CompArch Lab 2 Report

Guan Yu Cho

B04901126

1 Modules

Branch Predictor

A 2-bit dynamic branch predictor with initial state `S0` (strongly taken). Its output signal `predict_o` would select the source of PC between `target PC` or `PC + 4`. Plus, `predict_o` would be used as flush signal to `IF_ID` register and stored in `ID_EX` for later usage.

`S0`	`2'b00`	predict taken
`S1`	`2'b01`	predict taken
`S2`	`2'b10`	predict not taken
`S3`	`2'b11`	predict not taken

Pipeline Registers

We use 4 pipeline registers to keep the output data for the later pipeline stages. In general we keep store the control signal and arithmetic results within these registers, and update them at the posedge of clock.

Multiplexers

(a) **MUX2to1.v**

`select_i` selects one source to output out of the 2 input ports.

(b) **MUX4to1.v**

`select_i` selects one source to output out of the 4 input ports.

Control Units

(a) **Control.v**

`Control.v` takes in three inputs: `rst_i`, `noop` and `opcode`. `noop` signal comes from the hazard detection unit. When `noop == 1`, Control would set `MemWrite_o` and `RegWrite_o` to be 0, to prevent the stalled instruction from updating any memory and register data. When `noop != 0`, it would decide the control signal such as `ALUOp_o` and `ALUSrc_o` for the later use purely based on the last three bits of the opcode. Notice that when `opcode == 7'b0`, we set all the control signal to be zero to handle the flushed instruction.

(b) **ALU_Control.v**

With function code `func7 | func3` and operation type `ALUOp_i` from `Control.v`, `ALU_Control.v` is able to decide which arithmetic operation ALU has to perform.

(c) **HDU.v**

HDU stands for Hazard Detection Unit. It detects load and branch data hazards, and decides to stall or flush the pipeline. When the instruction in EX stage performs `MemRead` and its target register is the same as any the source register of the instruction at the ID stage, `FU.v` would set stall to 1, to stall the whole pipeline by preventing any update in PC and `IF_ID` register for one cycle. What's worth mentioning is that when the previous cycle was stalled (`noop == 1 && stall == 1 && PCWrite == 0`), HDU would flip these signal back to `noop = 1'b0; stall = 1'b0; PCWrite = 1'b1;`.

(d) **FU.v**

FU stands for Forwarding Unit. It detects if there's any dependencies between instructions in EX, MEM and WB stages by exaimining the their source and target register addresses. If EX or MEM hazard occurs, ForwardA and ForwardB would change accordingly to forward the right source data. By default, ForwardA and ForwardB would be `2'b00`.

(e) **BCU.v**

BCU stands for "Branch Correction Unit". The purpose of this unit is to determine if the previous prediction for branching is wrong. If it is wrong, then it sends `rollback` signal to `IF_ID` register and `ID_EX` register.

Arithmetic components

(a) **Adder.v**

When any of the two input 32-bit data changes, the output would be re-evalutated by `data1_i + data2_i`. This module is used on the generation of PC address.

(b) **ALU.v**

Controlled by `ALUCtrl_i`, ALU does the corresponding operation on the source data.

ImmGen.v

The whole 32 instruction bits are taken in, and the immediates are generated accroding to the `<func3 | opcode>`. The reason to do this is that the immediates are encoded in different ways in different types of instruction.

CPU.v

The place where all the modules are wired together.

testbench.v

As is provided by TAs.

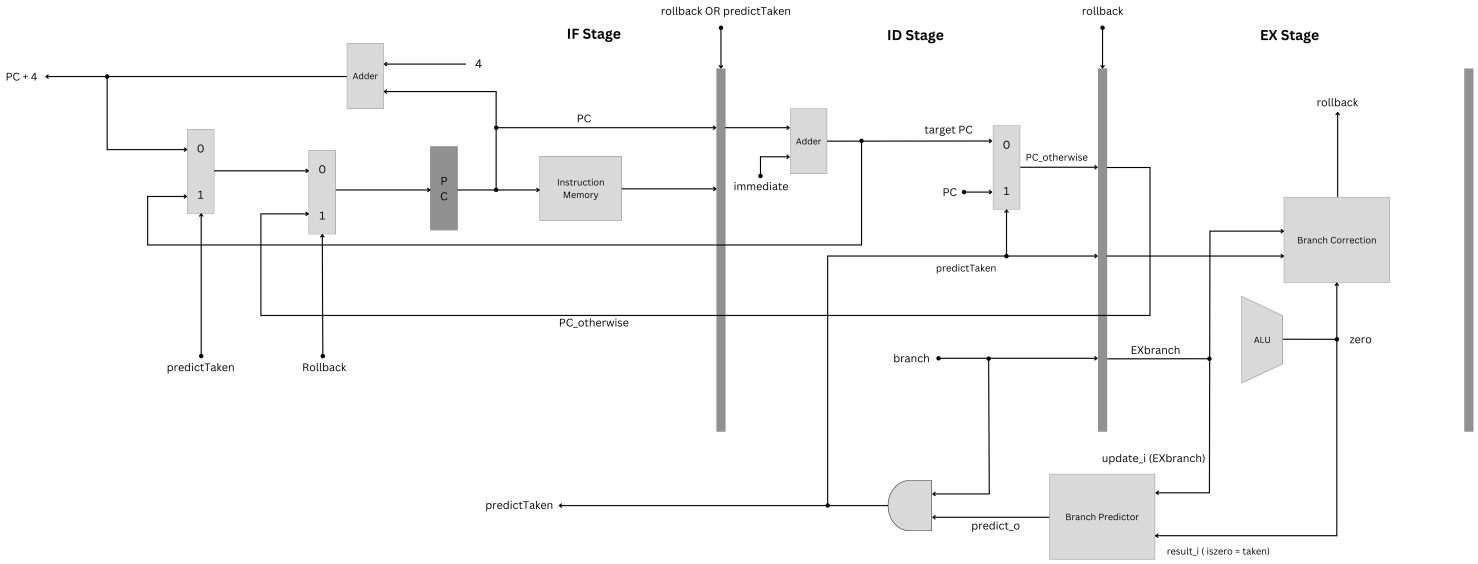

Figure 1: Architectural extension for lab2

2 Difficulty and Solutions

(a) Spent time mostly on coming up with the design. Draw the design before implementing can save up lots of time.

(b) I don't know how to print `ALU_o` as singed integer by modifying module, so I just add `$singed` in `$fdisplay`.

3 Dev Environment

MacOs + iverilog + vvp + gtkwave