

Problem M13.1: Sequential Consistency

For this problem we will be using the following sequences of instructions. These are small programs, each executed on a different processor, each with its own cache and register set. In the following **R** is a register and **X** is a memory location. Each instruction has been named (e.g., B3) to make it easy to write answers.

Assume data in location X is initially 0.

Processor A	Processor B	Processor C
A1: ST X, 1	B1: R := LD X	C1: ST X, 6
A2: R := LD X	B2: R := ADD R, 1	C2: R := LD X
A3: R := ADD R, R	B3: ST X, R	C3: R := ADD R, R
A4: ST X, R	B4: R := LD X	C4: ST X, R
	B5: R := ADD R, R	
	B6: ST X, R	

For each of the questions below, please circle the answer and provide a short explanation assuming the program is executing under the SC model. **No points will be given for just circling an answer!**

Problem M13.1.A

Can X hold value of 4 after all three threads have completed? Please explain briefly.

Yes / No

Problem M13.1.B

Can X hold value of 5 after all three threads have completed?

Yes / No

Problem M13.1.C

Can X hold value of 6 after all three threads have completed?

Yes / No

Problem M13.1.D

For this particular program, can a processor that reorders instructions but follows local dependencies produce an answer that cannot be produced under the SC model?

Yes / No

Problem M13.2: Relaxed Memory Models

Consider a system which uses Weak Ordering, meaning that a read or a write may complete before a read or a write that is earlier in program order if they are to different addresses and there are no data dependencies.

Our processor has four fine-grained memory barrier instructions:

- **MEMBAR_{RR}** guarantees that all read operations initiated before the MEMBAR_{RR} will be seen before any read operation initiated after it.
- **MEMBAR_{RW}** guarantees that all read operations initiated before the MEMBAR_{RW} will be seen before any write operation initiated after it.
- **MEMBAR_{WR}** guarantees that all write operations initiated before the MEMBAR_{WR} will be seen before any read operation initiated after it.
- **MEMBAR_{WW}** guarantees that all write operations initiated before the MEMBAR_{WW} will be seen before any write operation initiated after it.

We will study the interaction between two processes on different processors on such a system:

P1	P2
P1.1: LW R2, 0(R8)	P2.1: LW R4, 0(R9)
P1.2: SW R2, 0(R9)	P2.2: SW R5, 0(R8)
P1.3: LW R3, 0(R8)	P2.3: SW R4, 0(R8)

We begin with following values in registers and memory (same for both processes):

register/memory	Contents
R2	0
R3	0
R4	0
R5	8
R8	0x01234567
R9	0x89abcdef
M[R8]	6
M[R9]	7

After both processes have executed, is it possible to have the following machine state? Please circle the correct answer. If you circle **Yes**, please provide sequence of instructions that lead to the desired result (one sequence is sufficient if several exist). If you circle **No**, please explain which ordering constraint prevents the result.

Problem M13.2.A

Memory	contents
M[R8]	7
M[R9]	6

Yes

No

Problem M13.2.B

memory	Contents
M[R8]	6
M[R9]	7

Yes

No

Problem M13.2.C

Is it possible for M[R8] to hold 0?

Yes

No

Now consider the same program, but with two **MEMBAR** instructions.

P1	P2
P1.1: LW R2, 0(R8)	P2.1: LW R4, 0(R9)
P1.2: SW R2, 0(R9)	MEMBAR_{RW}
MEMBAR_{WR}	P2.2: SW R5, 0(R8)
P1.3: LW R3, 0(R8)	P2.3: SW R4, 0(R8)

We want to compare execution of the two programs on our system.

Problem M13.2.D

If both M[R8] and M[R9] contain 6, is it possible for R3 to hold 8?

Without **MEMBAR** instructions? **Yes** **No**

With **MEMBAR** instructions? **Yes** **No**

Problem M13.2.E

If both M[R8] and M[R9] contain 7, is it possible for R3 to hold 6?

Without **MEMBAR** instructions? **Yes** **No**

With **MEMBAR** instructions? **Yes** **No**

Problem M13.2.F

Is it possible for both M[R8] and M[R9] to hold 8?

Without **MEMBAR** instructions? **Yes** **No**

With **MEMBAR** instructions? **Yes** **No**

Problem M13.3: Memory Models

Consider a system which uses **Sequential Consistency (SC)**. There are three processes, **P1**, **P2** and **P3**, on different processors on such a system (the values of R_A , R_B , R_C were all zeros before the execution):

P1	P2	P3
P1.1: ST (A), 1	P2.1: ST (B), 1	P3.1: ST (C), 1
P1.2: LD R_C , (C)	P2.2: LD R_A , (A)	P3.2: LD R_B , (B)

Problem M13.3.A

After all processes have executed, it is possible for the system to have multiple machine states. For example, $\{R_A, R_B, R_C\} = \{1, 1, 1\}$ is possible if the execution sequence of instructions is $P1.1 \rightarrow P2.1 \rightarrow P3.1 \rightarrow P1.2 \rightarrow P2.2 \rightarrow P3.2$. Also, $\{R_A, R_B, R_C\} = \{1, 1, 0\}$ is possible if the sequence is $P1.1 \rightarrow P1.2 \rightarrow P2.1 \rightarrow P3.1 \rightarrow P2.2 \rightarrow P3.2$.

For each state of $\{R_A, R_B, R_C\}$ below, specify the execution sequence of instructions that results in the corresponding state. If the state is **NOT** possible with SC, just put X.

$\{0,0,0\}$:

$\{0,1,0\}$:

$\{1,0,0\}$:

$\{0,0,1\}$:

Problem M13.3.B

Now consider a system which uses **Weak Ordering(WO)**, meaning that a read or a write may complete before a read or a write that is earlier in program order if they are to different addresses and there are no data dependencies.

Does WO allow the machine state(s) that is not possible with SC? If yes, provide an execution sequence that will generate the machine states(s).

Problem M13.3.C

The WO system in Problem M13.3.B provides four fine-grained memory barrier instructions. Below is the description of these instructions.

- **MEMBAR_{RR}** guarantees that all read operations initiated before the MEMBAR_{RR} will be seen before any read operation initiated after it.
- **MEMBAR_{RW}** guarantees that all read operations initiated before the MEMBAR_{RW} will be seen before any write operation initiated after it.
- **MEMBAR_{WR}** guarantees that all write operations initiated before the MEMBAR_{WR} will be seen before any read operation initiated after it.
- **MEMBAR_{WW}** guarantees that all write operations initiated before the MEMBAR_{WW} will be seen before any write operation initiated after it.

Using the minimum number of memory barrier instructions, rewrite **P1**, **P2** and **P3** so the machine state(s) that is not possible with SC by the original programs is also not possible with WO by your programs.

P1	P2	P3
P1.1: ST (A), 1 P1.2: LD R _C , (C)	P2.1: ST (B), 1 P2.2: LD R _A , (A)	P3.1: ST (C), 1 P3.2: LD R _B , (B)

Problem M13.4: Memory Consistency (Spring 2020 Quiz 3, Part B)

Consider a shared-memory machine that executes the following two threads on two different cores. Assume that memory locations *a* and *b* contain initial value 0.

T1	T2
T1.1: Store (<i>a</i>) $\leftarrow$ 1 T1.2: Load <i>r1</i> $\leftarrow$ (<i>a</i>) T1.3: Load <i>r3</i> $\leftarrow$ (<i>b</i>)	T2.1: Store (<i>b</i>) $\leftarrow$ 1 T2.2: Load <i>r2</i> $\leftarrow$ (<i>b</i>) T2.3: Load <i>r4</i> $\leftarrow$ (<i>a</i>)

Problem M13.4.A

If the machine implements **sequential consistency**, what execution outcomes (i.e., values of *r1*, *r2*, *r3*, and *r4*) can this code produce?

Note: You can but *do not have to* express the result as (*r1*, *r2*, *r3*, *r4*) tuples

Problem M13.4.B

If the machine implements the **Total Store Order (TSO)** consistency model, what execution outcomes (i.e., values of r1, r2, r3, and r4) can this code produce?

Note: You can but *do not have to* express the result as (r1, r2, r3, r4) tuples

Problem M13.4.C

If the machine implements a **relaxed consistency model, RMO**, which allows loads and stores to be reordered after later loads and stores, what execution outcomes (i.e., values of r1, r2, r3, and r4) can this code produce?

Note: You can but *do not have to* express the result as (r1, r2, r3, r4) tuples

Problem M13.4.D

The relaxed consistency model (RMO) has the following fine-grained barrier instructions:

- **MEMBAR_{RR}** guarantees that all reads that precede MEMBAR_{RR} in program order will be performed before any read that follows the barrier.
- **MEMBAR_{RW}** guarantees that all reads that precede MEMBAR_{RW} in program order will be performed before any write that follows the barrier.
- **MEMBAR_{WR}** guarantees that all writes that precede MEMBAR_{WR} in program order will be performed before any read that follows the barrier.
- **MEMBAR_{WW}** guarantees that all writes that precede MEMBAR_{WW} in program order will be performed before any write that follows the barrier.

Add barrier instructions to T1 and T2 so that the RMO machine produces the same outputs as the SC machine for this code. Use the *minimum* number of memory barrier instructions. List the locations of each barrier below (e.g., “Add **MEMBAR_{RR}** after T1.1”).

T1	T2
T1.1: Store (a) $\leftarrow$ 1	T2.1: Store (b) $\leftarrow$ 1
T1.2: Load r1 $\leftarrow$ (a)	T2.2: Load r2 $\leftarrow$ (b)
T1.3: Load r3 $\leftarrow$ (b)	T2.3: Load r4 $\leftarrow$ (a)

Problem M13.4.E

Consider a shared-memory machine that executes the following four threads on four cores. Assume that memory location *a* contains initial value 0.

T1	T2	T3	T4
Store (<i>a</i>) $\leftarrow$ 1	Store (<i>a</i>) $\leftarrow$ 2	Load <i>r1</i> $\leftarrow$ (<i>a</i>) Load <i>r2</i> $\leftarrow$ (<i>a</i>)	Load <i>r3</i> $\leftarrow$ (<i>a</i>) Load <i>r4</i> $\leftarrow$ (<i>a</i>)

If the machine implements the **TSO** consistency model, can it produce the following execution outcome (*r1*, *r2*, *r3*, *r4*) = (1, 2, 2, 1)

Problem M13.4.F

Ben Bitdiddle modifies the above TSO machine. The original machine has one thread per core. Ben implements multi-threading, making each core support 2 thread contexts. The threads running on the same core share a single committed store buffer.

This machine executes the four threads in Question 5. T1 and T3 run on Core 0, and T2 and T4 run on Core 1. Can this machine produce the following execution outcome (*r1*, *r2*, *r3*, *r4*) = (1, 2, 2, 1)?

Problem M13.4.G

Does the machine described in Question 6 still maintain TSO