浙江大学

本科实验报告

课程名称：	计算机体系结构
姓 名：	张志心
学 院：	竺可桢学院
专 业：	混合班
学 号：	3210106357
指导教师：	常瑞
日 期：	2023 年 10 月 17 日

浙江大学实验报告

课程名称： 计算机体系结构 实验类型： 综合

实验项目名称： 实验1-流水线 RISC-V CPU 设计

学生姓名： 张志心 专业： 计算机科学与技术 学号： 3210106357

同组学生姓名： 无 指导教师： 常瑞 助教： 邱明冉

实验地点： 曹光彪西301 实验日期： 2023 年 10 月 18 日

1 实验目的及环境

1.1 实验目的

- 温故流水线 **CPU** 设计
- 了解并实现 **RV32I** 指令集
- 理解旁路优化（**Forwarding**）

1.2 实验环境

- **HDL**：**Verilog**、**SystemVerilog**
- **IDE**：**Vivado**
- **开发板**：**NEXYS A7**（**XC7A100TCSG324**）

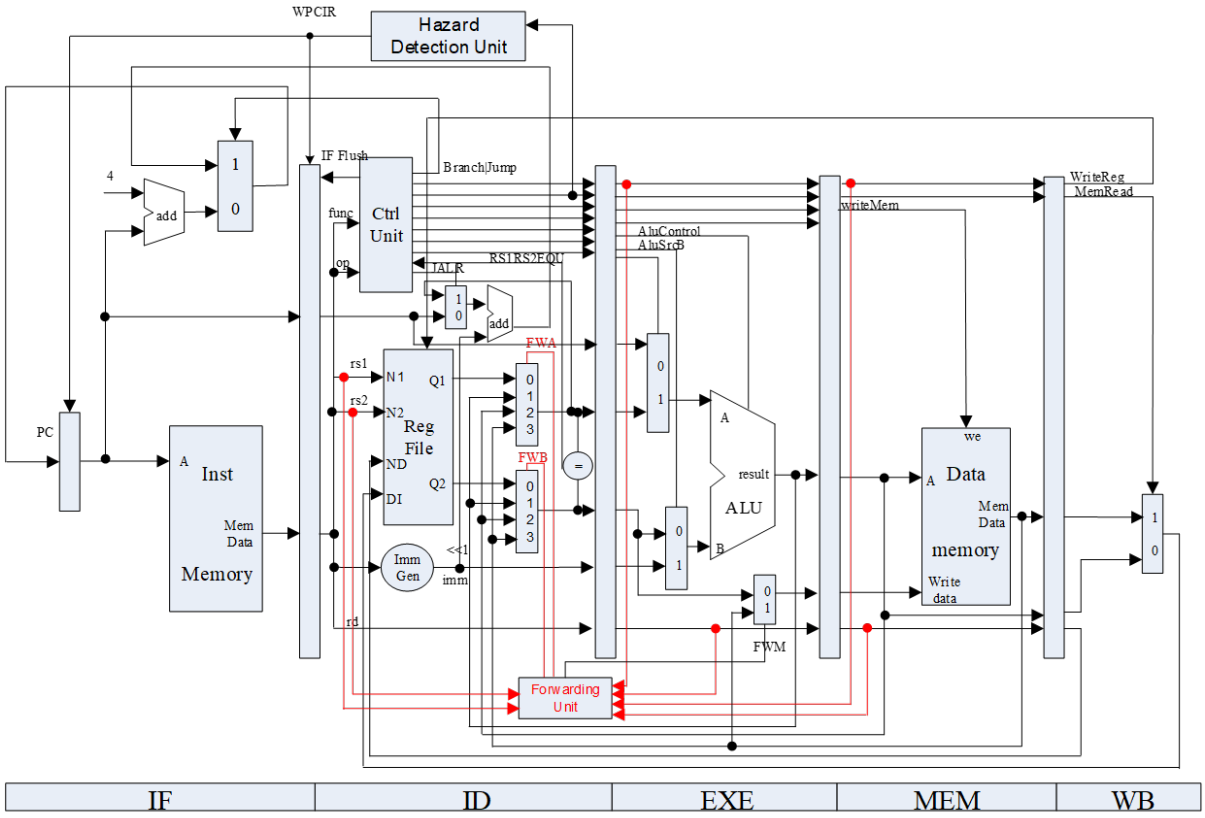
2 实验要求及步骤

2.1 实验要求

1. 实现 **RV32I** 中的所有指令(除了 **fence, ecall, ebreak)**
2. 实现流水线中的 **forwarding**
3. 通过仿真测试和上板验证

2.2 实验步骤

1. 根据 **RISC-V** 非特权级手册完成部分 **RV32I** 指令集
2. 在流水线中加入 **forwarding** 机制
3. 在给定的 **SOC** 中，加入自己的 **CPU**，通过仿真测试和上板验证



ID 段 Forwarding 基本实现原理

3 实验过程及记录

3.1 RV32core.v

```
    MUX2T1_32 mux_IF(.I0(PC_4_IF),.I1(jump_PC_ID),.s(Branch_ctrl),.o(next_PC_IF));

    MUX4T1_32
mux_forward_A(.I0(rs1_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
        .s(forward_ctrl_A),.o(rs1_data_ID));

    MUX4T1_32
mux_forward_B(.I0(rs2_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
        .s(forward_ctrl_B),.o(rs2_data_ID));

    MUX2T1_32
mux_A_EXE(.I0(PC_EXE),.I1(rs1_data_EXE),.s(ALUSrc_A_EXE),.o(ALUA_EXE));

    MUX2T1_32
mux_B_EXE(.I0(Imm_EXE),.I1(rs2_data_EXE),.s(ALUSrc_B_EXE),.o(ALUB_EXE));


    MUX2T1_32
mux_forward_EXE(.I0(rs2_data_EXE),.I1(Datain_MEM),.s(forward_ctrl_ls),.o(Dataout_EXE
));
```

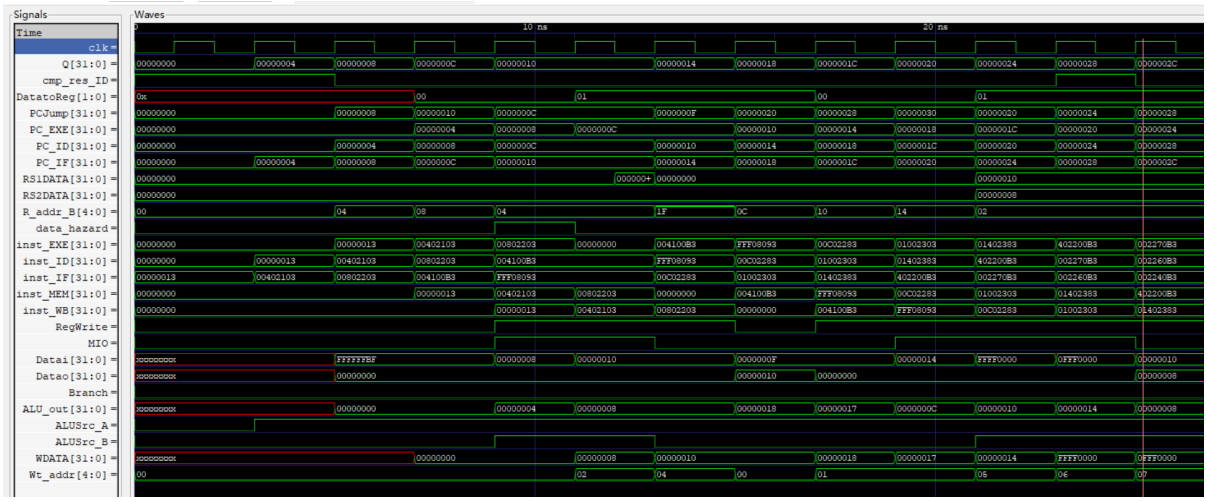
3.2 HazardDetectionUnit.v

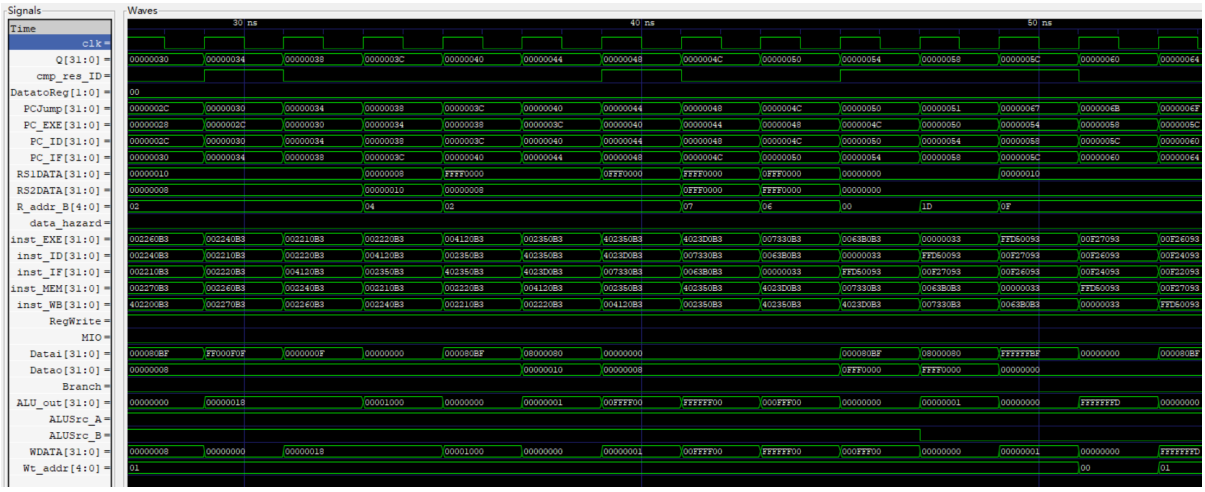
```
`timescale 1ps/1ps

module HazardDetectionUnit(
    input Branch_ID, rs1use_ID, rs2use_ID,
    input RegWrite_EX, RegWrite_MEM, MemRead_EX, MemRead_MEM, MemWrite_EX,
MemWrite_ID,
    input[4:0] rd_EXE, rd_MEM, rs1_ID, rs2_ID, rs2_EXE,
    output PC_EN_IF, reg_FD_EN, reg_FD_stall, reg_FD_flush,
        reg_DE_EN, reg_DE_flush, reg_EM_EN, reg_EM_flush, reg_MW_EN,
    output reg forward_ctrl_ls,
    output reg [1:0] forward_ctrl_A,
    output reg [1:0] forward_ctrl_B
);

    wire stall = ((rs1use_ID && rs1_ID == rd_EXE) || (rs2use_ID && ~MemWrite_ID &&
rs2_ID == rd_EXE))
                && MemRead_EX;

    assign reg_EM_flush = 1'b0;

    assign reg_EM_EN = 1'b1;
    assign reg_DE_EN = 1'b1;
    assign reg_MW_EN = 1'b1;
    assign reg_FD_EN = 1'b1;

    assign reg_FD_flush = Branch_ID;
    assign reg_FD_stall = stall;
    assign reg_DE_flush = stall;
    assign PC_EN_IF = ~stall;

    // forward_ctrl_ls
    always @(*) begin
        // save after load
        if ((|rs2_EXE) & MemWrite_EX) begin
            // MEM is load
            if (rs2_EXE == rd_MEM && MemRead_MEM)
                forward_ctrl_ls = 1'b1;
            else forward_ctrl_ls = 1'b0;
        end
        // EX is not save
        else forward_ctrl_ls = 1'b0;
    end

    // forward_ctrl_A
    always @(*) begin
        // need rs1
        if (rs1use_ID) begin
            // read after write (no load)
            if (rs1_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
                forward_ctrl_A = 2'b01;
            end else if (rs1_ID == rd_MEM && RegWrite_MEM) begin
                // MEM is load
                if (MemRead_MEM) begin
```

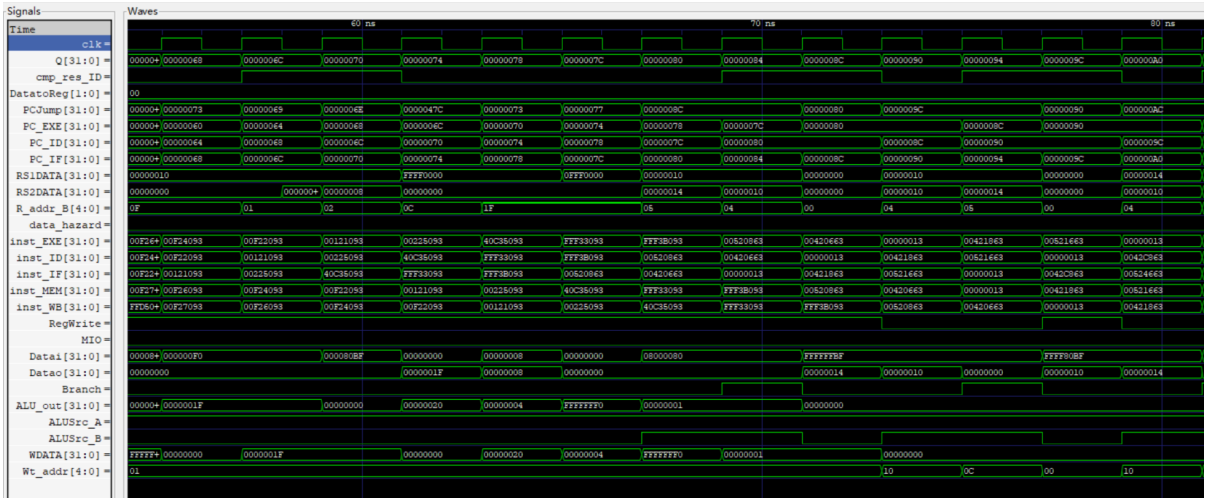
```
                    forward_ctrl_A = 2'b11;
                end else forward_ctrl_A = 2'b10; // MEM is write other than load
            end else forward_ctrl_A = 2'b00;
        end
        else forward_ctrl_A = 2'b00;
    end

    // forward_ctrl_B
    always @(*) begin
        // not save and need rs2
        if (rs2use_ID && ~MemWrite_ID) begin
            if (rs2_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
                // write and not load
                forward_ctrl_B = 2'b01;
            end else if (rs2_ID == rd_MEM && RegWrite_MEM) begin
                // MEM is load
                if (MemRead_MEM) begin
                    forward_ctrl_B = 2'b11;
                end else forward_ctrl_B = 2'b10; // MEM is write other than load
            end else forward_ctrl_B = 2'b00;
        end
        else forward_ctrl_B = 2'b00;
    end

endmodule
```

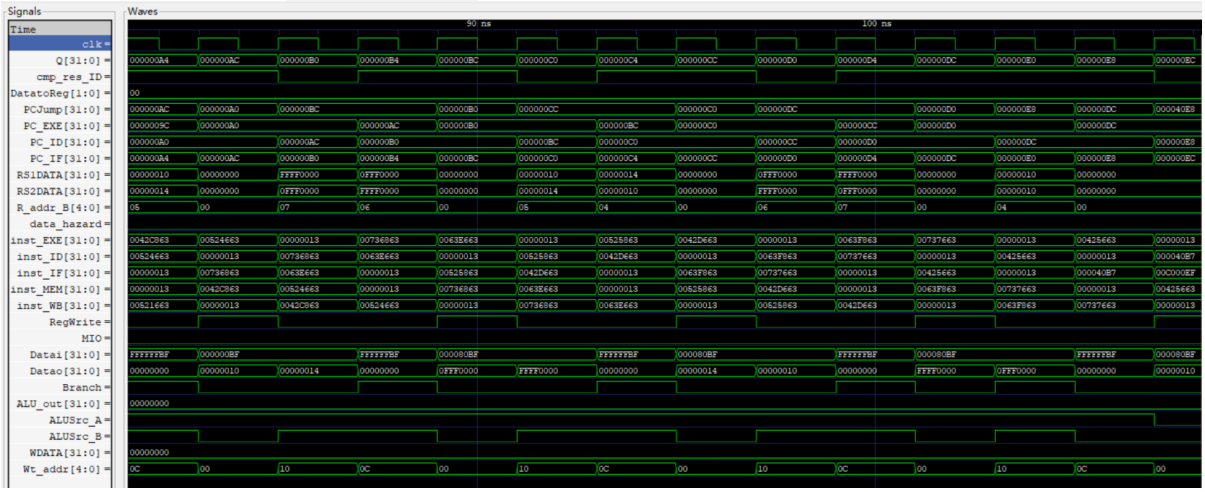
3.3 仿真验证

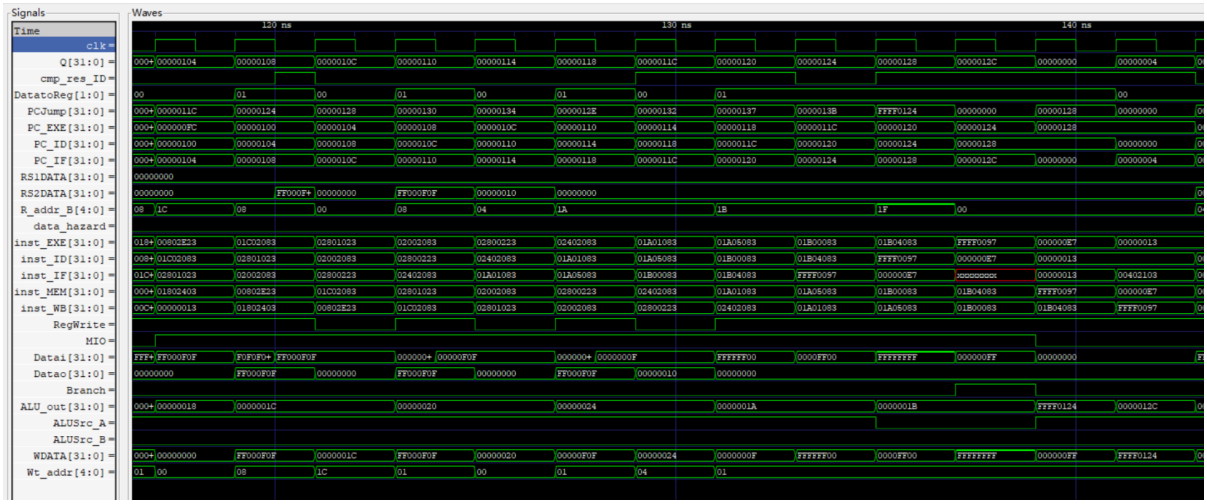
仿真结果正确:




Signals
Waves
Time
clk
Q[31:0]
cmp_res_ID
DatatoReg[1:0]
PCJump[31:0]
PC_EXE[31:0]
PC_ID[31:0]
PC_IF[31:0]
RS1DATA[31:0]
RS2DATA[31:0]
R_addr_B[4:0]
data_hazard
inst_EXE[31:0]
inst_ID[31:0]
inst_IF[31:0]
inst_MEM[31:0]
inst_WB[31:0]
RegWrite
MIO
Datai[31:0]
Datao[31:0]
Branch
ALU_out[31:0]
ALUSrc_A
ALUSrc_B
WDATA[31:0]
Wt_addr[4:0]


Signals
Waves
Time
clk
Q[31:0]
cmp_res_ID
DatatoReg[1:0]
PCJump[31:0]
PC_EXE[31:0]
PC_ID[31:0]
PC_IF[31:0]
RS1DATA[31:0]
RS2DATA[31:0]
R_addr_B[4:0]
data_hazard
inst_EXE[31:0]
inst_ID[31:0]
inst_IF[31:0]
inst_MEM[31:0]
inst_WB[31:0]
RegWrite
MIO
Datai[31:0]
Datao[31:0]
Branch
ALU_out[31:0]
ALUSrc_A
ALUSrc_B
WDATA[31:0]
Wt_addr[4:0]


Signals
Waves
Time
clk
Q[31:0]
cmp_res_ID
DatatoReg[1:0]
PCJump[31:0]
PC_EXE[31:0]
PC_ID[31:0]
PC_IF[31:0]
RS1DATA[31:0]
RS2DATA[31:0]
R_addr_B[4:0]
data_hazard
inst_EXE[31:0]
inst_ID[31:0]
inst_IF[31:0]
inst_MEM[31:0]
inst_WB[31:0]
RegWrite
MIO
Datai[31:0]
Datao[31:0]
Branch
ALU_out[31:0]
ALUSrc_A
ALUSrc_B
WDATA[31:0]
Wt_addr[4:0]

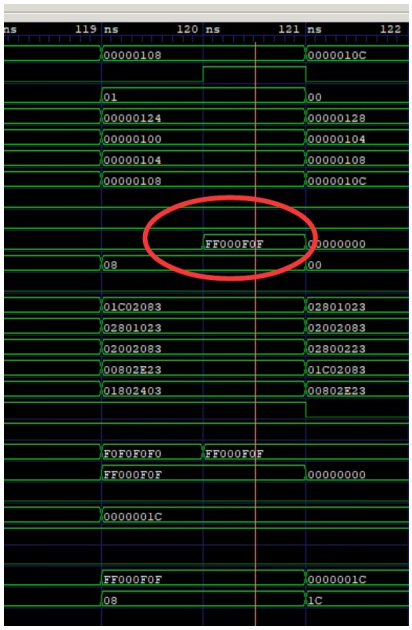


图中，在**0x128**指令后第二个时钟周期正确跳转回 **0x0**。

在 **0x100** 指令：

```
lw    x8, 24(x0)      ******************** PC = 0XF8
# x8 = 0xFF000F0F

sw    x8, 28(x0)

lw    x1, 28(x0)      ******************** PC = 0X100
```

其中 **x1** 为 `0xFF000F0F`：



3.4 上板验证

4 思考题

###

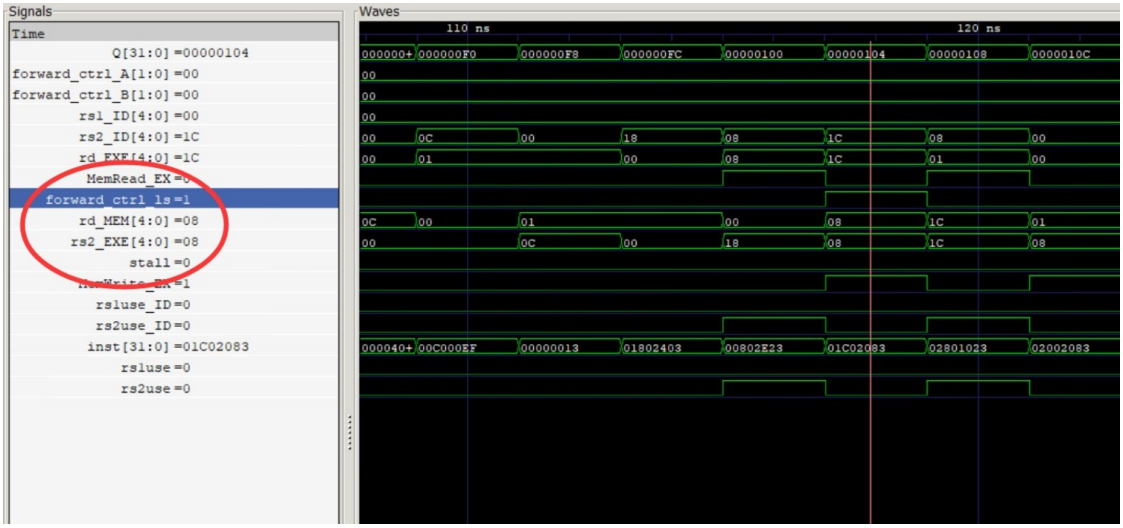
添加了 **forwarding** 机制后，是否观察到了 **stall** 延迟减少的情况？请在测试程序中给出 **forwarding** 机制起到实际作用的位置，并给出仿真图加以证明。

观察到三种情况的stall延迟减少：

- **save after load**，**stall** 从原先的两个周期减少到零。

```
// HazardDetectionUnit.v

// forward_ctrl_ls
    always @(*) begin
        // save after load
        if ((|rs2_EXE) & MemWrite_EX) begin
            // MEM is load
            if (rs2_EXE == rd_MEM && MemRead_MEM)
                forward_ctrl_ls = 1'b1;
            else forward_ctrl_ls = 1'b0;
        end
        // EX is not save
        else forward_ctrl_ls = 1'b0;
    end

// RV32core.v
MUX2T1_32 mux_forward_EXE(
    .I0(rs2_data_EXE),.I1(Datain_MEM),.s(forward_ctrl_ls),.o(Dataout_EXE)
);
```

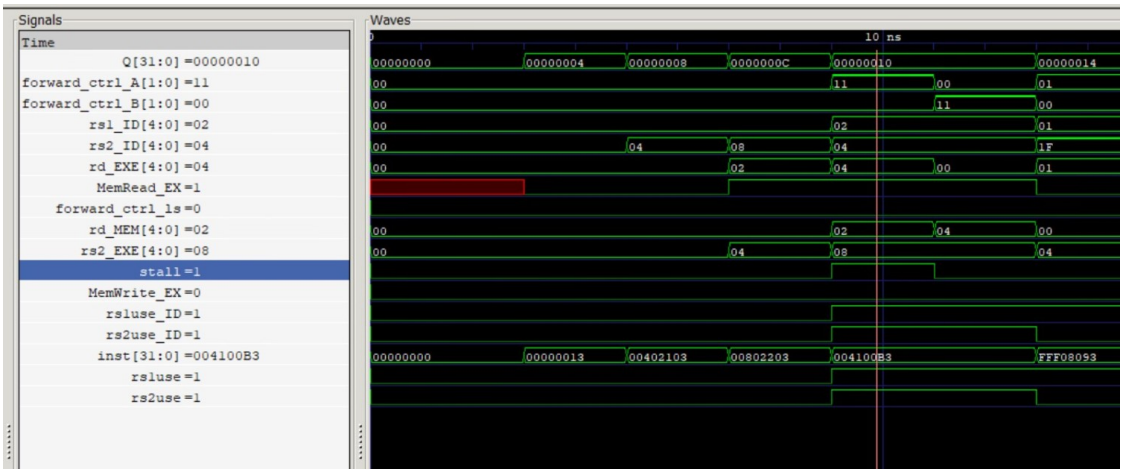
这种转发利用了 **save** 在 **EX** 阶段不使用 `rs2data` 的特点，在检测到 **EX** 阶段的 **save** 指令和 **MEM** 阶段的 **load** 指令发生数据冲突时，上条 **load** 指令得到结果后将 `forward_ctrl_ls` 信号置高电平，将 **load** 指令的结果转发到 **save** 指令 **EX** 阶段的末尾。



- **use after load**，**stall** 从原先的两个周期减少到一个周期。

```
wire stall = ((rs1use_ID && rs1_ID == rd_EXE) || (rs2use_ID && ~MemWrite_ID && rs2_ID == rd_EXE))
           && MemRead_EX;

assign reg_FD_stall = stall;
assign reg_DE_flush = stall;
assign PC_EN_IF = ~stall;
```

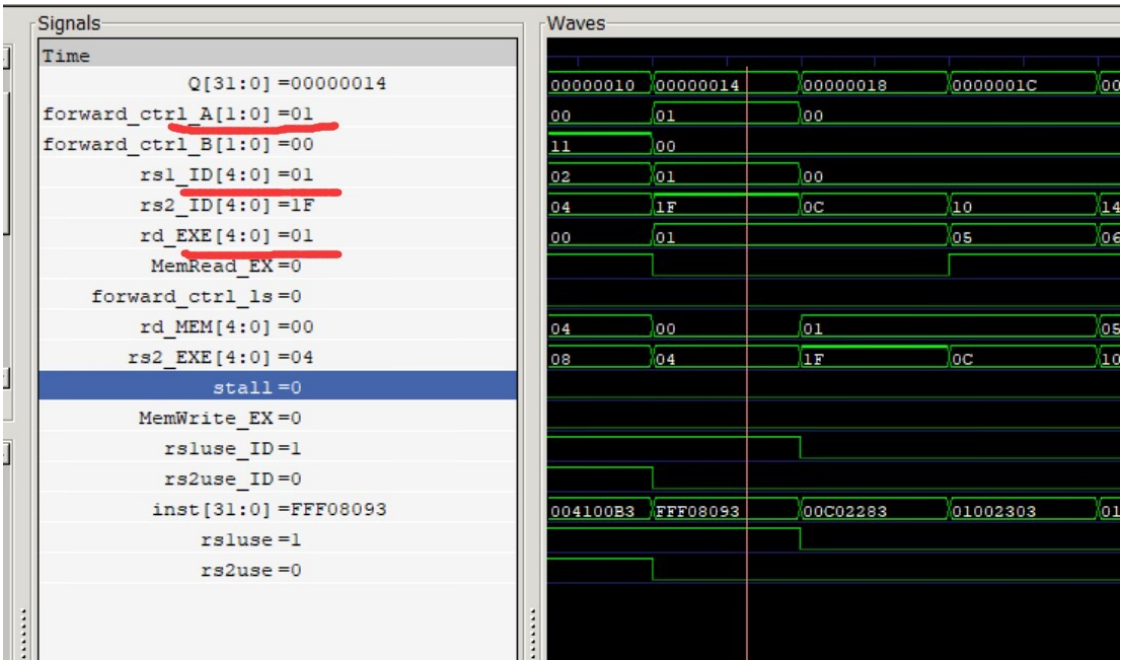
该方法在检测到上条出于 **EX** 阶段的 **load** 指令和下条处于 **ID** 阶段读取寄存器指令发生冲突（除了**save** 的 **rs2**）时进行 **stall**，暂停 **PC** 更新，阻塞 **IF-ID** 的数据传送，并在 **EX** 段冲刷一个气泡，使得 **load** 指令和读取寄存器指令至少相距两个周期。



- **use after write**，**stall** 从原先的两个周期减少到零。

```
    // forward_ctrl_A
    always @(*) begin
        // need rs1
        if (rs1use_ID) begin
            // read after write (no load)
            if (rs1_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
                forward_ctrl_A = 2'b01;
            end else if (rs1_ID == rd_MEM && RegWrite_MEM) begin
                // MEM is load
                if (MemRead_MEM) begin
                    forward_ctrl_A = 2'b11;
                end else forward_ctrl_A = 2'b10; // MEM is write other than load
            end else forward_ctrl_A = 2'b00;
        end
        else forward_ctrl_A = 2'b00;
    end

    // forward_ctrl_B
    always @(*) begin
        // not save and need rs2
        if (rs2use_ID && ~MemWrite_ID) begin
            if (rs2_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
                // write and not load
                forward_ctrl_B = 2'b01;
            end else if (rs2_ID == rd_MEM && RegWrite_MEM) begin
                // MEM is load
                if (MemRead_MEM) begin
                    forward_ctrl_B = 2'b11;
                end else forward_ctrl_B = 2'b10; // MEM is write other than load
            end else forward_ctrl_B = 2'b00;
        end
        else forward_ctrl_B = 2'b00;
    end

endmodule
```

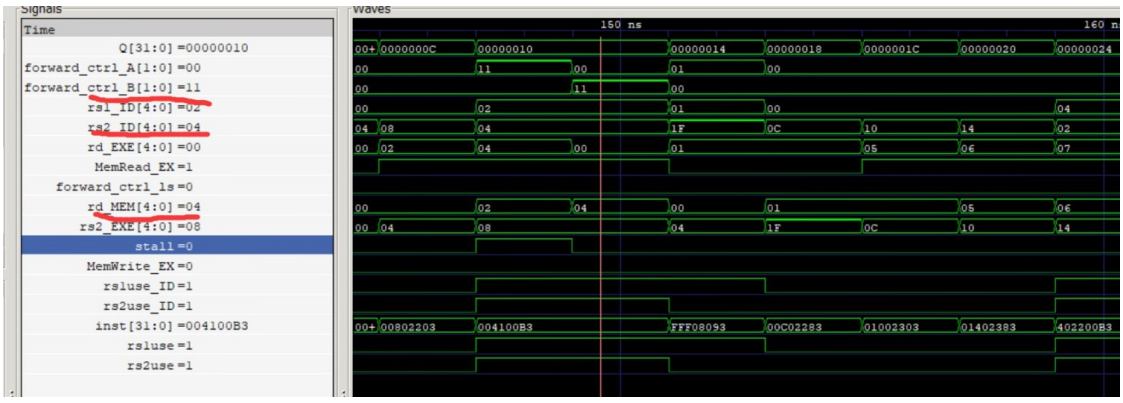
该方法在相邻周期的不涉及 load 的 RAW 数据冒险时，在捕获到 ID 和 EX 段数据冲突时便令转发多路选择器的选择端为 01，从而将 EX 段的 ALU 运算结果转发到 ID 段。



- 此外，所有距离两个周期的 RAW 数据冲突的 stall 都从一个周期减少到了零。该方法在 ID 和 MEM 段发生数据冲突的时候进行转发，在 MEM 阶段指令是 load 时，将转发多选器选择端置为 `11`，转发内存读取的数据；在 MEM 阶段指令时其他写寄存器指令时置为 `10` 转发 ALU 运算结果。

```
MUX4T1_32 mux_forward_A(.I0(rs1_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
    .s(forward_ctrl_A),.o(rs1_data_ID));

MUX4T1_32 mux_forward_B(.I0(rs2_data_reg),.I1(ALUout_EXE),.I2(ALUout_MEM),.I3(Datain_MEM),
    .s(forward_ctrl_B),.o(rs2_data_ID));
```

```
// forward_ctrl_A
always @(*) begin
    // need rs1
    if (rs1use_ID) begin
        // read after write (no load)
        if (rs1_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
            forward_ctrl_A = 2'b01;
        end else if (rs1_ID == rd_MEM && RegWrite_MEM) begin
            // MEM is load
            if (MemRead_MEM) begin
                forward_ctrl_A = 2'b11;
            end else forward_ctrl_A = 2'b10; // MEM is write other than load
        end else forward_ctrl_A = 2'b00;
    end
    else forward_ctrl_A = 2'b00;
end

// forward_ctrl_B
always @(*) begin
    // not save and need rs2
    if (rs2use_ID && ~MemWrite_ID) begin
        if (rs2_ID == rd_EXE && RegWrite_EX && ~MemRead_EX) begin
            // write and not load
            forward_ctrl_B = 2'b01;
        end else if (rs2_ID == rd_MEM && RegWrite_MEM) begin
            // MEM is load
            if (MemRead_MEM) begin
                forward_ctrl_B = 2'b11;
            end else forward_ctrl_B = 2'b10; // MEM is write other than load
        end else forward_ctrl_B = 2'b00;
    end
    else forward_ctrl_B = 2'b00;
end
```



###

有没有办法避免**注意事项**中提到的由 **Load** 所导致得需要额外 **stall** 一个周期或者两个周期这样的情况，即有没有办法做到只用 **forwarding** 解决 **data hazard**，不用额外的 **stall** 来解决 **data hazard**。如果有，请说明方法和利弊；如果没有，请说明理由。

转发延后到**EX**段就可以避免**stall**，代价是必须要在**EX**段才可以得到跳转结果，会使得**Branch**指令需要延迟两周期。