李驊祐 資工三 B10902078

Modules Explanation

Adder.v:
This module reads two 32-bits input *input_data_1* and *input_data_2*, and returns a 32-bits output *output_data*, which is equivalent to *input_data_1* + *input_data_2*.

Control.v:
This module reads a 7-bits input *opcode*, and has a 2-bits output *ALUOp*, 1-bit output *ALUSrc*, 1-bit output *RegWrite*.
If *opcode* == 0110011, then I assign *ALUOp* = 00 so that the ALU_Control module knows it is an R-type instruction, and assign *ALUSrc* = 0 so that the MUX32 module knows it should choose input data from Register.
Else, then I assign *ALUOp* = 01 so that the ALU_Control module knows it is an I-type instruction, and assign *ALUSrc* = 1 so that the MUX32 module knows it should choose input data from Sign_Extend.
Then, I assign *RegWrite* = 1, because all operations write to register.

ALU_Control:
This module reads a 7-bits input *funct7*, a 3-bits input *funct3*, a 2-bits input *ALUOP*, and has a 3-bits output *ALUControl*. Then, I assign *ALUControl* from 000 to 111 corresponding to instruction *and* to *srai*. The conditions I used are:
If *ALUOp* == 00: then check *funct7* and *funct3* to assign values 000 to 101 corresponding to instruction *and* to *mul*.
Else if *ALUOp == 01*: then check *funct3* to assign values 110 to 111 corresponding to instruction *addi* and *srai*.

ALU:
This module reads two 32-bits inputs *read_data_1* and *read_data_2*, a 3-bits input *ALUControl*, and has a 1-bit output *zero* and a 32-bits output *ALU_result*. Then, I do the corresponding arithmetic operation on the two 32-bits inputs according to the condition *ALUControl*, and assign the result to *ALU_result*. I also assign *zero* = 1 if *read_data_1* - *read_data_2* == 0, else 0.

MUX32:
This module reads two 32-bits input *input_data_1* and *input_data_2*, a 1-bit input *mux_select*, and has a 32-bits output *output_data*. If *mux_select* == 0, then I assign *output_data* = *input_data_1*, else *input_data_2*.

Sign_Extend:
This module reads a 12-bits input *instruction20to31*, and has a 32-bits output *SignExtend*. I assign *SignExtend[11:0]* = *instruction20to31*, and extend the 12th bit of *instruction20to31* (*instruction20to31[11]*) to fill in the remaining 20 bits of *SignExtend* (*SignExtend[31:12]*).

CPU:
In this module, I just use structure modeling to connect the input wires and output wires of the above modules following the datapath in Figure 1.

Development Environment: I use ubuntu (Ubuntu 22.04.1 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64)) and iverilog to run my code. To run the test case, I just run make.