Lab2 report

B10902105 何珮瑄

Modules

Besides modules implemented in lab1, I add new modules below:

ALU_Control

Besides the origin `ALUCtrl_i` s, I added two more determine for lw\sw and beq.

Branch Unit

There are five inputs `RS1data_i` `RS2data_i` `Branch_i` `ID_pc_i` `imm32_i` and two outputs `Flush_o` `jump_addr_o`.

if two register sources (`RS1data_i` and `RS2data_i`) are identical, a program jump is required, it sets `Flush_o` to 1. Additionally, it calculates and assigns the jump address (`jump_addr_o`) as the sum of the current program counter (`ID_pc_i`) and the left-shifted immediate value (`imm32_i` by 1).

Control

Besides the origin control signal, I added `NoOp_i` `MemtoReg_o` `MemRead_o` `MemWrite_o` `Branch_o`.

IFID

IFID has inputs `clk_i` `rst_i` `Stall_i` `Flush_i` `pc_i` `instr_i` and two outputs `pc_o` and `instr_o`.
When sensing every positive `clk_i` and `rst_i`, if not `Stall_i` it will update `pc_o` to `pc_i` and `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

Except for `clk_i` and `rst_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 corresponding output.

IDEX will update output registers to their corresponding input at every positive clock edge.

EXMEM

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

MEMWB

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

Forwarding Unit

Forwarding Unit has 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`. As written in the spec, there are four if-statements to decide if there is EX hazard on rs1\rs2 or MEM hazard on rs1\rs2 . If there are hazards, `ForwardA_o` and `ForwardB_o` will be to 10 or 01, indicating EX hazard and MEM hazard respectively.

Hazard Detection

There are four inputs `RS1addr_i` `RS2addr_i` `EX_MemRead_i` `EX_RDaddr_i` and three outputs `NoOp_o` `Stall_o` `PCWrite_o`.

As written in the slides, the detection unit will check whether there is a hazard caused by load instructions. If there is one, 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.

MUX32W

For the multiplexer with 3 inputs, I added a new module to implement. With inputs `data1_i` `data2_i` `data3_i` `select_i` and output `data_o`.

MUX_PC

I also added a new mux before updating the PC, with two inputs, the first one is the original PC+4 and the other one is `jump_addr_o` that comes from Branch_Unit. The select bit is `Flush` from Branch_Unit as well, therefore if `Flush` is set, the new PC will be updated to `jump_addr_o`.

CPU

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

Difficulty

Debugging during the lab proves challenging primarily due to the setup involving numerous wires, making it difficult to pinpoint the source of bugs. To make sure what value does each wire carry, I printed all out and manually checked for every cycle.

The other trouble I face is the reset signal. Initially I only assign `ALUOp_o <= ALUOp_i` in the pipeline registers, however the PC was never working since the first cycle. Not until I asked for help that I knew that these pipeline registers have to be set to 0 at `~rst_i`.

Development Environment:

macOS & iverilog