CA Lab 2

R11922211 葉小漓

1. Modules Explanation

- **Control and ALU_Control**
  Compared to lab 1, I add MemtoReg, MemRead and MemWrite control signals, and control the signal outputs according to the following truth table:

Instruction	ALUSrc	MemtoReg	RegWrite	MemRead	MemWrite	Branch	ALUOp1	ALUOp0
R-type	0	0	1	0	0	0	1	0
I-type	0	0	1	0	0	0	0	0
load	1	1	1	1	0	0	0	0
store	1	x	0	0	1	0	0	0
branch	0	0	0	0	0	1	0	1

  The Control module also takes a NoOP signal, which would output all zeros if NoOP is 1.
  I also modified ALU_Control to support load/store instructions, specifically: if ALUOP & func3 indicates the instruction is an load/store, output the ALU_Control signal for addition.
- **Pipeline Registers**
  Add IF_ID, ID_EX, EX_MEM, MEM_WB registers as pipelines. Their output signals are triggered by positive edge of clock or negative edge of reset. If reset, set all outputs of pipeline to zero. If clock, set output signal to input. IF_ID register stores the instruction read from pc. ID_EX stores the control signals, rs1 data and rs2 data, the immediate and rd's address. It also takes two signals: Flush and Stall. If Stall, then the output is not updated. If Flush, then the output is flushed to all zero's. EX_MEM stores RegWrite, MemtoReg, MemRead and MemWrite from the previous stage, the ALU result, rs2 data, and rd's address. MEM_WB register stores RegWrite, MemToReg, ALU result, Memory read data, and rd's address.
- **Forwarding Unit**
  Implement a Forwarding Unit and Two 4-1 muxes. Forwarding Unit takes ID/EX.RegisterRs1, ID/EX. RegisterRs2, EX/MEM.RegWrite, EX/MEM. RegisterRd, MEM/WB.RegWrite, MEM/WB. RegisterRd as input, and ForwardA, ForwardB as output. Then the combinational logic is implemented as P.3 of spec. The two muxes are implemented as follows:
  Mux1 (output connects ALU op1)

ForwardA value	00	01	10	11
Selected Output	ID/EX.RegisterRs1	WB.WriteData	MEM.ALUResult	x

Mux2 (output connects mux for ALU op2)

ForwardB value	00	01	10	11
Selected Output	ID/EX.RegisterRs2	WB.WriteData	MEM.ALUResult	x

- **Hazard Detection Unit**
  The Hazard Detection Unit takes EX_MemRead, EX_rd_addr, ID_rs1_addr, ID_rs2_addr, and determines whether there is load use hazard. Specifically, if EX_MemRead is 1 and EX_rd_addr equals to either ID_rs1_add or ID_rs2_add, then there is a hazard, and set output signals NoOP to 1, PCWrite to 0 and Stall to 1. Else, output signals are NoOP = 0, PCWrite t= 1 and Stall = 0.
- **Testbench**
  Initialize the output ports of IF_ID, ID_EX, EX_MEM, MEM_WB registers to 0 in the initial block of testbench.v.
- **Others**
  - **Branch Controller**
    This module determines to branch or not. It ANDs the branch control signal and the truth value of (ID/EX.RegisterRs1 == ID/EX.RegisterRs2), and outputs 1 if branch, 0 if not branch.
  - **Branch Address Resolver**
    This module resolves the branch address. It takes the pc value and adds (immediate << 1) to it, which is the next pc address if there is branch.
- **CPU**
  The CPU module is constructed as follows: first, construct a single cycle CPU with data memory. Then, add the pipelines. Then, integrate the forwarding unit, and finally, integrate the hazard detection unit and branch control modules.

2. Difficulties Encountered and Solutions in this Lab

The first difficulty I encountered was that I didn't know how to use gtkwave. This is solved by experimenting with the interface. The second was I want to test my implementation of each stage (after creating single cycle CPU, after implementing pipeline… etc.) but the TA's testbench can be used for the final implementation. But I realized that you just have to remove the lines regarding flush and stall, then you can use it for the previous stages.

There was one problem that my PC could not output any value after putting a mux before it. To debug, I opened gtkwave and checked whether the MUX module's I/O signals are correct. It turned out that my selector was not functioning properly and made it stuck at one, and this is because my Control module did not handle the branch case properly. The problem was solved after updating the branch case in Control module.

A final problem was the program didn’t output correct values after adding pipeline. So, I looked into gtkwave and found the output of a port in ID_EXE stage was incorrect. After looking that CPU.v and connecting the wires correctly, the problem was solved.

3. Development Environment

- OS: Ubuntu 22.04
- Compiler: iverilog