Rewritten Requirements

Table 1 of rewritten Requirements:

Requirement	Signals	Description
Req.A1	data_out	data_out can change only to result of OPERATION if posedge of clk signal occurs and acc_ce is HIGH.
Req.A2	data_out	data_out must operate on signed integer values between -128 to 127.
Req.A3	data_out	data_out must be reset when rst is LOW. After reset data_out must be equal to 0.
Req.A4	data_out	data_out must hold value written into it.
Req.B	cy	Must be set HIGH if OPERATION being ADD or SUB resulted in value higher/lower than 127/-128. Otherwise, set LOW.
Req.C	acc_ce	acc_ce must overwrite data_out to the result of OPERATION if posedge of clk occurs. Must be LOW at OPERATION: ST, NOP, *default*. Must be HIGH before posedge of clk for write to take place.
Req.D1-Req.D9	opcode	setting proper value results in execution of OPERATION that affects data_out output predictably (see the other table).
Req.E	clk	must receive a clock signal as input. All data_out changes are triggered on the posedge of this signal.
Req.F	rst	Must receive LOW signal to reset. Must receive HIGH signal to allow for DUT operation.
Req.G	data_in	data_in must operate on signed integer values between -128 to 127.

Table 2 of rewritten Requirements:

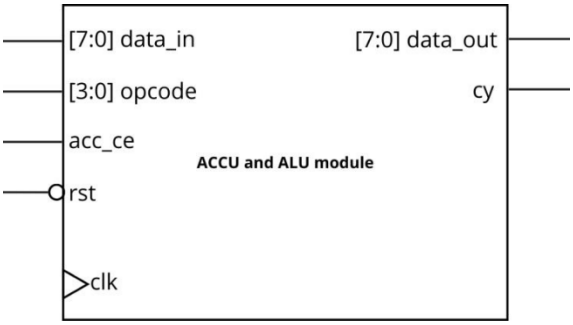
Requirement	Mnemonic/Opcode	Description / Behavior	
		accu_ce is HIGH	*accu_ce is LOW*
Req.D1	LD = 4'b0000 = 0	Must load data_in to data_out.	Must hold previous data_out value.
Req.D2	ST = 4'b0001 = 1	Must set data_out to value 0.	
Req.D3	ADD = 4'b0010 = 2	Must add data_in to data_out, display result at data_out.	
Req.D4	SUB = 4'b0011 = 3	Must subtract data_in from data_out, display result at data_out.	
Req.D5	AND = 4'b0100 = 4	Must perform bitwise AND operation on data_in and data_out, display result at data_out.	
Req.D6	OR = 4'b0101 = 5	Must perform bitwise OR operation on data_in and data_out, display result at data_out.	
Req.D7	XOR = 4'b0110 = 6	Must perform bitwise XOR operation on data_in and data_out, display result at data_out.	
Req.D8	NOT = 4'b0111 = 7	Must perform bitwise NOT operation on data_out, display result at data_out.	
Req.D9	NOP =4'b1111 = F	Must set data_out to value 0.	
Req.D10	*default*	Must set data_out to value 0.	

Brief description of tests:

Test ID	Description
test1	Checks ADD operation and overflow case of DUT.
test2	Checks SUB operation and underflow case of DUT.
test3	Checks LD, ST, NOP, *default* operations of DUT.
test4	Checks AND, OR, XOR, NOT operations of DUT.
pre test	Prepares DUT and Layered TB for verification process.
post test	Checks results.

Requirement traceability matrix and DUT's block diagram:

Requirement	test number			
	1	2	3	4
Req.A1	✓	✓	✓	✓
Req.A2	✓	✓	✓	✓
Req.A3	✓	✓	✓	✓
Req.A4	✓	✓	✓	✓
Req.B	✓	✓		
Req.C	✓	✓	✓	✓
Req.D1			✓	
Req.D2			✓	
Req.D3	✓			
Req.D4		✓		
Req.D5				✓
Req.D6				✓
Req.D7				✓
Req.D8				✓
Req.D9			✓	
Req.D10	✓	✓	✓	✓
Req.E	✓	✓	✓	✓
Req.F	✓	✓	✓	✓
Req.G	✓	✓	✓	✓



Test plan:

Pre test – "Prepares DUT and Layered TB for verification process."

1. Set rst LOW.
2. After 7 time units, set rst HIGH.
3. Check whether tests' transactions where created and loaded into mailbox by generator. Check Monitor, Driver values after first transaction.

Test1 – "Checks ADD operation and overflow case of DUT."

1. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce HIGH, opcode to mnemonic ADD value. Data_in should have random values within operating range.
2. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce LOW, opcode to mnemonic ADD value. Data_in should have random values within operating range.
3. Check scoreboard values.

Test2 – *"*Checks SUB operation and overflow case of DUT.*"*

1. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce HIGH, opcode to mnemonic SUB value. Data_in should have random values within operating range.
2. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce LOW, opcode to mnemonic SUB value. Data_in should have random values within operating range.
3. Check scoreboard values.

Test3 – *"*Checks LD, ST, NOP, default operations of DUT.*"*

1. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce HIGH, opcode to mnemonic AND value. Data_in should have random values within operating range.
2. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce LOW, opcode to mnemonic AND value. Data_in should have random values within operating range.
3. Repeat steps 1 and 2 for opcode values NOP and NOT;
4. Check scoreboard values.

Test4 – *"*Checks AND, OR, XOR, NOT operations of DUT."

1. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce HIGH, opcode to mnemonic LD value. Data_in should have random values within operating range.
2. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce LOW, opcode to mnemonic LD value. Data_in should have random values within operating range.
3. Repeat steps 1 and 2 for opcode values: ST, NOP.
4. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce HIGH, opcode to random value from undefined range. Data_in should have random values within operating range.
5. Set and run a sequence for 50 repetitions. Set sequence's: acc_ce LOW, opcode to random value from undefined range. Data_in should have random values within operating range.
6. Check scoreboard values.

Post test – *"*Checks results."

1. Check whether all tests have been executed.
2. Check success to failure ration of scoreboard.
3. Note the difference between DUT and scoreboard's model.

Verification results:

1. DUT had an error: the reg data type was used outside procedural block.
2. DUT had no negation on input rst.
3. DUT works as described in requirements – no further errors where found.