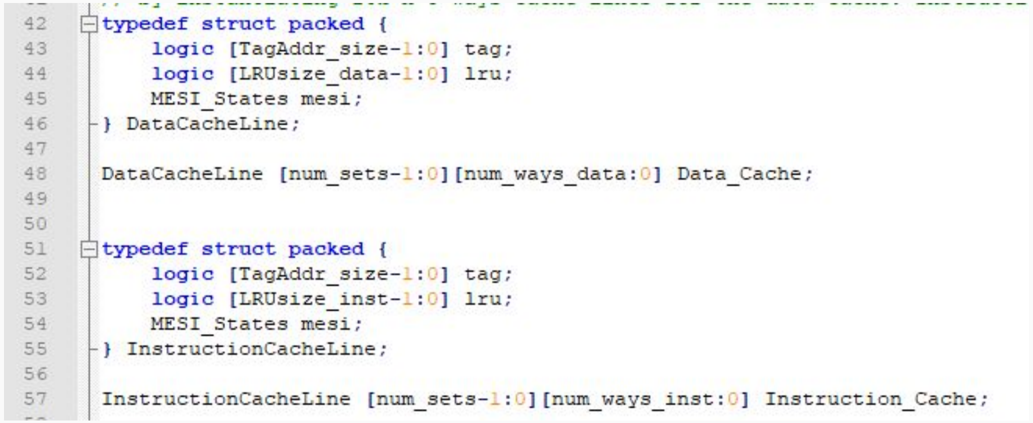
ECE485/585 Final Project

Team 3: Natalie Nguyen, Zack Fravel, Megha Jacob, Karla Barraza Lopez

Cache Design

To access set 0,
way 4 of a cache:

Data_Cache[0][4]

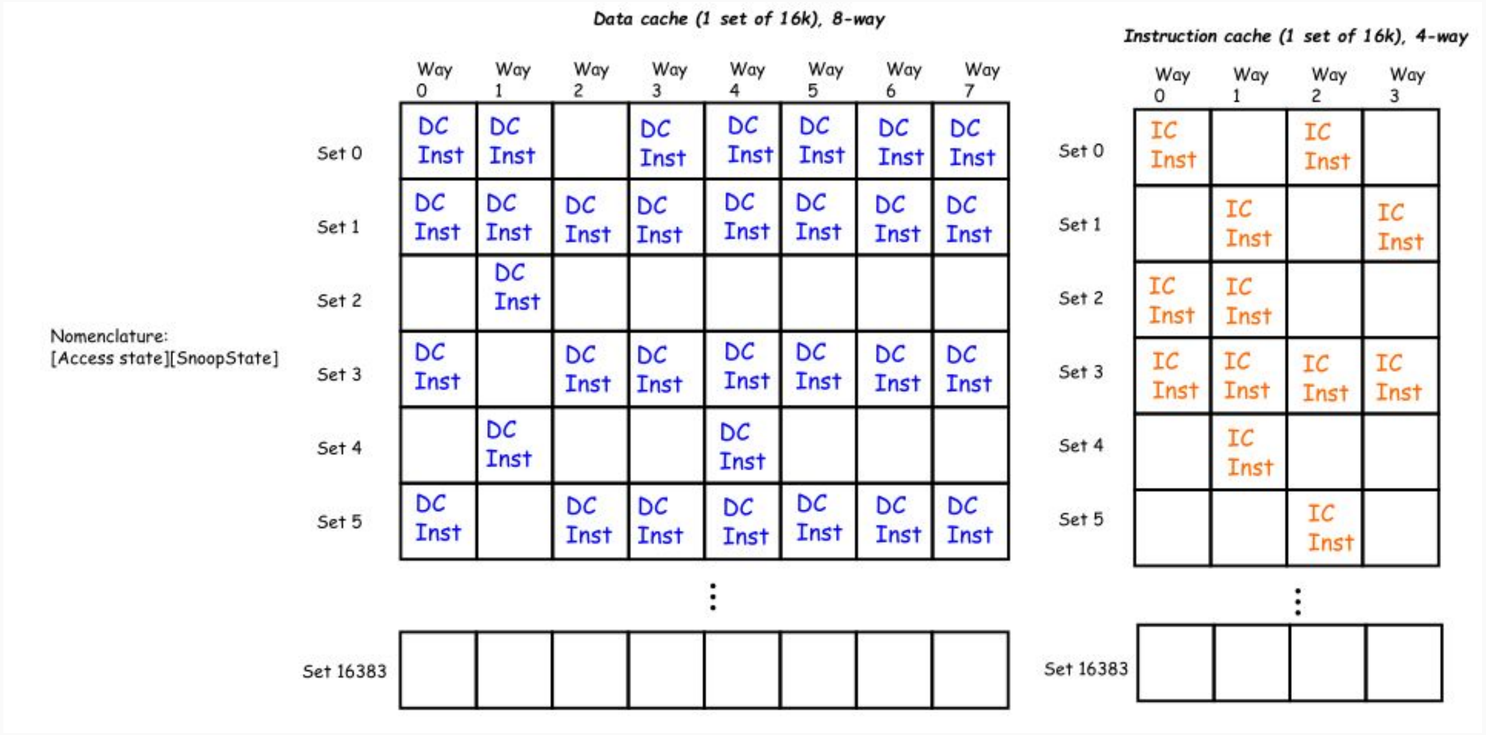


```
typedef struct packed {
    logic [TagAddr_size-1:0] tag;
    logic [LRUsize_data-1:0] lru;
    MESI_States mesi;
} DataCacheLine;

DataCacheLine [num_sets-1:0][num_ways_data:0] Data_Cache;


typedef struct packed {
    logic [TagAddr_size-1:0] tag;
    logic [LRUsize_inst-1:0] lru;
    MESI_States mesi;
} InstructionCacheLine;

InstructionCacheLine [num_sets-1:0][num_ways_inst:0] Instruction_Cache;
```

Cache Design

To access set 0, way 4 of a cache:

Data_Cache[0][4]



Data cache (1 set of 16k), 8-way

	Way 0	Way 1	Way 2	Way 3	Way 4	Way 5	Way 6	Way 7
Set 0	DC Inst	DC Inst		DC Inst	DC Inst	DC Inst	DC Inst	DC Inst
Set 1	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst
Set 2		DC Inst						
Set 3	DC Inst		DC Inst	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst
Set 4		DC Inst			DC Inst			
Set 5	DC Inst		DC Inst	DC Inst	DC Inst	DC Inst	DC Inst	DC Inst
⋮								
Set 16383								

Instruction cache (1 set of 16k), 4-way

	Way 0	Way 1	Way 2	Way 3
Set 0	IC Inst		IC Inst	
Set 1		IC Inst		IC Inst
Set 2	IC Inst	IC Inst		
Set 3	IC Inst	IC Inst	IC Inst	IC Inst
Set 4		IC Inst		
Set 5			IC Inst	
⋮				
Set 16383				

Code Organization

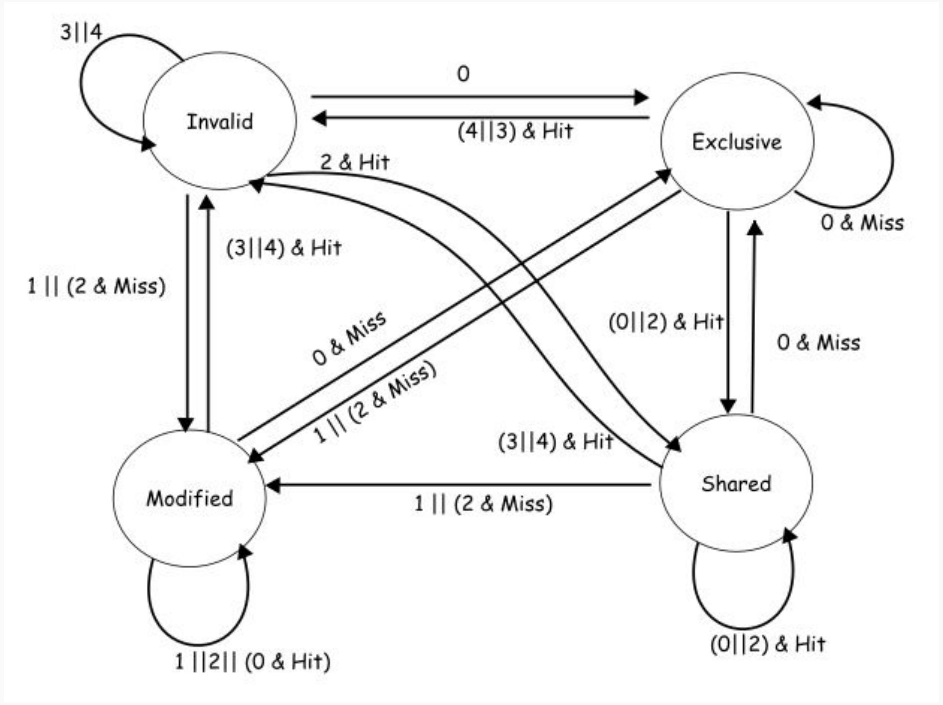
- Bitbucket used for version control
- File Structure:
  - /RTL/
    - Contains our SystemVerilog Modules and Top Level Testbench
    - **Cache_defs.sv**, **L1_Cache.sv**, **trace_parser.sv**, **tb_top.sv**
  - /STIMULUS/
    - Contains all our test cases to run through the design after compiling
  - Makefile
    - CLI for the user to run desired test cases

Assumptions

- LRU bits count up from (000 to 111)
    - ‘000’ indicates the most recently used way on the cache line.
    - ‘111’ indicates the least recently used way on the cache line.
- HIT=0 and HITM =0
- First read or cache hit: State = I -> E;   Second read: State = E -> S

  (Read is done by another processor)
- All assumptions given in Final Project Explanation

State Machine



State Change Mechanism

n	Hit/Miss	PS	NS	Display	
0- Read req to L1 data cache	Hit	**M**	**M**		*Ist read -> E state* *If PS =E, then next read is done by another processor and NS = S* *Assuming HIT & HITM =0*
		E	**S**		
		S	**S**		
		I	**E**	Read from L2	
	Miss	**M**	**E**	Write to L2 Read from L2	
		E	**E**	Read from L2	
		S	**E**	Read from L2	
		I	**E**	Read from L2	

State Change Mechanism

n	Hit/Miss	PS	NS	Display	
1 – Write req to L1 data cache	Hit	**M**	**M**		*First time write is a miss & will be a " write through policy" ->Write to L2* *else "write-back" policy*
		E	**M**		
		S	**M**		
		I	**M**	Read for Ownership from L2	
	Miss	**M**	**M**	Write to L2 Read for Ownership from L2	
		E	**M**	Read for Ownership from L2	
		S	**M**	Read for Ownership from L2	
		I	**M**	Read for Ownership from L2	

State Change Mechanism

n	Hit/Miss	PS	NS	Display	*Instrn Cache - Read only -> No M state* *Assuming HIT & HITM =0*
2 – Read req to L1 instr cache	Hit	**M**			
		E	**S**		
		S	**S**		
		I	**E**	Read from L2	
	Miss	**M**			
		E	**E**	Read from L2	
		S	**E**	Read from L2	
		I	**E**	Read from L2	

State Change Mechanism

n	Hit/Miss	PS	NS	Display	Comments
3 – Invalidate Command frm L2	Hit	**M**	**I**	Return data to L2	*PS = E, No Invalidate command from other processor*
		E	**I**		
		S	**I**		
		I	**I**		
	Miss	**-**	**-**		
4 – Data req frm L2 (RFO)	Hit	**M**	**I**	Return data to L2	
		E	**I**		
		S	**I**		
		I	**I**		
	Miss	**-**	**-**		
8 – Clear cache and reset	-	**M**	**I**	Write to L2	
	-	x	**I**		

Demonstration

- Run through Scenario 1 and Scenario 2 given in “Final Project Explanation”
- Simulate with assigned .trace file
- Show corner cases and typical operation
    - Typical Operations - *set0.trace*
    - Reset - *reset.trace*
    - Incorrect Commands - *incorrectCommands.trace* and *set0.trace*

Expected outputs for L1 Data Cache (set0.trace)

	L1 Data Cache					
	Set 0	way 0	way 1	way 2	way 3	way 4
0 10000000	Tag	100				
	LRU bits	000				
	MESI	E				
0 20000000	Tag	100	200			
	LRU bits	001	000			
	MESI	E	E			
1 10000000	Tag	100	200			
	LRU bits	000	001			
	MESI	M	E			
0 20000000	Tag	100	200			
	LRU bits	001	000			
	MESI	M	S			
0 30000000	Tag	100	200	300		
	LRU bits	010	001	000		
	MESI	M	S	E		
0 10000000	Tag	100	200	300		
	LRU bits	000	010	001		
	MESI	M	S	E		
0 40000000	Tag	100	200	300	400	
	LRU bits	001	011	010	000	
	MESI	M	S	E	E	
0 50000000	Tag	100	200	300	400	500
	LRU bits	010	100	011	001	000
	MESI	M	S	E	E	E
0 40000000	Tag	100	200	300	400	500
	LRU bits	010	100	011	000	001
	MESI	M	S	E	S	E

	L1 Data Cache								
	Set 0	way 0	way 1	way 2	way 3	way 4	way 5	way 6	way 7
0 60000000	Tag	100	200	300	400	500	600		
	LRU bits	011	101	100	001	010	000		
	MESI	M	S	E	S	E	E		
1 30000000	Tag	100	200	300	400	500	600		
	LRU bits	100	101	000	010	011	001		
	MESI	M	S	M	S	E	E		
0 70000000	Tag	100	200	300	400	500	600	700	
	LRU bits	101	110	001	011	100	010	000	
	MESI	M	S	M	S	E	E	E	
0 80000000	Tag	100	200	300	400	500	600	700	800
	LRU bits	110	111	010	100	101	011	001	000
	MESI	M	S	M	S	E	E	E	E
1 90000000	Tag	100	900	300	400	500	600	700	800
	LRU bits	111	000	011	101	110	100	010	001
	MESI	M	M	M	S	E	E	E	E
3 40000000	Tag	100	900	300	400	500	600	700	800
	LRU bits	111	001	100	000	110	101	011	010
	MESI	M	M	M	I	E	E	E	E
4 60000000	Tag	100	900	300	400	500	600	700	800
	LRU bits	111	010	101	001	110	000	100	011
	MESI	M	M	M	I	E	I	E	E
1 10300000	Tag	100	900	300	103	500	600	700	800
	LRU bits	111	010	101	000	110	001	100	011
	MESI	M	M	M	M	E	I	E	E

	L1 Instruction Cache				
	Set 0	**way 0**	**way 1**	**way 2**	**way 3**
2 19800000	**Tag**	198			
	LRU bits	00			
	MESI	E			
2 19900000	**Tag**	198	199		
	LRU bits	01	00		
	MESI	E	E		
2 19900000	**Tag**	198	199		
	LRU bits	01	00		
	MESI	E	S		
2 20100000	**Tag**	198	199	201	
	LRU bits	10	01	00	
	MESI	E	S	E	
2 20200000	**Tag**	198	199	201	202
	LRU bits	11	10	01	00
	MESI	E	S	E	E
2 20300000	**Tag**	203	199	201	202
	LRU bits	00	11	10	01
	MESI	E	S	E	E

Expected outputs for L1 Instruction Cache (set0.trace)

Questions?