ECE M116C - CS M151B Computer Architecture Systems - UCLA

Homework 5 - Solution

Fall 2021

Q1. Assume that we have a multi-core system with the following instruction sequences:

Initial Values: `x1=1, x2=2, x3=0, x4=0, M[A]=0, M[B]=0`

THREAD 1	THREAD 2	THREAD 3
`M[A] = x1;`	`while(M[A] == 0);`	`while (M[B] == 0);`
`M[B] = x2;`	`x3 = M[B];`	`x4 = M[A];`
`x3 = M[A];`	`M[B] = 4;`	`M[A] = 3;`

a) What are the possible values for `x3` and `x4` after running the above multi-thread program under a "weak" memory model (i.e., assume that any ordering is valid)? For each output, please show the sequence of instructions that lead to that output.

Without loss of generality, we assume that T2.1 and T3.1 execute only once.

We note $MA.B < MC.D$ if the memory operation of TA.B occurs before the memory operation of TC.D in the global memory order. We designate the final value of register xn on thread X as $TX(xn)$.

We first assume a relaxed memory model that has a global memory order, where each processor issues memory instruction in any causal order, and that does not ensure that RAW and WAR memory dependencies are correctly resolved within a processor. Because processors cannot speculatively commit memory operations, the following assumptions must hold:

$$M2.1 < M2.2$$
$$M2.1 < M2.3$$
$$M3.1 < M3.2$$
$$M3.1 < M3.3$$
$$(M1.1 < M2.1) \vee (M3.3 < M2.1)$$
$$(M1.2 < M3.1) \vee (M2.3 < M3.1)$$

Obviously, we have $T1(x4) = 0,\ T2(x4) = 0,\ T3(x3) = 0.$

On thread 1, x3 can only possibly be 0, 1 or 3. We have

$$T1(x3) = 0 \Leftrightarrow (M1.3 < M1.1) \wedge (M1.3 < M3.3)$$
$$T1(x3) = 1 \Leftrightarrow (M1.1 < M1.3 < M3.3) \vee (M3.3 < M1.1 < M1.3)$$
$$T1(x3) = 3 \Leftrightarrow (M3.3 < M1.3 < M1.1) \vee (M1.1 < M3.3 < M1.3)$$

On thread 2, x3 can only possibly be 0, 2, or 4. We have

$$T2(x3) = 0 \Leftrightarrow (M2.2 < M1.2) \wedge (M2.2 < M2.3)$$
$$T2(x3) = 2 \Leftrightarrow (M1.2 < M2.2 < M2.3) \vee (M2.3 < M1.2 < M2.2)$$
$$T2(x3) = 4 \Leftrightarrow (M1.2 < M2.3 < M2.2) \vee (M2.3 < M2.2 < M1.2)$$

On thread 3, x4 can only possible be 0, 1, or 3. We have

$$T3(x4) = 0 \Leftrightarrow (M3.2 < M1.1) \wedge (M3.2 < M3.3)$$
$$T3(x4) = 1 \Leftrightarrow (M1.1 < M3.2 < M3.3) \vee (M3.3 < M1.1 < M3.2)$$
$$T3(x4) = 3 \Leftrightarrow (M1.1 < M3.3 < M3.2) \vee (M3.3 < M3.2 < M1.1)$$

We can deduce that

$$(T2(x3) \neq 0) \vee (T3(x4) \neq 0)$$

The remaining potential outcomes are

T 1		T 2		T 3		Global memory order
x3	x4	x3	x4	x3	x4	
0	0	0	0	0	1	M1.3 < M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3
0	0	0	0	0	3	M1.3 < M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.3 < M3.2
0	0	2	0	0	0	M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.1 < M2.1 < M2.2 < M2.3
0	0	2	0	0	1	M1.3 < M1.1 < M2.1 < M1.2 < M3.1 < M3.2 < M3.3 < M2.2 < M2.3
0	0	2	0	0	3	M1.3 < M1.1 < M2.1 < M1.2 < M3.1 < M3.3 < M3.2 < M2.2 < M2.3
0	0	4	0	0	0	M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.1 < M2.1 < M2.3 < M2.2
0	0	4	0	0	1	M1.3 < M1.1 < M2.1 < M1.2 < M3.1 < M3.2 < M3.3 < M2.3 < M2.2
0	0	4	0	0	3	M1.3 < M1.1 < M2.1 < M1.2 < M3.1 < M3.3 < M3.2 < M2.3 < M2.2
1	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M1.3 < M3.3
1	0	0	0	0	3	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M1.3 < M3.3 < M3.2
1	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.2 < M2.3 < M1.3 < M3.3
1	0	2	0	0	1	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M2.2 < M2.3
1	0	2	0	0	3	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.3 < M3.2 < M2.2 < M2.3
1	0	4	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.3 < M2.2 < M1.3 < M3.3
1	0	4	0	0	1	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M2.3 < M2.2
1	0	4	0	0	3	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.3 < M3.2 < M2.3 < M2.2
3	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3
3	0	0	0	0	3	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.3 < M3.2 < M1.3
3	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M3.3 < M1.3 < M1.1 < M2.1 < M2.2 < M2.3
3	0	2	0	0	1	M1.2 < M3.1 < M3.3 < M1.3 < M1.1 < M2.1 < M2.2 < M2.3 < M3.2
3	0	2	0	0	3	M1.2 < M3.1 < M3.3 < M3.2 < M1.3 < M1.1 < M2.1 < M2.2 < M2.3
3	0	4	0	0	0	M1.2 < M3.1 < M3.2 < M3.3 < M1.3 < M1.1 < M2.1 < M2.3 < M2.2
3	0	4	0	0	1	M1.2 < M3.1 < M3.3 < M1.3 < M1.1 < M2.1 < M2.3 < M2.2 < M3.2
3	0	4	0	0	3	M1.2 < M3.1 < M3.3 < M3.2 < M1.3 < M1.1 < M2.1 < M2.3 < M2.2

If we impose that each processor resolves RAW memory dependencies correctly, then we have

$$M1.1 < M1.3$$

The list of possible outcomes becomes:

T 1		T 2		T 3		Global memory order
x3	x4	x3	x4	x3	x4	
0	0	0	0	0	1	Impossible
0	0	0	0	0	3	Impossible
0	0	2	0	0	0	Impossible
0	0	2	0	0	1	Impossible
0	0	2	0	0	3	Impossible
0	0	4	0	0	0	Impossible
0	0	4	0	0	1	Impossible
0	0	4	0	0	3	Impossible
1	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M1.3 < M3.3
1	0	0	0	0	3	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M1.3 < M3.3 < M3.2
1	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.2 < M2.3 < M1.3 < M3.3
1	0	2	0	0	1	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M2.2 < M2.3
1	0	2	0	0	3	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.3 < M3.2 < M2.2 < M2.3
1	0	4	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.3 < M2.2 < M1.3 < M3.3
1	0	4	0	0	1	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M2.3 < M2.2
1	0	4	0	0	3	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.3 < M3.2 < M2.3 < M2.2
3	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3
3	0	0	0	0	3	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.3 < M3.2 < M1.3
3	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.2 < M2.3 < M3.3 < M1.3
3	0	2	0	0	1	M1.1 < M2.1 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3 < M2.2 < M2.3
3	0	2	0	0	3	M1.1 < M2.1 < M1.2 < M3.1 < M3.3 < M3.2 < M1.3 < M2.2 < M2.3
3	0	4	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.3 < M2.2 < M3.3 < M1.3
3	0	4	0	0	1	M1.1 < M2.1 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3 < M2.3 < M2.2
3	0	4	0	0	3	M1.1 < M2.1 < M1.2 < M3.1 < M3.3 < M3.2 < M1.3 < M2.3 < M2.2

If we also impose that each processor resolves WAR memory dependencies correctly, we have

$$M2.2 < M2.3$$
$$M3.2 < M3.3$$

The list of possible outcomes becomes:

T 1		T 2		T 3		Global memory order
x3	x4	x3	x4	x3	x4	
0	0	0	0	0	1	Impossible
0	0	0	0	0	3	Impossible
0	0	2	0	0	0	Impossible
0	0	2	0	0	1	Impossible
0	0	2	0	0	3	Impossible
0	0	4	0	0	0	Impossible
0	0	4	0	0	1	Impossible
0	0	4	0	0	3	Impossible
1	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M1.3 < M3.3
1	0	0	0	0	3	Impossible
1	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.2 < M2.3 < M1.3 < M3.3
1	0	2	0	0	1	M1.1 < M2.1 < M1.3 < M1.2 < M3.1 < M3.2 < M3.3 < M2.2 < M2.3
1	0	2	0	0	3	Impossible
1	0	4	0	0	0	Impossible
1	0	4	0	0	1	Impossible
1	0	4	0	0	3	Impossible
3	0	0	0	0	1	M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3
3	0	0	0	0	3	Impossible
3	0	2	0	0	0	M1.2 < M3.1 < M3.2 < M1.1 < M2.1 < M2.2 < M2.3 < M3.3 < M1.3
3	0	2	0	0	1	M1.1 < M2.1 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3 < M2.2 < M2.3
3	0	2	0	0	3	Impossible
3	0	4	0	0	0	Impossible
3	0	4	0	0	1	Impossible
3	0	4	0	0	3	Impossible

b) If the memory model is TSO, what are the possible values for `x3` and `x4`? (Show the sequence).

Without loss of generality, we assume that T2.1 and T3.1 execute only once.

We note $MA.B < MC.D$ if the memory operation of TA.B occurs before the memory operation of TC.D in the global memory order. We designate the final value of register xn on thread X as $TX(xn)$.

Because of TSO, the following assumptions hold:

$$M1.1 < M1.2$$
$$M2.1 < M2.2 < M2.3$$
$$M3.1 < M3.2 < M3.3$$
$$(M1.1 < M2.1) \vee (M3.3 < M2.1)$$
$$(M2.1 < M3.1) \vee (M2.3 < M3.1)$$

Obviously, we have $T1(x4) = 0,\ T2(x4) = 0,\ T3(x3) = 0.$

On thread 1, x3 can only possibly be 1 or 3. We have

$$T1(x3) = 1 \Leftrightarrow (M3.3 < M1.1) \vee (M3.3 > M1.3)$$
$$T1(x3) = 3 \Leftrightarrow M1.1 < M3.3 < M1.3$$

On thread 2, x3 can only possibly be 0 or 2. We have

$$T2(x3) = 0 \Leftrightarrow (M2.1 < M2.3) \wedge (M2.1 < M1.2) \Leftrightarrow M2.1 < M1.2$$
$$T2(x3) = 2 \Leftrightarrow (M2.2 > M1.2) \wedge (M2.2 < M2.3) \Leftrightarrow M2.2 > M1.2$$

On thread 3, x4 can only possibly be 0 or 1. We have

$$T3(x4) = 0 \Leftrightarrow (M3.2 < M1.1) \wedge (M3.2 < M3.3) \Leftrightarrow M3.2 < M1.1 \Leftrightarrow false$$
$$T3(x4) = 1 \Leftrightarrow (M3.2 > M1.1) \wedge (M3.2 < M3.3) \Leftrightarrow M3.2 > M1.1 \Leftrightarrow true$$

The remaining possible outcomes are:

T 1		T 2		T 3		Global memory order
x3	x4	x3	x4	x3	x4	
1	0	0	0	0	1	$M1.1 < M2.1 < M1.3 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3$
1	0	2	0	0	1	$M1.1 < M2.1 < M1.2 < M2.2 < M2.3 < M3.1 < M3.2 < M1.3 < M3.3$
3	0	0	0	0	1	$M1.1 < M2.1 < M2.2 < M2.3 < M1.2 < M3.1 < M3.2 < M3.3 < M1.3$
3	0	2	0	0	1	$M1.1 < M2.1 < M1.2 < M2.2 < M2.3 < M3.1 < M3.2 < M3.3 < M1.3$

c) Under a "weak" memory model, add `fences` and `acquire(lock)`, `release(lock)` to create the following sequence: (`TX.Y` means the Yth instruction in thread X, e.g., `T2.2` is `x3 = M[B];`).

Sequence: `T1.1, T1.2, T2.1, T2.2, T3.1, T3.2, T3.3, T2.3, T1.3`

THREAD 1
`M[A] = x1;`
`fence;`
`M[B] = x2;`
`barrier(b1);`
`barrier(b4);`
`x3 = M[A];`

THREAD 2
`barrier(b1);`
`while(M[A] == 0);`
`x3 = M[B];`
`barrier(b2);`
`barrier(b3);`
`M[B] = 4;`
`barrier(b4);`

THREAD 3
`barrier(b2);`
`while (M[B] == 0);`
`x4 = M[A];`
`fence;`
`M[A] = 3;`
`barrier(b3);`

A **fence** is here an instruction that waits until all previous memory instructions are committed to memory.

A **barrier** (was referred to as "external fence" during lecture 16) is here a synchronization mechanism that blocks the execution of threads until the correct number of threads have reached it. Implementation of a barrier requires external fence instructions.

Note: This is not the only possible solution. In particular, it is possible to replace some of the above barriers with specific combinations of **locks**. However, the goal here is to generate an exact execution order, and not simply to prevent concurrent execution of two segments of code. As such, using barriers is a more natural solution. Locks alone cannot ensure that a given thread execute a protected segment first.

Q2. Under a "weak" memory model, find all the possible outputs for the following program: (assume initially: x=y=z=0)

THREAD 1	THREAD 2	THREAD 3
x=1;	barrier(b1);	barrier(b1);
print (y,z);	acquire (lock);	acquire (lock);
barrier(b1);	y=1;	z=1;
	print (x,z);	print (x,y);
	release (lock);	release (lock);

The possible orders of execution of the instructions are:

- T1: x = 1
  T1: print(y,z) -> prints (0,0)
  T*: fence -> here, x is 1 globally
  T2: acquire(lock)
  T2: y = 1
  T2: print(x,z) -> prints (1,0)
  T2: release(lock)
  T3: acquire(lock)
  T3: z = 1
  T3: print(x,y) -> prints (1,0) or (1,1) because there were no fence since (y = 1)
  T3: release(lock)

- T1: x = 1
  T1: print(y,z) -> prints (0,0)
  T*: fence -> here, x is 1 globally
  T3: acquire(lock)
  T3: z = 1
  T3: print(x,y) -> prints (1,0)
  T3: release(lock)
  T2: acquire(lock)
  T2: y = 1
  T2: print(x,z) -> prints (1,0) or (1,1) because there were no fence since (z = 1)
  T2: release(lock)

The two possible outputs are

- (0,0)
  (1,0)
  (1,0)

- (0,0)
  (1,0)
  (1,1)

In its most primitive form, the release method does not guarantee that memory is synchronized before the lock is effectively released. However, because that is a useful property in practice, implementations of the release method often execute a fence instruction before releasing the lock. If that is the case, then only the second outcome is allowed.

Q3. Suppose we have private fully-associative L1 caches with MSI coherence protocol for three cores as shown below. For each core, the initial state of each line is shown in the table.

P1

Line	Coherency State	Tag
0	I	100
1	S	120
2	M	140

P2

Line	Coherency State	Tag
0	M	100
1	S	120
2	I	140

P3

Line	Coherency State	Tag
0	I	100
1	S	120
2	I	140

If we have the following sequence of accesses, show the coherency state for each line (for each access, you can only show the lines whose states have been changed).

	P1	P2	P3
P1: read 120			
P1: write 120	Line 1 becomes M	Line 1 becomes I	Line 1 becomes I
P3: write 120	Line 1 becomes I	Line 1 becomes I	Line 1 becomes M
P2: read 100			
P3: write 100		Line 0 becomes I	Line 0 becomes M
P1: read 100	Line 0 becomes S		Line 0 becomes S
P3: read 100			
P2: read 140	Line 2 becomes S	Line 2 becomes S	
P2: write 140	Line 2 becomes I	Line 2 becomes M	

Q4. Assume we have MOESIF coherency protocol. Assume we have 4 cores, and all cores are sharing the memory line `M[Z]`. Complete the following table (show the writebacks with *, and if one cache forwards a value to another cache show it by ^):

Access	P1	P2	P3	P4
R1	E	I	I	I
R1	E	I	I	I
R2	F^	S	I	I
R3	F^	S	S	I
W3	I	I	M	I
R4	I	I	O^	S
W4	I	I	I*	M
W1	M	I	I	I*
R1	M	I	I	I
R2	O^	S	I	I