Modules

Besides modules I implemented in hw4, I add new modules below:

IFID:

IFID has six inputs, clk_i, start_i, Stall_i, Flush_i, pc_i and instr_i and two outputs pc_o and instr_o.
When sensing every positive clk_i and start_i, if not Stall_i it will update pc_o to pc_i, instr_o to instr_i. But if the Flush_i bit is on, it will set both pc_o and instr_o to 0.

IDEX:

IDEX has a lot of inputs and output, except for clk_i and start_i, there are ALUOp_i, ALUSrc_i, RegWrite_i, MemtoReg_i, MemRead_i, MemWrite_i, RS1addr_i, RS2addr_i, RS1data_i, RS2data_i, funct_i, imm32_i, RDaddr_i and their corresponding output.
IDEX will update output registers to their corresponding input at every positive clock edge if start_i is set.

EXMEM:

EXMEM acts just like IDEX, but having a different set of input/output, that is, ALUres, RegWrite, MemtoReg, MemRead, MemWrite, RS2data and RDaddr.
EXMEM will update output registers to their corresponding input at every positive clock edge if start_i is set.

MEMWB:

MEMWB acts just like IDEX and EXMEM, but having a different set of input/output, that is, ALUres, RegWrite, MemtoReg, Memdata and RDaddr.
MEMWB will update output registers to their corresponding input at every positive clock edge if start_i is set.

Forwarding_Unit:

Forwarding_Unit has six input, EX_RS1addr_i, EX_RS2addr_i, MEM_RegWrite_i, MEM_RDaddr_i, WB_RegWrite_i, WB_RDaddr_i and two outputs, ForwardA_o, ForwardB_o.
I use four if-statements to decide if there is EX hazard on rs1, EX hazard on rs2, MEM hazard on rs1, MEM hazard on rs2, using the formula given in the slide. And if there are some hazards, it will set ForwardA_o and ForwardB_o to 10 or 01 which indicates EX hazard and MEM hazard respectively.

MuxW MUX_ForwarA MUX_ForwarB:

MuxW is the wide version of Mux. It takes four 32-bit data and two select-bits as input, and outputs a 32-bit result. If the select-bits are 00, then the output data1, 01 then data2, 10 then data3, and 11 then data4.

I have two MuxW modules which handle forwarding two inputs of ALU. If the select-bits are 00, then output data from rs. If 01, output write data from MEMWB for solving MEM hazard. If 10, output ALU results from EXMEM for solving EX hazard. And their select-bits come from the output of Forwarding_Unit.

Hazard_Detection:
Hazard_Detection takes four inputs RS1addr_i, RS2addr_i, EX_MemRead_i, EX_RDaddr_i, and three outputs NoOp_o, Stall_o, PCWrite_o.
It will check if there is a hazard caused by load instructions by the formula in the slide. If there is, set NoOp_o to 1, Stall_o to 1 and PCWrite_o to 0, else, set NoOp_o to 0, Stall_o to 0 and PCWrite_o to 1.

Branch_Unit:
Branch_Unit takes five inputs, RS1data_i, RS2data_i, Branch_i, ID_pc_i, imm32_i and two outputs Flush_o, jump_addr_o.

It will first check if there is a branch instruction and the values of two rs are the same. If the condition is true, it means the program has to jump, and it will set Flush_o to 1, jump_addr_o to ID_pc_i + (imm32_i << 1).

Mux Mux_PC:
I also added a new mux before updating the PC, having two inputs, the first one is the original PC+4 and the other one is the jump address that comes from Branch_Unit. The select bit is Flush from Branh_Unit as well, therefore if Flush is set, the new PC will be updated to jump_addr.

CPU:
Finally, the CPU connects these components together as the final data path given in the spec.

Difficulty
The first difficulty is that debugging is quite hard when doing the lab. Because there are so many wires, tracing the source of the bug is not easy. A bug I have encountered is that the pipeline registers are set to x in the beginning although I have initialized them in testbench.v. My solution is to pass the start signal to pipeline registers, and update them only when the process is started.

The second difficulty is there are no branch instructions in the test data, so it is really difficult to test if our program is correct. And generating test data by ourselves is quite complex. It took me many hours just to make a simple judge system which can generate valid inputs.

And the last thing I want to mention is that the spec is not written clearly enough. Like the test data given, they have already initialized their registers and data memory to different values, and the spec says don’t change the register's initial value, so it’s really confusing to us. Also, the data memory is different too, the given data memory’s register is unsigned, therefore our output can’t display negative numbers correctly, we have to spend a lot of time to check if our program is wrong and finally find out it’s not our problem. And the main problem is, there is no information about those in any slides, and neither in any announcement. I suggest that you should have a discussion section on NTU cool, helping students to solve some common problems efficiently, instead of poor communication such as this.

Development Environment:
macOS Big Sur 11.4