MO401

IC/Unicamp

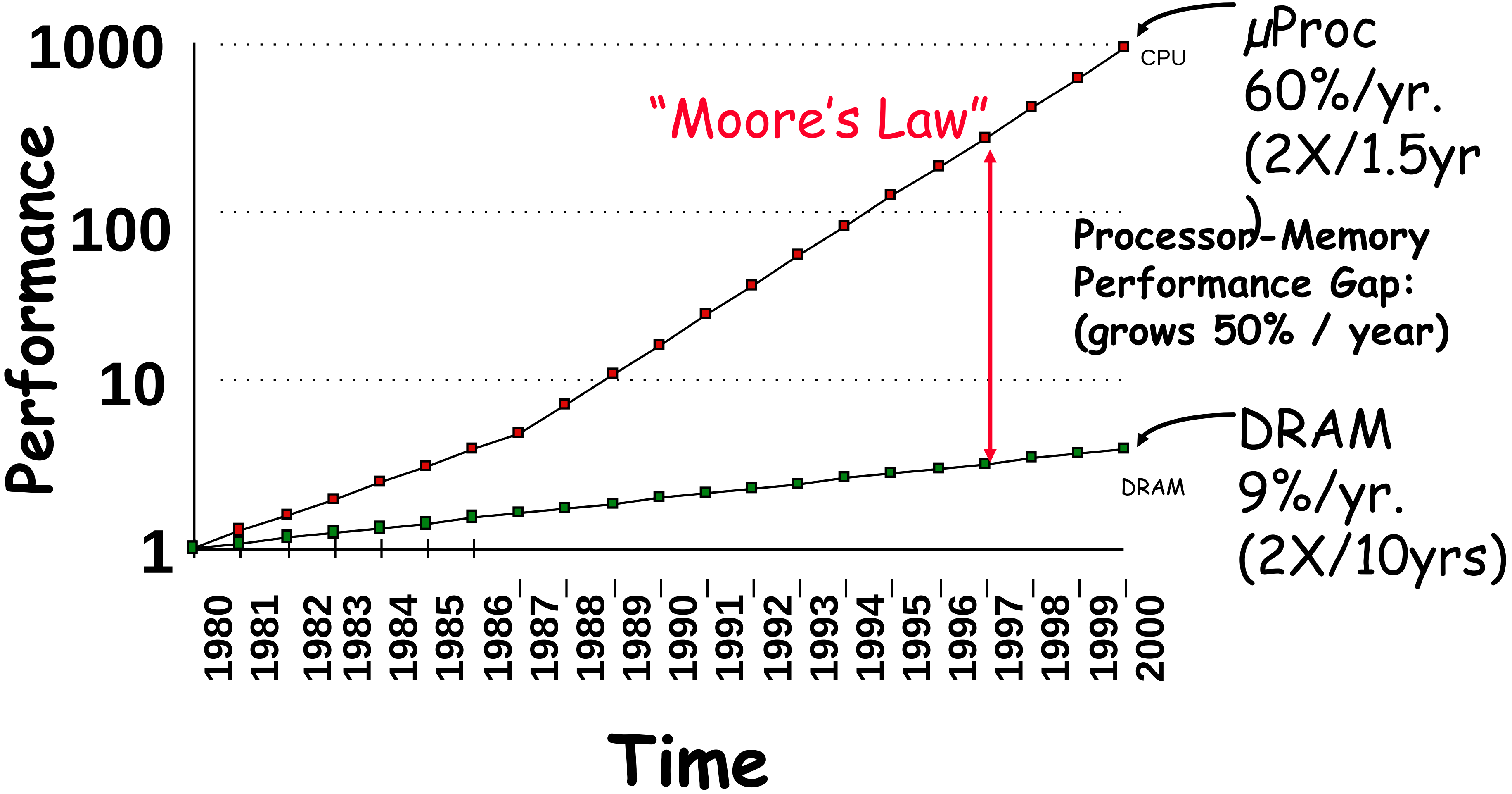
Prof Mario Côrtes

Appendix B: Review Memory Hierarchy

Tópicos

- Revisão básica (Paulo Centoducatte)
- B.1: Introduction
- B.2: cache performance
- B.3: Six basic cache optimization
- B.4: Virtual memory

Processor-DRAM Memory Gap (latency



Níveis em uma Hierarquia de Memórias

Capacidade
Access Time
Custo

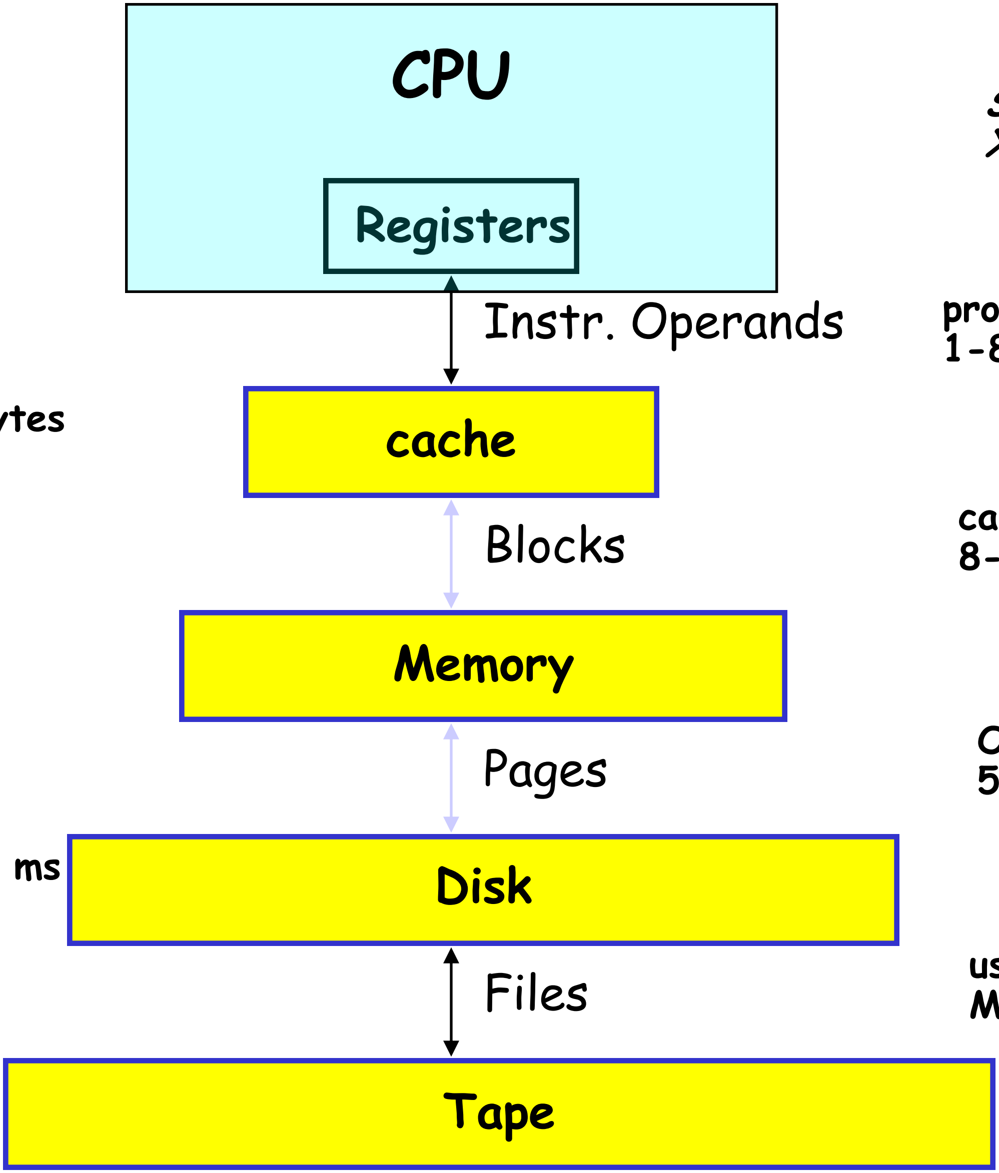
CPU Registers
100s Bytes
<1s ns

cache
10s-100s K Bytes
1-10 ns
$10/ MByte

Main Memory
M Bytes
50ns- 70ns
$1/ MByte

Disk
10s G Bytes, 10 ms
(10,000,000 ns)
$0.002/ MByte

Tape
“infinito”
sec-min
$0.0014/ MByte



CPU

Registers

Instr. Operands

cache

Blocks

Memory

Pages

Disk

Files

Tape

Upper Level

Staging
Xfer Unit

faster

prog./compiler
1-8 bytes

cache cntl
8-128 bytes

OS
512-4K bytes

user/operator
Mbytes

Larger

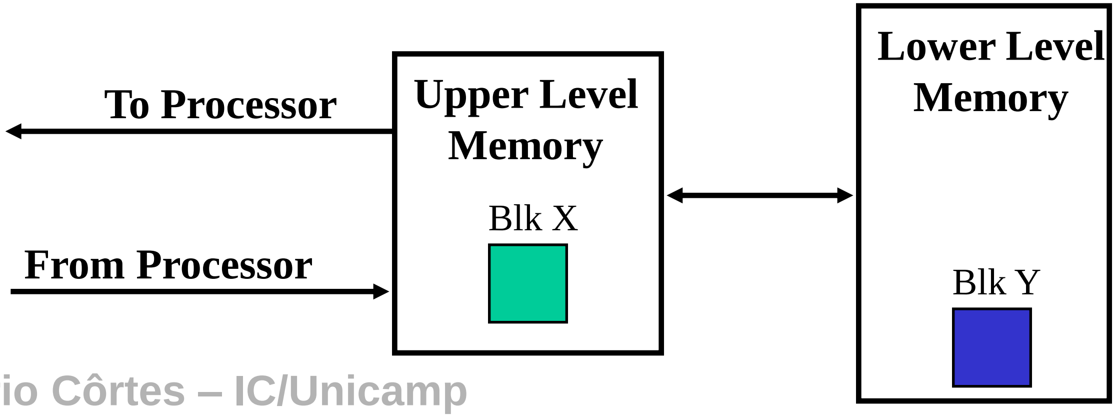
Lower Level

Princípio da Localidadde

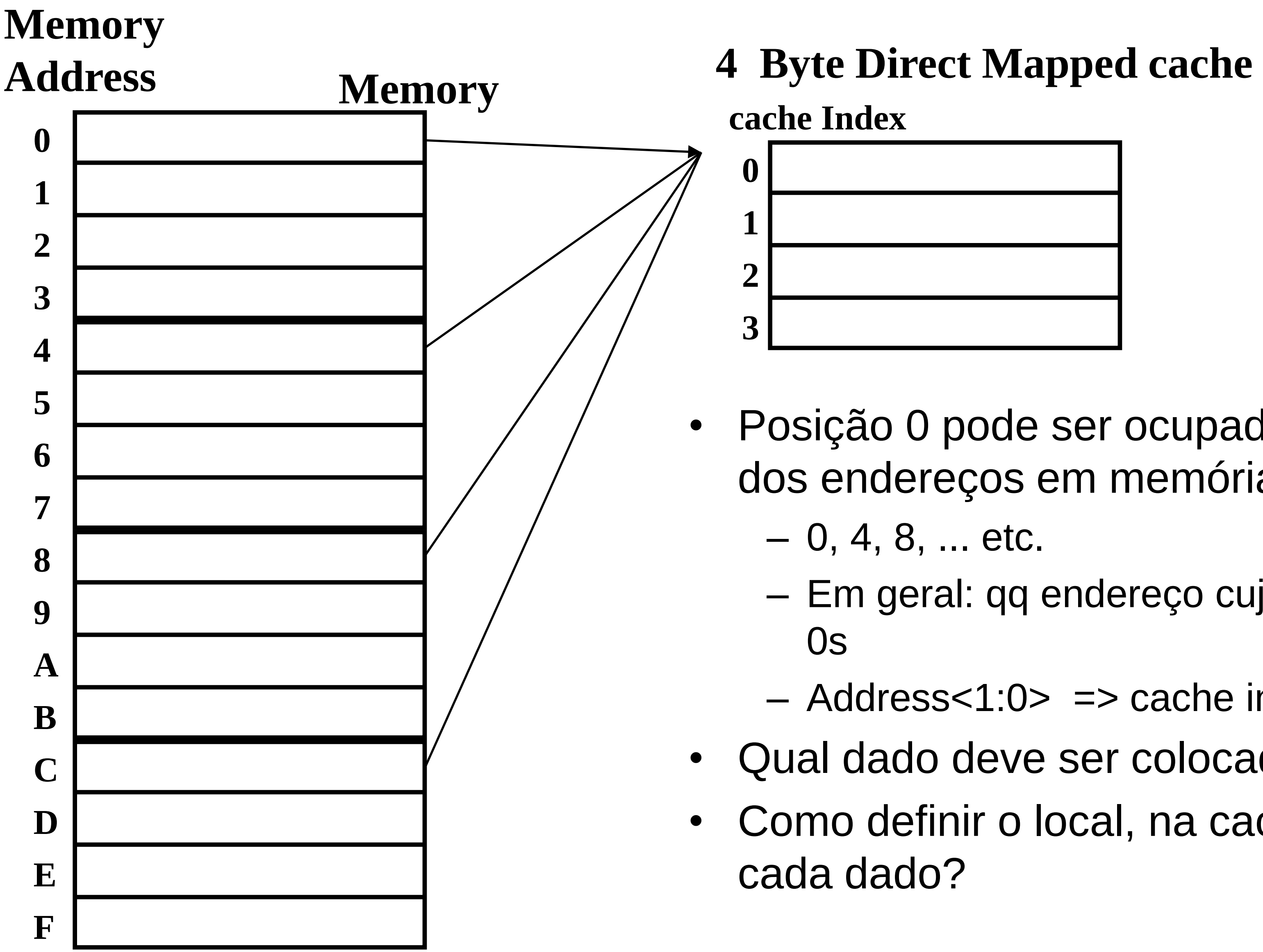
- Princípio da Localidade:
  - Programas acessam relativamente uma pequena porção do espaço de endereçamento em um dado instante de tempo.
- Dois tipos de Localidade:
  - Localidadde Temporal (Localidade no Tempo): Se um item é referenciado, ele tende a ser referenciado outra vez em um curto espaço de tempo (loops)
  - Localidade Espacial (Localidade no Espaço): Se um item é referenciado, itens próximos também tendem a serem referenciados em um curto espaço de tempo (acesso a array)

Hierarquia de Memórias: Terminologia

- Hit: o dado está no upper level (exemplo: Block X)
  - Hit Rate: taxa de hit no upper level no acesso à memória
  - Hit Time: Tempo para o acesso no upper level, consiste em:
    RAM access time + Time to determine hit/miss
- Miss: o dado precisa ser buscado em um bloco no lower level (Block Y)
  - Miss Rate = 1 - (Hit Rate)
  - Miss Penalty: Tempo para colocar um bloco no upper level +
    Tempo para disponibilizar o dado para o processador
- Hit Time << Miss Penalty (500 instruções no 21264!)



cache: Direct Mapped

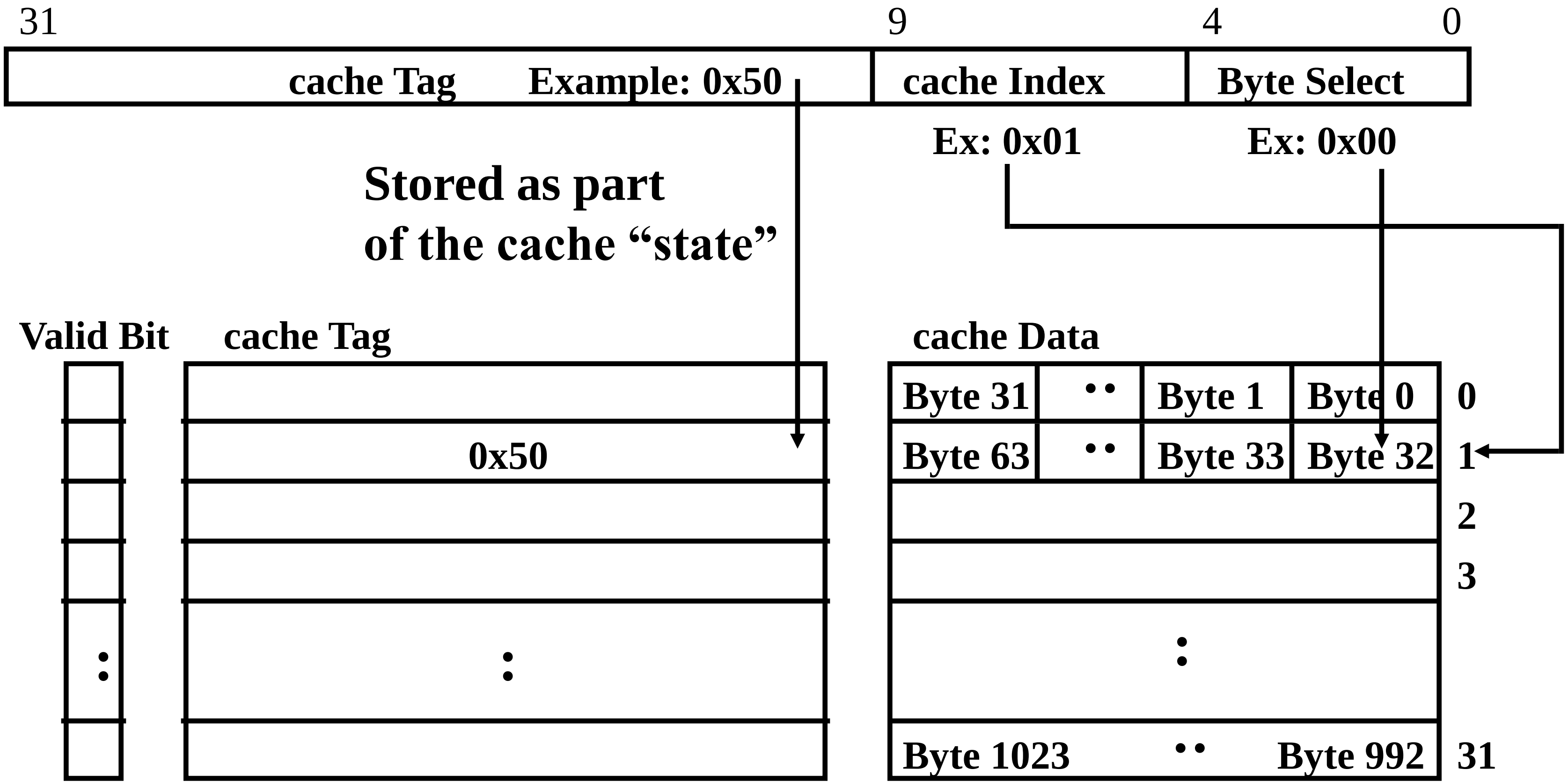


- Posição 0 pode ser ocupada por dados dos endereços em memória:
  - 0, 4, 8, ... etc.
  - Em geral: qq endereço cujos 2 LSBs são 0s
  - Address<1:0> => cache index
- Qual dado deve ser colocado na cache?
- Como definir o local, na cache, para cada dado?

IC-UNICAMP

1 KB Direct Mapped cache, 32B blocks

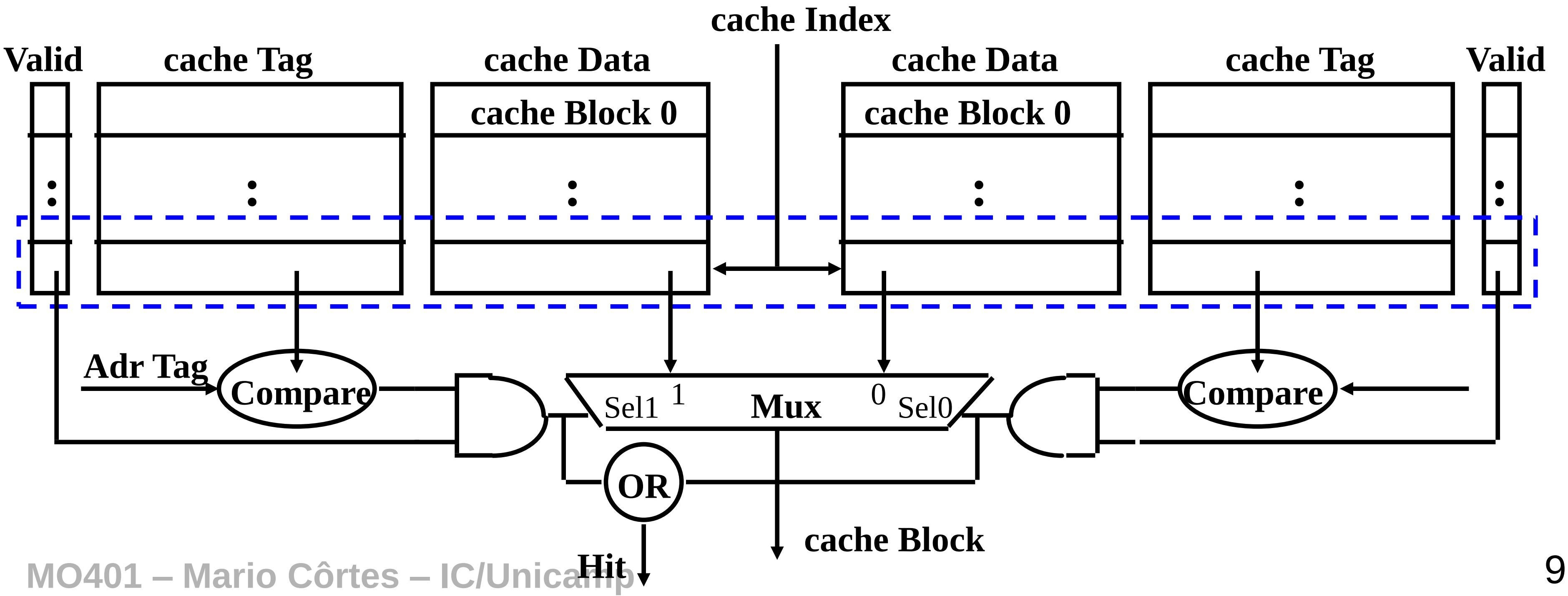
- Para uma cache de 2 ** N byte:
  - Os (32 - N) bits de mais alta ordem são “cache Tag”
  - Os M bits de mais baixa ordem são “Byte Select” (Block Size = 2 ** M)



IC-UNICAMP

Two-way Set Associative cache

- N-way set associative: N entradas para cada cache Index
  - – N direct mapped caches opera em paralelo (N típico: 2 a 4)
- Examplo: Two-way set associative cache
  - – cache Index: seleciona um “set” na cache
  - – As duas tags no set são comparadas em paralelo
  - – O Dado é selecionado basedo no resultado da comparação das tag

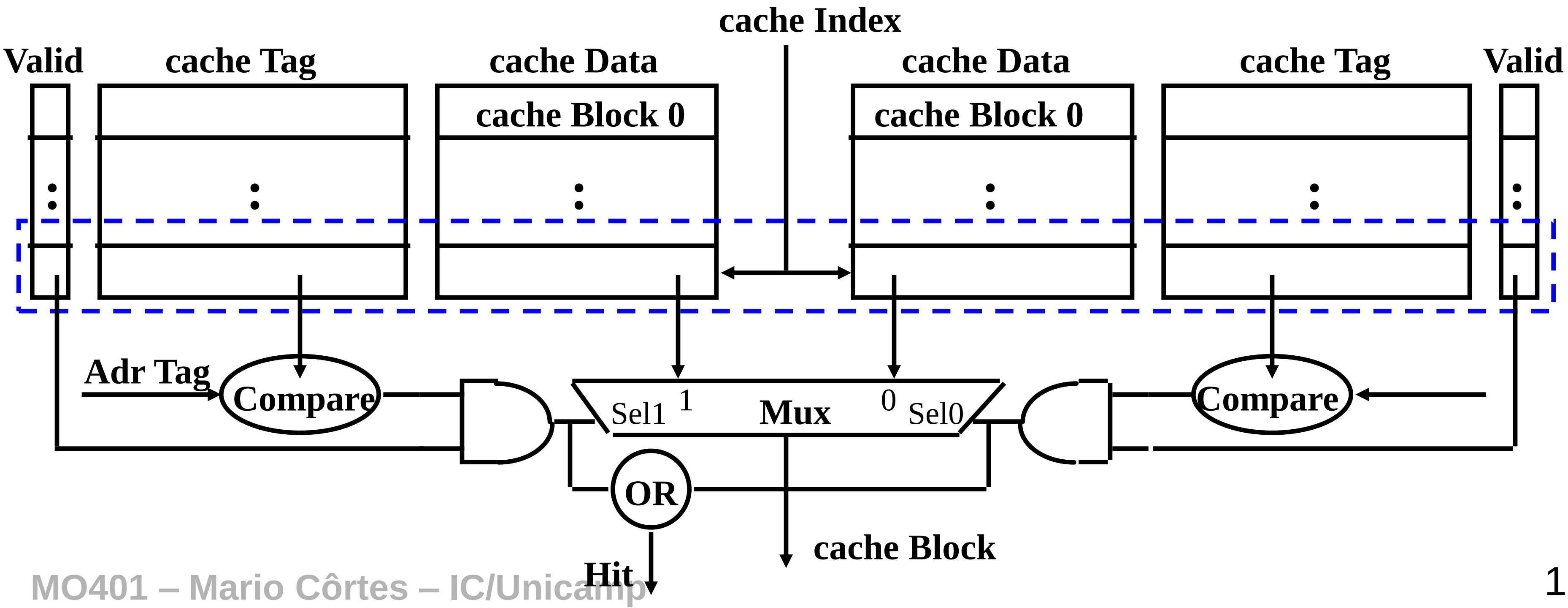


IC-UNICAMP

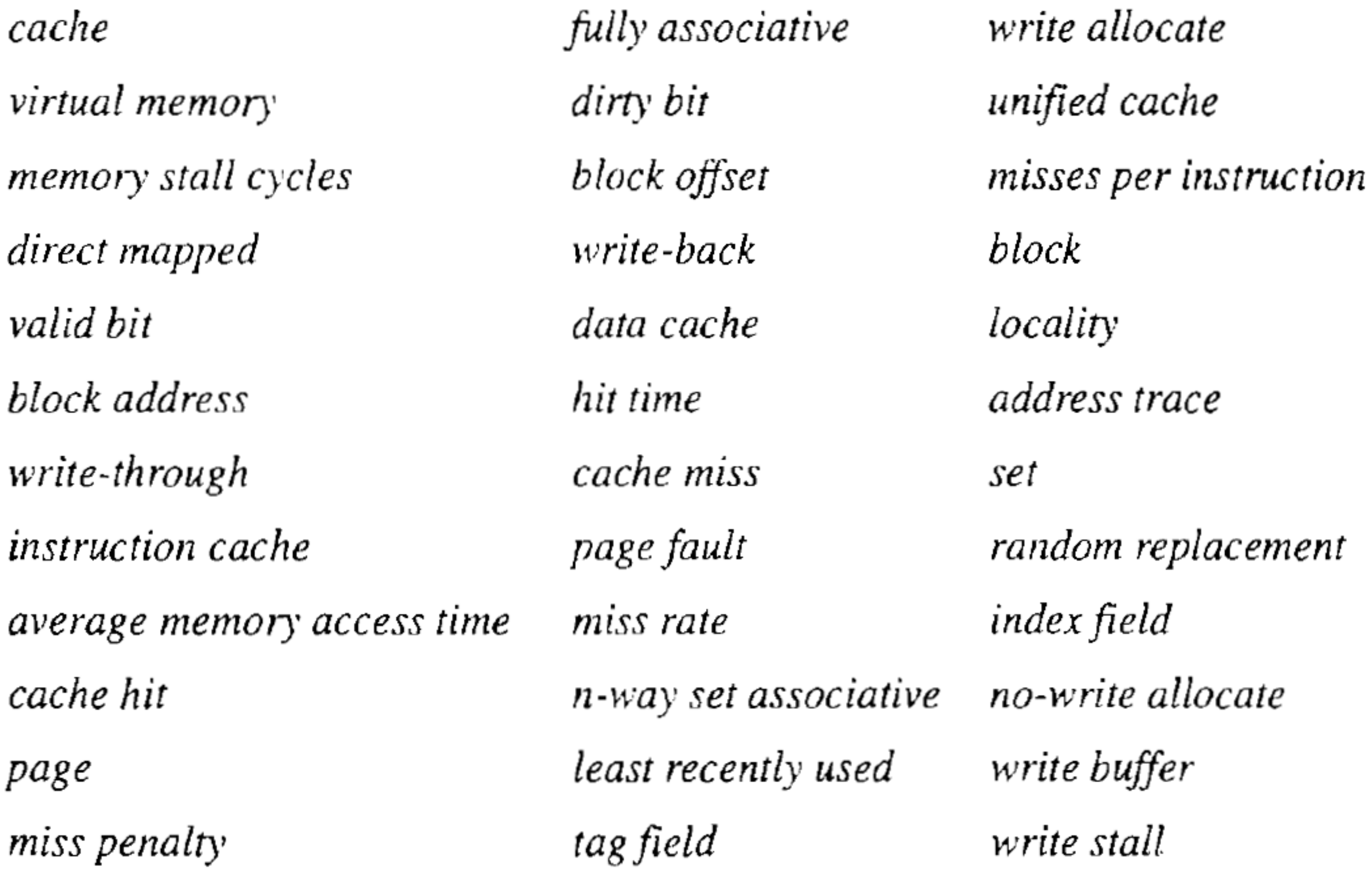
Desvantagem de Set Associative cache

- N-way Set Associative cache v. Direct Mapped cache:
  - N comparadores vs. 1
  - MUX extra, atrasa o acesso ao dado
  - Dado disponível APÓS Hit/Miss
- Direct mapped cache: cache Block disponível ANTES do Hit/Miss:

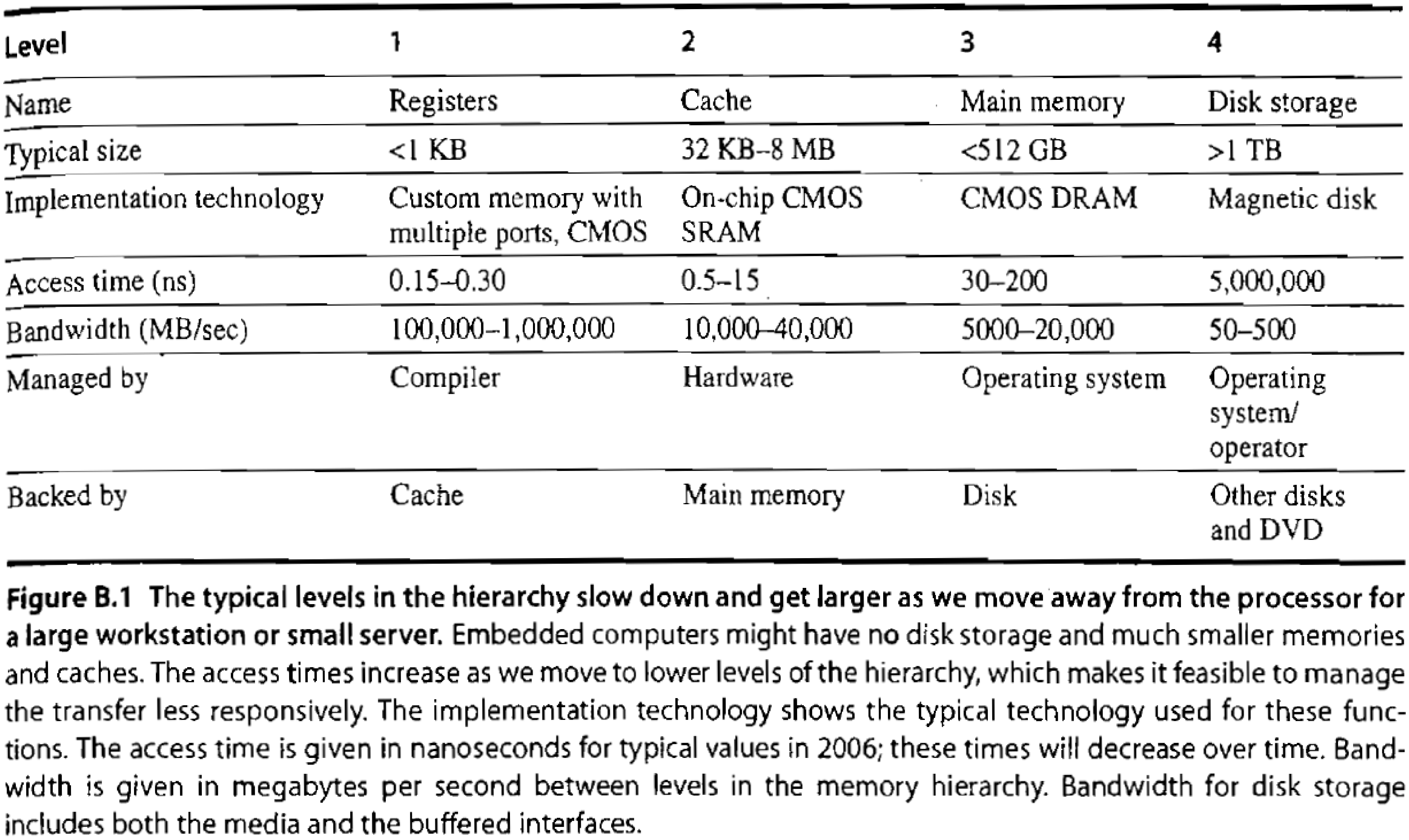
  É possível assumir um hit e continuar. Se miss, Recover.



Termos em revisão na seção B.1



cache
virtual memory
memory stall cycles
direct mapped
valid bit
block address
write-through
instruction cache
average memory access time
cache hit
page
miss penalty
fully associative
dirty bit
block offset
write-back
data cache
hit time
cache miss
page fault
miss rate
n-way set associative
least recently used
tag field
write allocate
unified cache
misses per instruction
block
locality
address trace
set
random replacement
index field
no-write allocate
write buffer
write stall



Level	1	2	3	4
Name	Registers	Cache	Main memory	Disk storage
Typical size	<1 KB	32 KB–8 MB	<512 GB	>1 TB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (ns)	0.15–0.30	0.5–15	30–200	5,000,000
Bandwidth (MB/sec)	100,000–1,000,000	10,000–40,000	5000–20,000	50–500
Managed by	Compiler	Hardware	Operating system	Operating system/ operator
Backed by	Cache	Main memory	Disk	Other disks and DVD

Figure B.1 The typical levels in the hierarchy slow down and get larger as we move away from the processor for a large workstation or small server. Embedded computers might have no disk storage and much smaller memories and caches. The access times increase as we move to lower levels of the hierarchy, which makes it feasible to manage the transfer less responsively. The implementation technology shows the typical technology used for these functions. The access time is given in nanoseconds for typical values in 2006; these times will decrease over time. Bandwidth is given in megabytes per second between levels in the memory hierarchy. Bandwidth for disk storage includes both the media and the buffered interfaces.

cache performance

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

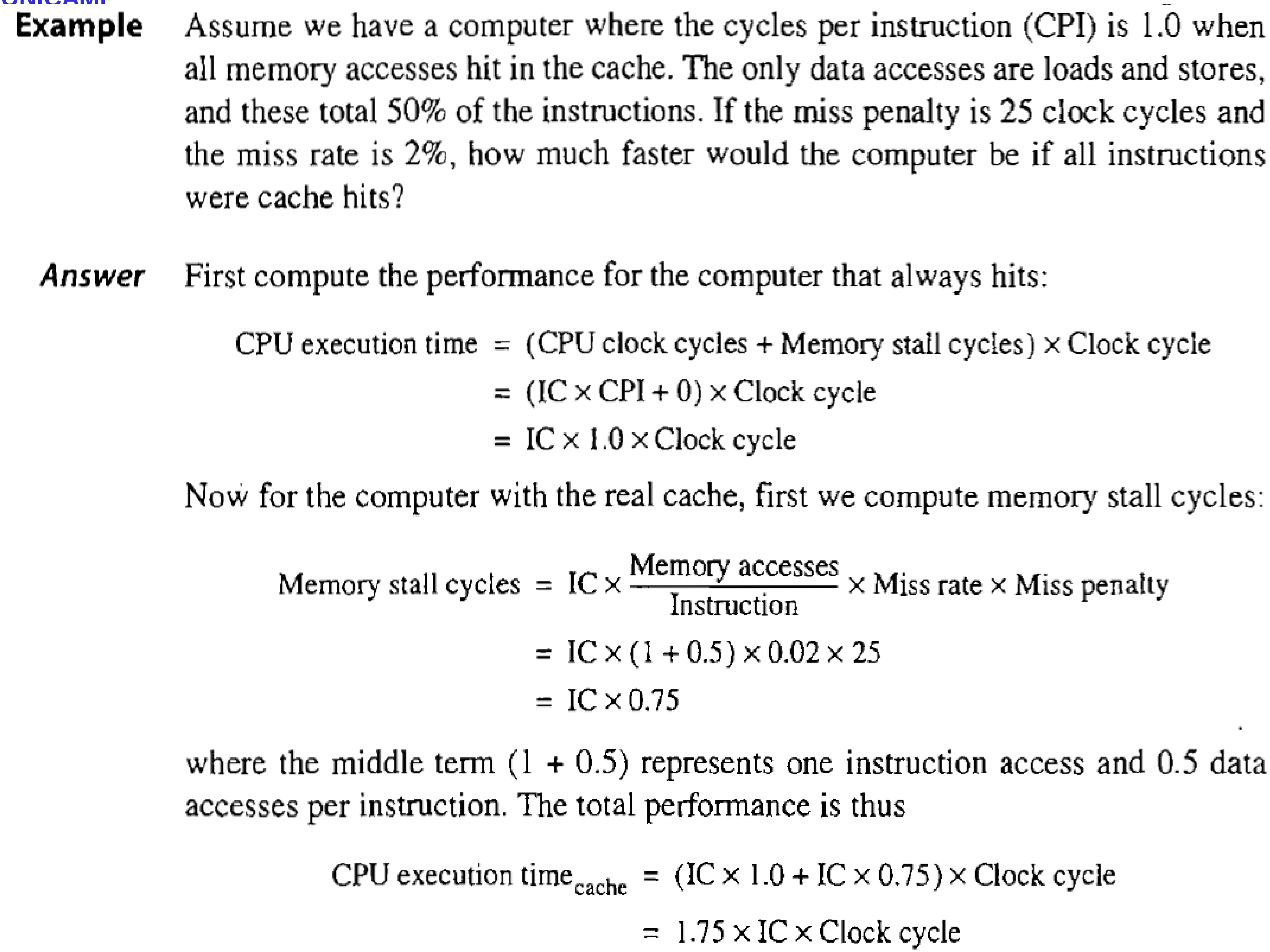
$$\text{Memory stall cycles} = \#\,\text{Misses} \,.\, \text{Miss penalty}$$

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

$$= IC \,.\, \frac{\text{Memory accesses}}{\text{Instruction}} \,.\, \text{Miss rate} \,.\, \text{Miss penalty}$$

Obs: assumindo mesmo comportamento para Wr e Rd

Exemplo pag B-5



Example 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 First compute the performance for the computer that always hits:

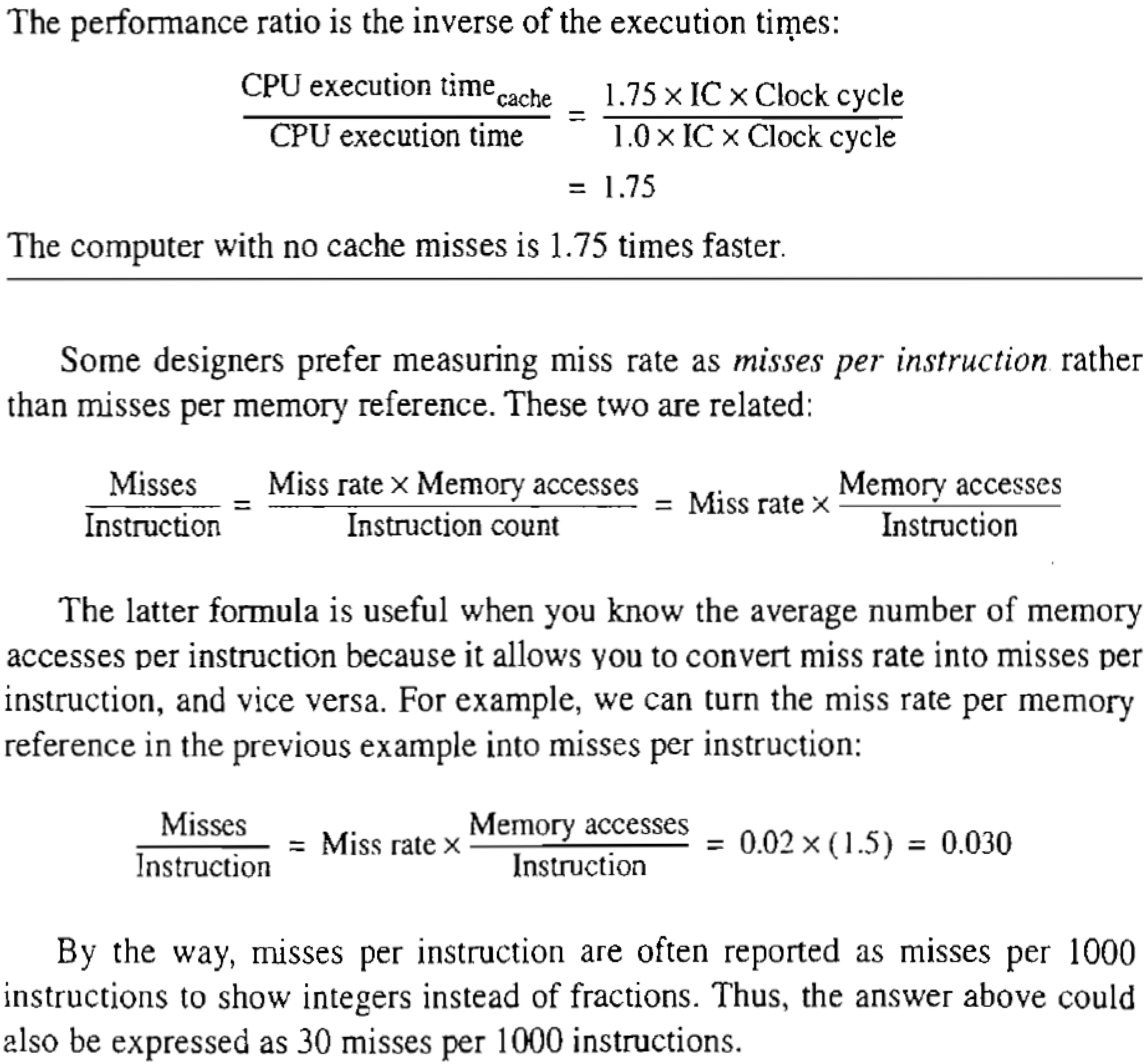
$$\begin{aligned}\text{CPU execution time} &= (\text{CPU clock cycles} + \text{Memory stall cycles}) \times \text{Clock cycle}\\ &= (\text{IC} \times \text{CPI} + 0) \times \text{Clock cycle}\\ &= \text{IC} \times 1.0 \times \text{Clock cycle}\end{aligned}$$

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:

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

$$= 1.75$$

The computer with no cache misses is 1.75 times faster.

Some designers prefer measuring miss rate as *misses per instruction* rather than misses per memory reference. These two are related:

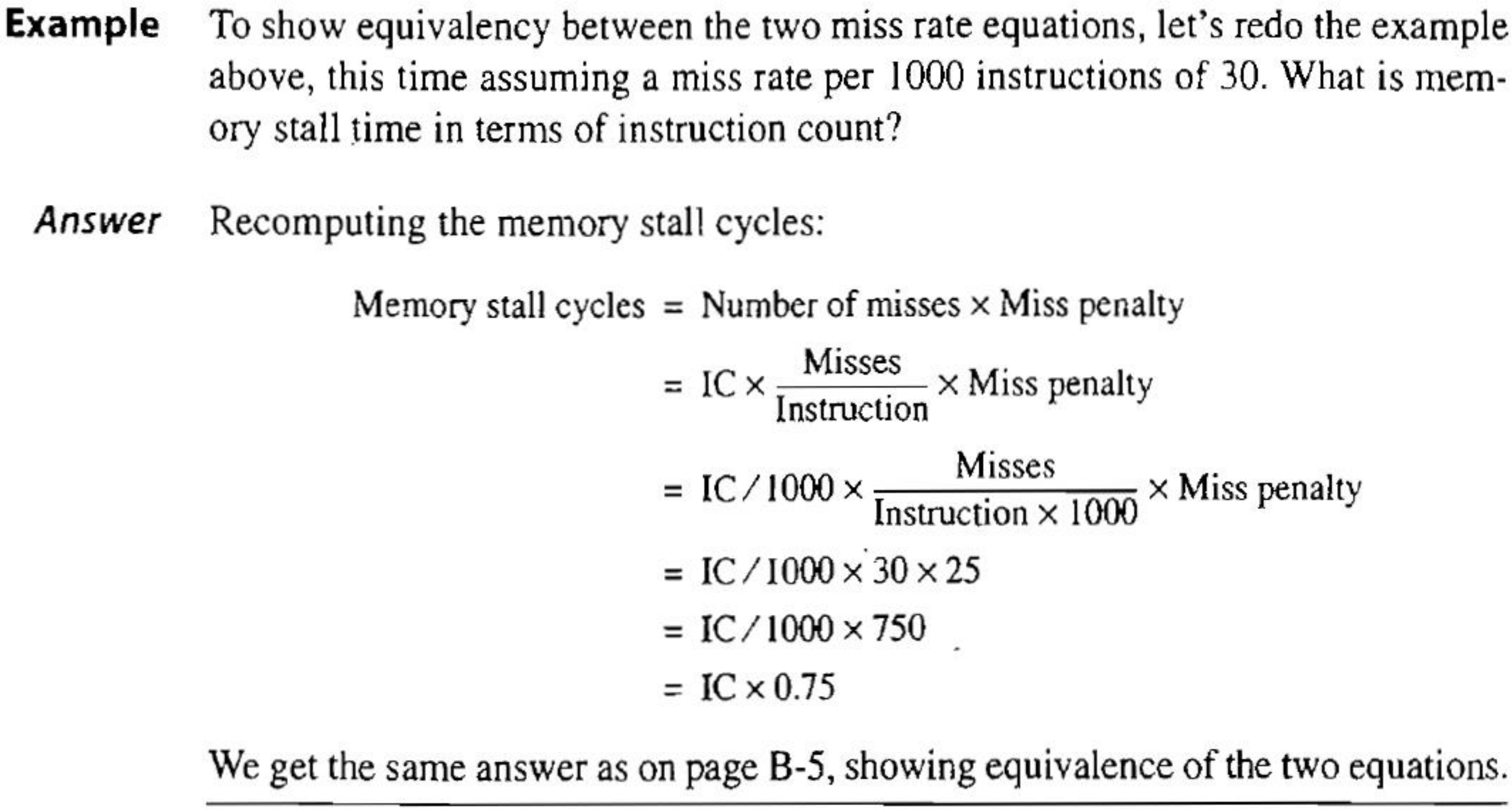
$$\frac{\text{Misses}}{\text{Instruction}} = \frac{\text{Miss rate} \times \text{Memory accesses}}{\text{Instruction count}} = \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}}$$

The latter formula is useful when you know the average number of memory accesses per instruction because it allows you to convert miss rate into misses per instruction, and vice versa. For example, we can turn the miss rate per memory reference in the previous example into misses per instruction:

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

By the way, misses per instruction are often reported as misses per 1000 instructions to show integers instead of fractions. Thus, the answer above could also be expressed as 30 misses per 1000 instructions.

Exemplo pag 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.

Hierarquia de Memória: 4 Questões

- Q1: Em que lugar colocar um bloco no upper level? → *(Block placement)*

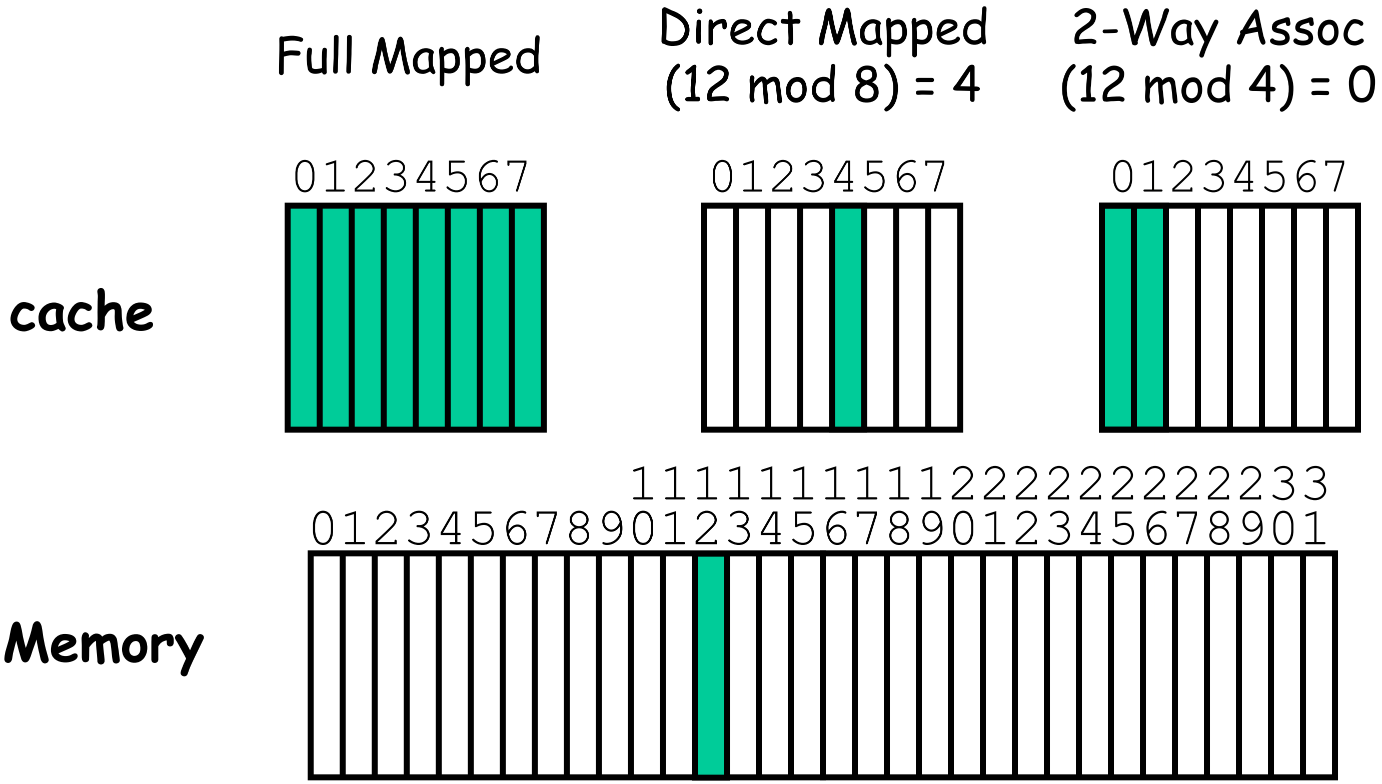
- Q2: Como localizar o bloco se ele está no upper level? → *(Block identification)*

- Q3: Qual bloco deve ser trocado em um miss? → *(Block replacement)*

- Q4: O que ocorre em um write? → .*(Write strategy)*

Q1: Em que lugar colocar um bloco no upper level? (política de endereçamento)

- Colocar o Bloco 12 em uma cache de 8 blocos :
  - Fully associative, direct mapped, 2-way set associative
  - S.A. Mapping = (Block Address) mod (# Sets in cache)



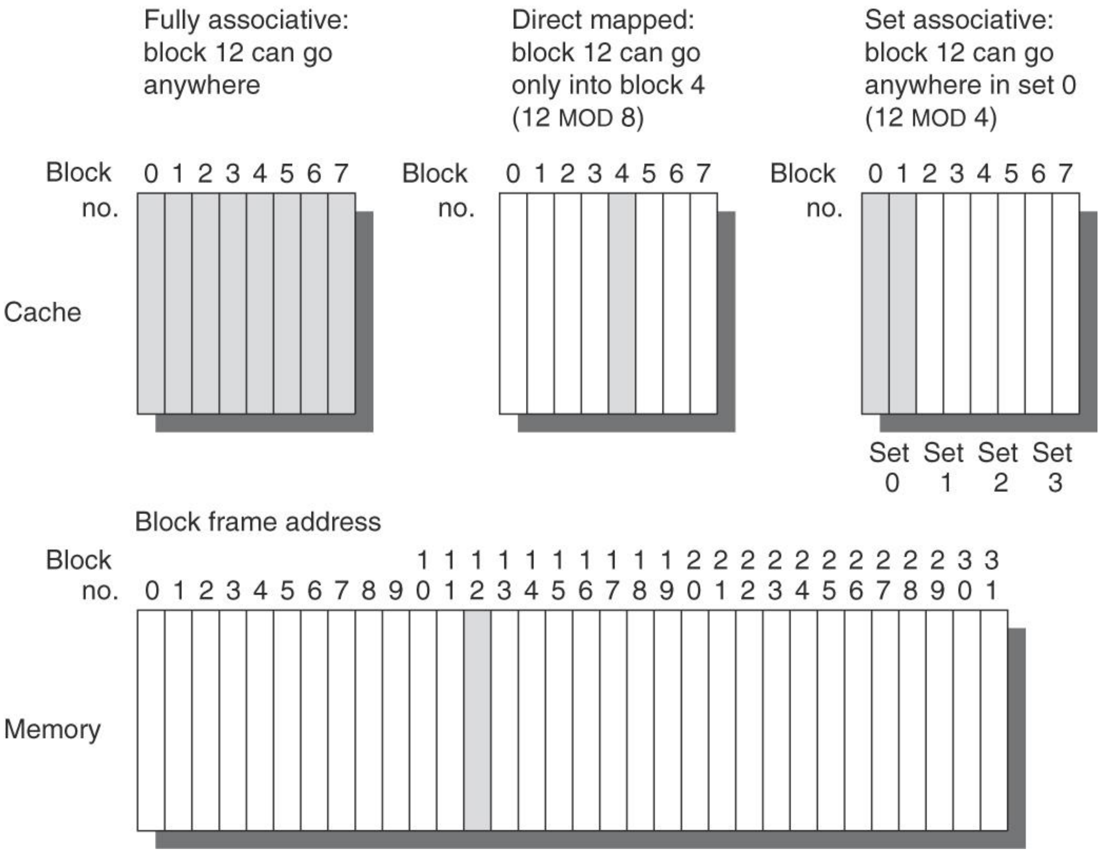
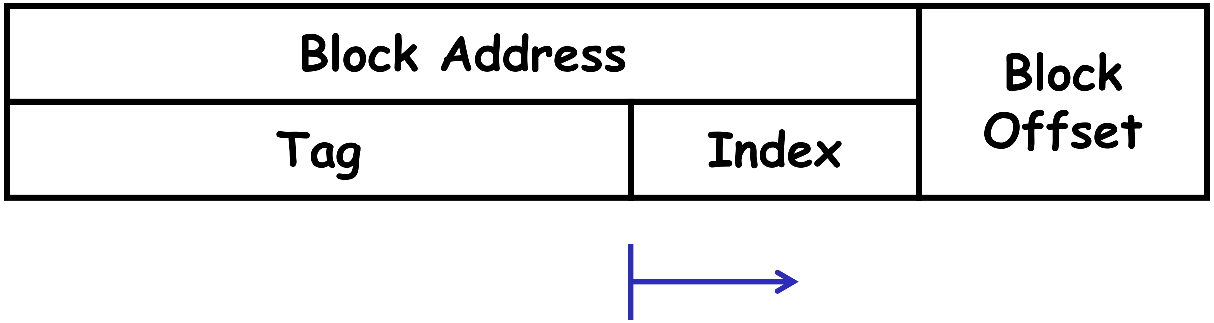


Figure B.2 This example cache has eight block frames and memory has 32 blocks. The three options for caches are shown left to right. In fully associative, block 12 from the lower level can go into any of the eight block frames of the cache. With direct mapped, block 12 can only be placed into block frame 4 (12 modulo 8). Set associative, which has some of both features, allows the block to be placed anywhere in set 0 (12 modulo 4). With two blocks per set, this means block 12 can be placed either in block 0 or in block 1 of the cache. Real caches contain thousands of block frames, and real memories contain millions of blocks. The set associative organization has four sets with two blocks per set, called *two-way set associative*. Assume that there is nothing in the cache and that the block address in question identifies lower-level block 12.

Q2: Como localizar o bloco se ele está no upper level?

- Tag em cada bloco
  - Não é necessário testar o index ou block offset

- O aumento da associatividade reduz o index e aumenta a tag

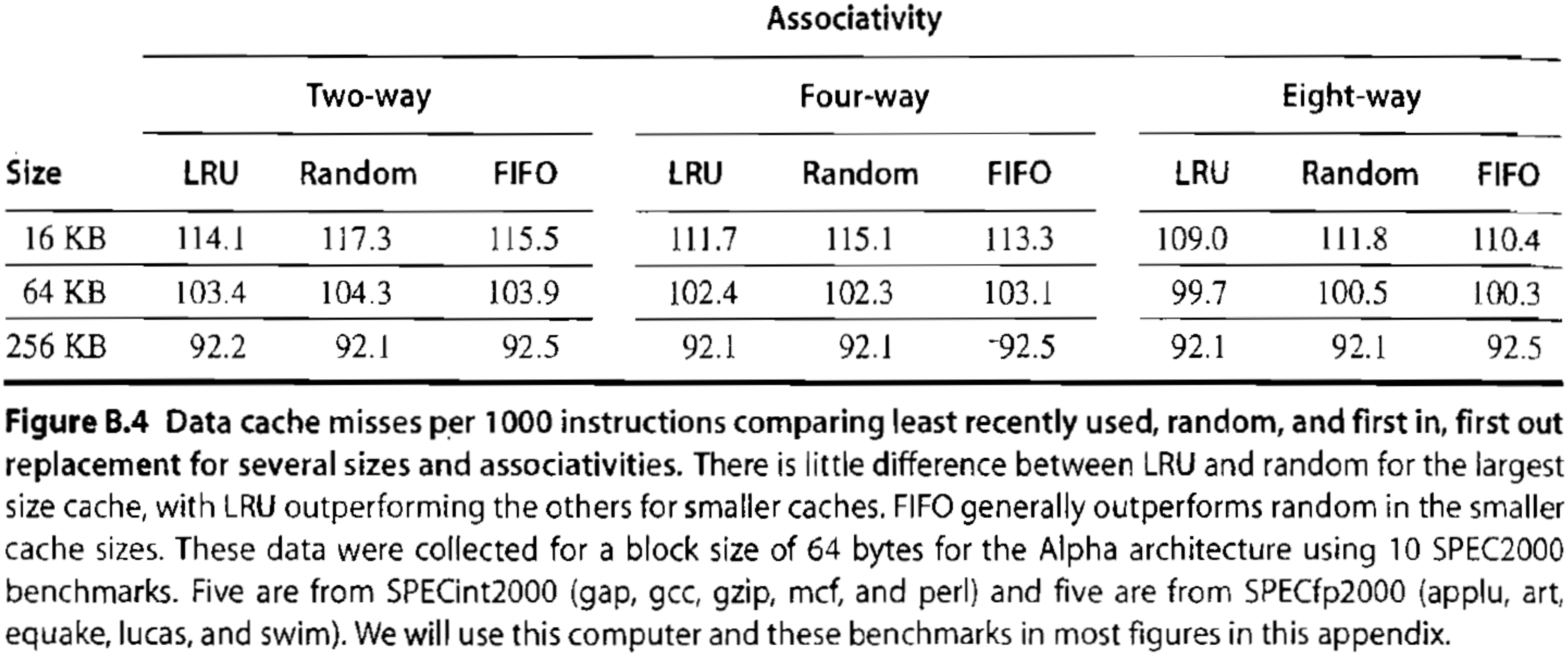


Block Address		Block Offset
Tag	Index	

Aumento associatividade

Q3: Qual bloco deve ser trocado em um miss? (política de substituição)

- Fácil para Direct Mapped
- Set Associative or Fully Associative:
  - – Random
  - – LRU (Least Recently Used)
  - – FIFO



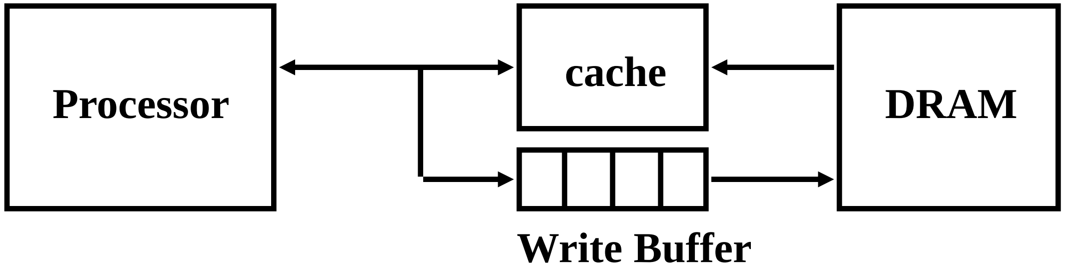
	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

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: O que ocorre em um write? (política de escrita)

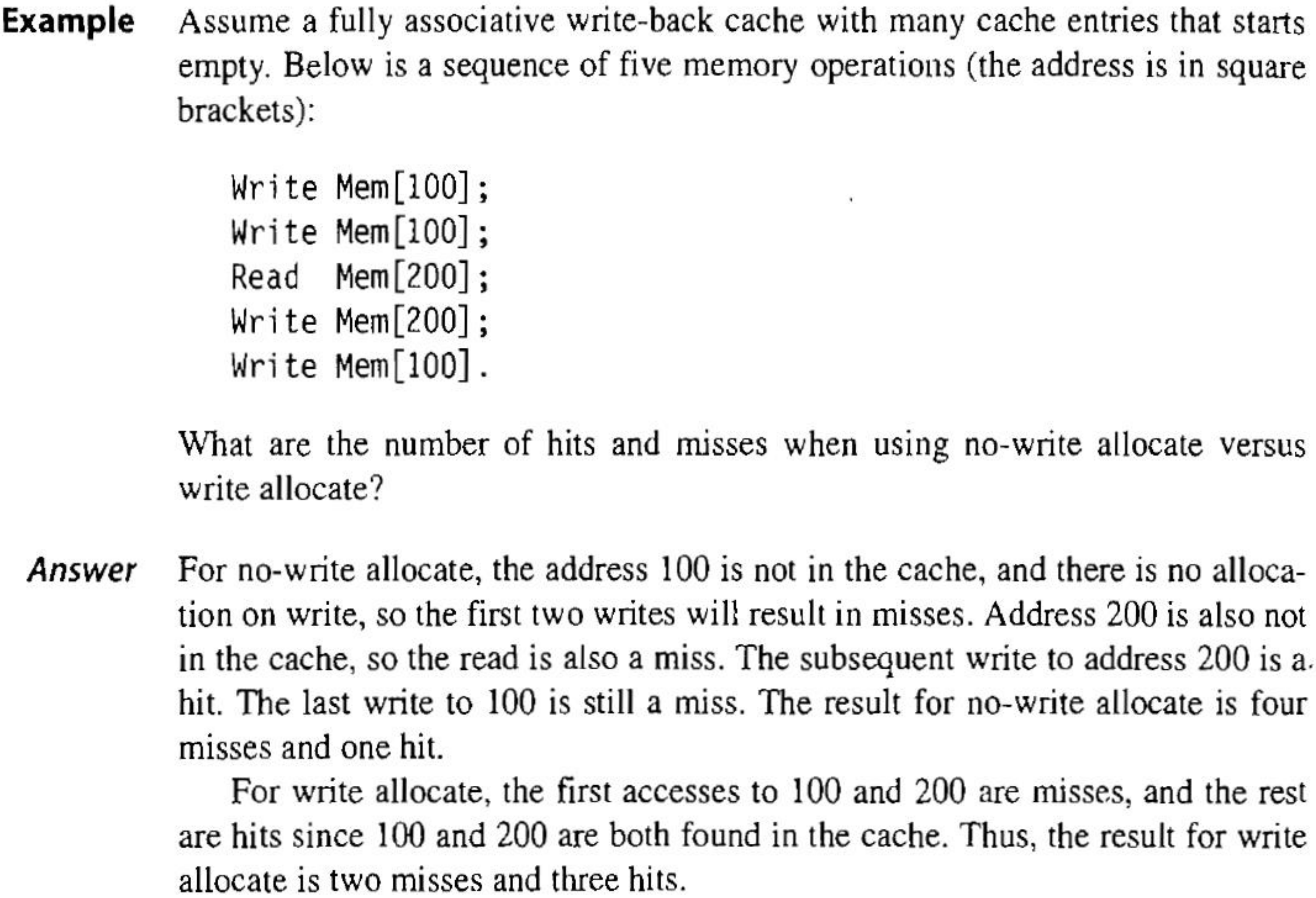
- *Write through* — A informação é escrita tanto no bloco da cache quanto no bloco do lower-level memory.
- *Write back* — A informação é escrita somente no bloco da cache. O bloco da cache modificado é escrito na memória principal somente quando ele é trocaddo.
  - block clean or dirty?
- Prós e Contras?
  - WT: cache is clean: read misses don´t cause WB substitution
  - WB: não há repetição de writes na mesma posição
- WT, em geral, é combinado com write buffers, assim não há espera pelo lower level memory
- Write miss:
  - WR allocate: semelhante a Rd miss seguido de Wr
  - No WR-allocate: escrita direta no nível inferior; cache não afetada

Write Buffer para Write Through



- Um Write Buffer colocado entre a cache e a Memory
  - Processador: escreve o dado na cache e no write buffer
  - Memory controller: escreve o conteúdo do buffer na memória
- Write buffer é uma FIFO:
  - Número típico de entradas: 4
  - Trabalha bem se: freqüência de escrita (w.r.t. time) << 1 / DRAM write cycle
- Memory system é um pesadelo para o projetista :
  - freqüência de escrita (w.r.t. time) -> 1 / DRAM write cycle
  - Saturação do Write buffer

Exmpl B-12



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?

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.

IC-UNICAMP

Fig B-6: Data cache – Opteron

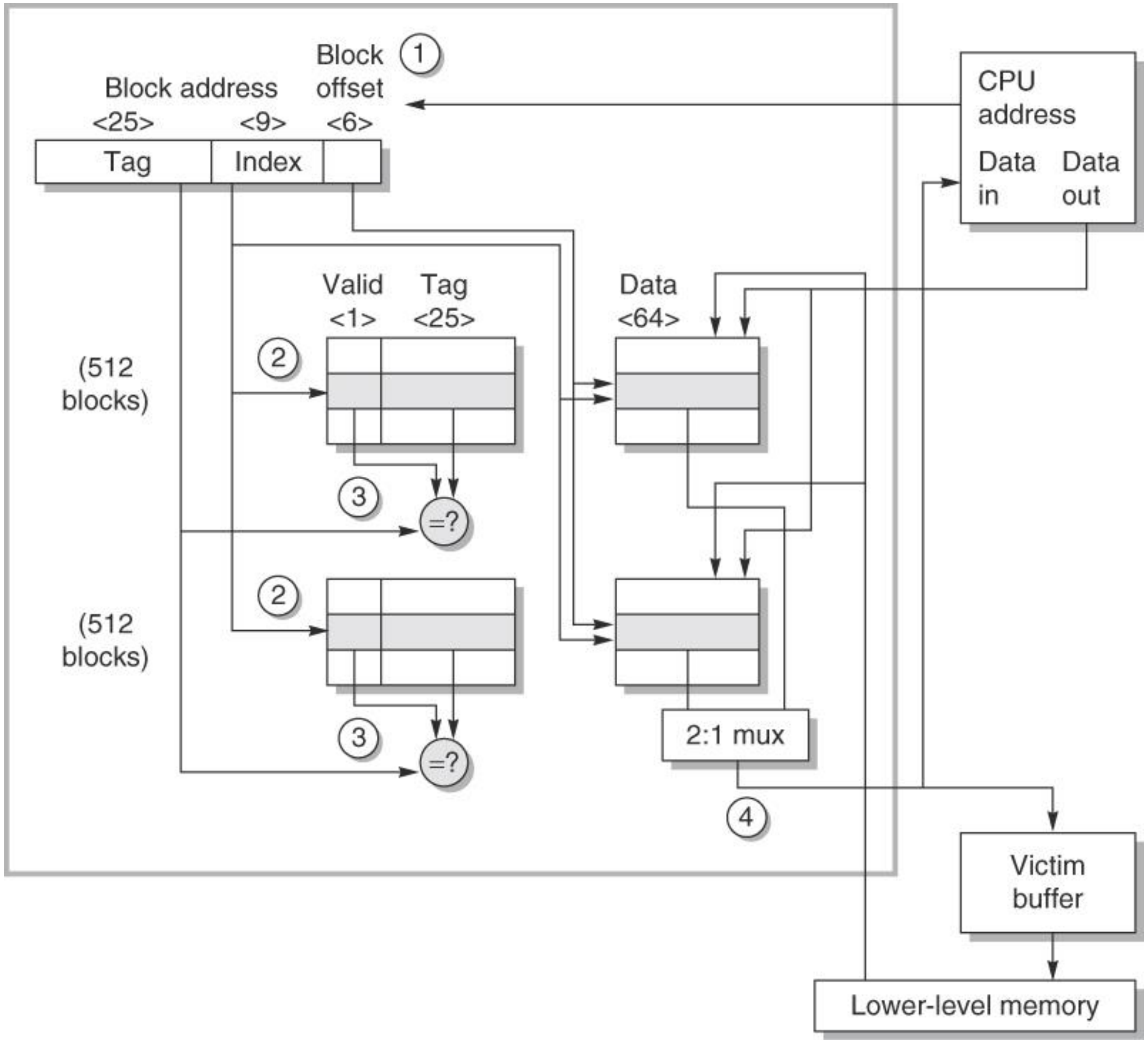
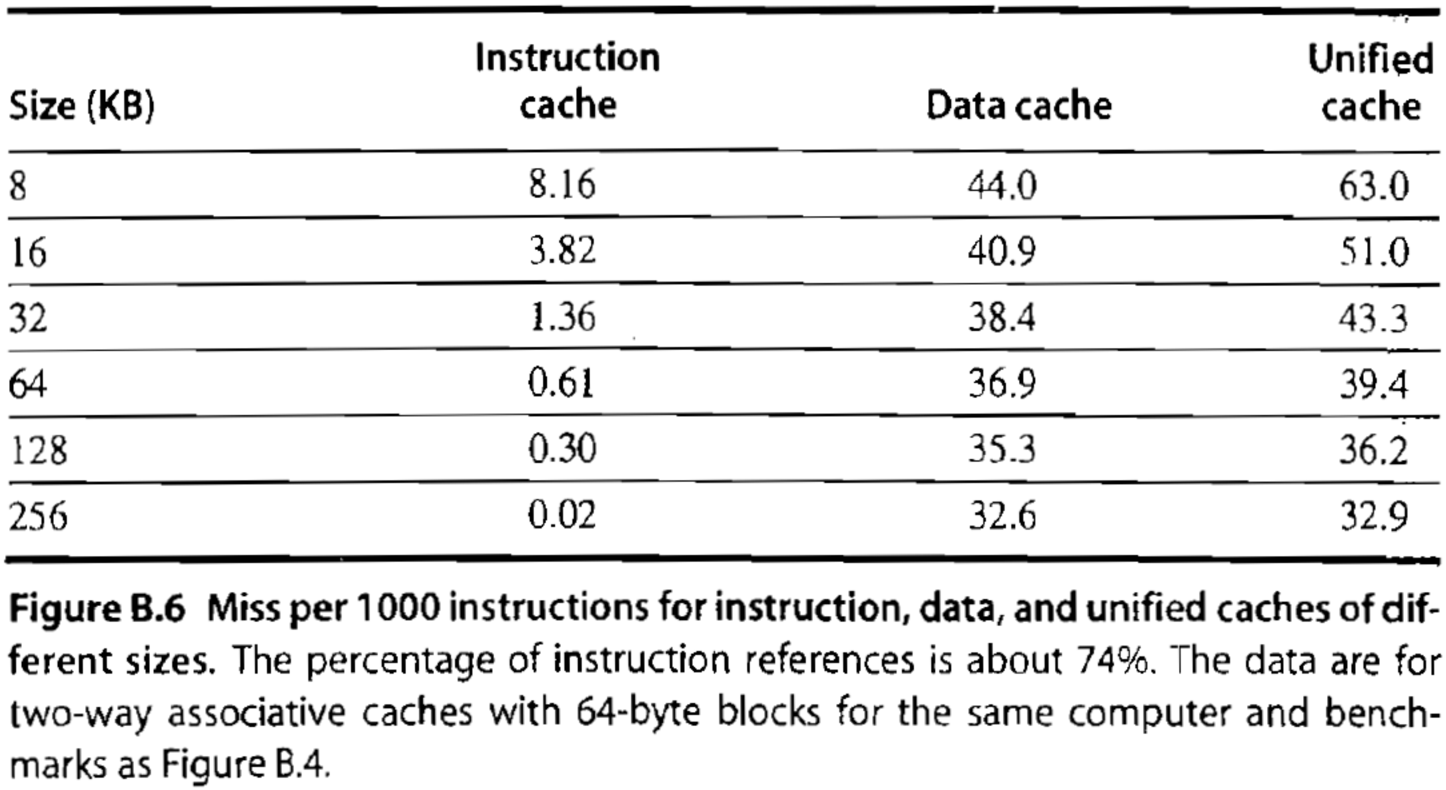


Figure B.5 The organization of the data cache in the Opteron microprocessor. The 64 KB cache is two-way set associative with 64-byte blocks. The 9-bit index selects among 512 sets. The four steps of a read hit, shown as circled numbers in order of occurrence, label this organization. Three bits of the block offset join the index to supply the RAM address to select the proper 8 bytes. Thus, the cache holds two groups of 4096 64-bit words, with each group containing half of the 512 sets. Although not exercised in this example, the line from lower-level memory to the cache is used on a miss to load the cache. The size of address leaving the processor is 40 bits because it is a physical address and not a virtual address. Figure B.24 on page B-47 explains how the Opteron maps from virtual to physical for a cache access.

caches unificadas ou I&D



Size (KB)	Instruction cache	Data cache	Unified cache
8	8.16	44.0	63.0
16	3.82	40.9	51.0
32	1.36	38.4	43.3
64	0.61	36.9	39.4
128	0.30	35.3	36.2
256	0.02	32.6	32.9

Figure B.6 Miss per 1000 instructions for instruction, data, and unified caches of different sizes. The percentage of instruction references is about 74%. The data are for two-way associative caches with 64-byte blocks for the same computer and benchmarks as Figure B.4.

B-2: cache performance

- Medida básica de desempenho:

Average Memory Access Time = AMAT =

Total Acess Time / Total # accesses =

(total time when hit + total time when miss) / # accesses =

Ave HitTime + (total time when miss) / # accesses =

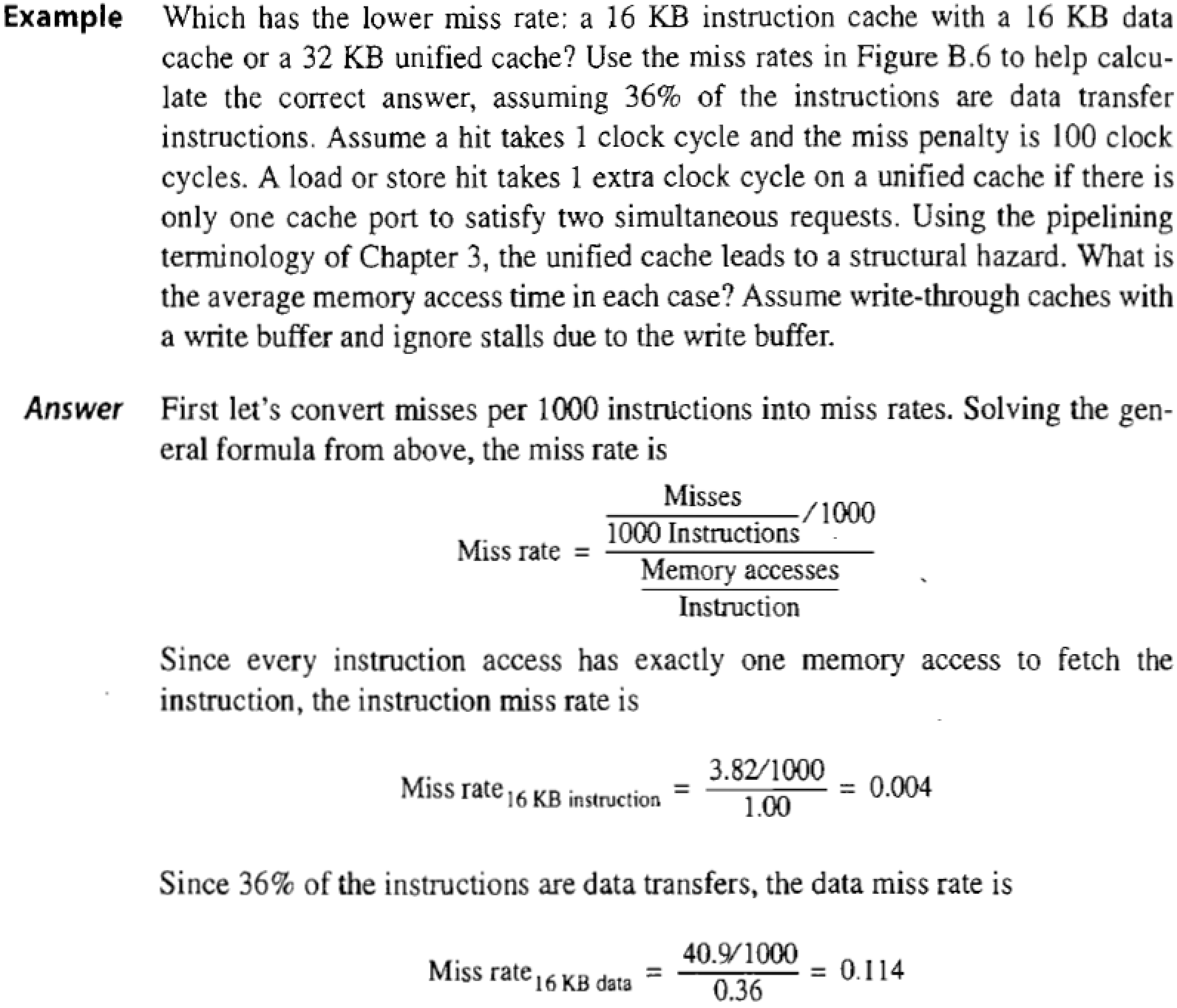
Ave HitTime + (# misses * miss time) / #accesses =

HitTime + (# missess / # accesses) * Miss penalty

AMAT = HitTime + Miss rate x Miss penalty

IC-UNICAMP

Ex B.16



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.

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

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

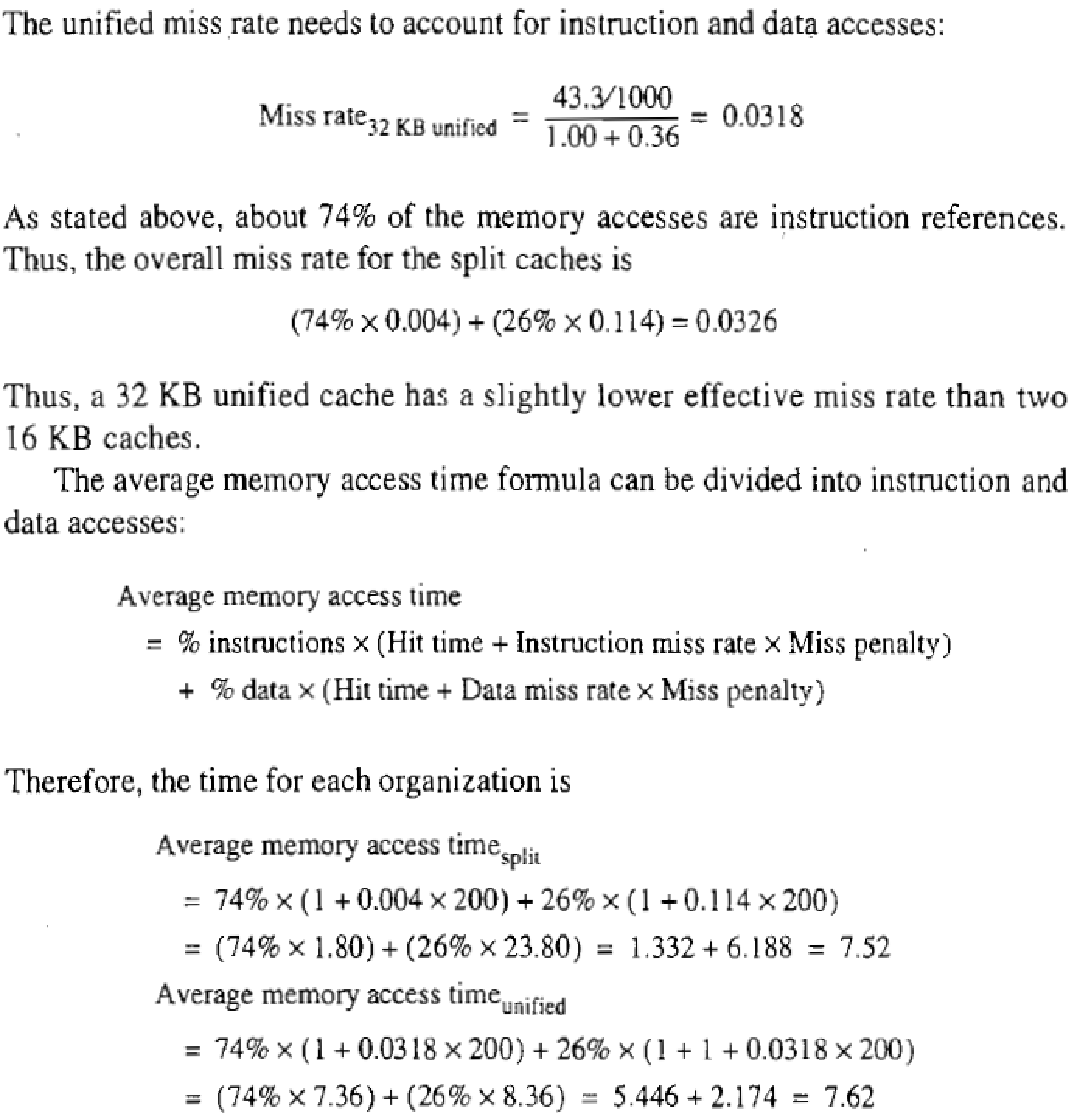
Since every instruction access has exactly one memory access to fetch the instruction, the instruction miss rate is

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

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

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

Ex B.16 (2)



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

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

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:

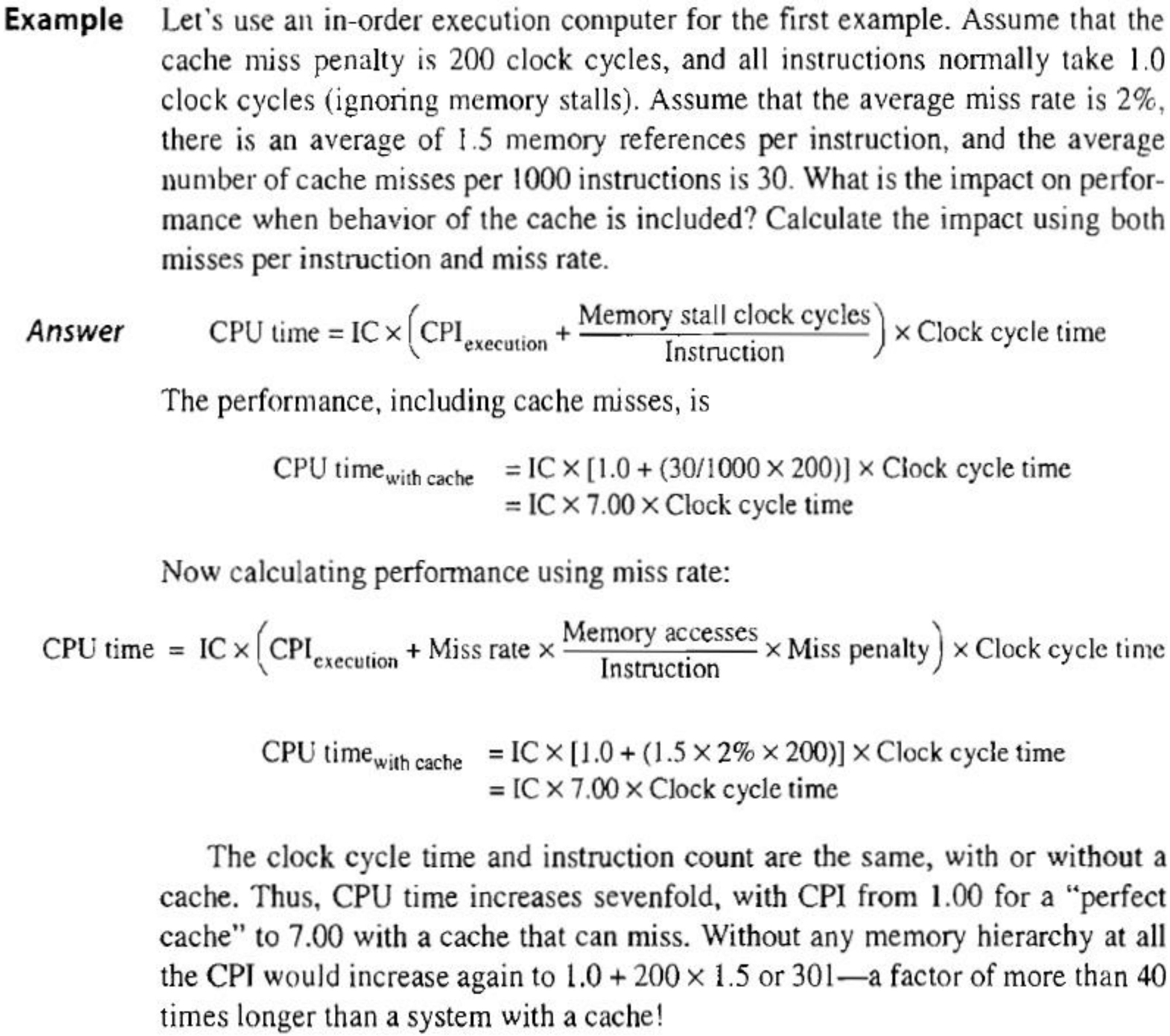
$$\begin{aligned} &\text{Average memory access time} \\ &\quad = \%\ \text{instructions} \times (\text{Hit time} + \text{Instruction miss rate} \times \text{Miss penalty}) \\ &\qquad + \%\ \text{data} \times (\text{Hit time} + \text{Data miss rate} \times \text{Miss penalty}) \end{aligned}$$

Therefore, the time for each organization is

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

IC-UNICAMP

Ex
B.18



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.

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:

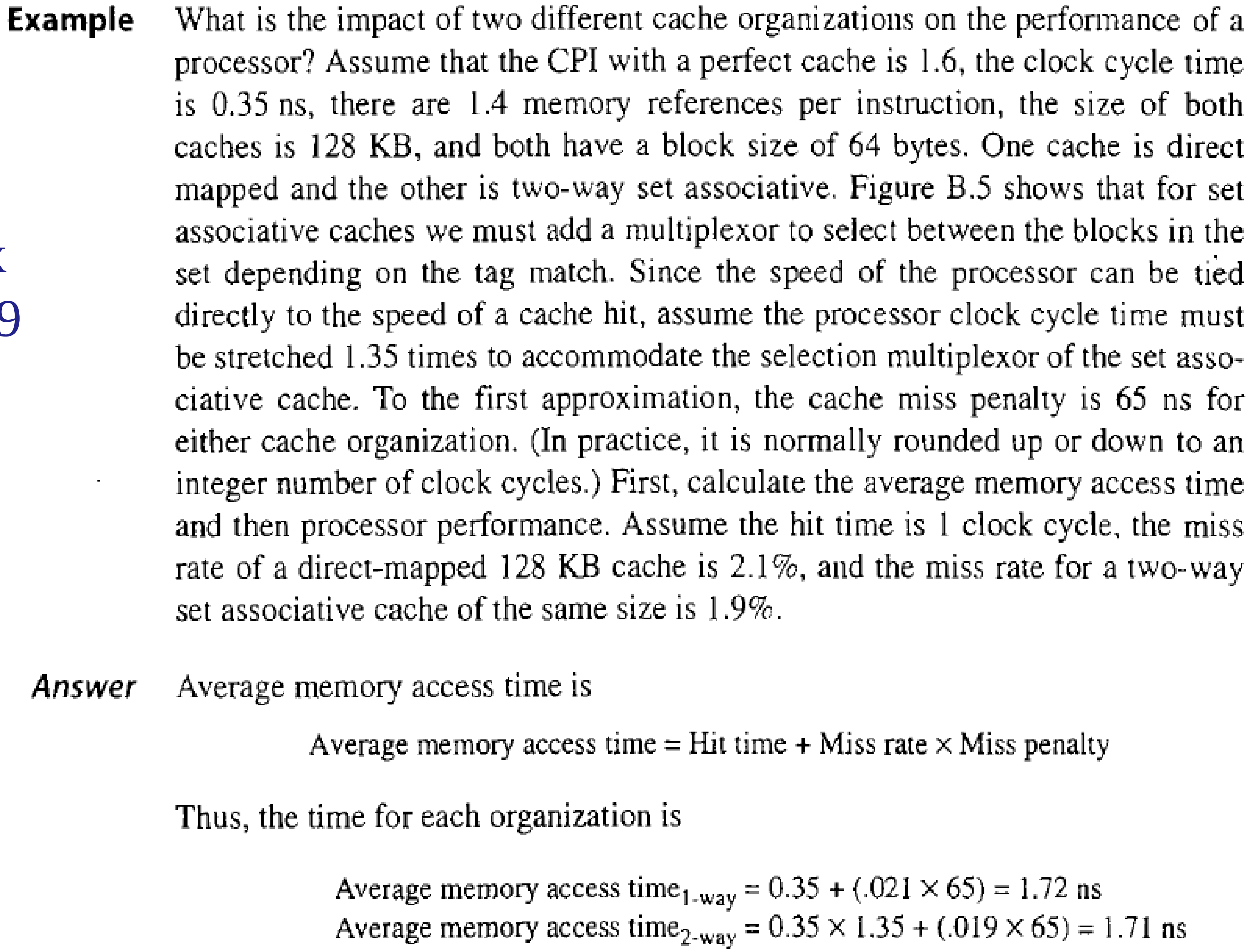
$$\text{CPU time} = \text{IC} \times \left(\text{CPI}_{\text{execution}} + \text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty}\right) \times \text{Clock cycle time}$$

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

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!

IC-UNI

Ex B.19



Example What is the impact of two different cache organizations on the performance of a processor? Assume that the CPI with a perfect cache is 1.6, the clock cycle time is 0.35 ns, there are 1.4 memory references per instruction, the size of both caches is 128 KB, and both have a block size of 64 bytes. One cache is direct mapped and the other is two-way set associative. Figure B.5 shows that for set associative caches we must add a multiplexor to select between the blocks in the set depending on the tag match. Since the speed of the processor can be tied directly to the speed of a cache hit, assume the processor clock cycle time must be stretched 1.35 times to accommodate the selection multiplexor of the set associative cache. To the first approximation, the cache miss penalty is 65 ns for either cache organization. (In practice, it is normally rounded up or down to an integer number of clock cycles.) First, calculate the average memory access time and then processor performance. Assume the hit time is 1 clock cycle, the miss rate of a direct-mapped 128 KB cache is 2.1%, and the miss rate for a two-way set associative cache of the same size is 1.9%.

Answer Average memory access time is

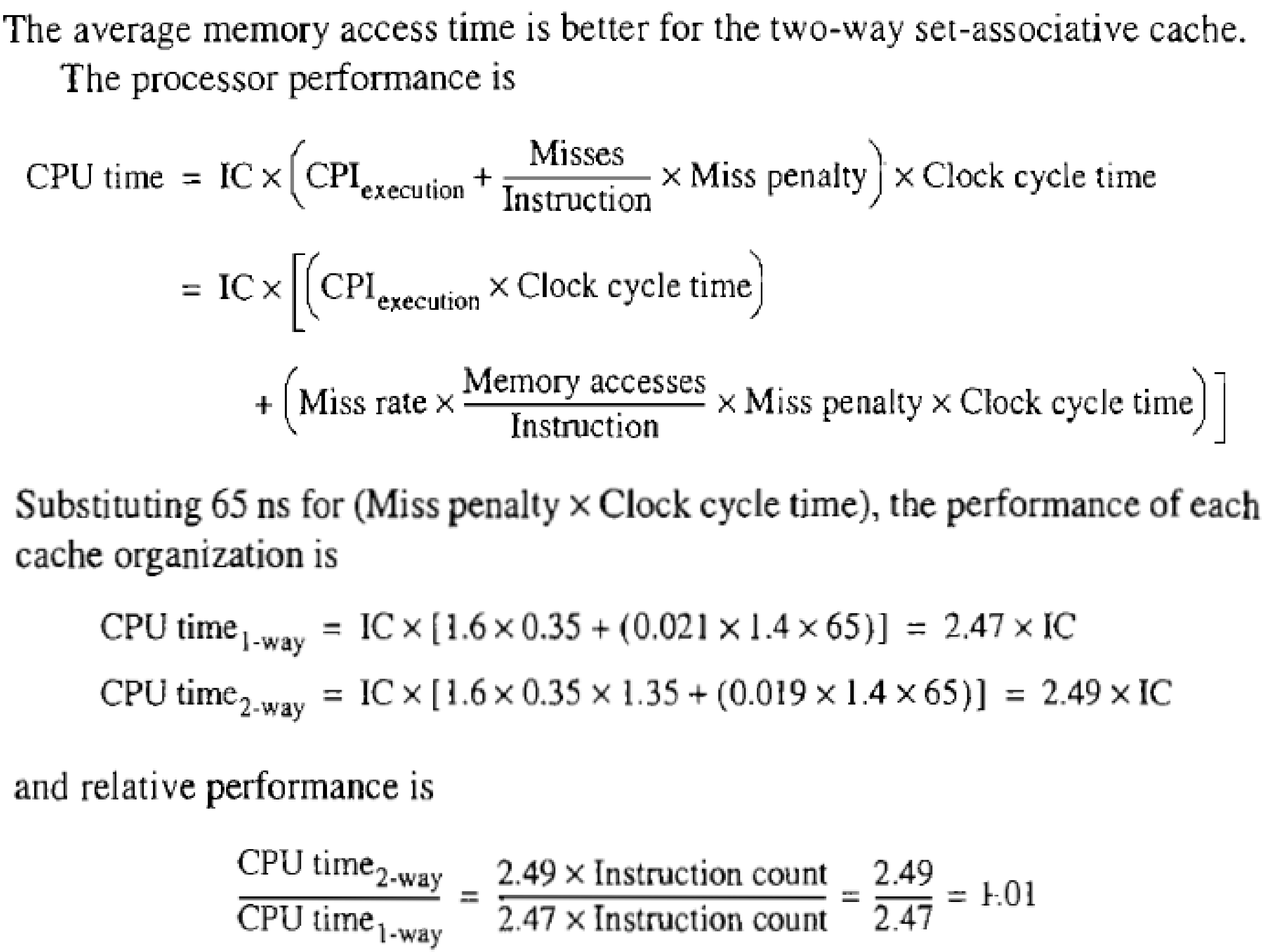
$$\text{Average memory access time} = \text{Hit time} + \text{Miss rate} \times \text{Miss penalty}$$

Thus, the time for each organization is

$$\text{Average memory access time}_{1\text{-way}} = 0.35 + (.021 \times 65) = 1.72 \text{ ns}$$
$$\text{Average memory access time}_{2\text{-way}} = 0.35 \times 1.35 + (.019 \times 65) = 1.71 \text{ ns}$$

IC-UNICAMP

Ex B.19 (2)



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

The processor performance is

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

$$= \text{IC} \times \left[\left(\text{CPI}_{\text{execution}} \times \text{Clock cycle time}\right)\right.$$

$$\left. + \left(\text{Miss rate} \times \frac{\text{Memory accesses}}{\text{Instruction}} \times \text{Miss penalty} \times \text{Clock cycle time}\right)\right]$$

Substituting 65 ns for (Miss penalty × Clock cycle time), the performance of each cache organization is

$$\text{CPU time}_{\text{1-way}} = \text{IC} \times [1.6 \times 0.35 + (0.021 \times 1.4 \times 65)] = 2.47 \times \text{IC}$$

$$\text{CPU time}_{\text{2-way}} = \text{IC} \times [1.6 \times 0.35 \times 1.35 + (0.019 \times 1.4 \times 65)] = 2.49 \times \text{IC}$$

and relative performance is

$$\frac{\text{CPU time}_{\text{2-way}}}{\text{CPU time}_{\text{1-way}}} = \frac{2.49 \times \text{Instruction count}}{2.47 \times \text{Instruction count}} = \frac{2.49}{2.47} = 1.01$$

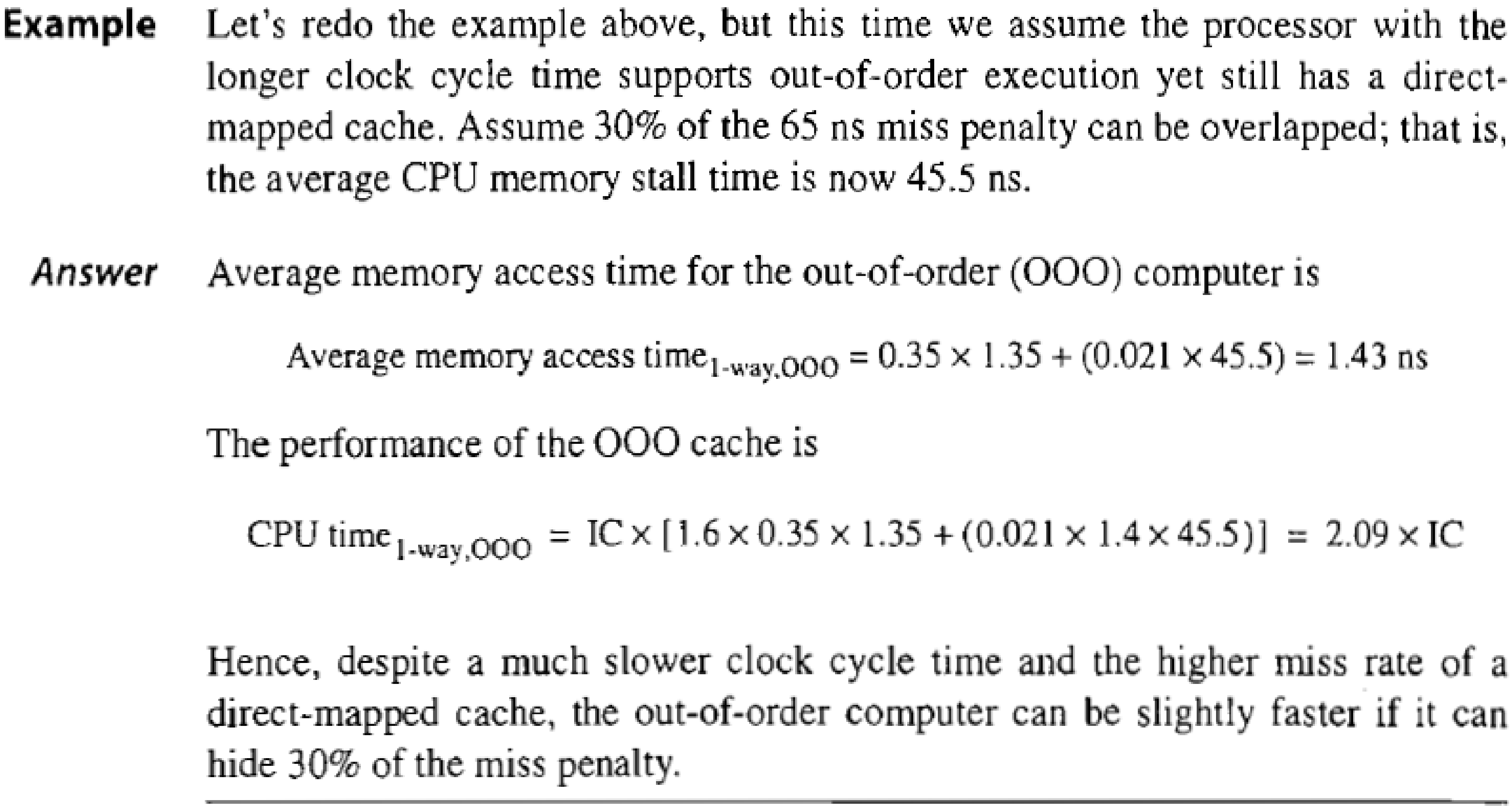
Miss penalty: out-of-order execution

- Out-or-order execution: how miss penalty is defined?
  - – Account for the full latency?
  - – Or just the “exposed” or “non-overlapped” latency?

- New definition of Miss Penalty for OOO

$$\frac{\textit{Memory stall cycles}}{\textit{Instruction}} = \frac{\textit{Misses}}{\textit{Instruction}} \textit{ x (Total miss latency – Overlapped miss latency)}$$

Ex B.21



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

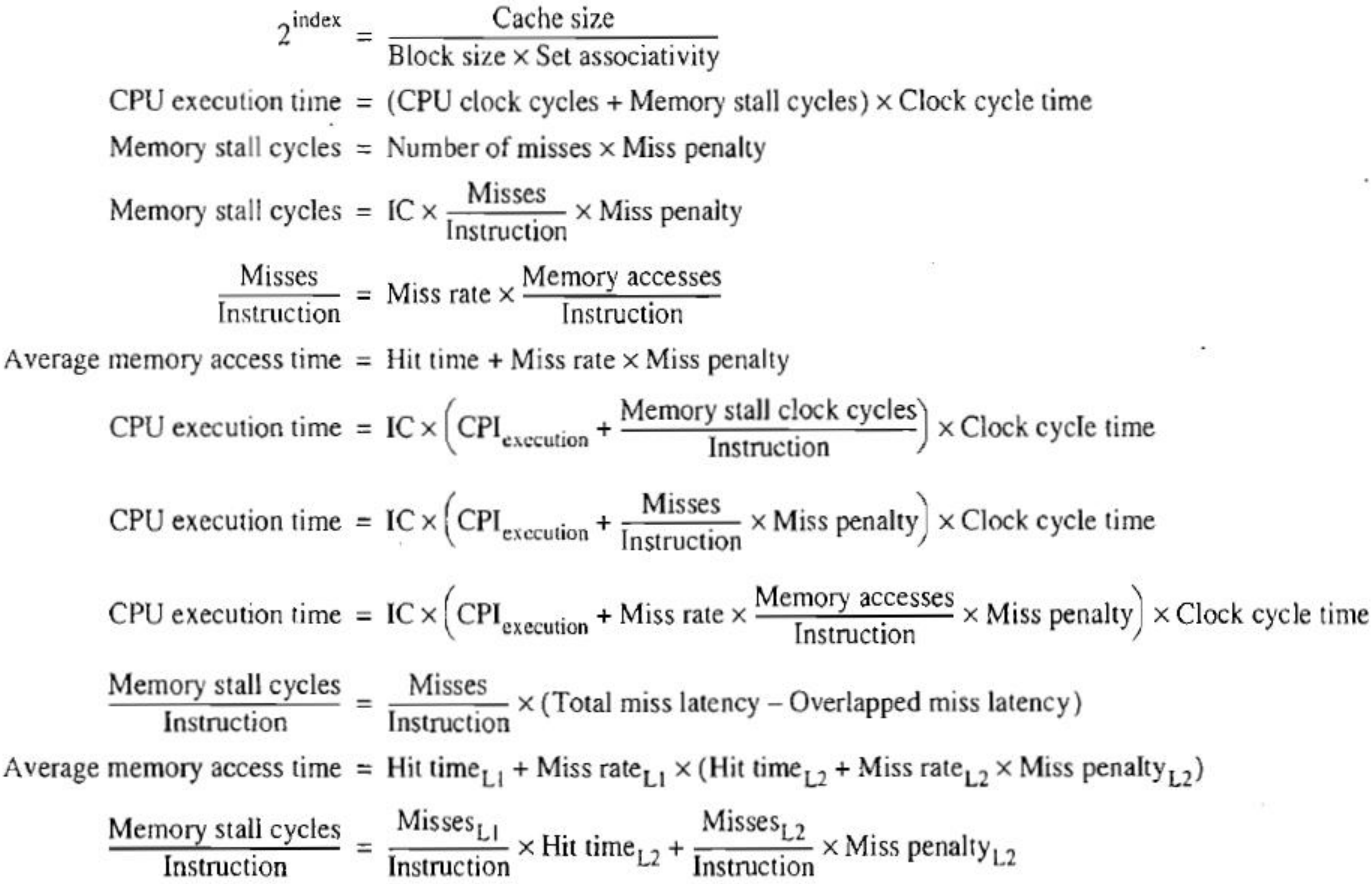
$$\text{Average memory access time}_{\text{1-way,OOO}} = 0.35 \times 1.35 + (0.021 \times 45.5) = 1.43 \text{ ns}$$

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.

Equações de desempenho: Resumo



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

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

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

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

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

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

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

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

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

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

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

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

B-3: Memory Hierarchy Basics

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

Para melhorar o desempenho da cache

1. Reduzir miss rate
2. Reduzir miss penalty

3. Reduzir *tempo de hit na cache*

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

Tipos de miss

- Compulsory
  - Cold-start or first-reference
- Capacity
  - (não cabe)
- Conflict
  - Colisão ou coerência

IC-UNICAMP

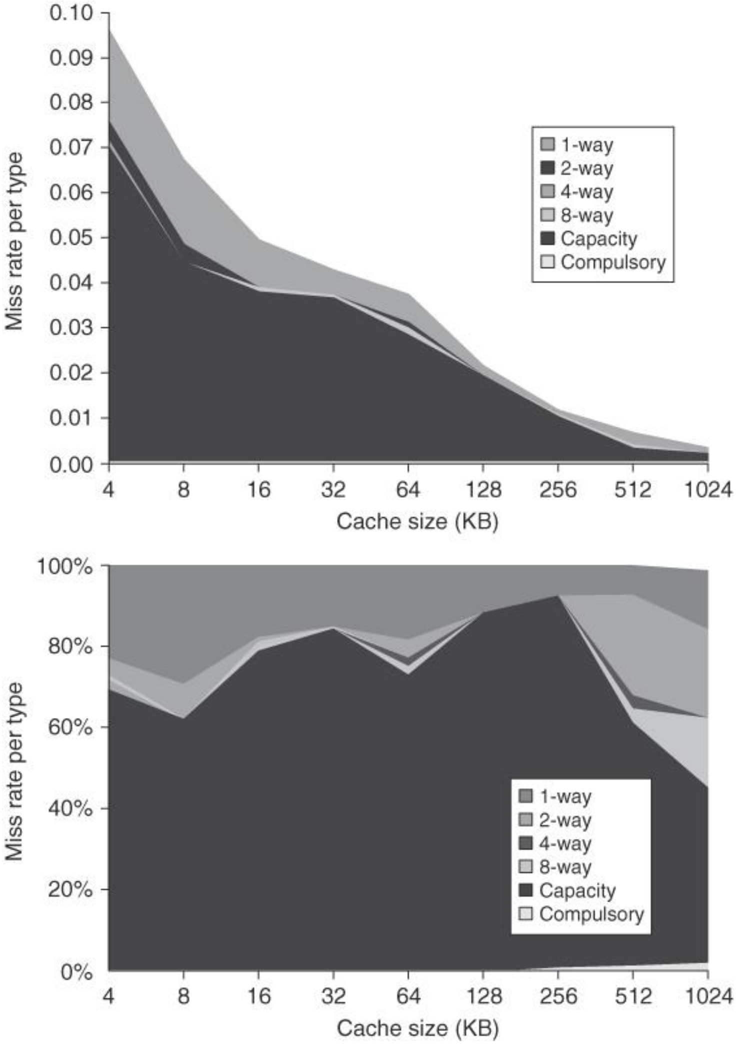


Figure B.9 Total miss rate (top) and distribution of miss rate (bottom) for each size cache according to the three C's for the data in Figure B.8. The top diagram shows the actual data cache miss rates, while the bottom diagram shows the percentage in each category. (Space allows the graphs to show one extra cache size than can fit in Figure B.8.)

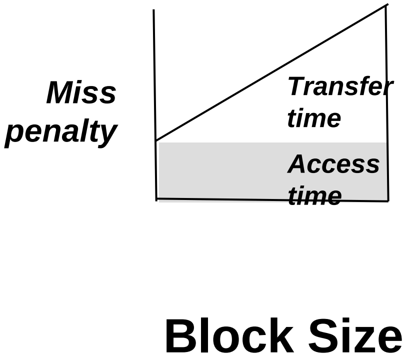
1- Larger block size

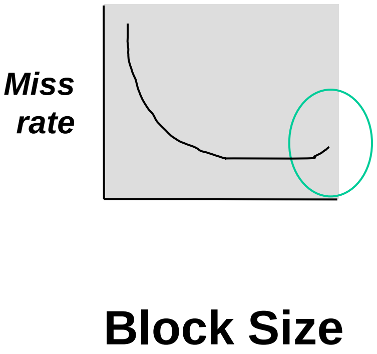
- Positive effect:
  - Reduces compulsory misses: prefetched data/instructions
- Negative effect :
  - Increases capacity misses: fewer blocks in cache
  - Increases conflict misses: fewer blocks → higher chance of conflict
  - Increases miss penalty: larger block → harder to handle

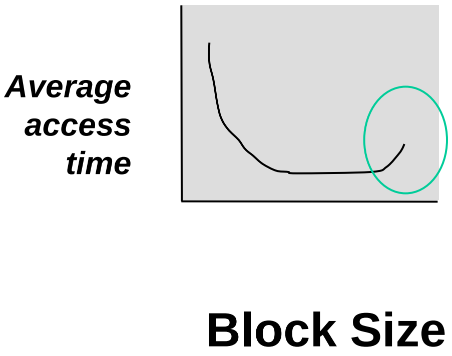
Miss Penalty e Miss Rate vs Tamanho do Bloco

- Maior tamanho do bloco →
  - → maior miss penalty
  - → menor miss rate

Miss Penalty × **Miss Rate** => **Avg. Memory Access Time**







Miss rate x block size

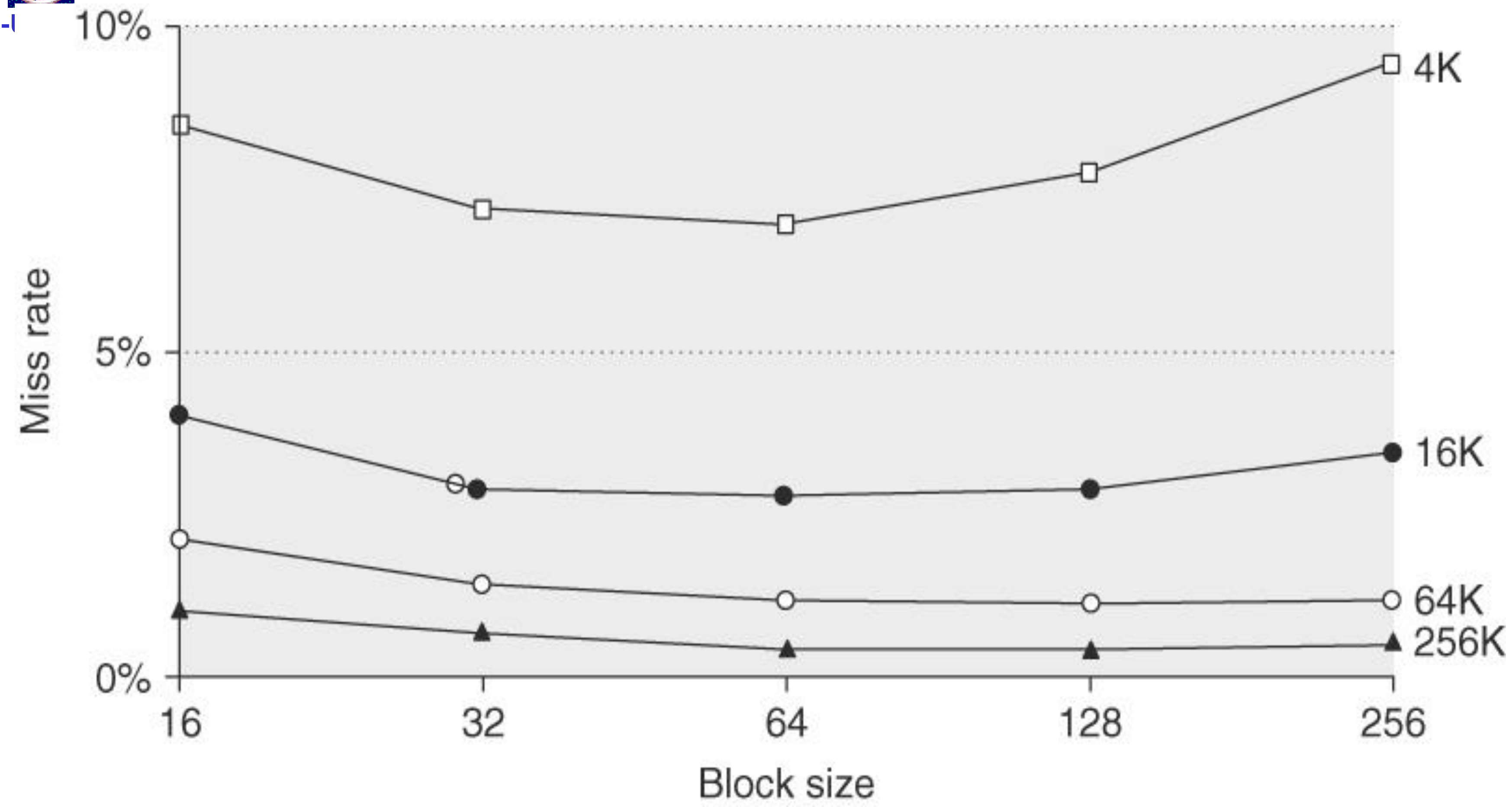
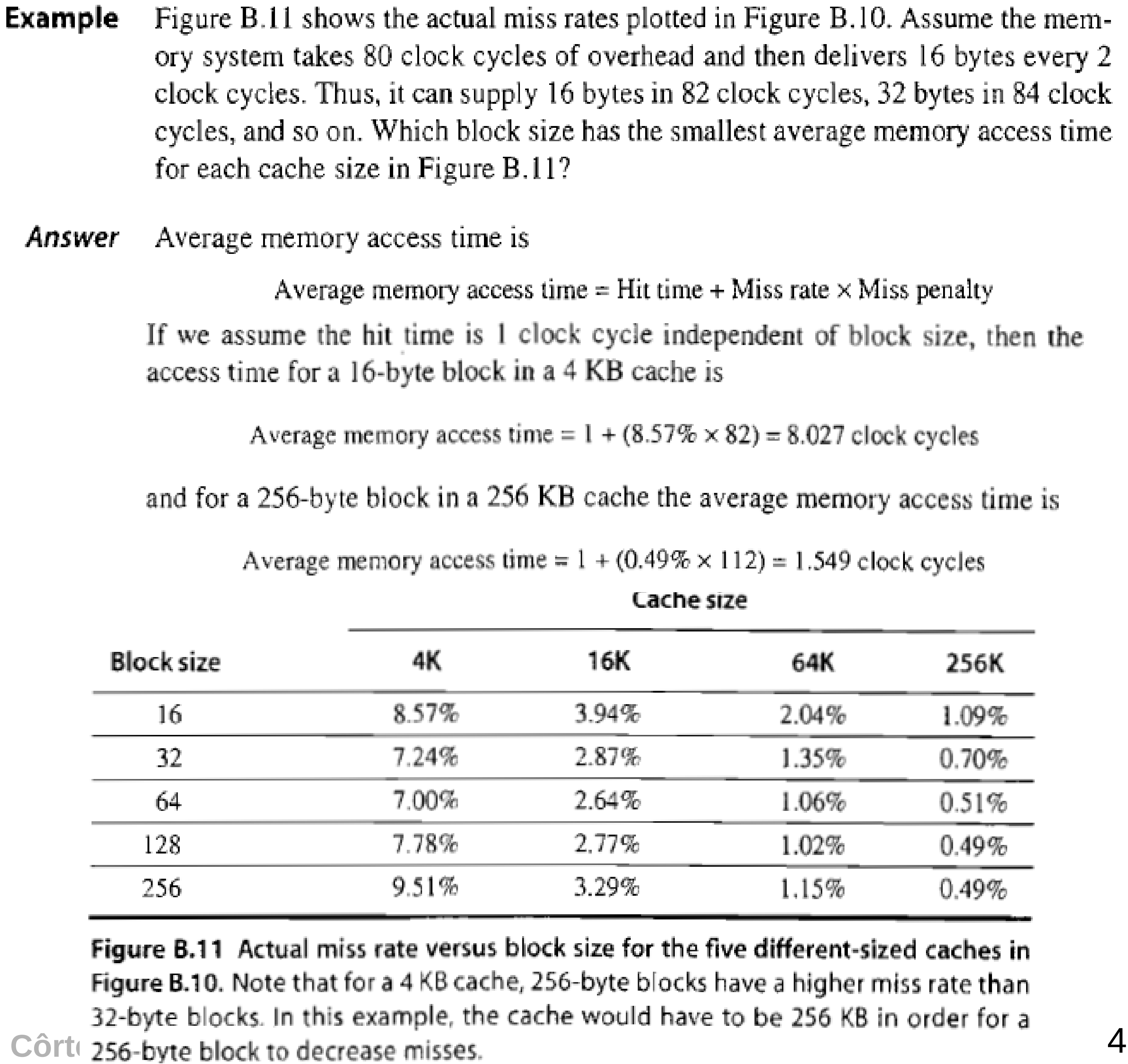


Figure B.10 Miss rate versus block size for five different-sized caches. Note that miss rate actually goes up if the block size is too large relative to the cache size. Each line represents a cache of different size. Figure B.11 shows the data used to plot these lines. Unfortunately, SPEC2000 traces would take too long if block size were included, so these data are based on SPEC92 on a DECstation 5000 [Gee et al. 1993].

IC-UNICAMP

Exmpl pag B-27: Miss rate e block size



Example Figure B.11 shows the actual miss rates plotted in Figure B.10. Assume the memory system takes 80 clock cycles of overhead and then delivers 16 bytes every 2 clock cycles. Thus, it can supply 16 bytes in 82 clock cycles, 32 bytes in 84 clock cycles, and so on. Which block size has the smallest average memory access time for each cache size in Figure B.11?

Answer Average memory access time is

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

If we assume the hit time is 1 clock cycle independent of block size, then the access time for a 16-byte block in a 4 KB cache is

$$\text{Average memory access time} = 1 + (8.57\% \times 82) = 8.027 \text{ clock cycles}$$

and for a 256-byte block in a 256 KB cache the average memory access time is

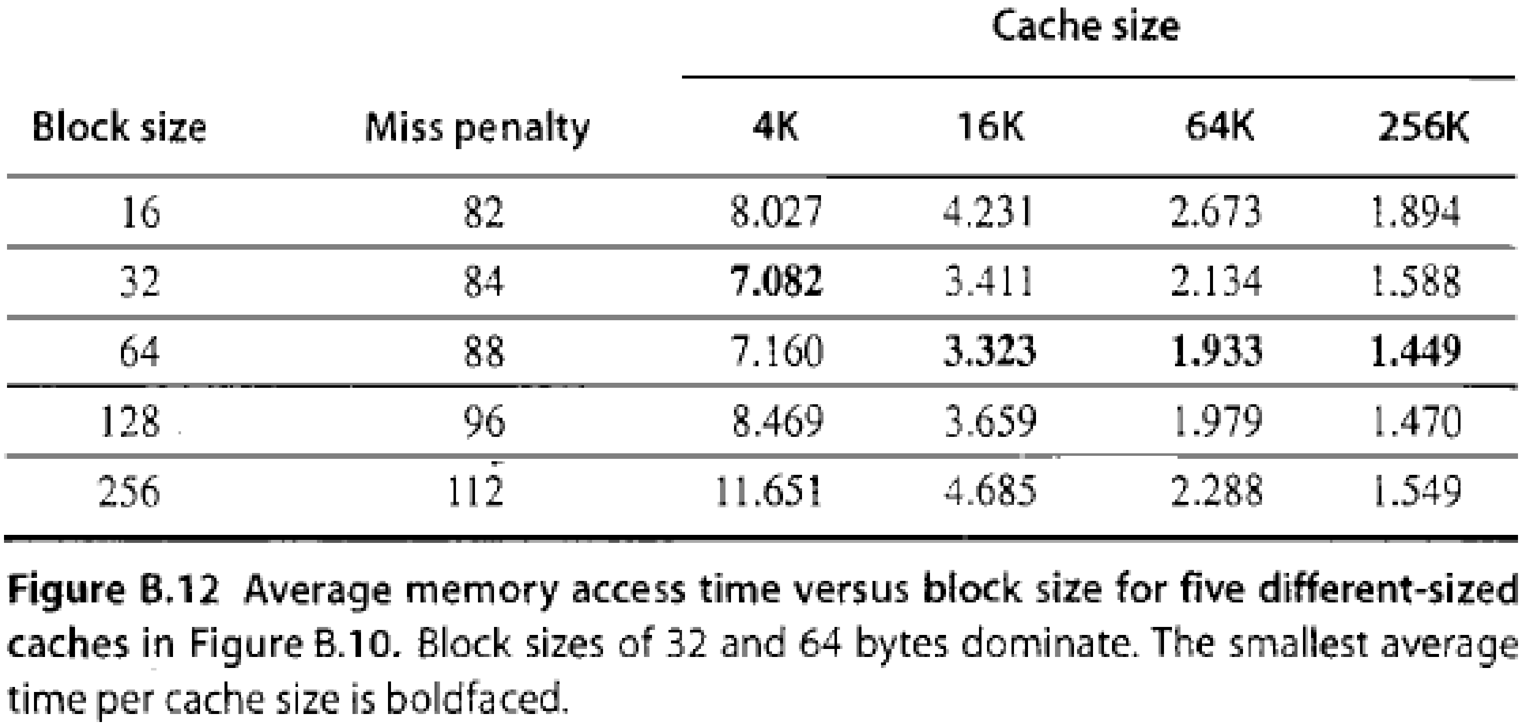
$$\text{Average memory access time} = 1 + (0.49\% \times 112) = 1.549 \text{ clock cycles}$$

	Cache size			
Block size	**4K**	**16K**	**64K**	**256K**
16	8.57%	3.94%	2.04%	1.09%
32	7.24%	2.87%	1.35%	0.70%
64	7.00%	2.64%	1.06%	0.51%
128	7.78%	2.77%	1.02%	0.49%
256	9.51%	3.29%	1.15%	0.49%

Figure B.11 Actual miss rate versus block size for the five different-sized caches in Figure B.10. Note that for a 4 KB cache, 256-byte blocks have a higher miss rate than 32-byte blocks. In this example, the cache would have to be 256 KB in order for a 256-byte block to decrease misses.

IC-UNICAMP

Exmpl p B-27: Miss rate e block size (2)



		Cache size			
Block size	Miss penalty	4K	16K	64K	256K
16	82	8.027	4.231	2.673	1.894
32	84	**7.082**	3.411	2.134	1.588
64	88	7.160	**3.323**	**1.933**	**1.449**
128	96	8.469	3.659	1.979	1.470
256	112	11.651	4.685	2.288	1.549

Figure B.12 Average memory access time versus block size for five different-sized caches in Figure B.10. Block sizes of 32 and 64 bytes dominate. The smallest average time per cache size is boldfaced.

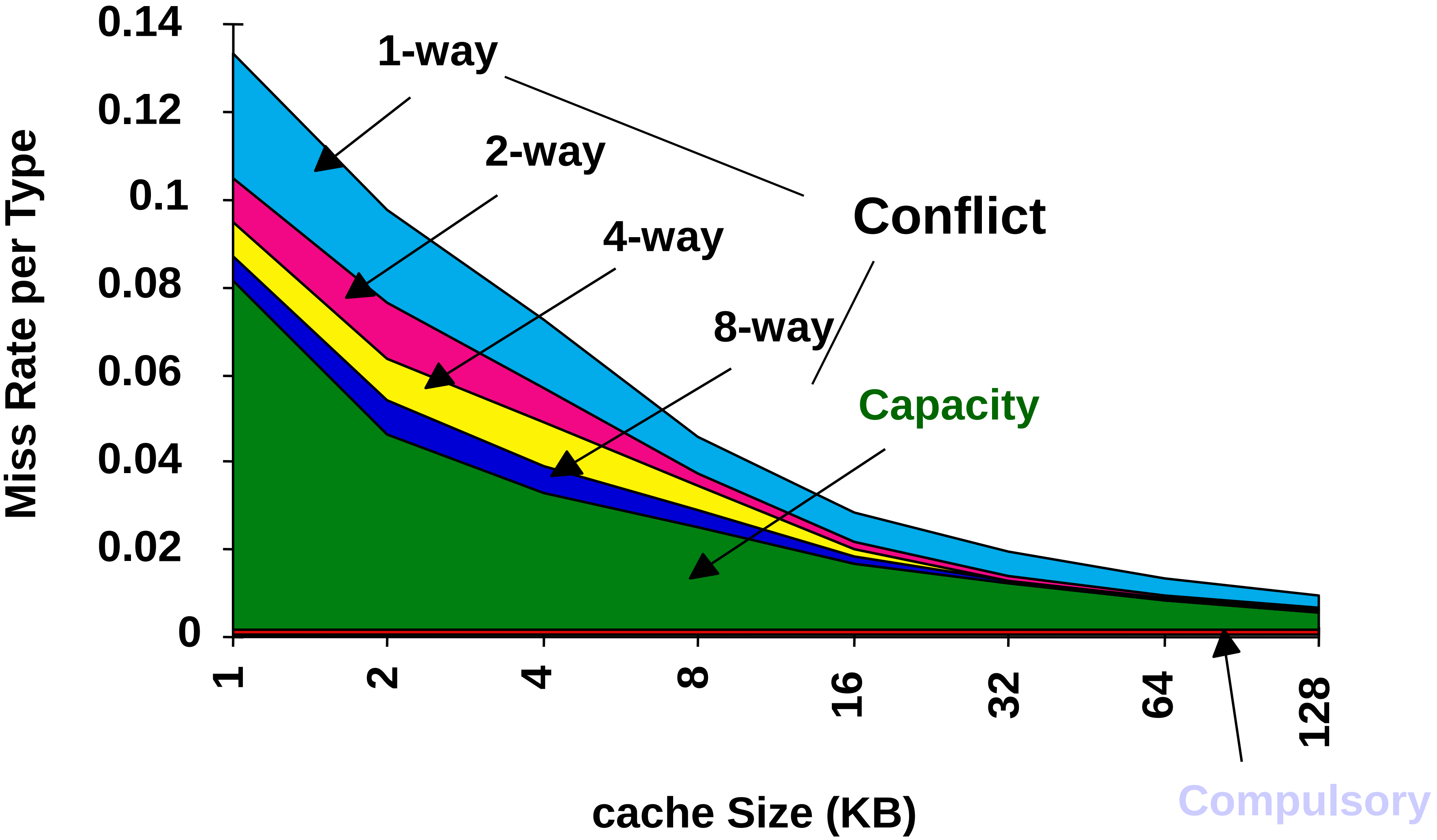
2- Larger total cache capacity to reduce miss rate

- Positive effect:
  - Reduces capacity misses
- Negative effect :
  - Increases hit time: large → slower
  - Increases power consumption

3- Higher associativity

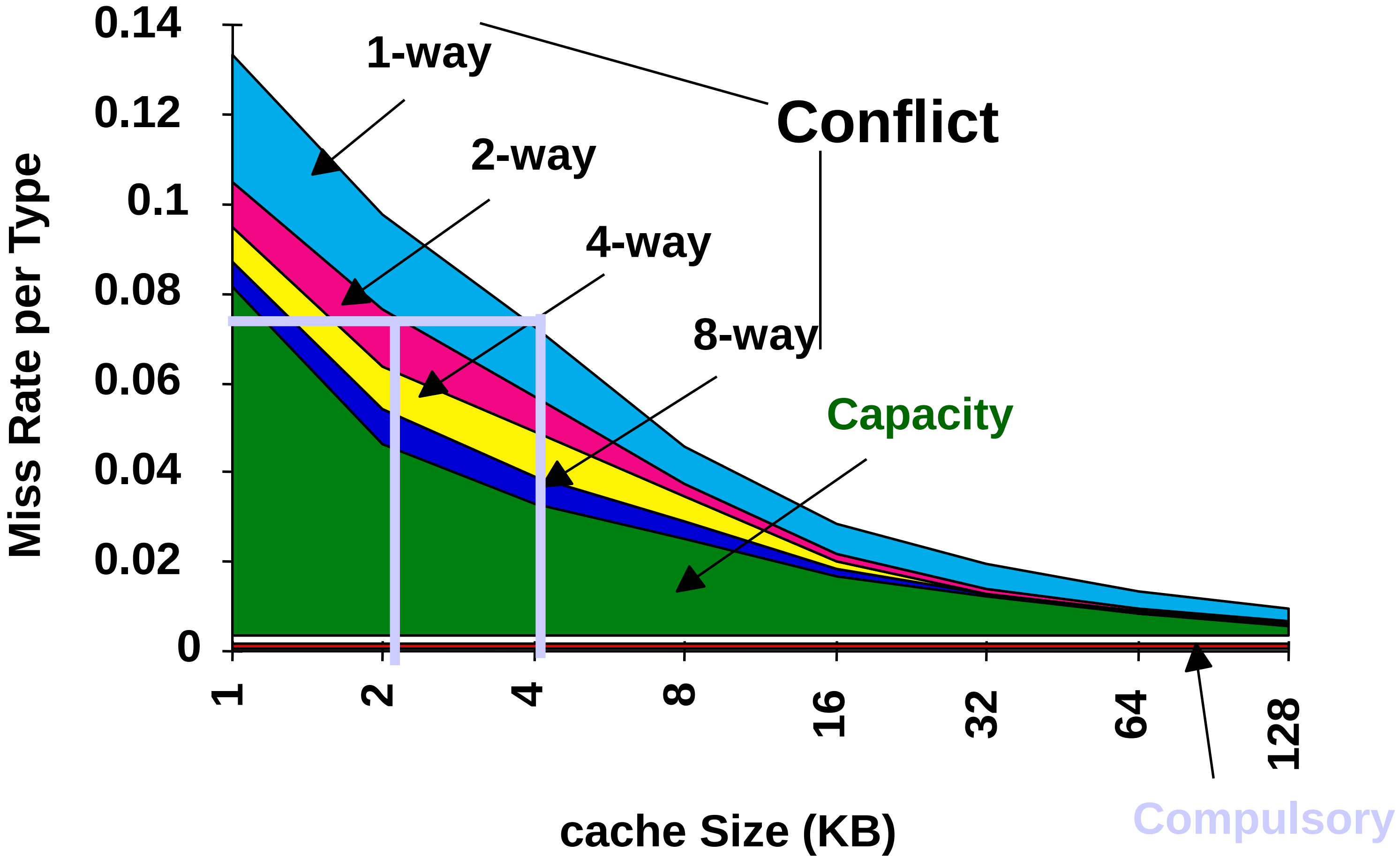
- Positive effect
  - Reduces conflict misses: possible to have k potentially conflicting copies present in a k-associative cache
- Negative effects:
  - Increases hit time: mux in the way, impossible to speculate (use data before hit/miss result)
  - Increases power consumption: a given index “activates” all blocks in that set

3Cs - Miss Rate Absoluto (SPEC92)



cache Misses

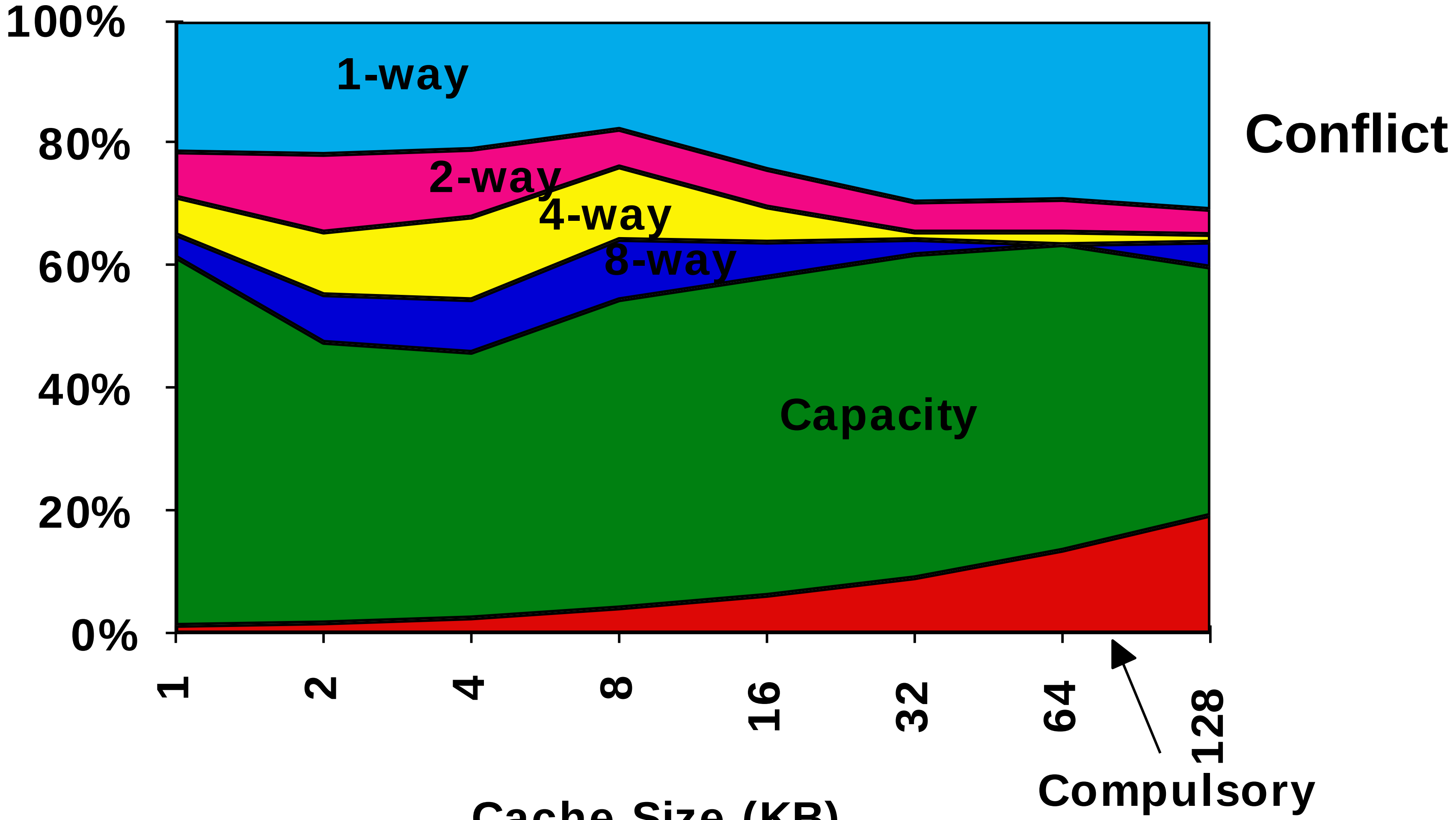
miss rate 1-way associative cache size X
= miss rate 2-way associative cache size X/2



3Cs Miss Rate Relativo

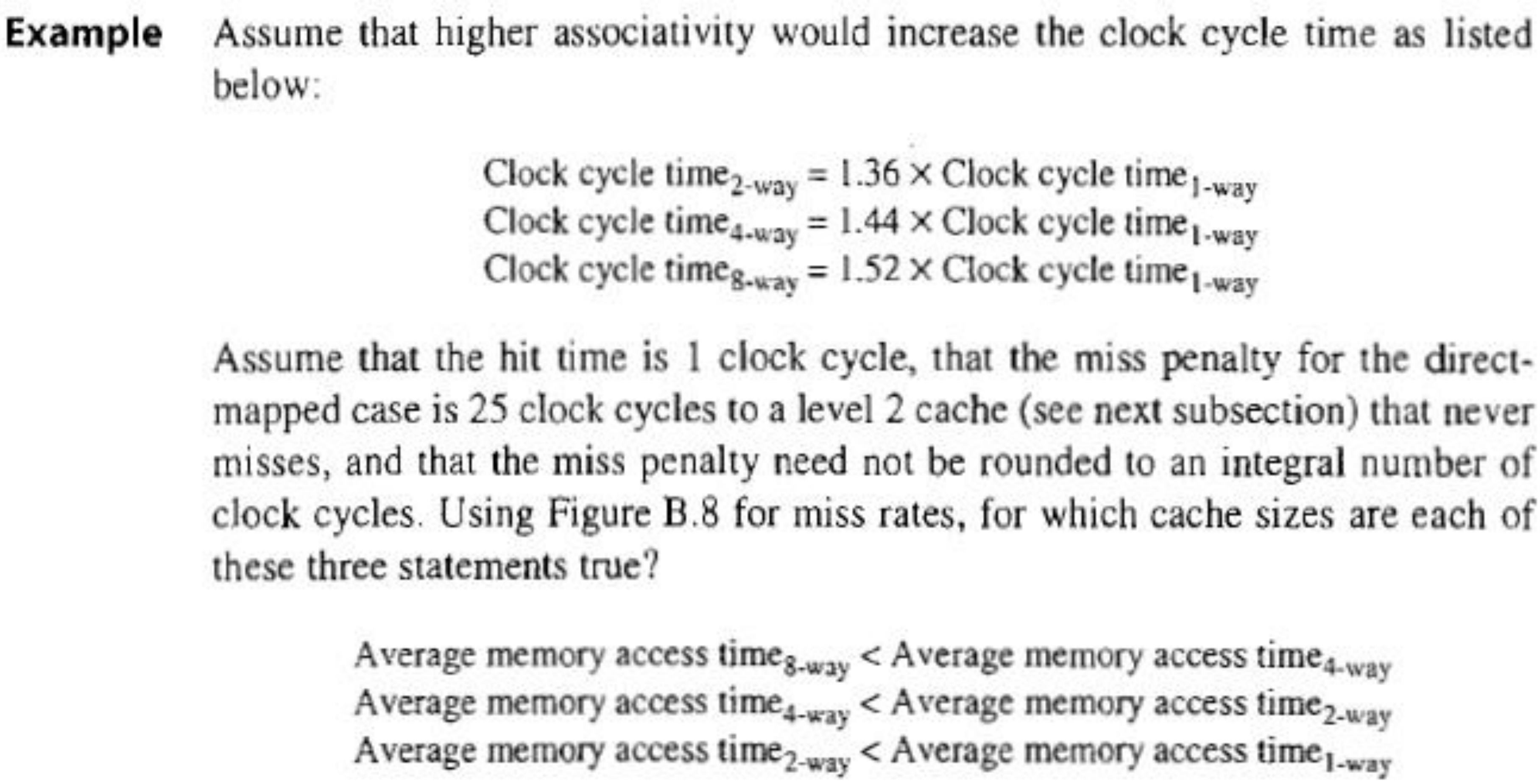
Flaws: for fixed block size

Good: insight => invention



IC-UNICAMP

Exmpl B-29: 3rd optimization (associativity)



Example Assume that higher associativity would increase the clock cycle time as listed below:

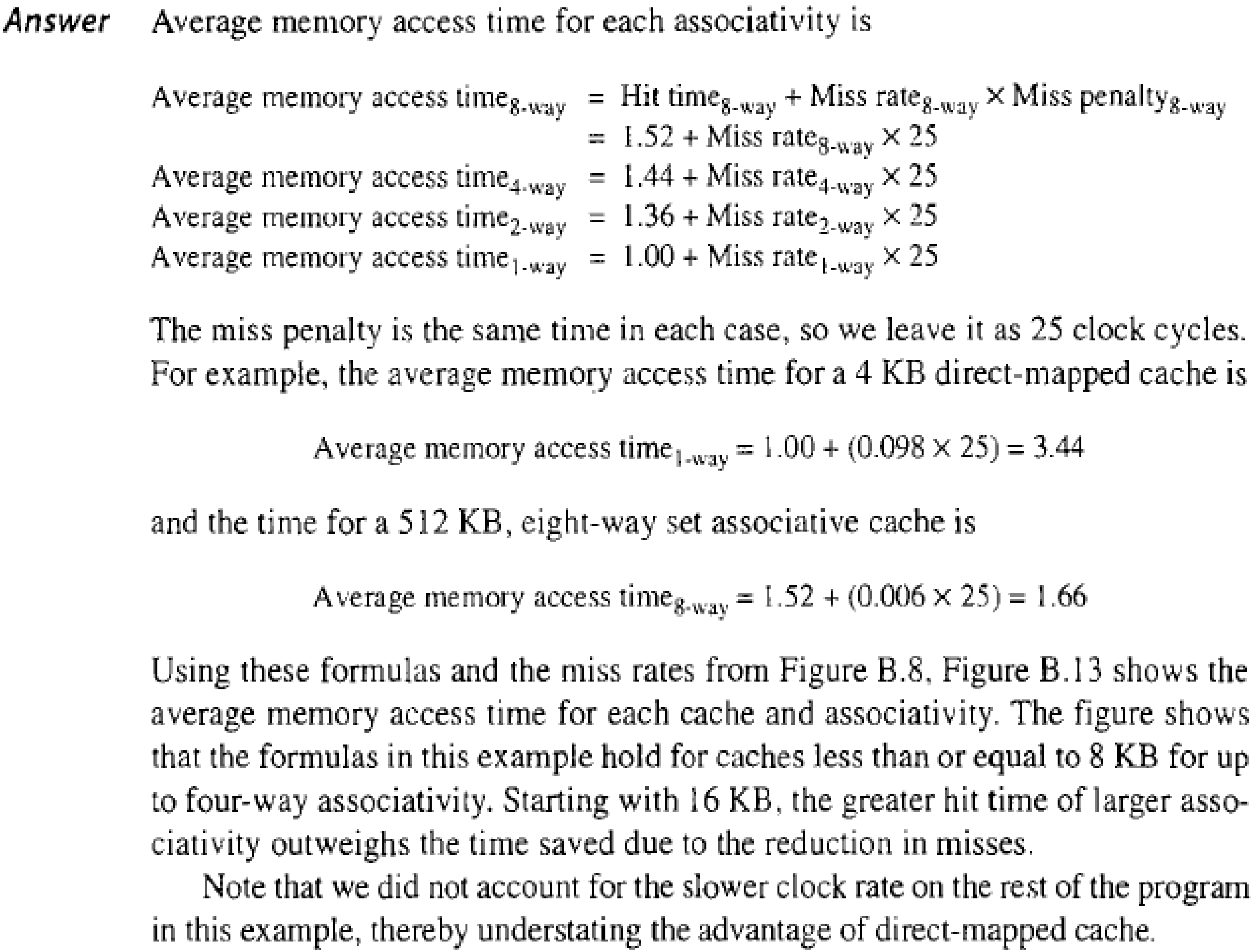
$$\text{Clock cycle time}_{2\text{-way}} = 1.36 \times \text{Clock cycle time}_{1\text{-way}}$$
$$\text{Clock cycle time}_{4\text{-way}} = 1.44 \times \text{Clock cycle time}_{1\text{-way}}$$
$$\text{Clock cycle time}_{8\text{-way}} = 1.52 \times \text{Clock cycle time}_{1\text{-way}}$$

Assume that the hit time is 1 clock cycle, that the miss penalty for the direct-mapped case is 25 clock cycles to a level 2 cache (see next subsection) that never misses, and that the miss penalty need not be rounded to an integral number of clock cycles. Using Figure B.8 for miss rates, for which cache sizes are each of these three statements true?

$$\text{Average memory access time}_{8\text{-way}} < \text{Average memory access time}_{4\text{-way}}$$
$$\text{Average memory access time}_{4\text{-way}} < \text{Average memory access time}_{2\text{-way}}$$
$$\text{Average memory access time}_{2\text{-way}} < \text{Average memory access time}_{1\text{-way}}$$

IC-UNICAMP

Exmpl B-29: 3rd optimization (cont)



Answer Average memory access time for each associativity is

$$\begin{aligned}
\text{Average memory access time}_{\text{8-way}} &= \text{Hit time}_{\text{8-way}} + \text{Miss rate}_{\text{8-way}} \times \text{Miss penalty}_{\text{8-way}} \\
&= 1.52 + \text{Miss rate}_{\text{8-way}} \times 25 \\
\text{Average memory access time}_{\text{4-way}} &= 1.44 + \text{Miss rate}_{\text{4-way}} \times 25 \\
\text{Average memory access time}_{\text{2-way}} &= 1.36 + \text{Miss rate}_{\text{2-way}} \times 25 \\
\text{Average memory access time}_{\text{1-way}} &= 1.00 + \text{Miss rate}_{\text{1-way}} \times 25
\end{aligned}$$

The miss penalty is the same time in each case, so we leave it as 25 clock cycles. For example, the average memory access time for a 4 KB direct-mapped cache is

$$\text{Average memory access time}_{\text{1-way}} = 1.00 + (0.098 \times 25) = 3.44$$

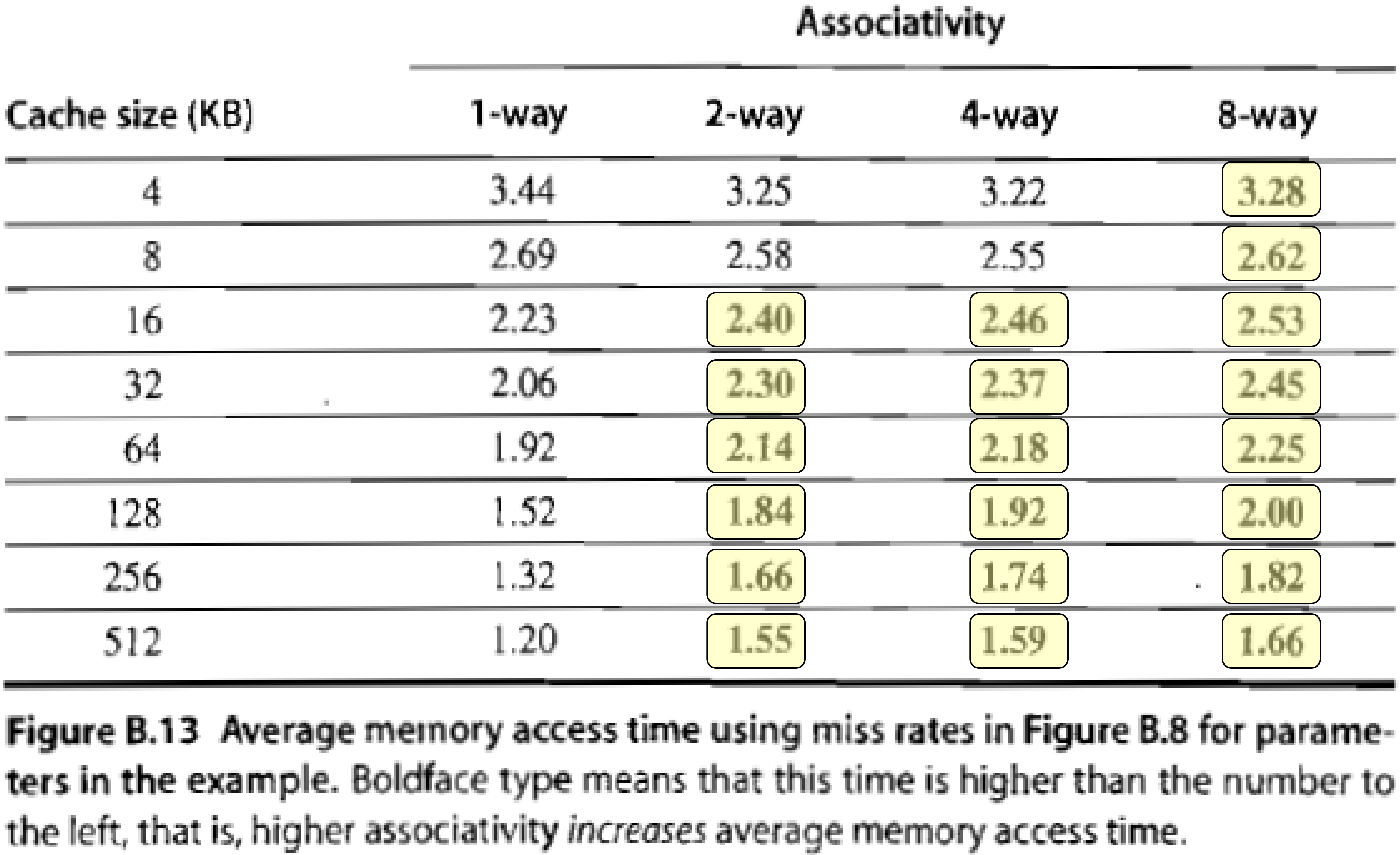
and the time for a 512 KB, eight-way set associative cache is

$$\text{Average memory access time}_{\text{8-way}} = 1.52 + (0.006 \times 25) = 1.66$$

Using these formulas and the miss rates from Figure B.8, Figure B.13 shows the average memory access time for each cache and associativity. The figure shows that the formulas in this example hold for caches less than or equal to 8 KB for up to four-way associativity. Starting with 16 KB, the greater hit time of larger associativity outweighs the time saved due to the reduction in misses.

Note that we did not account for the slower clock rate on the rest of the program in this example, thereby understating the advantage of direct-mapped cache.

Exmpl B-29: 3rd optimization (cont)



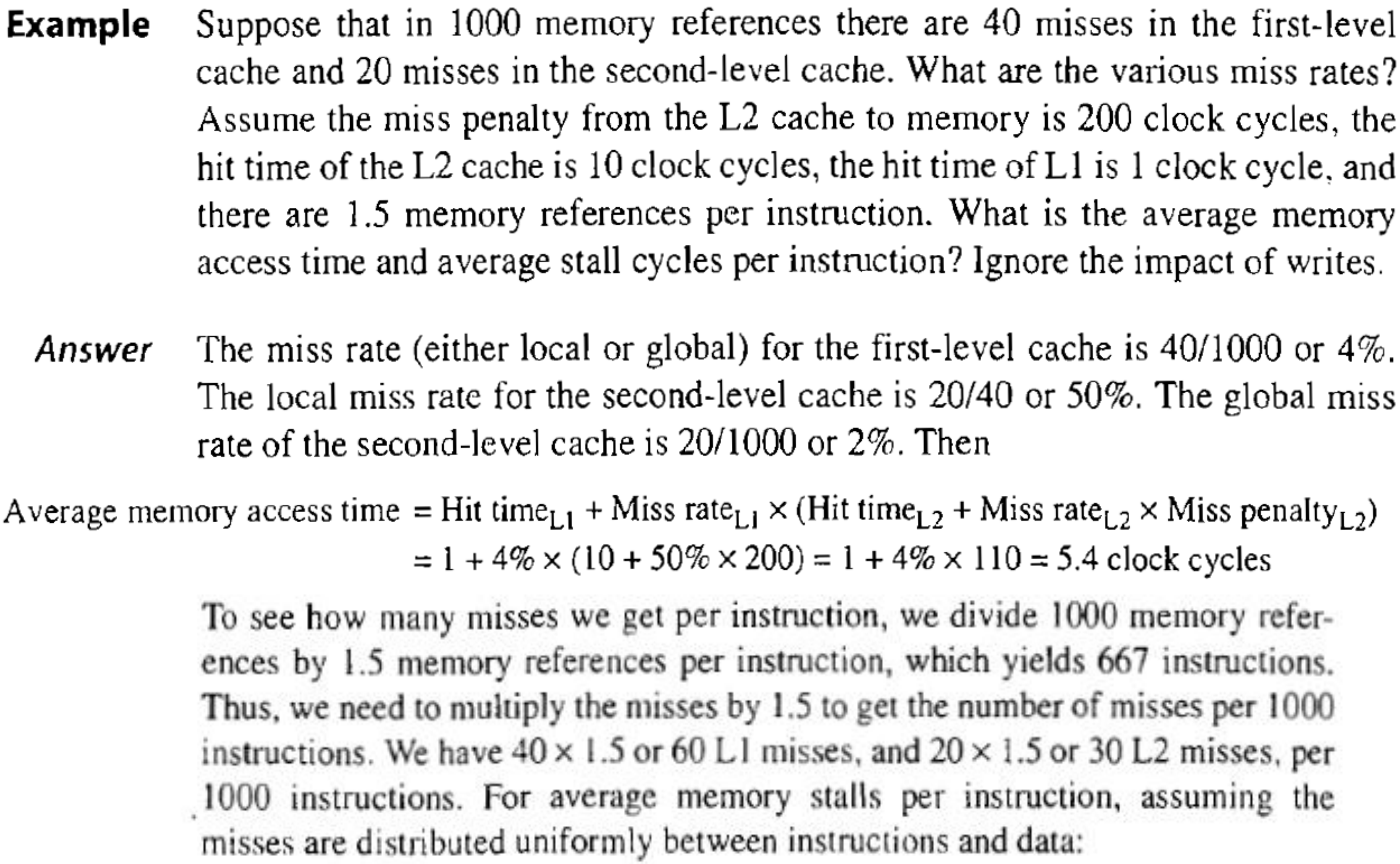
	Associativity			
Cache size (KB)	1-way	2-way	4-way	8-way
4	3.44	3.25	3.22	3.28
8	2.69	2.58	2.55	2.62
16	2.23	2.40	2.46	2.53
32	2.06	2.30	2.37	2.45
64	1.92	2.14	2.18	2.25
128	1.52	1.84	1.92	2.00
256	1.32	1.66	1.74	1.82
512	1.20	1.55	1.59	1.66

Figure B.13 Average memory access time using miss rates in Figure B.8 for parameters in the example. Boldface type means that this time is higher than the number to the left, that is, higher associativity *increases* average memory access time.

4- Higher number of cache levels

- Reduces overall memory access time (supor 2 níveis)
- AMAT = Hit time$_{L1}$ + Miss Rate$_{L1}$ x Miss Penalty$_{L1}$
- Miss Penalty$_{L1}$ = Hit time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$
- AMAT = Hit time$_{L1}$ + Miss Rate$_{L1}$ x
  (Hit time$_{L2}$ + Miss Rate$_{L2}$ x Miss Penalty$_{L2}$)
- Definições
  - Local Miss Rate = total de misses nesta cache / total de **acessos a esta cache**
    - para L1 = MissRate$_{L1}$ e para L2 = MissRate$_{L2}$
  - Global Miss Rate = total de misses nesta cache / total de **acessos de memória**
    - para L1 = MissRate$_{L1}$ e para L2 = MissRate$_{L1}$ x MissRate$_{L2}$
  - Ave Mem Stalls per Instruction = (Misses/Instruction$_{L1}$) x (HitTime$_{L2}$) + (Misses/Instruction$_{L2}$) x (MissPenalty$_{L2}$)

Exmpl B-31: multilevel caches



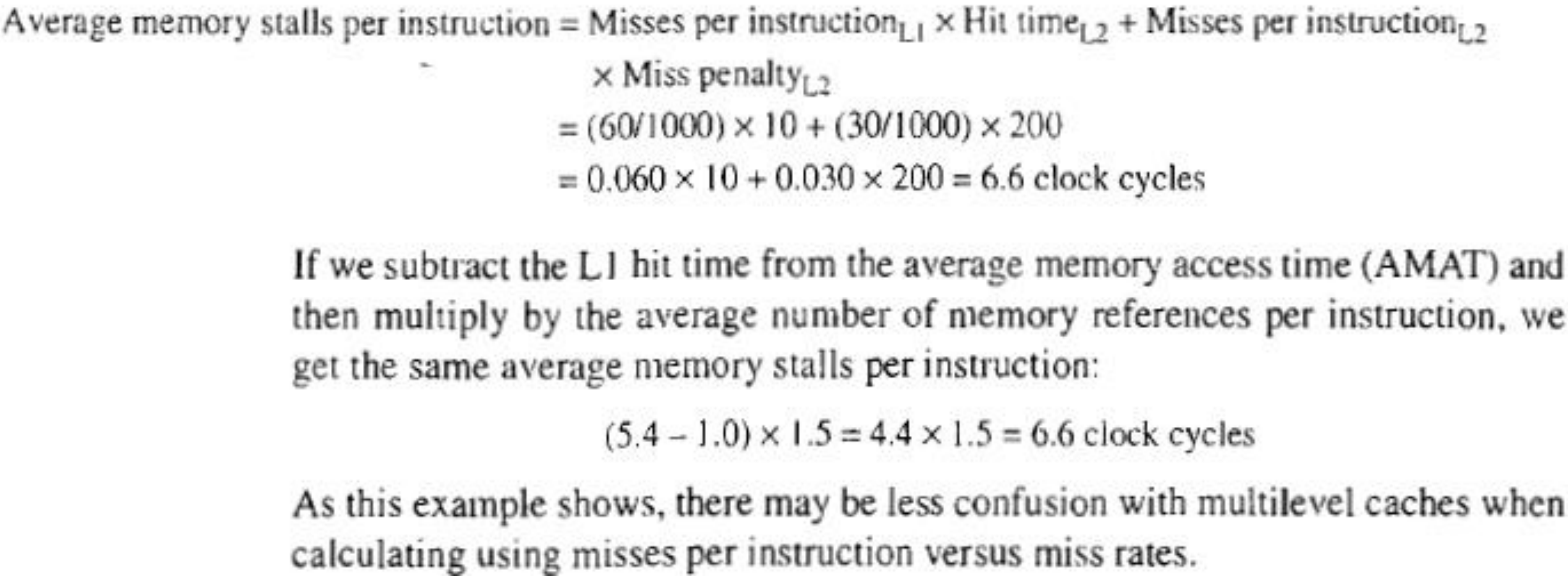
Example Suppose that in 1000 memory references there are 40 misses in the first-level cache and 20 misses in the second-level cache. What are the various miss rates? Assume the miss penalty from the L2 cache to memory is 200 clock cycles, the hit time of the L2 cache is 10 clock cycles, the hit time of L1 is 1 clock cycle, and there are 1.5 memory references per instruction. What is the average memory access time and average stall cycles per instruction? Ignore the impact of writes.

Answer The miss rate (either local or global) for the first-level cache is 40/1000 or 4%. The local miss rate for the second-level cache is 20/40 or 50%. The global miss rate of the second-level cache is 20/1000 or 2%. Then

$$\begin{aligned}\text{Average memory access time} &= \text{Hit time}_{L1} + \text{Miss rate}_{L1} \times (\text{Hit time}_{L2} + \text{Miss rate}_{L2} \times \text{Miss penalty}_{L2})\\ &= 1 + 4\% \times (10 + 50\% \times 200) = 1 + 4\% \times 110 = 5.4 \text{ clock cycles}\end{aligned}$$

To see how many misses we get per instruction, we divide 1000 memory references by 1.5 memory references per instruction, which yields 667 instructions. Thus, we need to multiply the misses by 1.5 to get the number of misses per 1000 instructions. We have 40 × 1.5 or 60 L1 misses, and 20 × 1.5 or 30 L2 misses, per 1000 instructions. For average memory stalls per instruction, assuming the misses are distributed uniformly between instructions and data:

Exmpl B-31: multilevel caches



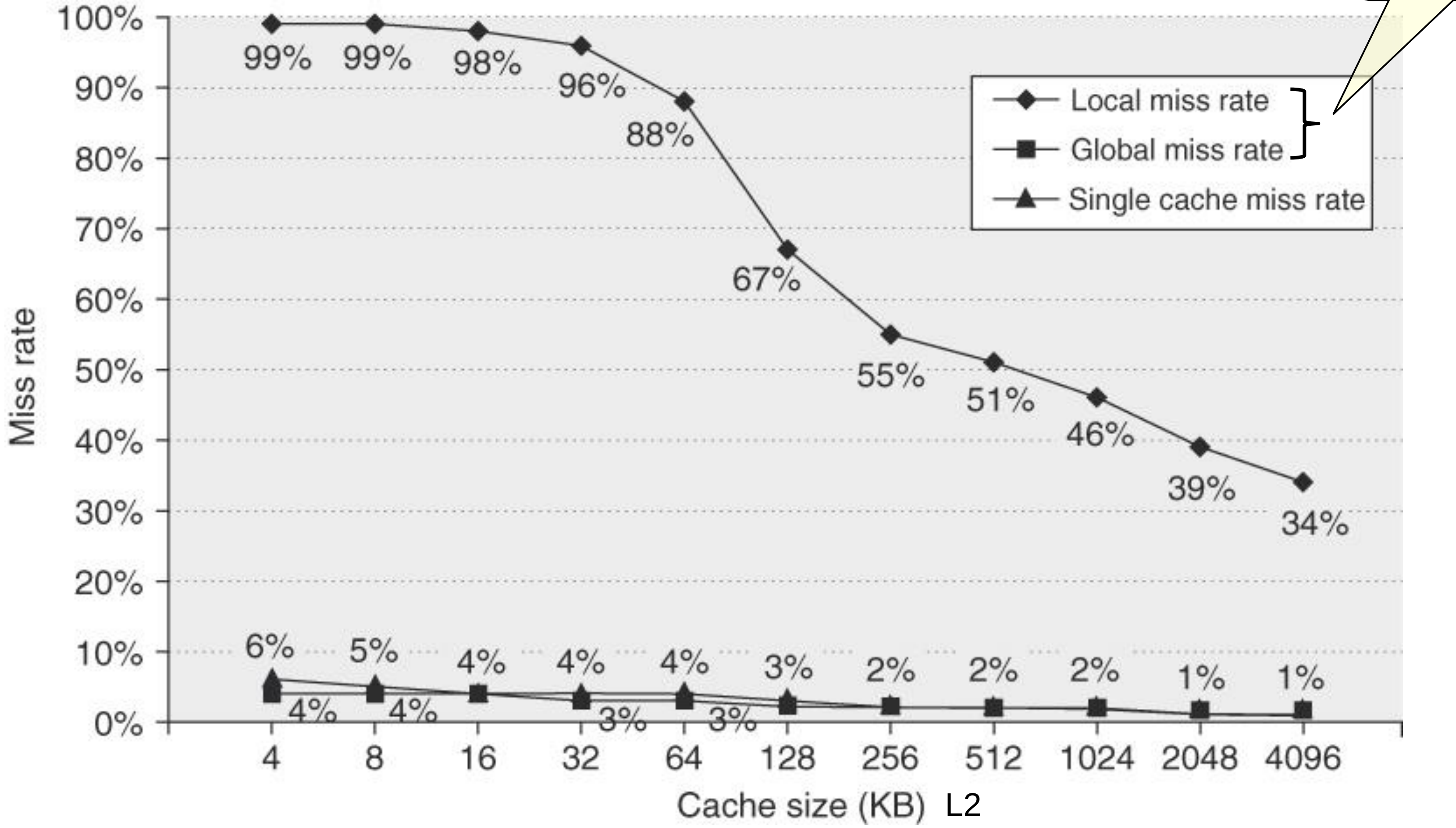
$$\begin{aligned}\text{Average memory stalls per instruction} &= \text{Misses per instruction}_{L1} \times \text{Hit time}_{L2} + \text{Misses per instruction}_{L2} \\ &\quad \times \text{Miss penalty}_{L2} \\ &= (60/1000) \times 10 + (30/1000) \times 200 \\ &= 0.060 \times 10 + 0.030 \times 200 = 6.6 \text{ clock cycles}\end{aligned}$$

If we subtract the L1 hit time from the average memory access time (AMAT) and then multiply by the average number of memory references per instruction, we get the same average memory stalls per instruction:

$$(5.4 - 1.0) \times 1.5 = 4.4 \times 1.5 = 6.6 \text{ clock cycles}$$

As this example shows, there may be less confusion with multilevel caches when calculating using misses per instruction versus miss rates.

Miss rate em cache (single e L2)



Of L2, with Split L1: 32kB + 32kB

Figure B.14 Miss rates versus cache size for multilevel caches. Second-level caches *smaller* than the sum of the two 64 KB first-level caches make little sense, as reflected in the high miss rates. After 256 KB the single cache is within 10% of the global miss rates. The miss rate of a single-level cache versus size is plotted against the local miss rate and global miss rate of a second-level cache using a 32 KB first-level cache. The L2 caches (unified) were two-way set associative with replacement. Each had split L1 instruction and data caches that were 64 KB two-way set associative with LRU replacement. The block size for both L1 and L2 caches was 64 bytes. Data were collected as in Figure B.4

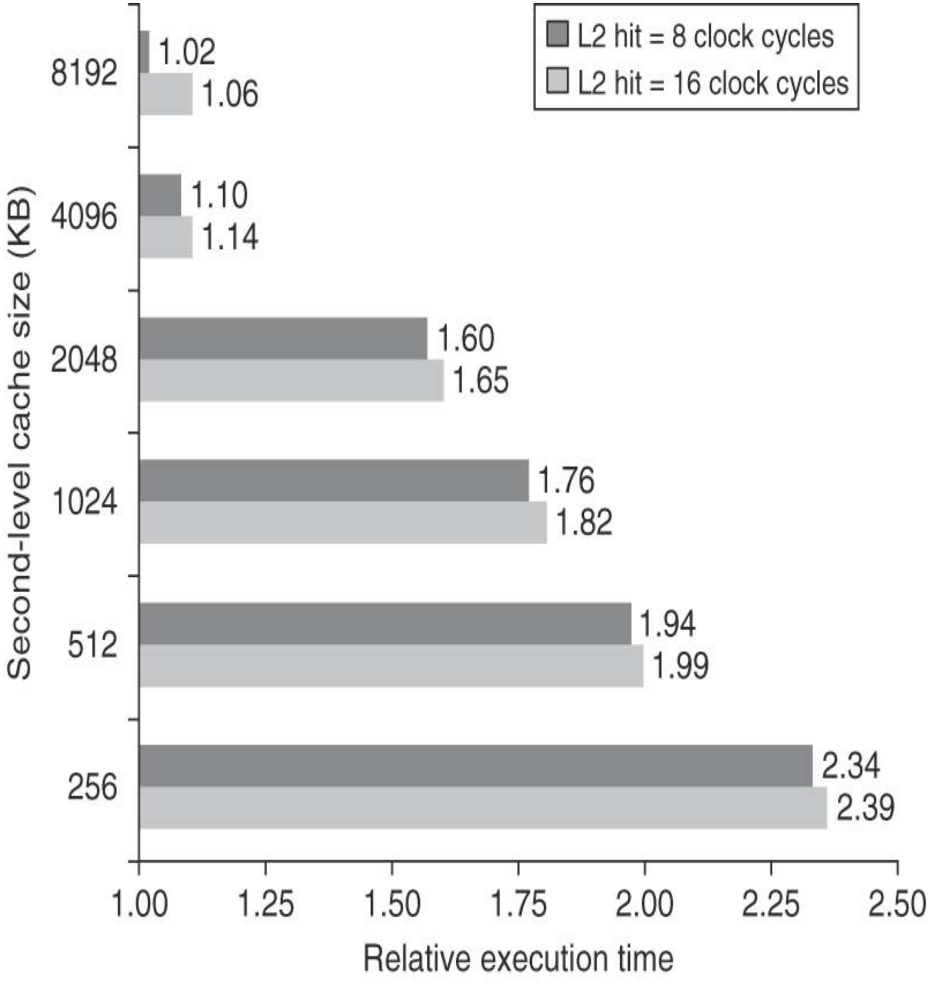
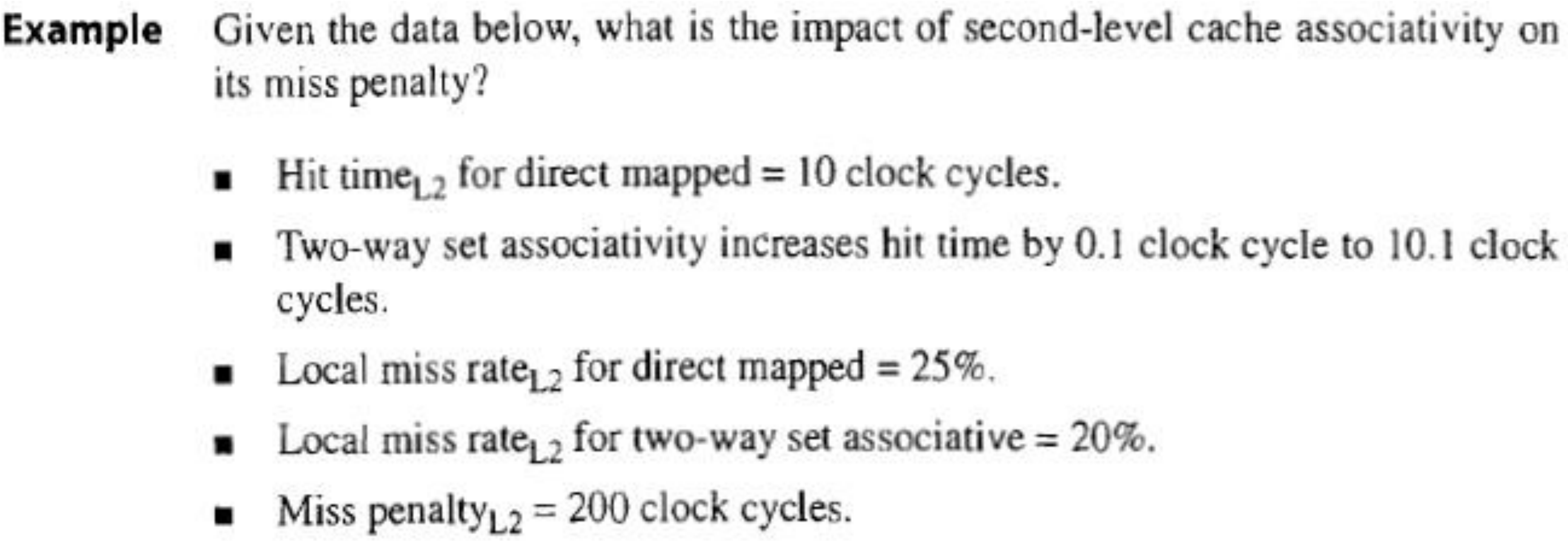


Figure B.15 Relative execution time by second-level cache size. The two bars are for different clock cycles for an L2 cache hit. The reference execution time of 1.00 is for an 8192 KB second-level cache with a 1-clock-cycle latency on a second-level hit. These data were collected the same way as in Figure B.14, using a simulator to imitate the Alpha 21264.

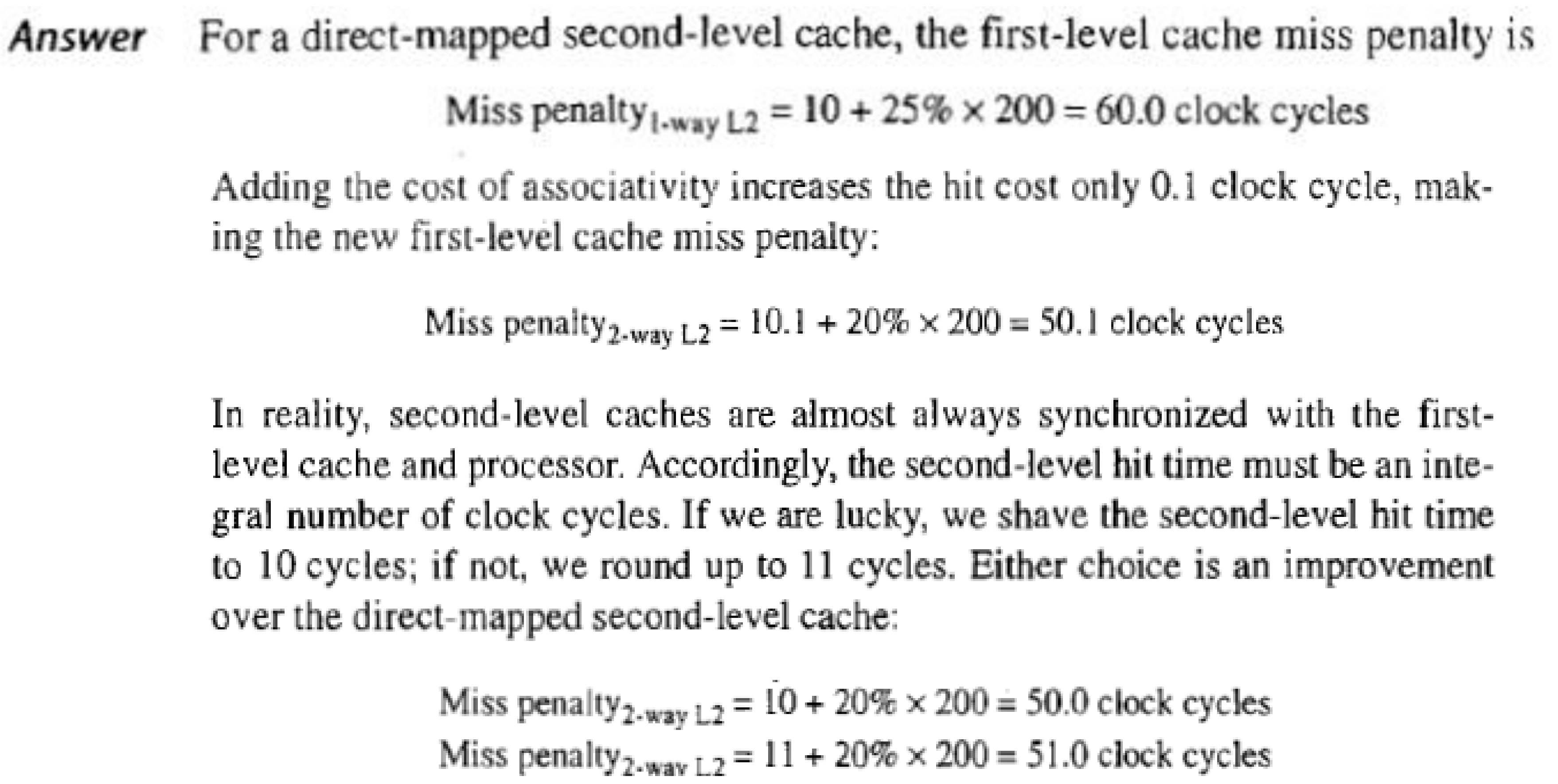
Exmpl B-33: associativity in L2?



Example Given the data below, what is the impact of second-level cache associativity on its miss penalty?

- Hit time_{L2} for direct mapped = 10 clock cycles.
- Two-way set associativity increases hit time by 0.1 clock cycle to 10.1 clock cycles.
- Local miss rate_{L2} for direct mapped = 25%.
- Local miss rate_{L2} for two-way set associative = 20%.
- Miss penalty_{L2} = 200 clock cycles.

Exmpl B-33: multilevel caches (cont)



Answer For a direct-mapped second-level cache, the first-level cache miss penalty is

$$\text{Miss penalty}_{\text{1-way L2}} = 10 + 25\% \times 200 = 60.0 \text{ clock cycles}$$

Adding the cost of associativity increases the hit cost only 0.1 clock cycle, making the new first-level cache miss penalty:

$$\text{Miss penalty}_{\text{2-way L2}} = 10.1 + 20\% \times 200 = 50.1 \text{ clock cycles}$$

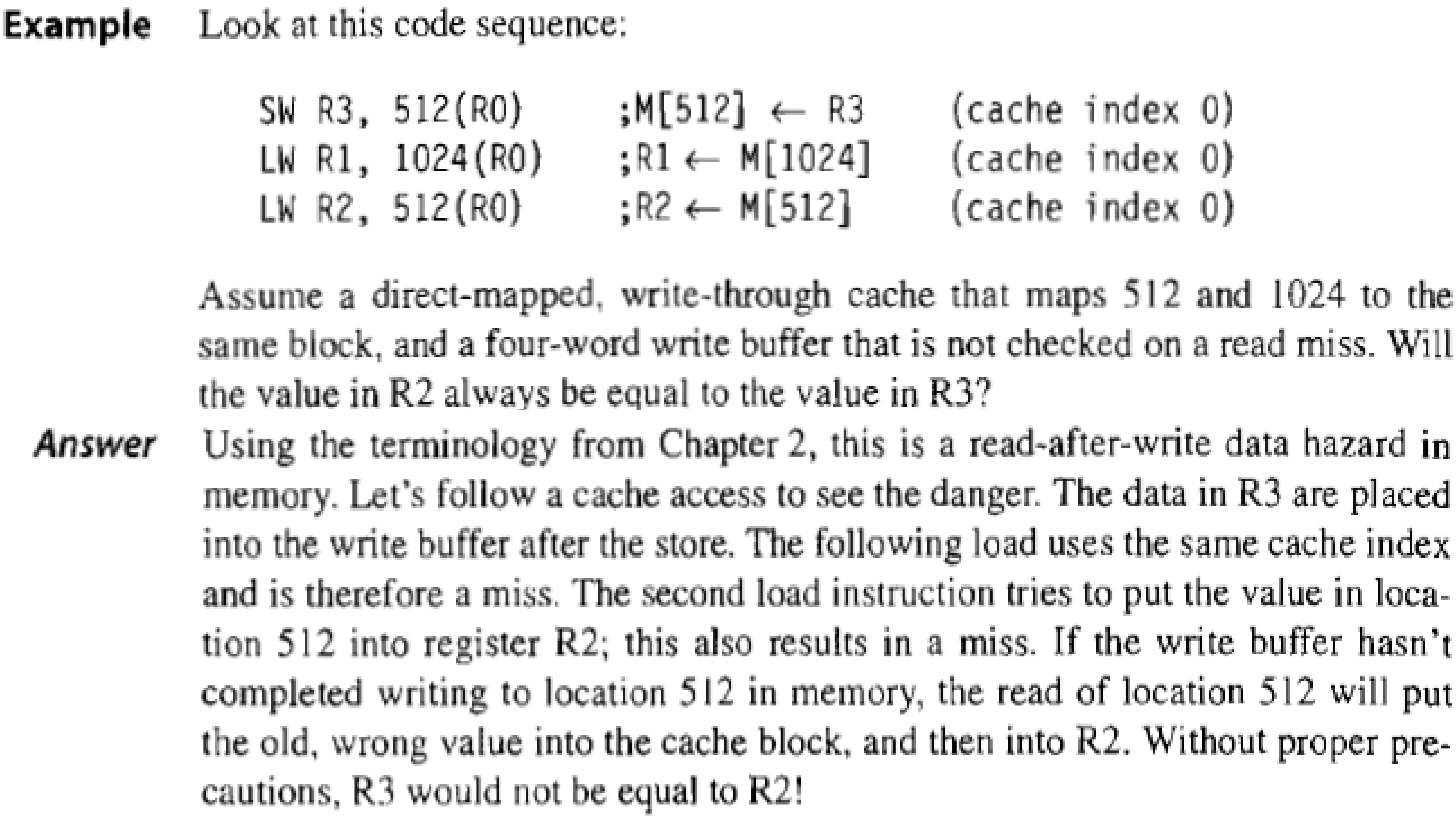
In reality, second-level caches are almost always synchronized with the first-level cache and processor. Accordingly, the second-level hit time must be an integral number of clock cycles. If we are lucky, we shave the second-level hit time to 10 cycles; if not, we round up to 11 cycles. Either choice is an improvement over the direct-mapped second-level cache:

$$\text{Miss penalty}_{\text{2-way L2}} = 10 + 20\% \times 200 = 50.0 \text{ clock cycles}$$
$$\text{Miss penalty}_{\text{2-wav L2}} = 11 + 20\% \times 200 = 51.0 \text{ clock cycles}$$

5- Giving priority to read misses over writes

- Reduces miss penalty
  - Read posterior “esconde” miss penalty do write
- Em caches com write-through: write buffer
  - pode trazer complicações RAW
  - solução: read verifica se há writes pendentes no buffer
- Em caches com write-back
  - funcionamento idêntico: substituição de block dirty → write buffer

Exmpl B-35: priority to read miss



Example Look at this code sequence:

```
SW R3, 512(R0)      ;M[512] ← R3     (cache index 0)
LW R1, 1024(R0)     ;R1 ← M[1024]    (cache index 0)
LW R2, 512(R0)      ;R2 ← M[512]     (cache index 0)
```

Assume a direct-mapped, write-through cache that maps 512 and 1024 to the same block, and a four-word write buffer that is not checked on a read miss. Will the value in R2 always be equal to the value in R3?

Answer Using the terminology from Chapter 2, this is a read-after-write data hazard in memory. Let's follow a cache access to see the danger. The data in R3 are placed into the write buffer after the store. The following load uses the same cache index and is therefore a miss. The second load instruction tries to put the value in location 512 into register R2; this also results in a miss. If the write buffer hasn't completed writing to location 512 in memory, the read of location 512 will put the old, wrong value into the cache block, and then into R2. Without proper precautions, R3 would not be equal to R2!

6- Avoiding address translation in cache indexing → Hit time

- 2 tarefas acesso à cache: indexar a cache e comparar tag
- Alternativa1: cache (com endereço) virtual:
  - tradução zero: na indexação e na comp. c tag
  - Problemas com cache virtual:
    - proteção: é verificada na tradução v → p (solução no App B)
    - na mudança de contexto, dados na cache são inúteis → flush the cache (solução no App B) (usar PID para só descartar do processo)
    - OS e user programs podem usar dois endereços virtuais para o mesmo endereço físico→ aliasing ou synonyms. Podem haver duas cópias do mesmo end. físico (“coerência”?) (solução no App B)
    - I/O usa endereço físico → necessária tradução em op de I/O
- Alternativa2, virtually indexed, physically tagged
  - indexação: usar parte da page offset do end virtual (sem tradução)
    - probl: cache size (direct) <= page size
  - simultaneamente, traduzir parte do end virtual e comparar com tag (end físico)

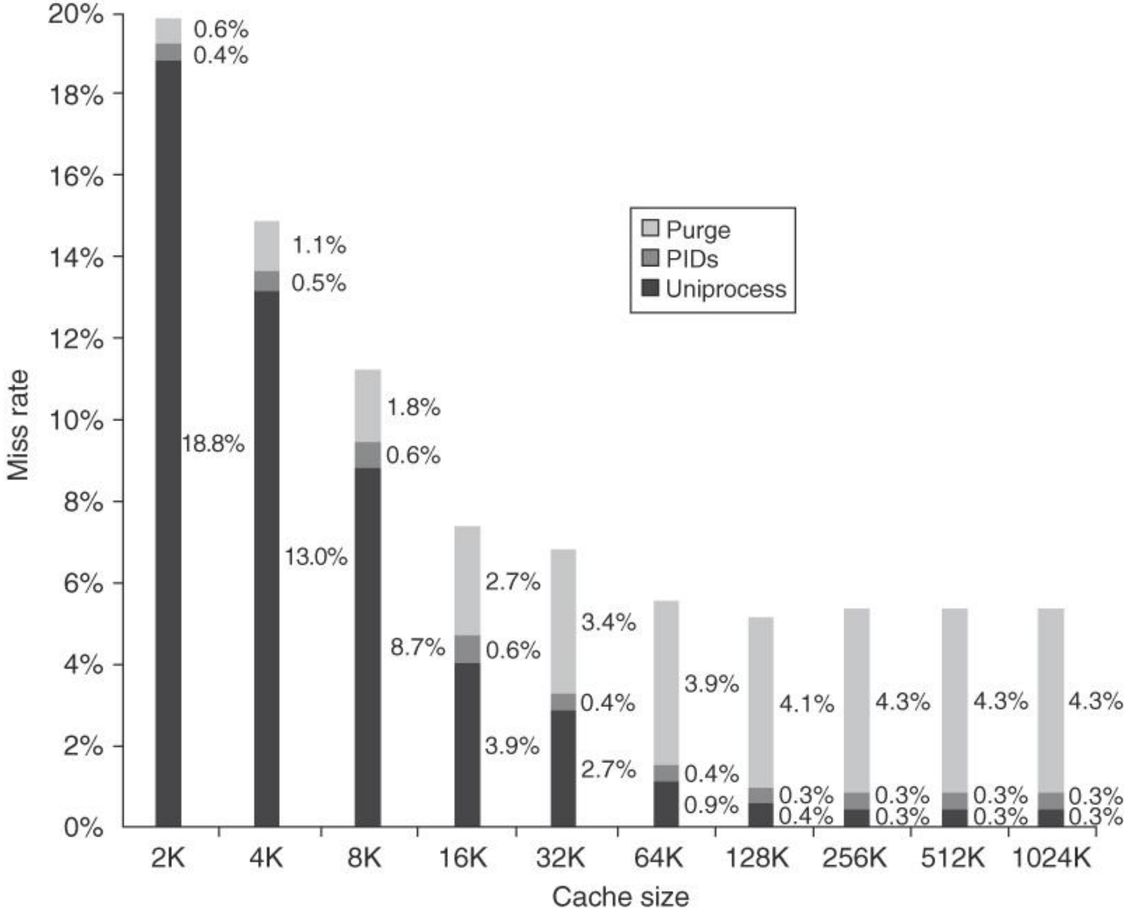


Figure B.16 Miss rate versus virtually addressed cache size of a program measured three ways: without process switches (uniprocess), with process switches using a process-identifier tag (PID), and with process switches but without PIDs (purge). PIDs increase the uniprocess absolute miss rate by 0.3% to 0.6% and save 0.6% to 4.3% over purging. Agarwal [1987] collected these statistics for the Ultrix operating system running on a VAX, assuming direct-mapped caches with a block size of 16 bytes. Note that the miss rate goes up from 128K to 256K. Such nonintuitive behavior can occur in caches because changing size changes the mapping of memory blocks onto cache blocks, which can change the conflict miss rate.

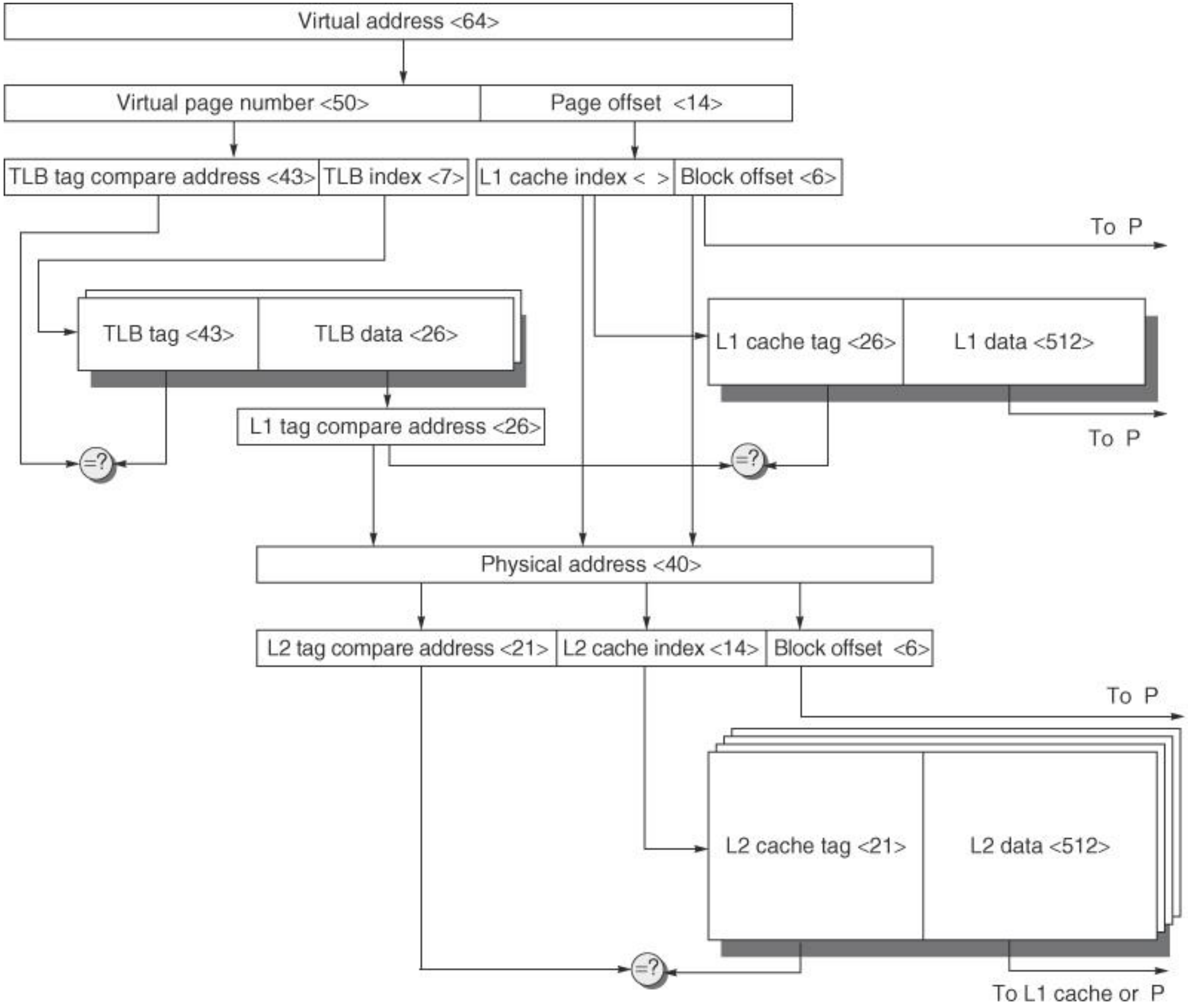
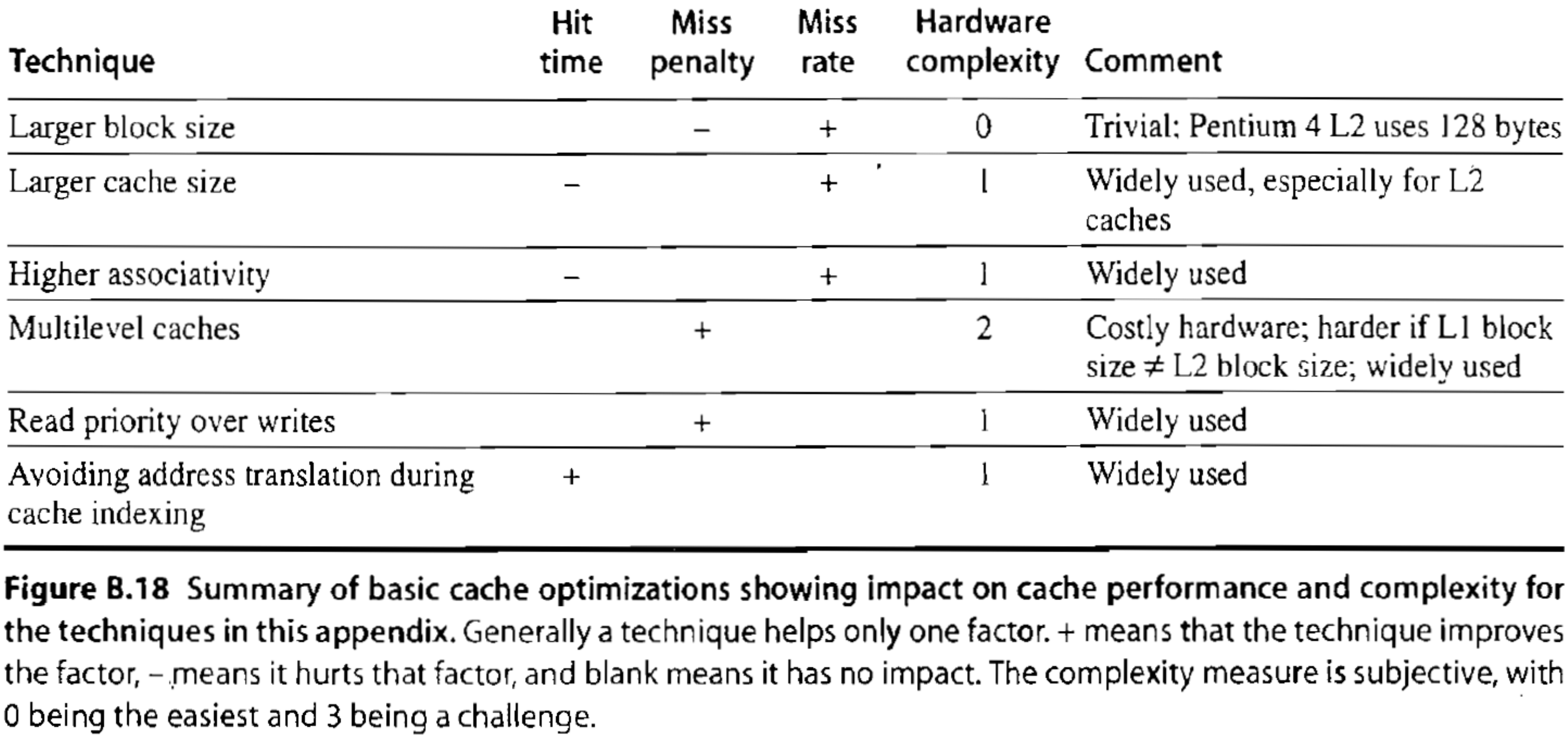


Figure B.17 The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache access. The page size is 16 KB. The TLB is two-way set associative with 256 entries. The L1 cache is a direct-mapped 16 KB, and the L2 cache is a four-way set associative with a total of 4 MB. Both use 64-byte blocks. The virtual address is 64 bits and the physical address is 40 bits.

IC-UNICAMP

Resumo das técnicas de otimização



Technique	Hit time	Miss penalty	Miss rate	Hardware complexity	Comment
Larger block size		–	+	0	Trivial: Pentium 4 L2 uses 128 bytes
Larger cache size	–		+	1	Widely used, especially for L2 caches
Higher associativity	–		+	1	Widely used
Multilevel caches		+		2	Costly hardware; harder if L1 block size ≠ L2 block size; widely used
Read priority over writes		+		1	Widely used
Avoiding address translation during cache indexing	+			1	Widely used

Figure B.18 Summary of basic cache optimizations showing impact on cache performance and complexity for the techniques in this appendix. Generally a technique helps only one factor. + 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.

B.4 Memória virtual

- Divide espaço físico em blocos e os aloca para processos
  - Proteção: acesso somente ao seu espaço privado
  - Redução do tempo para iniciar um programa: não é necessário carregar o programa todo
  - Execução de programas maiores do que a memória física
  - Facilita carga de programas e sua relocação

(Antes do aparecimento da memória virtual, essas funções tinham que ser implementadas manualmente pelo programador)

Programa em espaço virtual contíguo

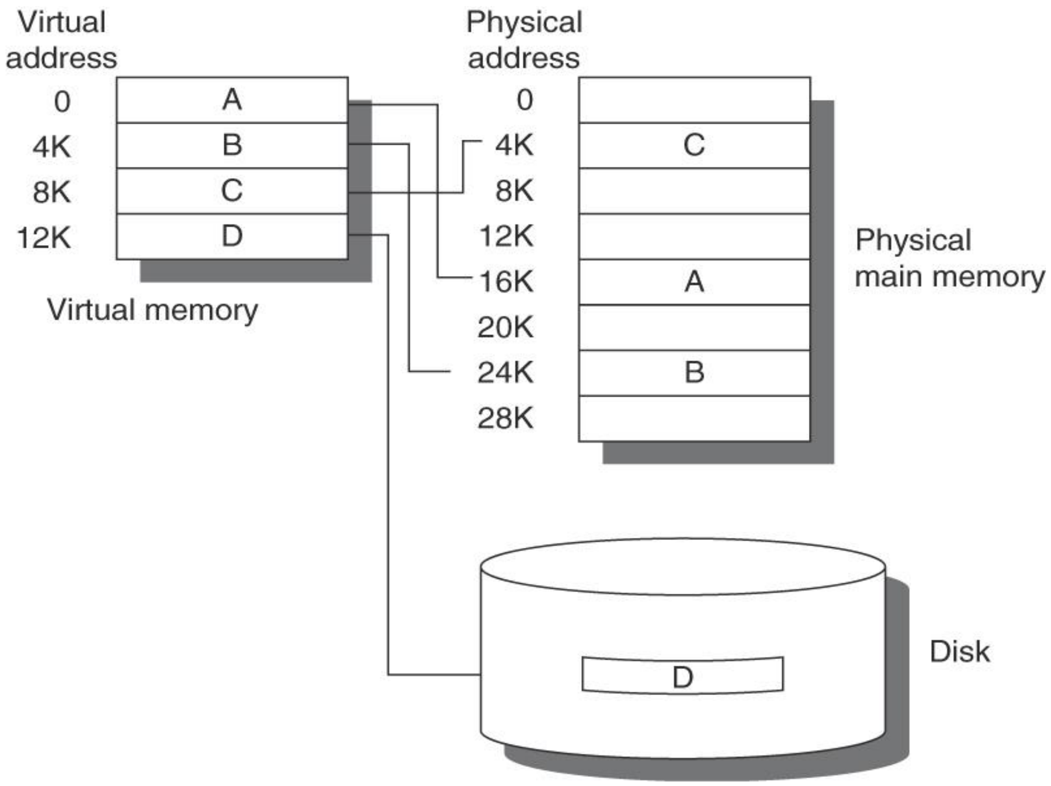
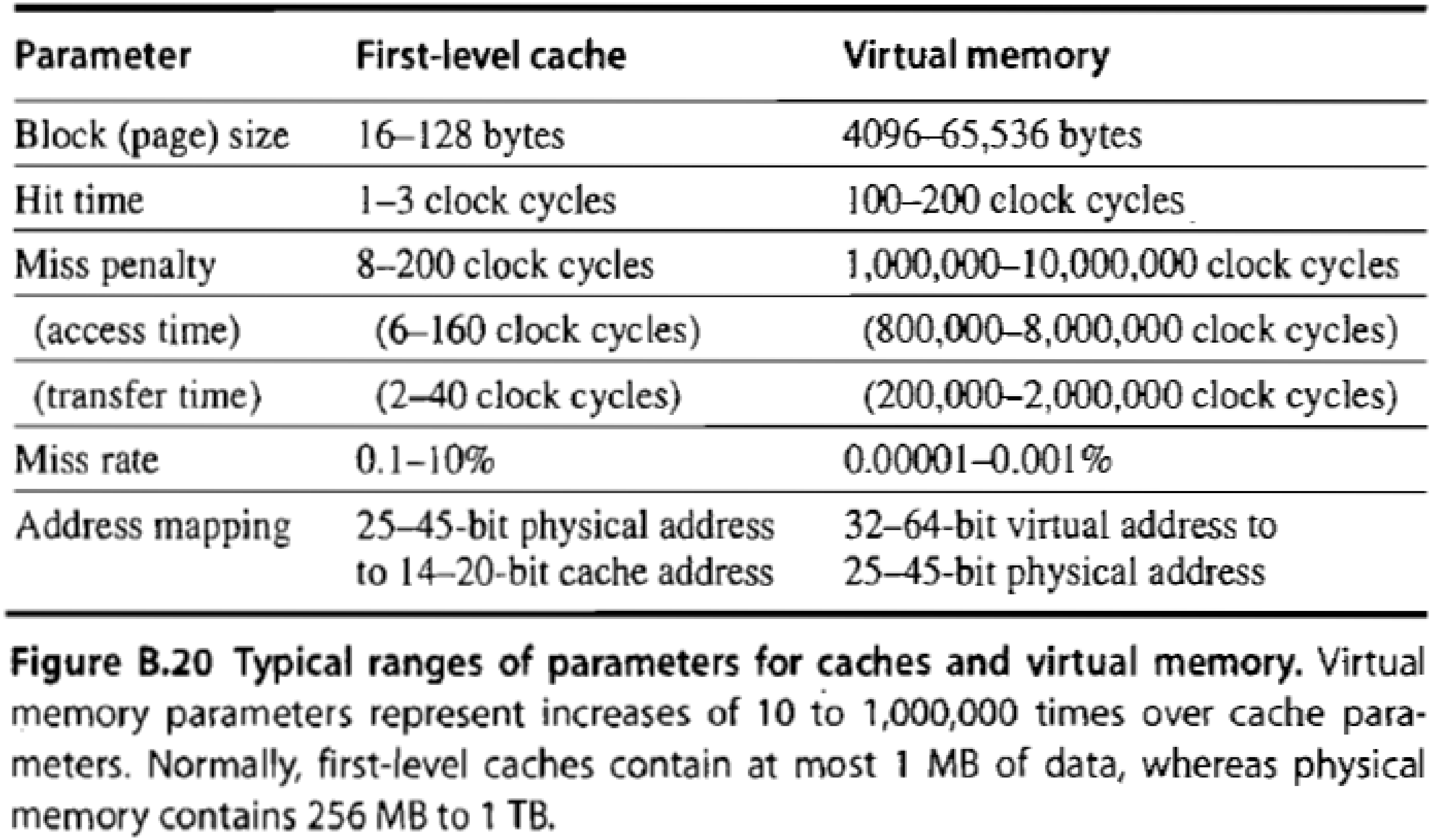


Figure B.19 The logical program in its contiguous virtual address space is shown on the left. It consists of four pages, A, B, C, and D. The actual location of three of the blocks is in physical main memory and the other is located on the disk.

IC-UNICAMP

cache L1 vs Memória Virtual



Parameter	First-level cache	Virtual memory
Block (page) size	16–128 bytes	4096–65,536 bytes
Hit time	1–3 clock cycles	100–200 clock cycles
Miss penalty	8–200 clock cycles	1,000,000–10,000,000 clock cycles
(access time)	(6–160 clock cycles)	(800,000–8,000,000 clock cycles)
(transfer time)	(2–40 clock cycles)	(200,000–2,000,000 clock cycles)
Miss rate	0.1–10%	0.00001–0.001%
Address mapping	25–45-bit physical address to 14–20-bit cache address	32–64-bit virtual address to 25–45-bit physical address

Figure B.20 Typical ranges of parameters for caches and virtual memory. Virtual memory parameters represent increases of 10 to 1,000,000 times over cache parameters. Normally, first-level caches contain at most 1 MB of data, whereas physical memory contains 256 MB to 1 TB.

Paginação e segmentação

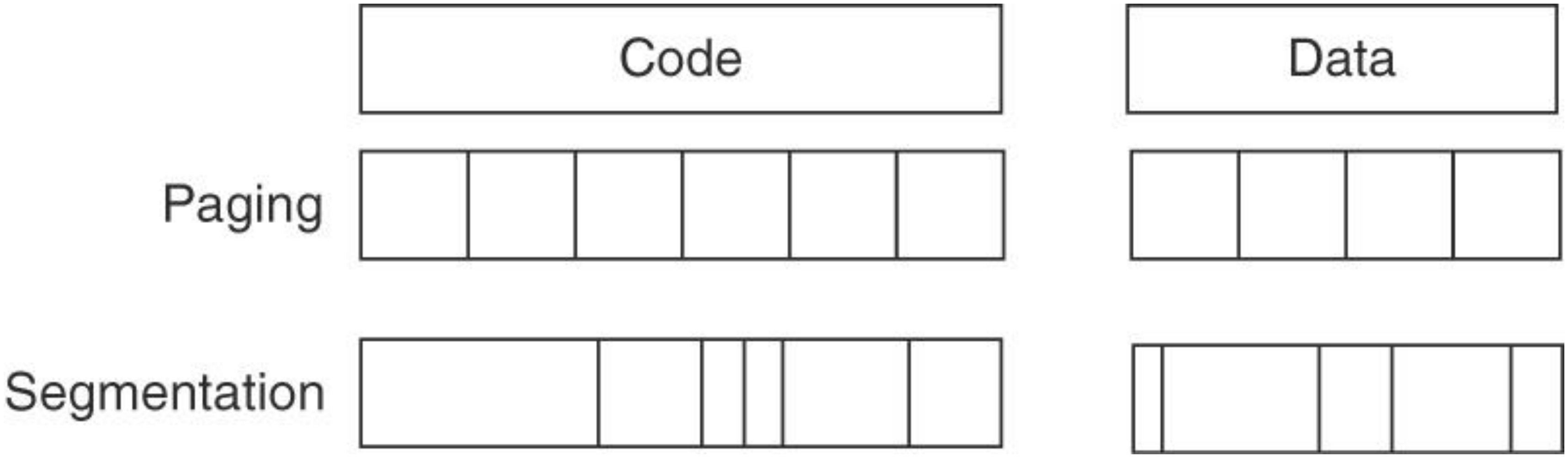


Figure B.21 Example of how paging and segmentation divide a program.

Virtual → Physical Mapping

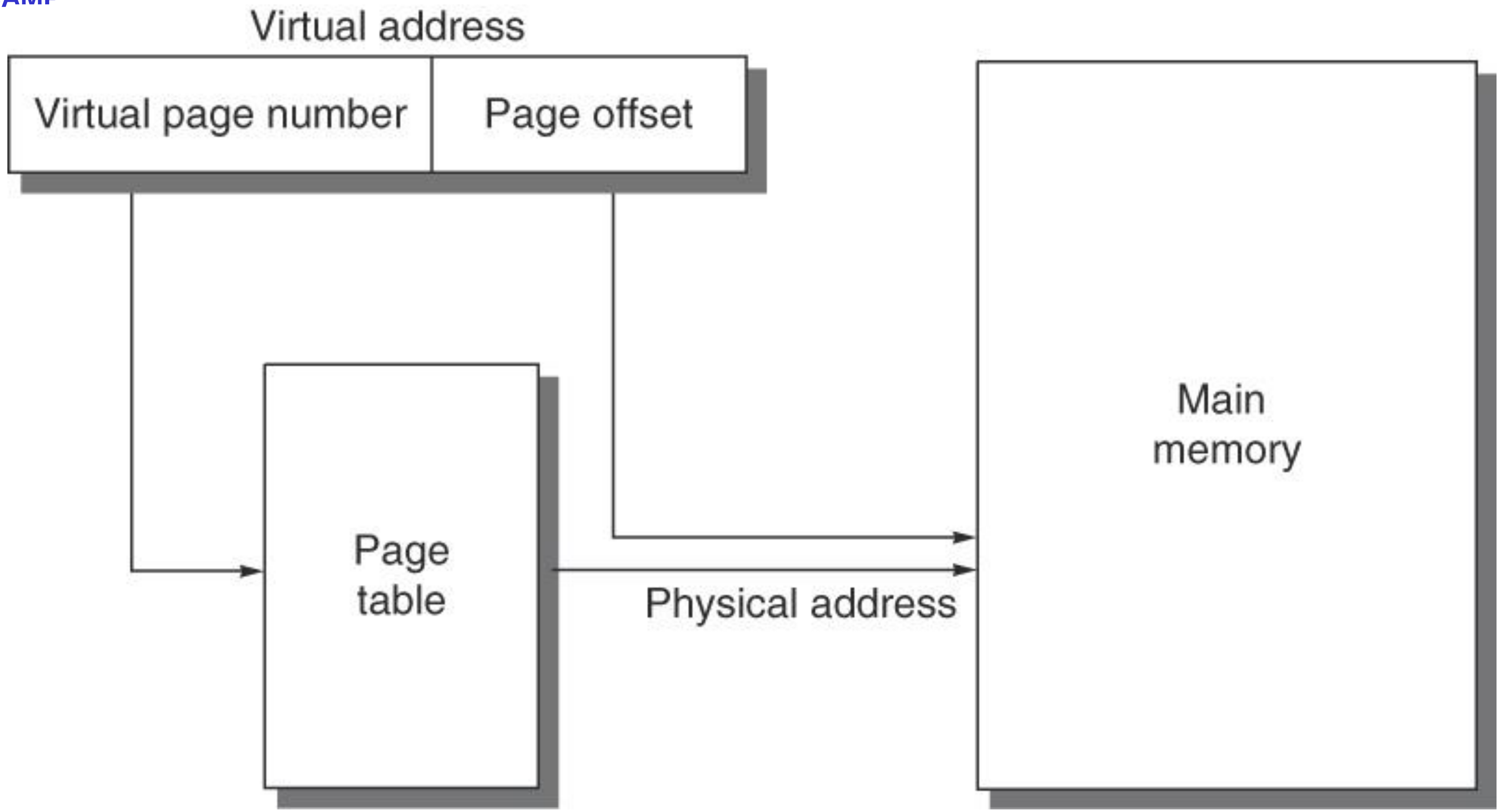


Figure B.23 The mapping of a virtual address to a physical address via a page table.

IC-UNICAMP

TLB: mapeamento rápido

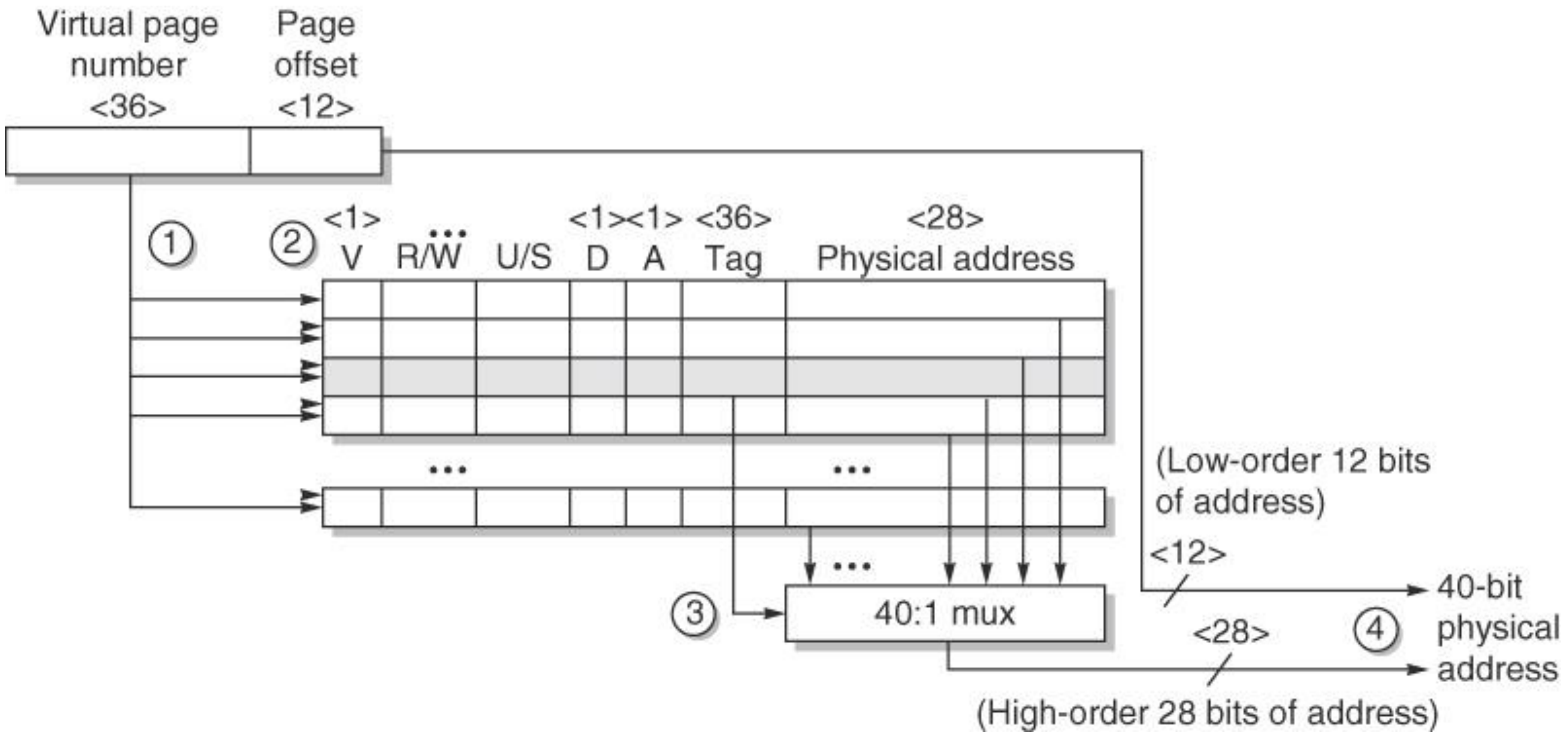


Figure B.24 Operation of the Opteron data TLB during address translation. The four steps of a TLB hit are shown as circled numbers. This TLB has 40 entries. Section B.5 describes the various protection and access fields of an Opteron page table entry.

IC-UNICAMP

Virtual → físico → L1 + L2

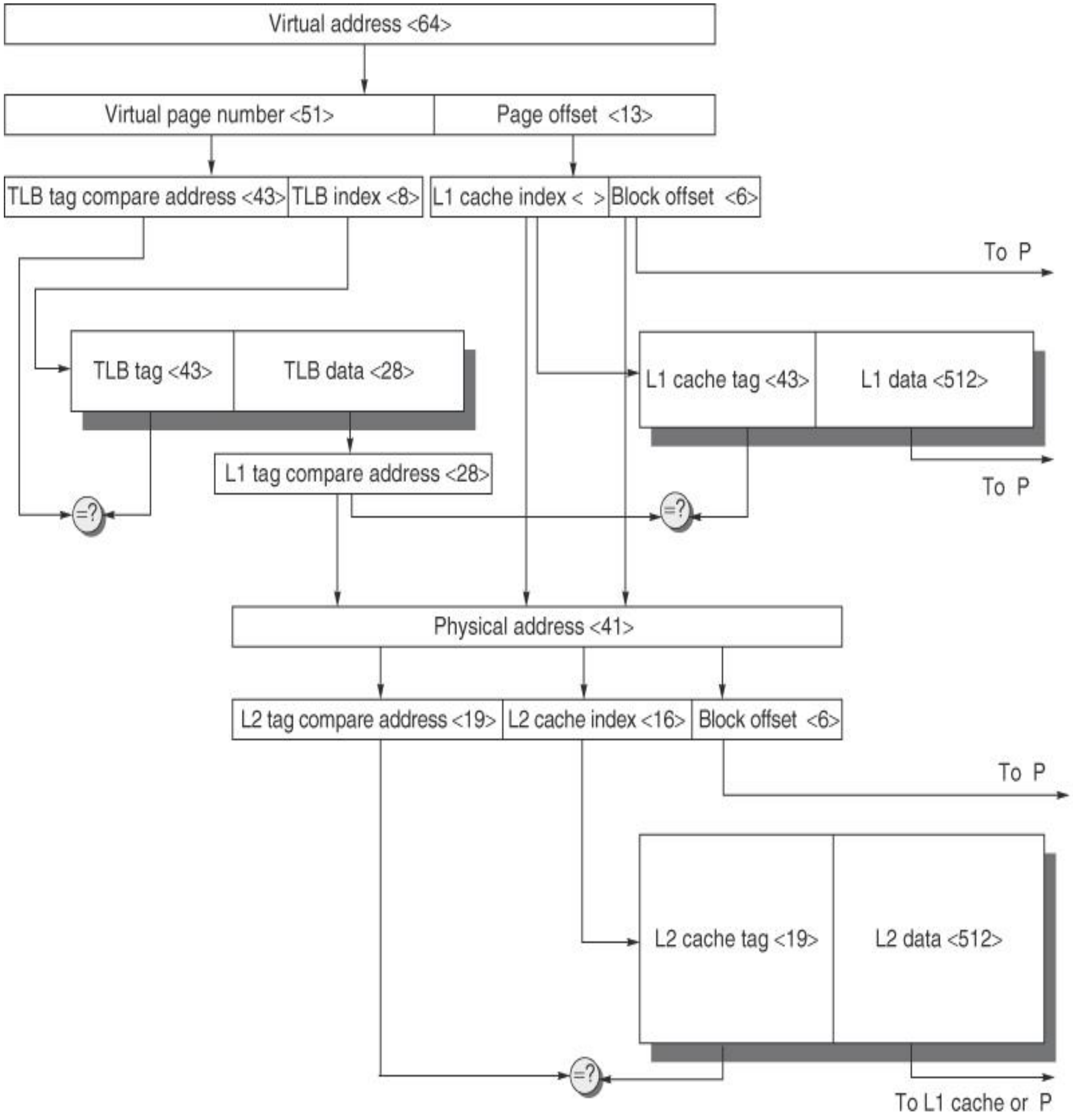


Figure B.25 The overall picture of a hypothetical memory hierarchy going from virtual address to L2 cache access. The page size is 8 KB. The TLB is direct mapped with 256 entries. The L1 cache is a direct-mapped 8 KB, and the L2 cache is a direct-mapped 4 MB. Both use 64-byte blocks. The virtual address is 64 bits and the physical address is 41 bits. The primary difference between this simple figure and a real cache is replication of pieces of this figure.