CS6560: Parallel Computer Architecture
Assignment #5
(Submission deadline: 12th April 2016)

1. Consider the following conditions proposed as sufficient conditions for SC:
    a. Every process issues memory requests in the order specified by the program
    b. After a read or write operation is issued, the issuing process waits for the operation to complete before issuing its next operation
    c. Before a processor P_j can return a value written by another processor P_i, all operations that were performed with respect to Pi before it issued the store must also be performed with respect to P_j.

    Are these conditions indeed sufficient to guarantee SC executions? If so, say why. If not, construct a counterexample.

2. The *fetch-and-add* atomic operation can be used to implement barriers, semaphores, and other synchronization mechanisms. The semantics of *fetch-and-add* is such that it adds its second argument to the memory location in its first argument and returns the value of the memory location as it was before the addition. Use the *fetch-and-add* primitive to implement a barrier operation suitable for a shared memory multiprocessor. To use the barrier, a processor must execute BARRIER(BAR, N), where BAR is the barrier name and N is the number of processes that need to arrive at the barrier before any of them can proceed. Assume that N has the same value in each use the barrier BAR. The barrier should be capable of supporting the following code:

```
while (condition) {
        Compute for a while
        BARRIER (BAR, N);
}
```

   A proposed solution for implementing the barrier is the following:

```
BARRIER (Var B: BarVariable, N: integer)
{
        if (fetch-and-add (B, 1) = N-1) then
                B: = 0;
        else
                while (B! = 0) do { };
    }
```

   What is the problem with this code? Write the code for BARRIER in a way that avoids the problem.

3. Consider the following program fragment running on a cache-coherent multiprocessor, assuming all values to be 0 initially.

P1	P2	P3	P4
A = 1	u = A	w = A	A = 2
	v = A	x = A	

   There is only one shared variable (A). Suppose that a writer magically knows where the cached copies are and sends updates to them directly without consulting the home node.
    a. Construct a situation in which write atomicity may be violated, assuming an update-based protocol.
    b. Show the violation of SC that occurs in the results.
    c. Can you produce a case where coherence is violated as well? How would you solve these problems?
    d. Can you construct the same problems for an invalidation-based protocol?
    e. Can you construct them for update protocols on a bus?