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)	$\text{MEMBAR}_{\text{RW}}$
$\text{MEMBAR}_{\text{WR}}$	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 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} :

{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.

- **$\text{MEMBAR}_{\text{RR}}$** guarantees that all read operations initiated before the $\texttt{MEMBAR}_{\texttt{RR}}$ will be seen before any read operation initiated after it.
- **$\text{MEMBAR}_{\text{RW}}$** guarantees that all read operations initiated before the $\texttt{MEMBAR}_{\texttt{RW}}$ will be seen before any write operation initiated after it.
- **$\text{MEMBAR}_{\text{WR}}$** guarantees that all write operations initiated before the $\texttt{MEMBAR}_{\texttt{WR}}$ will be seen before any read operation initiated after it.
- **$\text{MEMBAR}_{\text{WW}}$** guarantees that all write operations initiated before the $\texttt{MEMBAR}_{\texttt{WW}}$ 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	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 M13.4: Memory consistency models (Spring 2016 Quiz 3, Part B)

Consider two processes P1 and P2 running on two different processors.
Assume that memory locations X and Y contain initial value 0.

P1	P2
P1.1: LD R1 ← (Y) P1.2: LD R2 ← (X)	P2.1: ST (X) ← 1 P2.2: ST (Y) ← 1

Problem M13.4.A

Out of the following possible final values of (X, Y, R1, R2), circle the ones that could occur if the system is **Sequentially Consistent (SC)**.

(0,0,0,0)	**(0,0,0,1)**	**(0,0,1,0)**	**(0,0,1,1)**
(1,1,0,0)	**(1,1,0,1)**	**(1,1,1,0)**	**(1,1,1,1)**

Problem M13.4.B

Out of the following possible final values of (X, Y, R1, R2), circle the ones that could occur if the system enforces **RMO**, a weak memory model where **loads and stores can be reordered after prior loads or stores.**

(0,0,0,0)	**(0,0,0,1)**	**(0,0,1,0)**	**(0,0,1,1)**
(1,1,0,0)	**(1,1,0,1)**	**(1,1,1,0)**	**(1,1,1,1)**

Problem M13.4.C

The RMO machine has the following fine-grained barrier instructions:

- **MEMBAR$_{RR}$** guarantees that all reads initiated before MEMBAR$_{RR}$ will be performed before any read initiated after it.
- **MEMBAR$_{RW}$** guarantees that all reads initiated before MEMBAR$_{RW}$ will be performed before any write initiated after it.
- **MEMBAR$_{WR}$** guarantees that all writes initiated before MEMBAR$_{WR}$ will be performed before any read initiated after it.
- **MEMBAR$_{WW}$** guarantees that all writes initiated before MEMBAR$_{WW}$ will be performed before any write initiated after it.

Use the **minimum number of memory barrier instructions**, rewrite **P1** and **P2** such that the **RMO machine produces the same outputs as the SC machine for the given code.**

P1	**P2**
P1.1: LD R1 ← (Y)	P2.1: ST (X) ← 1
P1.2: LD R2 ← (X)	P2.2: ST (Y) ← 1

Again, consider two processes P1 and P2 running the code below on two different processors. **Assume that memory locations X, Y, and Z contain initial value 0.**

P1	P2
P1.1: LD R1 ← (Z) P1.2: ST (Y) ← 1 P1.3: LD R2 ← (X)	P2.1: ST (X) ← 1 P2.2: LD R3 ← (Y) P2.3: ST (Z) ← 1

Problem M13.4.D

Out of the following possible final values of (R1, R2, R3), circle the ones that could occur if the system is **Sequentially Consistent (SC)**.

(0,0,0)	**(0,1,0)**	**(1,0,0)**	**(1,1,0)**
(0,0,1)	**(0,1,1)**	**(1,0,1)**	**(1,1,1)**

Problem M13.4.E

Out of the following possible final values of (R1, R2, R3), circle the ones that could occur if the system enforces **RMO** (**loads and stores can be reordered after prior loads or stores**).

(0,0,0)	**(0,1,0)**	**(1,0,0)**	**(1,1,0)**
(0,0,1)	**(0,1,1)**	**(1,0,1)**	**(1,1,1)**

Problem M13.4.F

Using the **minimum number of memory barrier instructions** (given in Question 3), rewrite **P1** and **P2** such that the **RMO machine produces the same outputs as the SC machine for the given code.**

P1	**P2**
P1.1: LD R1 ← (Z)	P2.1: ST (X) ← 1
P1.2: ST (Y) ← 1	P2.2: LD R3 ← (Y)
P1.3: LD R2 ← (X)	P2.3: ST (Z) ← 1