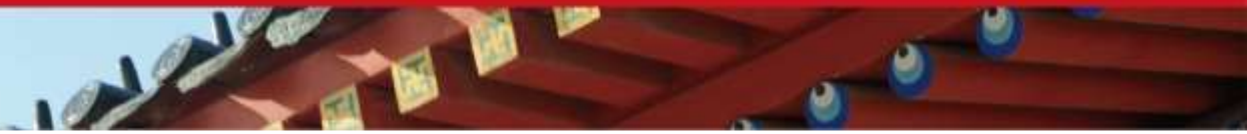
EI 338: Computer Systems Engineering (Operating Systems & Computer Architecture)

Dept. of Computer Science & Engineering
Chentao Wu
wuct@cs.sjtu.edu.cn

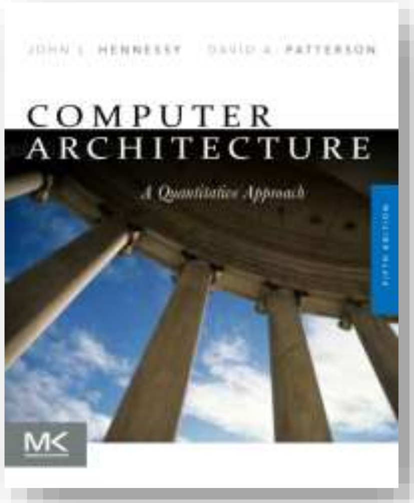
上海交通大學
SHANGHAI JIAO TONG UNIVERSITY



Download lectures

- ftp://public.sjtu.edu.cn
- User: wuct
- Password: wuct123456

- http://www.cs.sjtu.edu.cn/~wuct/cse/



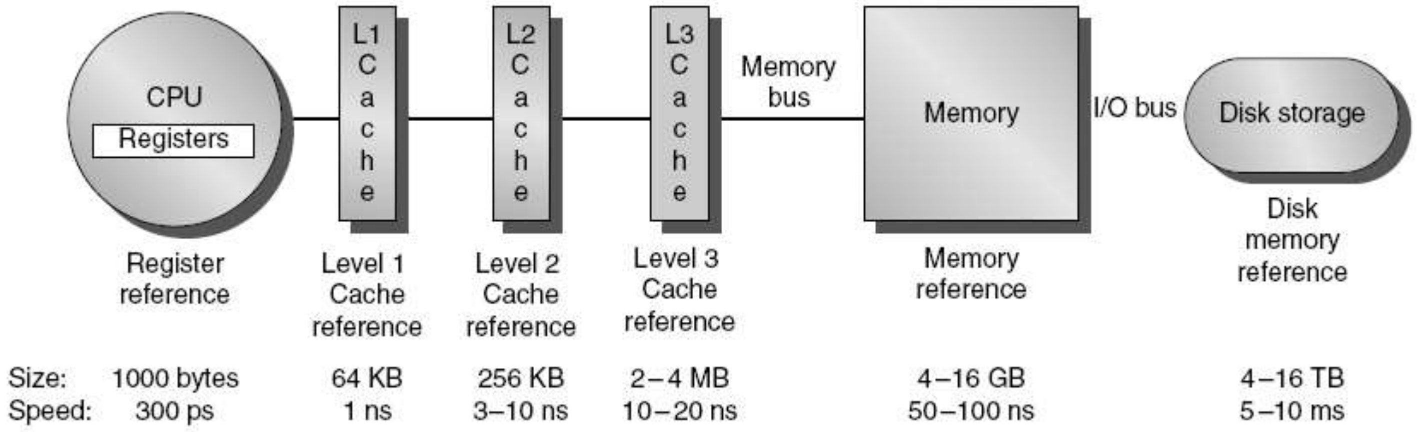
Chapter 2

Memory Hierarchy Design

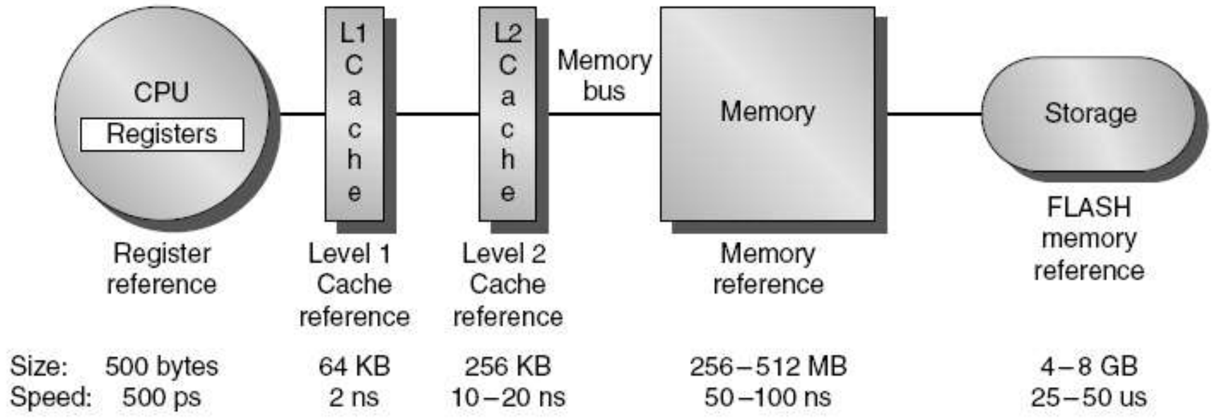
Introduction

- Programmers want unlimited amounts of memory with low latency
- Fast memory technology is more expensive per bit than slower memory
- Solution: organize memory system into a hierarchy
  - Entire addressable memory space available in largest, slowest memory
  - Incrementally smaller and faster memories, each containing a subset of the memory below it, proceed in steps up toward the processor
- Temporal and spatial locality insures that nearly all references can be found in smaller memories
  - Gives the allusion of a large, fast memory being presented to the processor

Memory Hierarchy

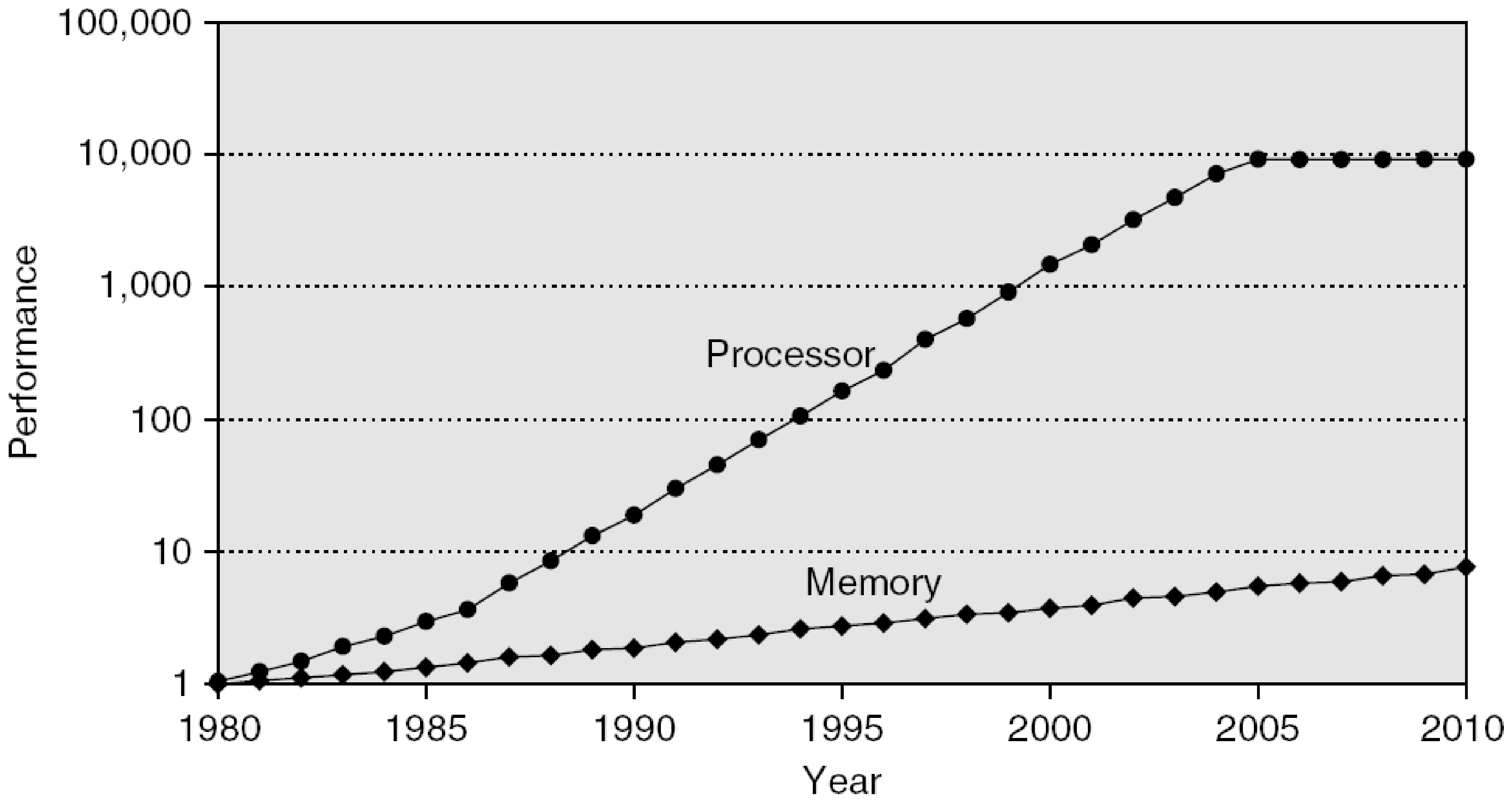


(a) Memory hierarchy for server



(b) Memory hierarchy for a personal mobile device

Memory Performance Gap



Memory Hierarchy Design

- Memory hierarchy design becomes more crucial with recent multi-core processors:
  - Aggregate peak bandwidth grows with # cores:
    - Intel Core i7 can generate two references per core per clock
    - Four cores and 3.2 GHz clock
      - 25.6 billion 64-bit data references/second +
      - 12.8 billion 128-bit instruction references
      - = 409.6 GB/s!
    - DRAM bandwidth is only 6% of this (25 GB/s)
    - Requires:
      - Multi-port, pipelined caches
      - Two levels of cache per core
      - Shared third-level cache on chip

Performance and Power

- High-end microprocessors have >10 MB on-chip cache
  - Consumes large amount of area and power budget

Memory Hierarchy Basics

- When a word is not found in the cache, a *miss* occurs:
  - Fetch word from lower level in hierarchy, requiring a higher latency reference
  - Lower level may be another cache or the main memory
  - Also fetch the other words contained within the *block*
    - Takes advantage of spatial locality
  - Place block into cache in any location within its *set*, determined by address
    - block address MOD number of sets

Memory Hierarchy Basics

- *n* sets => *n-way set associative*
  - *Direct-mapped cache =>* one block per set
  - *Fully associative =>* one set

- Writing to cache: two strategies
  - *Write-through*
    - Immediately update lower levels of hierarchy
  - *Write-back*
    - Only update lower levels of hierarchy when an updated block is replaced
  - Both strategies use *write buffer* to make writes asynchronous

Memory Hierarchy Basics

- Miss rate
  - Fraction of cache access that result in a miss

- Causes of misses
  - Compulsory
    - First reference to a block
  - Capacity
    - Blocks discarded and later retrieved
  - Conflict
    - Program makes repeated references to multiple addresses from different blocks that map to the same location in the cache

Memory Hierarchy Basics

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

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

- **Note that speculative and multithreaded processors may execute other instructions during a miss**
  - **Reduces performance impact of misses**

Equations on Appendix B-4

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

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

$$\begin{aligned}\text{Memory stall clock cycles} = {}& \text{IC} \times \text{Reads per instruction} \times \text{Read miss rate} \times \text{Read miss penalty}\\ &+ \text{IC} \times \text{Writes per instruction} \times \text{Write miss rate} \times \text{Write miss penalty}\end{aligned}$$

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

Example on B-5

Assume we have a computer where the cycles 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 on B-5

First compute the performance for the computer that always hits:

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

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

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

where the middle term (1 + 0.5) represents one instruction access and 0.5 data accesses per instruction. The total performance is thus

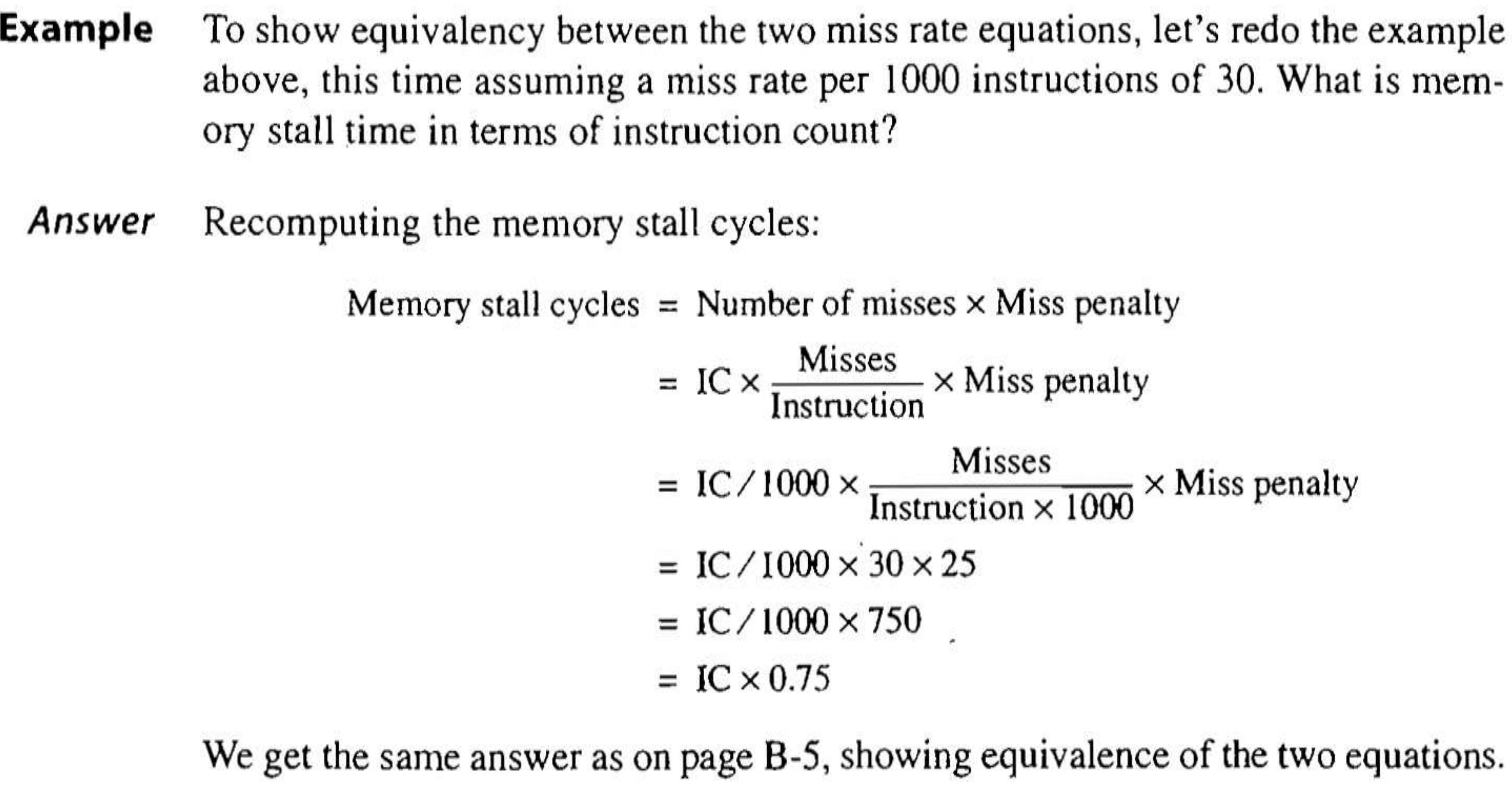
$$\begin{aligned}\text{CPU execution time}_{\text{cache}} &= (\text{IC} \times 1.0 + \text{IC} \times 0.75) \times \text{Clock cycle}\\ &= 1.75 \times \text{IC} \times \text{Clock cycle}\end{aligned}$$

The performance ratio is the inverse of the execution times:

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

The computer with no cache misses is 1.75 times faster.

Example on B-6



Example 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:

$$\begin{aligned}
\text{Memory stall cycles} &= \text{Number of misses} \times \text{Miss penalty} \\
&= \text{IC} \times \frac{\text{Misses}}{\text{Instruction}} \times \text{Miss penalty} \\
&= \text{IC} / 1000 \times \frac{\text{Misses}}{\text{Instruction} \times 1000} \times \text{Miss penalty} \\
&= \text{IC} / 1000 \times 30 \times 25 \\
&= \text{IC} / 1000 \times 750 \\
&= \text{IC} \times 0.75
\end{aligned}$$

We get the same answer as on page B-5, showing equivalence of the two equations.

B-7 Q1: where can a block be placed in a cache?

- If each block has only one place it can appear in the cache, the cache is said to be *direct mapped*. The mapping is usually

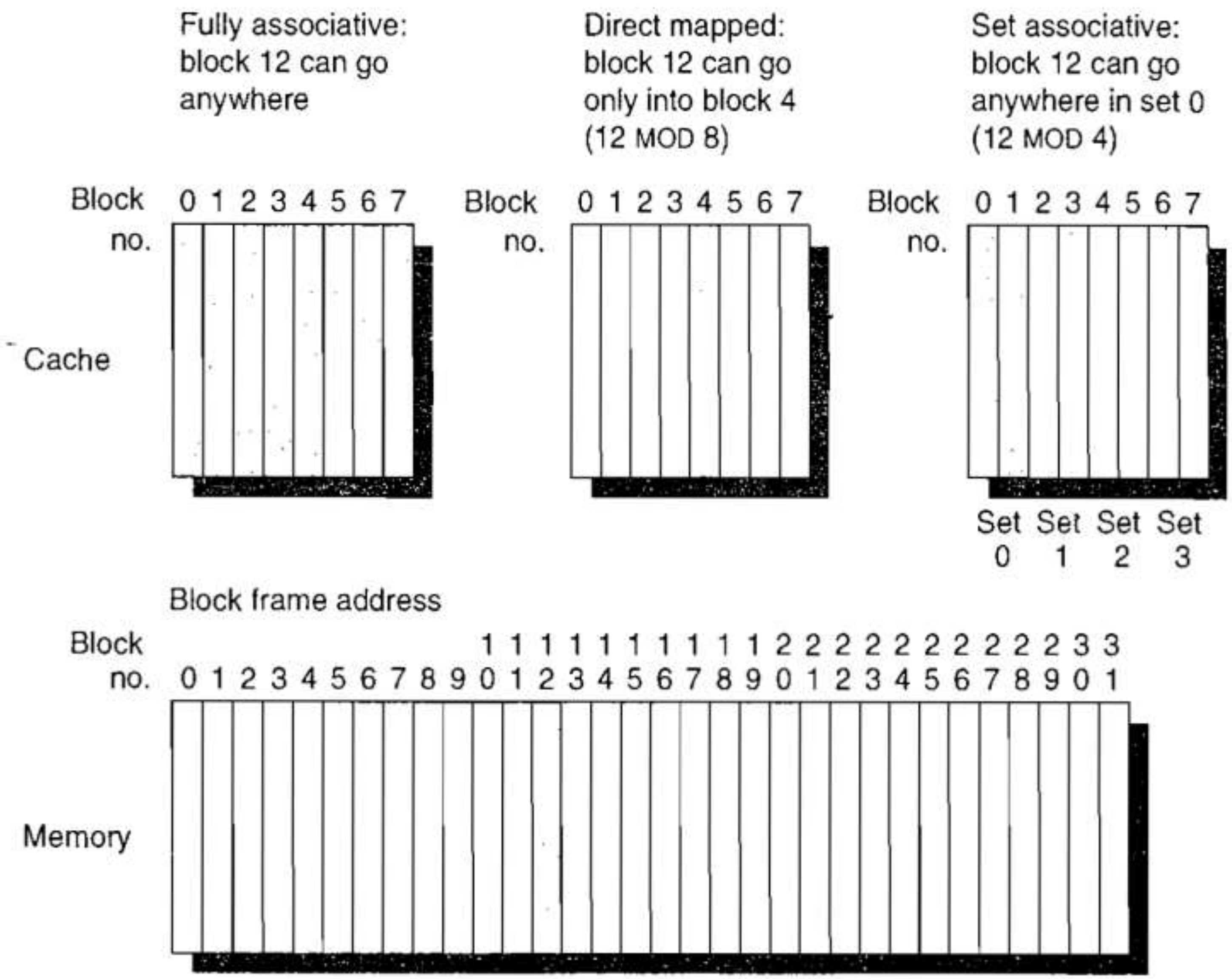
  (*Block address*) MOD (*Number of blocks in cache*)

- If a block can be placed anywhere in the cache, the cache is said to be *fully associative*.
- If a block can be placed in a restricted set of places in the cache, the cache is *set associative*. A *set* is a group of blocks in the cache. A block is first mapped onto a set, and then the block can be placed anywhere within that set. The set is usually chosen by *bit selection*; that is,

  (*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*.

B-7 Q2: how is a block found if it is in the cache?



B-9 Q2: how is a block found if it is in the cache? (contd.)

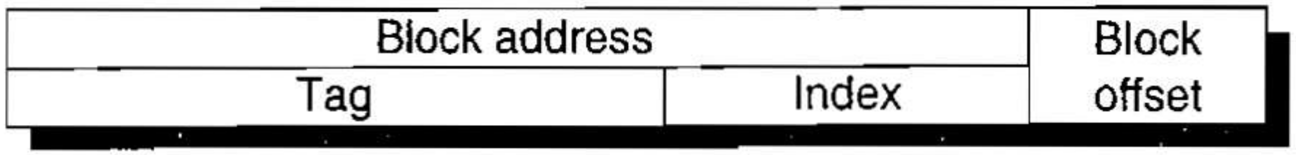
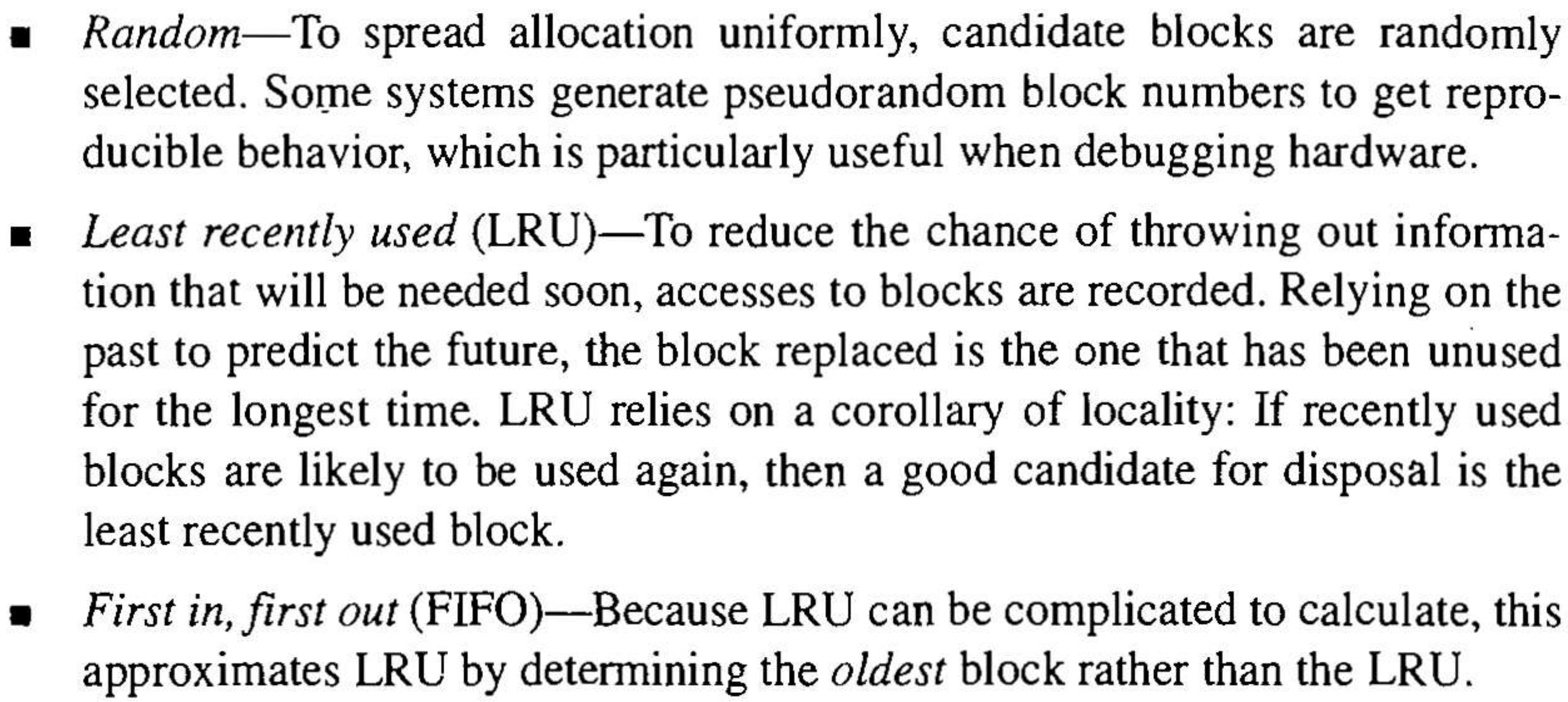


Figure B.3 The three portions of an address in a set associative or direct-mapped cache. The tag is used to check all the blocks in the set, and the index is used to select the set. The block offset is the address of the desired data within the block. Fully associative caches have no index field.

B-9 Q3: which block should be replaced on a cache miss?



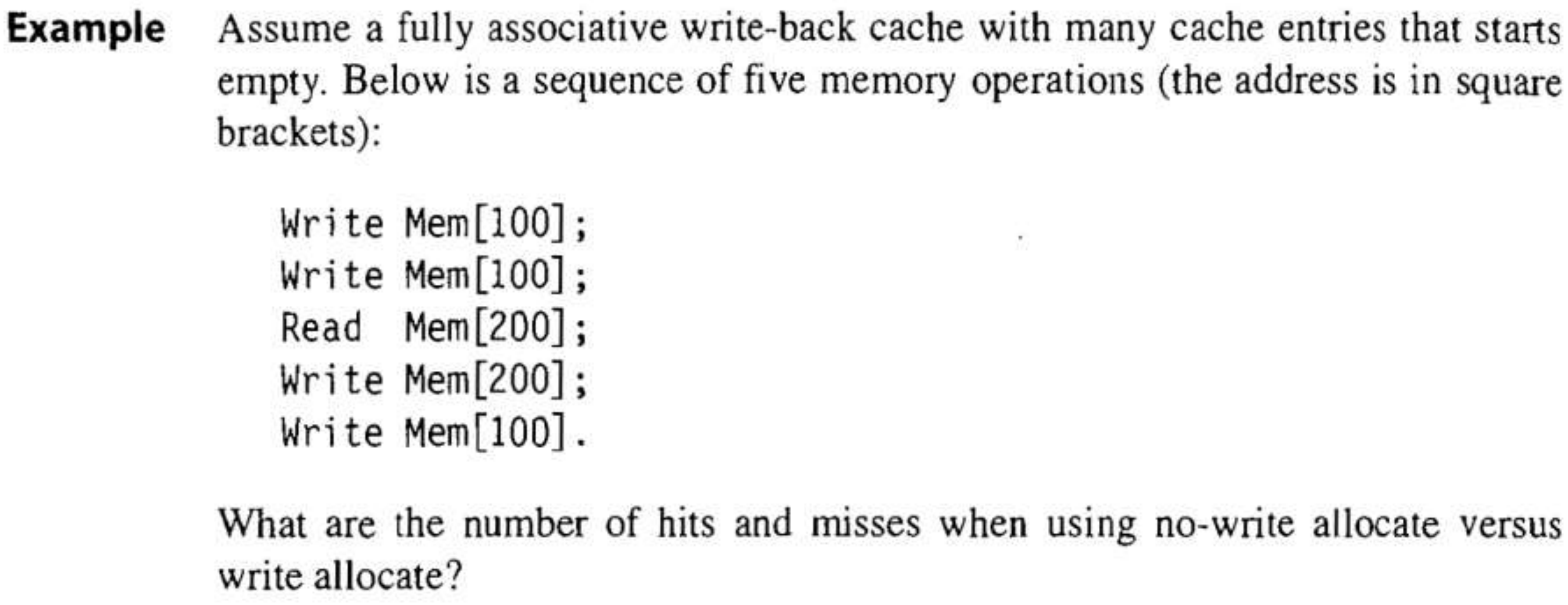
- *Random*—To spread allocation uniformly, candidate blocks are randomly selected. Some systems generate pseudorandom block numbers to get reproducible behavior, which is particularly useful when debugging hardware.
- *Least recently used* (LRU)—To reduce the chance of throwing out information that will be needed soon, accesses to blocks are recorded. Relying on the past to predict the future, the block replaced is the one that has been unused for the longest time. LRU relies on a corollary of locality: If recently used blocks are likely to be used again, then a good candidate for disposal is the least recently used block.
- *First in, first out* (FIFO)—Because LRU can be complicated to calculate, this approximates LRU by determining the *oldest* block rather than the LRU.

B-10 Q4: what happens on a write?

- *Write-through*—The information is written to both the block in the cache *and* to the block in the lower-level memory.
- *Write-back*—The information is written only to the block in the cache. The modified cache block is written to main memory only when it is replaced.

- *Write allocate*—The block is allocated on a write miss, followed by the write hit actions above. In this natural option, write misses act like read misses.
- *No-write allocate*—This apparently unusual alternative is write misses do *not* affect the cache. Instead, the block is modified only in the lower-level memory.

B-10 Q4: what happens on a write? (contd.)



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

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

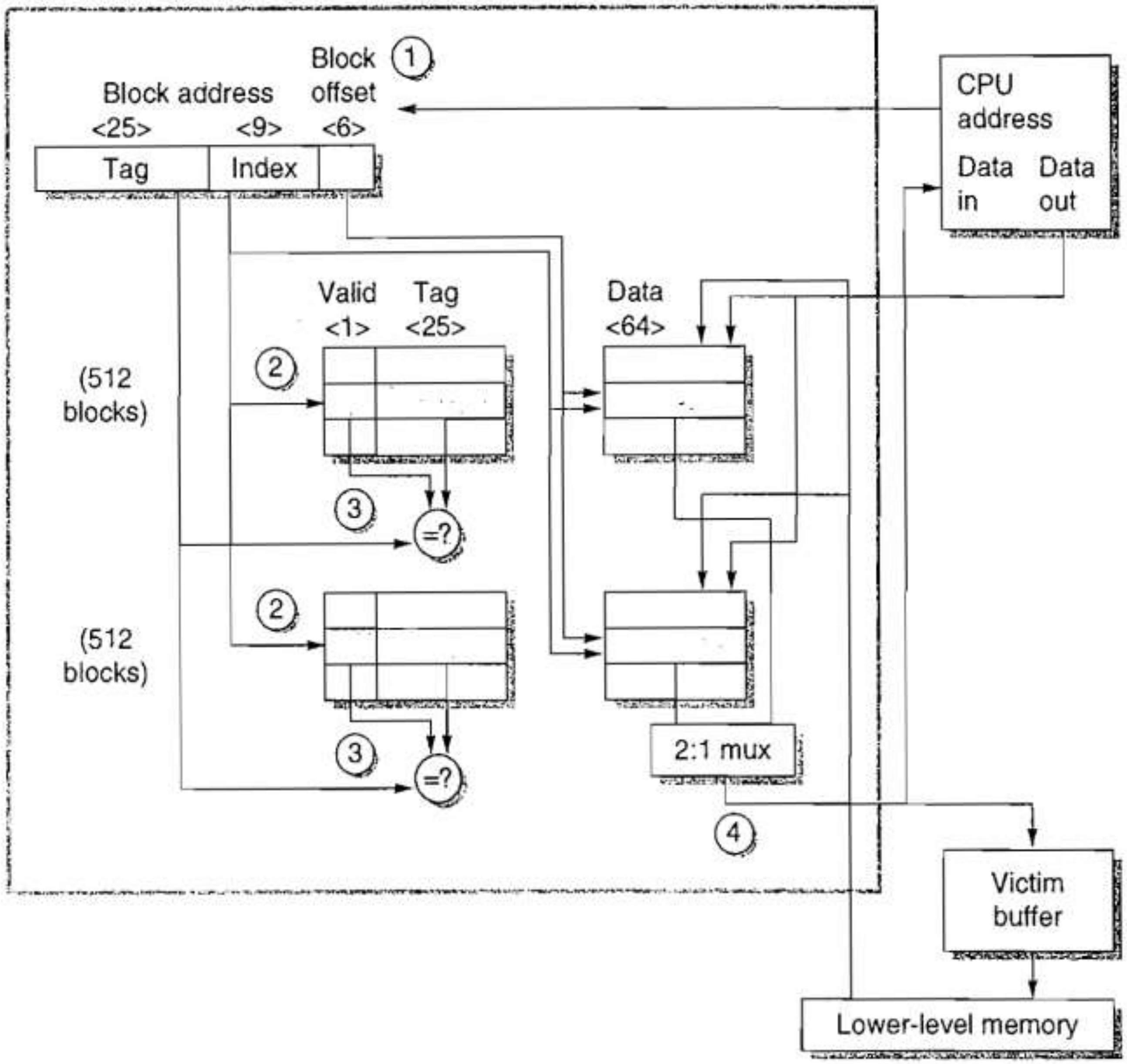
What are the number of hits and misses when using no-write allocate versus write allocate?

B-10 Q4: what happens on a write? (contd.)

Answer For no-write allocate, the address 100 is not in the cache, and there is no allocation on write, so the first two writes will result in misses. Address 200 is also not in the cache, so the read is also a miss. The subsequent write to address 200 is a hit. The last write to 100 is still a miss. The result for no-write allocate is four misses and one hit.

For write allocate, the first accesses to 100 and 200 are misses, and the rest are hits since 100 and 200 are both found in the cache. Thus, the result for write allocate is two misses and three hits.

AMD Opteron Processor



B-16 Cache Performance

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

Example Which has the lower miss rate: a 16 KB instruction cache with a 16 KB data cache 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 takes 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 3, the unified cache leads to a structural 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.

B-16 Cache Performance (contd.)

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

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

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 accesses:

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

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:

$$\begin{aligned}&\text{Average memory access time}\\&\quad = \%\text{ instructions} \times (\text{Hit time} + \text{Instruction miss rate} \times \text{Miss penalty})\\&\qquad + \%\text{ data} \times (\text{Hit time} + \text{Data miss rate} \times \text{Miss penalty})\end{aligned}$$

Therefore, the time for each organization is

$$\begin{aligned}&\text{Average memory access time}_{\text{split}}\\&\quad = 74\% \times (1 + 0.004 \times 200) + 26\% \times (1 + 0.114 \times 200)\\&\quad = (74\% \times 1.80) + (26\% \times 23.80) = 1.332 + 6.188 = 7.52\\&\text{Average memory access time}_{\text{unified}}\\&\quad = 74\% \times (1 + 0.0318 \times 200) + 26\% \times (1 + 1 + 0.0318 \times 200)\\&\quad = (74\% \times 7.36) + (26\% \times 8.36) = 5.446 + 2.174 = 7.62\end{aligned}$$

B-17 Avg. Memory Access Time and Processor Performance

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

Example Let's use an in-order execution computer for the first example. Assume that the cache miss penalty is 200 clock cycles, and all instructions normally take 1.0 clock cycles (ignoring memory stalls). Assume that 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.

B-17 Avg. Memory Access Time and Processor Performance

Answer

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

The performance, including cache misses, is

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

Now calculating performance using miss rate:

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

$$\begin{aligned}\text{CPU time}_{\text{with cache}} &= \text{IC} \times [1.0 + (1.5 \times 2\% \times 200)] \times \text{Clock cycle time} \\ &= \text{IC} \times 7.00 \times \text{Clock cycle time}\end{aligned}$$

The clock cycle time and instruction count are the same, with or without a cache. Thus, CPU time increases sevenfold, with CPI from 1.00 for a "perfect cache" to 7.00 with a cache that can miss. Without any memory hierarchy at all the CPI would increase again to 1.0 + 200 × 1.5 or 301—a factor of more than 40 times longer than a system with a cache!

B-20 Miss Penalty and Out-of-Order Execution Processors

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

- *Length of memory latency*—What to consider as the start and the end of a memory operation in an out-of-order processor
- *Length of latency overlap*—What is the start of overlap with the processor (or, equivalently, when do we say a memory operation is stalling the processor)

B-20 Miss Penalty and Out-of-Order Execution Processors

Example Let's redo the example above, but this time we assume the processor with the longer clock cycle time supports out-of-order execution yet still has a direct-mapped cache. Assume 30% of the 65 ns miss penalty can be overlapped; that is, the average CPU memory stall time is now 45.5 ns.

Answer Average memory access time for the out-of-order (OOO) computer is

$$\text{Average memory access time}_{\text{1-way,OOO}} = 0.35 \times 1.35 + (0.021 \times 45.5) = 1.43 \text{ ns}$$

The performance of the OOO cache is

$$\text{CPU time}_{\text{1-way,OOO}} = \text{IC} \times [1.6 \times 0.35 \times 1.35 + (0.021 \times 1.4 \times 45.5)] = 2.09 \times \text{IC}$$

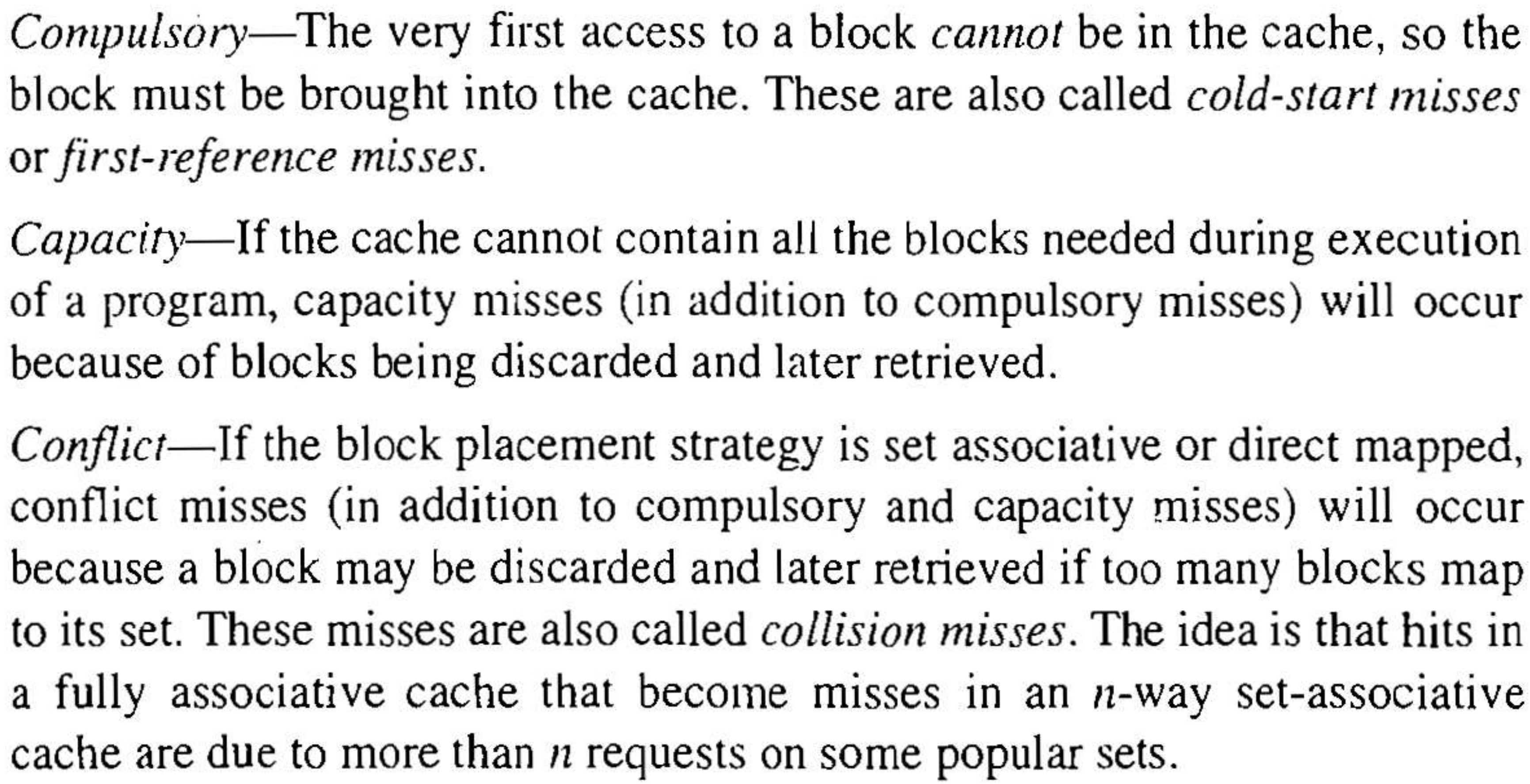
Hence, despite a much slower clock cycle time and the higher miss rate of a direct-mapped cache, the out-of-order computer can be slightly faster if it can hide 30% of the miss penalty.

Six basic cache optimizations

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

- *Reducing the miss rate*—larger block size, larger cache size, and higher associativity
- *Reducing the miss penalty*—multilevel caches and giving reads priority over writes
- *Reducing the time to hit in the cache*—avoiding address translation when indexing the cache

Six basic cache optimizations



- *Compulsory*—The very first access to a block *cannot* be in the cache, so the block must be brought into the cache. These are also called *cold-start misses* or *first-reference misses*.
- *Capacity*—If the cache cannot contain all the blocks needed during execution of a program, capacity misses (in addition to compulsory misses) will occur because of blocks being discarded and later retrieved.
- *Conflict*—If the block placement strategy is set associative or direct mapped, conflict misses (in addition to compulsory and capacity misses) will occur because a block may be discarded and later retrieved if too many blocks map to its set. These misses are also called *collision misses*. The idea is that hits in a fully associative cache that become misses in an *n*-way set-associative cache are due to more than *n* requests on some popular sets.

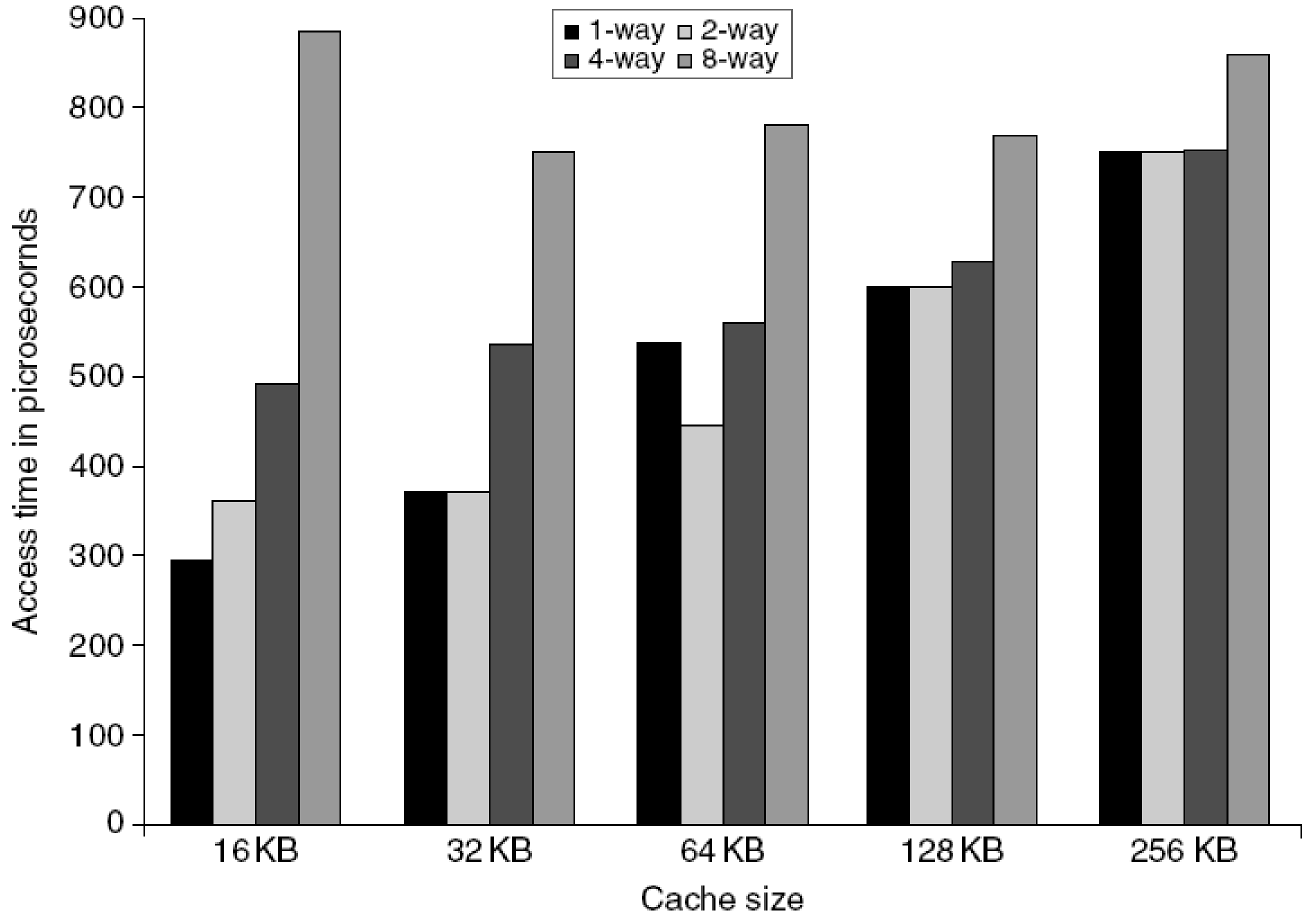
Memory Hierarchy Basics

- **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

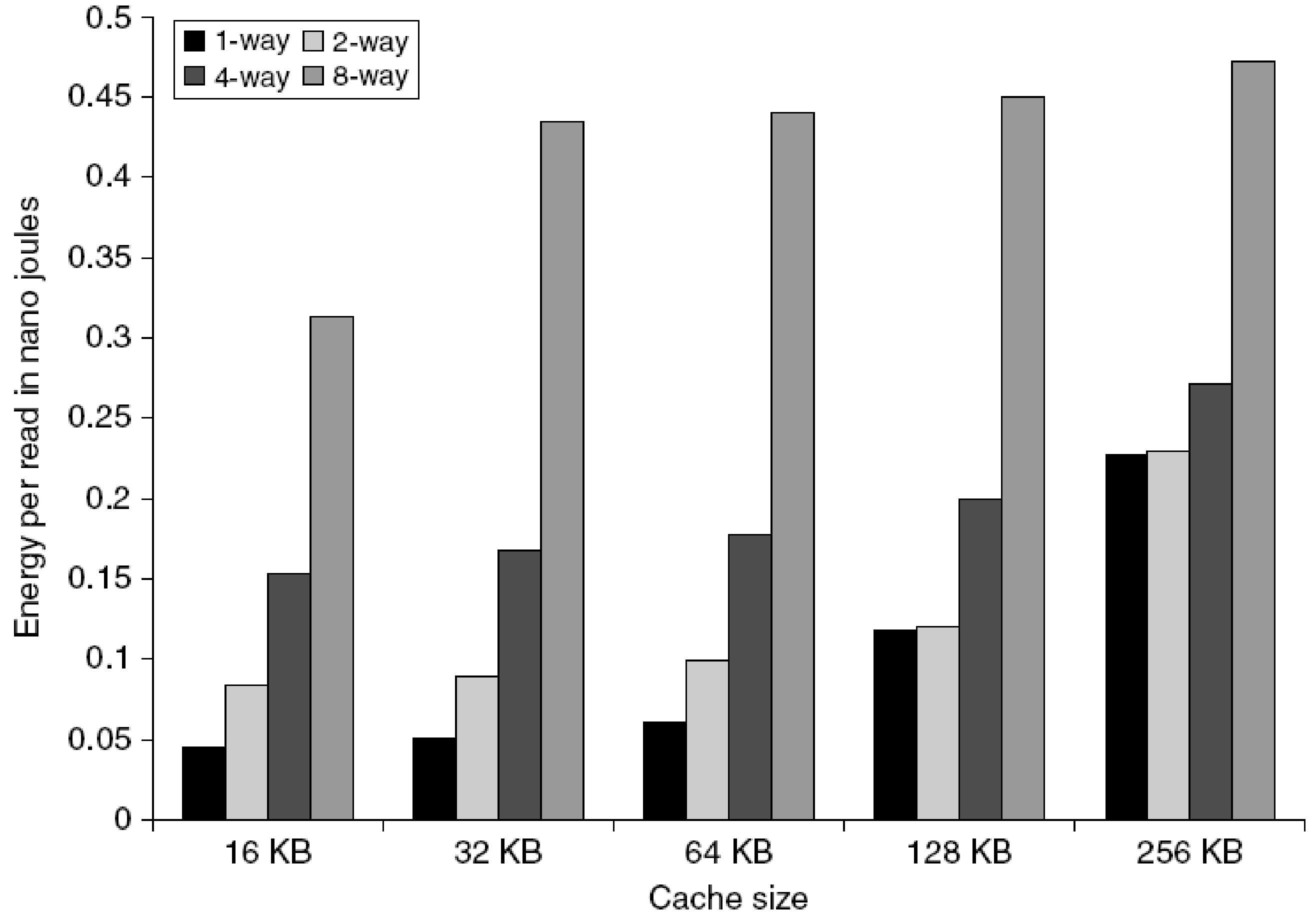
- Small and simple first level caches
  - Critical timing path:
    - addressing tag memory, then
    - comparing tags, then
    - selecting correct set
  - Direct-mapped caches can overlap tag compare and transmission of data
  - Lower associativity reduces power because fewer cache lines are accessed

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

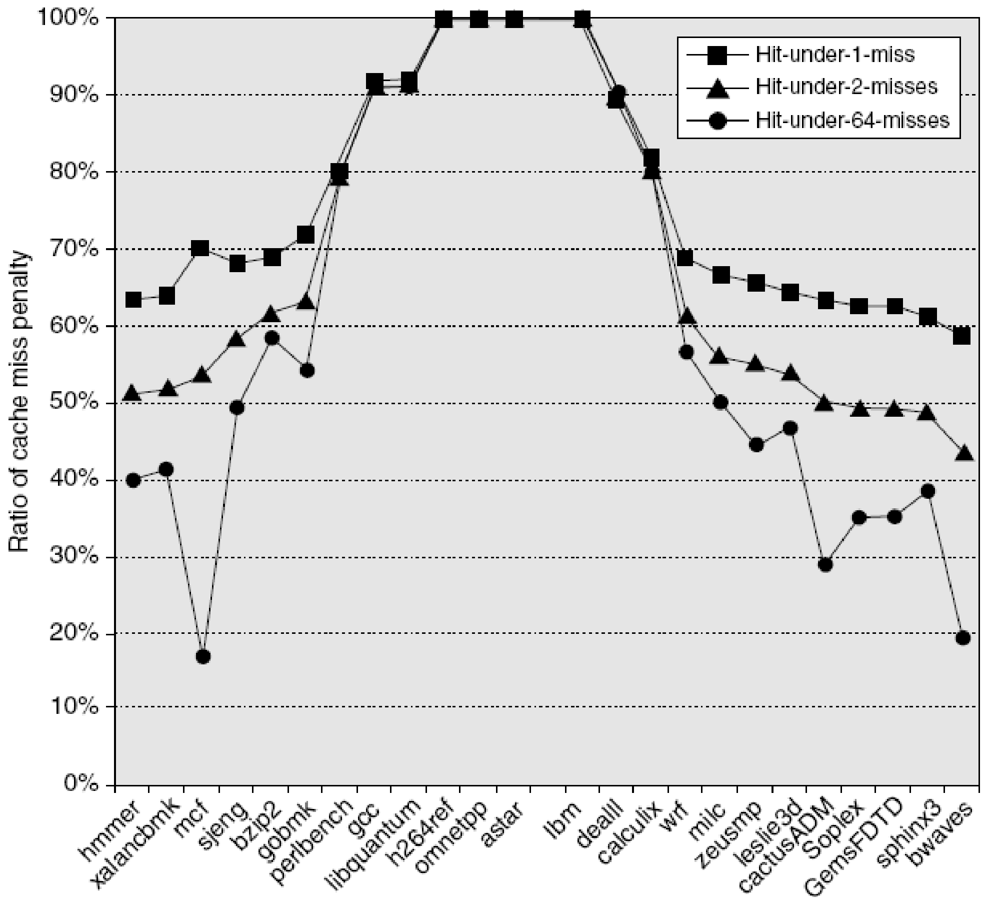
Pipelining Cache

- 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

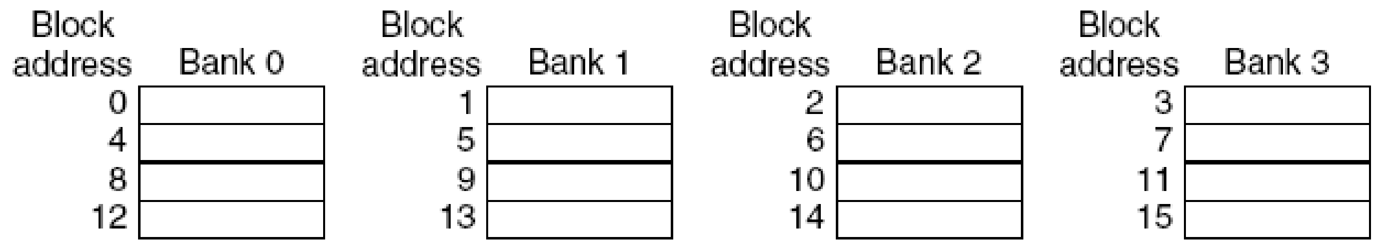


Figure 2.6 Four-way interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

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

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

No write buffering

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

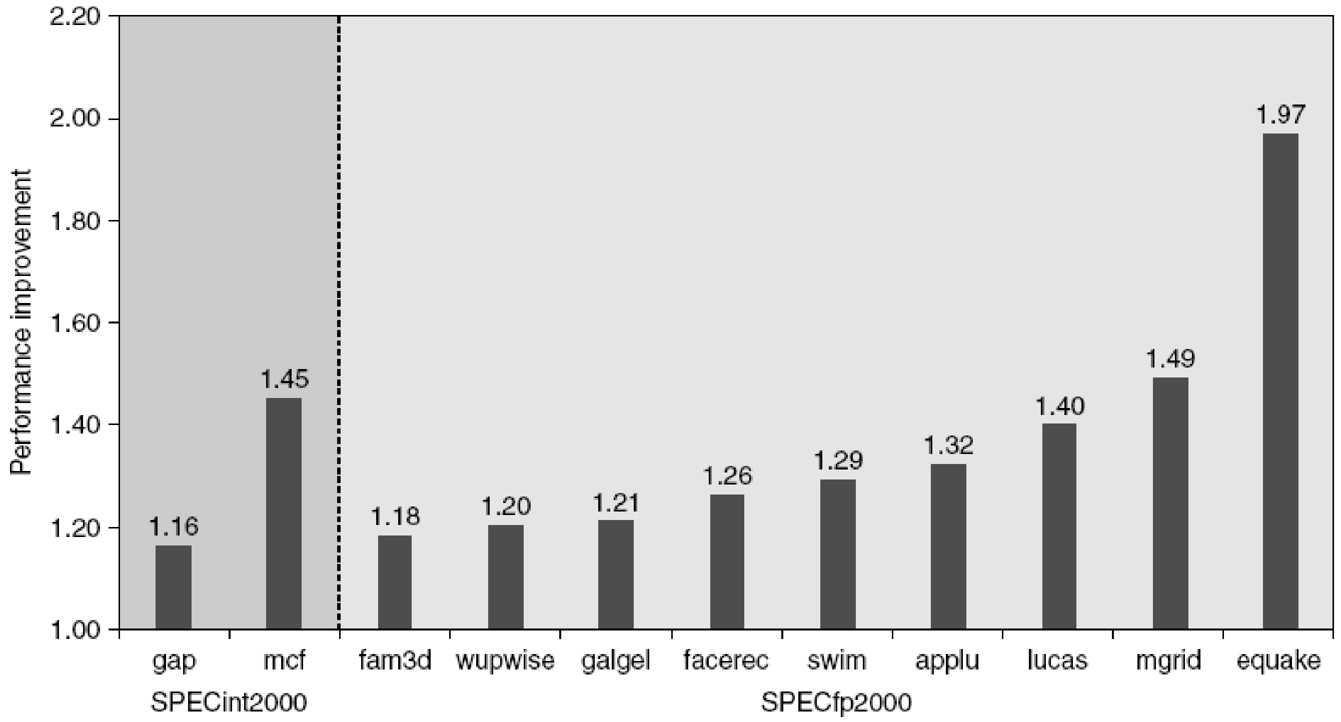
Write buffering

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

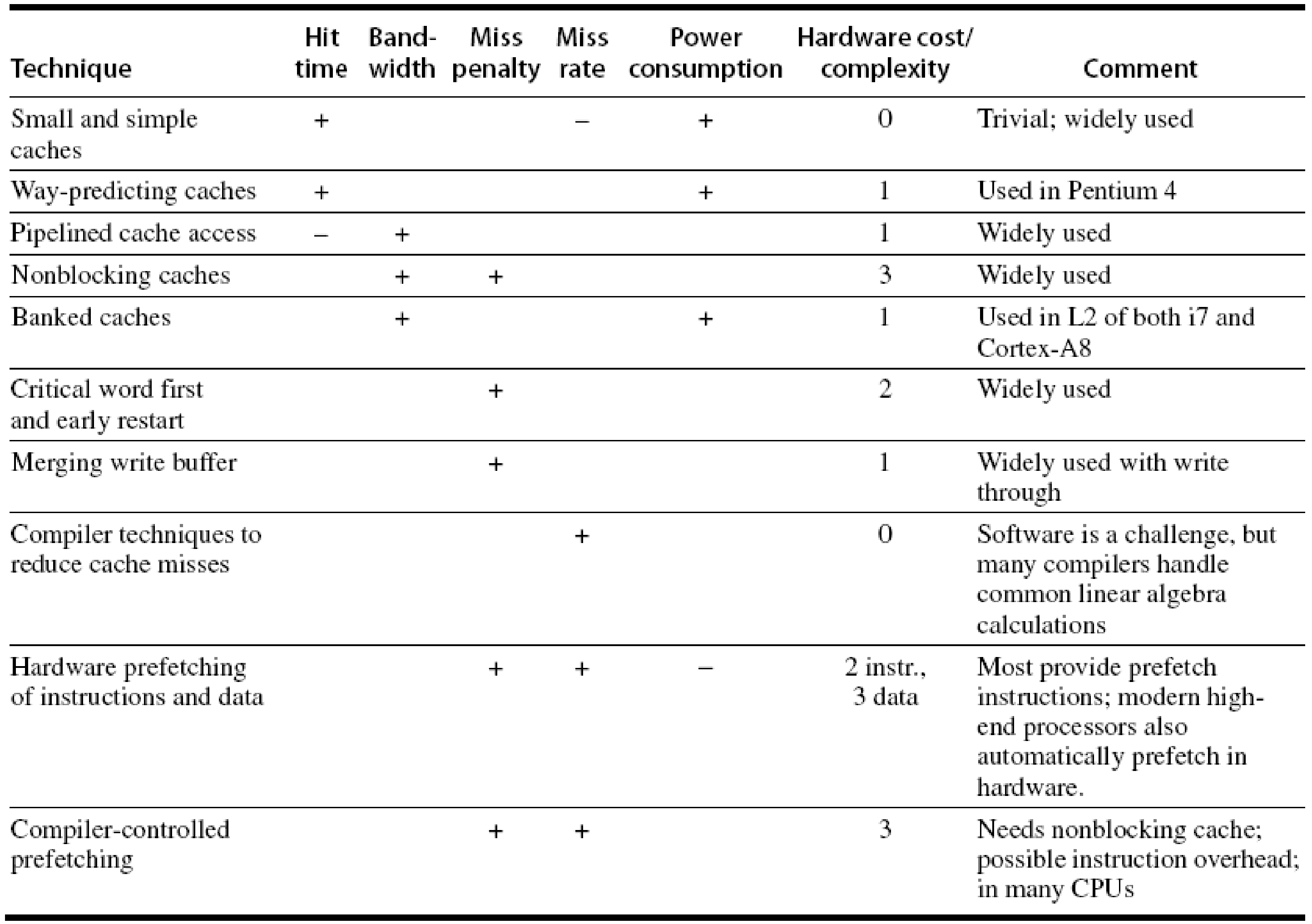
Compiler Prefetching

- 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.

Memory Technology

- Performance metrics
  - Latency is concern of cache
  - Bandwidth is concern of multiprocessors and I/O
  - Access time
    - Time between read request and when desired word arrives
  - Cycle time
    - Minimum time between unrelated requests to memory
- DRAM used for main memory, SRAM used for cache

Memory Technology

- SRAM
  - Requires low power to retain bit
  - Requires 6 transistors/bit

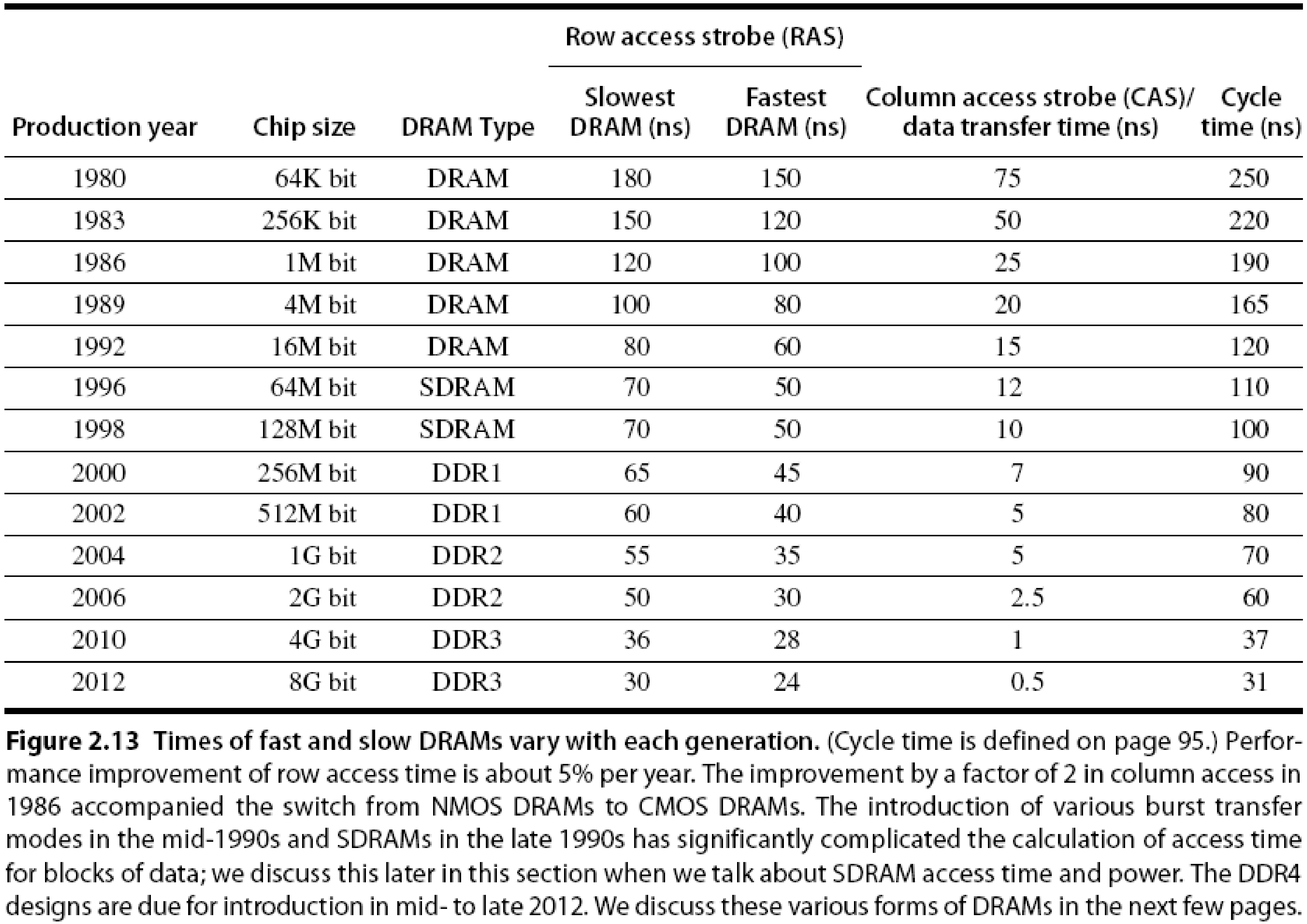
- DRAM
  - Must be re-written after being read
  - Must also be periodically refeshed
    - Every ~ 8 ms
    - Each row can be refreshed simultaneously
  - One transistor/bit
  - Address lines are multiplexed:
    - Upper half of address: row access strobe (RAS)
    - Lower half of address: column access strobe (CAS)

Memory Technology

- Amdahl:
  - Memory capacity should grow linearly with processor speed
  - Unfortunately, memory capacity and speed has not kept pace with processors

- Some optimizations:
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

Memory Optimizations



			Row access strobe (RAS)			
Production year	Chip size	DRAM Type	Slowest DRAM (ns)	Fastest DRAM (ns)	Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Figure 2.13 Times of fast and slow DRAMs vary with each generation. (Cycle time is defined on page 95.) Performance improvement of row access time is about 5% per year. The improvement by a factor of 2 in column access in 1986 accompanied the switch from NMOS DRAMs to CMOS DRAMs. The introduction of various burst transfer modes in the mid-1990s and SDRAMs in the late 1990s has significantly complicated the calculation of access time for blocks of data; we discuss this later in this section when we talk about SDRAM access time and power. The DDR4 designs are due for introduction in mid- to late 2012. We discuss these various forms of DRAMs in the next few pages.

Memory Optimizations

Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600

Figure 2.14 Clock rates, bandwidth, and names of DDR DRAMS and DIMMs in 2010. Note the numerical relationship between the columns. The third column is twice the second, and the fourth uses the number from the third column in the name of the DRAM chip. The fifth column is eight times the third column, and a rounded version of this number is used in the name of the DIMM. Although not shown in this figure, DDRs also specify latency in clock cycles as four numbers, which are specified by the DDR standard. For example, DDR3-2000 CL 9 has latencies of 9-9-9-28. What does this mean? With a 1 ns clock (clock cycle is one-half the transfer rate), this indicate 9 ns for row to columns address (RAS time), 9 ns for column access to data (CAS time), and a minimum read time of 28 ns. Closing the row takes 9 ns for precharge but happens only when the reads from that row are finished. In burst mode, transfers occur on every clock on both edges, when the first RAS and CAS times have elapsed. Furthermore, the precharge in not needed until the entire row is read. DDR4 will be produced in 2012 and is expected to reach clock rates of 1600 MHz in 2014, when DDR5 is expected to take over. The exercises explore these details further.

Memory Optimizations

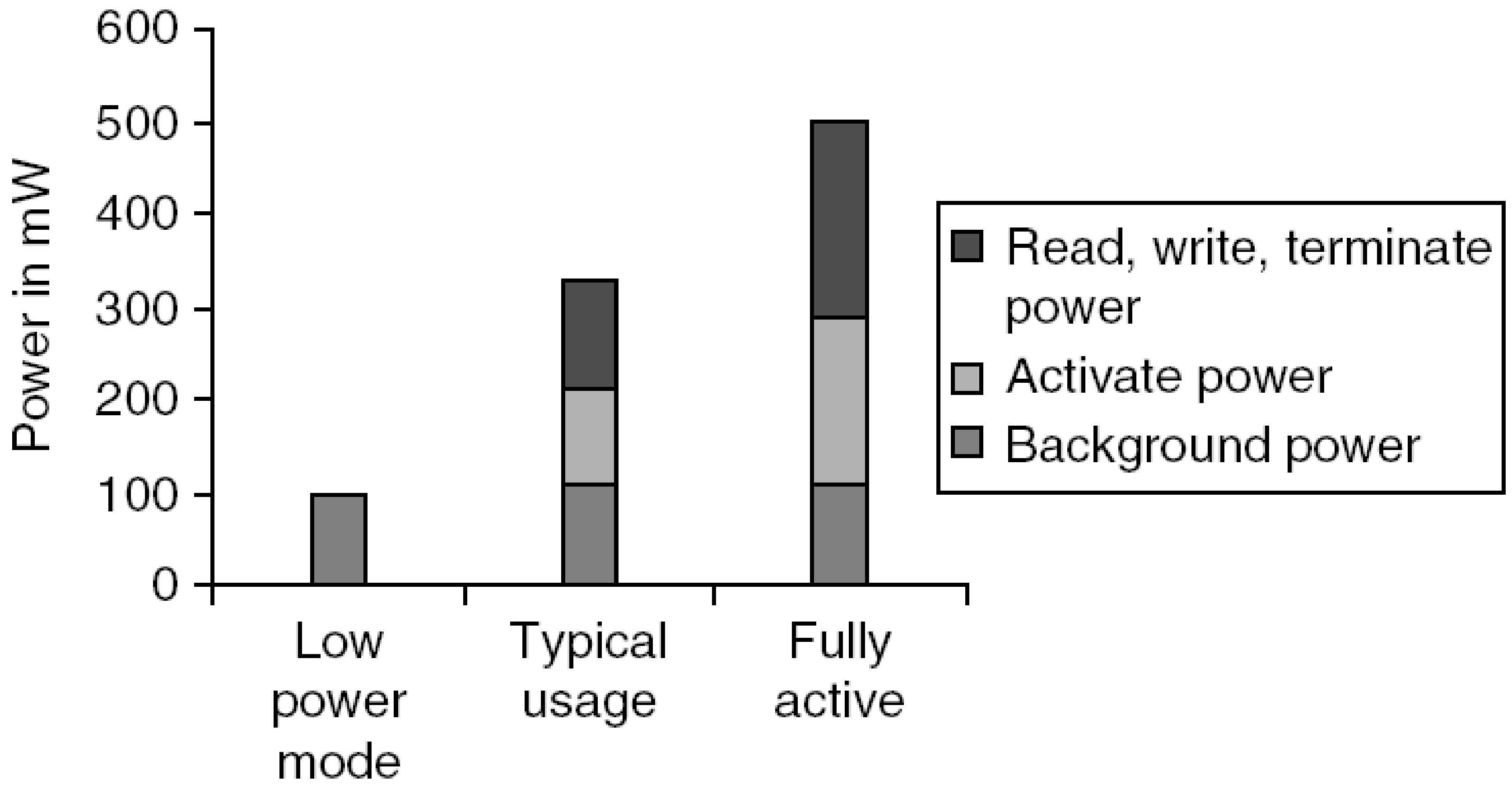
- DDR:
  - DDR2
    - Lower power (2.5 V -> 1.8 V)
    - Higher clock rates (266 MHz, 333 MHz, 400 MHz)
  - DDR3
    - 1.5 V
    - 800 MHz
  - DDR4
    - 1-1.2 V
    - 1600 MHz

- GDDR5 is graphics memory based on DDR3

Memory Optimizations

- Graphics memory:
  - Achieve 2-5 X bandwidth per DRAM vs. DDR3
    - Wider interfaces (32 vs. 16 bit)
    - Higher clock rate
      - Possible because they are attached via soldering instead of socketted DIMM modules
- Reducing power in SDRAMs:
  - Lower voltage
  - Low power mode (ignores clock, continues to refresh)

Memory Power Consumption



Flash Memory

- Type of EEPROM
- Must be erased (in blocks) before being overwritten
- Non volatile
- Limited number of write cycles
- Cheaper than SDRAM, more expensive than disk
- Slower than SRAM, faster than disk

Memory Dependability

- Memory is susceptible to cosmic rays
- *Soft errors*: dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- *Hard errors*: permanent errors
  - Use sparse rows to replace defective rows

- Chipkill: a RAID-like error recovery technique

Virtual Memory

- Protection via virtual memory
  - Keeps processes in their own memory space

- Role of architecture:
  - Provide user mode and supervisor mode
  - Protect certain aspects of CPU state
  - Provide mechanisms for switching between user mode and supervisor mode
  - Provide mechanisms to limit memory accesses
  - Provide TLB to translate addresses

Virtual Machines

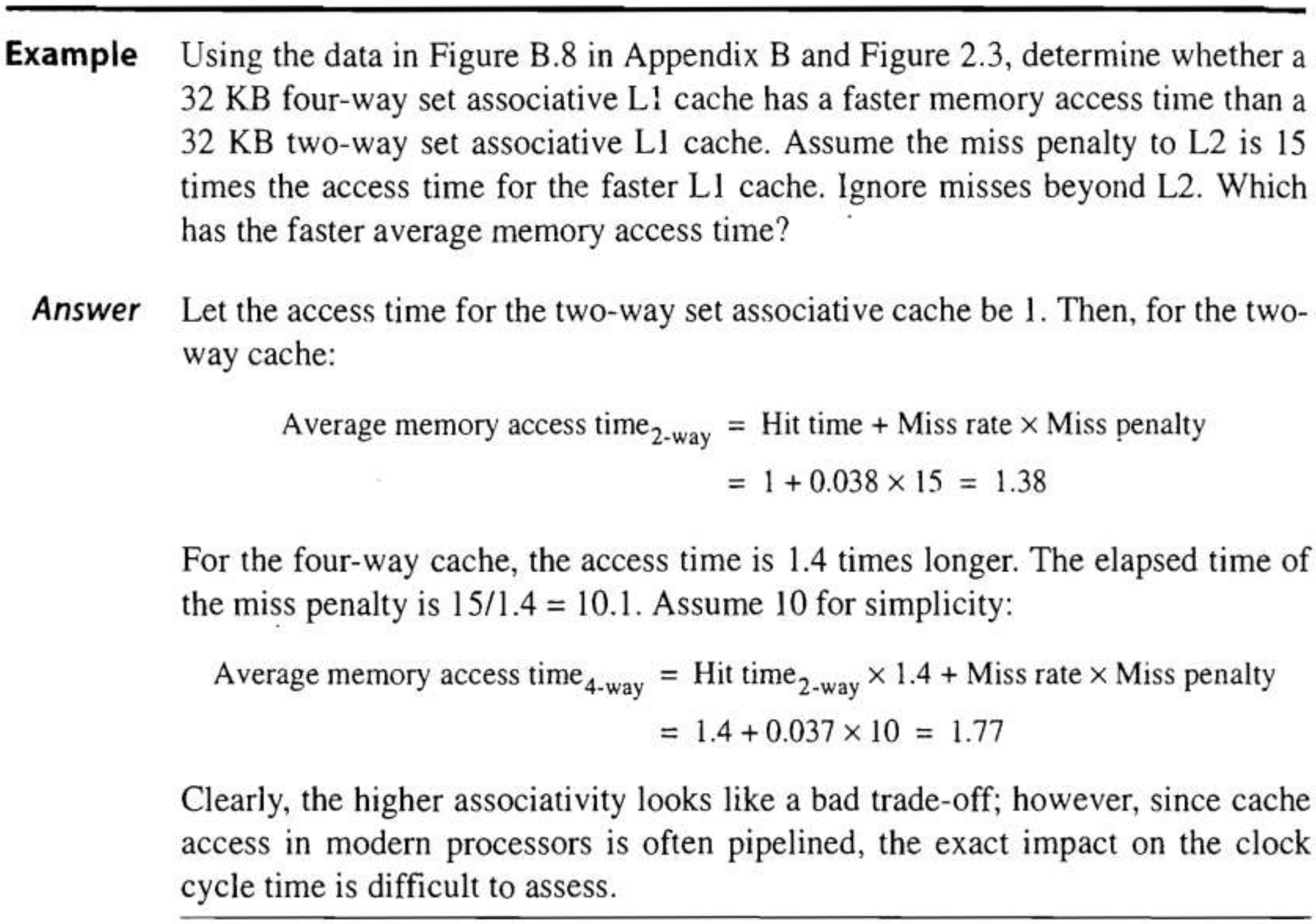
- Supports isolation and security
- Sharing a computer among many unrelated users
- Enabled by raw speed of processors, making the overhead more acceptable

- Allows different ISAs and operating systems to be presented to user programs
  - "System Virtual Machines"
  - SVM software is called "virtual machine monitor" or "hypervisor"
  - Individual virtual machines run under the monitor are called "guest VMs"

Impact of VMs on Virtual Memory

- Each guest OS maintains its own set of page tables
  - VMM adds a level of memory between physical and virtual memory called "real memory"
  - VMM maintains shadow page table that maps guest virtual addresses to physical addresses
    - Requires VMM to detect guest's changes to its own page table
    - Occurs naturally if accessing the page table pointer is a privileged operation

Example on Page 80



Example Using the data in Figure B.8 in Appendix B and Figure 2.3, determine whether a 32 KB four-way set associative L1 cache has a faster memory access time than a 32 KB two-way set associative L1 cache. Assume the miss penalty to L2 is 15 times the access time for the faster L1 cache. Ignore misses beyond L2. Which has the faster average memory access time?

Answer Let the access time for the two-way set associative cache be 1. Then, for the two-way cache:

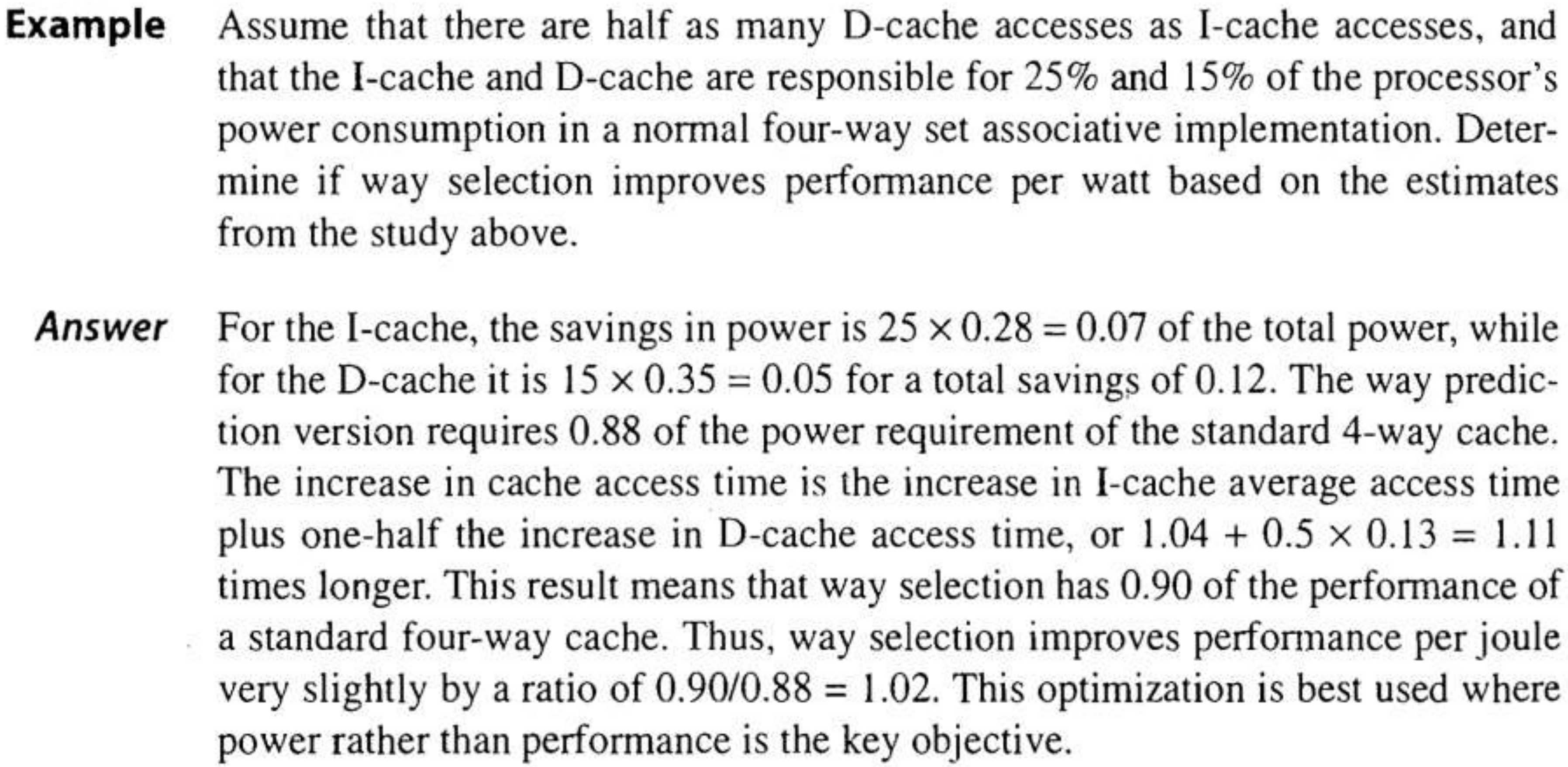
$$\begin{aligned}\text{Average memory access time}_{\text{2-way}} &= \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}\\ &= 1 + 0.038 \times 15 = 1.38\end{aligned}$$

For the four-way cache, the access time is 1.4 times longer. The elapsed time of the miss penalty is 15/1.4 = 10.1. Assume 10 for simplicity:

$$\begin{aligned}\text{Average memory access time}_{\text{4-way}} &= \text{Hit time}_{\text{2-way}} \times 1.4 + \text{Miss rate} \times \text{Miss penalty}\\ &= 1.4 + 0.037 \times 10 = 1.77\end{aligned}$$

Clearly, the higher associativity looks like a bad trade-off; however, since cache access in modern processors is often pipelined, the exact impact on the clock cycle time is difficult to assess.

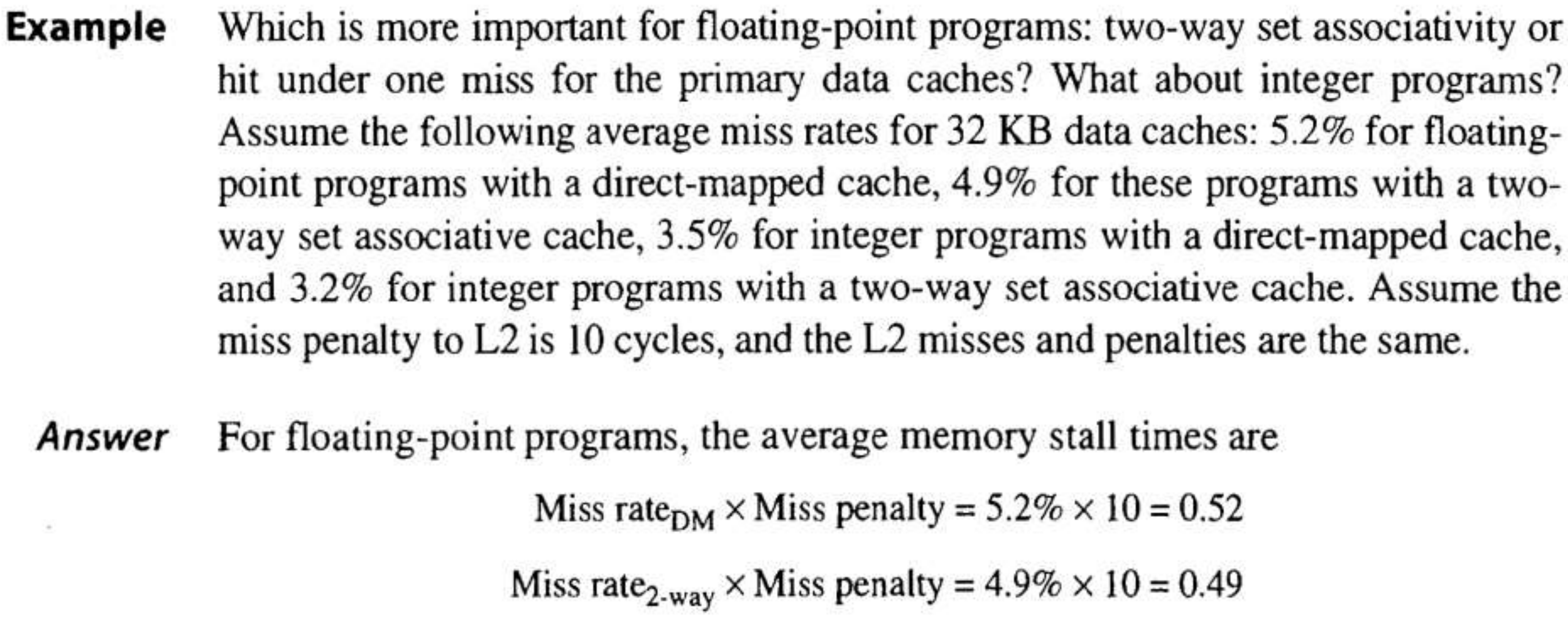
Example on Page 82



Example Assume that there are half as many D-cache accesses as I-cache accesses, and that the I-cache and D-cache are responsible for 25% and 15% of the processor's power consumption in a normal four-way set associative implementation. Determine if way selection improves performance per watt based on the estimates from the study above.

Answer For the I-cache, the savings in power is $25 \times 0.28 = 0.07$ of the total power, while for the D-cache it is $15 \times 0.35 = 0.05$ for a total savings of 0.12. The way prediction version requires 0.88 of the power requirement of the standard 4-way cache. The increase in cache access time is the increase in I-cache average access time plus one-half the increase in D-cache access time, or $1.04 + 0.5 \times 0.13 = 1.11$ times longer. This result means that way selection has 0.90 of the performance of a standard four-way cache. Thus, way selection improves performance per joule very slightly by a ratio of $0.90/0.88 = 1.02$. This optimization is best used where power rather than performance is the key objective.

Example on Page 83



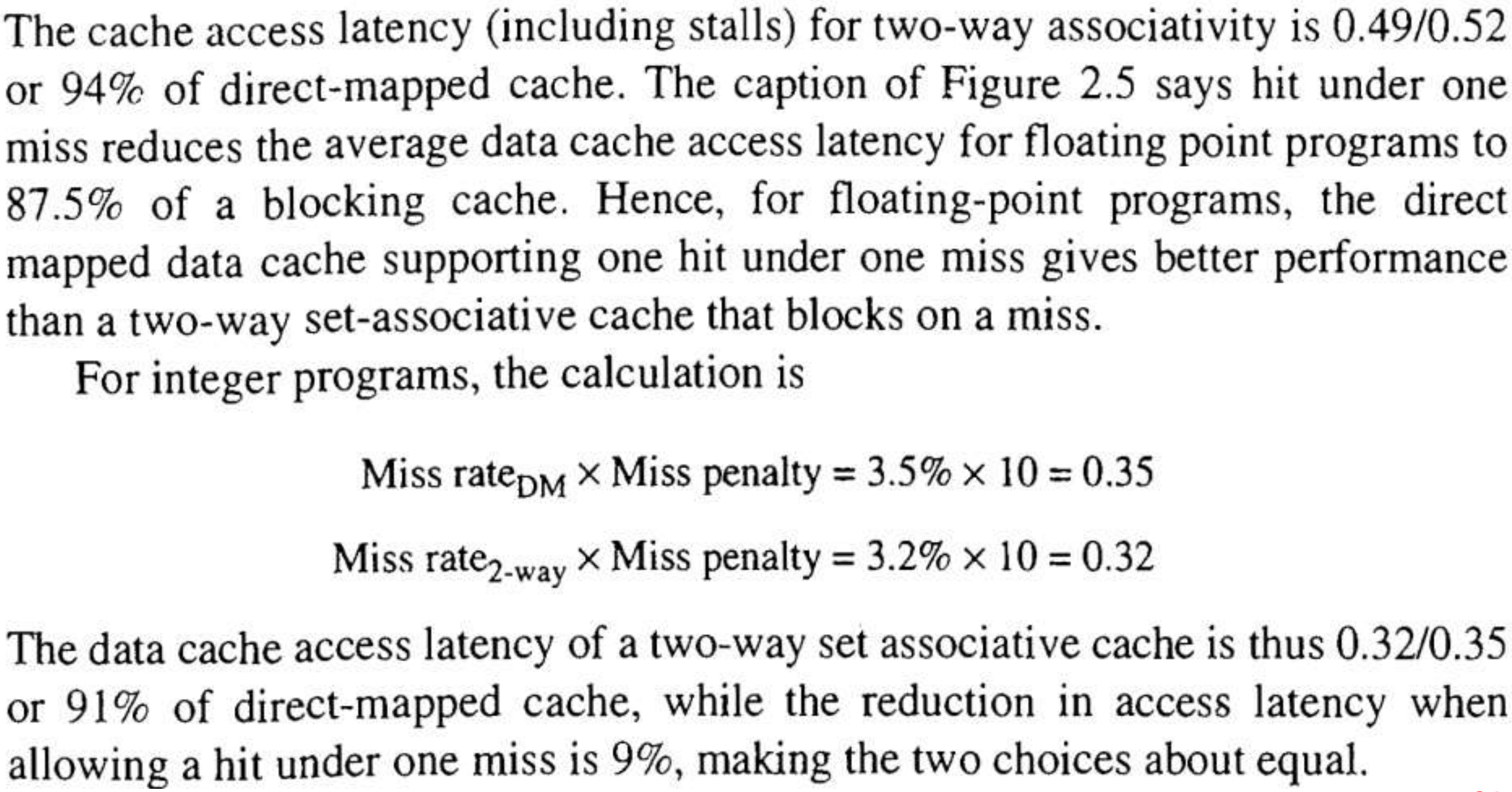
Example Which is more important for floating-point programs: two-way set associativity or hit under one miss for the primary data caches? What about integer programs? Assume the following average miss rates for 32 KB data caches: 5.2% for floating-point programs with a direct-mapped cache, 4.9% for these programs with a two-way set associative cache, 3.5% for integer programs with a direct-mapped cache, and 3.2% for integer programs with a two-way set associative cache. Assume the miss penalty to L2 is 10 cycles, and the L2 misses and penalties are the same.

Answer For floating-point programs, the average memory stall times are

$$\text{Miss rate}_{\text{DM}} \times \text{Miss penalty} = 5.2\% \times 10 = 0.52$$

$$\text{Miss rate}_{\text{2-way}} \times \text{Miss penalty} = 4.9\% \times 10 = 0.49$$

Example on Page 83 (contd.)



The cache access latency (including stalls) for two-way associativity is 0.49/0.52 or 94% of direct-mapped cache. The caption of Figure 2.5 says hit under one miss reduces the average data cache access latency for floating point programs to 87.5% of a blocking cache. Hence, for floating-point programs, the direct mapped data cache supporting one hit under one miss gives better performance than a two-way set-associative cache that blocks on a miss.

For integer programs, the calculation is

$$\text{Miss rate}_{\text{DM}} \times \text{Miss penalty} = 3.5\% \times 10 = 0.35$$

$$\text{Miss rate}_{\text{2-way}} \times \text{Miss penalty} = 3.2\% \times 10 = 0.32$$

The data cache access latency of a two-way set associative cache is thus 0.32/0.35 or 91% of direct-mapped cache, while the reduction in access latency when allowing a hit under one miss is 9%, making the two choices about equal.

Homework

- 2.8, B.1