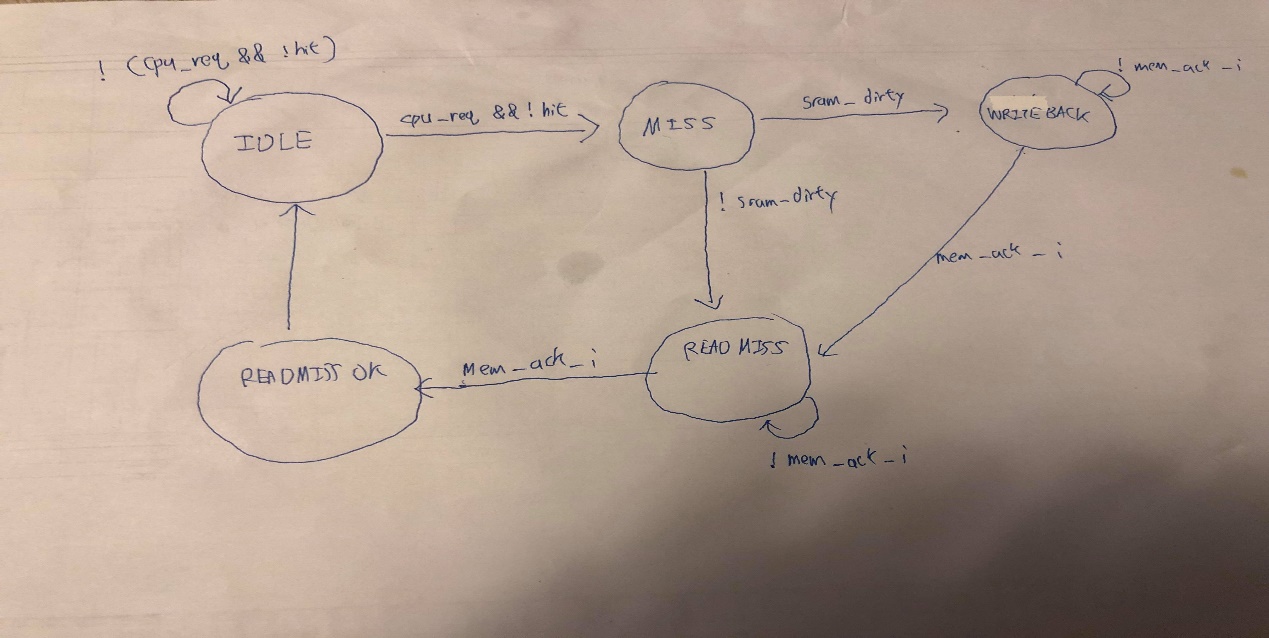
**Modules explanation**

**Dcache\_controller.v**

For read hit data, the sram\_cache\_data is assigned to it. Since it is a read hit, the data, the data is in the cache, therefore it is readed from there. The block size is 256 bits, but the CPU only wants one part of data, which is 32 bits, from it. The 32 bits are extracted according to the cpu\_offset, the data needed is from bit (cpu\_offset\*8) to bit (cpu\_offset\*8+31). The data will be sent back to CPU by cpu\_data\_o. For write hit data, the 32 bit data is provided by CPU and will be written to certain part of the cache block according to cpu\_offset. The 32 bits being replaced in the 256 bit block is bit (cpu\_offset\*8) to bit (cpu\_offset\*8+31).

The dcache controller has 5 states. The FSM is showed below.



When the instruction is not read(load) or write(store) or the read write is a hit, the state is remained IDLE. All state registers are 0. When the instruction is a read(load) or write(store) and it is a miss, the state is changed to MISS. In MISS state, the block needs to be replaced, so if the block is dirty, a write back is needed. Memory access needs to be enabled (mem\_enable = 1), memory write back should be enabled (mem\_write = 1, write\_back = 1). If not dirty, then proceed to READMISS state and enable memory access (mem\_enable = 1) to read from memory. If memory isn’t ready, stay in READMISS state. If memory is ready, the data should be written to cache, enable cache\_write and close mem\_enable, then proceed to READMISSOK. Since the miss has been handled, set every state register to 0 and return to IDLE state.

**Dcache\_sram.v**

A 16 row 2 column LRU bit is created to match the 2-way cache. Initialize data, tag and LRU to 0. If instruction is read write and is either cache write or write hit, the cache needs to be written. The write hit condition will be handled first. If tag input is same with the cache tag and it is valid, then the block will be written. The LRU of the block is set to 1 and valid bit set to 1. The LRU of the other block in the same set is set to 0. Then read miss will be handled. The block with LRU=0 will be replaced by memory data and LRU set to 1 and also valid. The LRU of the other block in the same set is set to 0.

If it is read hit and write hit, the block’s data with the same tag and valid bit on will be assigned to the output. The data will be used to generate either the write hit data for the cache or read hit data for the cpu. The tag input is the tag output and hit\_o set to 1, since it is a hit. The LRU of that block will be set to 1 and the LRU of the other block in the same set will be set to 0.

If the tag doesn’t match, it is a miss. The block with LRU bit 0 will be replaced by new data, so if the dirty bit is 1, old block needs to be written back. The data and tag of the block will be assigned to the output and hit\_o set to 0. For default situation, which both LRU=0, the hit\_o is set to 0.

If the instruction isn’t read or write (enable\_i = 0), data\_o, tag\_o and hit\_o are all set to 0.

**CPU.v**

Replace data memory with dcache. The CPU interface in dcache is connected with same old wires connected to data memory before. The data memory interface in dcache is connected with the new CPU module inputs and outputs provided in testbench. The connection between CPU and data memory module is done in testbench.

**Testbench.v**

Copy initialized pipeline registers and registers from project 1. Initialize data memory. Change tag[23:0] to [24:0] to include valid bit.

**Difficulties Encountered and Solutions in This Project**

Difficulties: Can’t understand the part of code provided by TA.

Solution: Thanks to 夏同學’s graph, and by reviewing concepts like write back and write allocate, I can finally track the code and understand it properly step by step.

Difficulties: The program counter will stop and always remain at 12.

Solution: Use gtkwave to track the signals and eventually found out the rst\_i of dcache was not connected in CPU module.

**Development environment**

My OS is win10, and compiler is iverilog. My IDE is vscode and I compiled under the

powershell terminal