Lecture 15 Multicore Exercises

Q1:
Assuming write-back caches, private L1 data caches (L1Ds) (no L2), shared memory, and a MESI coherence policy, indicate in the cache tables what would be the values (V) of the variables X and Y and their coherency (C) state (either M, E, S or I) in the L1D's for a dual core processor with the following sequence of reads and writes. As shown in the table, the coherency (C) state in Step 0 (the initial coherency state) of both variables in both caches is I (invalid). If a value doesn't change, you may leave that entry blank.

Step 0: Initially, X =5, Y = 6 in the shared memory, processor caches are empty
Step 1: Core 1 reads X (from the shared memory)
Step 2: Core 2 reads X
Step 3: Core 1 writes X = 3
Step 4: Core 1 writes Y = 4
Step 5: Core 2 reads Y

Please fill in the following tables with data values and cache coherence states.

Core 1's L1D Cache (private)												
	Step 0		Step 1		Step 2		Step 3		Step 4		Step 5	
	V	C	V	C	V	C	V	C	V	C	V	C
X	--	I										
Y	--	I										

Core 2's L1D Cache (private)												
	Step 0		Step 1		Step 2		Step 3		Step 4		Step 5	
	V	C	V	C	V	C	V	C	V	C	V	C
X	--	I										
Y	--	I										

Shared Memory						
	Step 0	Step1	Step2	Step3	Step4	Step5
	V	V	V	V	V	V
X	5					
Y	6					

Solution:

Core 1's L1D Cache (private)												
	Step 0		Step 1		Step 2		Step 3		Step 4		Step 5	
	V	C	V	C	V	C	V	C	V	C	V	C
X	--	I	5	E	5	S	3	M	3	M	3	M
Y	--	I	--	I	--	I	--	I	4	M	4	S

Core 2's L1D Cache (private)												
	Step 0		Step 1		Step 2		Step 3		Step 4		Step 5	
	V	C	V	C	V	C	V	C	V	C	V	C
X	--	I	--	I	5	S	5	I	5	I	5	I
Y	--	I	--	I	--	I	--	I	--	I	4	S

Shared Memory						
	Step 0	Step1	Step2	Step3	Step4	Step5
	V	V	V	V	V	V
X	5	5	5	5	5	5
Y	6	6	6	6	6	4

Q2:
Sequential Consistency allows

I. Stores in program order to appear in different memory order
II. Loads in program order to appear in different memory order
III. A store followed by a load in program order to appear in different memory order
IV. A load followed by a load in program order to appear in different memory order

A. I
B. II
C. I & II
D. III
E. IV
F. None of I, II, III or IV

Answer: F

Q3:
Two threads (A and B) are concurrently running on a dual-core processor that implements a sequentially consistent memory model. Assume that the value at address (R10) is initialized to 0. The instruction `st immediate, (R10)` writes an immediate number into the memory address stored in R10.

```
Thread A (Core 1)
1: st 0x1, (R10)
2: ld R1,  (R10)
3: st 0x2, (R10)
4: ld R2,  (R10)
```

```
Thread B (Core 2)
1: st 0x3, (R10)
2: ld R3,  (R10)
3: st 0x4, (R10)
4: ld R4,  (R10)
```

List all possible values that can be stored in R3 after both threads have finished executing.

Solution:
Let A1 represent Thread A Line 1, A2 be Thread A Line 2, …Then we can write the instructions of the two threads as the following:

Thread A:	Thread B:
A1: (R10) = 0x1	B1: (R10) = 0x3
A2: R1 = (R10)	B2: R3 = (R10)
A3: (R10) = 0x2	B3: (R10) = 0x2
A4: R2 = (R10)	B4: R2 = (R10)

Sequential consistency means that the order within each thread is enforced, respectively, i.e., much ensure

A1 → A2 → A3 → A4
B1 → B2 → B3 → B4

But there can be various permutations between the two threads, e.g., A1 → A2 → A3 → A4 → B1 → B2 → B3 → B4 or A1→B1→ A2→ A3→ B2→ A4 → B3 → B4
In these two examples, if we focus on one of the threads, say Thread B in blue color, we still maintain the order B1 → B2 → B3 → B4; the same for Thread A.

<u>Solution 1:</u> The easiest way to think of the problem: no matter how to permutate, B3 and B4 must execute after B2, which is the only instruction that makes change to R3. Therefore, the value of R3 could get 0x1, 0x2, or 0x3, but cannot be 0x4.

<u>Solution 2:</u> Another way to solve the problem is to think about whether 0x1, 0x2, 0x3, or 0x4 can be stored into R3 by the instruction of B2:

To allow R3 = 0x1, we must have "A1 → B2" somewhere in our permutation, that's of course possible
To allow R3 = 0x2, we must have "A3 → B2", that's also possible

To allow R3 = 0x3, we must have “B1 → B2”, that’s fine too
To allow R3 = 0x4, we must have “B3 → B2”, ah oh that’s against our sequential consistency requirement listed above “B1 → B2 → B3 → B4”

So, R3 could get 0x1, 0x2, or 0x3, but cannot be 0x4.

Solution 3: Or, you can list some permutations and see:

Thread A → Thread B:
A1: (R10) = 0x1 → A2: R1 = (R10) = 0x1 → A3: (R10) = 0x2 → A4: R2 = (R10) = 0x2 → B1: (R10) = 0x3 → B2: R3 = (R10) = 0x3 → … R3 will not change afterward
In this case, R3 = 0x3

Thread B → Thread A:
B1: (R10) = 0x3 → B2: R3 = (R10) = 0x3 → …R3 will not change afterward
In this case, R3 = 0x3

Interleave Thread A and Thread B:
B1: (R10) = 0x3 → A1: (R10) = 0x1 → B2: R3 = (R10) = 0x1 → …R3 will not change afterward
In this case, R3 = 0x1

Interleave Thread A and Thread B in another order:
A1: (R10) = 0x1 → B1: (R10) = 0x3 → A2: R1 = (R10) = 0x3 → A3: (R10) = 0x2 → B2: R3 = (R10) = 0x2 → …R3 will not change afterward
In this case, R3 = 0x2

No matter how we permutate, R3 won’t get 0x4