# Appendix A

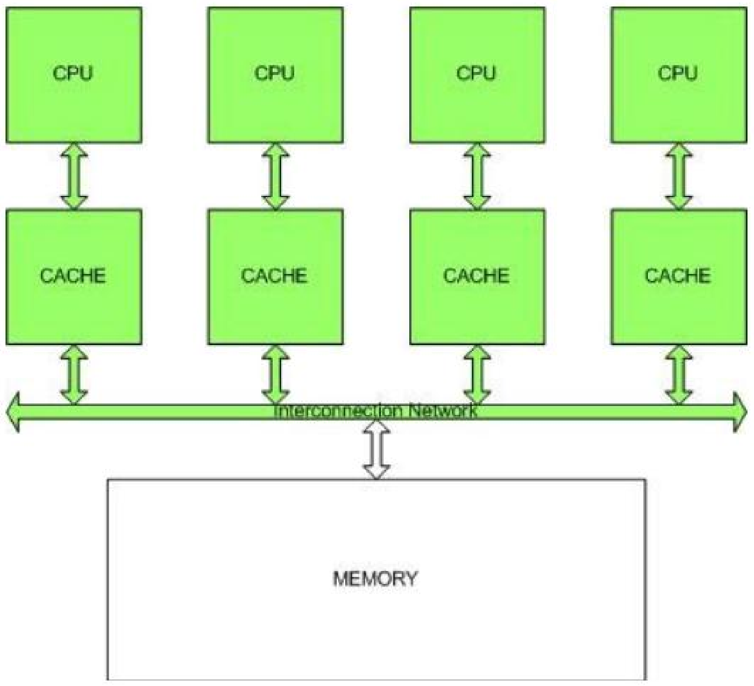


**Figure 6 - Cache associated with different processors in Shared Memory system**

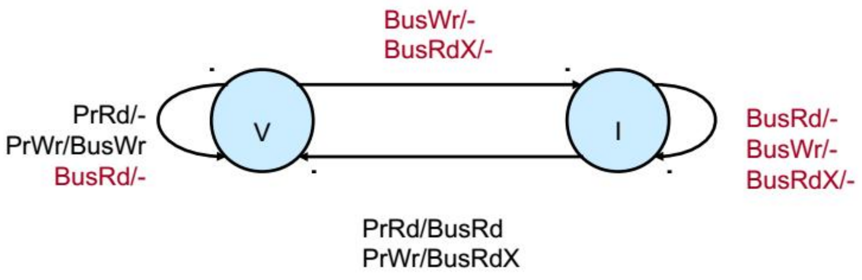
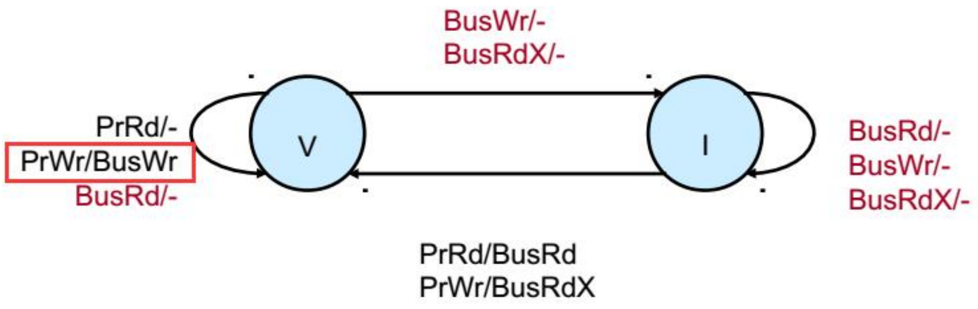


Abbildung 1: VALID – INVALID Cache Line State Transition Diagram

As above figures show, cache controller will receive instructions from both the processor and the bus.



***PrWr/BusWr***: The left means the cache controller receives the instruction from the If cache controller receives the instruction processor write from the processor and the cache controller find the specific data line is in the cache and the state of this specific data is valid. Then the cache controller will try to obtain the control of the bus and lock the bus (which may have to wait for some time if other cache controllers of other caches have already locked the bus.)

Once the cache controller obtains the control of the bus, the cache controller sends Bus write instruction on the bus. And all other cache snoop (hear) this instruction from the bus, and if anyone among them (except the cache which controls the bus now) find the specific cache line is valid in its own cache, then it turns the state of the cache line into invalid. If anyone among them (except the cache which controls the bus now) finds the specific cache line invalid in its own cache or it does not find the specific cache line in its own cache at all, then the cache controller does nothing.

At the same time (Once the cache controller obtains the control of the bus), the cache controller will update the data in its own cache（1cycle） and also update the data in the memory(100 cycles)

## ***requirement 0***

cache adopts write-through, write-allocate policy:

***When write hit happens***

***write-through,*** often distinguish cache designs:

● ***Write Through*** - the information is written to both the block in the cache and to the block in the lower-level memory.

***When write miss happens***

***write-allocate write miss***

● ***Write Allocate*** - the block is loaded on a write miss, followed by the write-hit action.

## ***requirement 2***

the format should behave like:

```
cpu0
#Reads    RHit/    RMiss/    #Writes Whit/ WMiss/ Hitrate
cpu1
#Reads    RHit/    RMiss/    #Writes Whit/ WMiss/ Hitrate
…
…
```

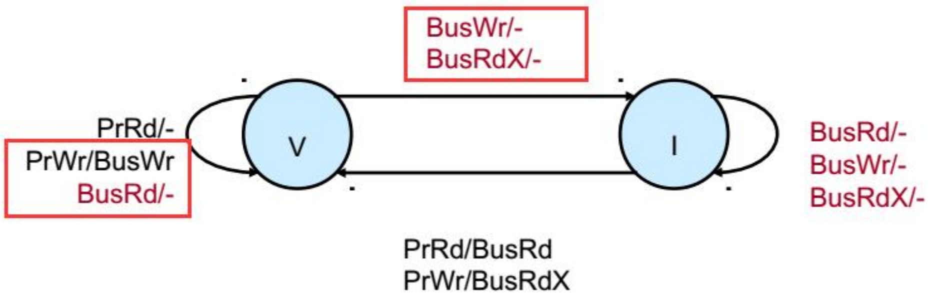
…
cpu8
#Reads RHit/ RMiss/ #Writes Whit/ WMiss/ Hitrate

Total
#Reads RHit/ RMiss/ #Writes Whit/ WMiss/ Hitrate
Sum of 1 to 8

Average
Average of 1 to 8

***requirement 3***

Per cache, while snooping on the bus, count and report the total number of read/write operations on the bus, in which ***the desired address is found in the snooping cache*** (probe read hits and probe write hits)



Just calculate above.

***requirement 4***
4. Average memory access time
**Including the waiting time of the lock of the bus+ memory access per request (write+read)**
***write hit***
***write-through,*** often distinguish cache designs:
● ***Write Through*** - the information is written to both the block in the cache and to the block in the lower-level memory.
***write miss:***
***write-allocate write miss***
● ***Write Allocate*** - the block is loaded on a write miss, followed by the write-hit action.

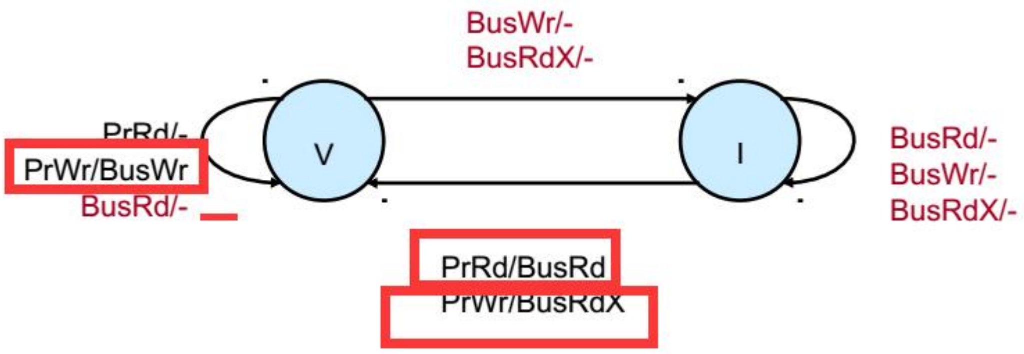
One write miss, 2 memory access(200cycles) +waiting time for bus acquisition+1 bus send instruction(1 cycle for bus write)+ 1 cycle for fetching data
One write hit, 1 memory access(100cycles) +waiting time for bus acquisition+1 bus send instruction(1 cycle for bus write)+ 1 cycle for fetching data

One read miss, 1 memory access; 1 memory access(100cycles) +1 cycle for fetching data

One read hit, 1 cache cycle; 1 cycle for fetching data

***requirement 5***

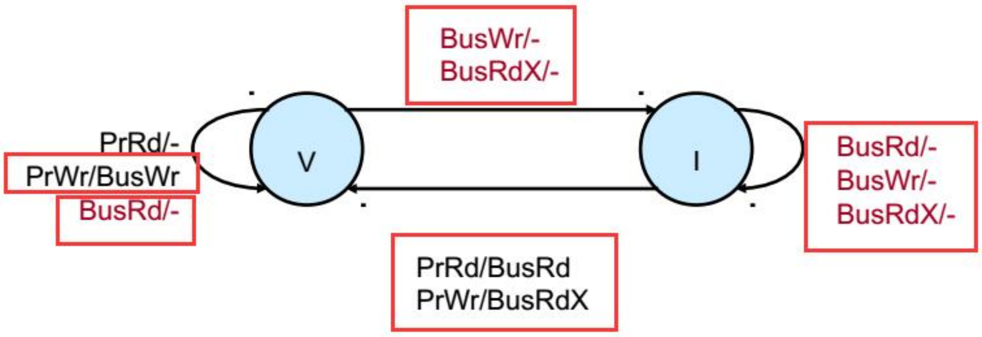
5. Bus acquisition statistics: total number of wait cycles trying to lock the bus, and the average waiting time per access (= #waits/(bus reads + bus writes))



***requirement 6***

6. Number of bus reads, bus writes, and total accesses (reads+writes)

I specify a benchmark in systemc



One write miss, 2 memory access;
One write hit, 1 memory access;
One read miss, 1 memory access;
One read hit, no memory access