Issue Queue - FIFO					
	#	inst	PC	pred_addr	pred_branch
wptr	0	-	-	-	-
rptr_1	1	0x....	0x...	0x...	1
rptr_2	2	0x...	0x...	0x...	0
	...	...	...	...	...
	n-1	0x...	0x...	0x...	0

Register Status - Table		
#	busy	ROB_index
1	0	-
2	1	2
...	...	...
31	1	7

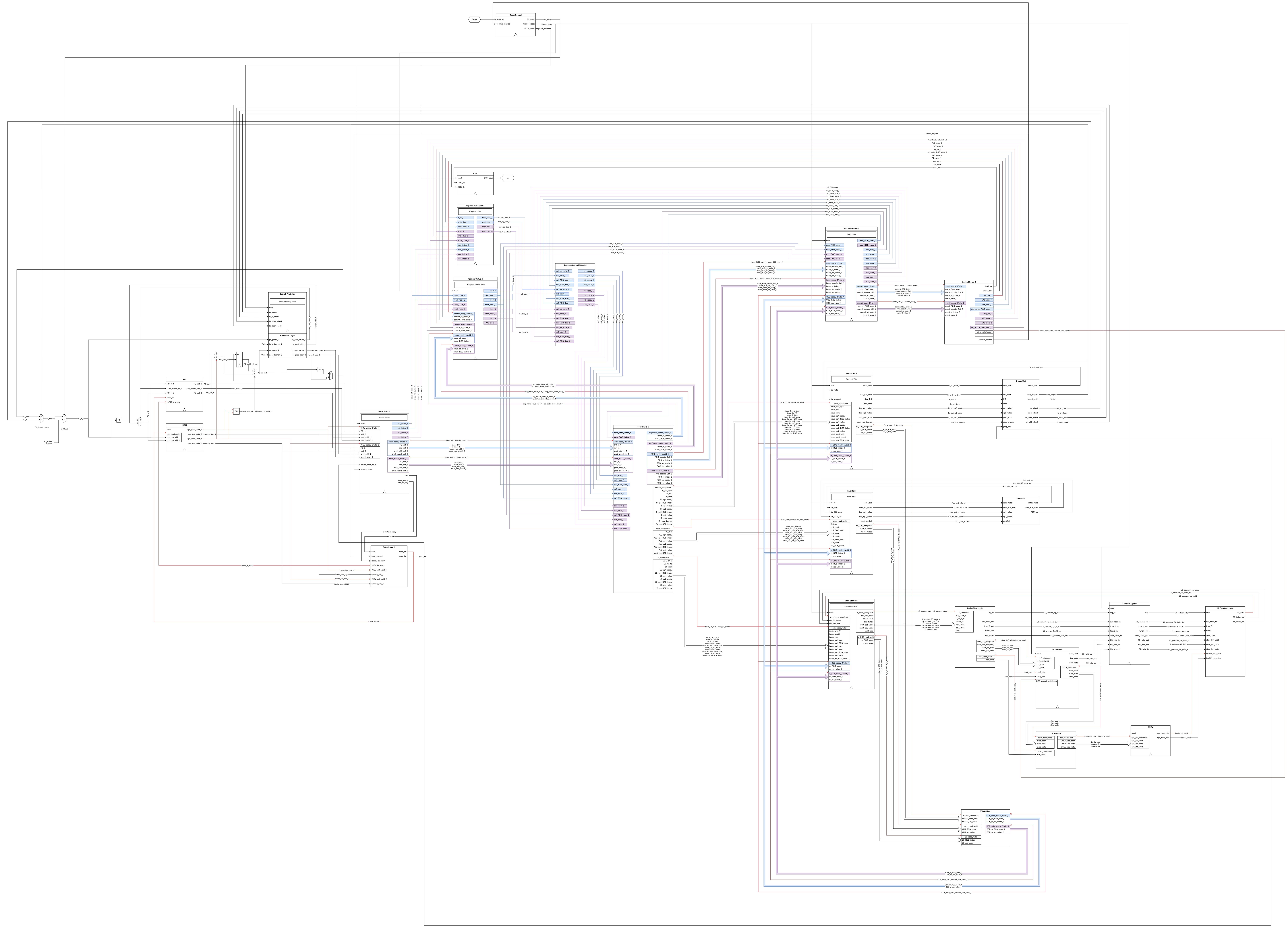
ALU Reservation Station - Data Structure												
#	valid	busy	ALUSel	op1_ready	op1_ROB_index	op1_value	op2_ready	op2_ROB_index	op2_value	res_ready	res_ROB_index	res_value
0	no	-	-	-	-	-	-	-	-	-	-	-
1	yes	no	add	no	2	-	yes	-	15	no	3	-
...	...	...	...	...	...	...	...	...	...	...	...	...
n-1	yes	yes	xor	yes	-	5	yes	-	-20	no	1	-

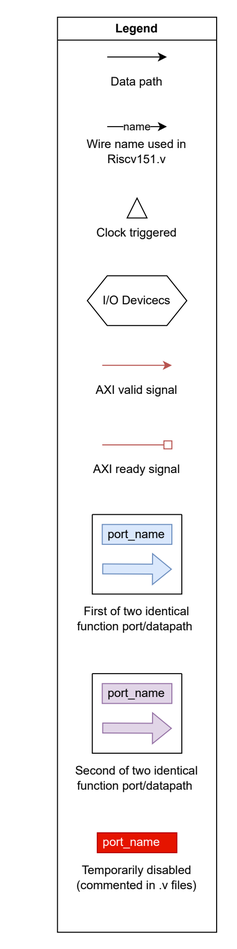
Branch Reservation Station - FIFO															
	#	inst_type	op1_ready	op1_ROB_index	op1_value	op2_ready	op2_ROB_index	op2_value	imm	PC	pred_addr	pred_branch	res_ready	res_ROB_index	res_value (mispredicted)
wptr	0	-	-	-	-	-	-	-	-	-	-				
rptr	1	bge	no	2	-	yes	-	15	16	0x1000000	0x10000010	yes	no	5	-
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	n-1	bltu	yes	-	5	yes	-	-20	-20	0x1000FFFF	-	no	no	7	-

Load/Store Reservation Station - FIFO													
	#	L_or_S (inst[5])	funct3	op1_ready	op1_ROB_index	op1_value	op2_ready	op2_ROB_index	op2_value	imm	res_ready	res_ROB_index	res_value
wptr	0	-	-	-	-	-	-	-	-	-		-	-
rptr	1	0	lw	no	2	-	-	-	-	0	no	2	-
	...	...	...	...	...	...	...	...	...	...	...	...	...
	n-1	1	sw	yes	-	0x1000000	yes	-	0x2000000	-20	no	4	-

Re-Order Buffer (ROB) - FIFO					
	#	opcode_5bit	rd_index	res_ready	res_value
wptr_2	0	-	-	-	-
rptr_1	1	5'b.....	13	0	-
rptr_2	2	5'b.....	0	1	0
	...	...	...	...	...
wptr_1	n-1	-	-	-	-

Store Buffer				
	#	address[31:0]	data[31:0]	write[3:0]
wptr	0	-	-	
rptr	1	0x...	101	0001
	...	...	...	
	n-1	0x...	999	1100





Remarks

1. **IMEM** should allow PC fetch and instruction output in the same cycle
2. **Branch RS** should clear all entries once the head entry needs to change PC
3. When fetch a register, if register status table indicates a ROB entry will write to it, fetch unit should check if it is already available in **ROB**. If so, need to copy the value to **RS**
4. jal & jalr can save PC + 4 to **ROB** at issue stage, then calculate new PC value in **Branch RS** and its adder
5. auipc can use **ALU RS** to calculate PC + imm
6. lui can directly save imm in **ROB**
7. csrrwi don't need **RS**, directly issue to **ROB**
8. csrrw needs to listen to rs1 value if rs1 is not ready during issue. Although there is no computation needed for csrrw, but **ROB** cannot listen to rs1 value, so csrrw needs to be issued to **ALU RS** in addition to **ROB**, with op2 set to 0 and ready, ALUSel set to 'add'
9. When the top entry of **Branch RS** is ready, it can update the **ROB** entry and wait for its turn to commit. But in this setting, if branch prediction is wrong, then a lot of meaningless instructions will be fetched from **IMEM**. A more efficient way is to clear the **Issue Queue**, update **PC** and **Branch Predictor** immediately, and stall issue until the corresponding ROB entry commits. In such way **Issue Queue** can continue fetching instructions to hide instruction fetch latency as much as possible.
10. **Issue Block** and **IMEM** are both synchronous. **IMEM** can output valid instruction and start a new read transaction in the same cycle. Thus it would be more difficult for **Issue Block** to control the **IMEM** read. Because if we use the signal that indicates whether the **Issue Block** is full to control **IMEM** read, then there is a chance that one instruction fetched by IMEM but is not accepted by the **Issue Block**. So we decide to use a signal called to_be_full to control **IMEM** read. Even if there is an additional instruction fetched in the cycle that to_be_full is about to be high, **Issue Block** is still able to catch the instruction.
11. **Issue Block** should check whether the instruction received is jal/jalr, if so should pause instruction fetch until Branch Unit update PC value
12. **Issue Block** should pause issue when **Branch Unit** indicates a misprediction, but continue fetching instructions from updated PC value. In case two caches (icache and dcache) are used but only one read port for external memory is available, this can let dcache occupy the external memory alone to let all instructions up to the latest jal/jalr commit faster.
13. **Issue Block** should resume issue when the mispredicted instruction commits from **ROB**
14. jal/jalr should be treated as misprediction
15. We use FSM to control **Issue Block**'s fetch and issue actions. Keep in mind Moore FSM would delay action by 1 cycle
16. jump_fire from **Fetch Logic** is used in **Branch Unit** to assert its output_valid when a misprediction happens. In this case, jump_fire means new PC address is read by **IMEM** and **PC.** Thus **Branch RS** can accept the result and clear all following entries and try to write the result to **CDB**.
17. In **Reset Control**, PC_reset should be delayed by one cycle using register. This can allow all table/FIFO to be reset first, such that IMEM can take the default PC value one cycle after reset
18. commit_mispred will reset **DMEM cache** but not **DMEM** main memory
19. **ROB** only needs opcode_5bit and rd_index, simplify **ROB** FIFO
20. **Branch RS** can use 3-bit as inst type. For branch instructions 3-bit can be funct3 which occupies 6 available 3-bit number. So jal and jalr can occupy the rest 2 3-bit number, namely 3'b010, 3'b011. Let jal use 3'b010, jalr use 3'b011.
21. **Branch Predictor**'s history table also stores the branch address from the latest call result of the branch instruction. This can remove the need for calculating the branch address during fetch stage, which was a big part of the critical path.
22. Modified **icache (IMEM)** can output two instructions at most. The condition is whether the difference between the second address and the first address is only in the range of 1st - 4th digit of the address (assuming address is 32-bit). If this condition doesn't hold then only the first instruction will be marked as valid. This is good enough to fetch two adjacent instructions in the same cycle if there is no branch. Even if the two instructions are not boundary aligned, it only takes one cycle to output 1 instruction, and then output 2 instructions in all of the following cycles without branch.