Computer Architecture

Lab 2

R12922147 魏子翔

Modules Explanation.

Below I explain the modifications for those code from lab 1 and the main idea for the new added modules.

1. CPU

   First of all, I initialize the value for the pipeline registers to zero, then add the new modules with their I/O wire. For the same signals from different pipeline stages, I add a post-fix to represent the wires are from which stage. For example, the Mem-to-Reg signal from the output of Control module, I named it *memtoreg_1* since it's from ID stage, when it transfer to the EX stage, it changed the name to *memtoreg_2*.

2. Control

   For control module, I simply add the condition for lw, sw, beq, and no op to assign their corresponding control signals. Also, I add a signal *Branch_o* to represent the branch signal.

3. Adder, MUX32, Sign_Extend, ALU

   Totally same as lab 1.

4. ALU_Control

   Add the condition for lw and sw, then assign their *aluctr* signal same as add instruction.

5. IFID, IDEX, EXMEM, MEMWB

   Handle the pipeline operation and the data and signals transferring between the stages. Only IFID module need to deal with the control signals such as Flush and Stall, while the others are simply pass the I/O value.

6. Forwarding unit

   Deal with the data hazard at EX stage and MEM stage, pass the control signal to the 4-way mux to decide whether to forward the value or not and which data should be forwarded.

7. MUX4

   As a 4-way mux, it use the 2 bit control signal to decide which input data should be the output data.

8. Hazard Detection

   Detect the control hazard when dealing with the branch instruction, and set the *np_op*, *stall*, and *pcwrite* signal to their corresponding registers when the stall happened. I use the condition from the class note, that is,

   If (ID/EX.MemRead and ((ID/EX.RegisterRd = IF/ID.RegisterRs1) or

   (ID/EX.RegisterRd = IF/ID.RegisterRs2)))

9. Branch unit

   Deal with the conditions to decide whether the branch is occurred, and pass the data and signals. Including check whether rs1 data is equal to rs2 data, whether it is a branch instruction, and add the PC with the extended immediate times 2 to get the branch target.

Difficulties Encountered and Solutions in This Lab.

1. At first, I do not know that I need to initialize the pipeline registers, which cost me a lot of time to debug cycle by cycle. I even doubt the given supplied codes have something wrong since I'm pretty sure my wires are connected correctly.
2. Although I've already named the same signals with different post-fix, I was still confused and made lots of bugs when dealing with them. It made me so dizzy.
3. I do not notice that the immediate part for the sw instruction is different from others, and I was confusing why my lw instructions work correctly but the sw instructions always take the data to the wrong place. It made me realize the importance of reading the spec carefully.

Development Environment.

1. My development environment is Windows with iverilog and wavetrace from vscode.