CA Lab2 Report

資工三 B10705005 陳思如

Q1 Modules Explanation

- Adder

  Take two signed value then add them together. It is used to add the pc counter and calculate the pc branch value.

- ALU_Control

  Take the opcode and the 10-bit function code in instruction to determine which control signal to sent to the ALU. The new implemented instructions `lw` and `sw` has the same control signal as `add`. And the `beq` has the same control signal as `sub`.

- ALU

  This module actually does the same action as the last lab. It does the operation based on the value of the ALU control signal.

- Control

  It decided the `regwrite`, `memtoreg`, `memread`, `memwrite`, `aluop`, `alusrc`, `branch` based on the 7-bit opcode value. However, if the control module get the noop signal, it will set all these 7 registers to zero since it cannot do anything in the noop cycle.

- CPU

  In this module, we just concat each module with wires. Two things to be careful is that we need to make some wire be **signed** value to make sure the operation will behaved as expected and we need to name some variables as the same in the `testbench.v` so that the testbench can call and check those variables.

- Pipelined redisters: IFID / IDEX / EXMEM / MEMWB

  I have implemented these four pipelined registers in different module but the logic of them is same. I will explain them together.

  Pipelined registers will taken in clock and reset signal. It will passses the values down to next stage when it is at the clock edge. It will reset the all the values to zeros when the reset signal is at edge.

  The special case is the IFID registers since it have another stall and flush signal. For stall signal, it will passes the same value again to the next stage to make sure that the CPU does wait for a cycle. For flush signal, it will does the same operation as reset. It sets all the values to zeros for the next stage.

- Forward

  This module determined whether the operation in EX stage will use what value in the registers. The default forward value is `00` to choose the data reads in the registers file. However, if the **MEM** stage has operation on the same registers and the `regwrite` signal is on, the foward signal will be set to `10`. If the **WB** stage has operation on the same registers and the `regwrite` signal is on, the forward signal then will be set as `01`.

  Also, we will check with the MEM stage first and make sure that if MEM and WB both have operations on the same registers, we will take the value in the MEM stage.

- Hazard

This module checks if the `lw` and its next operation are processed on the same registers. It will sends out `pcwrite`, `stall_o`, `noop` these three values to tell the CPU to stall the cycle or not. It checks with the `memread` signal in EX stage and compares with the read registers in ID stage and EX stage write registers.

The default value for pcwrite is `1`, stall_o is `0`, noop is `0`. If the above condition if satisfied which means it need to stall. The pcwrite will be set to `0`, stall_o be `1`, noop be `1`.

- ImmGen

  It takes in all 32-bit instructions and based on the opcode to decide which immediated value should extract from the instruction. Also, is will do the sign extend to the immediate value.

  This module will only extract 12-bit immediate value from the instruction. The front 20 bits will come from the sign extend.

  For the `beq` instruction, which is quite different from the others is the it has shift left one bit so the sign extend check will be checking the 12 bit and only extend 19 bits.

- MUX

  It chooses the value based on the select signal. It only takes two inputs and a 1-bit select signal. It is used in the PC address choice and the immediate value and data2 choice.

- MUX4

  It also chooses the value based on the select signal. I takes three inputs and a 2-bit select signal. It is used in the EX stage choosing the forwarding value to do the ALU operation.

Q2 Difficulties and Solutions

1. I have encountered that my simulation stucks at the first cycle and the output is also stucked. Then I have checked the verilog tutorial which found that I need to initialize my registers. Also, I have type error in the variable name with the uppercase and lowercase. So I solved it by initializing all the registers value and fix the type error.
2. I have used most of the output as reg format. This will stores the value in it. I forgot to set them back to the original values after every cycle at first. This causes some unexpected behavior on my program. I checked on the waveform output to find out this problem and solve it.
3. I forget to set the immediate value as signed wire. This cause my PC branch value to be a huge value which causes my programs errors. After I checked the waveform and trying to print out the immediate value, I have fixed the problem.

Q3 Enviroment

Window and iverilog. I didn't use the docker environment. I only do the second option for compilation.