CSC3050 Assignment 4

121020163 沈驰皓

1. Overview of 5-stage CPU And Implementation Ideas

In this assignment, we are asked to implement a 5-stage pipelined CPU, which can implement MIPS instructions in a pipelined way. Basically, it contains five stages, `IF`, `ID`, `EX`, `MEM` and `WB`. `IF` stage fetches instruction from the memory given the PC address, `ID` divides and sends the instruction to several parts and then decodes some of the parts by registers and the control unit. `EX` uses the ALU to do some arithmetic operations. `MEM` fetches the data from or stores the data to the data memory. Finally. `WB` writes the data back to the registers if needed. Also, we need another hazard unit to deal with the hazard problems.

Here we implement each stage in a separate file. Therefore, five files named `IF.v`, `ID.v`, `EX.v`, `MEM.v` and `WB.v` are needed. Each file contains several logic units to perform their functions in its corresponding stage. Additionally, a file named `Harzard.v` is implement to solve the hazard problems using data forwarding and stalling according to a specific hazard problem.

To simulate the real-time testing procedure, `test_cpu.v` is implemented to generate a clock signal for the CPU to operate synchronously. And in order to link all five stages, we also implement a `cpu.v` to connect all of the input and output wires together to achieve all functions.

2. Implementation Details

For all stage files except `IF.v`, we implement an intermediate pipeline registers module to receive the signals from previous stage and send them to the next stage positively triggered by the clock signal. For all hazard problems, we'll explain them in the `Hazard.v` part.

a. IF.v

In this file, we implement five modules. Firstly, a `PC_SRC` module receive several control signals to decide which source should be the next PC value, and then send the corresponding signal to an implemented a `MUX5_BIT32` multiplexor for selecting the 32-bit PC value. Then this 32-bit value is sent to a `PC` module which is waited to be triggered by a positive edge clock signal for a new cycle. After triggered, the value is sent to both `INSTR_MEM` module for fetching the machine code at the corresponding address, and `PC_ADD4` module to create a new address next to the current address for further use.

b. ID.v

It then comes to the `ID` stage. Firstly, a `CONTROL_UNIT` module is served as a control unit, which receives the opcode as well as the function code and outputs several control signals according to the instruction type. At the same time, a `REG_FILE` module is simulated as the register file in CPU. It receives the register numbers needed to be read and outputs the stored value of the required registers. It is also needed in stage five for writing back the given value to an asked register. Also, a `SIGN_EXT` module extends the immediate as a signed value for I-type instructions. Two modules named `BRANCH_GEN` and `JUMP_GEN` generate the branch destination and jump destination addresses, and send them to stage one for `PC_SRC` to choose the one that should be used in case of branch or jump instructions.

c. EX.v

For `EX` stage, first we need two 3 to 32-bit multiplexors named `MUX3_BIT32` to select the two operands for ALU from registers, the result of ALU or the write back data, due to the forwarding reason. The result from the second multiplexor branches and a wire links to it and directly transmits the result to the next stage served as the address for data memory for `sw` operation. Meanwhile, a `MUX2_BIT32` multiplexor is used to select the register result for R-type instructions, or the signed intermediate for I-type instructions to be the second result. After that, the most important part in the `EX` stage, the `ALU` module receives the operands and control signals passed down from the `CONTROL_UNIT`. It uses the control signals to decide which operation to perform, do the operation and output the calculation result to the next stage. At the same time, it'll raise the zero flag for `beq` and `bne` if the two operands are the same, and it'll raise the negative flag for `slt` if `Rs` is smaller than `Rt`. `MUX2_BIT5` multiplexor is also used here in order to choose which register to be written in the last stage from `Rt` or `Rs` and send the 5-bit register number to the next stage.

d. MEM.v

In this stage, a `DATA_MEM` module is simulated as the data memory in CPU. Controlled by the control signal, it reads or writes the data from or to the simulated data memory for `sw` or `lw` instructions. Otherwise, the output of the ALU unit will directly bypass the `DATA_MEM` module and is sent to the next stage.

e. WB.v

Here we only use a `MUX2-BIT32` multiplexor to select the output depending on whether the data comes from the data memory or the ALU unit. For the register written part, it will then be done in the `ID` stage according to the corresponding control signal.

f. Hazard.v

This file is created to deal with the hazard problems. For most of the operations, we use forwarding strategy. If the received register is in `ID` stage, we then insert a NOP cycle. If it is received in `EX` stage, we then directly link the ALU result or the write back result to the operands and let the multiplexor to decide which one to be chosen.

Also, for `lw` hazard, we use the stall strategy, which inserts NOPs before the required word is loaded. For example:

```
lw $t2, 20($t1)
and $t4, $t2, $t5
```

A flush operation in the `IF_ID` module would happen if a branch or jump operation occurs. However, if there is a branch or `jr` instruction after the `lw` instruction in which a hazard is happened, then we should not use stalling, since both flushing and stalling would cause an exception. In this case, forwarding strategy is still effective.

3. Test Cases That Can Be Passed

There are total 8 test files. And this 5-stage CPU can pass all 8 test cases as required. After loading the machine code to the `CPU_instruction.bin` file and executing the "make" command in the console, we can directly get the results in `data.bin`, which match the given correct results.