Problem M13.1: Sequential Consistency

Problem M13.1.A

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

(Yes) / No

C1-C4, B1-B3, A1-A4, B4- B6

Problem M13.1.B

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

Yes / (No)

All results must be even!

Problem M13.1.C

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

(Yes) / No

All of C, All of A, All of B

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)

All stores/loads must be done in order because they're to the same address, so no new results are possible.

Problem M13.2: Relaxed Memory Models [? Hours]

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)`

Problem M13.2.A

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

Yes **No**

P1.1 P2.1 P1.2 P1.3 P2.2 P2.3

Problem M13.2.B

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

Yes **No**

The result would require that the memory contents don't change. Since each thread reads a data value and writes it to another address, this simply impossible here.

Problem M13.2.C

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

Yes **No**

The only way that M[R8] could end up with 0 is if P2.3 is completed before P2.1 and P2.2. This violates Weak Ordering, so it is not possible.

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.

Here the intention was to keep the starting conditions the same as in first three questions, and ask about the final conditions. This wasn't clear, so we accepted both solutions. The yes/no answers don't actually change, but Questions 11 for 12 become simpler.

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**

Following sequence works with and without MEMBAR instructions:
P1.1 -> P1.2 -> P2.1 -> P2.2 -> P1.3 -> P2.3

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**)

If M[R8] and M[R9] are to end up with 7, we have to execute P2.3 before we execute P1.1 Since P1.3 has to come after P1.1 (Weak Ordering), R3, has to end up with 7 not 6.

Problem M13.2.F

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

Without **MEMBAR** instructions? (**Yes**) **No**

P2.2 P1.1 P1.2 P2.1 P2.3 P1.3

With **MEMBAR** instructions? **Yes** (**No**)

The sequence above violates the MEMBAR in P2—P2.2 executes before P2.1. That is the only way to get 8 into both memory locations, thus the result is impossible with MEMBARs insterted.

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 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 M13.3.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→P2.1→P3.1→P1.2→P2.2→P3.2. Also, $\{R_A, R_B, R_C\} = \{1,1,0\}$ is possible if the sequence is P1.1 → P1.2 → P2.1 → P3.1 → P2.2 → 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} : X

{0,1,0} : P2.1 P2.2 P1.1P1.2P3.1 P3.2

{1,0,0} : P1.1 P1.2 P3.1 P3.2 P2.1 P2.2

{0,0,1} : P3.1 P3.2 P2.1 P2.2 P1.1 P1.2

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).

Yes. {0,0,0} by P1.2→P2.2→P3.2→P1.1→P2.1→P3.1

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 MEMBAR$_{WR}$ P1.2: LD R$_C$, (C)	P2.1: ST (B), 1 MEMBAR$_{WR}$ P2.2: LD R$_A$, (A)	P3.1: ST (C), 1 MEMBAR$_{WR}$ 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 (a) ← 1 T1.2: Load r1 ← (a) T1.3: Load r3 ← (b)	T2.1: Store (b) ← 1 T2.2: Load r2 ← (b) T2.3: Load r4 ← (a)

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

r1 = 1
r2 = 1
r3, r4 can be any of (0,1) (1,0) or (1,1)

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

r1 = 1
r2 = 1
r3, r4 can be anything

Note that TSO allows the following execution:
T1.2 < T1.3 < T1.1 while requiring T1.2 to return the value stored by T.1.1 due to store-load forwarding. ("<" means happens-before.)

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

Same as TSO.

Store-load forwarding typically exists, so r1 and r2 are still 1s.
(0, 0) for (r3, r4) is valid because this simply requires store-load reordering, which is already allowed by TSO.

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

Place a **MEMBAR$_{WR}$** between T1.1 and either T1.2 or T1.3 for T1. Same for T2.

Note that **MEMBAR$_{RR}$** is not helpful since it does not prevent reordering T1.3 and T1.1

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 (a) ← 1	Store (a) ← 2	Load r1 ← (a) Load r2 ← (a)	Load r3 ← (a) Load r4 ← (a)

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

No. Stores from T1 and T2 should appear in the same order for T3 and T4

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)?

Yes. The shared committed store buffer allows T3 to observe T1's store first, and allows T4 to observe T2's store first

Problem M13.4.G

Does the machine described in Question 6 still maintain TSO

No. Compare the answers in Q5 and Q6.