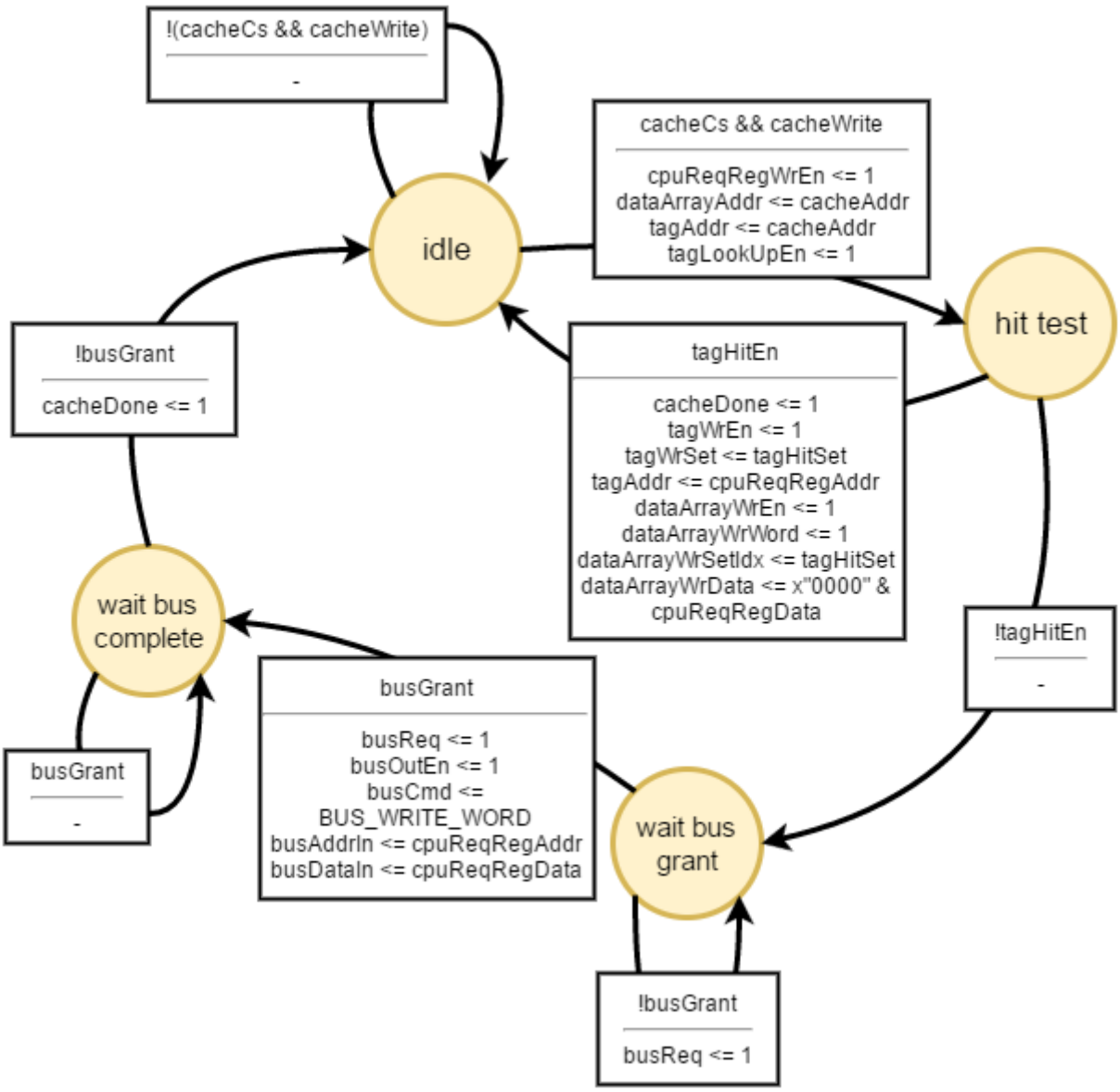
Homework 3

Modification to the datapaths and state machines

As asked we changed our datapaths and our state diagrams to make the cache become a write-through and write-no-allocate cache. We dropped the write-back behaviour.

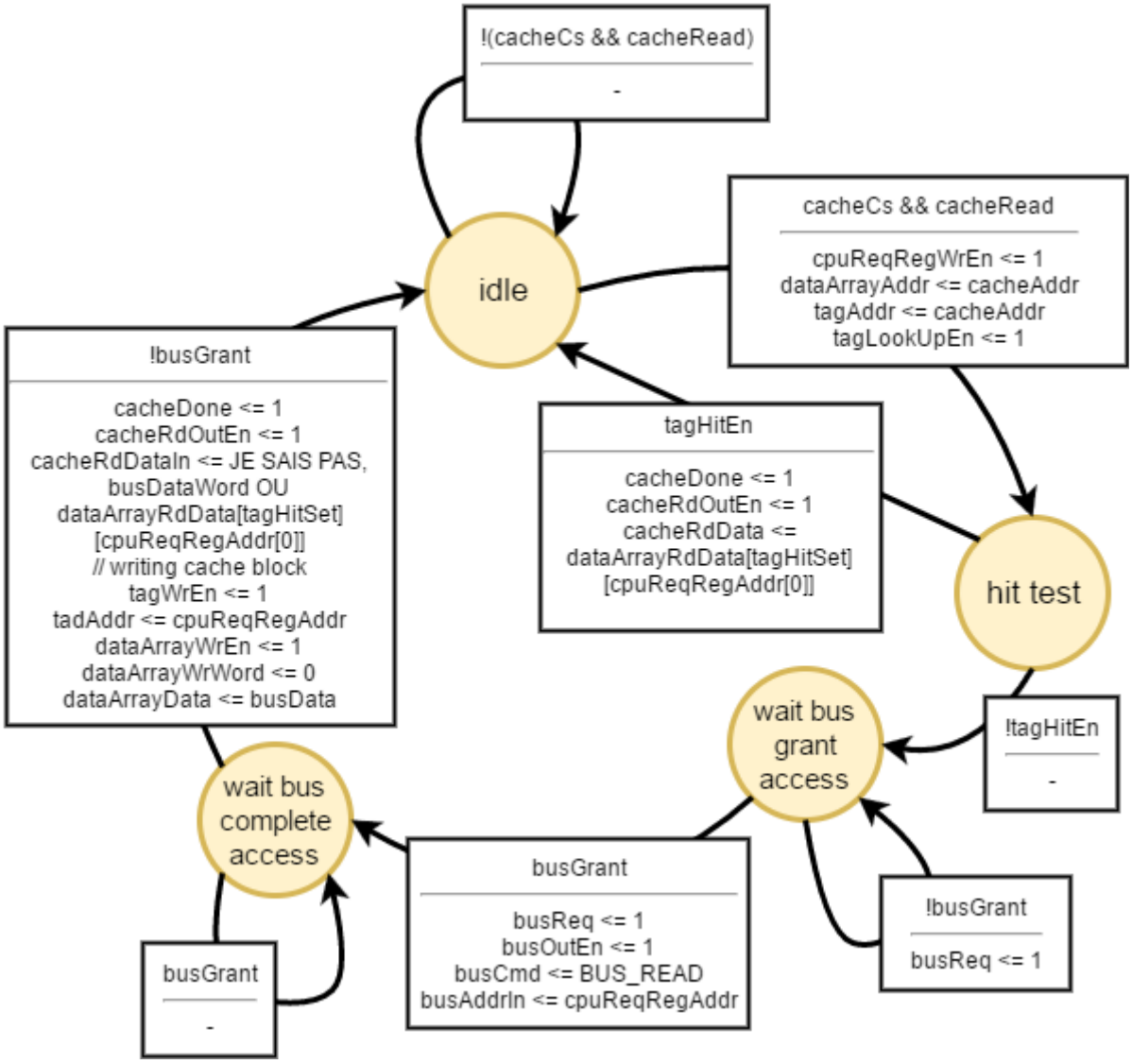
1) State machines modifications

Cache Controller Read



Here we removed the states « wait bus grant writeback » and « wait bus grant complete », and changed the transitions to be adequate. Indeed, this two states are only useful for writebacks as indicates their names, and now we can only use the write-through arrow going from « wait bus complete » to idle. All the transaction now are directly sent to the memory when we miss.

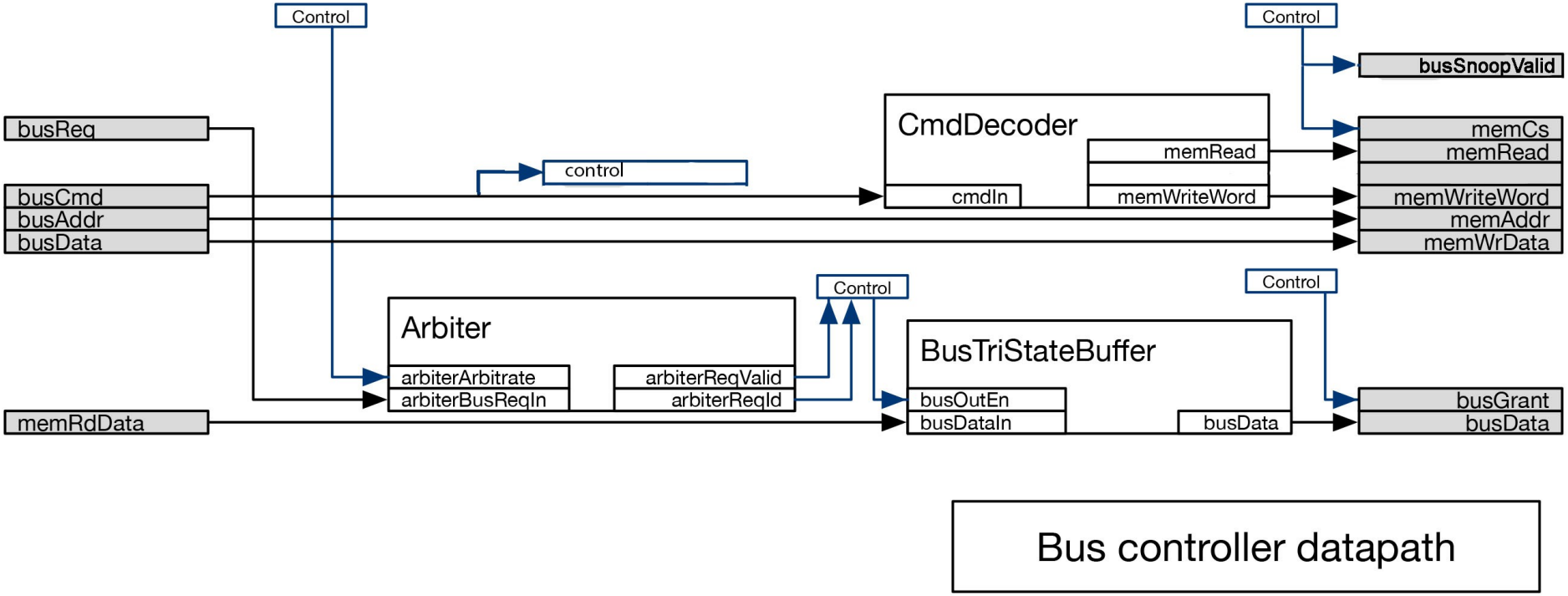
Cache Controller Write



Here, we only removed the « tagWrDirty » signal, because it does not more exist because we removed the write-back and replaced it with a write-through.

The Bus Controller State machine stays unchanged.

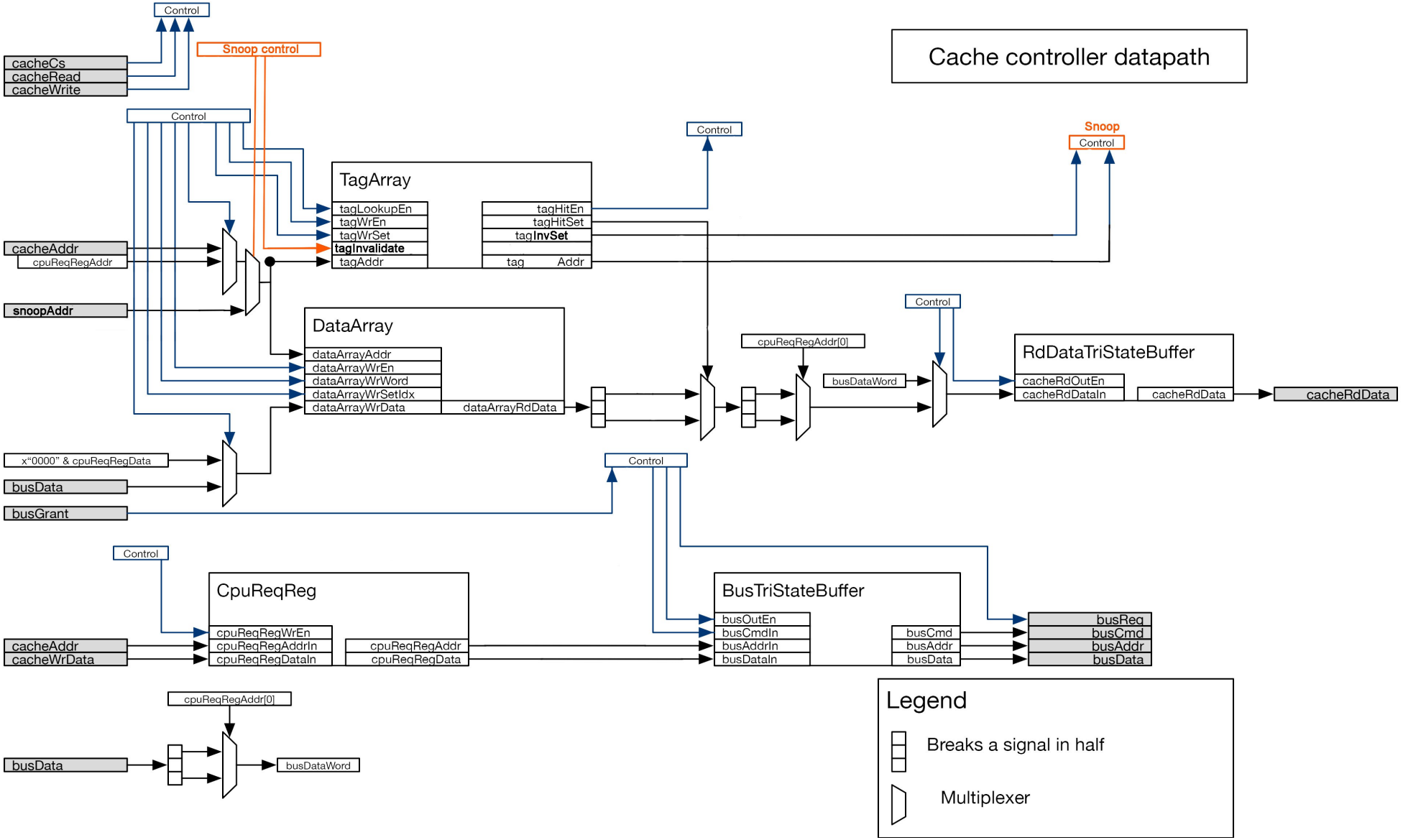
2) Datapaths modifications



Bus controller datapath

This is the new datapath for the *Bus Controler*. We have removed the *memWrite* out signal because now that the cache is write-through we will always write to memory word by word, so only keeping the *memWriteWord* signal is enough.
We also have added a new output signal named *busSnoopValid* used to allow the snooping machine to snoop. When this signal is zero the snoop machine stands by.
Also, we send the *busCmd* signal to the control to check if the signal is a *WRITE* and not a read, and in a case of a write we run the snoop machine.

Cache Controller Datapath and new TagArray interface



For the cache controller datapath, more modifications had to be done.

First, we had to edit the ***TagArray* interface**. It now doesnt take a *tagWrSetDirty* input anymore, but instead a *tagInvalidate* signal used to tell the *tagArray* if we want to read some tag line (0) or invalidate it (1). In accordance to that modification, the *tagInvalidate* signal also commands a mux to choose between the usual *cacheAddr* and the new *snoopAddr* input to send to the *tagAddr* input.

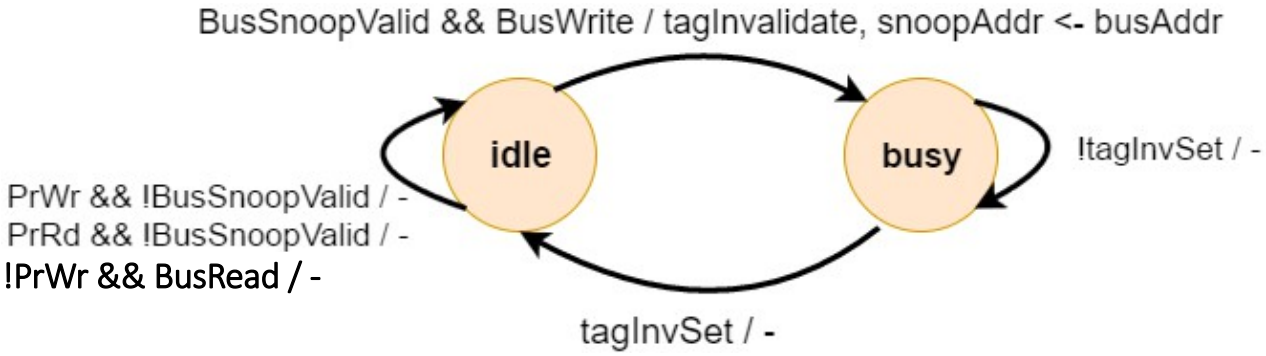
The outputs of *TagArray* are also modified : we don't need no *tagVictimDirty*, *tagVictimSet* and *tagVictimAddr* anymore as the *VictimReg* should be removed (it's only useful for write-backs). We can add 2 outputs signals named *toInvSet* to give some feedback to the control about the line invalidation status, and a *tagAddr* signal that represents the line that has been invalidated.

As said earlier, we had to remove the *victimReg*, which is a write-back feature. We also removed all the victim signal occurrences in the datapath, often used to command muxes (which we also removed). Now the *CpuReqReg* outputs are directly mapped to the *TriStateBuffer*

Registers to keep / remove

The registers to remove are the VictimReg.
The registers to keep are the others.

New Snoop State Machine



The new snoop machine stays in state *idle* when the *BusSnoopValid* signal is off. This signal is sent when we are authorized to snoop the bus. The corner case when we have a processor write or read signal coming and when we are yet not allowed to snoop the bus is treated in the left arrow (we stay in *idle* and ignore the *PrWr* and *PrRd* signal).

When we are authorized to snoop the bus and we see a *busWrite* signal, we can go to the *busy* state and send a *tagInvalidate* signal to the *TagArray* and send the *busAddr* to invalidate to the *snoopAddr* signal (that will also be used by the *TagArray*). Then we stay in the *busy* state and wait for the *tagInvSet* signal from the *TagArray* saying the invalidation has been done.

Another corner case we treated is when we recieve a *PrWr* and a *BusRd* at the same time, the *PrWr* is done first because it contains the youngest value. That means we only proceed to a *BusRd* when there is no *PrWr.*

Eventually, when the invalidation is over, we go back to the *idle* state.