UM–SJTU Joint Institute

Intro to Computer Organization
(VE370)

Lab Report

Lab 5

Resolving Hazards

Name: Yiwen Yang ID: 519370910053
Name: Rundong Tang ID: 519370910050
Name: Shuhui Yang ID: 519370910064

1 Introduction

Data hazards and control hazards may arise in pipelined processors, and can be resolved by adding the necessary forwarding mechanism, hazard detection components, and signals for stalls to the pipelined processor. In this lab, we use Verilog HDL to model these components based on the pieplined processor designed in the previous lab, and implement the processor in the Xilinix FPGA board.

1.1 Forwarding Unit

- Inputs: control signals and data from pipeline registers **IF_ID**, **ID_EX**, **EX_MEM**, **MEM_WB**.
- Outputs: ALU forwarding path select signals **forward_A**, **forward_B**, branch forwarding path select signals **forward_A_comp**, **forward_B_comp**.
- ALU forwarding: if there is EX hazard, **forward_A\forward_B** = 2'b10; if there is MEM hazard, **forward_A\forward_B** = 2'b01.
- Branch forwarding: if there is EX hazard, **forward_A_comp\forward_B_comp** = 2'b10; if there is MEM hazard, **forward_A_comp\forward_B_comp** = 2'b01.

1.2 Hazard Detection Unit

- Inputs: control signals and data from pipeline registers **IF_ID**, **ID_EX**, **EX_MEM**, **MEM_WB**.
- Outputs: control signals **control_src**, **pc_write**, **IF_ID_write**.
- Hazard types: data hazards and control hazards.
- Data hazards: load hazard, save hazard and load-save hazard.
- Control hazards: load-branch hazard, ALU instrcution-branch hazard.

1.3 Comparator

- Inputs: data to be compared.
- Outputs: control signals **zero_BR**, **lt_Zero_BR**.
- When **forward_A_comp\forward_B_comp** = 2'b00, data comes from **read_data**; When **forward_A_comp\forward_B_comp** = 2'b01, data comes from **write_data**; When **forward_A_comp\forward_B_comp** = 2'b10, data comes from **ALU_result_MEM**.

2 Illustrations for Data Hazards and Control Hazards

2.1 Data hazard for R-type instructions

For R-type data hazards, we choose the instruction "add t2 t0 t3" in line 8 and the instruction "and t1 t2 t3" in line 9 as an example. We can see from the following result that at clock cycle 13 when the instruction "add t2 t0 t3" is in the WB stage, t2 changes to ffffff32 correctly. Then at clock cycle 14 when the next R-type instruction "and t1 t2 t3" is in the WB stage t1 changes to ffffff10 correctly, which means the data hazard is resolved in our design.

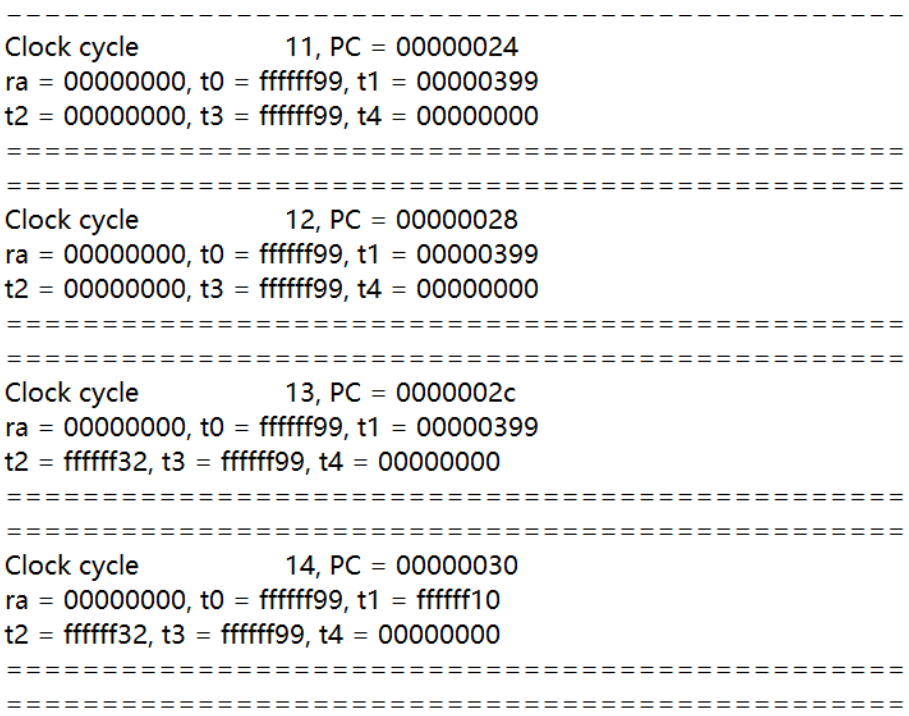

Figure 1: Data hazard for R-type instructions

2.2 Data hazards between load-use instruction and save instruction

For Data hazards between load-use instruction and save instruction, we choose the instruction "lb t0 4(x0)" at line 3 and the instruction "sw t0 0(x0)" at line 4 for an example. We can see that at clock 6, t0 changes to ffffff99 correctly and after executing the instruction "sw t0 0(x0)", the data memory at address 0(x0) becomes 0xffffff99 correctly, which means the data hazard between load-use instruction and save instruction is resolved in our design.

```
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              4, PC = 00000010
ra = 00000000, t0 = 00000000, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              5, PC = 00000014
ra = 00000000, t0 = 00000000, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              6, PC = 00000018
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              7, PC = 0000001c
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              8, PC = 0000001c
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
------------------------------------------------
```

Figure 2: Data hazards between load-use instruction and save instruction

0x00000024	0x0003f313	0x13	0xf3	0x03	0x00
0x00000020	0x01c3f333	0x33	0xf3	0xc3	0x01
0x0000001c	0x01c283b3	0xb3	0x83	0xc2	0x01
0x00000018	0x01c29c63	0x63	0x9c	0xc2	0x01
0x00000014	0x00002e03	0x03	0x2e	0x00	0x00
0x00000010	0x02030063	0x63	0x00	0x03	0x02
0x0000000c	0x00502023	0x23	0x20	0x50	0x00
0x00000008	0x00400283	0x83	0x02	0x40	0x00
0x00000004	0x00000399	0x99	0x03	0x00	0x00
0x00000000	0xffffff99	0x99	0xff	0xff	0xff
-	-	-	-	-	-
-	-	-	-	-	-

Figure 3: Data hazards between load-use instruction and save instruction

2.3 Control hazards between load-use instruction and branch

For control hazards between load-use instruction and branch, we choose the instruction "lw t3 0(x0)" at line 6 and the instruction "bne t0 t3 wrong_branch" at line 7 as an example. We can see that at clock cycle 9, t3 changes to ffffff99 correctly. Then, we can see that the next PC value is the last PC + 4, which means the branch is not taken, which is correct and means the control hazard between load-use instruction and branch is solved in our design.

```
Clock cycle              7, PC = 0000001c
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              8, PC = 0000001c
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = 00000000, t4 = 00000000
================================================
================================================
Clock cycle              9, PC = 0000001c
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = ffffff99, t4 = 00000000
================================================
================================================
Clock cycle              10, PC = 00000020
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = ffffff99, t4 = 00000000
================================================
================================================
Clock cycle              11, PC = 00000024
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = ffffff99, t4 = 00000000
================================================
================================================
Clock cycle              12, PC = 00000028
ra = 00000000, t0 = ffffff99, t1 = 00000399
t2 = 00000000, t3 = ffffff99, t4 = 00000000
================================================
================================================
Clock cycle              13, PC = 0000002c
```

Figure 4: Control hazards between load-use instruction and branch

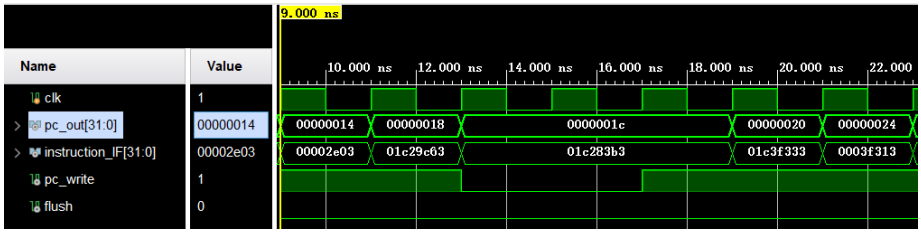

Figure 5: Control hazards between load-use instruction and branch

2.4 Control hazard between R-type instruction and branch instruction

Control hazard between R-type instruction and branch instructions, we choose the instruction "sub t0 t1 x0" at line 11 and the instruction "bge t0 t1 right_branch" in line 12 as an example. We can see from the simulation result that after the instruction "bge t0 t1 right_branch" the next PC is not equal to the current PC + 4, which means the branch is taken and is the correct result. This means the control hazard between R-type instruction and branch instruction is solved in our design.

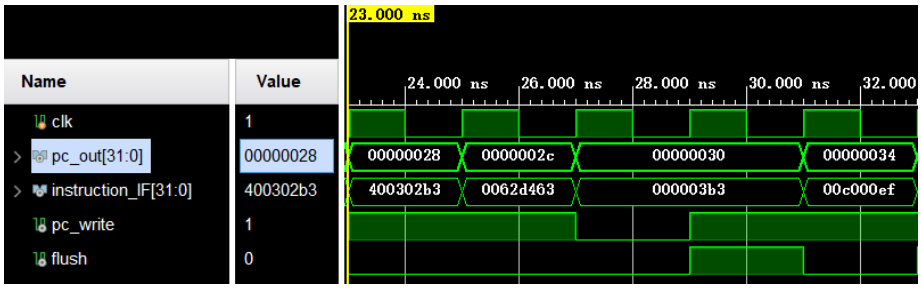

Figure 6: Control hazard between R-type instruction and branch instruction

2.5 Flush

Since we assume branch always not taken, we need to flush if the branch should be taken. For this condition, we choose the instruction "bge t0 t1 right_branch" at line 12 for an example. We can see from the simulation result that when PC turns from 0x00000030 to 0x00000034, the instruction from IF to ID stage changes from 0x000003b3 to 0x00000013, which means the instruction is flushed. This means our design is correct when there needs a flush.

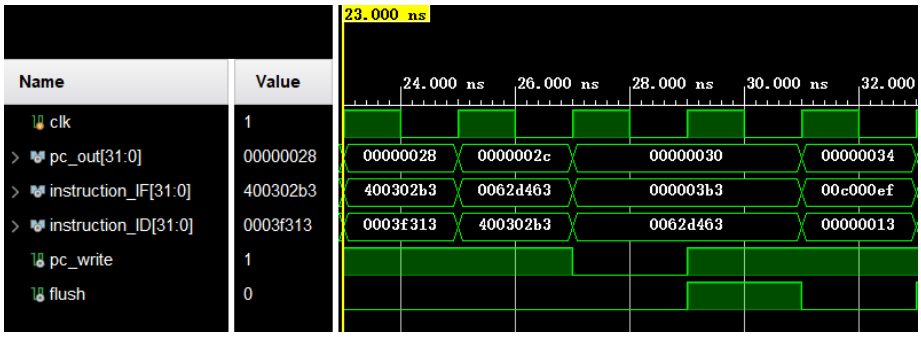

Figure 7: Illustration for flush

3 RTL Schematic

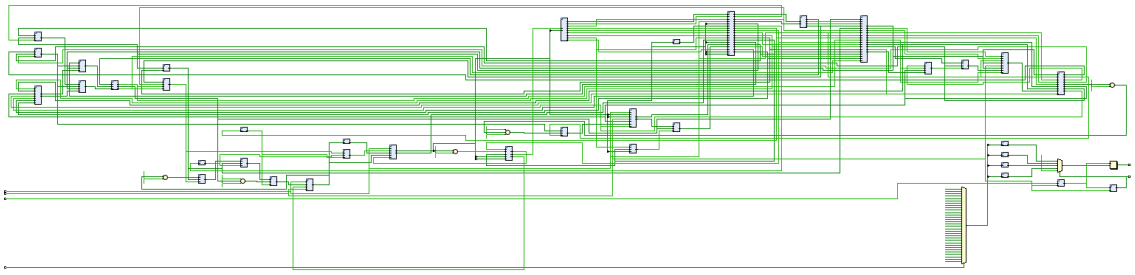

Figure 8: RTL Schematic