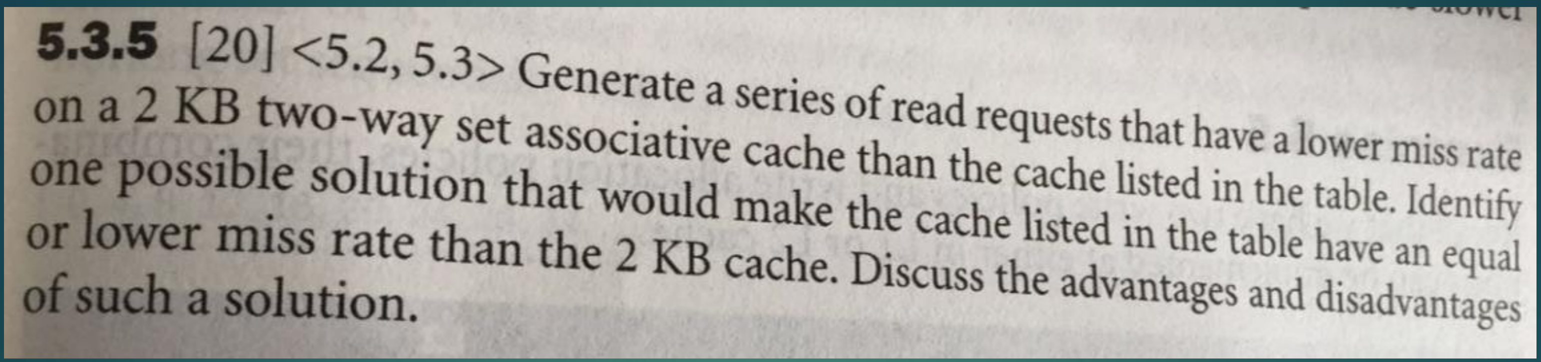
VE370 RC8

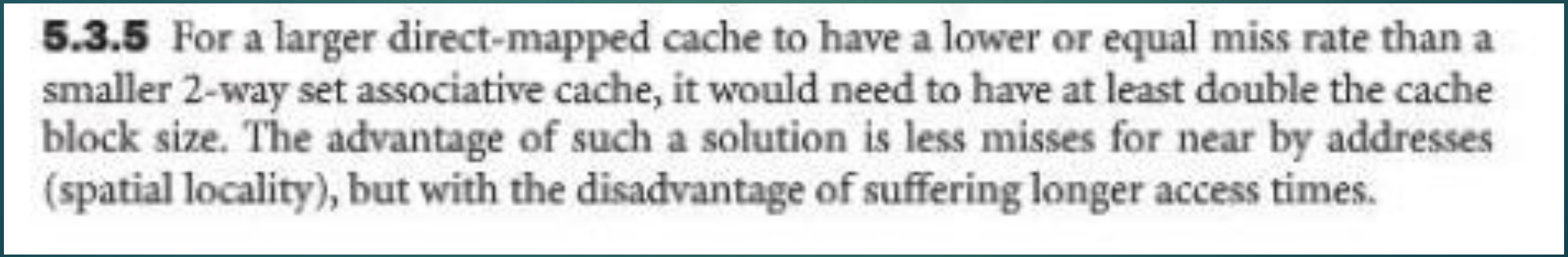
HW8 SOLUTION

5.3.5



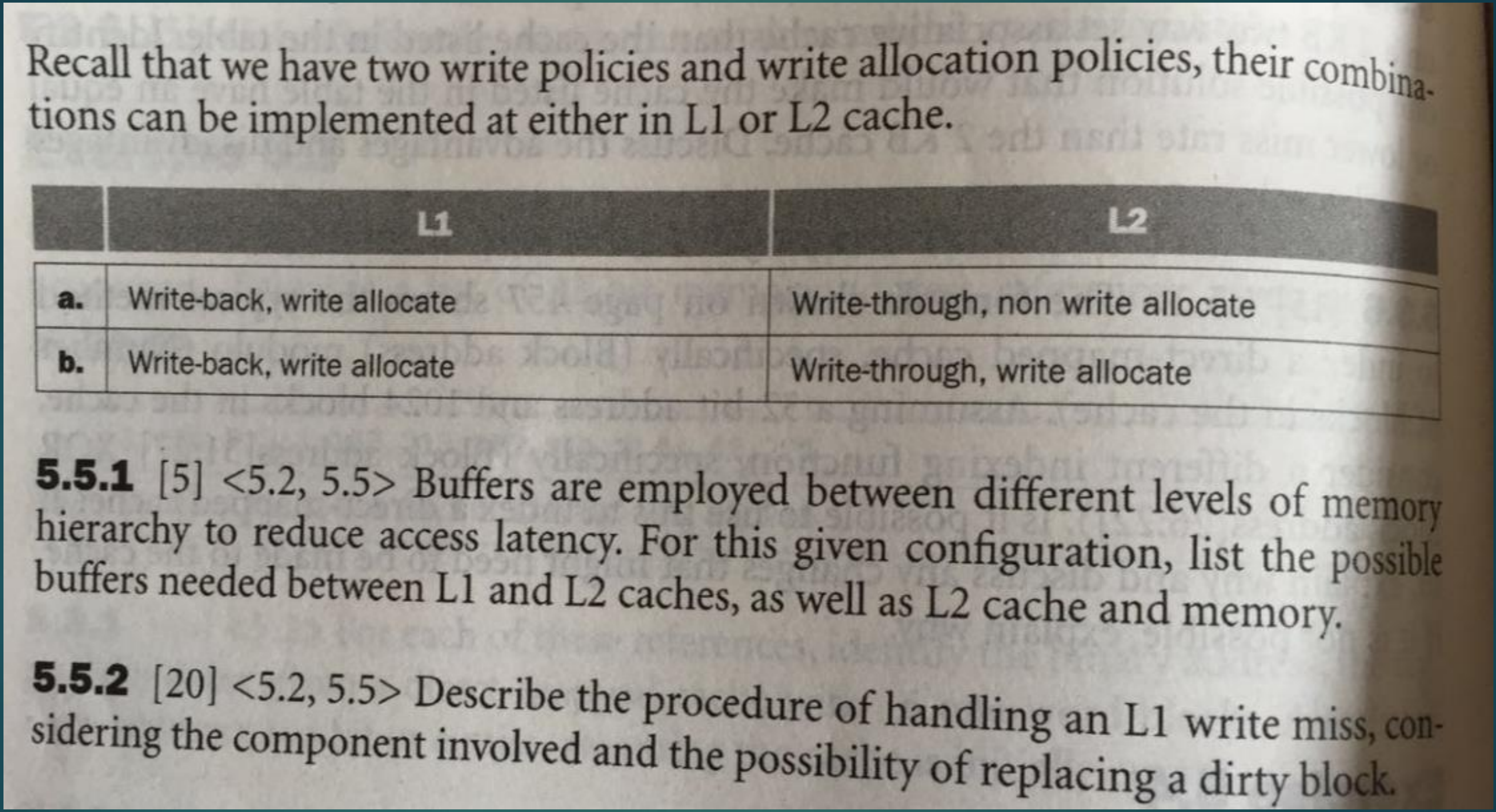
5.3.5 [20] <5.2, 5.3> Generate a series of read requests that have a lower miss rate on a 2 KB two-way set associative cache than the cache listed in the table. Identify one possible solution that would make the cache listed in the table have an equal or lower miss rate than the 2 KB cache. Discuss the advantages and disadvantages of such a solution.

E.g: word address: 0, 64, 0, 64, 0, 64...
(2KB cache, 16-word per block, 2-set)



5.3.5 For a larger direct-mapped cache to have a lower or equal miss rate than a smaller 2-way set associative cache, it would need to have at least double the cache block size. The advantage of such a solution is less misses for near by addresses (spatial locality), but with the disadvantage of suffering longer access times.

5.5.2

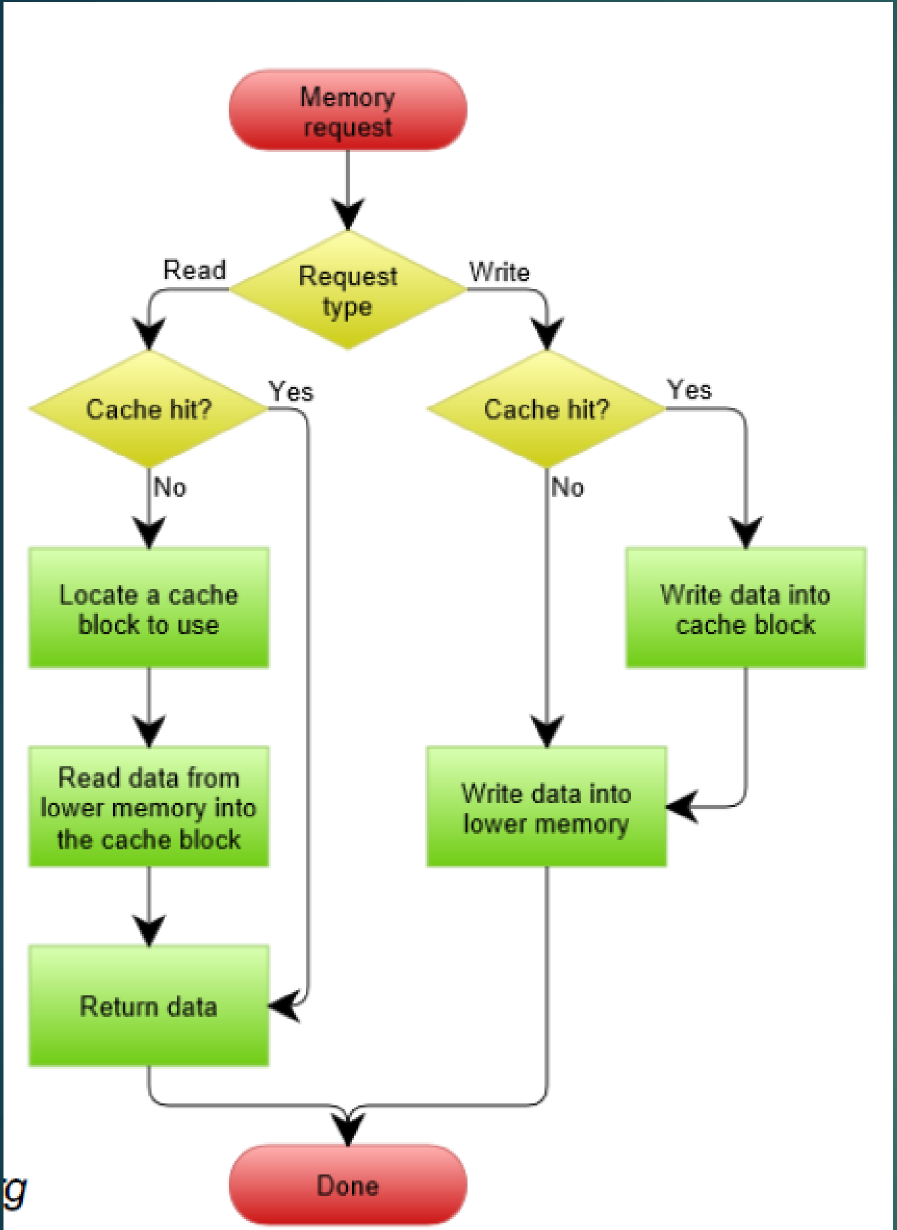


Recall that we have two write policies and write allocation policies, their combinations can be implemented at either in L1 or L2 cache.

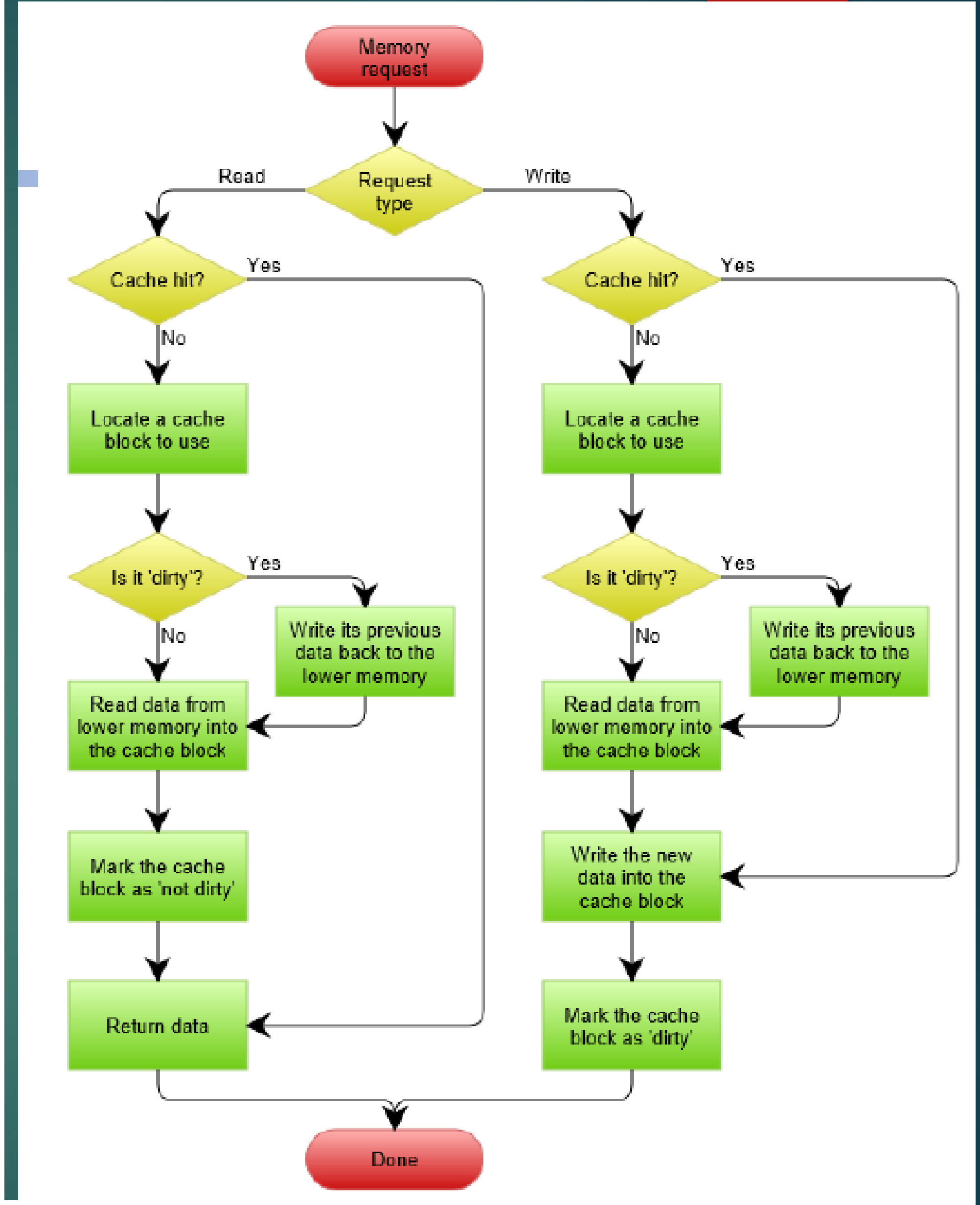
	L1	L2
a.	Write-back, write allocate	Write-through, non write allocate
b.	Write-back, write allocate	Write-through, write allocate

5.5.1 [5] <5.2, 5.5> Buffers are employed between different levels of memory hierarchy to reduce access latency. For this given configuration, list the possible buffers needed between L1 and L2 caches, as well as L2 cache and memory.

5.5.2 [20] <5.2, 5.5> Describe the procedure of handling an L1 write miss, considering the component involved and the possibility of replacing a dirty block.

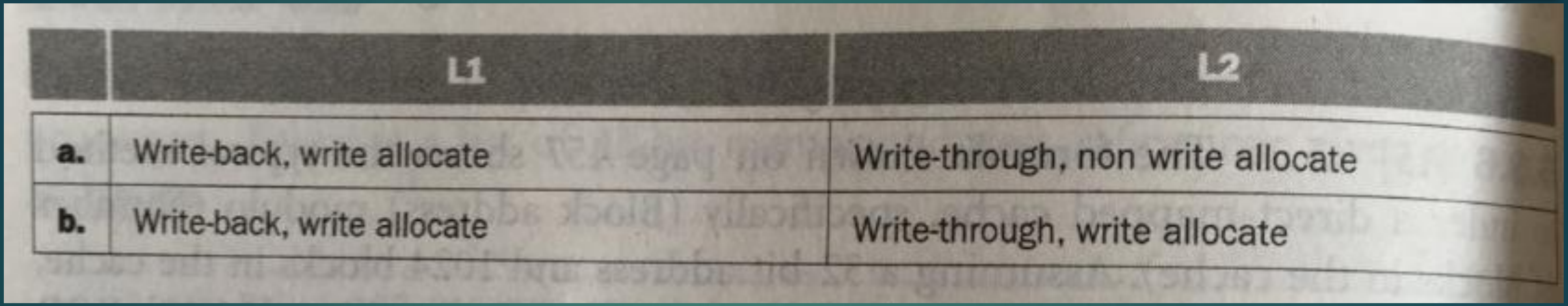

Memory request
Request type
Read
Write
Cache hit?
Yes
No
Locate a cache block to use
Read data from lower memory into the cache block
Return data
Cache hit?
Yes
No
Write data into cache block
Write data into lower memory
Done

g

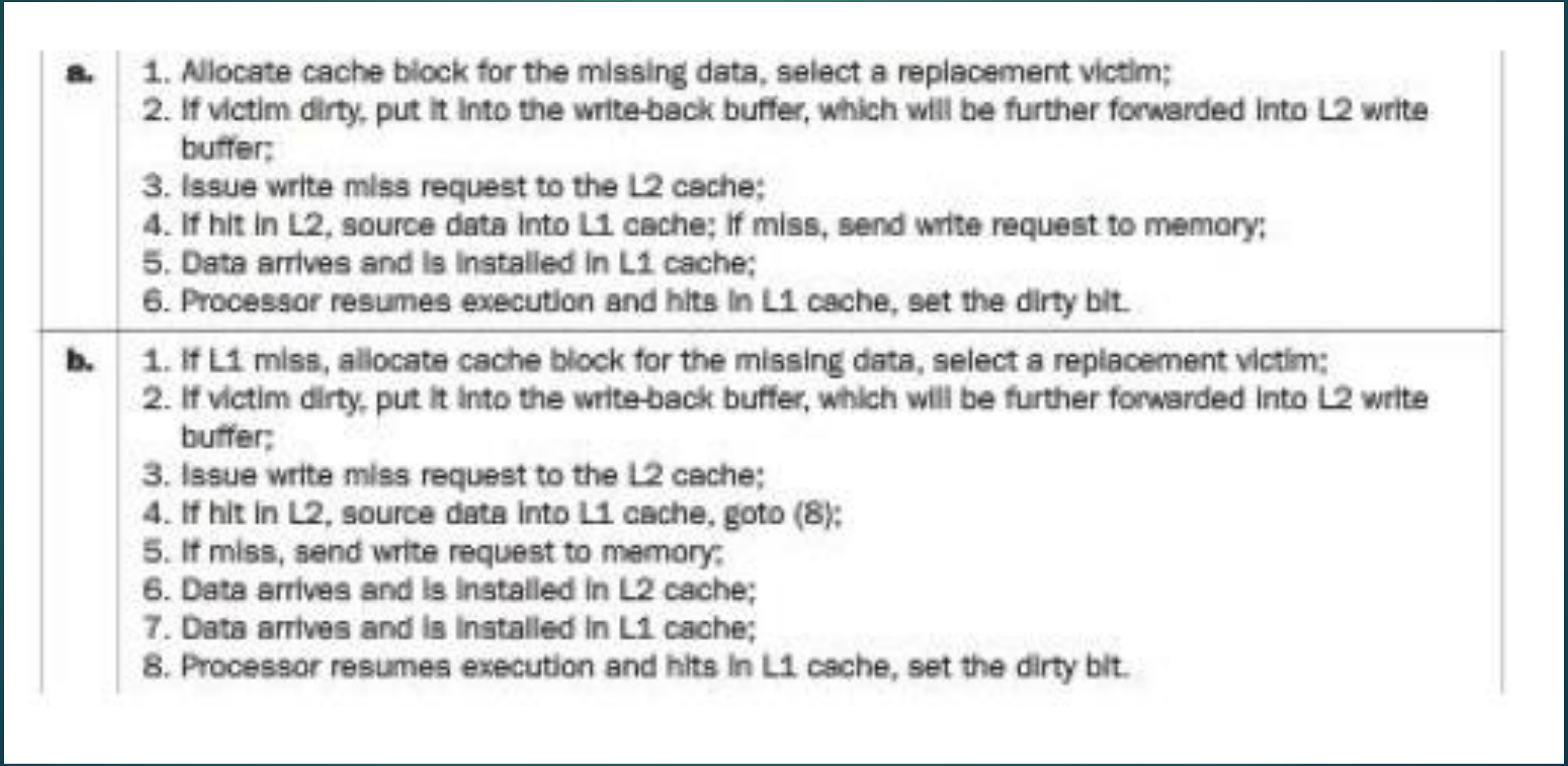

Memory request
Request type
Read
Write
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Mark the cache block as 'not dirty'
Return data
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as 'dirty'
Done

- *Write allocate* (also called *fetch on write*): data at the missed-write location is loaded to cache, followed by a write-hit operation. In this approach, write misses are similar to read misses.
- *No-write allocate* (also called *write-no-allocate* or *write around*): data at the missed-write location is not loaded to cache, and is written directly to the backing store. In this approach, only the reads are being cached.

5.5.2

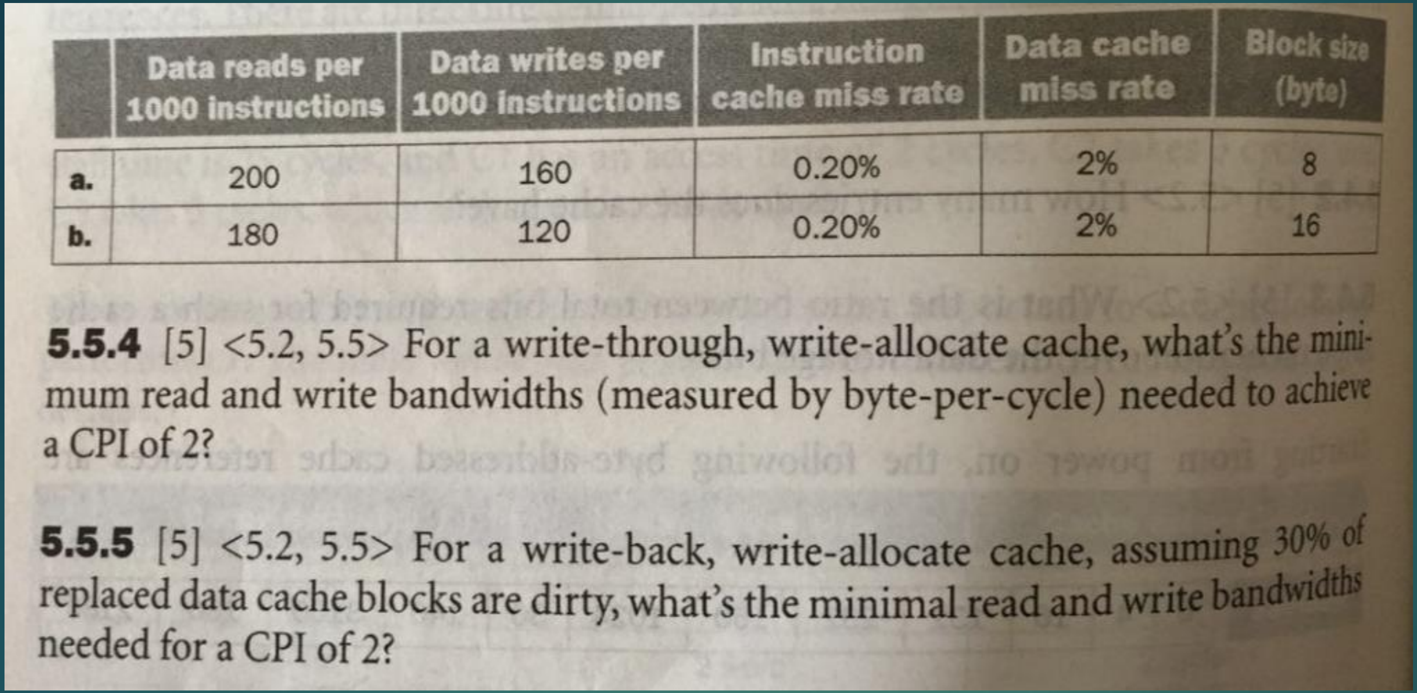


	L1	L2
a.	Write-back, write allocate	Write-through, non write allocate
b.	Write-back, write allocate	Write-through, write allocate



a.	1. Allocate cache block for the missing data, select a replacement victim; 2. If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; 3. Issue write miss request to the L2 cache; 4. If hit in L2, source data into L1 cache; if miss, send write request to memory; 5. Data arrives and is installed in L1 cache; 6. Processor resumes execution and hits in L1 cache, set the dirty bit.
b.	1. If L1 miss, allocate cache block for the missing data, select a replacement victim; 2. If victim dirty, put it into the write-back buffer, which will be further forwarded into L2 write buffer; 3. Issue write miss request to the L2 cache; 4. If hit in L2, source data into L1 cache, goto (8); 5. If miss, send write request to memory; 6. Data arrives and is installed in L2 cache; 7. Data arrives and is installed in L1 cache; 8. Processor resumes execution and hits in L1 cache, set the dirty bit.

5.5.5



	Data reads per 1000 instructions	Data writes per 1000 instructions	Instruction cache miss rate	Data cache miss rate	Block size (byte)
a.	200	160	0.20%	2%	8
b.	180	120	0.20%	2%	16

5.5.4 [5] <5.2, 5.5> For a write-through, write-allocate cache, what's the minimum read and write bandwidths (measured by byte-per-cycle) needed to achieve a CPI of 2?

5.5.5 [5] <5.2, 5.5> For a write-back, write-allocate cache, assuming 30% of replaced data cache blocks are dirty, what's the minimal read and write bandwidths needed for a CPI of 2?

5.5.5

With the write back policy, the cache content may be changed and inconsistent with that in the main memory. Thus, upon a read miss, if the replaced block is “dirty”, it will first be written back to the main memory, and then the desired block will be brought into the cache. With write allocation, when a write miss occurs, the corresponding block will be brought to the cache, and then new data will be written into this block. The replaced block, if the dirty bit is set, will need to be written into the main memory first before the write block is brought into the cache.

Suppose number of instruction is I, band width W, base CPI=1

a.

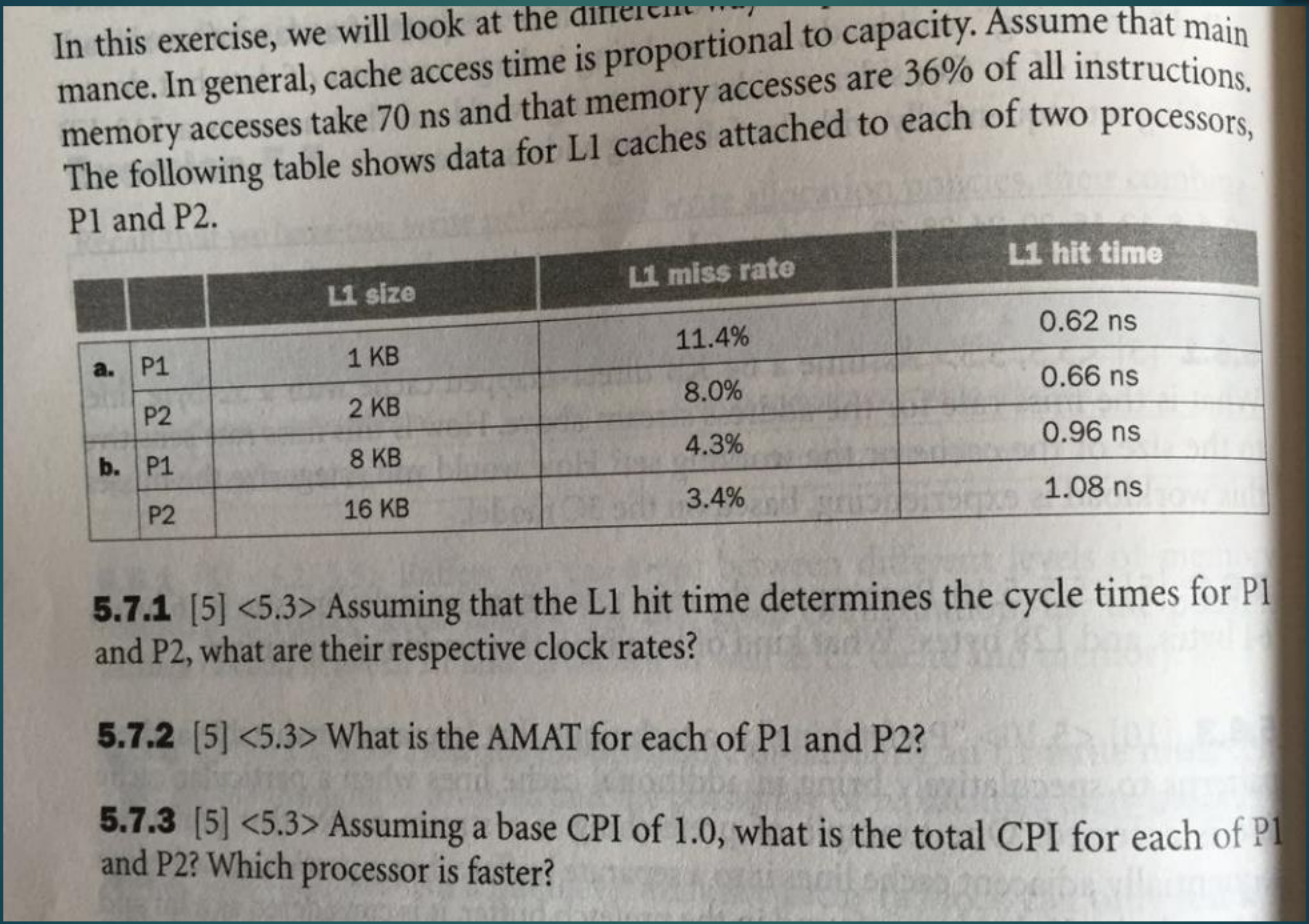
Read miss penalty: $I*0.2*0.02*(1+0.3)*(8/W)$

Write miss penalty: $I*0.16*0.02*(1+0.3)*(8/W)$

Instruction miss penalty: $I*0.002*(8/W)$

$I + I*0.2*0.02*(1+0.3)*(8/W)+I*0.16*0.02*(1+0.3)*(8/W)+I*0.002*(8/W)<=2I$

5.7.1-3



In this exercise, we will look at the different [illegible] mance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that memory accesses are 36% of all instructions. The following table shows data for L1 caches attached to each of two processors, P1 and P2.

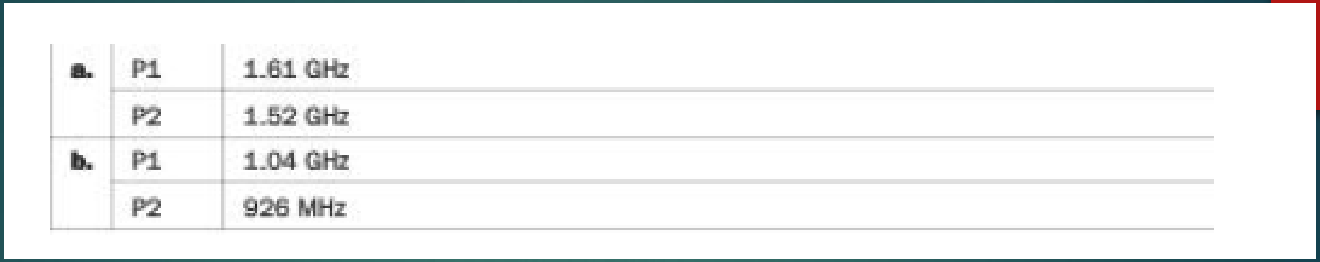
		L1 size	L1 miss rate	L1 hit time
a.	P1	1 KB	11.4%	0.62 ns
	P2	2 KB	8.0%	0.66 ns
b.	P1	8 KB	4.3%	0.96 ns
	P2	16 KB	3.4%	1.08 ns

5.7.1 [5] <5.3> Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

5.7.2 [5] <5.3> What is the AMAT for each of P1 and P2?

5.7.3 [5] <5.3> Assuming a base CPI of 1.0, what is the total CPI for each of P1 and P2? Which processor is faster?

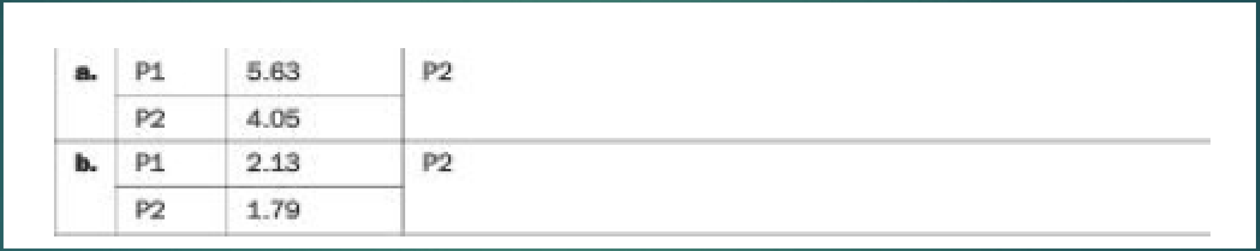
5.7.1



a.	P1	1.61 GHz
	P2	1.52 GHz
b.	P1	1.04 GHz
	P2	926 MHz

Clock rate = 1/hit time

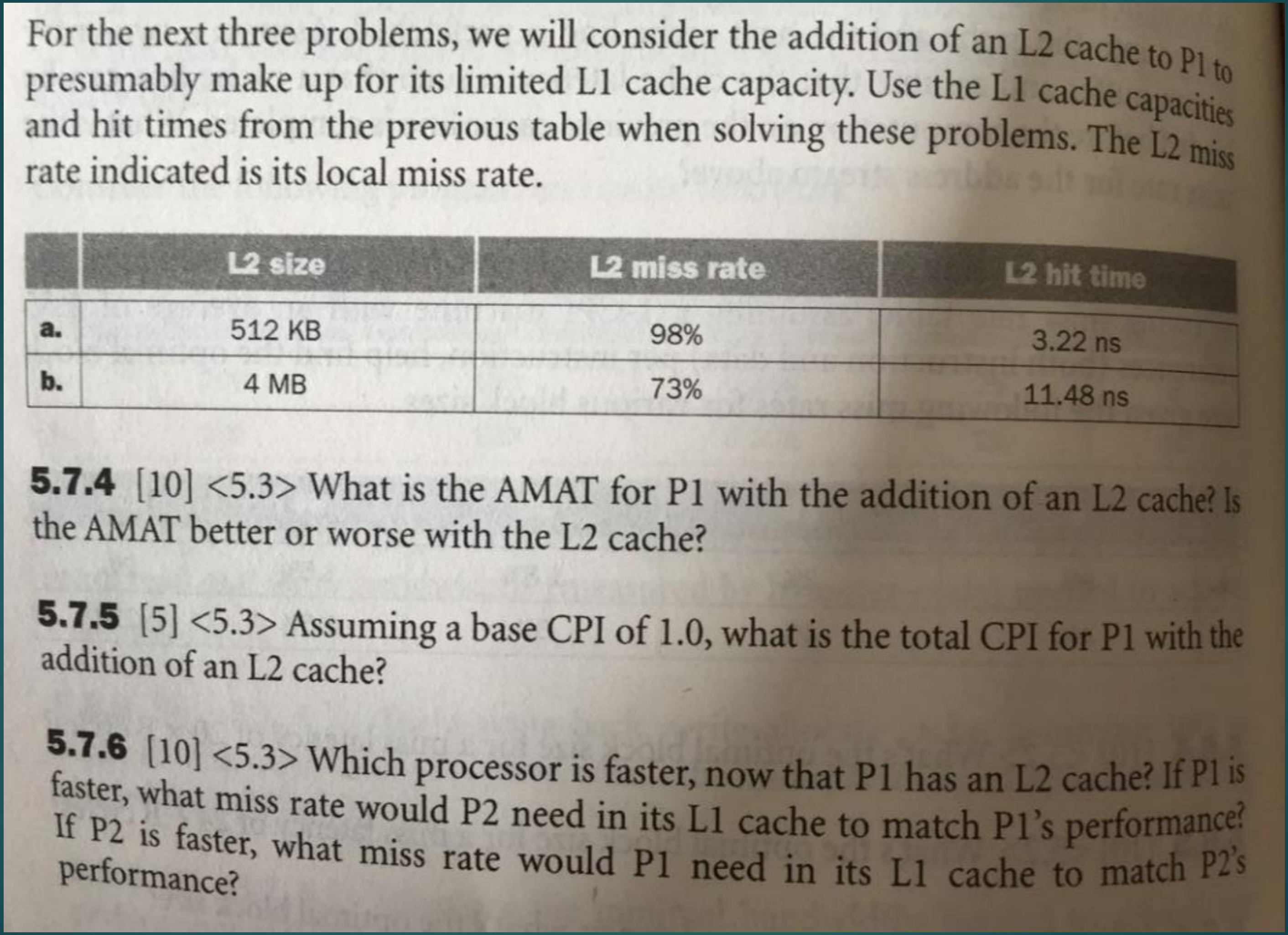
5.7.3



a.	P1	5.63	P2
	P2	4.05	
b.	P1	2.13	P2
	P2	1.79	

for (a), first calculate AMAT for P1 &P2
P1: 0.114*70 + 1*0.62 = 8.60ns, using 8.60/0.62 = 13.87 clock cycles;
P2 similarly. Answer is 6.26ns, using 9.48 clock cycles.
Second, P1: 13.87*0.36 + 1*0.64 = 5.63 clock cycles (approximation problems).
5.63*0.62(ns/clock) = 3.49 ns
P2: 9.48*0.36 + 1*0.64 = 4.05 clock cycles (approximation problems).
4.05*0.66 = 2.67 ns

5.7.4-6



For the next three problems, we will consider the addition of an L2 cache to P1 to presumably make up for its limited L1 cache capacity. Use the L1 cache capacities and hit times from the previous table when solving these problems. The L2 miss rate indicated is its local miss rate.

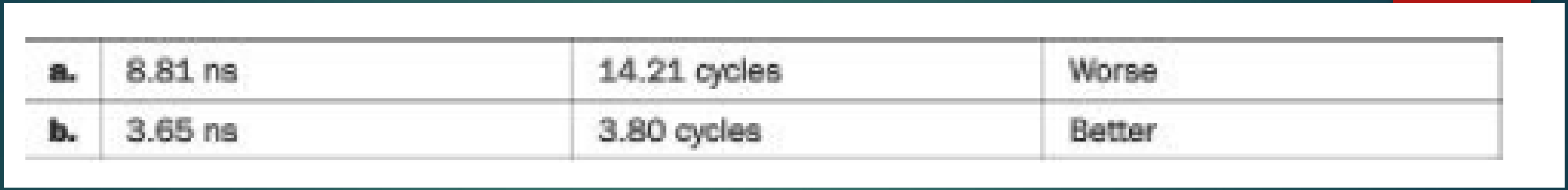
	L2 size	L2 miss rate	L2 hit time
a.	512 KB	98%	3.22 ns
b.	4 MB	73%	11.48 ns

5.7.4 [10] <5.3> What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

5.7.5 [5] <5.3> Assuming a base CPI of 1.0, what is the total CPI for P1 with the addition of an L2 cache?

5.7.6 [10] <5.3> Which processor is faster, now that P1 has an L2 cache? If P1 is faster, what miss rate would P2 need in its L1 cache to match P1's performance? If P2 is faster, what miss rate would P1 need in its L1 cache to match P2's performance?

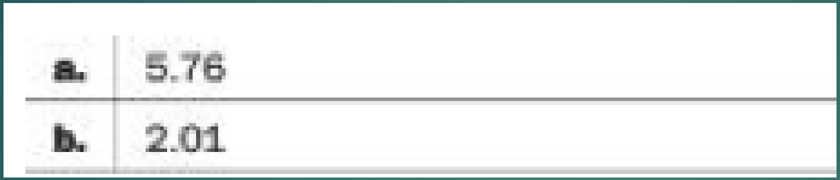
5.7.4



a.	8.81 ns	14.21 cycles	Worse
b.	3.65 ns	3.80 cycles	Better

for (a), first calculate AMAT.
P1: 1*0.62 + 0.114*3.22 + 0.114*0.98*70 = 8.81ns; 8.81/0.62 = 14.21 cycles > 13.87, worse.

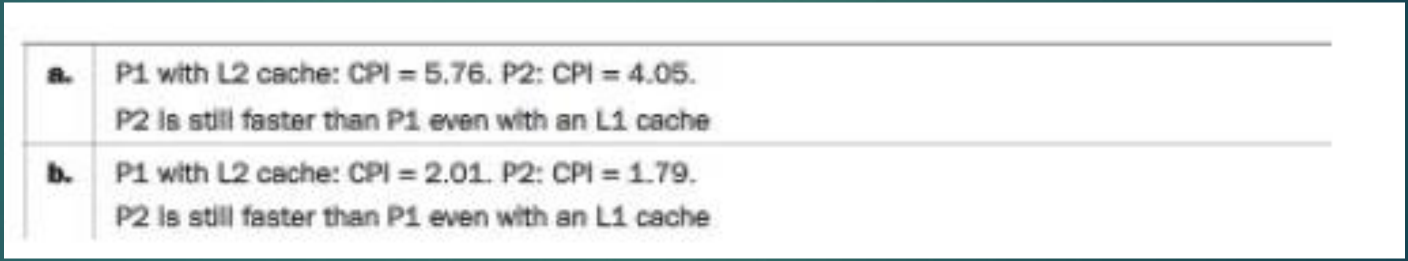
5.7.5



a.	5.76
b.	2.01

for (a), based on 5.7.4
P1: 14.21*0.36 + 1*0.64 = 5.76

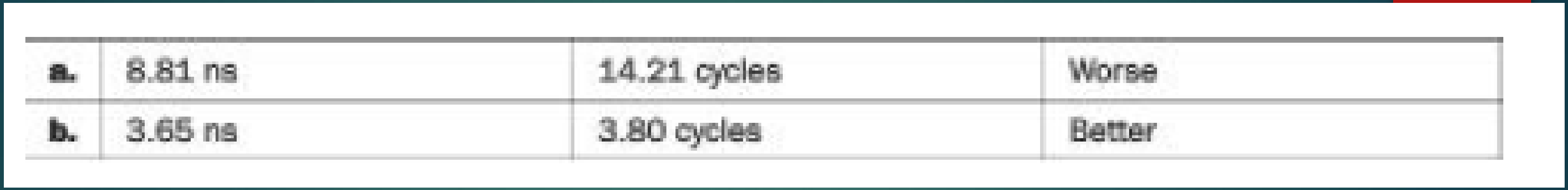
5.7.6



a.	P1 with L2 cache: CPI = 5.76. P2: CPI = 4.05. P2 is still faster than P1 even with an L1 cache
b.	P1 with L2 cache: CPI = 2.01. P2: CPI = 1.79. P2 is still faster than P1 even with an L1 cache

for (a), let r be the miss rate:
AMAT: 1*0.62 + 0.114*3.22 + 0.114*r*70 (ns)
[AMAT(P1) *0.62 + 0.62* 0.64]P1 ≤ [2.67]P2
solve the 'r' from this equation.

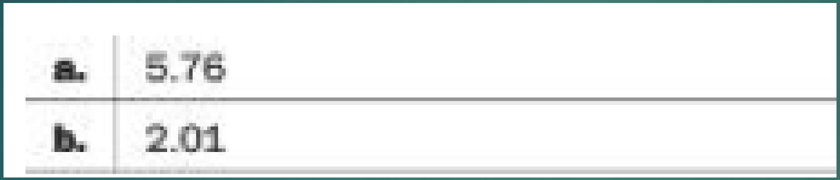
5.7.4



a.	8.81 ns	14.21 cycles	Worse
b.	3.65 ns	3.80 cycles	Better

for (a), first calculate AMAT.
P1: 1*0.62 + 0.114*3.22 + 0.114*0.98*70 = 8.81ns; 8.81/0.62 = 14.21 cycles > 13.87, worse.

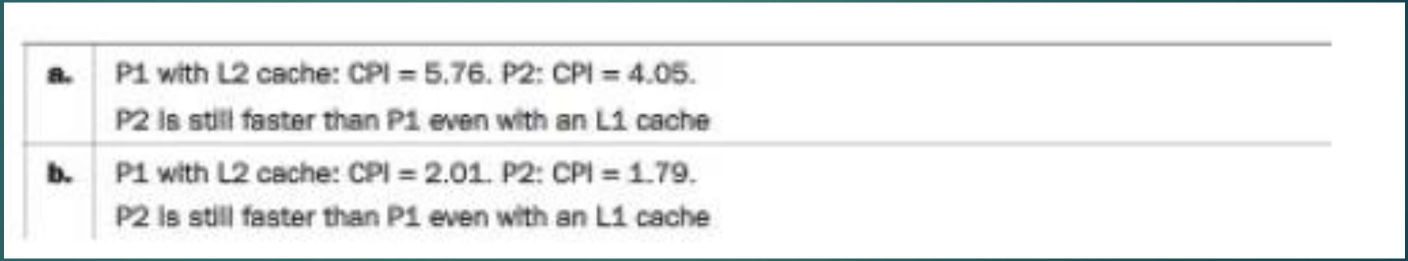
5.7.5



a.	5.76
b.	2.01

for (a), based on 5.7.4
P1: 14.21*0.36 + 1*0.64 = 5.76

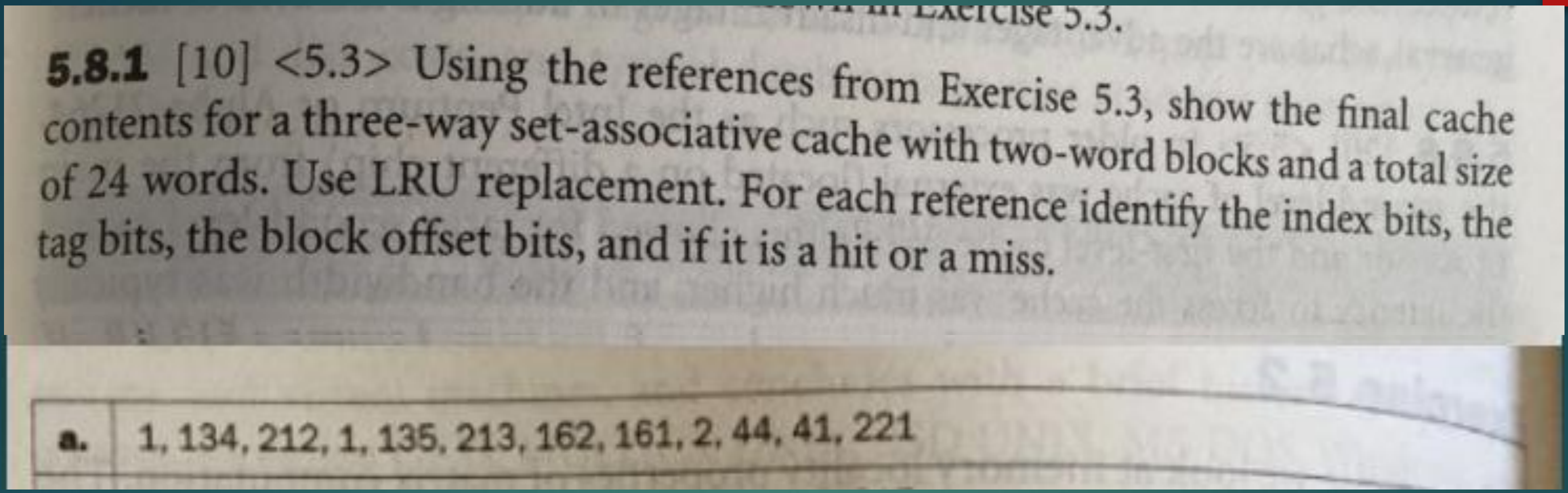
5.7.6



a.	P1 with L2 cache: CPI = 5.76. P2: CPI = 4.05. P2 is still faster than P1 even with an L1 cache
b.	P1 with L2 cache: CPI = 2.01. P2: CPI = 1.79. P2 is still faster than P1 even with an L1 cache

for (a), let r be the miss rate:
AMAT: 1*0.62 + 0.114*3.22 + 0.114*r*70 (ns)
[AMAT(P1) *0.62 + 0.62* 0.64]P1 ≤ [2.67]P2
solve the ‘r’ from this equation.

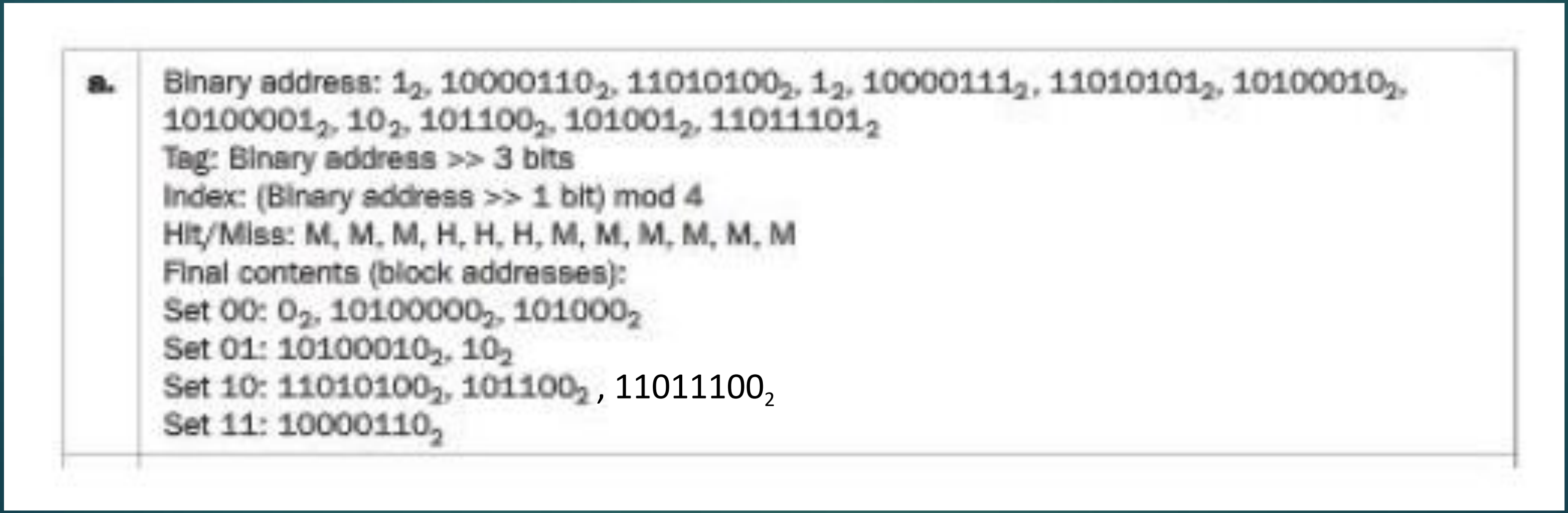
5.8.1



in Exercise 5.3.

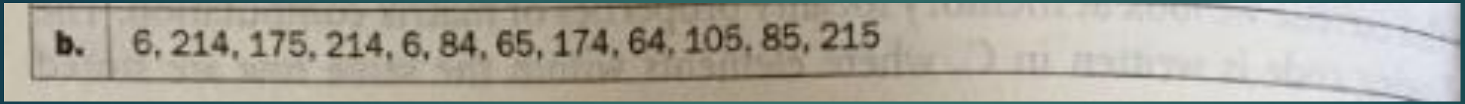
5.8.1 [10] <5.3> Using the references from Exercise 5.3, show the final cache contents for a three-way set-associative cache with two-word blocks and a total size of 24 words. Use LRU replacement. For each reference identify the index bits, the tag bits, the block offset bits, and if it is a hit or a miss.

a.	1, 134, 212, 1, 135, 213, 162, 161, 2, 44, 41, 221

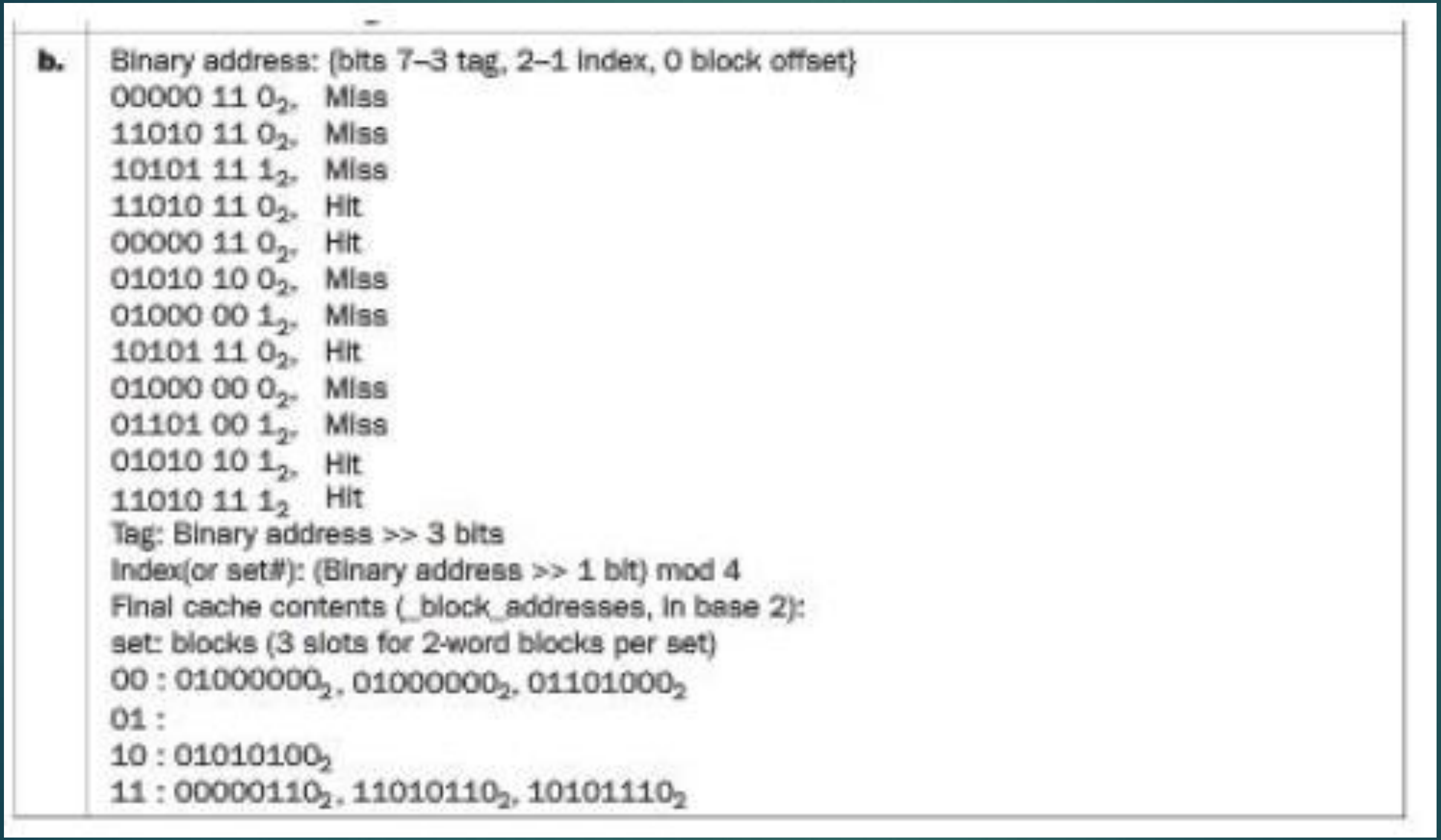


a.	Binary address: 1_2, 10000110_2, 11010100_2, 1_2, 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2 Tag: Binary address >> 3 bits Index: (Binary address >> 1 bit) mod 4 Hit/Miss: M, M, M, H, H, H, M, M, M, M, M, M Final contents (block addresses): Set 00: 0_2, 10100000_2, 101000_2 Set 01: 10100010_2, 10_2 Set 10: 11010100_2, 101100_2 , 11011100_2 Set 11: 10000110_2

5.8.1



b.	6, 214, 175, 214, 6, 84, 65, 174, 64, 105, 85, 215



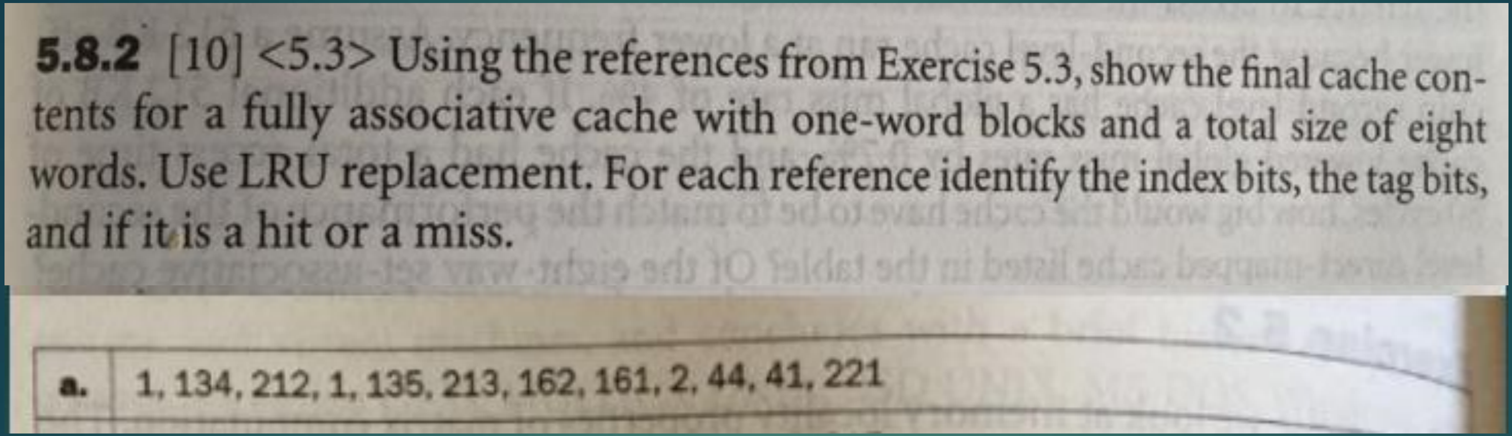
b. Binary address: {bits 7–3 tag, 2–1 index, 0 block offset}

$00000\ 11\ 0_2$, Miss
$11010\ 11\ 0_2$, Miss
$10101\ 11\ 1_2$, Miss
$11010\ 11\ 0_2$, Hit
$00000\ 11\ 0_2$, Hit
$01010\ 10\ 0_2$, Miss
$01000\ 00\ 1_2$, Miss
$10101\ 11\ 0_2$, Hit
$01000\ 00\ 0_2$, Miss
$01101\ 00\ 1_2$, Miss
$01010\ 10\ 1_2$, Hit
$11010\ 11\ 1_2$ Hit

Tag: Binary address >> 3 bits
Index(or set#): (Binary address >> 1 bit) mod 4
Final cache contents (_block_addresses, in base 2):
set: blocks (3 slots for 2-word blocks per set)

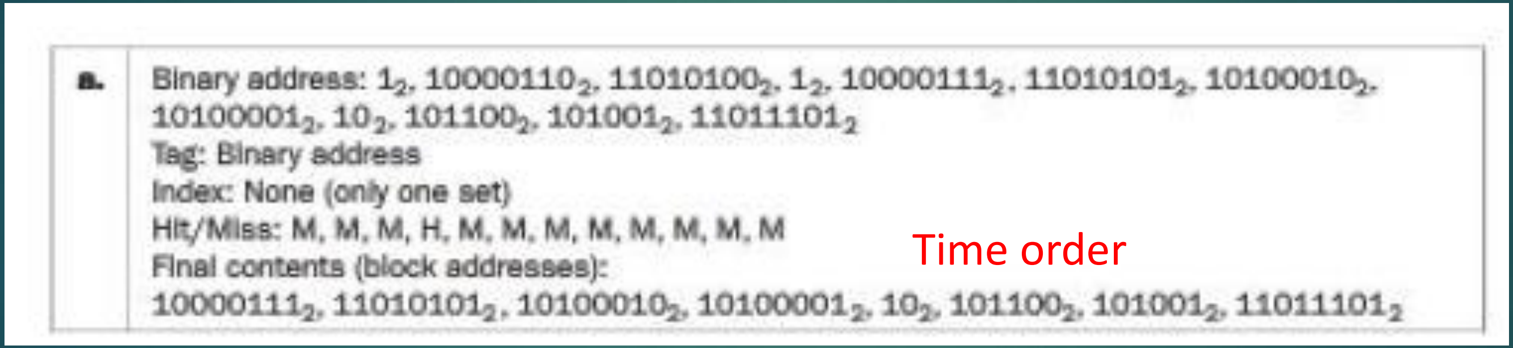
00 : 01000000_2, 01000000_2, 01101000_2
01 :
10 : 01010100_2
11 : 00000110_2, 11010110_2, 10101110_2

5.8.2



5.8.2 [10] <5.3> Using the references from Exercise 5.3, show the final cache contents for a fully associative cache with one-word blocks and a total size of eight words. Use LRU replacement. For each reference identify the index bits, the tag bits, and if it is a hit or a miss.

a.	1, 134, 212, 1, 135, 213, 162, 161, 2, 44, 41, 221

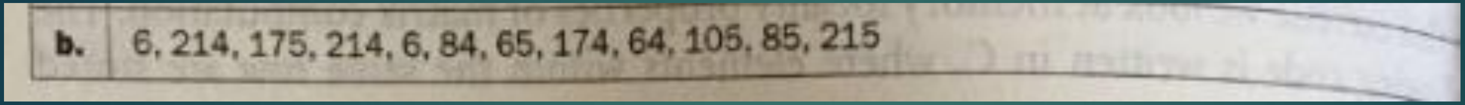


a.	Binary address: 1_2, 10000110_2, 11010100_2, 1_2, 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2 Tag: Binary address Index: None (only one set) Hit/Miss: M, M, M, H, M, M, M, M, M, M, M, M Final contents (block addresses): 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2

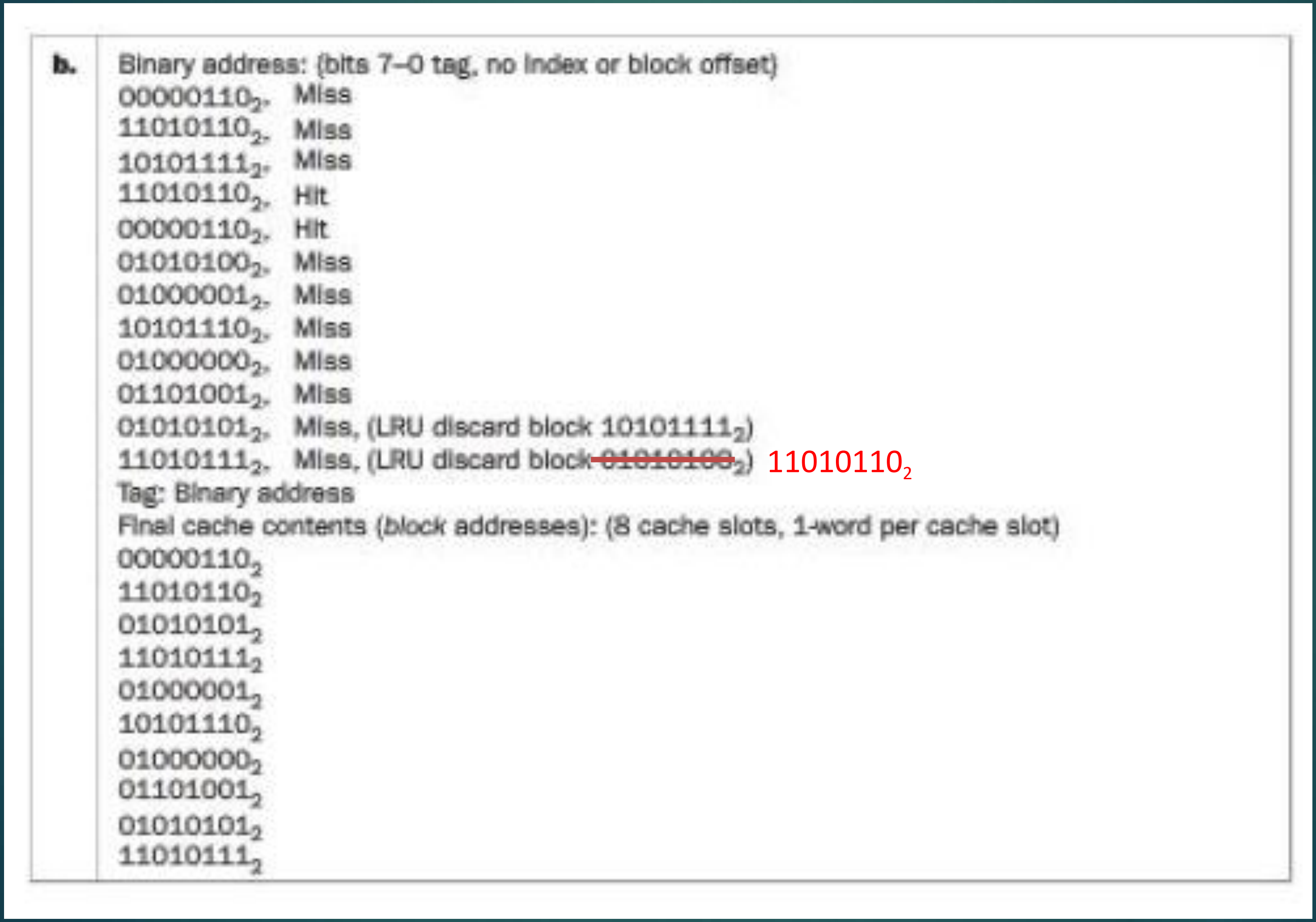
Time order

Block order: 221, 44, 41, 135, 213, 162, 161, 2

5.8.2



b.	6, 214, 175, 214, 6, 84, 65, 174, 64, 105, 85, 215



b. Binary address: (bits 7–0 tag, no index or block offset)

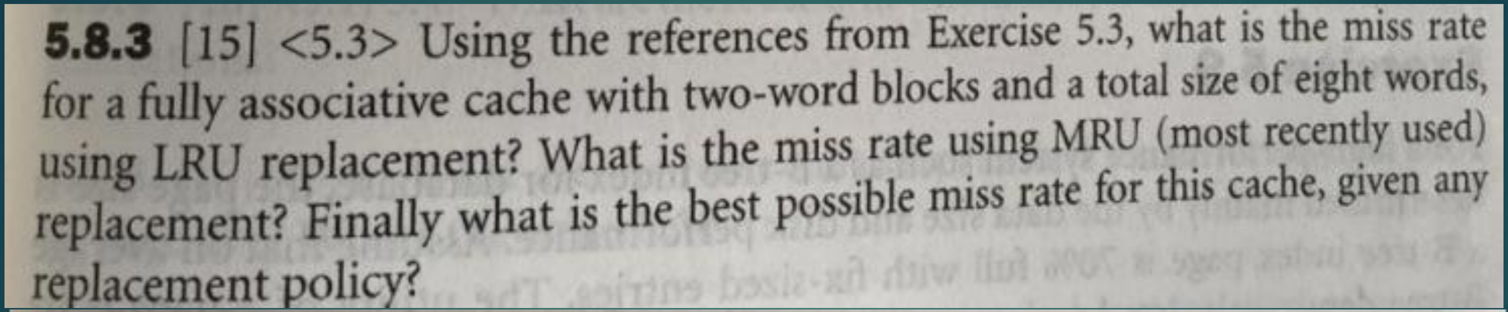
00000110_2. Miss
11010110_2. Miss
10101111_2. Miss
11010110_2. Hit
00000110_2. Hit
01010100_2. Miss
01000001_2. Miss
10101110_2. Miss
01000000_2. Miss
01101001_2. Miss
01010101_2. Miss, (LRU discard block 10101111_2)
11010111_2. Miss, (LRU discard block ~~01010100~~$_2$) 11010110_2

Tag: Binary address

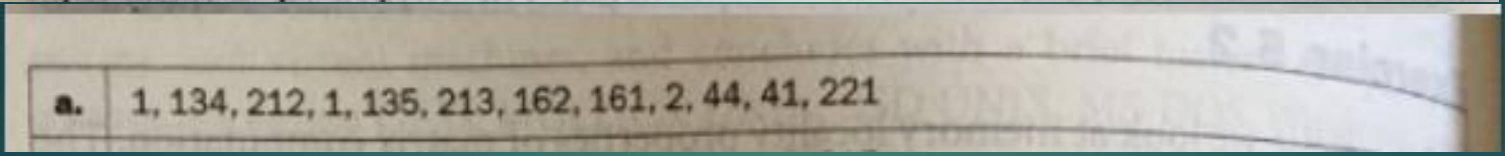
Final cache contents (*block addresses*): (8 cache slots, 1-word per cache slot)

00000110_2
11010110_2
01010101_2
11010111_2
01000001_2
10101110_2
01000000_2
01101001_2
01010101_2
11010111_2

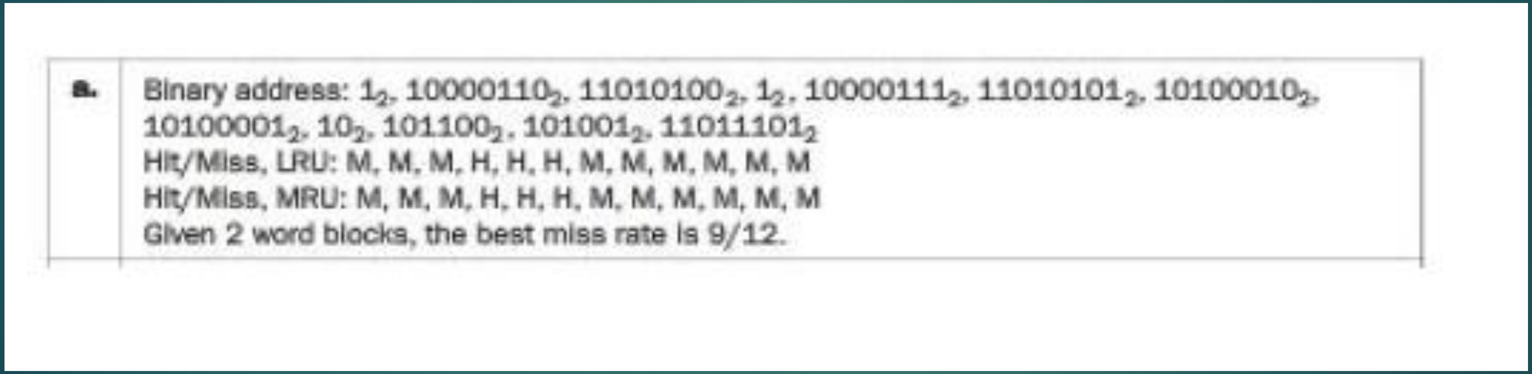
5.8.3



5.8.3 [15] <5.3> Using the references from Exercise 5.3, what is the miss rate for a fully associative cache with two-word blocks and a total size of eight words, using LRU replacement? What is the miss rate using MRU (most recently used) replacement? Finally what is the best possible miss rate for this cache, given any replacement policy?

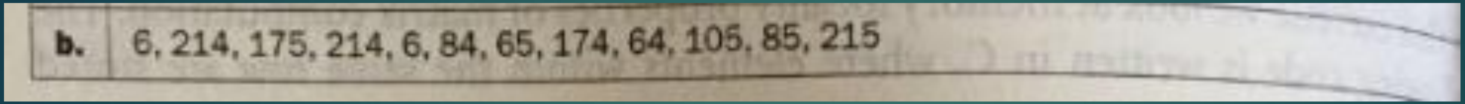


a.	1, 134, 212, 1, 135, 213, 162, 161, 2, 44, 41, 221

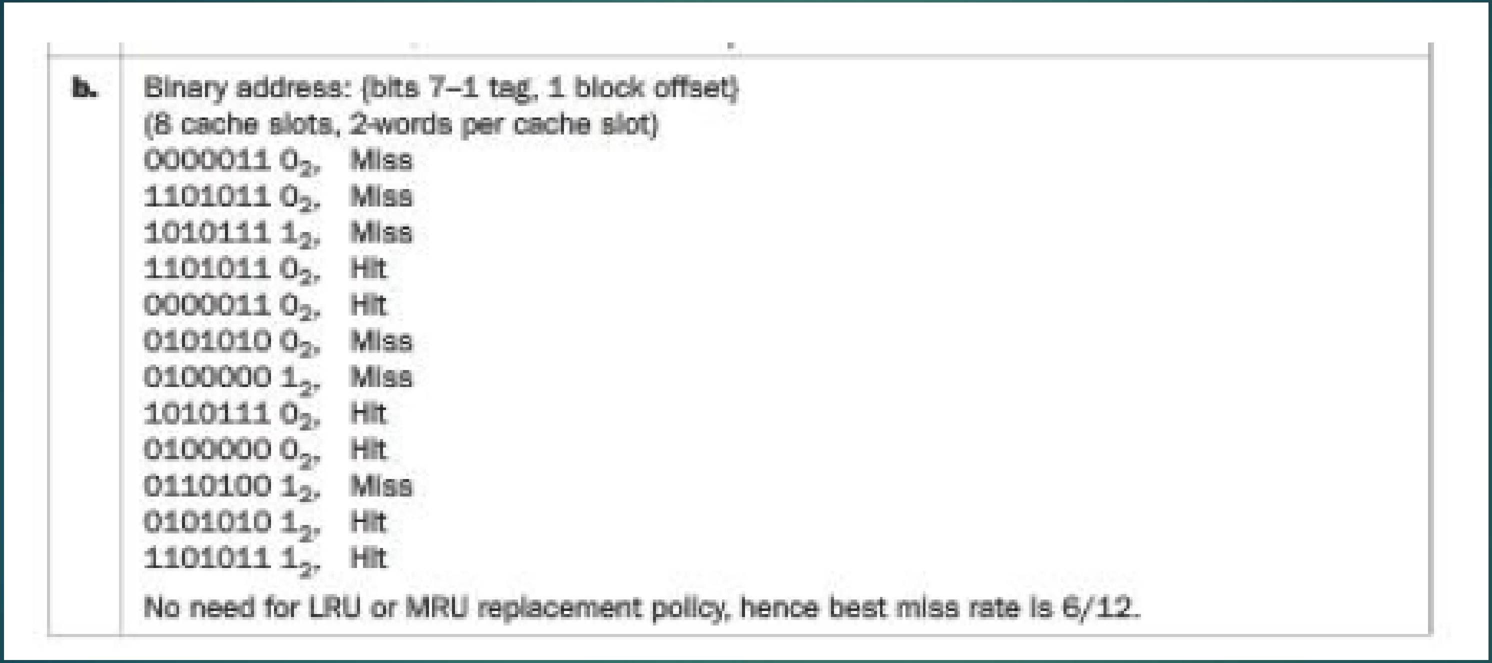


a.	Binary address: 1_2, 10000110_2, 11010100_2, 1_2, 10000111_2, 11010101_2, 10100010_2, 10100001_2, 10_2, 101100_2, 101001_2, 11011101_2
	Hit/Miss, LRU: M, M, M, H, H, H, M, M, M, M, M, M
	Hit/Miss, MRU: M, M, M, H, H, H, M, M, M, M, M, M
	Given 2 word blocks, the best miss rate is 9/12.

5.8.3

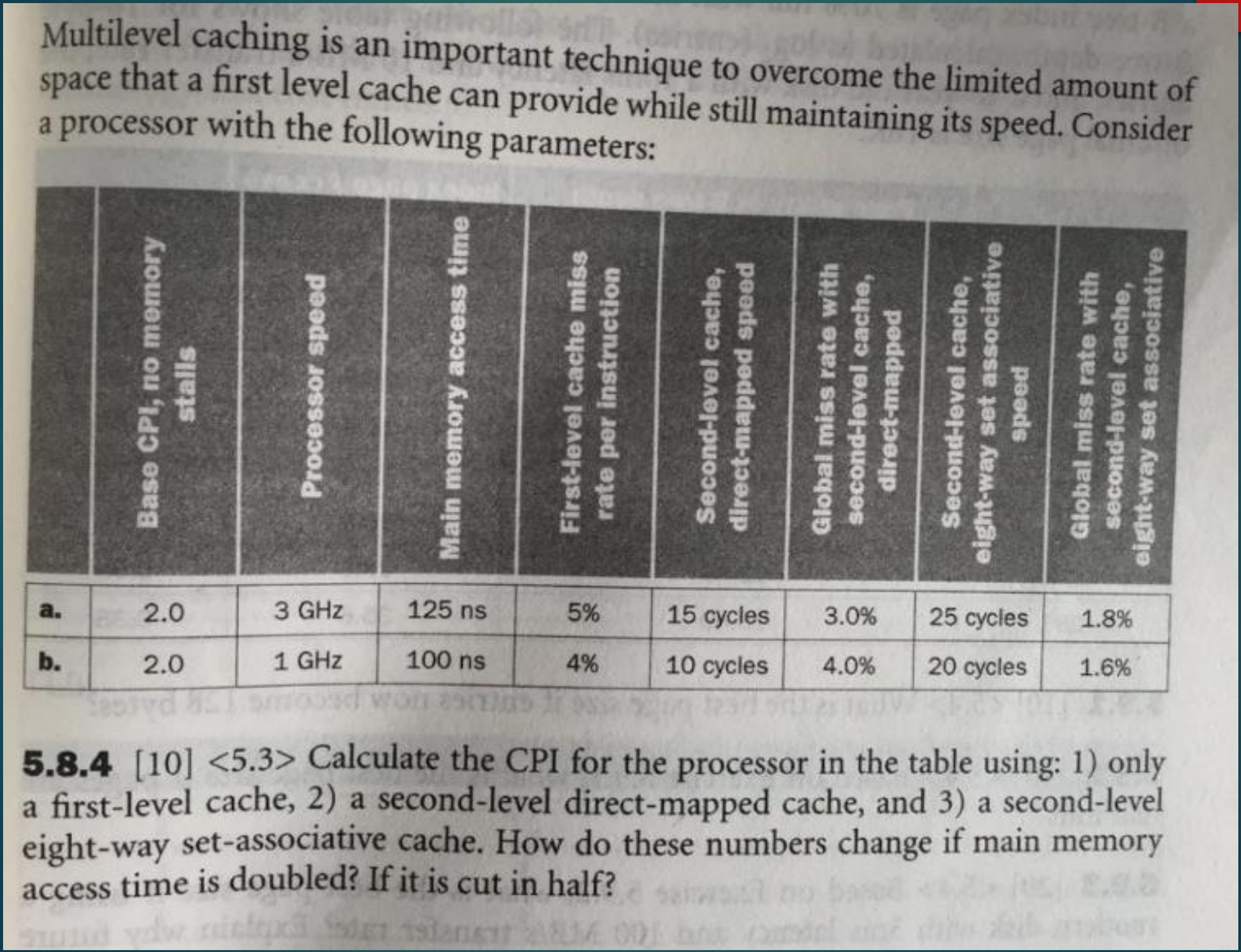


b.	6, 214, 175, 214, 6, 84, 65, 174, 64, 105, 85, 215



b.	Binary address: (bits 7–1 tag, 1 block offset) (8 cache slots, 2-words per cache slot) 0000011 0_2, Miss 1101011 0_2, Miss 1010111 1_2, Miss 1101011 0_2, Hit 0000011 0_2, Hit 0101010 0_2, Miss 0100000 1_2, Miss 1010111 0_2, Hit 0100000 0_2, Hit 0110100 1_2, Miss 0101010 1_2, Hit 1101011 1_2, Hit No need for LRU or MRU replacement policy, hence best miss rate is 6/12.

5.8.4

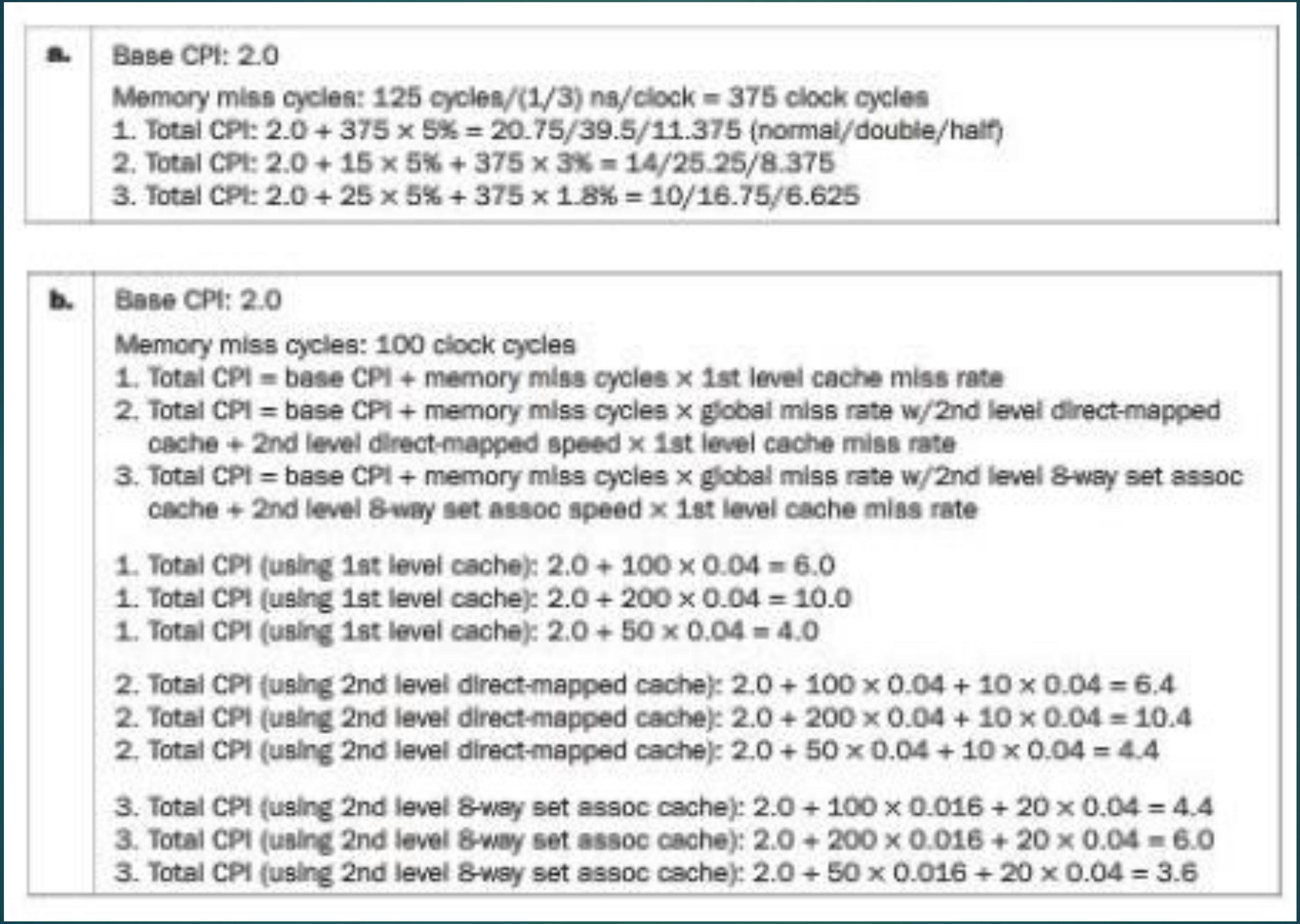


Multilevel caching is an important technique to overcome the limited amount of space that a first level cache can provide while still maintaining its speed. Consider a processor with the following parameters:

	Base CPI, no memory stalls	Processor speed	Main memory access time	First-level cache miss rate per instruction	Second-level cache, direct-mapped speed	Global miss rate with second-level cache, direct-mapped	Second-level cache, eight-way set associative speed	Global miss rate with second-level cache, eight-way set associative
a.	2.0	3 GHz	125 ns	5%	15 cycles	3.0%	25 cycles	1.8%
b.	2.0	1 GHz	100 ns	4%	10 cycles	4.0%	20 cycles	1.6%

5.8.4 [10] <5.3> Calculate the CPI for the processor in the table using: 1) only a first-level cache, 2) a second-level direct-mapped cache, and 3) a second-level eight-way set-associative cache. How do these numbers change if main memory access time is doubled? If it is cut in half?

5.8.4



a. Base CPI: 2.0

Memory miss cycles: 125 cycles/(1/3) ns/clock = 375 clock cycles

1. Total CPI: 2.0 + 375 × 5% = 20.75/39.5/11.375 (normal/double/half)
2. Total CPI: 2.0 + 15 × 5% + 375 × 3% = 14/25.25/8.375
3. Total CPI: 2.0 + 25 × 5% + 375 × 1.8% = 10/16.75/6.625

b. Base CPI: 2.0

Memory miss cycles: 100 clock cycles

1. Total CPI = base CPI + memory miss cycles × 1st level cache miss rate
2. Total CPI = base CPI + memory miss cycles × global miss rate w/2nd level direct-mapped cache + 2nd level direct-mapped speed × 1st level cache miss rate
3. Total CPI = base CPI + memory miss cycles × global miss rate w/2nd level 8-way set assoc cache + 2nd level 8-way set assoc speed × 1st level cache miss rate

1. Total CPI (using 1st level cache): 2.0 + 100 × 0.04 = 6.0
1. Total CPI (using 1st level cache): 2.0 + 200 × 0.04 = 10.0
1. Total CPI (using 1st level cache): 2.0 + 50 × 0.04 = 4.0

2. Total CPI (using 2nd level direct-mapped cache): 2.0 + 100 × 0.04 + 10 × 0.04 = 6.4
2. Total CPI (using 2nd level direct-mapped cache): 2.0 + 200 × 0.04 + 10 × 0.04 = 10.4
2. Total CPI (using 2nd level direct-mapped cache): 2.0 + 50 × 0.04 + 10 × 0.04 = 4.4

3. Total CPI (using 2nd level 8-way set assoc cache): 2.0 + 100 × 0.016 + 20 × 0.04 = 4.4
3. Total CPI (using 2nd level 8-way set assoc cache): 2.0 + 200 × 0.016 + 20 × 0.04 = 6.0
3. Total CPI (using 2nd level 8-way set assoc cache): 2.0 + 50 × 0.016 + 20 × 0.04 = 3.6