**Control:**

The control consists of two main components, a register write enable control unit, and a next state control unit. The register write control unit outputs the write enable signals for all the internal register based on the inputs which are: miss, cpuReq, dirty, current\_state, and last state. The output signals from the register write enable unit are: cpuWr\_we, cpuAddr\_we, cpuDin\_we, cpuDout\_we, cpuReady\_we, L2Addr\_we, L2Dout\_we, LRU\_we, prevState\_we, repAddr\_we, repData\_we, hit\_we, miss\_we, evict\_we. The cpuWr\_we, cpuAddr\_we, and cpuDin\_we signals all need to be high when there is a CPU request and it is in the idle state, because at the beginning of a new CPU request we need to register if it is a write, the data being fed in, and the address. The cpuDout\_we and cpuReady\_we signals need to be high when there is hit and it is in the compare tag state, because a hit in the compare tag state means that the data is now in the cache and another request can be issued. L2Addr\_we is high when the current state is either writeback or allocate and L2Dout is high when the current state is allocate. This is because the allocate and writeback states interface with memory so we need to provide the address to memory at these states. The memory outputs a value after the allocate state which is why L2Dout\_we is high in the allocate state. The repAddr\_we and repData\_we are high when there is a miss and it is in the compare tag state. We have these registers for the replacement address and data because we may need to write the data back that is being replaced if it is dirty. The hit\_we signal is high when the previous state is idle, the current state is compare tag, and there is not a miss. This is because we only want the hit to count once, and there is only a transition from idle to compare tag once per request. The miss\_we signal is determined by the last state being compare tag and the current state being either writeback or allocate. On a miss, the state machine transitions from compare tag to either writeback or allocate, so counting miss on these transitions ensures that a miss is counted only once.

The other control unit is the next state control outputs the next state and has inputs: current\_state, miss, cpuReq, L2Ready, dirty, prev\_state. This control unit follows the image provided in the book and uses the same states: idle, compare tag, writeback and allocate. The next state is idle when the current state is compare tag and there is a hit because this means the data is in the cache. The next state is compare tag when the current state is allocate, the previous state is allocate and the memory is ready. We used the previous state because we wanted the FSM to stay in allocate for at least one cycle before transitioning to compare tag. The FSM goes to writeback when the current state is compare tag, there is a miss, and the evicted data is dirty. If these three things happen, that means the evicted data has changed in the cache, so it needs to be written back in memory. Finally, the next state is allocate when: the current state is writeback, the last state was writeback, and the memory is ready, or when the current state is compare tag, there is a miss, and the evicted data is not dirty, or when the current state is allocate and the memory is not ready.