Cy D. Fect wants to run the following code sequences on processors P1 and P2, which are part of a two-processor MIPS64 machine. The sequences operate on memory values located at addresses A, B, C, D, E, F, and G, which are all sequentially located in memory (e.g., B is 8 bytes after A). Initially, M[A], M[B], M[C], M[D], and M[E] are all 0. M[F] is 1, and M[G] is 2. For each processor, R1 contains the address A (all of the addresses are located in a shared region of memory). Also, remember that for a MIPS processor, R0 is hardwired to 0. In the below sequences, a semicolon is used to indicate the start of a comment.

P1
ADDI R2, R0, #1 ; R2=1
SD R2, 0(R1) ; A=R2
LD R3, 40(R1) ; R3=F
SD R3, 16(R1) ; C=R3
LD R4, 8(R1) ; R4=B
SD R4, 24(R1) ; D=R4

P2
ADDI R2, R0, #1 ; R2=1
SD R2, 8(R1) ; B=R2
LD R3, 48(R1) ; R3=G
SD R3, 16(R1) ; C=R3
LD R4, 0(R1) ; R4=A
SD R4, 32(R1) ; E=R4

Problem 1.A Sequential Consistency

If Cy’s code runs on a system with a sequentially consistent memory model, what are the possible results of execution? List all possible results in terms of values of M[C], M[D], and M[E] (since the values in the other locations will be the same across all possible execution paths).

Problem 1.B Generalized Synchronization

Assume now that Cy’s code is run on a system that does not guarantee sequential consistency, but that memory dependencies are not violated for the accesses made by any individual processor. The system has a MEMBAR memory barrier instruction that guarantees the effects of all memory instructions executed before the MEMBAR will be made globally visible before any memory instruction after the MEMBAR is executed.

Add MEMBAR instructions to Cy’s code sequences to give the same results as if the system were sequentially consistent. Use the minimum number of MEMBAR instructions.
Problem 1.C  
Total Store Ordering

Now consider a machine that uses finer-grain memory barrier instructions. The following instructions are available:

- \text{MEMBAR}_{rr} \text{ guarantees that all read operations initiated before the } \text{MEMBAR}_{rr} \text{ will be seen before any read operation initiated after it.}
- \text{MEMBAR}_{rw} \text{ guarantees that all read operations initiated before the } \text{MEMBAR}_{rw} \text{ will be seen before any write operation initiated after it.}
- \text{MEMBAR}_{wr} \text{ guarantees that all write operations initiated before the } \text{MEMBAR}_{wr} \text{ will be seen before any read operation initiated after it.}
- \text{MEMBAR}_{ww} \text{ guarantees that all write operations initiated before the } \text{MEMBAR}_{ww} \text{ will be seen before any write operation initiated after it.}

There is no generalized \text{MEMBAR} instruction as in Part B of this problem.

In total store ordering (TSO), a read may complete before a write that is earlier in program order if the read and write are to different addresses and there are no data dependencies. For a machine using TSO, insert the minimum number of memory barrier instructions into the code sequences for P1 and P2 so that sequential consistency is preserved.

Problem 1.D  
Partial Store Ordering

In partial store ordering (PSO), a read or a write may complete before a write that is earlier in program order if they are to different addresses and there are no data dependencies. For a machine using PSO, insert the minimum number of memory barrier instructions from Part C into the code sequences for P1 and P2 so that sequential consistency is preserved.

Problem 1.E  
Weak Ordering

In weak ordering (WO), 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. For a machine using WO, insert the minimum number of memory barrier instructions from Part C into the code sequences for P1 and P2 so that sequential consistency is preserved.

Problem 1.F  
Release Consistency

Release consistency (RC) distinguishes between \textit{acquire} and \textit{release} synchronization operations. An \textit{acquire} must complete before any reads or writes following it in program order, while a read or a write before a \textit{release} must complete before the \textit{release}. However, reads and writes before an \textit{acquire} may complete after the \textit{acquire}, and reads and writes after a \textit{release} may complete before the release. Consider the following modified versions of the original code sequences.
In the above sequences, the *acquire* and *release* operations modify memory location \( H \), which is located sequentially after \( G \). The *acquire* operation performs a read and a write, while the *release* operation performs a write. For a machine using RC, insert the minimum number of memory barrier instructions from Part C into the above code sequences for P1 and P2 so that sequential consistency is preserved.