CSC/ECE 506: Architecture of Parallel Computers
Sample Test 2 with Answers

This was a 120-minute open-book test. You were allowed to use the textbook and any course notes that you might have. You could also use your laptop, but you were not allowed to communicate with any other student, by electronic or other means, during the test.

You were to answer five of the six questions. Each question was worth 20 points. If you answered all six questions, your five highest scores counted.

Question 1. *[Average score 18.7] (1 point per blank)* Here is a system with 3 processors using the MESI protocol. Each line holds a single integer value (e.g., block size of 4 bytes).

"– " indicates that the contents of a line are unknown. Initially, all caches are empty.

Memory contents	
Address	**Data**
1000	1
3000	3

Fill in the blank spaces.

Operation	**Processor 1**		**Processor 2**		**Processor 3**		**Memory contents**
	Value	**State**	**Value**	**State**	**Value**	**State**	
P2 reads from address 2000	–	–	3	E	–	–	3
P2 writes 5 to address 2000	–	–	5	M	–	–	3
P3 reads address 2000	–	–	5	S	5	S	5
P1 writes 7 to address 1000	7	M	–	–	–	–	1
P3 reads address 1000	7	S	–	–	7	S	7
P1 writes 6 to address 1000	6	M	–	–	7	I	7
P2 reads from address 1000	6	S	6	S	7	I	6
P3 writes 4 to address 2000	–	–	5	I	4	M	5
P1 writes 9 to address 2000	9	M	5	I	4	I	4
P1 writes 2 to address 2000	2	M	5	I	4	I	4
P2 writes 1 to address 2000	2	I	1	M	4	I	2

Question 2. *[Average score 17.1]* Consider a bus-based shared-memory system with four processors, P_1, P_2, P_3, and P_4, with individual caches. The sequence of accesses to location *r* in the shared memory is given below. Assume that all caches are initially empty.

1. P_1 writes value10 to location *r*.
2. P_4 reads from location *r*.
3. P_2 writes value 7 to location *r*.
4. P_3 reads from location *r*.
5. P_3 writes value 3 to location *r*.

(a) Using the Dragon update protocol, fill in the table below, showing the state transitions and bus transitions.

Answer: Answers you were to give are underlined.

	Instruction	State in P_1	P_2	P_3	P_4	Bus action	Data supplied by
1.	P_1 writes *r* = 10	M	-	-	-	BusRd	Memory
2.	P_4 reads *r*	Sm	-	-	Sc	BusRd	P_1 cache
3.	P_2 writes *r* = 7	Sc	Sm	-	Sc	BusRd, BusUpd	P_1 cache
4.	P_3 reads *r*	Sc	Sm	Sc	Sc	BusRd	P_2 cache
5.	P_3 writes *r* = 3	Sc	Sc	Sm	Sc	BusUpd	P_3 cache

(b) Each processor has a cache size of 4K bytes with 128 cache lines. How many bytes are in each line?

Answer:
Cache size is 4K = 2^{12} bytes
Number of cache lines = 128 = 2^7
So, each cache block or line size = $2^{12} / 2^7$
= 2^5 bytes
= 32 bytes

(c) Assume that the above sequence of instructions is repeated 10 times. Address and command in any message occupy 8 bytes. Of course, a cache miss needs to transfer the cache line as well as address and command. The total length of an update message is 16 bytes. Calculate the cost for update protocol in terms of cache misses and bus traffic at the end of 10 iterations.
Note: Assume no cache misses, other than those caused by executing the above instructions.

Answer: A cache miss (BusRd) requires 40 bytes to be transferred across the bus. A BusUpd transfers 16 bytes.

For the first iteration, the first four operations incur a cache miss
After the first iteration there are no more misses.
So, the *total number of cache misses* after all the iterations = 4.

Each time any processor writes a value, it generates an update.
So, for each iteration there are a total of 3 updates.

Total number of updates = number of iterations $\times$ updates on each iteration = $10 \times 3 = 30$.
Given that each update transfers 16 bytes, the # of bytes transferred across the bus for these updates = $16 \times 30 = 480$ bytes.

Total bus traffic for all the iterations =

(Number of cache misses × bytes per cache miss) + bytes transferred across the bus for the updates.

= (4 × 40) + 480
= 160 + 480
= 640 bytes.

Question 3. *[Average score 12.7]* This question concerns inclusion policies. Suppose we have a cache that has an equal number of L1 and L2 sets. The L1 cache is 2-way, and the L2 is 4-way associative. Let's focus first on Set 0 in both the L1 and the L2.

(a) *(8 points)* Assuming that this is an exclusive cache, fill in the mising block numbers (A–G) in the table below. If a line is empty, write a dash ("—").

Ref. #	Block refer-enced	L1 cache (Set 0)		Exclusive L2 cache (Set 0)			
		MRU line	LRU line	MRU line	…		LRU line
1	A	A	—	—	—	—	—
2	B	B	A	—	—	—	—
3	C	C	B	A	—	—	—
4	A	A	C	B	—	—	—
5	D	D	A	C	B	—	—
6	E	E	D	A	C	B	—
7	F	F	E	D	A	C	B
8	B	B	F	E	D	A	C
9	G	G	B	F	E	D	A

(b) *(2 point each)* What is the number of the first reference where—

- the exclusive cache would eject a block from the L2? *Answer:* Exclusive would eject when the 7th block from set 0 Is referenced. That is G, at reference 9.
- an *inclusive* cache would eject a block from the L2? *Answer:* Inclusive would eject when the 5th block from set 0 Is referenced. That is E, at reference 6.
- a *NINE* cache would eject a block from the L2? *Answer:* NINE keeps A in the L2 even after it is re-referenced. But before any ejection from the L2 occurs, A gets ejected from the L1 a second time, it gets ejected from the L1 a second time. So NINE first ejects a block from the L2 at the same time that exclusive would, i.e., at reference 9.
- the contents of the L2 differ between an exclusive cache and a NINE cache? *Answer:* At reference 4, when A is re-referenced, it stays in the L2 with NINE, but not with exclusive.
- one of the caches (inclusive, exclusive, or NINE) would have an L2 miss while another has an L2 hit? *Answer:* Reference 8, where B is referenced. This will be an L2 hit unless the policy is inclusive.

(c) *(2 points)* Suppose that the line size is 128 bytes and the L1 has 2^8 sets. What is the *minimum* possible distance between the first address in A and the first address in B? *Answer:* Since A and B both map to Set 0, their 8-bit index field must be the 0. The first address in each block has a 7-bit offset field of 0. So the least significant 15 bits of both addresses are all 0. Thus, A and B must be at least 2^{15} bytes apart.

Question 4. *(3 points each, except 5 points for part (f)) [Avg. score 10.6]* (a) In the TTSL implementation, what would happen if the "**`ld R1, &lockvar`**" were replaced with a "**`t&s R1, &lockvar`**" (and no other changes were made to the original code)?

Answer: No process would be able to enter the critical section. Suppose there is only one process in the system. It begins with a test&set, which causes **`lockvar`** to change to **`1`**. Then when it gets to the second loop, **`lockvar`** will already be **`1`**, causing it to loop back to the first instruction. The first process cannot enter, and, since **`lockvar`** is now **`1`**, neither can any other process.

(b) In the TTSL implementation, what would happen if the "**`t&s R1, &lockvar`**" were replaced with a "**`ld R1, &lockvar`**" (and no other changes were made to the original code)?

Answer: The code would be incorrect. Two (or more) processors could simultaneously load the value of **`&lockvar`**, find it to be 0, and proceed past the first **`bnz`**. Then they could both find **`&lockvar`** to be 0 a second time, and enter the critical section.

(c) Suppose that *both* the change of part (a) and part (b) were made, effectively causing the **`ld`** and **`t&s`** of **`&lockvar`** to change place. What would happen then?

Answer: This is similar to part (a). The first process to execute this code tests&sets the **`lockvar`**. After that, it tests the value of **`lockvar`**, fetching it with an ordinary load instruction. But the value is already **`1`**, so the first process loops back to the top. No other process can enter, so **`lockvar`** cannot be reset to 0, so no process can ever enter.

(d) In the code for the LL/SC implementation, what value is in **`LINKREG`** when the **`lock`** procedure returns?

Answer: Assuming that no context switch occurs between execution of the **`SC`** instruction and the **`ret`** instruction, **`LINKREG`** holds the address of **`lockvar`**.

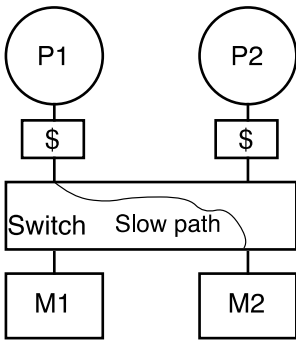
(e) One performance advantage of LL/SC vs. TTSL is that a failed SC does not generate a bus transaction, whereas *all* test&sets generate bus transactions. Explain why this is true.

Answer: Bus transactions are caused by writes to shared variables. All test&sets do a set, i.e., a write. Store-conditional only does a store if the condition is true, not if it fails.

(f) In each of the first three implementations of locking (test&set, TTSL, LL/SC), how many cache lines are invalidated in the example code?

Answer: For test&set, 8 invalidations occur. For TTSL and LL/SC, only five invalidations occur.

Question 5. *[Avg. score 13.4]* A two-processor system is shown at the right. Notice that the path from P1 to M2 is congested at the moment. Assume *X* and *Y* are initialized to 0. "←" means assignment.



(a) *(5 points)* Will there be a consistency problem if the following sequence of events occurs? Explain.

Time	P1 executes …	P2 executes …	Value of *X* in M2	Value of *Y* in M1
1	$X \leftarrow 1$	**while** ($Y = 0$) …	0	0
2	$Y \leftarrow 1$	**while** ($Y = 0$) …	0	0
3		**while** ($Y = 0$) …	0	1
4		$Y \leftarrow X + 1$	0	1
5			1	1

Answer: There will be a consistency problem. This is because of the slow path from P1 to M2. Processor P2 reads the updated *Y* value of 1 before *X* has been updated. Even though *X* was written before *Y*, P2 uses the old value of *X* in the calculation that it performs. *X*'does not get updated until after the calculation done on *Y*. This is a consistency problem because the writes to multiple locations are not seen in an order that makes sense for this program.

(b) *(15 points)* Suppose the Dragon protocol is in use. Give the state of the lines containing *X* and *Y* in the caches of both processors at the end of each statement. Also tell whether or not each variable is up to date in memory.

Time	P1 executes …	P2 executes …	State of *X* in P1's cache	State of *Y* in P1's cache	State of *X* in P2's cache	State of *Y* in P2's cache	*X* up to date in mem-ory?	*Y* up to date in mem-ory?
1	$X \leftarrow 1$	**while** ($Y = 0$) …	M	–	–	E	No	Yes
2	$Y \leftarrow 1$	**while** ($Y = 0$) …	M	Sm	–	Sc	No	No
3		**while** ($Y = 0$) …	M	Sm	–	Sc	No	No
4		$Y \leftarrow X + 1$	Sm	Sc	Sc	Sm	Yes	No

Question 6. *[Avg. score 19.2]* Suppose that two processors execute the following code. All variables start off equal to 0.

P_1		P_2	
1a	`A = 1`	2a	`b = 2`
1b	`c = 1`	2b	`a = A`
1c	`b = 1`	2c	`print b, c`
1d	`print a`		

Under sequential consistency, which of the twelve possible combinations of values *a*, *b*, and *c* may be printed?

(a) *(10 points)* For each combination, tell whether or not it is possible.

(b) *(10 points)* For at least *four* of the combinations that are possible, give a sequence of statement executions that would print it. For example, the sequence

1a, 1b, 1c, 1d, 2a, 2b, 2c

will cause the values *a* = 0, *b* = 2, *c* = 1 to be printed.

Answer: It is fairly easy to see that $b \neq 0$, because both P_1 and P_2 set *b* to a non-zero value, and at least one must be executed before *b* is printed. Also, *b* = 1 and *c* = 0 is impossible, because *c* is set to 1 before *b* is, so if *b* is 1, *c* must be 1 too. The other combinations are possible. Values you were to fill in are underlined in the table below.

a	*b*	*c*	(a) Possible?	(b) If the combination is possible, give a sequence that prints these values
0	0	0	<u>No</u>	
0	0	1	<u>No</u>	
0	1	0	<u>No</u>	
0	1	1	<u>Yes</u>	<u>2a, 2b, 1a, 1b, 1c, {2c, 1d}</u>
0	2	0	<u>Yes</u>	<u>2a, 2b, 2c, 1a, 1b, 1c, 1d</u>
0	2	1	Yes	1a, 1b, 1c, 1d, 2a, 2b, 2c
1	0	0	<u>No</u>	
1	0	1	<u>No</u>	
1	1	0	<u>No</u>	
1	1	1	<u>Yes</u>	<u>1a, 2a, 1b, 2b, 1c, {2c, 1d}</u>
1	2	0	<u>Yes</u>	<u>1a, 2a, 2b, 2c, 1b, 1c, 1d</u>
1	2	1	<u>Yes</u>	<u>1a, 1b, 1c, 2a, 2b, {2c, 1d}</u>

In general, if
a = 0, then we must have 1d → 2b or 2b → 1a.
a = 1, then we must have 1a → 2b → 1d.
b = 0 is impossible
b = 1, then we must have 2a → 1c → 2c.
b = 2, then we must have 1c → 2a or 2c → 1c.
c = 0, then we must have 2c → 1b.
c = 1, then we must have 1b → 2c