Exercises for Week 10
Textbook Sections 5.4 and 5.10
Sample Solutions

Question 1

Textbook Exercise 5.10 <§5.4>

In this exercise, we will look at the different ways capacity affects overall performance. In general, cache access time is proportional to capacity. Assume that main memory accesses take 70 ns and that 36% of all instructions access data memory. 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
P1	2 KB	8.0%	0.66 ns
P2	4 KB	6.0%	0.90 ns

a. Assuming that the L1 hit time determines the cycle times for P1 and P2, what are their respective clock rates?

P1: $f_c = 1/T_c = 1/0.66\text{ns} = 1.52\text{GHz}$.

P2: $f_c = 1/T_c = 1/0.90\text{ns} = 1.11\text{GHz}$.

b. What is the Average Memory Access Time for each of P1 and P2 (in cycles)?

P1: Miss penalty = 70ns/0.66ns = 106 cycles.
AMAT = $1 + 0.08 \times 106 = 9.48$ cycles.

P2: Miss penalty = 70ns/0.90ns = 78 cycles.
AMAT = $1 + 0.06 \times 78 = 5.67$ cycles.

c. Assuming a base CPI of 1.0, what is the total CPI for each of P1 and P2? Which processor is faster? (When we say a "base CPI of 1.0", we mean that instructions complete in one cycle, unless either the instruction access or the data access causes a cache miss.)

P1: Instruction fetch stall cycles per instruction = $0.08 \times 106 = 8.48$.
Data access stall cycles per instruction = $0.36 \times 0.08 \times 106 = 3.05$.
CPI = $1 + 8.48 + 3.05 = 12.5$.
Time per instruction = $12.5 / 1.52\times10^9 = 8.27$ns.

P2: Instruction fetch stall cycles per instruction = $0.06 \times 78 = 4.68$.
Data access stall cycles per instruction = $0.36 \times 0.06 \times 78 = 1.68$.
CPI = $1 + 4.68 + 1.68 = 7.36$.
Time per instruction = $7.36 / 1.11\times10^9 = 6.62$ns.

So P2 is faster, even though it has a slower clock. The reduced miss rate of the larger cache compensates, leading to fewer stalls per instruction.

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. [Note that the L2 miss rate here is different from that in the exercise in the textbook. The value there is unrealistically high, and is probably an error.)]

L2 size	L2 miss rate	L2 hit time
1 MB	5%	5.62 ns

d. What is the AMAT for P1 with the addition of an L2 cache? Is the AMAT better or worse with the L2 cache?

AMAT for the L2 cache is 5.62ns + 0.05 × 70ns = 9.12ns (13.8 cycles). This is the miss penalty for the L1 cache, so the new AMAT for P1 is 0.66 + 0.08 × 9.12 = 1.39ns, or 2.1 cycles. This is significantly better than with the L1 cache alone.

e. Assuming a base CPI of 1.0 without any memory stalls, what is the total CPI for P1 with the addition of an L2 cache?

Instruction fetch stall cycles per instruction = 0.08 × 13.8 = 1.10.
Data access stall cycles per instruction = 0.36 × 0.08 × 13.8 = 0.40.
CPI = 1 + 1.10 + 0.40 = 2.50.
Time per instruction = 2.50 / 1.52×10^9 = 1.64ns.

f. What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P1 without an L2 cache?

P1 with an L2 cache is faster than P1 without an L2 cache if the AMAT of the L2 cache is less than that of main memory. The AMAT of the L2 cache is 5.62ns + m_2 × 70ns, where is m_2 is the L2 miss rate.

$5.62\text{ns} + m_2 \times 70\text{ns} < 70\text{ns} \Rightarrow m_2 < (70\text{ns} - 5.62\text{ns}) / 70\text{ns} \Rightarrow m_2 < 92\%$

A program have to have astonishingly poor locality to exhibit a miss rate worst than 92%. The only kind of program that would behave like this is a synthetic test case deliberately constructed to have pathologically bad memory reference behaviour.

g. What would the L2 miss rate need to be in order for P1 with an L2 cache to be faster than P2 without an L2 cache?

P1 with an L2 cache is faster than P2 without an L2 cache if the time per instruction on P1 is less than that on P2, namely, 6.62ns. We can use our working from parts (d) and (e) in reverse.

Total CPI = time per instruction × clock rate = 6.62ns × 1.52×10^9 = 10.06.

Stall cycles per instruction = total CPI – base CPI = 10.06 - 1.0 = 9.06.

Stall cycles per instruction
= instruction fetch stalls/instruction + data access stalls/instruction
= 0.08 × L2-AMAT + 0.36 × 0.08 × L2-AMAT = 0.1088 × L2-AMAT = 9.06
=> L2-AMAT = 83.29 cycles = 54.97ns.

$\text{L2-AMAT} = 5.62\text{ns} + m_2 \times 70\text{ns} = 54.97\text{ns} \Rightarrow m_2 = (54.97\text{ns} - 5.62\text{ns}) / 70\text{ns} \Rightarrow m_2 = 71\%$

Again, a program have to have astonishingly poor locality to exhibit a miss rate worst than 71%.

Question 2

Consider a 2-way set-associative data cache of size 128Kbytes with block size of 16 bytes. The cache uses LRU replacement strategy (assuming initially that entry 0 in a set is least recently used) and write-back/write-allocate write strategy.

a. Show how a 32-bit data address is subdivided into tag, index and offset fields, giving the number of bits in each field.

The block size is 16 bytes, so the offset field is $\log_2 16 = 4$ bits.

The number of cache entries is $128K/16 = 2^{17}/2^4 = 2^{13} = 8192$.

Each set has 2 entries, so the number of sets = 4096, and the index field is $\log_2 4096 = 12$ bits.

The tag field is the remainder of the address, 16 bits.

The address is subdivided into the fields as follows:

tag	index	offset

Note: These sizes are cunningly selected, to make the rest of the exercise easier, so that an address expressed in hexadecimal nicely partitions into the fields. The offset is the least-significant hex digit, the index is the next three hex digits, and the tag is the leftmost four hex digits.

b. Starting from power on, the following byte-addressed cache references by the CPU are recorded:

Read	0x20080128
Read	0x2008012C
Read	0x20080130
Write	0xA1150120
Read	0xA1150128
Write	0x20080130
Read	0x07F00124
Write	0x96880120

Trace the accesses in the cache, showing for each access:

- the address offset, index and tag value
- whether the access is a hit or a miss
- any memory operations required

Following is the trace of the cache accesses, also showing the state of the cache entry and the set LRU state after completion of the access:

CPU		Address						Cache entry 0 in set			Cache entry 1 in set			
Op	Address	Offset	Index	Tag	Hit/ Miss	Entry	Memory op	Valid	Tag	Dirty	Valid	Tag	Dirty	Set LRU
Read	0x20080128	8	012	2008	Miss	0	Read	1	2008	0				1
Read	0x2008012C	C	012	2008	Hit	0	–	1	2008	0				1
Read	0x20080130	0	013	2008	Miss	0	Read	1	2008	0				1
Write	0xA1150120	0	012	A115	Miss	1	Read	1	2008	0	1	A115	1	0
Read	0xA1150128	8	012	A115	Hit	1	–	1	2008	0	1	A115	1	0
Write	0x20080130	0	013	2008	Hit	0	–	1	2008	1				1

CPU		Address						Cache entry 0 in set			Cache entry 1 in set			
Op	Address	Offset	Index	Tag	Hit/ Miss	Entry	Memory op	Valid	Tag	Dirty	Valid	Tag	Dirty	Set LRU
Read	0x07F00124	4	012	07F0	Miss	0	Read	1	07F0	0	1	A115	1	1
Write	0x96880120	0	012	9688	Miss	1	Write Read	1	07F0	0	1	9688	1	0

Question 3

Suppose a system has two processors P1 and P2, each with a write-back cache using an invalidation coherence protocol, as described in Figure 5.42. When a cache intervenes on another cache's memory read, the intervening cache also updates memory. Variables X[0] and X[1], both initially 0 in memory, are contained within the same block. The processors access the variables in the following order:

P1 reads X[0]
P2 reads X[1]
P1 writes X[0] = 9
P1 reads X[0]
P2 reads X[1]
P1 reads X[0]

Trace the bus actions performed in each cache to maintain coherence, showing the contents of the block in each cache and in memory after each access.

The actions and block changes are:

P1 cache			P2 cache			Memory
Access	Bus action	Content	Access	Bus action	Content	Content
						0, 0
P1 reads X[0]	Read miss for block	0, 0				0, 0
		0,0	P2 reads X[1]	Read miss for block	0,0	0, 0
P1 writes X[0] = 9	Invalidate for block	9, 0				0, 0
P1 reads X[0]	–	9, 0				0, 0
	Cancel memory response, Write block to memory, Respond to P2's read miss	9, 0	P2 reads X[1]	Read miss for block	9, 0	9, 0
P1 reads X[0]	–	9, 0				

Note: This example illustrates false sharing. The two variables X[0] and X[1] are not actually shared between the two programs; P1 only accesses X[0], and P2 only accesses X[1]. However, since the variables are in the same block, they are both subject to each coherence operation on that block. So far as cache coherence is concerned, it is the block that is shared between P1 and P2, not specifically any variable within the block.