Problem 1

This problem considers the simple MSI, bus-based snooping protocol for cache coherence discussed in class. There are several processor/cache nodes connected to the bus, along with main memory. Each processor has a private L1 cache that is direct mapped and contains 4 blocks of two words each. The initial state of main memory and each cache block is shown in the figure below. To simplify the figure, the cache address tag contains the full address and each word shows only two hex characters (the rest of the data word is all zeroes, not shown). The lower addressed word of the block is on the right; i.e., for block B0 of cache P0, the data for address x100 is on the right, and the data for x104 is on the left. The coherence states are denoted M, S, and I for Modified, Shared, and Invalid, respectively. Addresses and data are given in hexadecimal.
List the cache block states that change for each of the actions below. Treat each as an independent action applied to the initial state shown, do not accumulate changes from one part to the next. Simply list the blocks that change state in any cache and memory for each action. List your answer in the form “P0.B0: (I, 100, 00, 10)” to mean processor cache P0's Block B0 is Invalid, the tag holds 100, the data words are 00 and 10; for memory use “M: 100 (00, 00)”.

The action “write addr ← data” means write data word data to address addr. When a block changes to Invalid, assume only the state bit changes, so your answer should include the tag and data field’s actual (current) values.

i) P15: read 118
   P15.B3: (S, 118, 00, 18)

ii) P15: write 100 ← 48
    P15.B0: (M, 100, 00, 48)

iii) P15: write 118 ← 80
    P15.B3: (M, 118, 00, 80)
    P1.B3: (I, 118, 00, 18)

iv) P15: write 108 ← 80
    P15.B1: (M, 108, 00, 80)
    P0.B1: (I, 108, 00, 08)

v) P15: read 110
    P0.B2: (S, 110, 00, 30)
    P15.B2: (S, 110, 00, 30)
    M[110]: (00, 30)

vi) P15: read 128
    P1.B1: (S, 128, 00, 68)
    P15.B1: (S, 128, 00, 68)
    M[128]: (00, 68)

vii) P15: write 110 ← 40
     P0.B2: (I, 110, 00, 30)
     P15.B2: (M, 110, 00, 40)
Problem 2

This problem involves implementing a stack using an array in a multiprocessor system. The elements of the array can be accessed in parallel by multiple processors.

You have to write the following two functions:

- **Push**: This will add an element to the top of the stack.
- **Pop**: This will delete an element from the top of the stack.

Assume that Push is never called on a full stack and Pop is never called on an empty stack (i.e., you do not have to worry about overflow and underflow conditions).

Write the Push and Pop functions using an atomic test&set instruction to achieve synchronization. Add C-like pseudo code to the following stub (complete the incomplete statements as well):

```c
int top; // index for the top of the stack
int index; // current index for adding or deleting an element
Lock lock_var; // Lock variable for synchronization

Push (item) {

    index =

    stack[index] = item;
}
```
Pop (void) {

    index =

    item = stack[index];

    return item;
}

Push (item) {

    while (test&set(lock_var) == 1); /* Spin until lock is obtained */

    index = top;
    top++;
    stack[index] = item;
    lock_var = 0; /*Unlock*/
}

Pop (void) {
    while (test&set(lock_var) == 1); /* Spin until lock is obtained */
    index = top-1;
    top--; 
    item = stack[index];
    lock_var = 0; /*Unlock*/
    return item;
}
Problem 3

Systems with a relaxed consistency model usually offer a memory fence instruction to allow the programmer to enforce correct behavior. A full fence is a barrier to reordering memory operations, requiring all previous (by program order) reads and writes to complete before subsequent (by program order) accesses begin. Consider a machine with a very relaxed memory model, where any program ordered pair of accesses to different variables may be reordered. Program order between accesses to the same variable, however, must be enforced. Consider the code segments below. What are the different combinations of values that the four reads on A, B, and C can return (i.e., write the possible combinations of final values of reg1, reg2, reg3, and reg4)? After identifying these possible results, indicate which of these results are possible under Sequential Consistency. Insert the minimal number of fence instructions in the code fragment below to limit the results on our very relaxed machine to only those possible under Sequential Consistency.

All variables are initially 0.

**Processor 0:**

A = 1  
B = 1  
Flag = 1  
while (Flag == 1) spin;  
reg1 = B  
reg2 = C

**Processor 1:**

while (Flag == 0) spin;  
reg3 = A  
reg4 = B  
C = 1  
Flag = 0

reg1’s final value is always 1 because Processor 0’s write of B must occur before its read of B and there is no other write to B. reg2, reg3, and reg4 could have either 0 or 1 as their final values since there is no ordering constraint between the writes of A, B, C and the reads for those register values.

With sequential consistency, the only allowed result is: \( \text{reg1} = \text{reg2} = \text{reg3} = \text{reg4} = 1 \). To achieve this result on our relaxed machine, we need a fence before each write to flag and after each read of flag since these are the synchronizing or racing instructions.