Group R

Project Milestone 2:

Simulation of RISC CPU using Verilog Hardware Design

Part B:

We decided to add the functionality of the AND instruction to the RISC CPU.

Part C:

AND RD, RS2, RS1: The logical operations on the values in registers rs1 and rs2, then write the result to register rd.

0000000	Rs2	Rs1	111	Rd	0110011	
func 7	rs2	rs1	Func 3	rd	opcode	R-Type

The CPU each time starting up will be reset to the predefined initial state. When after resetting, the first clock triggers the operation of Stage-1(Fetch Instruction stage). In stage-I control set the load_inst from 0 to 1, which enables the reading of the read only memory that stores your instruction (inst_ROM). Then inst_ROM put the value at the address indexed by the current PC value to its output port.

Second clock edge comes, control set the instruction decoder which triggers the execution of the inst_decoder. After decoding done ALU will know which operation has to be done. Finally the result of the AND function will be written into the register designated by the register rd.

ALU.v

The ALU had to be modified in order to accommodate the new AND parameter that was added. The number of bits for the parameters and the command input needed to be increased from 5 bits to 6 bits in size, so that 6'b100000 could be used to call the AND operation.

Before	After
5 initial parameters	Changed to 6 parameters, to include “parameter AND=6'b100000;”
5 bit parameter size	Adjusted to 6 bit parameters, in order to accommodate the new AND parameter
input[4:0] command	input[5:0] command, to allow the command 6'b100000 to be used

control.v

The number of bits for the existing parameters was increased from 5 bits to 6 bits and the 11 bit parameters were increased to 12 bits. An additional ALUAND parameter was added as a 6 bit parameter and an AND parameter was added as a 12 bit parameter. The bit sizes need to be increased in order to accommodate these two new parameters. AND functionality was added in order to set the ALUcommand to ALUAND if AND was the command to be executed.

Before	After
5 initial parameters	Changed to 6 parameters, to include "parameter ALUAND=6'b100000;"
5 bit parameter size	Adjusted to 6 bit parameters, in order to accommodate the new ALUAND parameter
11 bit parameters	Changed to 12 bit parameters, additional "parameter AND=12'b100000000000;" was added
input[10:0] execution	input[11:0] execution, so "parameter AND=12'b100000000000;" can be executed
input[4:0] ALUcommand	input[5:0] ALUcommand, to allow the command 6'b100000 to be used
reg[4:0] ALUcommand_r	reg[5:0] ALUcommand_r - Size increase, from 5 bits to 6 bits
"if (rst ==1)" section: ALUcommand_r<=5'b00000	"if (rst ==1)" section: ALUcommand_r<=6'b000000 - Size increase, from 5 bits to 6 bits

	Added in the "control:" section:
	AND:
	begin
	ALUsrc<=1'b0;
	ALUcommand_r<=ALUAND;
	state<=executing;
	end

“executing:” section: - Other operations excluding AND	“executing:” section: - AND added to this section - writes the result to the register file
“change_pc:” section - Other instructions excluding AND	“change_pc:” section - AND added to this section
“default:” section ALUcommand_r<=5'b00000;	“default:” section ALUcommand_r<=6'b000000; - Size increase, from 5 bits to 6 bits

CPU_top.v

The bit sizes need to be increased in order to accommodate the AND functionality. Since there is now one additional command the bit size needs to be incremented by 1 in order for the AND functionality to be called and executed.

Before	After
wire[10:0] execution	wire[11:0] execution - Size increase, from 11 bits to 12 bits
wire[4:0] ALUcommand	wire[5:0] ALUcommand - Size increase, from 5 bits to 6 bits

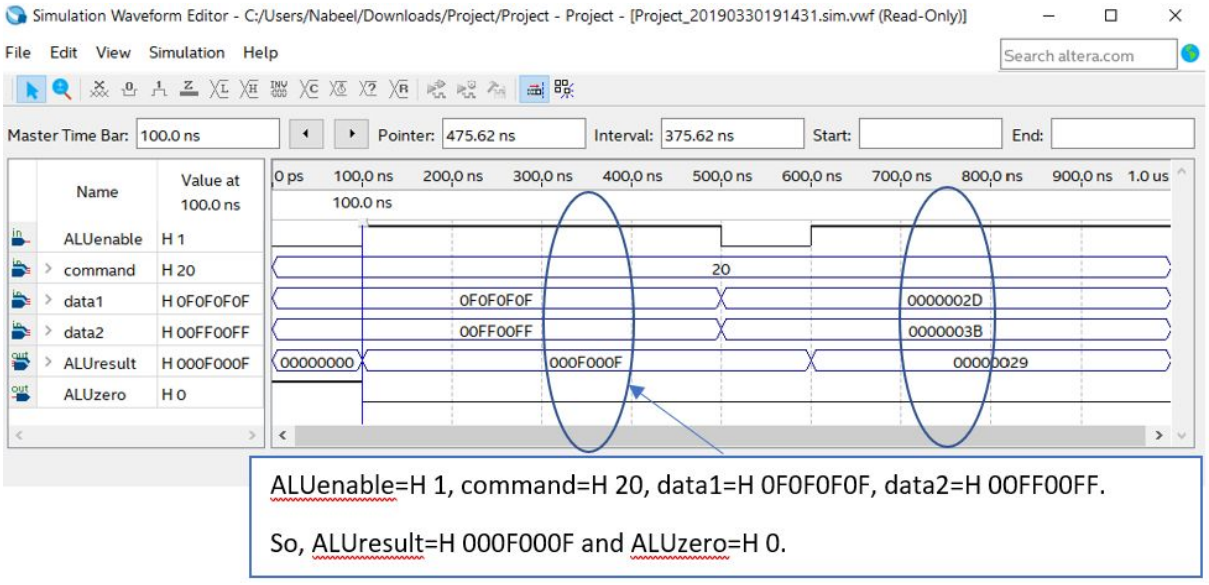
inst_decoder.v

Since AND is an R-Type instruction, the if statement below was added to the instruction decoder module. This if statements decodes the AND command and assigns the AND parameter to execution_r. The bit sizes also need to be increased here so that the AND command could be accessed and assigned.

Before	After
output[10:0] execution	output[11:0] execution - Size increase, from 11 bits to 12 bits
11, 11-bit parameters	Changed to 12, 12-bit parameters, in order to accommodate the new “parameter AND=12'b100000000000;”

R-Type instructions:
- Other instructions (ADD, SUB, SLL, XOR, OR) not including AND
	- Added AND functionality as one of the R-type instructions. As follows:
	if ((inst[14:12]==3'b111)&&(inst[31:25]==7'b0000000))
	begin
	execution_r<=AND;
	rr1_r<=inst[19:15];
	rr2_r<=inst[24:20];
	wr_r<=inst[11:7];
	end
	- Note: inst[14:12]==3'b111, is set to 3'b111 since this is the funct3 for AND

Part D:

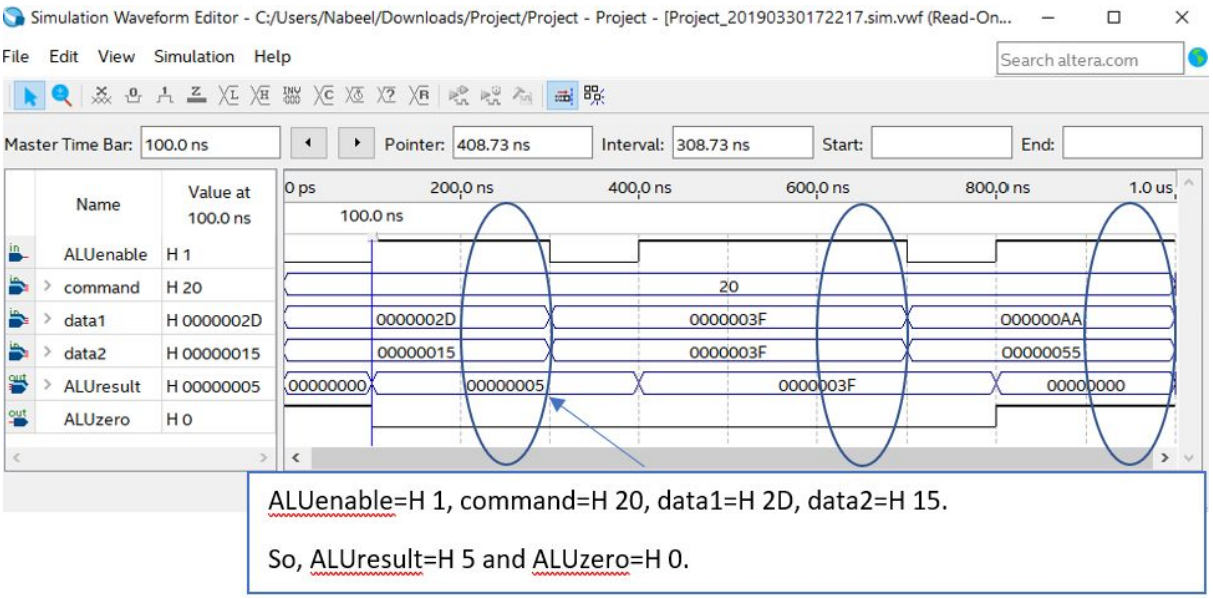


Input/Output	Hexadecimal	Binary
ALUenable (input)	0 1	0 1
command (input)	20	100000
data1 (input)	0F0F0F0F 2D	11110000111100001111000011111 1

		101101
data2 (input)	00FF00FF 3B	11111111000000011111111 111011
ALUresult (output)	000F000F 29	11110000000000001111 101001
ALUzero (output)	1 0	1 0

When the ALUenable is set to 1 and the AND command is called (by setting the command to hexadecimal 20 or 100000 in binary), we can see the ALU performs the AND operation in the waveform above.

First, the inputs data1 and data2 were initialized to 0F0F0F0F and 00FF00FF respectively. An AND operation on these values should have an expected output of 000F000F. As seen in the waveform above ALUresult does in fact contain 000F000F, after the AND operation takes place. There are a few addition test cases to ensure that the AND operation is working correctly. The additional test cases are (the following values are all in hexadecimal format) 2D & 3B = 29, 2D & 15 = 5, 3F & 3F = 3F, AA & 55 = 0. All of these additional test cases passed since these are the expected values for the AND operation. From these test cases, we can see that the AND operation has been correctly implemented in the modules.



Input/Output	Hexadecimal	Binary
ALUenable (input)	0 1	0 1
command (input)	20	100000
data1 (input)	2D 3F AA	101101 111111 10101010
data2 (input)	15 3F 55	10101 111111 1010101
ALUresult (output)	5 3F 0	101 111111 0
ALUzero (output)	1 0	1 0

Part E: Conclusion

In general, this project has been teaching us how to apply the course material practically. This comes in the form of designing circuits, logical systems, modules, and the cpu using the Verilog programming language and Quartus Simulation.

The goal of Milestone 2 was to change some of the modules of the CPU to introduce a new instruction of our choosing; in this case the AND instruction. The exercise required that we recognize which modules will be affected by this instruction and make the appropriate changes.

We also had to learn how to make waveforms in order to test our modules and verify that the implementation of our AND instruction did in fact produce the correct output.