计算机组成与设计
实验报告

课程名称：	计算机组成与设计
姓 名:	王晓宇
学 院:	计算机科学与技术学院
专 业:	计算机科学与技术
指导教师:	刘海风
报告日期:	2024年 5 月 19 日

Lab5 实现解决冲突的五级流水线 CPU

Lab5 实现解决冲突的五级流水线 CPU

课程名称： 计算机组成与设计 实验类型： 综合

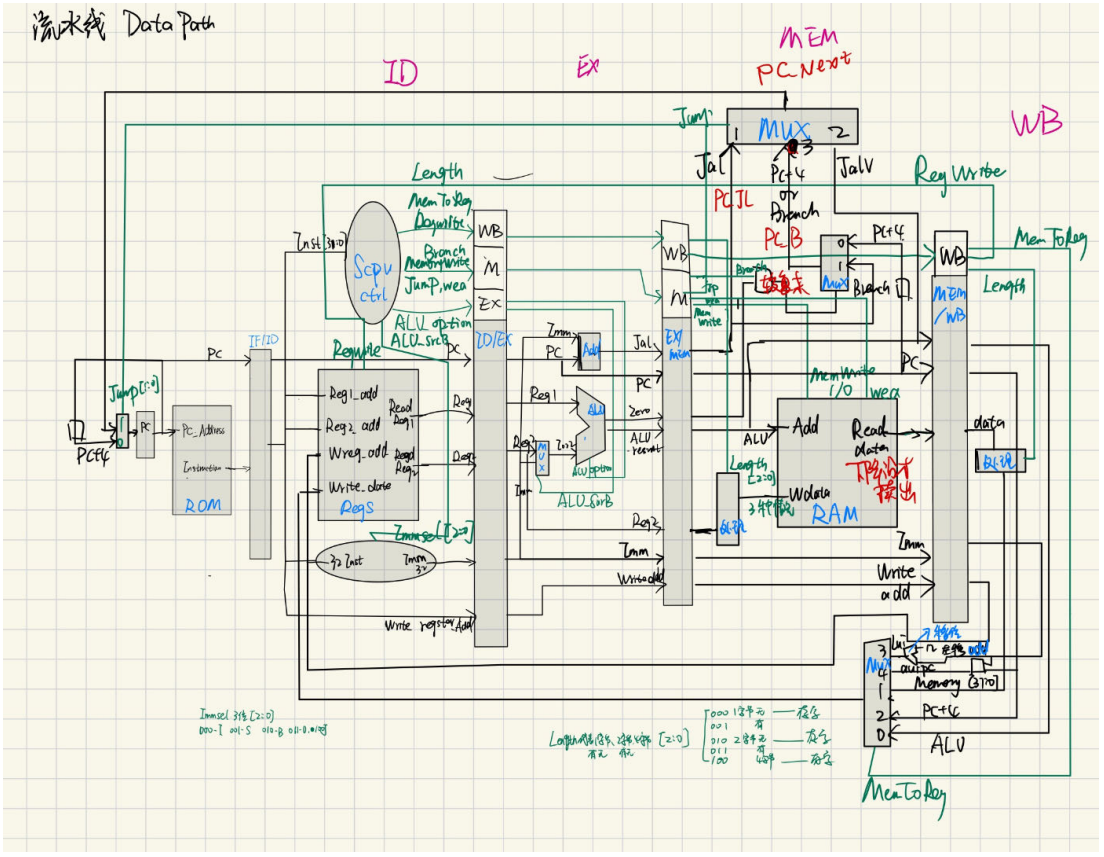
实验项目名称： Lab5 实现解决冲突的五级流水线 CPU

学生姓名： 王晓宇 学号： 3220104364 同组学生姓名： 无

实验地点： 紫金港东四509室 实验日期： 2024 年 5 月 20 日

1 模块功能与仿真介绍

1.1 DataPath示意图



本次实验基于 Lab4-3的工程修改，大部分控制信号的含义并无变化，只是在跳转指令处控制信号Jump[1:0]==2'b11时意味着PC指令为Branch跳转。

部分控制信号的含义在图中标出，其余信号含义在SCPU_Ctrl_W.v模块中给出

1.2 SCPU_W_1.v模块

由于在本次流水线工程中，SCPU_W_1.v实现以DataPath_W.v为主框架,意味着把SCPU_ctrl_W.v作为实例置于DataPath中.

```
module SCPU_W_1(
  input wire clk,
  input wire rst,
  input wire MIO_ready,
  input wire [31:0]inst_in,
  input wire [31:0]Data_in,
  output wire CPU_MIO,
  output wire MemRW,
  output wire [31:0]PC_out,
  output wire [31:0]Data_out,
  output wire [31:0]Addr_out,
  output wire [3:0]wea,
  output wire [31:0] Reg00,
  .......
  output wire[31:0] Reg29,
  output wire[31:0] Reg30,
  output wire[31:0] Reg31
);

  DataPath_W DataPath1(
    .clk(clk),
    .rst(rst),
    .OPcode(inst_in[6:2]),
    .Fun3(inst_in[14:12]),
    .Fun7(inst_in[30]),
    .MIO_ready(MIO_ready),
    .inst_field(inst_in),
    .Data_in(Data_in),
    .EX_MEM_MemRW(MemRW),
    .PC_out(PC_out),
    .Data_out(Data_out),
    .EX_MEM_ALU_out(Addr_out),
```

```
    .wea(wea),
    .Reg00(Reg00),
    .Reg01(Reg01),
    //......
    .Reg31(Reg31)
    );
endmodule
```

1.3 SCPU_ctrl_W.v模块

本模块是CPU初始控制信号的生成模块：

输出信号及其含义：

1. **ImmSel[2:0]**

   立即数选择信号，决定ALU操作中使用的立即数类型。

   - 3'b000:表示I型指令对应立即数。
   - 3'b001:表示S型指令对应立即数。
   - 3'b010:表示B型指令对应立即数。
   - 3'b011:表示U型指令对应立即数。
   - 3'b100:表示J型指令对应立即数。

2. **ALUSrc_B**

   ALU第二个操作数来源选择。

   - 1'b0: 第二个操作数来自寄存器。
   - 1'b1: 第二个操作数来自立即数。

3. **MemtoReg[2:0]**

   控制从内存到寄存器的数据传输。

   - 3'b000: ALU运算结果的数据传输。
   - 3'b001: RAM取出值的数据传输。
   - 3'b010: 表示PC+4。
   - 3'b011: lui指令立即数传输。
   - 3'b100: auipc指令数据传输。

4. **Jump[1:0]**

控制跳转操作。

- `2'b00`: 不执行跳转。
- `2'b01`: 执行Jal操作。
- `2'b10`: 执行Jalr无条件跳转。
- `2'b11'`: 执行Branch分支条件跳转。

5. **Branch_Beq, Branch_Bne, Branch_Blt, Branch_Bltu, Branch_Bge, Branch_Bgeu**

   分支控制信号，根据比较结果决定是否跳转。

   - `1'b0`: 不执行分支。
   - `1'b1`: 根据条件执行分支。

6. **Length[2:0]**

   Store或者Load操作的数据长度。

   - `3'b000`: 无符号单字节数据。
   - `3'b001`: 有符号单字节数据。
   - `3'b010`: 无符号双字节数据。
   - `3'b011`: 有符号双字节数据。
   - `3'b100`: 字长度数据。

7. **RegWrite**

   寄存器写入使能。

   - `1'b0`: 禁止写入寄存器。
   - `1'b1`: 允许写入寄存器。

8. **MemRW**

   内存读写控制。

   - `1'b0`: 不执行内存读写操作。
   - `1'b1`: 允许执行内存读写操作。

9. **ALU_Control[3:0]**

   ALU操作控制。

   - `4'b0000`: 执行加操作。
   - `4'b0001`: 执行减操作。
   - `4'b0010`: 执行左移操作。

- `4'b0011`: 执行比较操作。
- `4'b0100`: 执行无符号比较操作。
- `4'b0101`: 执行逻辑异或操作。
- `4'b0111`: 执行右移操作。
- `4'b0110`: 执行算术右移操作。
- `4'b1000`: 执行逻辑或操作。
- `4'b1001`: 执行逻辑与操作。

1.4 DataPath_W.v模块

该模块设计思路为上文提到的DataPath图，部分模块有特殊作用将在后文提到。

该模块使用Forwarding处理**数据冲突**，而且可以不使用stall、延长时钟周期的情况下处理load_use冲突；使用branch prediction（分支预测）中的predict-not-taken（预测分支总是不发生）处理**结构冒险**。将在稍后分别介绍两种冒险对应新建的实例。

1.4.1 基本思路

现介绍模块大体思路：

参照流水线CPU的原理图，我们引入三个流水线寄存器来存储上一步运算结果和控制信号，并利用其进行下一流程的运算。

1. 初始控制信号由PC信号和SCPU_Ctrl模块形成，之后通过两个时钟周期传导到`Reg_IF_ID`、`Reg_ID_EX`模块，其中rst处信号为解决结构冒险的处理，之后会提到。可以看到寄存器保留了前一步EX、MEM、WB控制信号和寄存器值、立即数值、寄存器地址（为了解决数据冒险存储）、写入寄存器地址（传递到WB阶段使用）。

```
Reg_IF_ID IF_ID(
  .clk(clk),
  .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
  .PC(PC_out),
  .inst_field(inst_field),
  .IF_ID_PC(IF_ID_PC),
  .IF_ID_inst_field(IF_ID_inst_field),
  .IF_ID_Fun3(IF_ID_Fun3),
  .IF_ID_OPcode(IF_ID_OPcode),
```

```
  .IF_ID_Fun7(IF_ID_Fun7)
);

//------------------//
Reg_ID_EX ID_EX(
  .clk(clk),
  .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
  .PC(IF_ID_PC),
  .Rs1_addr(IF_ID_inst_field[19:15]),
  .Rs2_addr(IF_ID_inst_field[24:20]),
  .Rs1_data(Rs1_data),
  .Rs2_data(Rs2_data),
  .Imm(Imm_out),
  .Wt_addr(IF_ID_inst_field[11:7]),
  .Length(Length),
  .MemtoReg(MemtoReg),
  .RegWrite(RegWrite),
  .Branch_Beq(Branch_Beq),
  .Branch_Bne(Branch_Bne),
  .Branch_Bge(Branch_Bge),
  .Branch_Blt(Branch_Blt),
  .Branch_Bgeu(Branch_Bgeu),
  .Branch_Bltu(Branch_Bltu),
  .MemRW(MemRW),
  .Jump(Jump),
  .wea(wea),
  .ALU_Control(ALU_Control),
  .ALUSrc_B(ALUSrc_B),
  .ID_EX_PC(ID_EX_PC),
  .ID_EX_Rs1_data(ID_EX_Rs1_data),
  .ID_EX_Rs2_data(ID_EX_Rs2_data),
  .ID_EX_Imm(ID_EX_Imm),
  .ID_EX_Length(ID_EX_Length),
  .ID_EX_MemtoReg(ID_EX_MemtoReg),
  .ID_EX_RegWrite(ID_EX_RegWrite),
  .ID_EX_Branch_Beq(ID_EX_Branch_Beq),
```

```
    .ID_EX_Branch_Bne(ID_EX_Branch_Bne),
    .ID_EX_Branch_Bge(ID_EX_Branch_Bge),
    .ID_EX_Branch_Blt(ID_EX_Branch_Blt),
    .ID_EX_Branch_Bgeu(ID_EX_Branch_Bgeu),
    .ID_EX_Branch_Bltu(ID_EX_Branch_Bltu),
    .ID_EX_MemRW(ID_EX_MemRW),
    .ID_EX_Jump(ID_EX_Jump),
    .ID_EX_wea(ID_EX_wea),
    .ID_EX_ALU_Control(ID_EX_ALU_Control),
    .ID_EX_ALUSrc_B(ID_EX_ALUSrc_B),
    .ID_EX_Rs1_addr(ID_EX_Rs1_addr),
    .ID_EX_Rs2_addr(ID_EX_Rs2_addr),
    .ID_EX_Wt_addr(ID_EX_Wt_addr)
);
```

2. EX阶段，利用Reg_ID_EX寄存器信号进行ALU运算，后将WB、MEM信号存入Reg_EX_MEM模块。可以看到寄存器保留了前一步MEM、WB控制信号和寄存器值、立即数值、寄存器rs2地址（为了解决RAM处数据冒险存储）、写入寄存器地址（传递到WB阶段使用）。

```
Reg_EX_MEM EX_MEM(
  .clk(clk),
  .rst(rst),
  .PC(ID_EX_PC),
  .PC_Jal(PC_Jal),
  .ALU_out(EX_ALU_out),
  .zero(zero),
  .Rs2_data(ID_EX_Rs2_data),
  .Rs2_addr(ID_EX_Rs2_addr),
  .Imm(ID_EX_Imm),
  .Wt_addr(ID_EX_Wt_addr),
  .Length(ID_EX_Length),
  .MemtoReg(ID_EX_MemtoReg),
  .RegWrite(ID_EX_RegWrite),
  .Branch_Beq(ID_EX_Branch_Beq),
  .Branch_Bne(ID_EX_Branch_Bne),
  .Branch_Bge(ID_EX_Branch_Bge),
```

```
  .Branch_Blt(ID_EX_Branch_Blt),
  .Branch_Bgeu(ID_EX_Branch_Bgeu),
  .Branch_Bltu(ID_EX_Branch_Bltu),
  .MemRW(ID_EX_MemRW),
  .Jump(ID_EX_Jump),
  .wea(ID_EX_wea),
  .EX_MEM_PC(EX_MEM_PC),
  .EX_MEM_PC_Jal(EX_MEM_PC_Jal),
  .EX_MEM_ALU_out(EX_MEM_ALU_out),
  .EX_MEM_zero(EX_MEM_zero),
  .EX_MEM_Rs2_data(EX_MEM_Rs2_data),
  .EX_MEM_Rs2_addr(EX_MEM_Rs2_addr),
  .EX_MEM_Imm(EX_MEM_Imm),
  .EX_MEM_Wt_addr(EX_MEM_Wt_addr),
  .EX_MEM_Length(EX_MEM_Length),
  .EX_MEM_MemtoReg(EX_MEM_MemtoReg),
  .EX_MEM_RegWrite(EX_MEM_RegWrite),
  .EX_MEM_Branch_Beq(EX_MEM_Branch_Beq),
  .EX_MEM_Branch_Bne(EX_MEM_Branch_Bne),
  .EX_MEM_Branch_Bge(EX_MEM_Branch_Bge),
  .EX_MEM_Branch_Blt(EX_MEM_Branch_Blt),
  .EX_MEM_Branch_Bgeu(EX_MEM_Branch_Bgeu),
  .EX_MEM_Branch_Bltu(EX_MEM_Branch_Bltu),
  .EX_MEM_MemRW(EX_MEM_MemRW),
  .EX_MEM_Jump(EX_MEM_Jump),
  .EX_MEM_wea(EX_MEM_wea)
);
```

3. MEM阶段，利用`Reg_EX_MEM`寄存器信号进行MEM运算，后将WB、MEM信号存入`Reg_MEM_WB`模块。可以看到寄存器保留了前一步WB控制信号和寄存器值、立即数值、写入寄存器地址（传递到WB阶段使用）。输出信号新增`PC_next`信号表示之后跳转地址。

```
Reg_MEM_WB MEM_WB(
  .clk(clk),
  .rst(rst),
  .PC(EX_MEM_PC),
```

```
  .ALU_out(EX_MEM_ALU_out),
  .Data_in(Data_in),
  .Imm(EX_MEM_Imm),
  .Wt_addr(EX_MEM_Wt_addr),
  .Length(EX_MEM_Length),
  .MemtoReg(EX_MEM_MemtoReg),
  .RegWrite(EX_MEM_RegWrite),
  .MEM_WB_PC(MEM_WB_PC),
  .MEM_WB_ALU_out(MEM_WB_ALU_out),
  .MEM_WB_Data_in(MEM_WB_Data_in),
  .MEM_WB_Imm(MEM_WB_Imm),
  .MEM_WB_Wt_addr(MEM_WB_Wt_addr),
  .MEM_WB_Length(MEM_WB_Length),
  .MEM_WB_MemtoReg(MEM_WB_MemtoReg),
  .MEM_WB_RegWrite(MEM_WB_RegWrite)
);
//---------//
always@(*)begin
  case(EX_MEM_Jump)
    2'b11: PC_next <= PC_Branch;
    2'b01: PC_next <= EX_MEM_PC_Jal;
    2'b10: PC_next <= {EX_MEM_ALU_out[31:1],1'b0};
  endcase
end
```

4. WB阶段，利用`Reg_MEM_WB`寄存器信号进行写入操作。利用WB控制信号和寄存器值、立即数值、写入寄存器地址（传递到Reg模块作为写入寄存器地址）来进行寄存器写入。其中写入寄存器的值由`MEM_WB_Length`、`MEM_WB_MemtoReg`共同决定。

```
Regs U1(
  .clk(clk),
  .rst(rst),
  .Rs1_addr(inst_field[19:15]),
  .Rs2_addr(inst_field[24:20]),
  .Wt_addr(MEM_WB_Wt_addr),
  .Wt_data(Wt_data),
```

```
  .RegWrite(MEM_WB_RegWrite),
  .Rs1_data(Rs1_data),
  .Rs2_data(Rs2_data),
);
//----------//
always@(*)begin
  case(MEM_WB_Length)//memtoreg
    3'b000:
    case(MEM_WB_ALU_out[1:0])
      2'b00:Tem_mem <= {24'b0,MEM_WB_Data_in[7:0]};
      2'b01:Tem_mem <= {24'b0,MEM_WB_Data_in[15:8]};
      2'b10:Tem_mem <= {24'b0,MEM_WB_Data_in[23:16]};
      2'b11:Tem_mem <= {24'b0,MEM_WB_Data_in[31:24]};
    endcase
    3'b001:
    case(MEM_WB_ALU_out[1:0])
      2'b00:Tem_mem <= {{24{MEM_WB_Data_in[7]}},MEM_WB_Data_in[7:0]};
      2'b01:Tem_mem <= {{24{MEM_WB_Data_in[15]}},MEM_WB_Data_in[15:8]};
      2'b10:Tem_mem <= {{24{MEM_WB_Data_in[23]}},MEM_WB_Data_in[23:16]};
      2'b11:Tem_mem <= {{24{MEM_WB_Data_in[31]}},MEM_WB_Data_in[31:24]};
    endcase
    3'b010:
    case(MEM_WB_ALU_out[1:0])
      2'b00:Tem_mem <= {16'b0,MEM_WB_Data_in[15:0]};
      2'b01:Tem_mem <= {16'b0,MEM_WB_Data_in[23:8]};
      2'b10:Tem_mem <= {16'b0,MEM_WB_Data_in[31:16]};
    endcase
    3'b011:
    case(MEM_WB_ALU_out[1:0])
      2'b00:Tem_mem <= {{16{MEM_WB_Data_in[15]}},MEM_WB_Data_in[15:0]};
```

```
38        2'b01:Tem_mem <=
{{16{MEM_WB_Data_in[23]}},MEM_WB_Data_in[23:8]};
39        2'b10:Tem_mem <=
{{16{MEM_WB_Data_in[31]}},MEM_WB_Data_in[31:16]};
40      endcase
41      3'b100:
42       Tem_mem <= MEM_WB_Data_in;
43    endcase
44    case(MEM_WB_MemtoReg)
45      3'b000: Wt_data <= MEM_WB_ALU_out;
46      3'b001: Wt_data <= reg_mem;
47      3'b010: Wt_data <= MEM_WB_PC+32'd4;
48      3'b011: Wt_data <= MEM_WB_Imm<<12;
49      3'b100: Wt_data <= MEM_WB_PC+(MEM_WB_Imm<<12);
50    endcase
51  end
```

1.4.2 Forwarding 处理数据冲突

数据冲突指在ALU阶段或是memory写入阶段读取的值并不是之前指令完整操作结束后的最新值。这里将用Frowarding操作将操作值更新，此处修改可以做到将load_use冲突解决。

1.4.2.1 冲突检测模块

这段代码定义了一个名为`Forwarding_Ctrl`的Verilog模块，其功能是实现寄存器前递（Forwarding）控制。在CPU的流水线设计中，前递是一种减少数据相关（Data Hazard）的技术，它允许执行阶段（EX）直接使用未写回（WB）寄存器的数据，而不是等待数据从寄存器文件中写回。

输入信号：

- `ID_EX_Rs1_addr[4:0]`: 取值1的寄存器地址，存在于指令解码（ID）阶段和执行（EX）阶段。
- `ID_EX_Rs2_addr[4:0]`: 取值2的寄存器地址，同样存在于ID和EX阶段。
- `EX_MEM_Wt_addr[4:0]`: 执行（EX）阶段写回（Wt）的寄存器地址。
- `MEM_WB_Wt_addr[4:0]`: 存储（MEM）和写回（WB）阶段写回的寄存器地址。

- `EX_MEM_Rs2_addr[4:0]`: 执行阶段取值2的寄存器地址。
- `EX_MEM_RegWrite`: 执行阶段的寄存器写回使能信号。
- `MEM_WB_RegWrite`: 存储和写回阶段的寄存器写回使能信号。

输出信号：

- `For_A[1:0]`: 前递控制信号A，用于确定执行阶段的Rs1来源。
- `For_B[1:0]`: 前递控制信号B，用于确定执行阶段的Rs2来源。
- `For_M`: 前递控制信号M，用于确定存储阶段的Rs2来源。

前递控制逻辑：

1. **For_A信号生成条件：**
    - 如果`EX_MEM_RegWrite`为真，且`ID_EX_Rs1_addr`等于`EX_MEM_Wt_addr`，并且`EX_MEM_Wt_addr`不是无效地址（非`5'b0`），则`For_A`设置为`2'b01`，表示Rs1应从前递路径获取数据。
    - 否则，如果`MEM_WB_RegWrite`为真，且`ID_EX_Rs1_addr`等于`MEM_WB_Wt_addr`，并且`MEM_WB_Wt_addr`不是无效地址，则`For_A`设置为`2'b10`，表示Rs1应从上一个写回的寄存器获取数据。
    - 其他情况下，`For_A`设置为`2'b00`，表示不使用前递。
2. **For_B信号生成条件：**
    - 如果`EX_MEM_RegWrite`为真，且`ID_EX_Rs2_addr`等于`EX_MEM_Wt_addr`，并且`EX_MEM_Wt_addr`不是无效地址，则`For_B`设置为`2'b01`，表示Rs2应从前递路径获取数据。
    - 否则，如果`MEM_WB_RegWrite`为真，且`ID_EX_Rs2_addr`等于`MEM_WB_Wt_addr`，并且`MEM_WB_Wt_addr`不是无效地址，则`For_B`设置为`2'b10`，表示Rs2应从上一个写回的寄存器获取数据。
    - 其他情况下，`For_B`设置为`2'b00`，表示不使用前递。
3. **For_M信号生成条件：**
    - 如果`MEM_WB_RegWrite`为真，且`MEM_WB_Wt_addr`等于`EX_MEM_Rs2_addr`，则`For_M`设置为`1'b1`，表示MEM阶段的Rs2应从上一个写回的寄存器获取数据。
    - 其他情况下，`For_M`设置为`1'b0`，表示不使用前递。

```
module Forwarding_Ctrl(
    input [4:0]ID_EX_Rs1_addr,
    input [4:0]ID_EX_Rs2_addr,
    input [4:0]EX_MEM_Wt_addr,
    input [4:0]MEM_WB_Wt_addr,
    input [4:0]EX_MEM_Rs2_addr,
    input EX_MEM_RegWrite,
    input MEM_WB_RegWrite,
    output reg [1:0]For_A,
    output reg [1:0]For_B,
    output reg For_M
    );
    always@(*)begin
        if(EX_MEM_RegWrite && ID_EX_Rs1_addr == EX_MEM_Wt_addr && EX_MEM_Wt_addr != 5'b0)begin
            For_A <= 2'b01;
        end
        else if(MEM_WB_RegWrite && ID_EX_Rs1_addr == MEM_WB_Wt_addr && MEM_WB_Wt_addr != 5'b0)begin
            For_A <= 2'b10;
        end
        else begin
            For_A <= 2'b00;
        end
        if(EX_MEM_RegWrite &&ID_EX_Rs2_addr == EX_MEM_Wt_addr && EX_MEM_Wt_addr != 5'b0)begin
            For_B <= 2'b01;
        end
        else if(MEM_WB_RegWrite && ID_EX_Rs2_addr == MEM_WB_Wt_addr && MEM_WB_Wt_addr != 5'b0)begin
            For_B <= 2'b10;
        end
        else begin
            For_B <= 2'b00;
        end
```

```
        if(MEM_WB_RegWrite && MEM_WB_Wt_addr==EX_MEM_Rs2_addr) begin
            For_M <= 1'b1;
        end
        else begin
            For_M <= 1'b0;
        end
    end
endmodule
```

1.4.2.2 解决Regs寄存器处数据冲突

通过Forward_A、Forward_B的信号的控制来决定ALU的两个运算单元。寄存器的算子来源有三个：前一步的流水线寄存器值、后一步MEM阶段将在下一步WB阶段的待写入寄存器的值、WB阶段的待写入寄存器的值。

第二个来源通过Tem_RegWrite模块生成，输出结果和WB阶段流程基本一致

```
Forwarding_Ctrl forward_strl(
  .ID_EX_Rs1_addr(ID_EX_Rs1_addr),
  .ID_EX_Rs2_addr(ID_EX_Rs2_addr),
  .EX_MEM_Wt_addr(EX_MEM_Wt_addr),
  .MEM_WB_Wt_addr(MEM_WB_Wt_addr),
  .EX_MEM_RegWrite(EX_MEM_RegWrite),
  .MEM_WB_RegWrite(MEM_WB_RegWrite),
  .EX_MEM_Rs2_addr(EX_MEM_Rs2_addr),
  .For_A(Forward_A),
  .For_B(Forward_B),
  .For_M(For_M)
);
//----------//
Tem_RegWrite Temreg(
  .MemtoReg(EX_MEM_MemtoReg),
  .Length(EX_MEM_Length),
  .Data_in(Data_in),
  .ALU_out(EX_MEM_ALU_out),
  .Imm(EX_MEM_Imm),
  .PC_out(EX_MEM_PC),
```

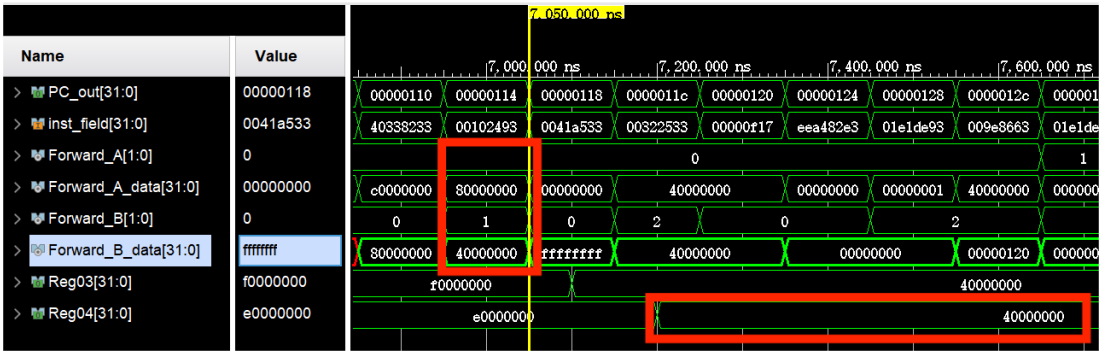
```
  .Tem_RegWrite(Tem_RegWrite)
);
//----------//
wire [31:0] Ior2;
wire [31:0] Forward_A_data,Forward_B_data;
assign Forward_A_data = (Forward_A==2'b00)?ID_EX_Rs1_data:
((Forward_A==2'b01)?Tem_RegWrite:((Forward_A==2'b10)?
Wt_data:32'b0));
assign Forward_B_data = (Forward_B==2'b00)?ID_EX_Rs2_data:
((Forward_B==2'b01)?Tem_RegWrite:((Forward_B==2'b10)?
Wt_data:32'b0));
assign Ior2 = ID_EX_ALUSrc_B? ID_EX_Imm: Forward_B_data;
//----------//
ALU U3(
  .A(Forward_A_data),
  .B(Ior2),
  .ALU_operation(ID_EX_ALU_Control),
  .res(EX_ALU_out),
  .zero(zero)
);
```

- 仿真截图

部分汇编码

```
pass_1:
    nop
    li    x31, 2
    sub   x3, x6, x7        # x3=40000000
    sub   x4, x7, x3        # x4=40000000
```

这里的第五行使用寄存器x3作为运算算子，而x3的结果上一条刚刚算出来，这里的For_B应该为1，最后结果x4正确结果为0x40000000



1.4.2.3 解决RAM处数据存储数据冲突

通过Forward_M的信号的控制来决定RAM的存储值。

```
Forwarding_Ctrl forward_strl(
  .ID_EX_Rs1_addr(ID_EX_Rs1_addr),
  .ID_EX_Rs2_addr(ID_EX_Rs2_addr),
  .EX_MEM_Wt_addr(EX_MEM_Wt_addr),
  .MEM_WB_Wt_addr(MEM_WB_Wt_addr),
  .EX_MEM_RegWrite(EX_MEM_RegWrite),
  .MEM_WB_RegWrite(MEM_WB_RegWrite),
  .EX_MEM_Rs2_addr(EX_MEM_Rs2_addr),
  .For_A(Forward_A),
  .For_B(Forward_B),
  .For_M(For_M)
);
//-------------------//
assign Data_out = Tem_Data;
assign wea = Tem_wea;
assign reg_Data_out = For_M?Wt_data:EX_MEM_Rs2_data;
always@(*)begin
  case(EX_MEM_Length)//RAM
    3'b000:begin
      case(EX_MEM_ALU_out[1:0])
        2'b00: begin
          Tem_wea <= 4'b0001;
          Tem_Data <= {24'b0,reg_Data_out[7:0]};
        end
        2'b01: begin
```

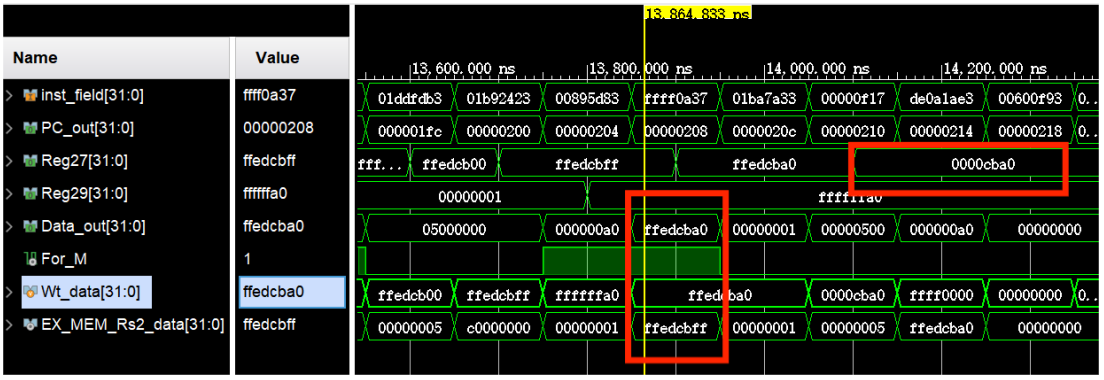
```
          Tem_wea <= 4'b0010;
          Tem_Data <= {16'b0,reg_Data_out[7:0],8'b0};
        end
        2'b10: begin
          Tem_wea <= 4'b0100;
          Tem_Data <= {8'b0,reg_Data_out[7:0],16'b0};
        end
        2'b11: begin
          Tem_wea <= 4'b1000;
          Tem_Data <= {reg_Data_out[7:0],24'b0};
        end
      endcase
    end
    3'b010:begin
      case(EX_MEM_ALU_out[1:0])
        2'b00: begin
          Tem_wea <= 4'b0011;
          Tem_Data <= {16'b0,reg_Data_out[15:0]};
        end
        2'b01: begin
          Tem_wea <= 4'b0110;
          Tem_Data <= {8'b0,reg_Data_out[15:0],8'b0};
        end
        2'b10: begin
          Tem_wea <= 4'b1100;
          Tem_Data <= {reg_Data_out[15:0],16'b0};
        end
      endcase
    end
    3'b100:begin
    Tem_wea <= 4'b1111;
    Tem_Data <= reg_Data_out;
    end
  endcase
end
```

- 仿真截图

部分汇编码

```
and   x27, x27, x29    # x27=FFEDCBA0
sw    x27, 8(x18)      # mem[0x28]=FFEDCBA0
lhu   x27, 8(x18)      # x27=0000CBA0
```

首先从x27的旧值在MEM阶段时钟下降沿写入内存，但是此时的x27是上一步还没更新完成的，此时需要将将要写入x27的值提前赋值给内存接口，即`For_M`应该为1'b1，选择`Wt_data`作为写入值。`Wt_data`应该为0xFFEDCBA0而不是之前的0xFFEDCBFF



1.4.3 Branch Prediction处理结构冒险

使用branch prediction（分支预测）中的predict-not-taken（预测分支总是不发生）处理结构冒险。

我们假设分支跳转不会发生，所以动作流程和正常流水线行为一致，只有当分支确实发生时，我们将分支跳转之前的ID、EX阶段的控制信号全部置零即可，即将`Reg_ID_EX`中的控制信号置零即可。有个简单的办法，我们注意到rst信号可以将流水线的寄存器的存储值置零。分析`Jump`的编码含义我们很容易写出当`Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])`时产生跳转。代码如下:

```
assign Branch = (EX_MEM_Branch_Beq&EX_MEM_zero)|(EX_MEM_Branch_Bne&
(~EX_MEM_zero))|(EX_MEM_Branch_Blt&EX_MEM_ALU_out[0])|
(EX_MEM_Branch_Bltu&EX_MEM_ALU_out[0])|(EX_MEM_Branch_Bge&
(~EX_MEM_ALU_out[0]))|(EX_MEM_Branch_Bgeu&(~EX_MEM_ALU_out[0]));
//-------------//
Reg_IF_ID IF_ID(
  .clk(clk),
```

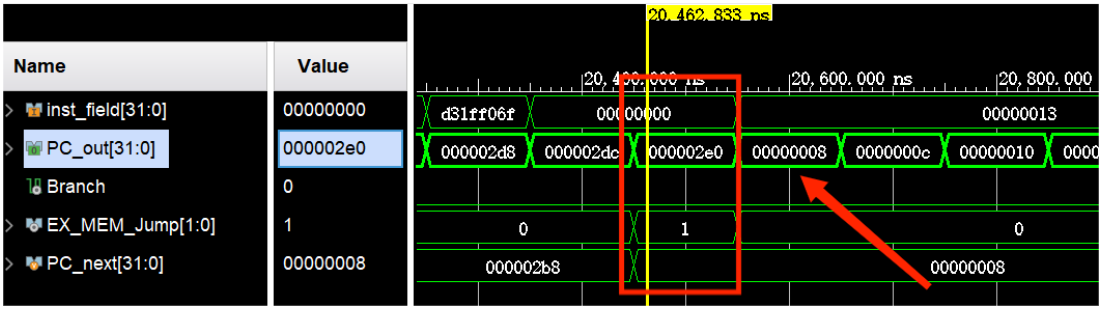
```
  .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
//省略其他端口
);

//---------//
Reg_ID_EX ID_EX(
  .clk(clk),
  .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
//省略其他端口
);
```

- 仿真截图

部分汇编码：

```
    li    x31, 0x666
    j     dummy
```



1.4.4 模块源代码

1.4.4.1 CSSTE.v

```
module CSSTE(
    input         clk_100mhz,
    input         RSTN,
    input  [3:0]  BTN_y,
    input  [15:0] SW,
    output [3:0]  Blue,
    output [3:0]  Green,
    output [3:0]  Red,
    output        HSYNC,
    output        VSYNC,
```

```
    output [15:0] LED_out,
    output [7:0] AN,
    output wire tx,
    output [7:0] segment
);
wire [3:0]U9_btnok;
wire [15:0]U9_swok;
wire U9_rst;
wire [31:0]U8_clkdiv;
wire U8_clkcpu;
wire U10_cnt0,U10_cnt1,U10_cnt2;
wire [31:0]U10_cntout;
wire [31:0]U2_spo;
wire U1_memrw;
wire [31:0]U1_add;
wire [31:0]U1_data;
wire [31:0]U1_pc;
wire [31:0]U3_douta;
wire [31:0]U4_cpu;
wire [31:0]U4_ramin;
wire [9:0]U4_ramadd;
wire U4_datawe,U4_gpiofwe,U4_gpioewe,U4_cntwe;
wire [31:0]U4_pin;
wire [7:0]U5_LEout;
wire [7:0]U5_ptout;
wire [31:0]U5_dispnum;
wire [1:0]U7_cntset;
SAnti_jitter U9(.clk(clk_100mhz), .RSTN(RSTN), .Key_y(BTN_y),
.SW(SW),.BTN_OK(U9_btnok), .SW_OK(U9_swok), .rst(U9_rst));

clk_div U8(.clk(clk_100mhz), .rst(U9_rst), .SW2(U9_swok[2]),
.SW8(U9_swok[8]), .STEP(U9_swok[10]), .clkdiv(U8_clkdiv),
.Clk_CPU(U8_clkcpu));

```

```
42 Counter_x U10(.clk(~U8_clkcpu), .rst(U9_rst), .clk0(U8_clkdiv[6]),
   .clk1(U8_clkdiv[9]), .clk2(U8_clkdiv[11]), .counter_we(U4_cntwe),
   .counter_val(U4_pin),.counter_ch(U7_cntset),
   .counter0_OUT(U10_cnt0), .counter1_OUT(U10_cnt1),
   .counter2_OUT(U10_cnt2), .counter_out(U10_cntout));
43
44 ROM_1 U2(
45     .a(U1_pc[11:2]),
46     .spo(U2_spo)
47     );
48
49 wire [3:0]wea;
50 wire        CPU_MIO;
51 wire
   [31:0]Reg00,Reg01,Reg02,Reg03,Reg04,Reg05,Reg06,Reg07,Reg08,Reg09,
   Reg10,Reg11,Reg12,Reg13,Reg14,Reg15,Reg16,Reg17,Reg18,Reg19,Reg20,
   Reg21,Reg22,Reg23,Reg24,Reg25,Reg26,Reg27,Reg28,Reg29,Reg30,Reg31;
52 SCPU_W_1 U1(
53     .clk(U8_clkcpu),
54     .rst(U9_rst),
55     .inst_in(U2_spo),
56     .Data_in(U3_douta),
57     .MIO_ready(CPU_MIO),
58     .MemRW(U1_memrw),
59     .CPU_MIO(CPU_MIO),
60     .PC_out(U1_pc),
61     .Data_out(U1_data),
62     .Addr_out(U1_add),
63 .wea(wea),
64 .Reg00(Reg00),
65 .Reg01(Reg01),
66 .Reg02(Reg02),
67 .Reg03(Reg03),
68 .Reg04(Reg04),
```

```
69  .Reg05(Reg05),.Reg06(Reg06),.Reg07(Reg07),.Reg08(Reg08),.Reg09(Reg
    09),.Reg10(Reg10),.Reg11(Reg11),.Reg12(Reg12),.Reg13(Reg13),.Reg14
    (Reg14),.Reg15(Reg15),.Reg16(Reg16),.Reg17(Reg17),.Reg18(Reg18),.R
    eg19(Reg19),.Reg20(Reg20),.Reg21(Reg21),.Reg22(Reg22),.Reg23(Reg23
    ),.Reg24(Reg24),.Reg25(Reg25),.Reg26(Reg26),.Reg27(Reg27),.Reg28(R
    eg28),.Reg29(Reg29),.Reg30(Reg30),.Reg31(Reg31)
70  );
71
72  RAM_1 U3(
73      .clka(~U8_clkcpu),
74      .wea({4{U1_memrw}}&wea),
75      .addra(U1_add[11:2]),
76      .dina(U1_data),
77      .douta(U3_douta));
78
79  MIO_BUS U4(
80      .clk(clk_100mhz),
81      .rst(U9_rst),
82      .BTN(U9_btnok),
83      .SW(U9_swok),
84      .mem_w(U1_memrw),
85      .Cpu_data2bus(U1_data),
86      .addr_bus(U1_add),
87      .ram_data_out(U3_douta),
88      .led_out(LED_out),
89      .counter_out(U10_cntout),
90      .counter0_out(U10_cnt0),
91      .counter1_out(U10_cnt1),
92      .counter2_out(U10_cnt2),
93      .Cpu_data4bus(U4_cpu),
94      .ram_data_in(U4_ramin),
95      .ram_addr(U4_ramadd),
96      .data_ram_we(U4_datawe),
97      .GPIOf0000000_we(U4_gpiofwe),
98      .GPIOe0000000_we(U4_gpioewe),
99      .counter_we(U4_cntwe),
```

```
100     .Peripheral_in(U4_pin));
101
102 Multi_8CH32 U5(.clk(~U8_clkcpu), .rst(U9_rst), .EN(U4_gpioewe),
    .Test(U9_swok[7:5]), .point_in({U8_clkdiv,U8_clkdiv}),
    .LES(64'b0), .Data0(U4_pin), .data1({2'b0,U1_pc[31:2]}),
    .data2(U2_spo), .data3(U10_cntout), .data4(U1_add),
    .data5(U1_data), .data6(U4_cpu), .data7(U1_pc),
    .point_out(U5_ptout), .LE_out(U5_LEout), .Disp_num(U5_dispnum));
103
104 Seg7_Dev_0
    U6(.disp_num(U5_dispnum),.point(U5_ptout),.les(U5_LEout),.scan(U8_
    clkdiv[18:16]),.AN(AN),.segment(segment));
105
106 SPIO U7(.clk(~U8_clkcpu), .rst(U9_rst), .Start(U8_clkdiv[20]),
    .EN(U4_gpiofwe), .P_Data(U4_pin), .counter_set(U7_cntset),
    .LED_out(LED_out));
107
108 UART uart_inst(
109     .clk(clk_100mhz),
110     .rst(SW[0]),
111     .tx(tx),
112     .pc(U1_pc),
113     .inst(U2_spo),
114     .x0(Reg00),
115
    .ra(Reg01),.sp(Reg02),.gp(Reg03),.tp(Reg04),.t0(Reg05),.t1(Reg06),
    .t2(Reg07),.s0(Reg08),.s1(Reg09),.a0(Reg10),.a1(Reg11),.a2(Reg12),
    .a3(Reg13),
116
    .a4(Reg14),.a5(Reg15),.a6(Reg16),.a7(Reg17),.s2(Reg18),.s3(Reg19),
    .s4(Reg20),.s5(Reg21),.s6(Reg22),.s7(Reg23),.s8(Reg24),.s9(Reg25),
    .s10(Reg26),
117     .s11(Reg27),.t3(Reg28),.t4(Reg29),.t5(Reg30),.t6(Reg31)
118     );
119 endmodule
```

1.4.4.2 SCPU_W_1.v

```
module SCPU_W_1(
  input wire clk,
  input wire rst,
  input wire MIO_ready,
  input wire [31:0]inst_in,
  input wire [31:0]Data_in,
  output wire CPU_MIO,
  output wire MemRW,
  output wire [31:0]PC_out,
  output wire [31:0]Data_out,
  output wire [31:0]Addr_out,
  output wire [3:0]wea,
  output wire [31:0] Reg00,
  output wire [31:0] Reg01,
  output wire [31:0] Reg02,
  output wire[31:0] Reg03,
  output wire[31:0] Reg04,
  output wire[31:0] Reg05,
  output wire[31:0] Reg06,
  output wire[31:0] Reg07,
  output wire[31:0] Reg08,
  output wire[31:0] Reg09,
  output wire[31:0] Reg10,
  output wire[31:0] Reg11,
  output wire[31:0] Reg12,
  output wire[31:0] Reg13,
  output wire[31:0] Reg14,
  output wire[31:0] Reg15,
  output wire[31:0] Reg16,
  output wire[31:0] Reg17,
  output wire[31:0] Reg18,
  output wire[31:0] Reg19,
  output wire[31:0] Reg20,
  output wire[31:0] Reg21,
```

```
  output wire[31:0] Reg22,
  output wire[31:0] Reg23,
  output wire[31:0] Reg24,
  output wire[31:0] Reg25,
  output wire[31:0] Reg26,
  output wire[31:0] Reg27,
  output wire[31:0] Reg28,
  output wire[31:0] Reg29,
  output wire[31:0] Reg30,
  output wire[31:0] Reg31
);

  DataPath_W DataPath1(
    .clk(clk),
    .rst(rst),
    .OPcode(inst_in[6:2]),
    .Fun3(inst_in[14:12]),
    .Fun7(inst_in[30]),
    .MIO_ready(MIO_ready),
    .inst_field(inst_in),
    .Data_in(Data_in),
    .EX_MEM_MemRW(MemRW),
    .PC_out(PC_out),
    .Data_out(Data_out),
    .EX_MEM_ALU_out(Addr_out),
    .wea(wea),
    .Reg00(Reg00),
    .Reg01(Reg01),
    .Reg02(Reg02),
    .Reg03(Reg03),
    .Reg04(Reg04),
    .Reg05(Reg05),
    .Reg06(Reg06),
    .Reg07(Reg07),
    .Reg08(Reg08),
    .Reg09(Reg09),
```

```
    .Reg10(Reg10),
    .Reg11(Reg11),
    .Reg12(Reg12),
    .Reg13(Reg13),
    .Reg14(Reg14),
    .Reg15(Reg15),
    .Reg16(Reg16),
    .Reg17(Reg17),
    .Reg18(Reg18),
    .Reg19(Reg19),
    .Reg20(Reg20),
    .Reg21(Reg21),
    .Reg22(Reg22),
    .Reg23(Reg23),
    .Reg24(Reg24),
    .Reg25(Reg25),
    .Reg26(Reg26),
    .Reg27(Reg27),
    .Reg28(Reg28),
    .Reg29(Reg29),
    .Reg30(Reg30),
    .Reg31(Reg31)
    );
endmodule
```

1.4.4.3 SCPU_ctrl_W.v

```
module SCPU_ctrl_W(
  input [4:0]       OPcode,
  input [2:0]       Fun3,
  input             Fun7,
  input             MIO_ready,
  output reg [2:0]  ImmSel,
  output reg        ALUSrc_B,
  output reg [2:0]  MemtoReg,
  output reg [1:0]      Jump,
  output reg        Branch_Beq,
```

```
  output reg        Branch_Bne,
  output reg        Branch_Blt,
  output reg        Branch_Bltu,
  output reg        Branch_Bge,
  output reg        Branch_Bgeu,
  output reg [2:0]  Length,
  output reg        RegWrite,
  output reg        MemRW,
  output reg [3:0]  ALU_Control,
  output reg        CPU_MIO
    );
always@(*)begin
    case(OPcode)
        5'b01100:begin
            ImmSel <= 3'b000;
            ALUSrc_B <= 1'b0;
            MemtoReg <= 3'b000;
            Jump <= 2'b00;
            Branch_Beq <= 1'b0;
            Branch_Bne <= 1'b0;
            Branch_Blt <= 1'b0;
            Branch_Bltu <= 1'b0;
            Branch_Bge <= 1'b0;
            Branch_Bgeu <= 1'b0;
            Length <= 3'b000;
            RegWrite <= 1'b1;
            MemRW <= 1'b0;
            CPU_MIO <= 1'b0;
            case(Fun3)
                3'b000: ALU_Control <= Fun7?4'b0001:4'b0000;
                3'b001: ALU_Control <= 4'b0010;
                3'b010: ALU_Control <= 4'b0011;
                3'b011: ALU_Control <= 4'b0100;
                3'b100: ALU_Control <= 4'b0101;
                3'b101: ALU_Control <= Fun7?4'b0111:4'b0110;
                3'b110: ALU_Control <= 4'b1000;
```

```
47                  3'b111: ALU_Control <= 4'b1001;
48              endcase
49          end
50          5'b00100: begin
51              ImmSel <= 3'b000;
52              ALUSrc_B <= 1'b1;
53              MemtoReg <= 3'b000;
54              Jump <= 2'b00;
55              Branch_Beq <= 1'b0;
56              Branch_Bne <= 1'b0;
57              Branch_Blt <= 1'b0;
58              Branch_Bltu <= 1'b0;
59              Branch_Bge <= 1'b0;
60              Branch_Bgeu <= 1'b0;
61              Length <= 3'b000;
62              RegWrite <= 1'b1;
63              MemRW <= 1'b0;
64              CPU_MIO <= 1'b0;
65              case(Fun3)
66                  3'b000: ALU_Control <= 4'b0000;
67                  3'b001: ALU_Control <= 4'b0010;
68                  3'b010: ALU_Control <= 4'b0011;
69                  3'b011: ALU_Control <= 4'b0100;
70                  3'b100: ALU_Control <= 4'b0101;
71                  3'b101: ALU_Control <= Fun7?4'b0111:4'b0110;
72                  3'b110: ALU_Control <= 4'b1000;
73                  3'b111: ALU_Control <= 4'b1001;
74              endcase
75          end
76          5'b00000:begin
77              ImmSel <= 3'b000;
78              ALUSrc_B <= 1'b1;
79              MemtoReg <= 3'b001;
80              Jump <= 2'b00;
81              Branch_Beq <= 1'b0;
82              Branch_Bne <= 1'b0;
```

```
            Branch_Blt <= 1'b0;
            Branch_Bltu <= 1'b0;
            Branch_Bge <= 1'b0;
            Branch_Bgeu <= 1'b0;
            RegWrite <= 1'b1;
            MemRW <= 1'b0;
            CPU_MIO <= 1'b0;
            ALU_Control <= 4'b0000;
            case(Fun3)
                3'b000: Length <= 3'b001;
                3'b001: Length <= 3'b011;
                3'b010: Length <= 3'b100;
                3'b100: Length <= 3'b000;
                3'b101: Length <= 3'b010;
            endcase
        end
        5'b11001:begin
            ImmSel <= 3'b000;
            ALUSrc_B <= 1'b1;
            MemtoReg <= 3'b010;
            Jump <= 2'b10;
            Branch_Beq <= 1'b0;
            Branch_Bne <= 1'b0;
            Branch_Blt <= 1'b0;
            Branch_Bltu <= 1'b0;
            Branch_Bge <= 1'b0;
            Branch_Bgeu <= 1'b0;
            Length <= 3'b000;
            RegWrite <= 1'b1;
            MemRW <= 1'b0;
            CPU_MIO <= 1'b1;
            ALU_Control <= 4'b0000;
        end
        5'b01000:begin
            ImmSel <= 3'b001;
            ALUSrc_B <= 1'b1;
```

```
119             MemtoReg <= 3'b001;
120             Jump <= 2'b00;
121             Branch_Beq <= 1'b0;
122             Branch_Bne <= 1'b0;
123             Branch_Blt <= 1'b0;
124             Branch_Bltu <= 1'b0;
125             Branch_Bge <= 1'b0;
126             Branch_Bgeu <= 1'b0;
127             RegWrite <= 1'b0;
128             MemRW <= 1'b1;
129             CPU_MIO <= 1'b0;
130             ALU_Control <= 4'b0000;
131             case(Fun3)
132                 3'b000: Length <= 3'b000;
133                 3'b001: Length <= 3'b010;
134                 3'b010: Length <= 3'b100;
135             endcase
136         end
137         5'b11000:begin
138             ImmSel <= 3'b010;
139             ALUSrc_B <= 1'b0;
140             MemtoReg <= 3'b000;
141             Jump <= 2'b11;
142             Branch_Beq <= (Fun3 == 3'b000)?1'b1:1'b0;
143             Branch_Bne <= (Fun3 == 3'b001)?1'b1:1'b0;
144             Branch_Blt <= (Fun3 == 3'b100)?1'b1:1'b0;
145             Branch_Bltu <= (Fun3 == 3'b110)?1'b1:1'b0;
146             Branch_Bge <= (Fun3 == 3'b101)?1'b1:1'b0;
147             Branch_Bgeu <= (Fun3 == 3'b111)?1'b1:1'b0;
148             Length <= 3'b000;
149             RegWrite <= 1'b0;
150             MemRW <= 1'b0;
151             CPU_MIO <= 1'b0;
152             case(Fun3)
153                 3'b000: ALU_Control <= 4'b0001;
154                 3'b001: ALU_Control <= 4'b0001;
```

```
155                 3'b100: ALU_Control <= 4'b0011;
156                 3'b101: ALU_Control <= 4'b0011;
157                 3'b110: ALU_Control <= 4'b0100;
158                 3'b111: ALU_Control <= 4'b0100;
159             endcase
160         end
161         5'b11011:begin
162             ImmSel <= 3'b100;
163             ALUSrc_B <= 1'b1;
164             MemtoReg <= 3'b010;
165             Jump <= 2'b01;
166             Branch_Beq <= 1'b0;
167             Branch_Bne <= 1'b0;
168             Branch_Blt <= 1'b0;
169             Branch_Bltu <= 1'b0;
170             Branch_Bge <= 1'b0;
171             Branch_Bgeu <= 1'b0;
172             Length <= 3'b000;
173             RegWrite <= 1'b1;
174             MemRW <= 1'b0;
175             CPU_MIO <= 1'b0;
176             ALU_Control <= 4'b0000;
177         end
178         5'b01101:begin
179             ImmSel <= 3'b011;
180             ALUSrc_B <= 1'b0;
181             MemtoReg <= 3'b011;
182             Jump <= 2'b00;
183             Branch_Beq <= 1'b0;
184             Branch_Bne <= 1'b0;
185             Branch_Blt <= 1'b0;
186             Branch_Bltu <= 1'b0;
187             Branch_Bge <= 1'b0;
188             Branch_Bgeu <= 1'b0;
189             Length <= 3'b000;
190             RegWrite <= 1'b1;
```

```
            MemRW <= 1'b0;
            CPU_MIO <= 1'b0;
            ALU_Control <= 4'b0000;
        end
        5'b00101:begin
            ImmSel <= 3'b011;
            ALUSrc_B <= 1'b0;
            MemtoReg <= 3'b100;
            Jump <= 2'b00;
            Branch_Beq <= 1'b0;
            Branch_Bne <= 1'b0;
            Branch_Blt <= 1'b0;
            Branch_Bltu <= 1'b0;
            Branch_Bge <= 1'b0;
            Branch_Bgeu <= 1'b0;
            Length <= 3'b000;
            RegWrite <= 1'b1;
            MemRW <= 1'b0;
            CPU_MIO <= 1'b0;
            ALU_Control <= 4'b0000;
        end
    endcase
end
endmodule
```

1.4.4.4 DataPath_W.v

```
module DataPath_W(
  input clk,
  input rst,
  input [4:0]       OPcode,
  input [2:0]       Fun3,
  input             Fun7,
  input             MIO_ready,
  input [31:0]inst_field,//instruction field
  input [31:0]Data_in,//memory in data
  output wire EX_MEM_MemRW,
```

```
11  output reg [31:0]PC_out,//next PC/NOW PC
12  output wire [31:0]Data_out,//store data out
13  output wire [31:0]EX_MEM_ALU_out,//ADDR OF MEM
14  output wire [3:0]wea,
15  output [31:0] Reg00,
16  output [31:0] Reg01,
17  output [31:0] Reg02,
18  output [31:0] Reg03,
19  output [31:0] Reg04,
20  output [31:0] Reg05,
21  output [31:0] Reg06,
22  output [31:0] Reg07,
23  output [31:0] Reg08,
24  output [31:0] Reg09,
25  output [31:0] Reg10,
26  output [31:0] Reg11,
27  output [31:0] Reg12,
28  output [31:0] Reg13,
29  output [31:0] Reg14,
30  output [31:0] Reg15,
31  output [31:0] Reg16,
32  output [31:0] Reg17,
33  output [31:0] Reg18,
34  output [31:0] Reg19,
35  output [31:0] Reg20,
36  output [31:0] Reg21,
37  output [31:0] Reg22,
38  output [31:0] Reg23,
39  output [31:0] Reg24,
40  output [31:0] Reg25,
41  output [31:0] Reg26,
42  output [31:0] Reg27,
43  output [31:0] Reg28,
44  output [31:0] Reg29,
45  output [31:0] Reg30,
46  output [31:0] Reg31
```

```
);
initial begin
  PC_out <= 32'd0;
end

wire [1:0] EX_MEM_Jump;
wire Branch;
wire [31:0]IF_ID_inst_field,IF_ID_PC;
wire [2:0]IF_ID_Fun3;
wire IF_ID_Fun7;
wire [4:0]IF_ID_OPcode;
Reg_IF_ID IF_ID(
  .clk(clk),
  .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
  .PC(PC_out),
  .inst_field(inst_field),
  .IF_ID_PC(IF_ID_PC),
  .IF_ID_inst_field(IF_ID_inst_field),
  .IF_ID_Fun3(IF_ID_Fun3),
  .IF_ID_OPcode(IF_ID_OPcode),
  .IF_ID_Fun7(IF_ID_Fun7)
);

wire [2:0] ImmSel;
wire ALUSrc_B;
wire [2:0] MemtoReg;
wire [1:0] Jump;
wire
Branch_Beq,Branch_Bne,Branch_Blt,Branch_Bltu,Branch_Bge,Branch_Bge
u;
wire [2:0] Length;
wire RegWrite;

wire [3:0] ALU_Control;
wire CPU_MIO;
wire MemRW;
```

```
SCPU_ctrl_W SCPU_CTRL(
  .OPcode(IF_ID_OPcode),
  .Fun3(IF_ID_Fun3),
  .Fun7(IF_ID_Fun7),
  .MIO_ready(MIO_ready),
  .ImmSel(ImmSel),
  .ALUSrc_B(ALUSrc_B),
  .MemtoReg(MemtoReg),
  .Jump(Jump),
  .Branch_Beq(Branch_Beq),
  .Branch_Bne(Branch_Bne),
  .Branch_Blt(Branch_Blt),
  .Branch_Bltu(Branch_Bltu),
  .Branch_Bge(Branch_Bge),
  .Branch_Bgeu(Branch_Bgeu),
  .Length(Length),
  .RegWrite(RegWrite),
  .MemRW(MemRW),
  .ALU_Control(ALU_Control),
  .CPU_MIO(CPU_MIO)
);

wire [31:0] Rs1_data, Rs2_data;
reg [31:0]Wt_data;

wire [4:0] MEM_WB_Wt_addr;
wire MEM_WB_RegWrite;
wire [1:0]Forward_A,Forward_B;
Regs U1(
.clk(clk),
.rst(rst),
.Rs1_addr(IF_ID_inst_field[19:15]),
.Rs2_addr(IF_ID_inst_field[24:20]),
.Wt_addr(MEM_WB_Wt_addr),
.Wt_data(Wt_data),
.RegWrite(MEM_WB_RegWrite),
```

```
.Rs1_data(Rs1_data),
.Rs2_data(Rs2_data),
.Reg00(Reg00),
.Reg01(Reg01),
.Reg02(Reg02),
.Reg03(Reg03),
.Reg04(Reg04),
.Reg05(Reg05),
.Reg06(Reg06),
.Reg07(Reg07),
.Reg08(Reg08),
.Reg09(Reg09),
.Reg10(Reg10),
.Reg11(Reg11),
.Reg12(Reg12),
.Reg13(Reg13),
.Reg14(Reg14),
.Reg15(Reg15),
.Reg16(Reg16),
.Reg17(Reg17),
.Reg18(Reg18),
.Reg19(Reg19),
.Reg20(Reg20),
.Reg21(Reg21),
.Reg22(Reg22),
.Reg23(Reg23),
.Reg24(Reg24),
.Reg25(Reg25),
.Reg26(Reg26),
.Reg27(Reg27),
.Reg28(Reg28),
.Reg29(Reg29),
.Reg30(Reg30),
.Reg31(Reg31)
);
//building
```

```
153 wire [31:0] Imm_out;
154 ImmGen U2(
155   .inst_field(IF_ID_inst_field),
156   .ImmSel(ImmSel),
157   .Imm_out(Imm_out)
158 );
159
160 wire [31:0]ID_EX_PC,ID_EX_Rs1_data,ID_EX_Rs2_data,ID_EX_Imm;
161 wire [2:0]ID_EX_Length,ID_EX_MemtoReg;
162 wire ID_EX_RegWrite;
163 wire
    ID_EX_Branch_Beq,ID_EX_Branch_Bne,ID_EX_Branch_Blt,ID_EX_Branch_Bl
    tu,ID_EX_Branch_Bge,ID_EX_Branch_Bgeu;
164 wire ID_EX_MemRW;
165 wire [1:0]ID_EX_Jump;
166
167 wire [3:0]ID_EX_wea;
168 wire [3:0]ID_EX_ALU_Control;
169 wire ID_EX_ALUSrc_B;
170 wire [4:0]ID_EX_Wt_addr,ID_EX_Rs1_addr,ID_EX_Rs2_addr;
171
172 Reg_ID_EX ID_EX(
173   .clk(clk),
174   .rst(rst|Branch|(EX_MEM_Jump[0]^EX_MEM_Jump[1])),
175   .PC(IF_ID_PC),
176   .Rs1_addr(IF_ID_inst_field[19:15]),
177   .Rs2_addr(IF_ID_inst_field[24:20]),
178   .Rs1_data(Rs1_data),
179   .Rs2_data(Rs2_data),
180   .Imm(Imm_out),
181   .Wt_addr(IF_ID_inst_field[11:7]),
182   .Length(Length),
183   .MemtoReg(MemtoReg),
184   .RegWrite(RegWrite),
185   .Branch_Beq(Branch_Beq),
186   .Branch_Bne(Branch_Bne),
```

```
187     .Branch_Bge(Branch_Bge),
188     .Branch_Blt(Branch_Blt),
189     .Branch_Bgeu(Branch_Bgeu),
190     .Branch_Bltu(Branch_Bltu),
191     .MemRW(MemRW),
192     .Jump(Jump),
193     .wea(wea),
194     .ALU_Control(ALU_Control),
195     .ALUSrc_B(ALUSrc_B),
196     .ID_EX_PC(ID_EX_PC),
197     .ID_EX_Rs1_data(ID_EX_Rs1_data),
198     .ID_EX_Rs2_data(ID_EX_Rs2_data),
199     .ID_EX_Imm(ID_EX_Imm),
200     .ID_EX_Length(ID_EX_Length),
201     .ID_EX_MemtoReg(ID_EX_MemtoReg),
202     .ID_EX_RegWrite(ID_EX_RegWrite),
203     .ID_EX_Branch_Beq(ID_EX_Branch_Beq),
204     .ID_EX_Branch_Bne(ID_EX_Branch_Bne),
205     .ID_EX_Branch_Bge(ID_EX_Branch_Bge),
206     .ID_EX_Branch_Blt(ID_EX_Branch_Blt),
207     .ID_EX_Branch_Bgeu(ID_EX_Branch_Bgeu),
208     .ID_EX_Branch_Bltu(ID_EX_Branch_Bltu),
209     .ID_EX_MemRW(ID_EX_MemRW),
210     .ID_EX_Jump(ID_EX_Jump),
211     .ID_EX_wea(ID_EX_wea),
212     .ID_EX_ALU_Control(ID_EX_ALU_Control),
213     .ID_EX_ALUSrc_B(ID_EX_ALUSrc_B),
214     .ID_EX_Rs1_addr(ID_EX_Rs1_addr),
215     .ID_EX_Rs2_addr(ID_EX_Rs2_addr),
216     .ID_EX_Wt_addr(ID_EX_Wt_addr)
217 );
218 
219 wire [31:0] Ior2;
220 wire [31:0] Forward_A_data,Forward_B_data;
```

```
assign Forward_A_data = (Forward_A==2'b00)?ID_EX_Rs1_data:((Forward_A==2'b01)?Tem_RegWrite:((Forward_A==2'b10)?Wt_data:32'b0));
assign Forward_B_data = (Forward_B==2'b00)?ID_EX_Rs2_data:((Forward_B==2'b01)?Tem_RegWrite:((Forward_B==2'b10)?Wt_data:32'b0));
assign Ior2 = ID_EX_ALUSrc_B? ID_EX_Imm: Forward_B_data;
wire zero;
wire [31:0] EX_ALU_out;
wire [31:0] Tem_RegWrite;
ALU U3(
  .A(Forward_A_data),
  .B(Ior2),
  .ALU_operation(ID_EX_ALU_Control),
  .res(EX_ALU_out),
  .zero(zero)
);
wire [31:0] PC_Jal ;
assign PC_Jal = ID_EX_PC+ID_EX_Imm;

wire [31:0] EX_MEM_PC,EX_MEM_PC_Jal,EX_MEM_Rs2_data,EX_MEM_Imm;
wire [4:0] EX_MEM_Wt_addr,EX_MEM_Rs2_addr;
wire EX_MEM_zero;
wire EX_MEM_Branch_Beq,EX_MEM_Branch_Bne,EX_MEM_Branch_Blt,EX_MEM_Branch_Bltu,EX_MEM_Branch_Bge,EX_MEM_Branch_Bgeu;
wire [2:0] EX_MEM_Length,EX_MEM_MemtoReg;
wire EX_MEM_RegWrite;

wire [3:0] EX_MEM_wea;
Reg_EX_MEM EX_MEM(
  .clk(clk),
  .rst(rst),
  .PC(ID_EX_PC),
  .PC_Jal(PC_Jal),
  .ALU_out(EX_ALU_out),
```

```
251     .zero(zero),
252     .Rs2_data(ID_EX_Rs2_data),
253     .Rs2_addr(ID_EX_Rs2_addr),
254     .Imm(ID_EX_Imm),
255     .Wt_addr(ID_EX_Wt_addr),
256     .Length(ID_EX_Length),
257     .MemtoReg(ID_EX_MemtoReg),
258     .RegWrite(ID_EX_RegWrite),
259     .Branch_Beq(ID_EX_Branch_Beq),
260     .Branch_Bne(ID_EX_Branch_Bne),
261     .Branch_Bge(ID_EX_Branch_Bge),
262     .Branch_Blt(ID_EX_Branch_Blt),
263     .Branch_Bgeu(ID_EX_Branch_Bgeu),
264     .Branch_Bltu(ID_EX_Branch_Bltu),
265     .MemRW(ID_EX_MemRW),
266     .Jump(ID_EX_Jump),
267     .wea(ID_EX_wea),
268     .EX_MEM_PC(EX_MEM_PC),
269     .EX_MEM_PC_Jal(EX_MEM_PC_Jal),
270     .EX_MEM_ALU_out(EX_MEM_ALU_out),
271     .EX_MEM_zero(EX_MEM_zero),
272     .EX_MEM_Rs2_data(EX_MEM_Rs2_data),
273     .EX_MEM_Rs2_addr(EX_MEM_Rs2_addr),
274     .EX_MEM_Imm(EX_MEM_Imm),
275     .EX_MEM_Wt_addr(EX_MEM_Wt_addr),
276     .EX_MEM_Length(EX_MEM_Length),
277     .EX_MEM_MemtoReg(EX_MEM_MemtoReg),
278     .EX_MEM_RegWrite(EX_MEM_RegWrite),
279     .EX_MEM_Branch_Beq(EX_MEM_Branch_Beq),
280     .EX_MEM_Branch_Bne(EX_MEM_Branch_Bne),
281     .EX_MEM_Branch_Bge(EX_MEM_Branch_Bge),
282     .EX_MEM_Branch_Blt(EX_MEM_Branch_Blt),
283     .EX_MEM_Branch_Bgeu(EX_MEM_Branch_Bgeu),
284     .EX_MEM_Branch_Bltu(EX_MEM_Branch_Bltu),
285     .EX_MEM_MemRW(EX_MEM_MemRW),
286     .EX_MEM_Jump(EX_MEM_Jump),
```

```
  .EX_MEM_wea(EX_MEM_wea)
);


Tem_RegWrite Temreg(
  .MemtoReg(EX_MEM_MemtoReg),
  .Length(EX_MEM_Length),
  .Data_in(Data_in),
  .ALU_out(EX_MEM_ALU_out),
  .Imm(EX_MEM_Imm),
  .PC_out(EX_MEM_PC),
  .Tem_RegWrite(Tem_RegWrite)
);
wire [31:0] MEM_WB_PC,MEM_WB_ALU_out,MEM_WB_Data_in,MEM_WB_Imm;

wire [2:0] MEM_WB_Length,MEM_WB_MemtoReg;

Reg_MEM_WB MEM_WB(
  .clk(clk),
  .rst(rst),
  .PC(EX_MEM_PC),
  .ALU_out(EX_MEM_ALU_out),
  .Data_in(Data_in),
  .Imm(EX_MEM_Imm),
  .Wt_addr(EX_MEM_Wt_addr),
  .Length(EX_MEM_Length),
  .MemtoReg(EX_MEM_MemtoReg),
  .RegWrite(EX_MEM_RegWrite),
  .MEM_WB_PC(MEM_WB_PC),
  .MEM_WB_ALU_out(MEM_WB_ALU_out),
  .MEM_WB_Data_in(MEM_WB_Data_in),
  .MEM_WB_Imm(MEM_WB_Imm),
  .MEM_WB_Wt_addr(MEM_WB_Wt_addr),
  .MEM_WB_Length(MEM_WB_Length),
  .MEM_WB_MemtoReg(MEM_WB_MemtoReg),
  .MEM_WB_RegWrite(MEM_WB_RegWrite)
```

```
);

wire For_M;
Forwarding_Ctrl forward_strl(
  .ID_EX_Rs1_addr(ID_EX_Rs1_addr),
  .ID_EX_Rs2_addr(ID_EX_Rs2_addr),
  .EX_MEM_Wt_addr(EX_MEM_Wt_addr),
  .MEM_WB_Wt_addr(MEM_WB_Wt_addr),
  .EX_MEM_RegWrite(EX_MEM_RegWrite),
  .MEM_WB_RegWrite(MEM_WB_RegWrite),
  .EX_MEM_Rs2_addr(EX_MEM_Rs2_addr),
  .For_A(Forward_A),
  .For_B(Forward_B),
  .For_M(For_M)
);

assign Branch = (EX_MEM_Branch_Beq&EX_MEM_zero)|
(EX_MEM_Branch_Bne&(~EX_MEM_zero))|
(EX_MEM_Branch_Blt&EX_MEM_ALU_out[0])|
(EX_MEM_Branch_Bltu&EX_MEM_ALU_out[0])|(EX_MEM_Branch_Bge&
(~EX_MEM_ALU_out[0]))|(EX_MEM_Branch_Bgeu&(~EX_MEM_ALU_out[0]));
wire [31:0] PC_Branch = Branch?EX_MEM_PC_Jal:EX_MEM_PC+32'd4;
reg [31:0] PC_next;
wire [31:0] reg_mem;
wire [31:0] reg_Data_out;
reg [31:0] Tem_mem,Tem_Data;
reg [3:0]Tem_wea;
assign reg_Data_out = For_M?Wt_data:EX_MEM_Rs2_data;
assign reg_mem = Tem_mem;
assign wea = Tem_wea;
assign Data_out = Tem_Data;
always @(*)begin

  case(MEM_WB_Length)//memtoreg
    3'b000:
    case(MEM_WB_ALU_out[1:0])
```

```
355       2'b00:Tem_mem <= {24'b0,MEM_WB_Data_in[7:0]};
356       2'b01:Tem_mem <= {24'b0,MEM_WB_Data_in[15:8]};
357       2'b10:Tem_mem <= {24'b0,MEM_WB_Data_in[23:16]};
358       2'b11:Tem_mem <= {24'b0,MEM_WB_Data_in[31:24]};
359     endcase
360     3'b001:
361     case(MEM_WB_ALU_out[1:0])
362       2'b00:Tem_mem <=
{{24{MEM_WB_Data_in[7]}},MEM_WB_Data_in[7:0]};
363       2'b01:Tem_mem <=
{{24{MEM_WB_Data_in[15]}},MEM_WB_Data_in[15:8]};
364       2'b10:Tem_mem <=
{{24{MEM_WB_Data_in[23]}},MEM_WB_Data_in[23:16]};
365       2'b11:Tem_mem <=
{{24{MEM_WB_Data_in[31]}},MEM_WB_Data_in[31:24]};
366     endcase
367     3'b010:
368     case(MEM_WB_ALU_out[1:0])
369       2'b00:Tem_mem <= {16'b0,MEM_WB_Data_in[15:0]};
370       2'b01:Tem_mem <= {16'b0,MEM_WB_Data_in[23:8]};
371       2'b10:Tem_mem <= {16'b0,MEM_WB_Data_in[31:16]};
372     endcase
373     3'b011:
374     case(MEM_WB_ALU_out[1:0])
375       2'b00:Tem_mem <=
{{16{MEM_WB_Data_in[15]}},MEM_WB_Data_in[15:0]};
376       2'b01:Tem_mem <=
{{16{MEM_WB_Data_in[23]}},MEM_WB_Data_in[23:8]};
377       2'b10:Tem_mem <=
{{16{MEM_WB_Data_in[31]}},MEM_WB_Data_in[31:16]};
378     endcase
379     3'b100:
380      Tem_mem <= MEM_WB_Data_in;
381   endcase
382   case(MEM_WB_MemtoReg)
383     3'b000: Wt_data <= MEM_WB_ALU_out;
```

```
    3'b001: Wt_data <= reg_mem;
    3'b010: Wt_data <= MEM_WB_PC+32'd4;
    3'b011: Wt_data <= MEM_WB_Imm<<12;
    3'b100: Wt_data <= MEM_WB_PC+(MEM_WB_Imm<<12);
  endcase
  case(EX_MEM_Jump)
    2'b11: PC_next <= PC_Branch;
    2'b01: PC_next <= EX_MEM_PC_Jal;
    2'b10: PC_next <= {EX_MEM_ALU_out[31:1],1'b0};
  endcase
  case(EX_MEM_Length)//RAM
    3'b000:begin
      case(EX_MEM_ALU_out[1:0])
        2'b00: begin
          Tem_wea <= 4'b0001;
          Tem_Data <= {24'b0,reg_Data_out[7:0]};
        end
        2'b01: begin
          Tem_wea <= 4'b0010;
          Tem_Data <= {16'b0,reg_Data_out[7:0],8'b0};
        end
        2'b10: begin
          Tem_wea <= 4'b0100;
          Tem_Data <= {8'b0,reg_Data_out[7:0],16'b0};
        end
        2'b11: begin
          Tem_wea <= 4'b1000;
          Tem_Data <= {reg_Data_out[7:0],24'b0};
        end
      endcase
    end
    3'b010:begin
      case(EX_MEM_ALU_out[1:0])
        2'b00: begin
          Tem_wea <= 4'b0011;
          Tem_Data <= {16'b0,reg_Data_out[15:0]};
```

```
420         end
421         2'b01: begin
422           Tem_wea <= 4'b0110;
423           Tem_Data <= {8'b0,reg_Data_out[15:0],8'b0};
424         end
425         2'b10: begin
426           Tem_wea <= 4'b1100;
427           Tem_Data <= {reg_Data_out[15:0],16'b0};
428         end
429       endcase
430     end
431     3'b100:begin
432     Tem_wea <= 4'b1111;
433     Tem_Data <= reg_Data_out;
434     end
435   endcase
436 
437 end
438 
439 always @(posedge clk or posedge rst)begin
440   if(rst)PC_out<=32'd0;
441   else if(EX_MEM_Jump==2'b00)begin
442     PC_out <= PC_out+32'd4;
443   end
444   else PC_out<=PC_next;
445 end
446 endmodule
```

1.4.4.5 Reg_IF_ID.v

```
1  module Reg_IF_ID(
2      input clk,
3      input rst,
4      input [31:0] PC,
5      input [31:0]inst_field,
6      output reg [31:0]IF_ID_PC,
7      output reg [31:0]IF_ID_inst_field,
```

```
    output reg [2:0]IF_ID_Fun3,
    output reg [4:0]IF_ID_OPcode,
    output reg IF_ID_Fun7
    );

always@(posedge clk or posedge rst)begin
    if(rst)begin
        IF_ID_PC <= 32'b0;
        IF_ID_inst_field <= 32'h00000013;
        IF_ID_Fun3 <= 3'b0;
        IF_ID_Fun7 <= 7'b0;
        IF_ID_OPcode <= 5'b0;
    end
    else begin
        IF_ID_PC <= PC;
        IF_ID_inst_field <= inst_field;
        IF_ID_Fun3 <= inst_field[14:12];
        IF_ID_Fun7 <= inst_field[30];
        IF_ID_OPcode <= inst_field[6:2];
    end
end
endmodule
```

1.4.4.6 Reg_ID_EX.v

```
module Reg_ID_EX(
    input clk,
    input rst,
    input [31:0] PC,
    input [4:0]Rs1_addr,
    input [4:0]Rs2_addr,
    input [31:0] Rs1_data,
    input [31:0] Rs2_data,
    input [31:0] Imm,
    input [4:0]Wt_addr,
    input [2:0]Length,
    input [2:0]MemtoReg,
```

```
13      input  RegWrite,
14      input  Branch_Beq,
15      input  Branch_Bne,
16      input  Branch_Bge,
17      input  Branch_Blt,
18      input  Branch_Bgeu,
19      input  Branch_Bltu,
20      input  MemRW,
21      input [1:0] Jump,
22      input [3:0] wea,
23      input [3:0]ALU_Control,
24      input ALUSrc_B,
25      output reg [4:0]ID_EX_Rs1_addr,
26      output reg [4:0]ID_EX_Rs2_addr,
27      output reg [31:0]ID_EX_PC,
28      output reg [31:0]ID_EX_Rs1_data,
29      output reg [31:0]ID_EX_Rs2_data,
30      output reg [31:0]ID_EX_Imm,
31      output reg [2:0]ID_EX_Length,
32      output reg [2:0]ID_EX_MemtoReg,
33      output reg ID_EX_RegWrite,
34      output reg ID_EX_Branch_Beq,
35      output reg ID_EX_Branch_Bne,
36      output reg ID_EX_Branch_Bge,
37      output reg ID_EX_Branch_Blt,
38      output reg ID_EX_Branch_Bgeu,
39      output reg ID_EX_Branch_Bltu,
40      output reg ID_EX_MemRW,
41      output reg [1:0]ID_EX_Jump,
42      output reg [3:0]ID_EX_wea,
43      output reg [3:0]ID_EX_ALU_Control,
44      output reg ID_EX_ALUSrc_B,
45      output reg [4:0]ID_EX_Wt_addr
46      );
47  always@(posedge clk or posedge rst)begin
48      if(rst)begin
```

```
49          ID_EX_PC <= 32'b0;
50          ID_EX_Rs1_data <= 32'b0;
51          ID_EX_Rs2_data <= 32'b0;
52          ID_EX_Imm <= 32'b0;
53          ID_EX_Length <= 3'b0;
54          ID_EX_MemtoReg <= 3'b0;
55          ID_EX_RegWrite <= 1'b0;
56          ID_EX_Branch_Beq <= 1'b0;
57          ID_EX_Branch_Bne <= 1'b0;
58          ID_EX_Branch_Bge <= 1'b0;
59          ID_EX_Branch_Blt <= 1'b0;
60          ID_EX_Branch_Bgeu <= 1'b0;
61          ID_EX_Branch_Bltu <= 1'b0;
62          ID_EX_MemRW <= 1'b0;
63          ID_EX_Jump <= 2'b0;
64          ID_EX_wea <= 4'b0;
65          ID_EX_ALU_Control <= 4'b0;
66          ID_EX_ALUSrc_B <= 1'b0;
67          ID_EX_Wt_addr <= 5'b0;
68          ID_EX_Rs1_addr <= 5'b0;
69          ID_EX_Rs2_addr <= 5'b0;
70      end
71      else begin
72          ID_EX_PC <= PC;
73          ID_EX_Rs1_data <= Rs1_data;
74          ID_EX_Rs2_data <= Rs2_data;
75          ID_EX_Imm <= Imm;
76          ID_EX_Length <= Length;
77          ID_EX_MemtoReg <= MemtoReg;
78          ID_EX_RegWrite <= RegWrite;
79          ID_EX_Branch_Beq <= Branch_Beq;
80          ID_EX_Branch_Bne <= Branch_Bne;
81          ID_EX_Branch_Bge <= Branch_Bge;
82          ID_EX_Branch_Blt <= Branch_Blt;
83          ID_EX_Branch_Bgeu <= Branch_Bgeu;
84          ID_EX_Branch_Bltu <= Branch_Bltu;
```

```
        ID_EX_MemRW <= MemRW;
        ID_EX_Jump <= Jump;
        ID_EX_wea <= wea;
        ID_EX_ALU_Control <= ALU_Control;
        ID_EX_ALUSrc_B <= ALUSrc_B;
        ID_EX_Wt_addr <= Wt_addr;
        ID_EX_Rs1_addr <= Rs1_addr;
        ID_EX_Rs2_addr <= Rs2_addr;
    end
end
```

1.4.4.7 Reg_EX_MEM.v

```
module Reg_EX_MEM(
    input clk,
    input rst,
    input [31:0] PC,
    input [31:0] PC_Jal,
    input [31:0] ALU_out,
    input zero,
    input [31:0] Rs2_data,
    input [4:0] Rs2_addr,
    input [31:0] Imm,
    input [4:0]Wt_addr,
    input [2:0]Length,
    input [2:0]MemtoReg,
    input  RegWrite,
    input  Branch_Beq,
    input  Branch_Bne,
    input  Branch_Bge,
    input  Branch_Blt,
    input  Branch_Bgeu,
    input  Branch_Bltu,
    input  MemRW,
    input [1:0] Jump,
    input [3:0] wea,
    output reg [31:0]EX_MEM_PC,
```

```
    output reg [31:0]EX_MEM_PC_Jal,
    output reg [31:0]EX_MEM_ALU_out,
    output reg EX_MEM_zero,
    output reg [31:0]EX_MEM_Rs2_data,
    output reg [4:0]EX_MEM_Rs2_addr,
    output reg [31:0]EX_MEM_Imm,
    output reg [4:0]EX_MEM_Wt_addr,
    output reg [2:0]EX_MEM_Length,
    output reg [2:0]EX_MEM_MemtoReg,
    output reg EX_MEM_RegWrite,
    output reg EX_MEM_Branch_Beq,
    output reg EX_MEM_Branch_Bne,
    output reg EX_MEM_Branch_Bge,
    output reg EX_MEM_Branch_Blt,
    output reg EX_MEM_Branch_Bgeu,
    output reg EX_MEM_Branch_Bltu,
    output reg EX_MEM_MemRW,
    output reg [1:0]EX_MEM_Jump,
    output reg [3:0]EX_MEM_wea
    );
always@(posedge clk or posedge rst)begin
    if(rst) begin
        EX_MEM_PC <= 32'b0;
        EX_MEM_PC_Jal <= 32'b0;
        EX_MEM_ALU_out <= 32'b0;
        EX_MEM_zero <= 1'b0;
        EX_MEM_Rs2_data <= 32'b0;
        EX_MEM_Imm <= 32'b0;
        EX_MEM_Wt_addr <= 5'b0;
        EX_MEM_Length <= 3'b0;
        EX_MEM_MemtoReg <= 3'b0;
        EX_MEM_RegWrite <= 1'b0;
        EX_MEM_Branch_Beq <= 1'b0;
        EX_MEM_Branch_Bne <= 1'b0;
        EX_MEM_Branch_Bge <= 1'b0;
        EX_MEM_Branch_Blt <= 1'b0;
```

```
61          EX_MEM_Branch_Bgeu <= 1'b0;
62          EX_MEM_Branch_Bltu <= 1'b0;
63          EX_MEM_MemRW <= 1'b0;
64          EX_MEM_Jump <= 2'b0;
65          EX_MEM_wea <= 4'b0;
66          EX_MEM_Rs2_addr <= 5'b0;
67      end
68      else begin
69          EX_MEM_PC <= PC;
70          EX_MEM_PC_Jal <= PC_Jal;
71          EX_MEM_ALU_out <= ALU_out;
72          EX_MEM_zero <= zero;
73          EX_MEM_Rs2_data <= Rs2_data;
74          EX_MEM_Imm <= Imm;
75          EX_MEM_Wt_addr <= Wt_addr;
76          EX_MEM_Length <= Length;
77          EX_MEM_MemtoReg <= MemtoReg;
78          EX_MEM_RegWrite <= RegWrite;
79          EX_MEM_Branch_Beq <= Branch_Beq;
80          EX_MEM_Branch_Bne <= Branch_Bne;
81          EX_MEM_Branch_Bge <= Branch_Bge;
82          EX_MEM_Branch_Blt <= Branch_Blt;
83          EX_MEM_Branch_Bgeu <= Branch_Bgeu;
84          EX_MEM_Branch_Bltu <= Branch_Bltu;
85          EX_MEM_MemRW <= MemRW;
86          EX_MEM_Jump <= Jump;
87          EX_MEM_wea <= wea;
88          EX_MEM_Rs2_addr <= Rs2_addr;
89      end
90  end
91  endmodule
```

1.4.4.8 Reg_MEM_WB.v

```
module Reg_MEM_WB(
    input clk,
    input rst,
    input [31:0] PC,
    input [31:0] ALU_out,
    input [31:0] Data_in,
    input [31:0] Imm,
    input [4:0]Wt_addr,
    input [2:0]Length,
    input [2:0]MemtoReg,
    input  RegWrite,

    output reg [31:0]MEM_WB_PC,
    output reg [31:0]MEM_WB_ALU_out,
    output reg [31:0]MEM_WB_Data_in,
    output reg [31:0]MEM_WB_Imm,
    output reg [4:0]MEM_WB_Wt_addr,
    output reg [2:0]MEM_WB_Length,
    output reg [2:0]MEM_WB_MemtoReg,
    output reg MEM_WB_RegWrite

    );

always@(posedge clk or posedge rst)begin
    if(rst) begin
        MEM_WB_PC <= 32'b0;
        MEM_WB_ALU_out <= 32'b0;
        MEM_WB_Data_in <= 32'b0;
        MEM_WB_Imm <= 32'b0;
        MEM_WB_Wt_addr <= 5'b0;
        MEM_WB_Length <= 3'b0;
        MEM_WB_MemtoReg <= 3'b0;
        MEM_WB_RegWrite <= 1'b0;
    end
```

```
    else begin
        MEM_WB_PC <= PC;
        MEM_WB_ALU_out <= ALU_out;
        MEM_WB_Data_in <= Data_in;
        MEM_WB_Imm <= Imm;
        MEM_WB_Wt_addr <= Wt_addr;
        MEM_WB_Length <= Length;
        MEM_WB_MemtoReg <= MemtoReg;
        MEM_WB_RegWrite <= RegWrite;
    end
end
endmodule
```

1.4.4.9 Forwarding_Ctrl.v

```
module Forwarding_Ctrl(
    input [4:0]ID_EX_Rs1_addr,
    input [4:0]ID_EX_Rs2_addr,
    input [4:0]EX_MEM_Wt_addr,
    input [4:0]MEM_WB_Wt_addr,
    input [4:0]EX_MEM_Rs2_addr,
    input EX_MEM_RegWrite,
    input MEM_WB_RegWrite,
    output reg [1:0]For_A,
    output reg [1:0]For_B,
    output reg For_M
    );
    // assign For_M = (MEM_WB_RegWrite &&
MEM_WB_Wt_addr==EX_MEM_Rs2_addr)?1'b1:1'b0;
    always@(*)begin
        if(EX_MEM_RegWrite && ID_EX_Rs1_addr == EX_MEM_Wt_addr &&
EX_MEM_Wt_addr != 5'b0)begin
            For_A <= 2'b01;
        end
        else if(MEM_WB_RegWrite && ID_EX_Rs1_addr == MEM_WB_Wt_addr
&& MEM_WB_Wt_addr != 5'b0)begin
            For_A <= 2'b10;
```

```
        end
        else begin
            For_A <= 2'b00;
        end
        if(EX_MEM_RegWrite &&ID_EX_Rs2_addr == EX_MEM_Wt_addr &&
EX_MEM_Wt_addr != 5'b0)begin
            For_B <= 2'b01;
        end
        else if(MEM_WB_RegWrite && ID_EX_Rs2_addr == MEM_WB_Wt_addr
&& MEM_WB_Wt_addr != 5'b0)begin
            For_B <= 2'b10;
        end
        else begin
            For_B <= 2'b00;
        end
        if(MEM_WB_RegWrite && MEM_WB_Wt_addr==EX_MEM_Rs2_addr)
begin
            For_M <= 1'b1;
        end
        else begin
            For_M <= 1'b0;
        end
    end
endmodule
```

1.4.4.10 Tem_RegWrite.v

```
module Tem_RegWrite(
    input [2:0]MemtoReg,
    input [2:0]Length,
    input [31:0]Data_in,
    input [31:0]ALU_out,
    input [31:0]Imm,
    input [31:0]PC_out,
    output reg [31:0]Tem_RegWrite
    );

```

```
wire [31:0] reg_mem;
reg [31:0]tem_mem;
assign reg_mem = tem_mem;
always@(*)begin
    case(Length)
        3'b000:
        case(ALU_out[1:0])
            2'b00: tem_mem = {24'b0, Data_in[7:0]};
            2'b01: tem_mem = {24'b0, Data_in[15:8]};
            2'b10: tem_mem = {24'b0, Data_in[23:16]};
            2'b11: tem_mem = {24'b0, Data_in[31:24]};
        endcase
        3'b001:
        case(ALU_out[1:0])
            2'b00: tem_mem = {{24{Data_in[7]}}, Data_in[7:0]};
            2'b01: tem_mem =  {{24{Data_in[15]}}, Data_in[15:8]};
            2'b10: tem_mem =  {{24{Data_in[23]}}, Data_in[23:16]};
            2'b11: tem_mem =  {{24{Data_in[31]}}, Data_in[31:24]};
        endcase
        3'b010:
        case(ALU_out[1:0])
            2'b00: tem_mem = {16'b0, Data_in[15:0]};
            2'b01: tem_mem = {16'b0, Data_in[23:8]};
            2'b10: tem_mem = {16'b0, Data_in[31:16]};
        endcase
        3'b011:
        case(ALU_out[1:0])
            2'b00: tem_mem = {{16{Data_in[15]}}, Data_in[15:0]};
            2'b01: tem_mem =  {{16{Data_in[23]}}, Data_in[23:8]};
            2'b10: tem_mem =  {{16{Data_in[31]}}, Data_in[31:16]};
        endcase
        3'b100:
            tem_mem = Data_in;
    endcase
    case(MemtoReg)
        3'b000: Tem_RegWrite = ALU_out;
```

```
        3'b001: Tem_RegWrite = reg_mem;
        3'b010: Tem_RegWrite = PC_out+32'd4;
        3'b011: Tem_RegWrite = Imm<<12;
        3'b100: Tem_RegWrite = PC_out+(Imm<<12);
    endcase
end
endmodule
```

1.4.4.11 ImmGen.v

```
module ImmGen(
  input wire [2:0]   ImmSel,
  input wire [31:0]  inst_field,
  output reg [31:0]  Imm_out
);
//OI-type, 1S-type, 2B-type, 3U-type, 4J-type
always @(*) begin
case (ImmSel)
    3'd0: Imm_out = {{20{inst_field[31]}},inst_field[31:20]};
    3'd1: Imm_out = {{20{inst_field[31]}},inst_field[31:25],inst_field[11:7]};
    3'd2: Imm_out = {{19{inst_field[31]}},inst_field[31],inst_field[7],inst_field[30:25],inst_field[11:8],1'b0};
    3'd3: Imm_out = {{12{inst_field[31]}},inst_field[31:12]};
    3'd4: Imm_out = {{11{inst_field[31]}},inst_field[31],inst_field[19:12],inst_field[20],inst_field[30:21],1'b0};
    3'd5: Imm_out = {27'b0,inst_field[19:15]};
endcase
end
endmodule
```

1.4.4.12 ALU.v

```
module ALU (
  input [31:0]  A,
  input [31:0]  B,
  input [3:0]   ALU_operation,
  output[31:0]  res,
  output  zero
);
reg [31:0] tem;
reg flag;
always@(*)begin
case(ALU_operation)
    4'd0: tem = $signed(A) + $signed(B);
    4'd1: tem =  $signed(A) - $signed(B);
    4'd2: tem = A << B[4:0];
    4'd3: tem = $signed(A)<$signed(B)?32'd1:32'd0 ;
    4'd4: tem = $unsigned(A)<$unsigned(B)?32'd1:32'd0;
    4'd5: tem = A ^ B;
    4'd6: tem = A >> B[4:0];
    4'd7: tem = $signed(A)>>>B[4:0];
    4'd8: tem = A | B;
    4'd9: tem = A & B;
    default: tem = 32'dz;
endcase
flag <= tem?1'b0:1'b1;
end

assign zero = flag;
assign res = tem;
endmodule
```

1.4.4.13 Regs.v

```
module Regs(
    input clk,
    input rst,
```

```
    input [4:0] Rs1_addr,
    input [4:0] Rs2_addr,
    input [4:0] Wt_addr,
    input [31:0]Wt_data,
    input RegWrite,
    output wire [31:0] Rs1_data,
    output wire[31:0] Rs2_data,
    output wire[31:0] Reg00,
    output wire[31:0] Reg01,
    output wire[31:0] Reg02,
    output wire[31:0] Reg03,
    output wire[31:0] Reg04,
    output wire[31:0] Reg05,
    output wire[31:0] Reg06,
    output wire[31:0] Reg07,
    output wire[31:0] Reg08,
    output wire[31:0] Reg09,
    output wire[31:0] Reg10,
    output wire[31:0] Reg11,
    output wire[31:0] Reg12,
    output wire[31:0] Reg13,
    output wire[31:0] Reg14,
    output wire[31:0] Reg15,
    output wire[31:0] Reg16,
    output wire[31:0] Reg17,
    output wire[31:0] Reg18,
    output wire[31:0] Reg19,
    output wire[31:0] Reg20,
    output wire[31:0] Reg21,
    output wire[31:0] Reg22,
    output wire[31:0] Reg23,
    output wire[31:0] Reg24,
    output wire[31:0] Reg25,
    output wire[31:0] Reg26,
    output wire[31:0] Reg27,
    output wire[31:0] Reg28,
```

```
    output wire[31:0] Reg29,
    output wire[31:0] Reg30,
    output wire[31:0] Reg31
);

reg [31:0]tem[1:31];
assign Rs1_data = Rs1_addr? tem[Rs1_addr]:32'd0;
assign Rs2_data = Rs2_addr? tem[Rs2_addr]:32'd0;
assign Reg00 = 32'd0;
assign Reg01 = tem[1];
assign Reg02 = tem[2];
assign Reg03 = tem[3];
assign Reg04 = tem[4];
assign Reg05 = tem[5];
assign Reg06 = tem[6];
assign Reg07 = tem[7];
assign Reg08 = tem[8];
assign Reg09 = tem[9];
assign Reg10 = tem[10];
assign Reg11 = tem[11];
assign Reg12 = tem[12];
assign Reg13 = tem[13];
assign Reg14 = tem[14];
assign Reg15 = tem[15];
assign Reg16 = tem[16];
assign Reg17 = tem[17];
assign Reg18 = tem[18];
assign Reg19 = tem[19];
assign Reg20 = tem[20];
assign Reg21 = tem[21];
assign Reg22 = tem[22];
assign Reg23 = tem[23];
assign Reg24 = tem[24];
assign Reg25 = tem[25];
assign Reg26 = tem[26];
assign Reg27 = tem[27];
```

```
assign Reg28 = tem[28];
assign Reg29 = tem[29];
assign Reg30 = tem[30];
assign Reg31 = tem[31];
always @(posedge ~clk) begin
if(RegWrite&&!rst&&Wt_addr)begin
   tem[Wt_addr] <= Wt_data;
 end
end
endmodule
```

2 SCPU仿真结果与上板实验结果

2.1 仿真结果

我在Lab4-3的测试代码后新加了一段，有三个典型数据冒险示例测试，通过后会到达0x666的数值表示完整测试通过。

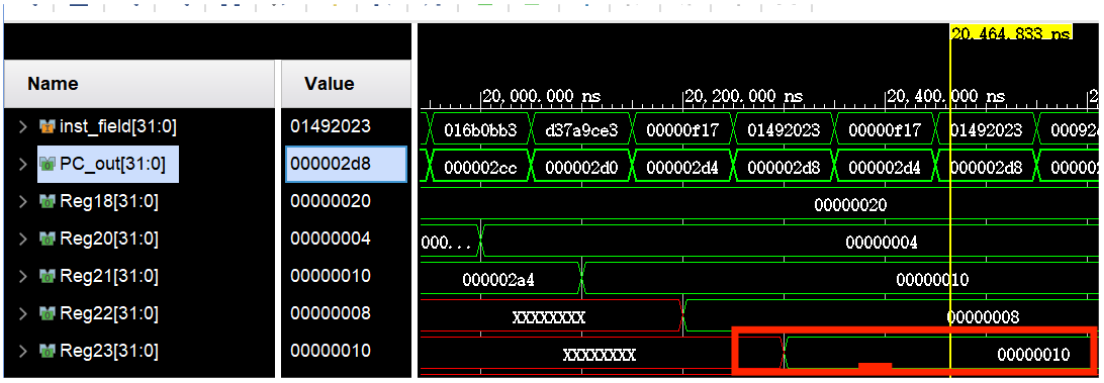
上文提到的数据冒险、结构冒险例子均为4-3原测试代码部分，上文测试仿真通过已经说明上文例子得到验证，在此处不再重复仿真说明。

现结合三个数据冲突举例和新加入仿真代码说明：

2.1.1 数据来自MEM阶段，取到寄存器值未更新

```
##x18 = 0x20
li x31,8
auipc x30, 0 #data-hazard-rs1&rs2
addi x20,x0, 0x4 #x20=0x4
addi x21,x0, 0x10 #x21=0x10
add x22,x20,x20 #x22=0x8
add x23,x22,x22 #x23=0x10,rs1&rs2 x22 is rd before
bne x21,x23,dummy
```

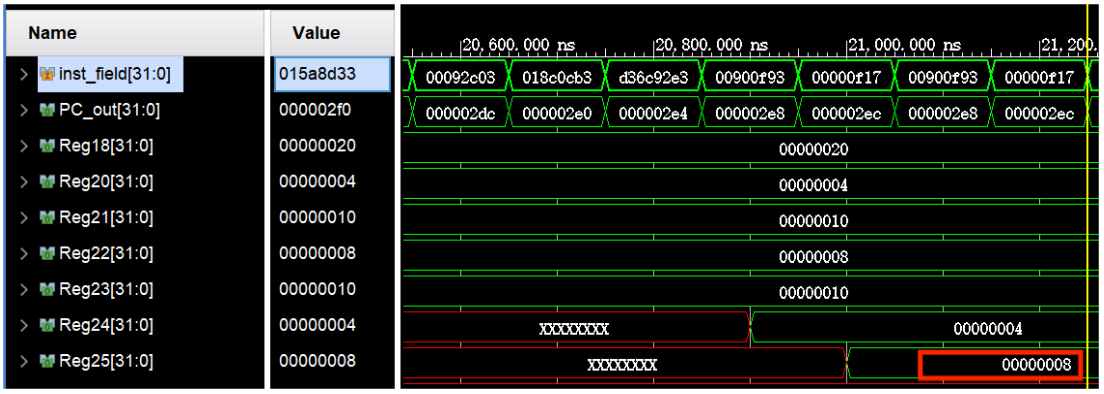
x31赋值0x8表示进入第一个测试阶段，这里的第6行刚刚算出x22的寄存器值，暂未写入寄存器，也就是说第7行add的两个算子都是旧值，这里选择前递更新掉这里的旧值来达到正确计算。



2.1.2 load_use问题

```
1  ##x18 = 0x20
2  auipc x30, 0 #data-hazard-load_use
3  sw x20,0(x18) #mem[20] = 0x4
4  lw x24,0(x18) #x24 = 0x4
5  add x25,x24,x24 #x25 = 0x8
6  bne x25,x22,dummy
```

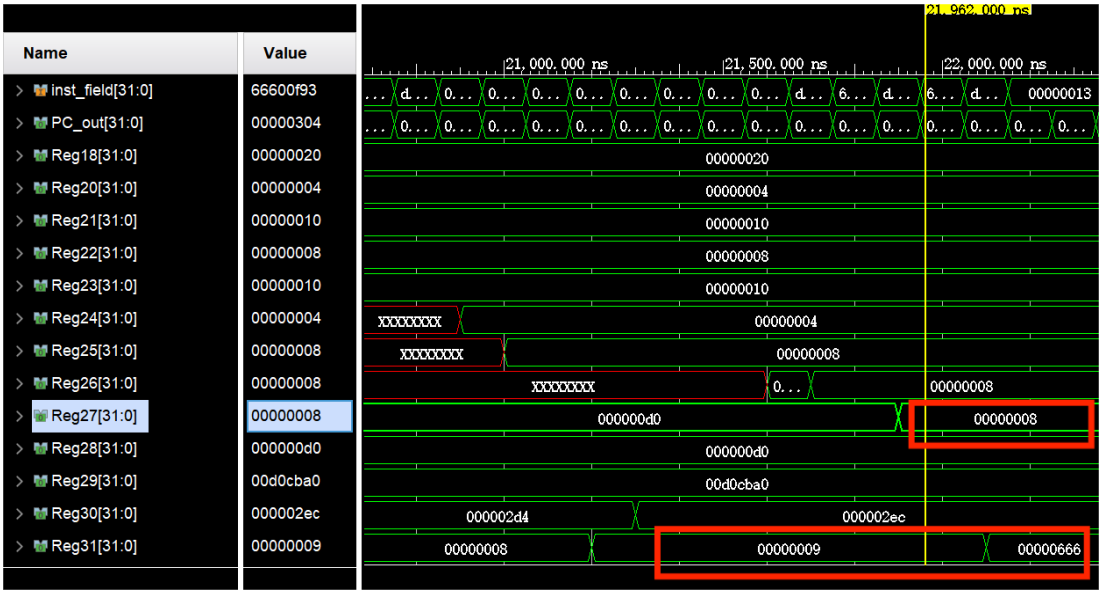
这里的第4行刚刚load出内存值存到x24中，x24的值暂无更新。第5行处紧接着使用x24的值，这里选择前递更新掉这里的旧值来达到正确计算。



2.1.3 数据来自WB阶段，RAM写入值未更新

```
##x18 = 0x20
li x31,9
auipc x30, 0 #data-hazard-regwrite-store
add x26,x21,x21 #x26 = 0x20
add x26,x20,x20 #x26 = 0x8
sw  x26, 0(x18) #mem[20] = 0x8
lw   x27,0(x18)  #x27 = mem[20] = 0x8
bne x27,x22,dummy
li    x31, 0x666
j     dummy
```

x31赋值0x9表示进入最后一个测试阶段。这里的第5行刚刚算出的结果还没存入寄存器x26，第6行即将从x26读将写入内存，如果没有进行处理写入内存的是0x20而不是0x8。在之后的`bne`处验证了确实不是0x20，0x8是正确答案。通过`bne`x31变为0x666表示测试通过。



2.2 实验上板

上板代码在Lab4-3的测试代码后新加了一段，有三个典型数据冒险例子测试，通过后x31会先以0x8->0x9->0x666的数值表示完整测试通过。

这里上板标出x31变化的三个阶段，详细分析在上文的仿真测试中提到了。

测试阶段1：`t6 = 0x7` -> `t6 = 0x8`

```
RV32I Single Cycle CPU

pc: 00000288    inst: D81FF06F  csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
x0: 000000000000ra: FFFFFFFF 000sp: 00000000 000gp: 40000000    tp: 40000000
t0: F8000000    t1: C0000000    t2: 80000000    s0: 00000001    s1: 00000001
a0: 00000000    a1: C0000000    a2: 00000001    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000020    s3: 00000000
s4: 00000000    s5: 00000010    s6: 00000008    s7: 00000010    s8: 00000004
s9: 00000008    s10:00000008    s11:000000D0    t3: 000000D0    t4: 00D0CBA0
t5: 00000278    t6: 00000007
```

```
RV32I Single Cycle CPU

pc: 000002CC    inst: 016B0BB3  csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
x0: 000000000000ra: FFFFFFFF 000sp: 00000000 000gp: 40000000    tp: 40000000
t0: F8000000    t1: C0000000    t2: 80000000    s0: 00000001    s1: 00000001
a0: 00000000    a1: C0000000    a2: 00000001    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000020    s3: 00000000
s4: 000002A4    s5: 000002A4    s6: 00000008    s7: 00000010    s8: 00000004
s9: 00000008    s10:00000008    s11:000000D0    t3: 000000D0    t4: 00D0CBA0
t5: 000002BC    t6: 00000008
```

测试阶段2：t6 = 0x9

```
RV32I Single Cycle CPU

pc: 000002F8    inst: 01A92023  csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
x0: 000000000000ra: FFFFFFFF 000sp: 00000000 000gp: 40000000    tp: 40000000
t0: F8000000    t1: C0000000    t2: 80000000    s0: 00000001    s1: 00000001
a0: 00000000    a1: C0000000    a2: 00000001    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000020    s3: 00000000
s4: 00000004    s5: 00000010    s6: 00000008    s7: 00000010    s8: 00000004
s9: 00000008    s10:00000008    s11:000000D0    t3: 000000D0    t4: 00D0CBA0
t5: 000002D4    t6: 00000009
```

测试终局0x666:t6 = 0x666

```
RV32I Single Cycle CPU

pc: 00000008    inst: 00000013  csr_ctrl: 0     csr_r_data: 00000000
mstatus: 00000000       mcause: 00000000        mepc: 00000000  mtval: 00000000
x0: 000000000000ra: FFFFFFFF 000sp: 00000000 000gp: 40000000    tp: 40000000
t0: F8000000    t1: C0000000    t2: 80000000    s0: 00000001    s1: 00000001
a0: 00000000    a1: C0000000    a2: 00000001    a3: 00000000    a4: 00000000
a5: 00000000    a6: 00000000    a7: 00000000    s2: 00000020    s3: 00000000
s4: 00000004    s5: 00000010    s6: 00000008    s7: 00000010    s8: 00000004
s9: 00000008    s10:00000008    s11:00000008    t3: 000000D0    t4: 00D0CBA0
t5: 000002EC    t6: 00000666
```

3 讨论、心得

本次流水线工程难度不大，重点是读懂数据通路，依照自己的数据通路执行即可，且控制信号大部分与单周期一致，比较容易上手，没有想象中复杂。重点是数据冒险和结构冒险的处理，通过数据通路和时钟边沿信号事件分析，运用数据前递和插入stall等方法解决即可。

在本次实验刚开始忘记写IF_ID流水线寄存器，导致最后的CPU为四级流水线，后来改正之后化为五级流水线，虽然CPI小有提升，但是结构层次更为明晰。

一个verilog文件中只能有对同一信号的一次边沿触发:在寄存器模块中上升沿置零下降沿写入不可行，导致寄存器值没变化。这也是之前Lab4实验中上板失败的原因(发现了就不算晚)。

感谢所有在实验中帮助我的同学、助教大Q&瓜豪，在流水线学习中感谢刘海风老师的悉心教导，计组的实验就做到这里，是一次很愉快的思想碰撞。

今天是5月20日情人节，祝天下有情人终成眷属，有爱者依然有爱，无爱者自由！

纵有千重浪，心有不灭光！

4 思考题

4.1 Thinking-1

基于你完成的流水线，对于以下两段代码分别分析：不同指令之间是否存在冲突（如果有，请逐条列出）、在你的流水线上运行的 CPI 为何。

4.1.1 TP-0

```
addi    x1, x0, 0
addi    x2, x0, -1
addi    x3, x0, 1
addi    x4, x0, -1
addi    x5, x0, 1
addi    x6, x0, -1
addi    x1, x1, 0
addi    x2, x2, 1
addi    x3, x3, -1
addi    x4, x4, 1
addi    x5, x5, -1
addi    x6, x6, 1
```

不存在冲突

$CPI = \frac{12+4}{12} = 1.33$

4.1.2 TP-1

```
  addi    x1, x0, 1
  addi    x2, x1, 2
  addi    x3, x1, 3
  addi    x4, x3, 4
```

```
#冲突1：
addi    x1, x0, 1
addi    x2, x1, 2
#x1作为第一步的写入寄存器在MEM阶段还未及时更新，下一步的ALU计算已经要用
```

```
#冲突2：
addi    x1, x0, 1
addi    x2, x1, 2
addi    x3, x1, 3
#x1作为第一步的写入寄存器在WB阶段还未及时更新，第三步的ALU计算已经要用
```

```
#冲突3：
addi    x3, x1, 3
addi    x4, x3, 4
#x3作为第一步的写入寄存器在MEM阶段还未及时更新，下一步的ALU计算已经要用
```

$$CPI = \frac{4+4}{4} = 2$$

4.2 Thinking-2

请根据你的实现，在 testbench 上仿真以下代码，给出仿真结果，并写出完成所有指令用了多少拍，必须给出的信号有 clk, IF-PC, ID-PC 以及所有用到的寄存器值。**请务必注意调整数制为十六进制，缩放能够看到所有信号值！！！**

如果你实现的流水线支持 Forwarding：

```
#TP-3
addi    x1, x0, 1 #x1 = 0x1
addi    x2, x1, 2 #x2 = 0x3
addi    x3, x2, 3 #x3 = 0x6
sw      x3, 0(x0) #memory[0] = 0x6
lw      x4, 0(x0) #x4 = 0x6
addi    x5, x4, 4 #x5 = 0xa
addi    x6, x4, 5 #x1 = 0xb
```

使用周期数:$7+4=11$