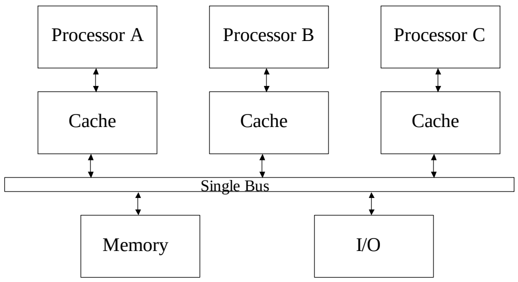
Multiprocessors Connected by a Single Bus



4/16/01 BR 1

Cache Consistency (note: this protocol is different from book)

- How do we maintain cache consistency between these multiple processors?
  - i.e, if Processor A and Processor B caches both have the same data, and Processor A modifies its data, what happens to the data in Processor B?
- Assume a Copyback cache scheme.
  - With single CPU, only needed two status bits per block – ‘V’ (valid bit), and M (modified bit or dirty bit).
  - With multiple CPUs over shared bus, add another status bit – ‘S’ (shared bit).
- We will also take advantage of the fact that processors can see ALL transactions over shared bus
  - Processors can ‘snoop’ the bus – this is sometimes called a ‘snooping’ cache protocol

4/16/01 BR 2

Cache Line States

- Using the 3 status bits: V (valid), M (modified), S (shared), define 5 states:
  - Invalid (I) – cache line not valid
  - Exclusive-Clean (EC) – only one cache module has a copy and it is clean
  - Exclusive Modified (EM) – only one cache module has a copy and it is modified
  - Shared Clean (SC) – same cache line may exist in other modules
  - Shared Modified (SM) - same cache line may exist in other modules, but this cache module is the owner
- A CPU may have to provide a cache block to other processors if the processor has modified the block

4/16/01 BR 3

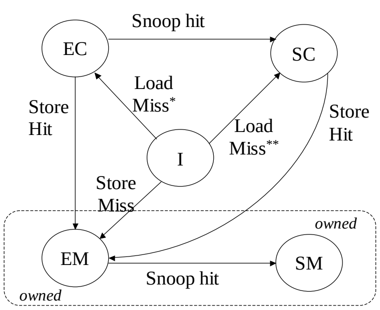
Bus Transactions for Cache Consistency

- Bus transactions
  - Coherent Invalidate – all caches that contain this block should mark it invalid
  - Coherent Read – a read which maintains cache coherency
  - Coherent Write – a write which maintains cache coherency
  - Coherent Write and Invalidate (combined for performance)
  - Coherent Read and Invalidate (combined for performance)

- Signals used cache coherency
  - MSH# : input/output, open drain, shared. Asserted by cache controller in response to coherent read request if this cache has a valid copy of this cache block. More than one cache controller can assert this line.
  - MIH# : input/output, tri-state: Asserted by owner of a cache block to inform main memory that the current coherent read should be ignored – data will be supplied by this cache controller.

Cache State Diagram (Copy-back scheme)

Snoop hit: another processor requests block

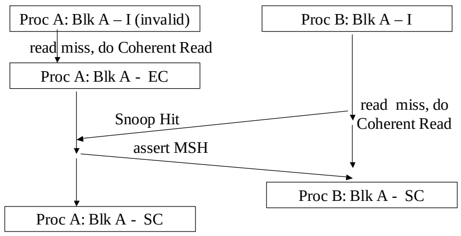


* MSH# not asserted
** MSH# asserted

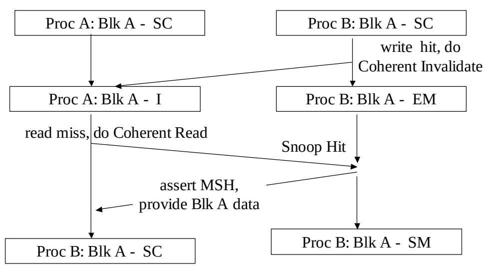
Notes on Cache States

- If same block is shared among processors
  - only one processor (the owner) will be in Shared Modified state (SM)
  - other processors will have the block in Shared Clean State
- If owner of a SM block does a write, then the processor will issue an invalidate transaction on the bus that will cause other processors to invalidate the block. Cache block then goes to EM state.
  - Must invalidate other blocks because the local write by the processor will not be 'seen' by other processors
- If a processor does a Write to a block in a Shared Clean state must do the same as above.

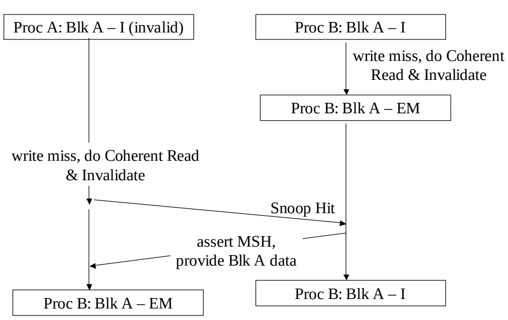
Example Sequences



Example Sequences (cont)



Example Sequences (cont)



A Possible Problem?

- What if two Processors both have block A in SC (shared clean) state.
- Both processors have a write hit to block A. What happens?
  – Each processor tries to access the bus to do an Invalidate transaction.
  – The arbiter will give the bus to one of the two processors (simultaneous bus request).
  – The winning processor will set its block A state to EM.
  – The losing processor postpones the write, invalidates its cache. The losing processor then issues a Read & Invalidate
  – Winning processor provides block A, changes block state to I
  – Losing processor ends up with Block A in EM state.
- No problem, caches still coherent.

Multiprocessor Synchronization

- Lock variables (semaphores) used to coordinate tasks between multiple processors
- An *atomic swap* operation can be used to implement locks
  – swaps a register with a memory location atomically (i.e., cannot be interrupted during the swap).
- Let '0' be the 'unlocked' state. A lock is a fixed memory location.
- A processor checks to see if the lock is free (= 0). To grab the lock, swap the memory location with a register that is non-zero (i.e, register value = '1').
  – cache coherency protocol and atomic operation ensures that two processors executing the swap command simultaneously will update the lock variable sequentially, which hands off the lock correctly.

Lock Example (a little different from Figure 9.7 which assume write-thru cache)

Step	Proc. P0	Proc. P1	Proc P2	Bus Activity
1	Has lock in register, (SC)	Spins, testing if lock =0 (SC)	Spins, testing if lock =0 (SC)	None
2	Sets lock=0 (EM)	Spins, testing if lock =0 (SC)	Spins, testing if lock =0 (SC)	Write invalidate from P0
3	Lock = 0 (EM)	Cache miss (I)	Cache miss (I)	Arbiter picks P2 for miss
4		waits for bus	Coherent Read, Lock=0, SC	Cache miss for P2 satisfied
5		Coherent Read, Lock=0, SC	Swap, read lock=0, set to '1'.	Cache miss for P1 satisfied
6	Lock = 1, but not in cache (I)	cache miss (I) because of cache invalidate from P2	Has Lock, lock = 1, update shared data	Write Invalidate from P2
7		Swap reads lock = 1. Writes lock = 1	Owns lock	
		Lock = 1 (EM), spins, testing for lock = 0.	Lock = '1', but not in cache (I)	Write invalidate from P1