CS 530, Fall 2014 **Computer Architecture** Homework #5
Due Date: 2 Dec 2014

---

1. H&P Problem 5.1, part a

a. P0 read 120 -> P0.B0: (S, 120, 0020) returns 0020

2. H&P Problem 5.9, part a

a. P0,0 read 100 -> L1 hit returns 0x0010, state unchanged (M)

3. H&P Problem 5.10, part a

a. P0,0 write 100 <- 80, write hit only seen by P0,0

4. H&P Problem 5.21

To keep the figures from becoming cluttered, the coherence protocol is split into two parts as was done in Figure 5.6 in the text. Figure S.34 presents the CPU portion of the coherence protocol, and Figure S.35 presents the bus portion of the protocol. In both of these figures, the arcs indicate transitions and the text along each arc indicates the stimulus (in normal text) and bus action (in bold text) that occurs during the transition between states. Finally, like the text, we assume a write hit is handled as a write miss.

Figure S.34 presents the behavior of state transitions caused by the CPU itself. In this case, a write to a block in either the invalid or shared state causes us to broad- cast a "write invalidate" to flush the block from any other caches that hold the block and move to the exclusive state. We can leave the exclusive state through either an invalidate from another processor (which occurs on the bus side of the coherence protocol state diagram), or a read miss generated by the CPU (which occurs when an exclusive block of data is displaced from the cache by a second block). In the shared state only a write by the CPU or an invalidate from another processor can move us out of this state. In the case of transitions caused by events external to the CPU, the state diagram is fairly simple, as shown in Figure S.35. When another processor writes a block that is resident in our cache, we uncondi- tionally invalidate the corresponding block in our cache. This ensures that the next time we read the data, we will load the updated value of the block from memory. Also, whenever the bus sees a read miss, it must change the state of an exclusive block to shared as the block is no longer exclusive to a single cache.

The major change introduced in moving from a write-back to write-through cache is the elimination of the need to access dirty blocks in another processor's caches. As a result, in the write-through protocol it is no longer necessary to pro- vide the hardware to force write back on read accesses or to abort pending mem- ory accesses. As memory is updated during any write on a write-through cache, a processor that generates a read miss will always retrieve the correct information from memory. Basically, it is not possible for valid cache blocks to be incoherent with respect to main memory in a system with write-through caches.

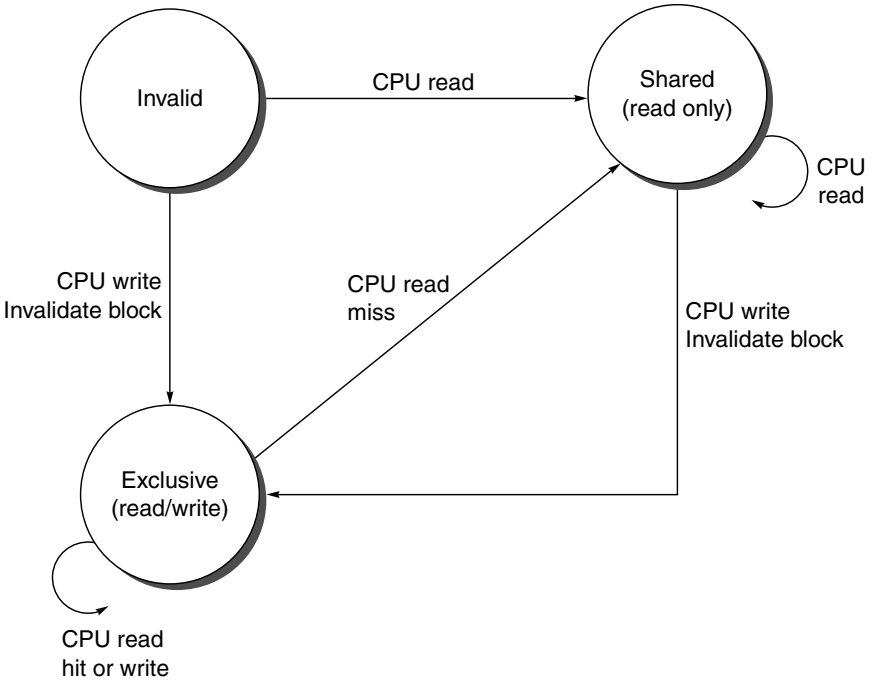

Figure S.34 CPU portion of the simple cache coherency protocol for write-through caches.

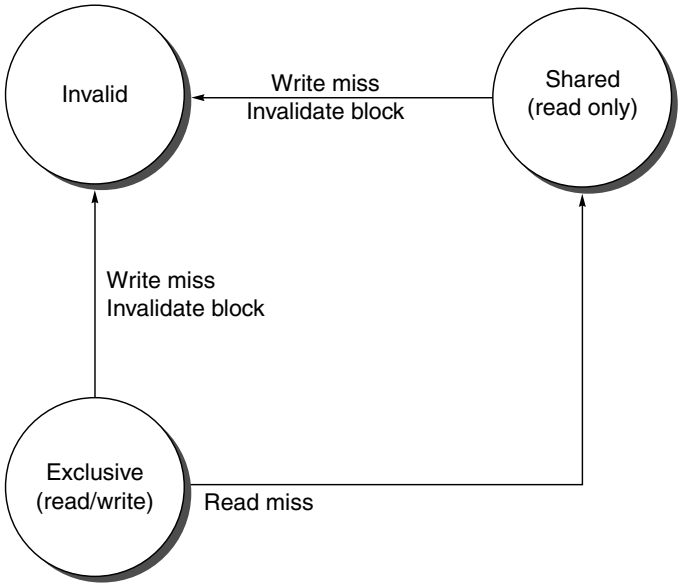

Figure S.35 Bus portion of the simple cache coherency protocol for write-through caches.