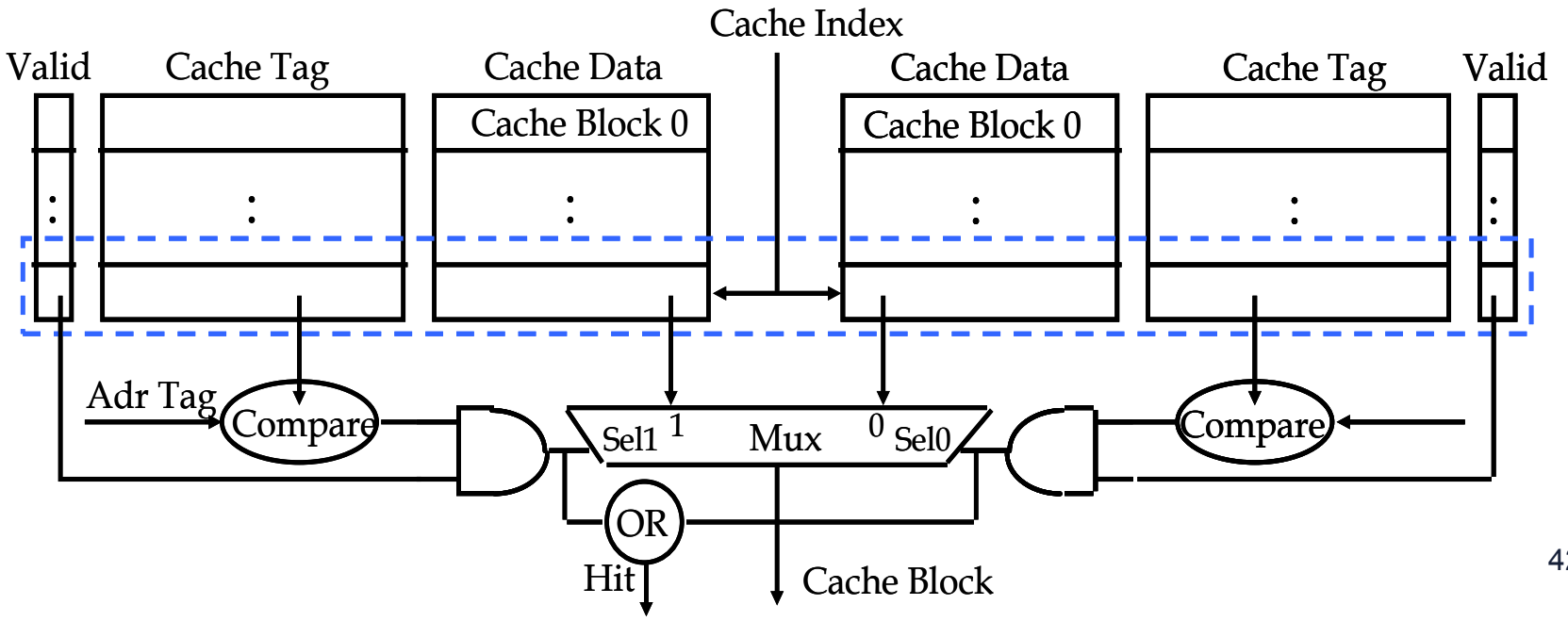
1 KB Direct Mapped Cache, 32B blocks

- For a 2^N byte cache
  - The uppermost (32 - N) bits are always the **Cache Tag**
  - The lowest M bits are the **Byte Select** (Block Size = 2^M)



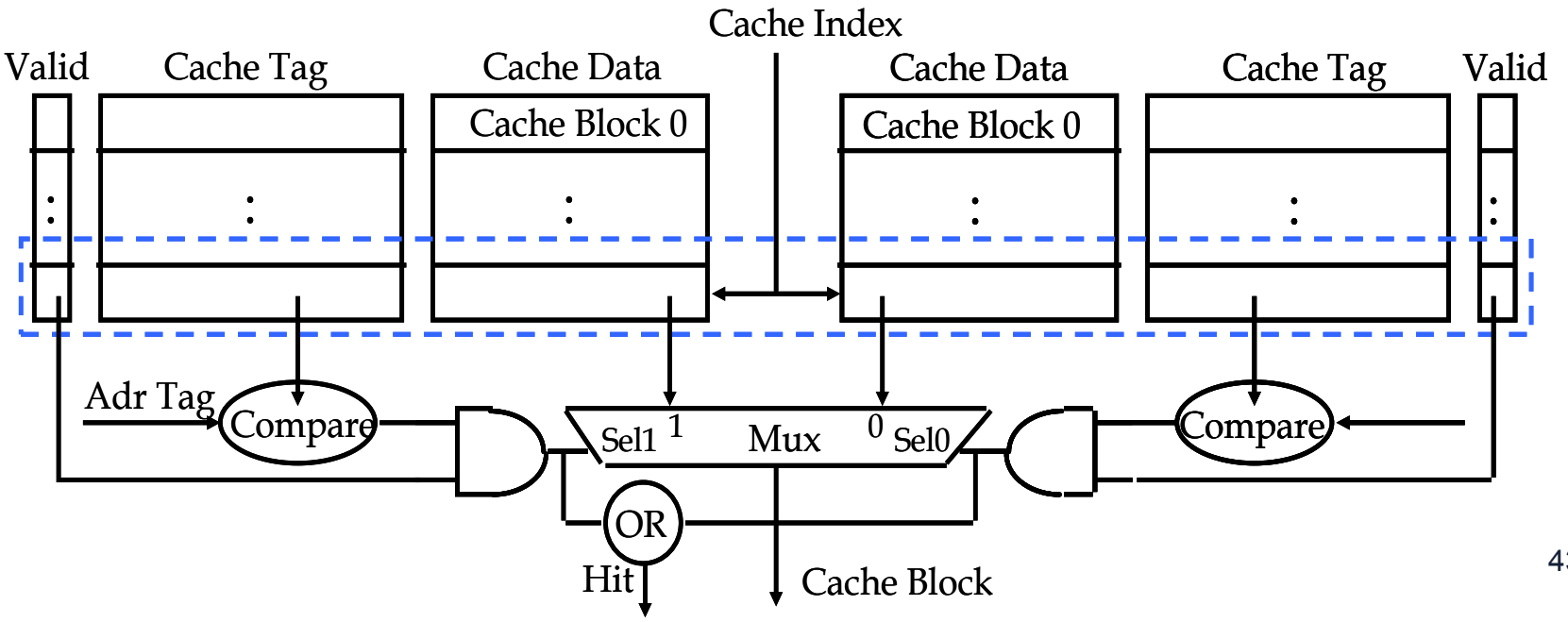
Set Associative Cache

- *N*-way set associative: *N* entries for each **Cache Index**
  - *N* direct mapped caches operates in parallel
- Example: Two-way set associative cache
  - **Cache Index** selects a "set" from the cache;
  - The two tags in the set are compared to the input in parallel;
  - Data is selected based on the tag result.



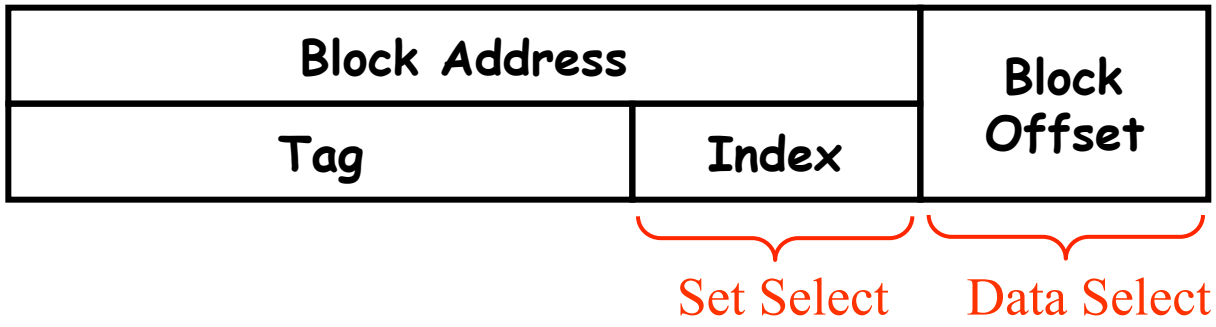
Disadvantage of Set Associative Cache

- *N*-way Set Associative Cache versus Direct Mapped Cache:
  - N comparators vs. 1
  - Extra MUX delay for the data
  - Data comes AFTER Hit/Miss decision and set selection
- In a direct mapped cache, Cache Block is available BEFORE Hit/Miss:
  - Possible to assume a hit and continue. Recover later if miss.



Q2: Block Identification

- Tag on each block
  - No need to check index or block offset
- Increasing associativity shrinks index, expands tag



$$\text{Cache size} = \text{Associativity} \times 2^{\text{index\_size}} \times 2^{\text{offest\_size}}$$

Q3: Which block should be replaced on a miss?

- Easy for Direct Mapped
- Set Associative or Fully Associative
  - Random
  - LRU (Least Recently Used)
  - First in, first out (FIFO)

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

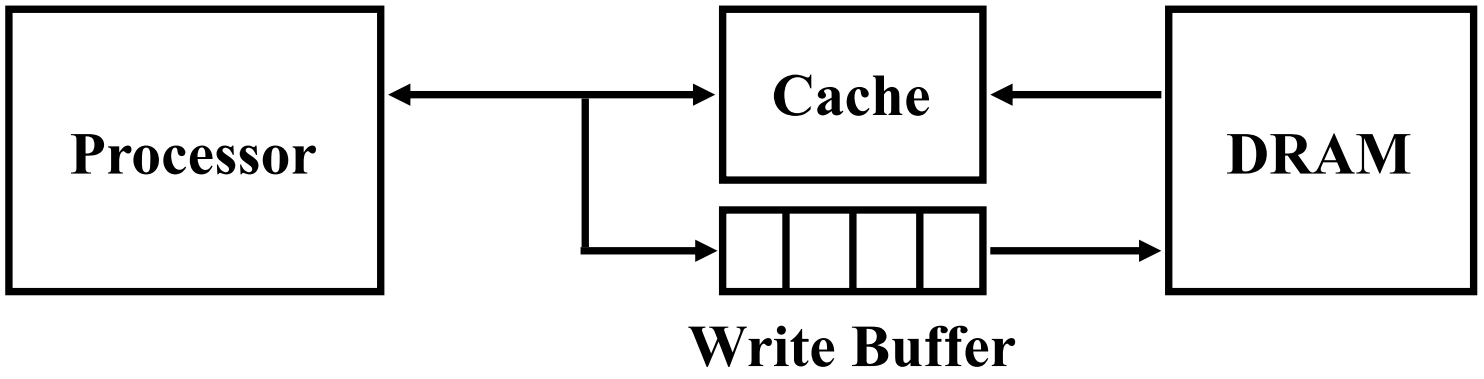
Figure B.4 Data cache misses per 1000 instructions comparing least recently used, random, and first in, first out replacement for several sizes and associativities. There is little difference between LRU and random for the largest size cache, with LRU outperforming the others for smaller caches. FIFO generally outperforms random in the smaller cache sizes. These data were collected for a block size of 64 bytes for the Alpha architecture using 10 SPEC2000 benchmarks. Five are from SPECint2000 (gap, gcc, gzip, mcf, and perl) and five are from SPECfp2000 (applu, art, equake, lucas, and swim). We will use this computer and these benchmarks in most figures in this appendix.

Q4: What Happens on a Write?

	Write-Through	Write-Back
Policy	Data written to cache block, also written to lower-level memory	1. Write data only to the cache 2. Update lower level when a block falls out of the cache
Debug	Easy	Hard
Do read misses produce writes?	No	Yes
Do repeated writes make it to lower level?	Yes	No

Additional option -- let writes to an un-cached address allocate a new cache line ("write-allocate").

Write Buffers for Write-Through Caches



- Q. Why a write buffer ?
  - A. So CPU doesn't stall
- Q. Why a buffer, why not just one register ?
  - A. Bursts of writes are common.
- Q. Are Read After Write (RAW) hazards an issue for write buffer?
  - A. Yes! Drain buffer before next read, or send read 1st after check write buffers.

Write-Miss Policy

- Two options on a write miss
  - Write allocate – the block is allocated on a write miss, followed by the write hit actions.
    - Write misses act like read misses.
  - No-write allocate – write misses do not affect the cache. The block is modified only in the lower-level memory.
    - Block stay out of the cache in no-write allocate until the program tries to read the blocks, but with write allocate even blocks that are only written will still be in the cache.

Write-Miss Policy Example

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

  Write Mem[100];
  Write Mem[100];
  Read Mem[200];
  Write Mem[200];
  Write Mem[100].

  What are the number of hits and misses (inclusive reads and writes) when using no-write allocate versus write allocate?

- *Answer*

No-write Allocate:

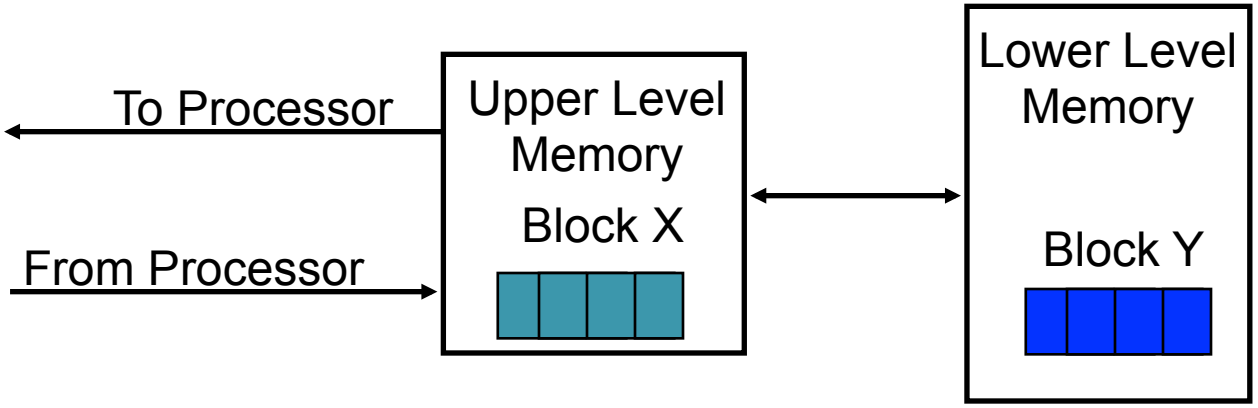
Write Mem[100]; 1 write miss
Write Mem[100]; 1 write miss
Read Mem[200]; 1 read miss
Write Mem[200]; 1 write hit
Write Mem[100]. 1 write miss
4 misses; 1 hit

Write allocate:

Write Mem[100]; 1 write miss
Write Mem[100]; 1 write hit
Read Mem[200]; 1 read miss
Write Mem[200]; 1 write hit
Write Mem[100]; 1 write hit
2 misses; 3 hits

Memory Hierarchy Performance: Terminology

- **Hit:** data appears in some block in the upper level (example: Block X)
  - **Hit Rate:** the fraction of memory access found in the upper level.
  - **Hit Time:** Time to access the upper level which consists of RAM access time + Time to determine hit/miss.
- **Miss**: data needs to be retrieve 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 the processor.
- **Hit Time << Miss Penalty (500 instructions on 21264!)**



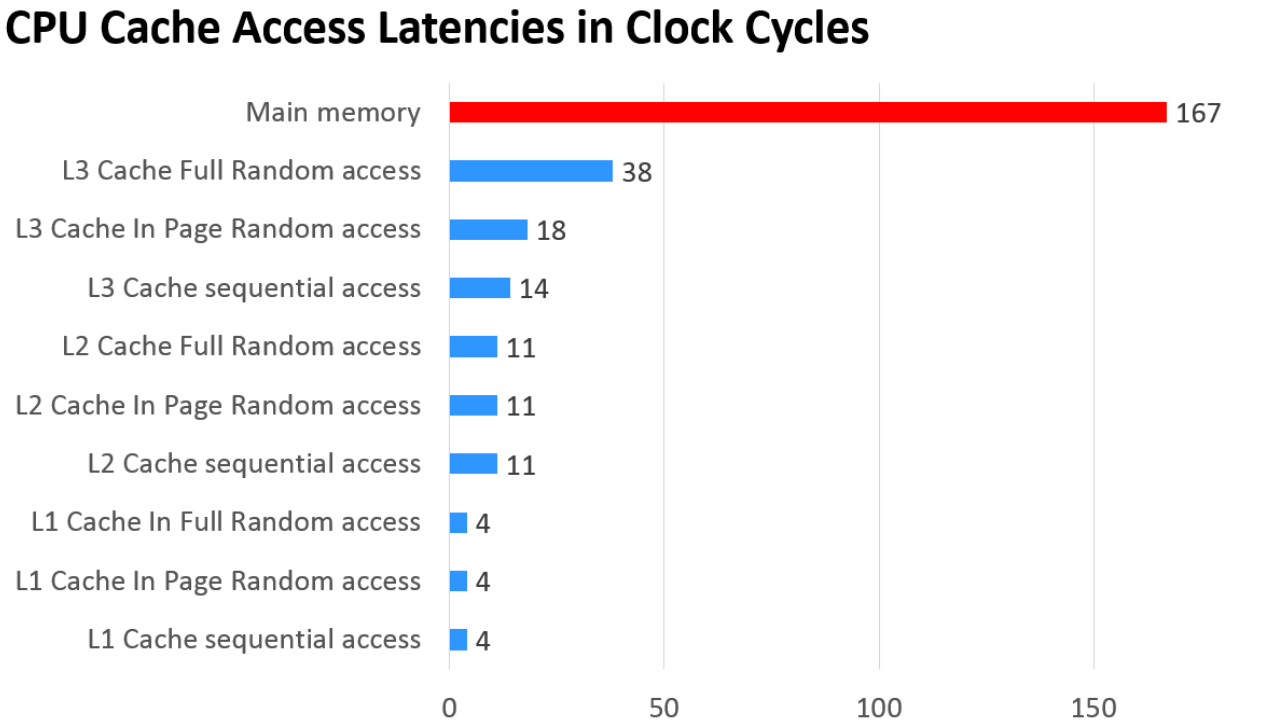
Cache Measures

- **Hit rate:** fraction found in that level
  - So high that usually talk about Miss rate
  - Miss rate fallacy: as MIPS to CPU performance, miss rate to average memory access time in memory
- Average memory-access time = Hit time + Miss rate x Miss penalty (ns or clocks)
- **Miss penalty:** time to replace a block from lower level, including time to replace in CPU
  - Access time: time to lower level = f(latency to lower level)
  - Transfer time: time to transfer block =f(BW between upper & lower levels)

CPU Performance Revisit

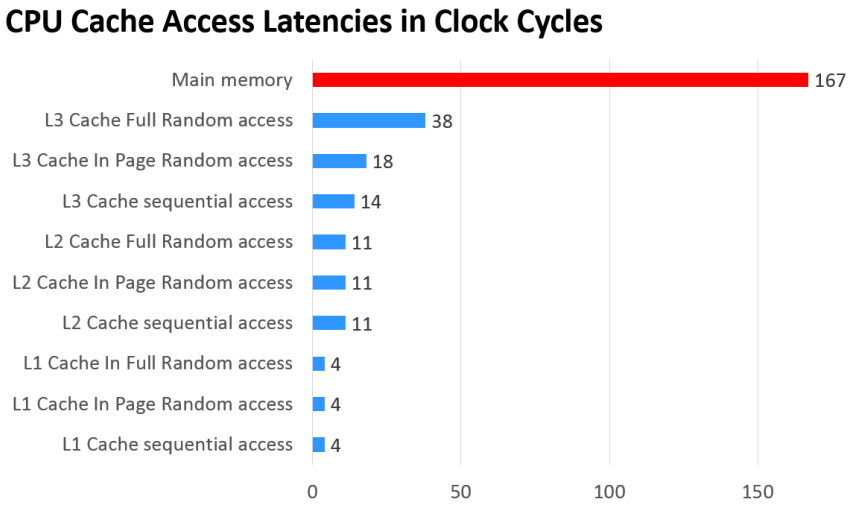
- CPU performance factors
  - Instruction count
    - Determined by ISA and compiler
  - CPI and Cycle time
    - Determined by CPU hardware

$$CPU\ \text{Time} = \frac{\text{Instructions}}{\text{Program}} * \frac{\text{Cycles}}{\text{Instruction}} * \frac{\text{Time}}{\text{Cycle}}$$



CPU Performance with Memory Factor

- Components of CPU time
  - Program execution cycles
    - Includes cache hit time
  - **Memory stall cycles**
    - **Mainly from cache misses**
- With simplifying assumptions:



http://www.7-cpu.com/cpu/Haswell.html

$$\text{Memory stall cycles}$$

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

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

Cache Performance (1/3)

- **Memory Stall Cycles:** the number of cycles during which the processor is stalled waiting for a memory access.
- Rewriting the CPU performance time

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

- The number of memory stall cycles depends on both the number of misses and the cost per miss, which is called the miss penalty:

$$\begin{aligned}\text{Memory stall cycles} &= \text{Number of misses} \times \text{Miss penalty} \\ &= \text{IC} \times \frac{\text{Misses}}{\text{Instrution}} \times \text{Miss Penalty} \\ &= \text{IC} \times \frac{\text{Memory accesses}}{\text{Instrution}} \times \text{Miss rate} \times \text{Miss Penalty}\end{aligned}$$

† The advantage of the last form is the component can be easily measured.

Cache Performance

- Given
  - I-cache miss rate = 2%
  - D-cache miss rate = 4%
  - Miss penalty = 100 cycles
  - Base CPI (ideal cache) = 2
  - Load & stores are 36% of instructions

- Miss cycles per instruction
  - I-cache: 0.02 × 100 = 2
  - D-cache: 0.36 × 0.04 × 100 = 1.44

- Actual CPI = 2 + 2 + 1.44 = 5.44
  - Ideal CPU is 5.44/2 =2.72 times faster

Cache Performance (2/3)

- Miss penalty depends on
  - Prior memory requests or memory refresh;
  - Different clocks of the processor, bus, and memory;
  - Thus, using miss penalty be a constant is a simplification.
- Miss rate: the fraction of cache access that result in a miss (i.e., number of accesses that miss divided by number of accesses).
- Extract formula for R/W

$$\text{Memory stall cycles} = \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read Miss Penalty} + \text{IC} \times \text{Writes per instruction} \times \text{Write miss rate} \times \text{Write Miss Penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Memory accesses}}{\text{Instrution}} \times \text{Miss rate} \times \text{Miss penalty}$$

† Simplify the complete formula by combining the R/W.

Example (C-5)

- Assume we have a computer where the clocks per instruction (CPI) is 1.0 when all memory accesses hit in the cache. The only data accesses are loads and stores, and these total 50% of the instructions. If the miss penalty is 25 clock cycles and the miss rate is 2%, how much faster would the computer be if all instructions were cache hits?
- Answer:

1. Compute the performance for the computer that always hits:

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$
$$= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle} = \text{IC} \times 1.0 \times \text{Clock cycle}$$

2. For the computer with the real cache, we compute memory stall cycles:

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss rate} \times \text{Miss penalty}$$
$$= \text{IC} \times (1 + 0.5) \times 0.02 \times 25 = \text{IC} \times 0.75$$

1 instruction memory access + 0.5 data memory access

3. Compute the total performance

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$
$$= 1.75 \times \text{IC} \times \text{Clock cycle}$$

4. Compute the performance ratio which is the inverse of the execution times

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

Cache Performance (3/3)

- Usually, measuring miss rate as misses per instruction rather than misses per memory reference.

$$\frac{\text{Misses}}{\text{Instrution}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory} \times \text{accesses}}{\text{Instruction}}$$

† The latter formula is useful when you know the average number of memory accesses per instruction.

- For example, in the previous example into misses per instruction:

$$\frac{\text{Misses}}{\text{Instrution}} = \text{Miss rate} \times \frac{\text{Memory} \times \text{accesses}}{\text{Instruction}} = 0.02 \times 1.5 = 0.030$$

Example (C-6)

- To show equivalency between the two miss rate equations, let's redo the example above, this time assuming a miss rate per 1000 instructions of 30. What is memory stall time in terms of instruction count?
- Answer

  Recomputing the memory stall cycles:

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

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

$$= \frac{\text{IC}}{1000} \times \frac{\text{Misses}}{\text{Instruction} \times 1000} \times \text{Miss penalty}$$

$$= \frac{\text{IC}}{1000} \times 30 \times 25$$

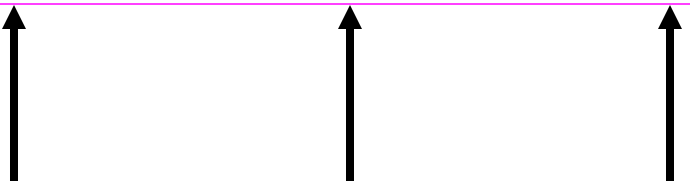
$$= \frac{\text{IC}}{1000} \times 750$$

$$= \text{IC} \times 0.75$$

Improving Cache Performance

The next few sections in the text book look at ways to improve cache and memory access times.

Average Memory Access Time = Hit Time + Miss Rate * Miss Penalty



$$CPU\ Time = IC * (CPI_{Execution} + \frac{Memory\ Accesses}{Instruction} \times Miss\ Rate \times Miss\ Penalty) \times Clock\ Cycle\ Time$$

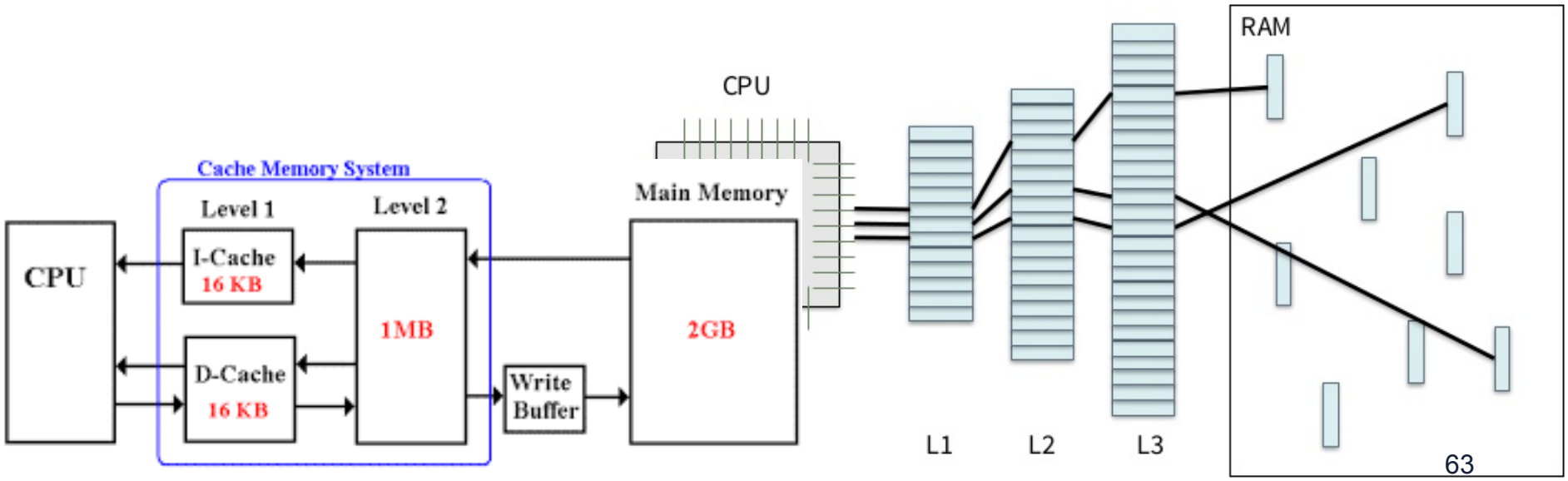
Performance Summary

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

Additional

Multilevel Caches

- Primary cache attached to CPU
  - Small, but fast
- Level-2 cache services misses from primary cache
  - Larger, slower, but still faster than main memory
- Main memory services L-2 cache misses
- Some high-end systems include L-3 cache



Multilevel 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
- Results
  - L-1 cache usually smaller than a single cache
  - L-1 block size smaller than L-2 block size

Multilevel Cache Example

- Given
  - CPU base CPI = 1, clock rate = 4GHz
  - Miss rate/instruction = 2%
  - Main memory access time = 100ns
- With just primary cache
  - Miss penalty = 100ns/0.25ns = 400 cycles
  - Effective CPI = 1 + 0.02 × 400 = 9

Example (cont.)

- Now add L-2 cache
  - Access time = 5ns
  - Global miss rate to main memory = 0.5%
- Primary miss with L-2 hit
  - Penalty = 5ns/0.25ns = 20 cycles
- Primary miss with L-2 miss
  - Extra penalty = 500 cycles
- CPI = 1 + 0.02 × 20 + 0.005 × 400 = 3.4
- Performance ratio = 9/3.4 = 2.6

Memory Hierarchy Performance

- Two indirect performance measures have waylaid many a computer designer.
  - *Instruction count* is independent of the hardware;
  - *Miss rate* is independent of the hardware.
- A better measure of memory hierarchy performance is the *Average Memory Access Time* (AMAT) per instructions

$$\text{AMAT} = \text{Hit time} + \text{Mis srate} \times \text{Miss penalty}$$

  - CPU with 1ns clock, hit time = 1 cycle, miss penalty = 20 cycles, I-cache miss rate = 5%
  - AMAT = 1 + 0.05 × 20 = 2ns
    - 2 cycles per instruction

Example (B-16): Separate vs Unified Cache

- Which has the lower miss rate: a 16 KB instruction cache with a 16KB data or a 32 KB unified cache? Use the miss rates in Figure B.6 to help calculate the correct answer, assuming 36% of the instructions are data transfer instructions. Assume a hit take 1 clock cycle and the miss penalty is 100 clock cycles. A load or store hit takes 1 extra clock cycle on a unified cache if there is only one cache port to satisfy two simultaneous requests. Using the pipelining terminology of Chapter 2, the unified cache leads to a structure hazard. What is the average memory access time in each case? Assume write-through caches with a write buffer and ignore stalls due to the write buffer.
- Answer:

  First let's convert misses per 1000 instructions into miss rates. Solving the general formula from above, the miss rate is

$$\text{Miss rate} = \frac{\dfrac{\text{Misses}}{1000\ \text{Instructions}} / 1000}{\dfrac{\text{Memory accesses}}{\text{Instruction}}}$$

  Since every instruction access has exactly one memory access to fetch the instruction, the instruction miss rate is

$$\text{Miss rate}_{16\,\text{KB instruction}} = \frac{3.82/1000}{1.00} = 0.004$$

Example (B-16)

Since 36% of the instructions are data transfers, the data miss rate is

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

The unified miss rate needs to account for instruction and data access:

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

As stated above, about 74% of the memory accesses are instruction references. Thus, the overall miss rate for the split caches is

$$(74\% \times 0.004) + (26\% \times 0.114) = 0.0326$$

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

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

$$\text{Average memory access time} = \%\,\text{instructions} \times (\text{Hit time} + \text{Instruction miss rate} \times \text{Miss penalty})$$
$$\%\,\text{data} \times (\text{Hit time} + \text{Data miss rate} \times \text{Miss penalty})$$

Therefore, the time for each organization is

$$\text{AMAT}_{\text{split}} = 74\% \times (1 + 0.004 \times 200) + 26\% \times (1 + 0.114 \times 200) = 7.52$$

$$\text{AMAT}_{\text{unified}} = 74\% \times (1 + 0.0.0318 \times 200) + 26\% \times (1 + +1 + 0.0318 \times 200) = 7.62$$

Example (B-18)

- Let's use an in-order execution computer for the first example. Assume the cache miss penalty is 200 clock cycles, and all instructions normally take 1.0 clock cycles (ignoring memory stalls). Assume the average miss rate is 2%, there is an average of 1.5 memory references per instruction, and the average number of cache misses per 1000 instructions is 30. what is the impact on performance when behavior of the cache is included? Calculate the impact using both misses per instruction and miss rate.
- Answer:

  The performance, including cache misses, is

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

  Now calculating performance using miss rate:

$$\begin{aligned} \text{CPU time}_{\text{with cache}} &= \text{IC} \times \left(1.0 + \left(30/1000 \times 200\right)\right) \times \text{Clock cycle time} \\ &= \text{IC} \times 7.00 \times \text{Clock cycle time} \end{aligned}$$

$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\text{CPU time}_{\text{with cache}} = \text{IC} \times \left(1.0 + \left(1.5 \times 2.0\% \times 200\right)\right) \times \text{Clock cycle time} = \text{IC} \times 7.00 \times \text{Clock cycle time}$$

Other Examples (B-19, B-21)

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

$$\text{CPU execution time} = (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle time}$$

$$\text{Memory stall cycles} = \text{Number of misses} \times \text{Miss penalty}$$

$$\text{Memory stall cycles} = \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty}$$

$$\frac{\text{Misses}}{\text{Instruction}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Memory stall clock cycles}}{\text{Instruction}} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\text{CPU execution time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \right) \times \text{Clock cycle time}$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}}{\text{Instruction}} \times (\text{Total miss latency} - \text{Overlapped miss latency})$$

$$\text{Average memory access time} = \text{Hit time}_{\text{L1}} + \text{Miss rate}_{\text{L1}} \times (\text{Hit time}_{\text{L2}} + \text{Miss rate}_{\text{L2}} \times \text{Miss penalty}_{\text{L2}})$$

$$\frac{\text{Memory stall cycles}}{\text{Instruction}} = \frac{\text{Misses}_{\text{L1}}}{\text{Instruction}} \times \text{Hit time}_{\text{L2}} + \frac{\text{Misses}_{\text{L2}}}{\text{Instruction}} \times \text{Miss penalty}_{\text{L2}}$$