李驊祐 資工三 B10902078

Modules Explanation

Adder.v: Read two 32-bits input data, and output the sum of them

ALU_Control.v: It has funct7_in, funct3_in, ALUOp_in as input and ALUControl_out as output. I assign ALUControl_out from 000 to 111 based on the three input signals. (Just like lab1).

ALU_v: It has two 32-bits input data (read_data_1_in and read_data_2_in), an ALUControl_in input signal, and zero_out and 32-bits ALU_result_out as output. I assign ALU_result_out based on the value of ALUControl_in and perform its respective operations. Zero_out is disregarded in this part. (Just like lab 1).

Branch_Unit.v: It has two 32-bits input data (read_data_1_in and read_data_2_in), and 32-bits imme32_in, ID_pc_in, and a 1-bit Branch_in as input. It has a 32-bits output next_pc_out (indicating the next address pc should jump to), and a 1-bit output Flush_out.
I assign next_pc_out as ID_pc_in + (imme32_in << 1). In the normal case, Flush_out is 0, however, when read_data_1_in is equal to read_data_2_in and Branch_in is also true, then Flush_out should be 1.

Control.v: It has an input signal NoOp_in and a 7-bits input opcode_in. It has RegWrite_out, MemtoReg_out, MemRead_out, MemWrite_out, a 2-bits ALUOP_out, ALUSrc_out, and Branch_out has output.

For RegWrite_out, if NoOp_in is set to 1, then RegWrite_out should be 0. Otherwise, R-type, I-type, and lw should set RegWrite_out to 1, and sw and beq should set RegWrite_out to 0.

For MemtoReg_out, if NoOp_in is set to 1, then MemtoReg_out should be 0. Otherwise, only lw should set MemtoReg_out to 1.

For MemRead_out, if NoOp_in is set to 1, then MemRead_out should be 0. Otherwise, only sw should set MemWrite_out to 1.

For ALUOp_out, if it is a I-type, lw, or sw instruction, then ALUOp_out should be 00. If it is an R-type instruction, then ALUOp_out should be 10. If it is a branch instruction, then ALUOp_out should be 01.

For ALUSrc_out, if it is a I-type, lw, or sw instruction, then ALUSrc_out should be 1, because it should choose the output from Sign_Extend. If it is a R-type or branch instruction, then ALUSrc_out should be 0, because it should choose from the output of Forward_B_MUX.

Finally, For Branch_out, if NoOp_in is set to 1, then Branch_out should be 0. Otherwise, only the branch instruction should set Branch_out to 1.

Two_Mux32.v: It has 2 32-bits input and a 1-bit select input, and assigns the output based on the select input.

Four_Mux32.v: Same with Two_Mux32 but with 4 32-bits input and a 2-bit select input.

Sign_Extend.v: Extend the 12-bits input to a signed 32-bits output.

IF_ID_Register.v: It has reset, clock, Flush_in, Stall_in, and a 32-bits instruction_in and a 32-bits pc_in as input. It has a 32-bits instruction_out and 32-bits pc_out as output. First, I will check the reset signal. If it is on, then I will set instruction and pc to 0. Otherwise, if Flush_in is also true, then I will set instruction and pc to 0. If stall is 0, then I load the proper values for instruction_out and pc_out.

ID_EX_Register.v: There are a lot of inputs for this module, including reset, clock, RegWrite, MemtoReg, MemRead, MemWrite, ALUOp, ALUSrc, read_data_1_in, read_data_2_in, imme_in, inst31_25_in, inst14_12_in, inst19_15_in, inst24_20_in, inst11_7_in. There are the corresponding output wires with the same name, except for reset and clock. The logic is when reset is on, I set those output wires to 0, otherwise, I set those output wires to the values of the input wires.

EX_MEM_Register.v: There are also a lot of inputs for this module, including reset, clock, RegWrite, MemtoReg, MemRead, MemWrite, ALU_result, MUX, inst11_7. There are the corresponding output wires with the same name, except for reset and clock. The logic is the same with the above. When reset is on, I set those output wires to 0, otherwise, I set those output wires to the values of the input wires.

MEM_WB_Register.v: The logic is the same with the above pipeline register. There are lots of inputs, including reset, clock RegWrite, MemtoReg, ALU_result, Read_data, inst11_7. There are the corresponding output wires with the same name, except for reset and clock. When reset is on, I set those output wires to 0, otherwise, I set those output wires to the values of the input wires.

Forwarding_Unit.v: It has MEM_RegWrite_in, Mem_Rd_in, WB_RegWrite_in, WB_Rd_in, EX_Rs1_in, EX_Rs2_in as input, and has Forward_A_out and Forward_B_out has output. For this part, I just implement the If condition provided in the slides. In the normal case, Forward_A_out and Forward_B_out is 00. However, when the If conditions are true, it will be set to 10 or 01.

Hazard_Detection_Unit.v: It has reset, inst19_15_in, inst24_20_in, EX_MemRead_in, EX_inst11_7_in as input, and has NoOp_out, PCWrite_out, and Stall_out as output. Since the output signals are needed at first, they need to be initialized to 0 at the beginning. Hence, if reset is on, I will first initialize the NoOp and Stall output to be 0, and PCWrite output to be 1. In the normal case, these 3 output signals should maintain their value. However, when it is a load stall, the CPU needs to be stalled. Hence, when EX_MemRead_in is on (indicating a load instruction) and EX_inst11_7_in is equal to inst19_15_in or EX_inst11_7_in is equal to inst24_20_in, NoOp, PCWrite, Stall will be set to 1, 0, 1.

CPU.v: In this module, I just connect all the wires together following the diagram provided in the spec. The only thing to be cautious about is the input of Sign_Extend, because the Immediate value will vary according to different types of instructions.

Difficulties:

The hardest part of this lab for me was debugging. There are literally thousands of wires, all with similar names, so it is extremely difficult to trace the bug. I spent 4 hours trying to debug my code, and I eventually found out that the mistake was because I misused lowercase and uppercase in the naming of one wire, and I put 1 bracket in the wrong place.

Environment:

Ubuntu, Ubuntu 22.04.1 LTS (GNU/Linux 5.10.16.3-microsoft-standard-WSL2 x86_64).