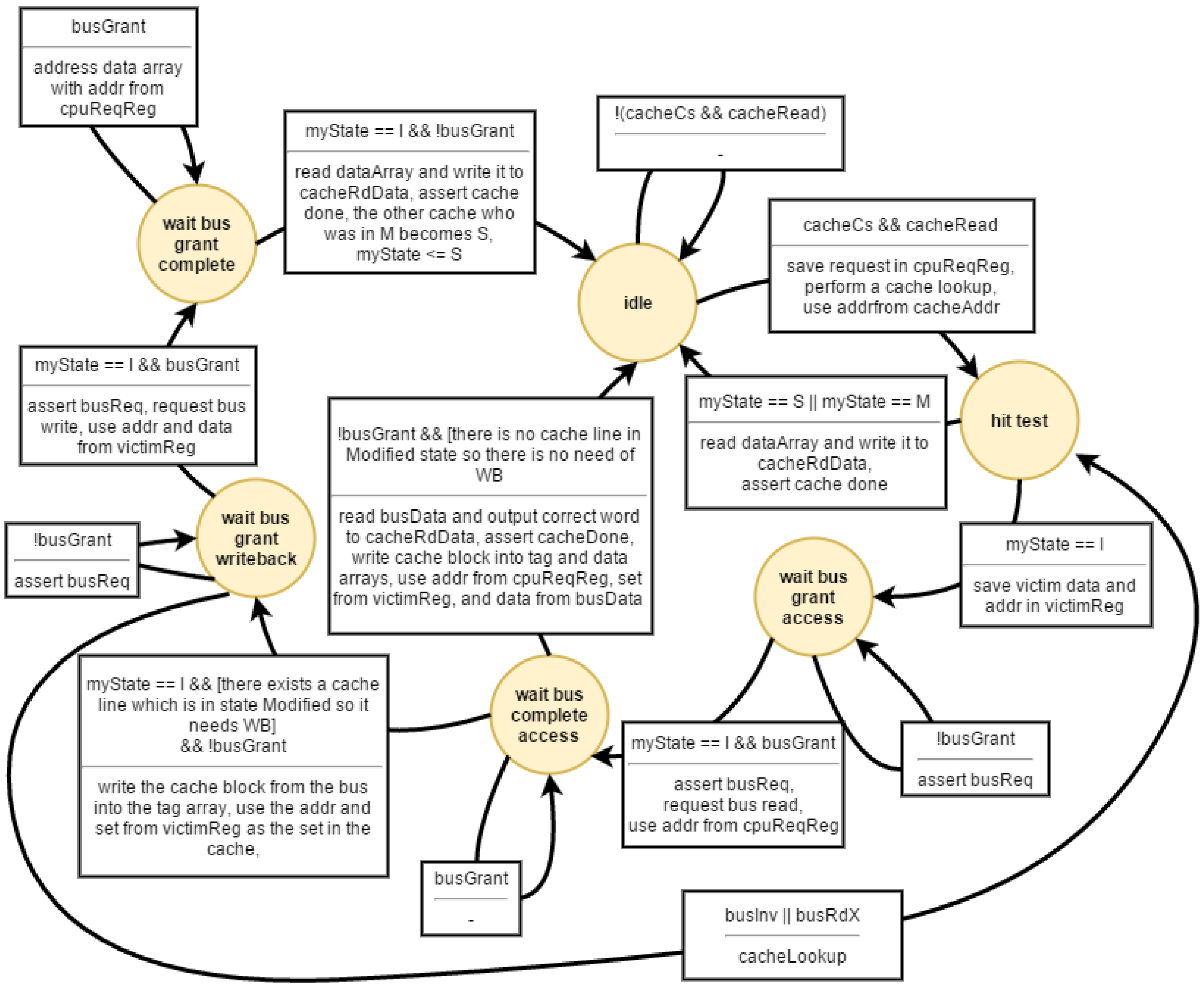
Tim Nguyen (250961) et Lucie Perrotta (246970)

ArchMul - Lab 5

New cache read machine


busGrant
address data array with addr from cpuReqReg
myState == I && !busGrant
read dataArray and write it to cacheRdData, assert cache done, the other cache who was in M becomes S, myState <= S
!(cacheCs && cacheRead)
-
cacheCs && cacheRead
save request in cpuReqReg, perform a cache lookup, use addrfrom cacheAddr
wait bus grant complete
idle
myState == I && busGrant
assert busReq, request bus write, use addr and data from victimReg
!busGrant && [there is no cache line in Modified state so there is no need of WB
read busData and output correct word to cacheRdData, assert cacheDone, write cache block into tag and data arrays, use addr from cpuReqReg, set from victimReg, and data from busData
myState == S || myState == M
read dataArray and write it to cacheRdData, assert cache done
hit test
!busGrant
assert busReq
wait bus grant writeback
myState == I
save victim data and addr in victimReg
wait bus grant access
myState == I && [there exists a cache line which is in state Modified so it needs WB] && !busGrant
write the cache block from the bus into the tag array, use the addr and set from victimReg as the set in the cache,
wait bus complete access
myState == I && busGrant
assert busReq, request bus read, use addr from cpuReqReg
!busGrant
assert busReq
busGrant
-
busInv || busRdX
cacheLookup

In the new read machine, we stay *idle* until there is some activity on the cache (Cs or Read) and then proceed to a "hit test", which will actually check **what state the cache line is in**. By "what state", we mean that we look for the *asked cache line by the CPU to the cache* and check its state (M, S, I).

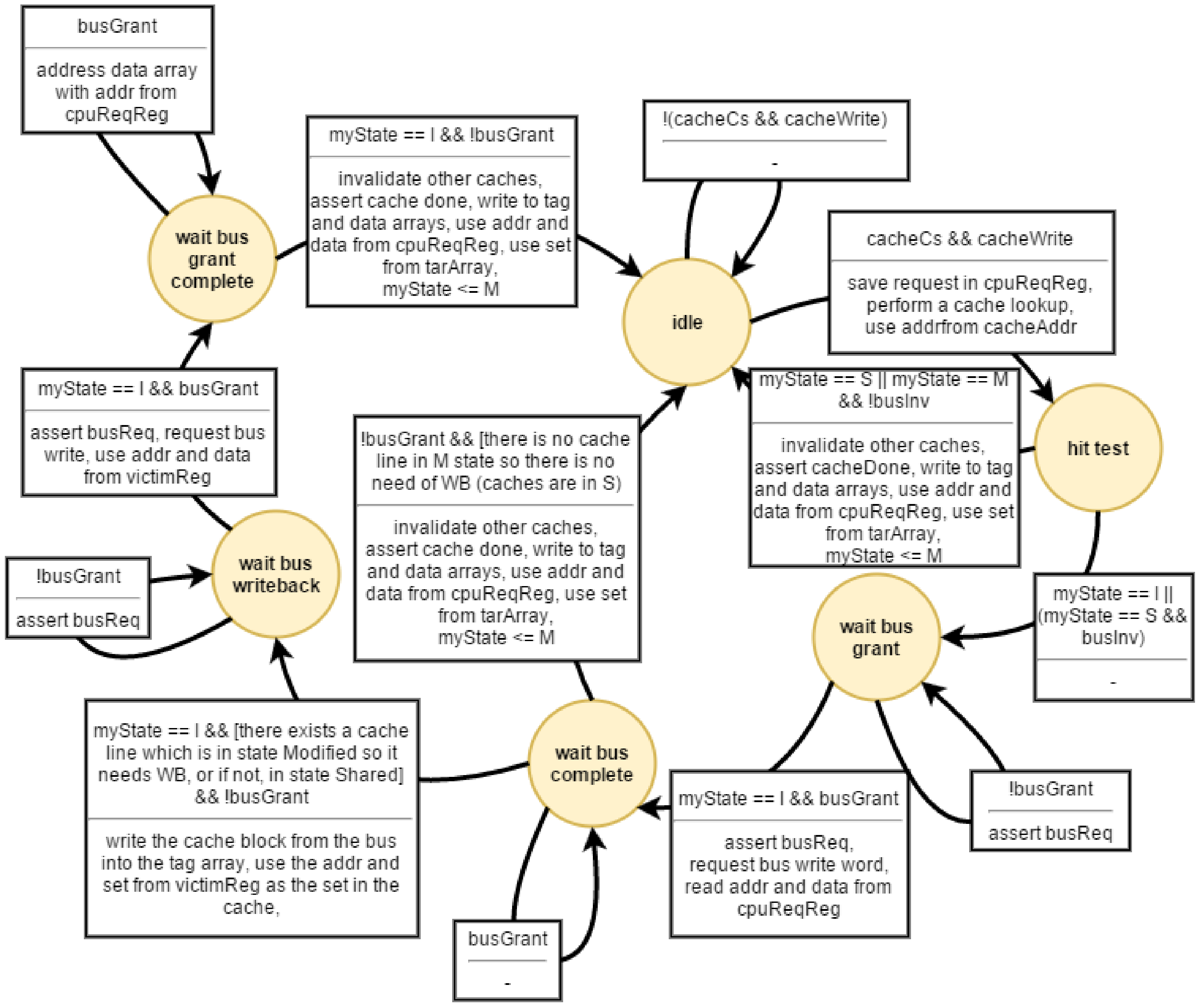
If it's **M** or **S** then we go back to idle as the CPU can read the value, which is correct.
If the line state is **I**, we must go to some further steps.

We must ask the bus to give us the correct value in the line, which can be either in memory or in another cache. Therefore, we ask for access like usual and ask the bus (*wait bus complete access*) **if some other cache has a copy of the line** in M state.

If it is not the case, then we load the line from memory and go back to idle while becoming **S**.
If another cache owns it, we ask for a writeback and go to state *wait bus grant writeback*, we wait for the bus to proceed, and when we get the new value, we become **S** (and the other cache which was in **M** as well).

If a write happens while we are in *wait bus grant writeback* state, we can't just continue and become *shared*. We must go back to hit test state and proceed again, taking this new write in account.
If there is no write, we can just proceed.
Finally we go back to idle.

New cache write machine


busGrant
address data array with addr from cpuReqReg
myState == I && !busGrant
invalidate other caches, assert cache done, write to tag and data arrays, use addr and data from cpuReqReg, use set from tarArray, myState <= M
!(cacheCs && cacheWrite)
-
wait bus grant complete
idle
cacheCs && cacheWrite
save request in cpuReqReg, perform a cache lookup, use addrfrom cacheAddr
myState == I && busGrant
assert busReq, request bus write, use addr and data from victimReg
!busGrant && [there is no cache line in M state so there is no need of WB (caches are in S)
invalidate other caches, assert cache done, write to tag and data arrays, use addr and data from cpuReqReg, use set from tarArray, myState <= M
myState == S || myState == M && !busInv
invalidate other caches, assert cacheDone, write to tag and data arrays, use addr and data from cpuReqReg, use set from tarArray, myState <= M
hit test
!busGrant
assert busReq
wait bus writeback
wait bus grant
myState == I || (myState == S && busInv)
-
myState == I && [there exists a cache line which is in state Modified so it needs WB, or if not, in state Shared] && !busGrant
write the cache block from the bus into the tag array, use the addr and set from victimReg as the set in the cache,
wait bus complete
myState == I && busGrant
assert busReq, request bus write word, read addr and data from cpuReqReg
!busGrant
assert busReq
busGrant
-

Same as before, we wait for some CPU activity to go to *hit test* state. Again, we check what state the cache line is in.
If the state is **S** or **M** and that nothing happens on the bus, then we simply go back to the idle state.
If we are in state **S**, me must also invalidate other **S** caches and become **M**.

If we are in **I** state, we go to next step and ask the bus if another cache owns the correct value.

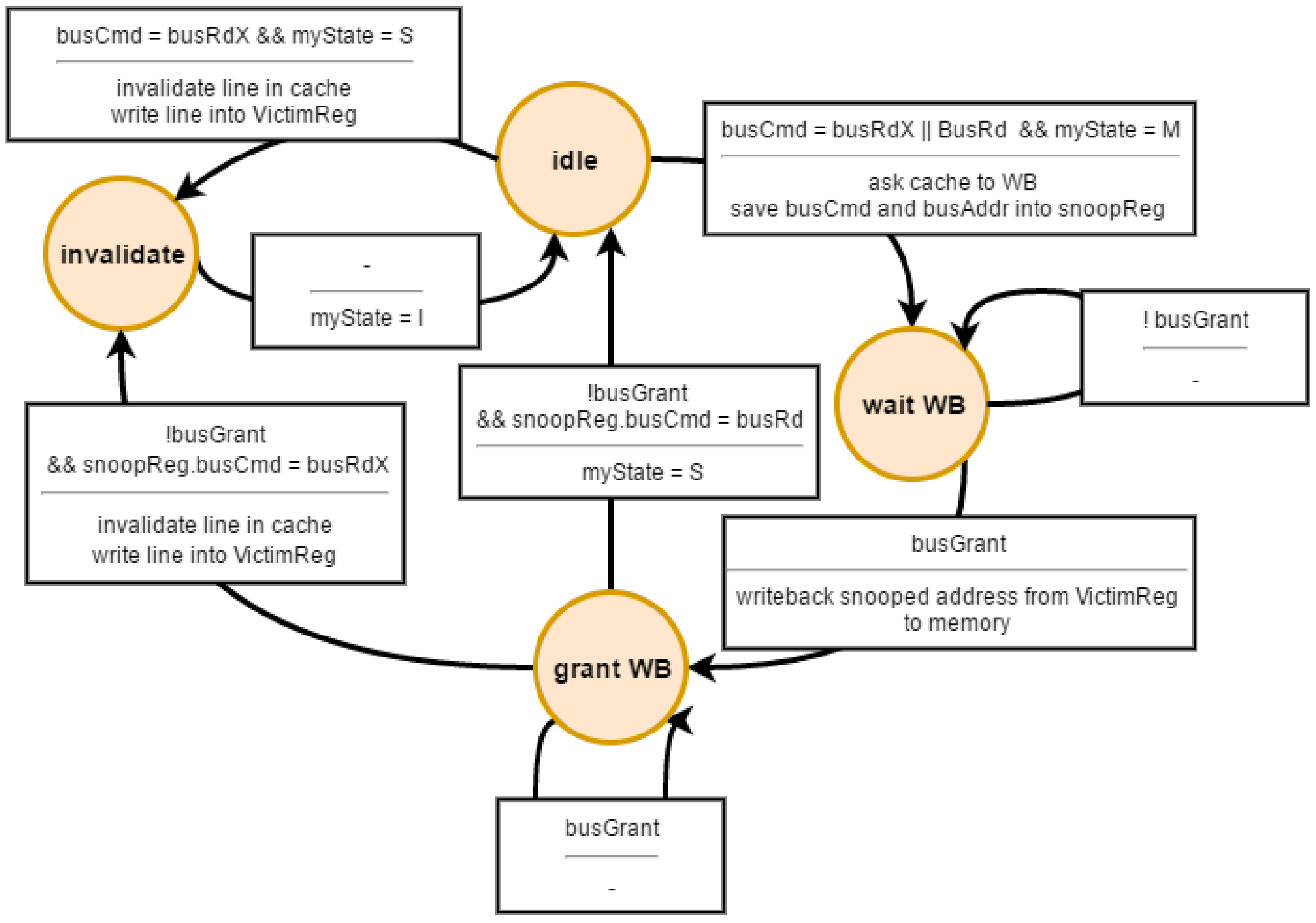
Now if we are in state **S** and a *BusInv* happends on the bus (transcient state !), then we proceed as if we were in **I** state (that is ask who owns the value now), as the given graph indicates.

Then the bus answers if there is some cache in **M** state. If not, all the other caches are either in **S** (valid) or **I** (invalid). So we just become **M** and invalidate other caches.
If there is some **M** cache, we must ask it to writeback to the memory before invalidating it and becoming M. All the steps to achieve this are the same as usual.

New snoop machine

In the new snoop state machine, we use a signal called "*myState*". This signal is generated from the *TagArray* By passing to it the *cmdAddr* we just have snooped on the bus, and ask it to send back the state (M, S, I) of the line corresponding to the *cmdAddr* in the *TagArray*.

We also use a *SnoopReg* to store the Snooped address and command while doing our instruction. The address and command are red during the snoop and then stored and used to know if a writeback is needed for example.
Also, whenever we want to invalidate some line in the cache (that is, set it to *myState* = I), we copy it into the *VictimReg*.


busCmd = busRdX && myState = S
invalidate line in cache
write line into VictimReg
idle
invalidate
-
myState = I
busCmd = busRdX || BusRd && myState = M
ask cache to WB
save busCmd and busAddr into snoopReg
wait WB
! busGrant
-
!busGrant
&& snoopReg.busCmd = busRd
myState = S
!busGrant
&& snoopReg.busCmd = busRdX
invalidate line in cache
write line into VictimReg
busGrant
writeback snooped address from VictimReg to memory
grant WB
busGrant
-