6 Cache Coherence [60 points]

We have a system with 4 byte-addressable processors. Each processor has a private 256-byte, direct-mapped, write-back L1 cache with a block size of 64 bytes. Coherence is maintained using the Illinois Protocol (MESI), which sends an invalidation to other processors on writes, and the other processors invalidate the block in their caches if *the block is present* (NOTE: On a write hit in one cache, a cache block in Shared state becomes Modified in that cache).

Accessible memory addresses range from 0x50000000 − 0x5FFFFFFF. Assume that the offset within a cache block is 0 for all memory requests. We use a snoopy protocol with a shared bus.

Cosmic rays strike the MESI state storage in your coherence modules, causing the state of a *single* cache line to instantaneously change to another state. This change causes an inconsistent state in the system. We show below the initial tag store state of the four caches, *after* the inconsistent state is induced.

Inital State

Cache 0		
	Tag	*MESI state*
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	I

Cache 1		
	Tag	*MESI state*
Set 0	0x522222	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 2		
	Tag	*MESI state*
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	S

Cache 3		
	Tag	*MESI state*
Set 0	0x5FF000	E
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

(a) [15 points] What is the inconsistency in the above initial state? Explain with reasoning.

Cache 2, Set 1 should be in S state. Or Cache 3, Set 1 should be in I state.

Explanation. If the MESI protocol performs correctly, it is *not* possible for the same cache line to be in S and E states in different caches.

(b) [20 points] Consider that, after the initial state, there are several paths that the program can follow that access different memory instructions. In b.1-b.4, we will examine whether the followed path can potentially lead to incorrect execution, i.e., an incorrect result.

b.1) [10 points] Could the following path potentially lead to incorrect execution? Explain.

order	**Processor 0**	**Processor 1**	**Processor 2**	**Processor 3**
1^{st}			ld 0x51110040	
2^{nd}	st 0x5FFFFF40			
3^{rd}				st 0x51110040
4^{th}		ld 0x5FFFFF80		
5^{th}		ld 0x51110040		
6^{th}		ld 0x5FFFFF40		

No.

Explanation. The 3^{rd} instruction (st 0x51110040 in Processor 3) will invalidate the same line in Processor 2, and the whole system will be back to a consistent state (only one valid copy of 0x51110040 in the caches). Thus, the originally-inconsistent state does not affect the architectural state.

b.2) [10 points] Could the following path potentially lead to incorrect execution? Explain.

order	**Processor 0**	**Processor 1**	**Processor 2**	**Processor 3**
1^{st}				ld 0x51110040
2^{nd}	ld 0x5FFFFF00			
3^{rd}			ld 0x51234540	
4^{th}	st 0x5FFFFF40			
5^{th}				ld 0x51234540
6^{th}	ld 0x5FFFFF00			

Yes.

Explanation. The 1^{st} instruction could read invalid data. This would be the case if the cosmic-ray-induced change was from I to S in Cache 3 for cache line 0x51110040.

After some time executing a particular path (which could be a path *different* from the paths in parts b.1-b.4) and with no further state changes caused by cosmic rays, we find that the final state of the caches is as follows.

Final State

Cache 0		
	Tag	*MESI state*
Set 0	0x5FFFFF	M
Set 1	0x5FFFFF	E
Set 2	0x5FFFFF	S
Set 3	0x5FFFFF	E

Cache 1		
	Tag	*MESI state*
Set 0	0x5FF000	I
Set 1	0x510000	S
Set 2	0x5FFFFF	S
Set 3	0x533333	I

Cache 2		
	Tag	*MESI state*
Set 0	0x5F111F	M
Set 1	0x511100	E
Set 2	0x5FFFFF	S
Set 3	0x533333	I

Cache 3		
	Tag	*MESI state*
Set 0	0x5FF000	M
Set 1	0x511100	S
Set 2	0x5FFFF0	I
Set 3	0x533333	I

(c) [25 points] What is the *minimum* set of memory instructions that leads the system from the initial state to the final state? Indicate the set of instructions in order, and clearly specify the access type (ld/st), the address of each memory request, and the processor from which the request is generated.

The minimum set of instructions is:

(1) st 0x533333C0 // Processor 0

(2) ld 0x5FFFFFC0 // Processor 0

(3) ld 0x5FF00000 // Processor 1

(4) st 0x5FF00000 // Processor 3

Alternatively, as instructions (1)(2) and instructions (3)(4) touch different cache lines, we just need to keep the order between (1)(2), and between (3)(4). These are valid reorderings: (3)(4)(1)(2), (1)(3)(2)(4), (3)(1)(4)(2), (1)(3)(4)(2) or (3)(1)(2)(4).

Explanation.

(1) The instruction sets the line 0x533333C0 to M state in Cache 0, and invalidates the line 0x533333C0 in Cache 1 and Cache 2.

(2) The instruction evicts 0x533333C0 from Cache 0, and sets the line 0x5FFFFFC0 to E state in Cache 0.

(3) The instruction sets the line 0x5FF00000 to S state in Cache 1, as well as in Cache 3.

(4) The instruction sets the line 0x5FF00000 to M state in Cache 3, and it invalidates the line 0x5FF00000 in Cache 1.