ECSE 425 – Tutorial 10

Cache Coherence

Cache Coherence

- In multi-processor systems, data can reside in multiple levels of cache, as well as in main memory. This can cause problems if all CPUs don't see the same value for a given memory location.

Cache Coherence

- To ensure coherence and consistency, you want all caches to see all memory accesses in program order.
- A memory system is **coherent** if it sees memory accesses to a single location in order
  - A read to P following a write to P returns P, regardless of which processor reads/writes
  - Writes are serialized so each processor sees them in the same order
- A memory system is **consistent** if it sees memory accesses to different locations in order

Cache Coherence

Two classes of protocols to ensure cache coherence

- Directory based: a central location (directory) contains the sharing status of all blocks in all caches (more scalable).
- Snooping: each cache listens on a shared bus for events regarding cache blocks (less scalable).

Snooping Protocol

Two ways to ensure cache coherence

- Write invalidate protocol: on a write, broadcast a block invalidation message on the bus so everyone knows their local copy is dirty.
- Write update protocol: on a write, the value is broadcast on the bus and everyone updates their local copy.

Snooping Protocol

- We use write-invalidate + write-back caches
- Track block sharing status
- When writing to a shared block, must first acquire the bus to broadcast the invalidation
- Finding data on a read
  - most recent version of the block might be in another cache (with *write-back* caches)
  - all caches must monitor the address line for *dirty* blocks they might have and update the status of those blocks so they don't use those values
  - See Figure 4.4 (p.209)

Snooping Protocol

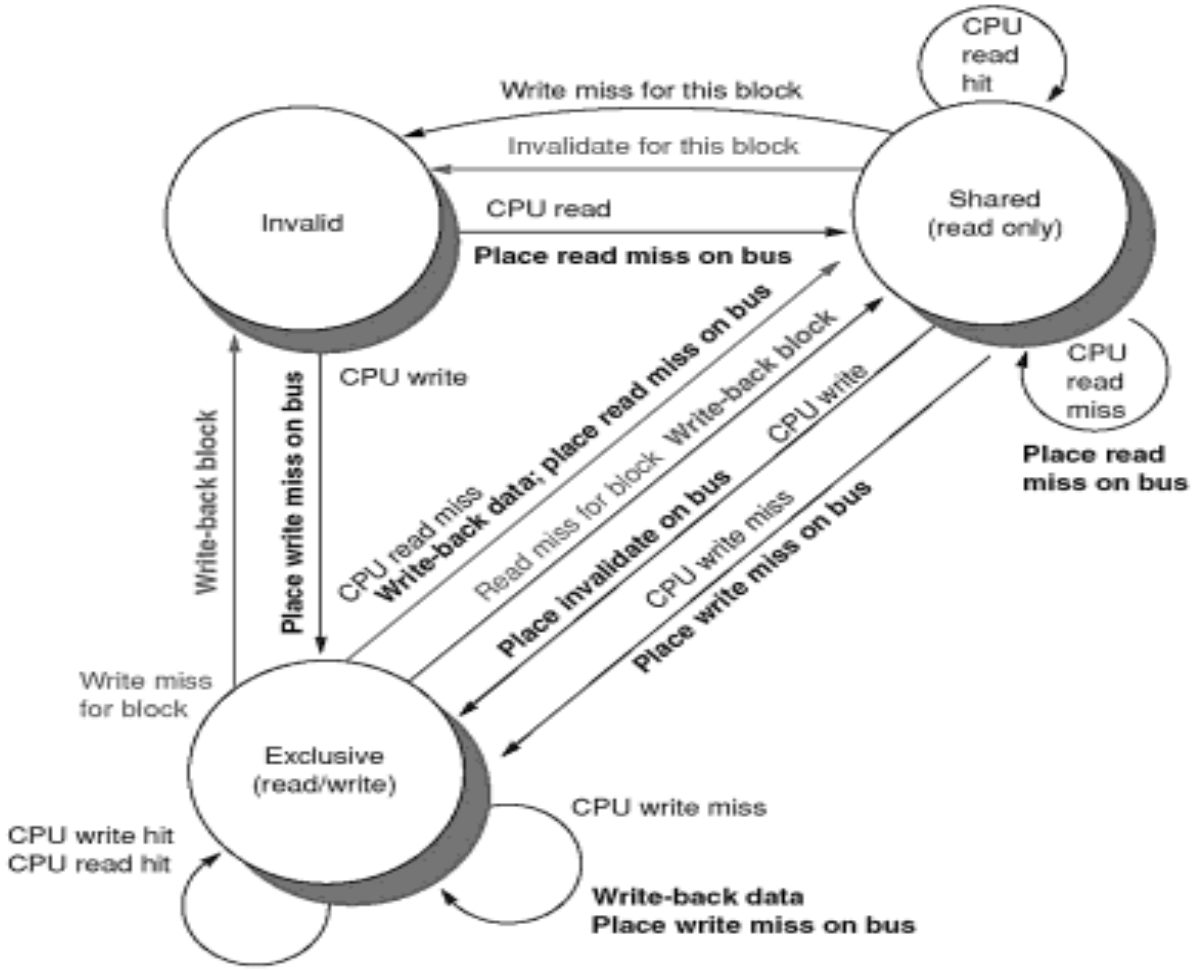
- Each cache block can have 3 states: **Invalid**, **Modified** or **Shared**.

- When a processor writes to a shared block, it must broadcast an invalidation message for this block and then mark the block as exclusive

Snooping Protocol

- When an invalidate or write miss is put on the bus, all other caches invalidate their copy of this block.
- When a read miss is put on the bus, a cache with this block in **Modified** mode will mark it as **Shared** and puts its value on the bus (bypassing main memory).
- See Figure 4.7 (p.215) and make sure you understand it well.

Snooping Protocol



Example

.1 [10/10/10/10/10/10/10] <4.2> For each part of this exercise, assume the initial cache and memory state as illustrated in Figure 4.37. Each part of this exercise specifies a sequence of one or more CPU operations of the form:

```
P#: <op> <address> [ <-- <value> ]
```

where P# designates the CPU (e.g., P0), <op> is the CPU operation (e.g., read or write), <address> denotes the memory address, and <value> indicates the new word to be assigned on a write operation.

Treat each action below as independently applied to the initial state as given in Figure 4.37. What is the resulting state (i.e., coherence state, tags, and data) of the caches and memory after the given action? Show only the blocks that change, for example, P0.B0: (I, 120, 00 01) indicates that CPU P0's block B0 has the final state of I, tag of 120, and data words 00 and 01. Also, what value is returned by each read operation?

a. [10] <4.2> P0: read 120

b. [10] <4.2> P0: write 120 <-- 80

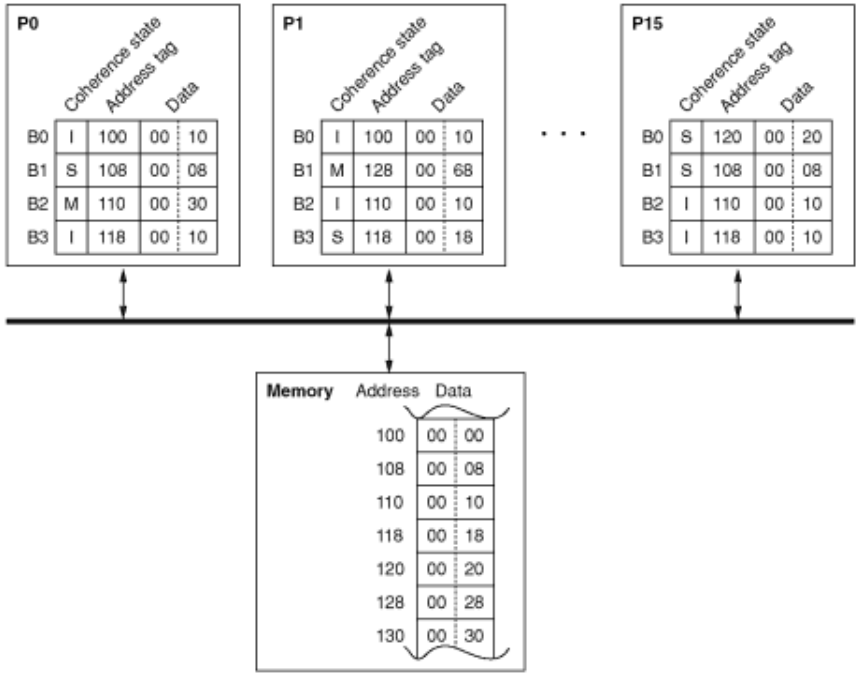


Figure 4.37 Bus-based snooping multiprocessor.

c. [10] <4.2> P15: write 120 <-- 80

d. [10] <4.2> P1: read 110

e. [10] <4.2> P0: write 108 <-- 48

f. [10] <4.2> P0: write 130 <-- 78

g. [10] <4.2> P15: write 130 <-- 78