Computer Architecture: Project #1

b07902047 羅啓帆
b07902139 鄭豫澤

December 2019

1 Module implementations

We implemented pipeline first, then plug Forwarding_unit and HazardDetection unit in.

1.1 CPU.v

We put the four pipeline registers into four modules, and each stage of pipeline is implemented directly in CPU. In short, CPU is just a place for wires connecting all components together.

1.2 Pipeline registers: IFID.v, IDEX.v, EXMEM.v, MEMWB.v

Apart from data that needs to be passed down to the next stage, these pipeline registers are all given the clock signal (*clk_i*). They will pass data down on the rising edge of clock. Though it seems that they are easy to implement, we encountered some (time-consuming) bugs. First, we forgot to initialize them. As we assume that this CPU only reset once at the beginning, this problem is solved by add the *initial* block. Second, we (mis)ued $'='$ instead of $'<='$ in *always @ (posedge clk_i)* block. Using $'='$ will cause pipeline to malfunction.

Note that we implemented the control unit in IDEX module for convenience.

1.3 Forwarding part: Forwarding_unit.v

The forwarding rules are as follow:

```
if (RegWrite_EXMEM && rd_EXMEM != 0 && rd_EXMEM == rs1_IDEX) Forward EX result to rs1
else if (RegWrite_MEMWB && rd_MEMWB != 0 && rd_MEMWB == rs1_IDEX) Forward MEM result
    to rs1

if (RegWrite_EXMEM && rd_EXMEM != 0 && rd_EXMEM == rs2_IDEX) Forward EX result to rs2
else if (RegWrite_MEMWB && rd_MEMWB != 0 && rd_MEMWB == rs2_IDEX) Forward MEM result
    to rs2
```

Listing 1: Rules

1.4 Hazard dectection part: HazardDetection.v

The stall and flush rules are as follow:

```
if (MemRead_IDEX && rd_IDEX != 0 && (rd_IDEX == rs1_IFID || rd_IDEX == rs2_IFID)) :
    stall
if (ID_equal && isBranch) : flush
```

Listing 2: Rules

1.5 Adder.v

Just a simple Adder, read from *data1_i*, *data2_i* and put *data1_i* + *data2_i* to *data_o*.

1.6 ALU_MUX.v

This is a 3-to-1 multiplexer. It's used in front of the two input of ALU.

1.7 ALU.v

A unit for calculation. More specifically, given two numbers, it is able to perform addition(0010), subtraction(0110), bitwise and(0000), bitwise or(0001) and multiplication(1111). The binary number in parenthesis is the corresponding opcode.

1.8 Control.v

1.9 CtrlMux.v

This is a 2-to-1 multiplexer used is ID stage. The difference between this and MUX is the size of input data. The data size for CtrlMux 5 bits, while the data size for MUX is 32 bits.

1.10 EQUAL.v

This module will output 1 when the two inputs of it is equal, otherwise it will output 0. This is used for getting the result of command *beq* earlier.

1.11 MUX.v

This is a 2-to-1 multiplexer with data size of 32 bits used at both WB and IF stage.

1.12 SignExtend.v

This module sign extends the input from 12 bits to 32 bits.

2 Work division

1. Basic pipeline (CPU): b07902139
2. Data forwarding & hazard dectection: b07902047
3. xReport: b07902047 & b07902139