

## 李毅PB22051031

### 第一部分 实验内容

在本次实验中，我们尝试将上一次实验设计的 CPU 流水化，形成一个不考虑冒险的流水线 CPU。

#### 1.前递模块

根据实验文档的内容，根据传入的信号与优先级的判断，正确进行前递模块的设计。

#### 2.加入前递的流水线

将前递模块正确接入 CPU，形成加入前递的流水线，并通过对应仿真测试。

#### 3.段间寄存器控制模块

根据实验文档的内容，根据传入的信号，正确根据输入信号进行段间寄存器的控制模块设计。

#### 4.完整流水线 CPU

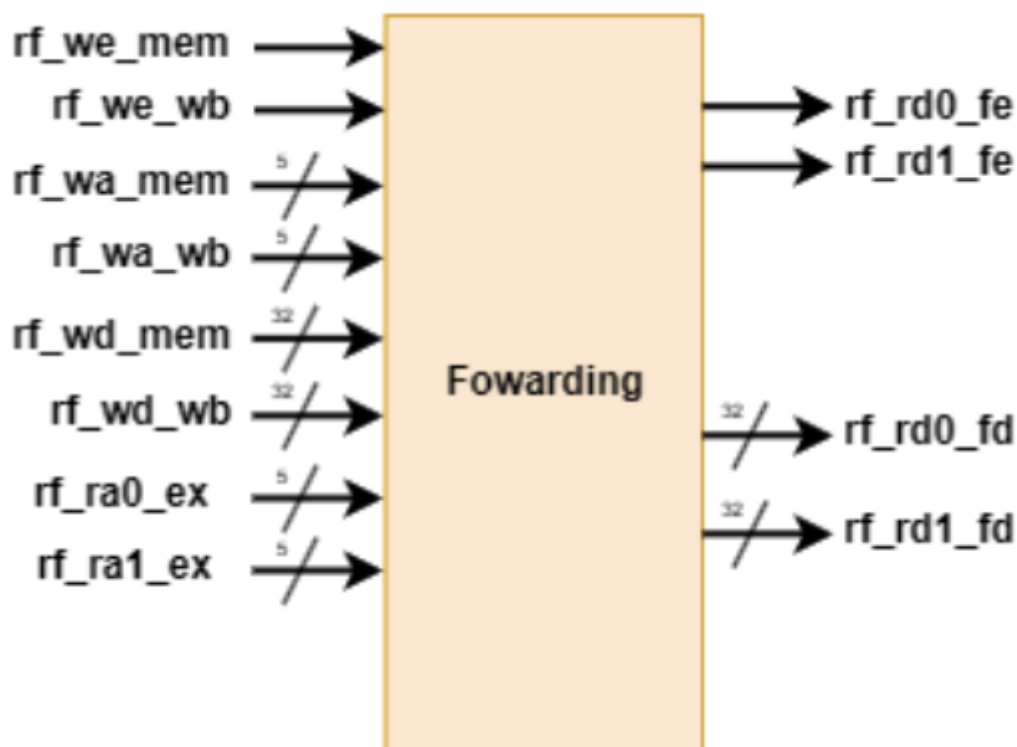
将段间寄存器控制模块正确接入 CPU，形成最终的流水线 CPU，并通过仿真、上板测试。

### 第二部分 实验过程

#### 2.1 实验设计

##### 2.1.1 前递模块

数据通路：

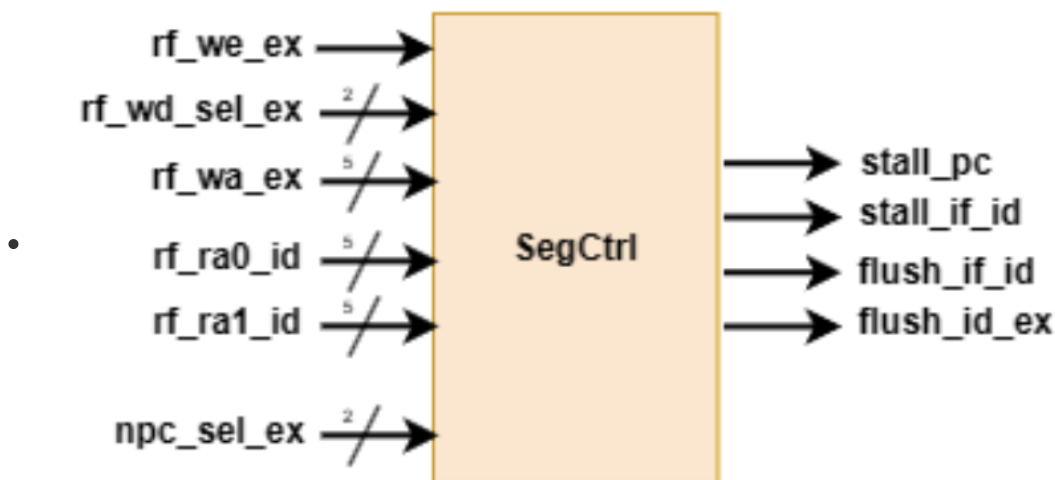


- rf\_we 指寄存器堆写使能信号，WB 与 MEM 段都需传入前递单元。

- rf\_wa 指寄存器堆写地址，WB 与 MEM 段都需传入前递单元，用于比对。
- rf\_wd 指寄存器堆写数据，在 WB 段直接传入即可，在 MEM 段可将 alu\_res 传入（见下方解释）。
- rf\_ra0\_ex 与 rf\_ra1\_ex 为本次实验中要求传入 EX 段的信号，用于与 MEM 段、WB 段信号比对确定前递是否发生。
- rf\_rd0\_fe 与 rf\_rd1\_fe 为前递使能信号，为 1 代表前递发生。
- rf\_rd0\_fd 与 rf\_rd1\_fd 为前递数据信号，当前递使能为 1 时即代表前递的数据。

通过两级判断，EX/MEM冒险和EX/WB冒险。都发生时，EX/MEM冒险优先级较高。

### 2.1.2 段间寄存器控制模块



- rf\_we 指寄存器堆写使能信号，EX 段需要传入以判断 Load-Use 冒险。
- rf\_wd\_sel 指寄存器堆写数据的选择，EX 段需要传入以判断 Load-Use 冒险。
- rf\_wa 指寄存器堆写地址，EX 段需要传入以判断 Load-Use 冒险。
- rf\_ra0\_id 与 rf\_ra1\_id 用于比对确定 Load-Use 冒险是否发生。
- npc\_sel 为下个 PC 选择器的控制信号，用于确定控制冒险是否发生。
- 输出信号为各段间寄存器对应的 stall 与 flush 信号。
- 

通过两级判断解决解决 Load-Use 冒险与控制冒险。

## 2.2 核心代码

### 2.2.1 前递模块

```
module Forwarding(
    input [0:0] rf_we_mem,
    input [0:0] rf_we_wb,

    input [4:0] rf_wa_mem,
    input [4:0] rf_wa_wb,

    input [31:0] rf_wd_mem,
    input [31:0] rf_wd_wb,

    input [4:0] rf_ra0_ex,
```

```

    input [4:0] rf_ra1_ex,

    output reg [0:0] rf_rd0_fe,
    output reg [0:0] rf_rd1_fe,

    output reg [31:0] rf_rd0_fd,
    output reg [31:0] rf_rd1_fd
);

always @(*) begin
    if(rf_we_mem && rf_wa_mem!=0 && (rf_wa_mem==rf_ra0_ex))begin
        rf_rd0_fe=1'b1;
        rf_rd0_fd=rf_wd_mem;
    end
    else if(rf_we_wb && rf_wa_wb!=0 && (rf_wa_wb==rf_ra0_ex))begin
        rf_rd0_fe=1'b1;
        rf_rd0_fd=rf_wd_wb;
    end
    else begin
        rf_rd0_fe=1'b0;
    end

    if(rf_we_mem && rf_wa_mem!=0 && (rf_wa_mem==rf_ra1_ex))begin
        rf_rd1_fe=1'b1;
        rf_rd1_fd=rf_wd_mem;
    end
    else if(rf_we_wb && rf_wa_wb!=0 && (rf_wa_wb==rf_ra1_ex))begin
        rf_rd1_fe=1'b1;
        rf_rd1_fd=rf_wd_wb;
    end
    else begin
        rf_rd1_fe=1'b0;
    end
end

endmodule

```

### 2.2.2 段间寄存器控制模块

```

module Segctr(
    input [0:0] rf_we_ex,
    input [1:0] rf_wd_sel_ex,
    input [4:0] rf_wa_ex,
    input [4:0] rf_ra0_id,
    input [4:0] rf_ra1_id,
    input [1:0] npc_sel_ex,

    output reg [0:0] stall_pc,
    output reg [0:0] stall_if_id,
    output reg [0:0] flush_if_id,
    output reg [0:0] flush_id_ex
);

always @(*) begin
    if(rf_we_ex && rf_wa_ex!=0 && rf_wd_sel_ex==2'b10 && ((rf_wa_ex==rf_ra0_id)||
(rf_wa_ex==rf_ra1_id)))begin

```

```

        stall_pc=1'b1;
        stall_if_id=1'b1;
        flush_id_ex=1'b1;
        flush_if_id=1'b0;
    end
    else if(npc_sel_ex==2'b01||npc_sel_ex==2'b10)begin
        stall_pc=1'b0;
        stall_if_id=1'b0;
        flush_id_ex=1'b1;
        flush_if_id=1'b1;
    end
    else begin
        stall_pc=1'b0;
        stall_if_id=1'b0;
        flush_id_ex=1'b0;
        flush_if_id=1'b0;
    end
end
endmodule

```

### 2.2.3 完整流水线 CPU数据通路

```

wire [ 0 : 0]  commit_if;
wire [ 0 : 0]  commit_id;
wire [ 0 : 0]  commit_ex;
wire [ 0 : 0]  commit_mem;
wire [ 0 : 0]  commit_wb;

wire [0:0] stall;
wire [0:0] flush;
wire [0:0] stall_pc;
wire [0:0] stall_if_id;
wire [0:0] flush_if_id;
wire [0:0] flush_id_ex;
wire [31:0] pcadd4_if;
wire [31:0] inst_if;
wire [31:0] pc_if;
wire [31:0] pcadd4_id;
wire [31:0] pc_id;
wire [31:0] pc_mem;
wire [31:0] pc_wb;
wire [31:0] inst_id;
wire [31:0] inst_ex;
wire [31:0] inst_mem;
wire [31:0] inst_wb;
wire [31:0] npc_ex;

assign stall=0;
assign flush=0;

PC mypc(
    .clk(clk),
    .rst(rst),
    .stall(stall_pc),
    .flush(flush),

```

```

        .en(global_en),
        .npc(npc_ex),
        .pc(pc_if)
    );

    ADD4 add4(
        .pc(pc_if),
        .npc(pcadd4_if)
    );

    assign imem_raddr=pc_if;
    assign inst_if=imem_rdata;

    Intersegment_reg IF2ID(
        .clk(clk),
        .rst(rst),
        .en(global_en),
        .stall(stall_if_id),
        .flush(flush_if_id),
        .pcadd4_in(pcadd4_if),
        .inst_in(inst_if),
        .pc_in(pc_if),
        .rf_rd0_in(32'b0),
        .rf_rd1_in(32'b0),
        .imm_in(32'b0),
        .rf_wa_in(5'b0),
        .alu_res_in(32'b0),
        .dmem_rd_out_in(32'b0),
        .alu_op_in(5'b0),
        .alu_src0_sel_in(1'b0),
        .alu_src1_sel_in(1'b0),
        .rf_we_in(1'b0),
        .br_type_in(4'b0),
        .dmem_access_in(4'b0),
        .rf_wd_sel_in(2'b0),
        .commit_in(commit_if),
        .dmem_we_in(1'b0),

        .rf_ra0_in(5'b0),
        .rf_ra1_in(5'b0),

        .pcadd4_out(pcadd4_id),
        .pc_out(pc_id),
        .inst_out(inst_id),
        .commit_out(commit_id)
    );

    wire [4:0] alu_op_id;
    wire [3:0] dmem_access_id;
    wire [0:0] dmem_we_id;
    wire [0:0] dmem_we_ex;
    wire [0:0] dmem_we_mem;
    wire [31:0] imm_id;
    wire [4:0] rf_ra0_id;
    wire [4:0] rf_ra1_id;
    wire [4:0] rf_ra0_ex;
    wire [4:0] rf_ra1_ex;

```

```

wire [4:0] rf_wa_id;
wire [0:0] rf_we_id;
wire [1:0] rf_wd_sel_id;
wire [0:0] alu_src0_sel_id;
wire [0:0] alu_src1_sel_id;
wire [3:0] br_type_id;
wire [4:0] rf_wa_wb;
wire [0:0] rf_we_wb;
wire [31:0] rf_wd_wb;
wire [31:0] rf_rd0_id;
wire [31:0] rf_rd1_id;
wire [31:0] pcadd4_ex;
wire [31:0] pc_ex;
wire [31:0] rf_rd0_ex;
wire [31:0] rf_rd1_ex;
wire [31:0] rf_rd0_raw_ex;
wire [31:0] rf_rd1_raw_ex;
wire [31:0] imm_ex;
wire [4:0] rf_wa_ex;
wire [4:0] alu_op_ex;
wire [0:0] alu_src0_sel_ex;
wire [0:0] alu_src1_sel_ex;
wire [0:0] rf_we_ex;
wire [3:0] br_type_ex;
wire [3:0] dmem_access_ex;
wire [1:0] rf_wd_sel_ex;

```

```

DECODER decoder(
    .inst(inst_id),
    .alu_op(alu_op_id),
    .dmem_access(dmem_access_id),
    .dmem_we(dmem_we_id),
    .imm(imm_id),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_id),
    .rf_we(rf_we_id),
    .rf_wd_sel(rf_wd_sel_id),
    .alu_src0_sel(alu_src0_sel_id),
    .alu_src1_sel(alu_src1_sel_id),
    .br_type(br_type_id)
);

```

```

REG_FILE reg_file(
    .clk(clk),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_wb),
    .rf_we(rf_we_wb),
    .rf_wd(rf_wd_wb),
    .rf_rd0(rf_rd0_id),
    .rf_rd1(rf_rd1_id),
    .debug_reg_ra(debug_reg_ra),
    .debug_reg_rd(debug_reg_rd)
);

```

```

Intersegment_reg ID2EX(

```

```

.clk(clk),
.rst(rst),
.en(global_en),
.stall(stall),
.flush(flush_id_ex),

.pcadd4_in(pcadd4_id),
.inst_in(inst_id),
.pc_in(pc_id),
.rf_rd0_in(rf_rd0_id),
.rf_rd1_in(rf_rd1_id),
.imm_in(imm_id),
.rf_wa_in(rf_wa_id),
.alu_res_in(32'b0),
.dmem_rd_out_in(32'b0),
.alu_op_in(alu_op_id),
.alu_src0_sel_in(alu_src0_sel_id),
.alu_src1_sel_in(alu_src1_sel_id),
.rf_we_in(rf_we_id),
.br_type_in(br_type_id),
.dmem_access_in(dmem_access_id),
.rf_wd_sel_in(rf_wd_sel_id),
.commit_in(commit_id),
.dmem_we_in(dmem_we_id),

.rf_ra0_in(rf_ra0_id),
.rf_ra1_in(rf_ra1_id),

.rf_ra0_out(rf_ra0_ex),
.rf_ra1_out(rf_ra1_ex),

.pcadd4_out(pcadd4_ex),
.inst_out(inst_ex),
.pc_out(pc_ex),
.rf_rd0_out(rf_rd0_raw_ex),
.rf_rd1_out(rf_rd1_raw_ex),
.imm_out(imm_ex),
.rf_wa_out(rf_wa_ex),
.alu_op_out(alu_op_ex),
.alu_src0_sel_out(alu_src0_sel_ex),
.alu_src1_sel_out(alu_src1_sel_ex),
.rf_we_out(rf_we_ex),
.br_type_out(br_type_ex),
.dmem_access_out(dmem_access_ex),
.rf_wd_sel_out(rf_wd_sel_ex),
.dmem_we_out(dmem_we_ex),
.commit_out(commit_ex)
);

wire [31:0] alu_src0_ex;
wire [31:0] alu_src1_ex;
wire [31:0] alu_res_ex;
wire [1:0] npc_sel_ex;
wire [31:0] pc_j_ex;

MUX1 rf_rd0_pc(
    .src0(rf_rd0_ex),
    .src1(pc_ex),

```

```

        .sel(alu_src0_sel_ex),
        .res(alu_src0_ex)
    );

MUX1 rf_rd1_imm(
    .src0(rf_rd1_ex),
    .src1(imm_ex),
    .sel(alu_src1_sel_ex),
    .res(alu_src1_ex)
);

ALU alu(
    .alu_src0(alu_src0_ex),
    .alu_src1(alu_src1_ex),
    .alu_op(alu_op_ex),
    .alu_res(alu_res_ex)
);

BRANCH branch(
    .br_type(br_type_ex),
    .br_src0(rf_rd0_ex),
    .br_src1(rf_rd1_ex),
    .npc_sel(npc_sel_ex)
);

assign pc_j_ex=alu_res_ex & (~32'b1);

NPCMUX npcmux(
    .pc_add4(pcadd4_if),
    .pc_offset(alu_res_ex),
    .pc_j(pc_j_ex),
    .npc_sel(npc_sel_ex),
    .res(npc_ex),
    .rst(rst)
);

wire [31:0] pcadd4_mem;
wire [31:0] alu_res_mem;
wire [31:0] rf_rd1_mem;
wire [4:0] rf_wa_mem;
wire [0:0] rf_we_mem;
wire [3:0] dmem_access_mem;
wire [1:0] rf_wd_sel_mem;

Intersegment_reg EX2MEM(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .stall(stall),
    .flush(flush),

    .pcadd4_in(pcadd4_ex),
    .inst_in(inst_ex),
    .pc_in(pc_ex),
    .rf_rd0_in(32'b0),
    .rf_rd1_in(rf_rd1_ex),
    .imm_in(32'b0),
    .rf_wa_in(rf_wa_ex),

```



```

.alu_res_in(alu_res_ex),
.dmem_rd_out_in(32'b0),
.alu_op_in(5'b0),
.alu_src0_sel_in(1'b0),
.alu_src1_sel_in(1'b0),
.rf_we_in(rf_we_ex),
.br_type_in(4'b0),
.dmem_access_in(dmem_access_ex),
.rf_wd_sel_in(rf_wd_sel_ex),
.commit_in(commit_ex),
.dmem_we_in(dmem_we_ex),

.rf_ra0_in(5'b0),
.rf_ra1_in(5'b0),

.pcadd4_out(pcadd4_mem),
.inst_out(inst_mem),
.pc_out(pc_mem),
.rf_rd1_out(rf_rd1_mem),
.rf_wa_out(rf_wa_mem),
.alu_res_out(alu_res_mem),
.rf_we_out(rf_we_mem),
.dmem_access_out(dmem_access_mem),
.rf_wd_sel_out(rf_wd_sel_mem),
.commit_out(commit_mem),
.dmem_we_out(dmem_we_mem)
);

wire [31:0] dmem_wd_in_mem;
wire [31:0] dmem_rd_out_mem;
wire [31:0] dmem_rd_in_mem;
wire [31:0] dmem_wd_out_mem;

assign dmem_addr=alu_res_mem;
assign dmem_wd_in_mem=rf_rd1_mem;
assign dmem_wdata=dmem_wd_out_mem;
assign dmem_rd_in_mem=dmem_rdata;
assign dmem_we=dmem_we_mem;
SLU slu(
    .addr(dmem_addr),
    .wd_in(dmem_wd_in_mem),
    .rd_out(dmem_rd_out_mem),
    .rd_in(dmem_rd_in_mem),
    .wd_out(dmem_wd_out_mem),
    .dmem_access(dmem_access_mem)
);

wire [31:0] pcadd4_wb;
wire [31:0] alu_res_wb;
wire [1:0] rf_wd_sel_wb;
wire [31:0] dmem_rd_out_wb;
wire [0:0] dmem_we_wb;
Intersegment_reg MEM2WB(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .stall(stall),
    .flush(flush),

```

```

.pcadd4_in(pcadd4_mem),
.inst_in(inst_mem),
.pc_in(pc_mem),
.rf_rd0_in(32'b0),
.rf_rd1_in(32'b0),
.imm_in(32'b0),
.rf_wa_in(rf_wa_mem),
.alu_res_in(alu_res_mem),
.dmem_rd_out_in(dmem_rd_out_mem),
.alu_op_in(5'b0),
.alu_src0_sel_in(1'b0),
.alu_src1_sel_in(1'b0),
.rf_we_in(rf_we_mem),
.br_type_in(4'b0),
.dmem_access_in(4'b0),
.rf_wd_sel_in(rf_wd_sel_mem),
.commit_in(commit_mem),

.rf_ra0_in(5'b0),
.rf_ra1_in(5'b0),

.dmem_we_in(dmem_we_mem),

.pcadd4_out(pcadd4_wb),
.inst_out(inst_wb),
.pc_out(pc_wb),
.alu_res_out(alu_res_wb),
.rf_wa_out(rf_wa_wb),
.rf_we_out(rf_we_wb),
.rf_wd_sel_out(rf_wd_sel_wb),
.dmem_rd_out_out(dmem_rd_out_wb),
.commit_out(commit_wb),
.dmem_we_out(dmem_we_wb)
);

MUX2 mux2(
    .src0(pcadd4_wb),
    .src1(alu_res_wb),
    .src2(dmem_rd_out_wb),
    .src3(32'b0),
    .res(rf_wd_wb),
    .sel(rf_wd_sel_wb)
);

wire [0:0] rf_rd0_fe;
wire [0:0] rf_rd1_fe;
wire [31:0] rf_rd0_fd;
wire [31:0] rf_rd1_fd;

Forwarding forwarding(
    .rf_we_mem(rf_we_mem),
    .rf_we_wb(rf_we_wb),
    .rf_wa_mem(rf_wa_mem),
    .rf_wa_wb(rf_wa_wb),
    .rf_wd_mem(alu_res_mem),
    .rf_wd_wb(rf_wd_wb),
    .rf_ra0_ex(rf_ra0_ex),
    .rf_ra1_ex(rf_ra1_ex),

```

```

        .rf_rd0_fe(rf_rd0_fe),
        .rf_rd1_fe(rf_rd1_fe),
        .rf_rd0_fd(rf_rd0_fd),
        .rf_rd1_fd(rf_rd1_fd)
    );

    MUX1 rf_rd0(
        .src0(rf_rd0_raw_ex),
        .src1(rf_rd0_fd),
        .sel(rf_rd0_fe),
        .res(rf_rd0_ex)
    );

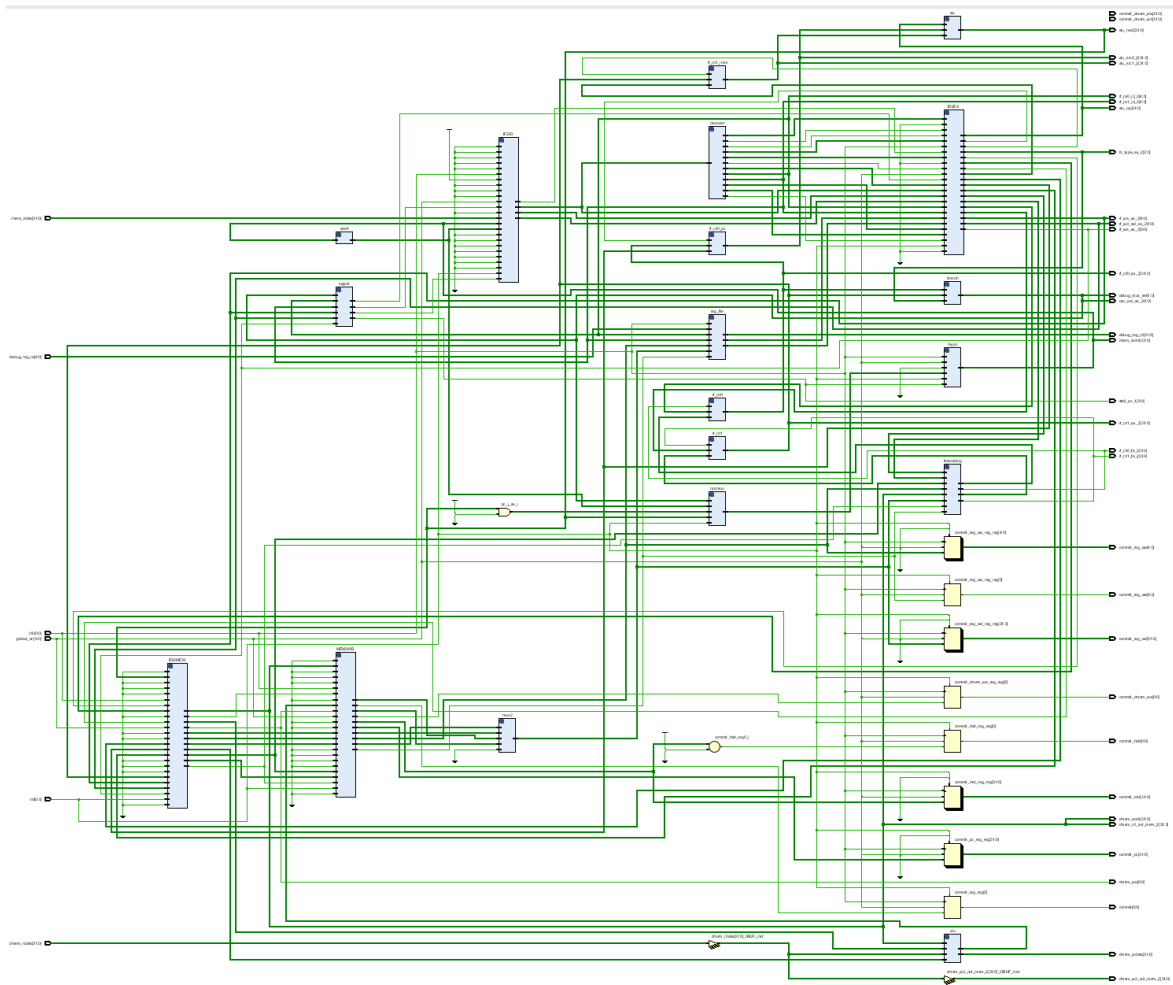
    MUX1 rf_rd1(
        .src0(rf_rd1_raw_ex),
        .src1(rf_rd1_fd),
        .sel(rf_rd1_fe),
        .res(rf_rd1_ex)
    );

    Segctr segctr(
        .rf_we_ex(rf_we_ex),
        .rf_wd_sel_ex(rf_wd_sel_ex),
        .rf_wa_ex(rf_wa_ex),
        .rf_ra0_id(rf_ra0_id),
        .rf_ra1_id(rf_ra1_id),
        .npc_sel_ex(npc_sel_ex),
        .stall_pc(stall_pc),
        .stall_if_id(stall_if_id),
        .flush_if_id(flush_if_id),
        .flush_id_ex(flush_id_ex)
    );

```

## 第三部分 实验结果

### 3.1 电路分析结果（文档尾部有高清图）



## 3.2 电路仿真结果

仿真文件如下：

```
module CPU_tb();

    reg [0:0] clk;
    reg [0:0] rst;
    reg [0:0] global_en;
    wire [31:0] imem_raddr;
    wire [31:0] imem_rdata;
    reg [0:0] we;
    reg [31:0] d;

    wire [31 : 0] dmem_rdata; // Unused
    wire [ 0 : 0] dmem_we;    // Unused
    wire [31 : 0] dmem_raddr; // Unused
    wire [31 : 0] dmem_wdata; // Unused

    reg [ 4 : 0] debug_reg_ra;
    wire [31 : 0] debug_reg_rd;

    wire [ 0 : 0] commit;
    wire [31 : 0] commit_pc;
    wire [31 : 0] commit_inst;
    wire [ 0 : 0] commit_halt;
    wire [ 0 : 0] commit_reg_we;
    wire [ 4 : 0] commit_reg_wa;
    wire [31 : 0] commit_reg_wd;
```

```

wire [0 : 0]          commit_dmem_we;
wire [31 : 0]         commit_dmem_wa;
wire [31 : 0]         commit_dmem_wd;

wire [31:0]          alu_res2;
wire [4:0]           alu_op2;
wire [31:0]          alu_src0_2;
wire [31:0]          alu_src1_2;
wire [1:0]           debug_mux_sel;

wire [0:0] stall_pc_1;
wire [1:0] rf_wd_sel_ex_2;
wire [4:0] rf_wa_ex_2;
wire [4:0] rf_ra0_id_2;
wire [4:0] rf_ra1_id_2;

wire [0:0] rf_rd0_fe_2;
wire [0:0] rf_rd1_fe_2;
wire [0:0] rf_we_ex_2;

wire [31:0] rf_rd0_ex_2;
wire [31:0] rf_rd1_ex_2;

wire [3:0] br_type_ex_2;
wire [1:0] npc_sel_ex_2;

wire [31:0] dmem_rd_out_mem_2;
wire [31:0] dmem_wd_out_mem_2;

INST_MEM mem (
    .a(imem_raddr[10:2]),      // input wire [8 : 0] a
    .d(d),                    // input wire [31 : 0] d
    .clk(clk),                // input wire clk
    .we(we),                  // input wire we
    .spo(imem_rdata)          // output wire [31 : 0] spo
);

DATA_MEM mem2 (
    .a(dmem_raddr >> 2),      // input wire [8 : 0] a
    .d(dmem_wdata),           // input wire [31 : 0] d
    .clk(clk),                // input wire clk
    .we(dmem_we),             // input wire we
    .spo(dmem_rdata)          // output wire [31 : 0] spo
);

CPU cpu(
    .clk(clk),
    .rst(rst),
    .global_en(global_en),
    .imem_rdata(imem_rdata),
    .imem_raddr(imem_raddr),
    .dmem_addr(dmem_raddr),
    .dmem_rdata(dmem_rdata),
    .dmem_wdata(dmem_wdata),
    .dmem_we(dmem_we),
    .debug_reg_ra(debug_reg_ra),
    .debug_reg_rd(debug_reg_rd),
    .commit(commit),

```

```

.commit_pc(commit_pc),
.commit_inst(commit_inst),
.commit_halt(commit_halt),
.commit_reg_we(commit_reg_we),
.commit_reg_wa(commit_reg_wa),
.commit_reg_wd(commit_reg_wd),
.commit_dmem_wa(commit_dmem_wa),
.commit_dmem_wd(commit_dmem_wd),
.commit_dmem_we(commit_dmem_we),
.alu_res2(alu_res2),
.alu_op2(alu_op2),
.alu_src0_2(alu_src0_2),
.alu_src1_2(alu_src1_2),
.debug_mux_sel(debug_mux_sel),
.stall_pc_1(stall_pc_1),
.rf_wd_sel_ex_2(rf_wd_sel_ex_2),
.rf_we_ex_2(rf_we_ex_2),
.rf_wa_ex_2(rf_wa_ex_2),
.rf_ra0_id_2(rf_ra0_id_2),
.rf_ra1_id_2(rf_ra1_id_2),
.rf_rd0_fe_2(rf_rd0_fe_2),
.rf_rd1_fe_2(rf_rd1_fe_2),
.rf_rd0_ex_2(rf_rd0_ex_2),
.rf_rd1_ex_2(rf_rd1_ex_2),
.br_type_ex_2(br_type_ex_2),
.npc_sel_ex_2(npc_sel_ex_2),
.dmem_rd_out_mem_2(dmem_rd_out_mem_2),
.dmem_wd_out_mem_2(dmem_wd_out_mem_2)
);

```

```

initial begin
    clk = 0;
    global_en=1;
    #1
    rst = 1;
    #1
    rst = 0;
    we = 0;
    d=32'b0;
    #1150
    debug_reg_ra=0;
    #1
    repeat(32) begin
        #0.1
        debug_reg_ra=debug_reg_ra+1;
    end
    $finish;
end

always #1 clk = ~clk;
endmodule

```

3.2.1 测试程序1

仿真结果如下：

> 🐞 debug_reg_ra[4:0]	00	00	01	02	03	04	05	06	07	08
> 🐞 debug_reg_rd[31:0]	b006f628	00000000	00000001	000006e3	f4b2e614	ffffffff	00000000	00000004	7f77ffff	dffcffd5

> 🐞 debug_reg_ra[4:0]	08	09	0a	0b	0c	0d	0e	0f	10	11
> 🐞 debug_reg_rd[31:0]	dffcffd5	60cbe000	006ffe00	f1165000	f4b2e614	7594d000	00000000	0a438000	00000000	60cbe000

> 🐞 debug_reg_ra[4:0]	08	12	13	14	15	16	17	18	19	1a
> 🐞 debug_reg_rd[31:0]	dffcffd5	00000001	006ffe00	b474b650	00000000	c2a83000	9b4ad664	00000000	b474b650	a08b002a

> 🐞 debug_reg_ra[4:0]	08	17	18	19	1a	1b	1c	1d	1e	1f
> 🐞 debug_reg_rd[31:0]	dffcffd5	9b4ad664	00000000	b474b650	a08b002a	00000004	97f8e63c	b006f628	9b4ad664	b006f628

RARS运行结果如下：

zero	0	0x00000000
ra	1	0x00000001
sp	2	0x000006e3
gp	3	0xf4b2e614
tp	4	0xffffffff
t0	5	0x00000000
t1	6	0x00000004
t2	7	0x7f77ffff
s0	8	0xdffeffd5
s1	9	0x60cbe000
a0	10	0x006ffeff
a1	11	0xf1165000
a2	12	0xf4b2e614
a3	13	0x7594d000
a4	14	0x00000000
a5	15	0x0a438000
a6	16	0x0a438000
a7	17	0x60cbe000
s2	18	0x00000001
s3	19	0x006ffeff
s4	20	0xb474b650
s5	21	0x00000000
s6	22	0xc2a83000
s7	23	0x9b4ad664
s8	24	0x00000000
s9	25	0xb474b650
s10	26	0xa08b002a
s11	27	0x00000004
t3	28	0x97f8e63c
t4	29	0xb006f628
t5	30	0x9b4ad664
t6	31	0xb006f628
pc		0x004006a0


保持一致。

### 3.2.2 测试程序2



仿真结果如下：

> debug_reg_ra[4:0]	07	00	01	02	03	04	05	06	07	08
> debug_reg_rd[31:0]	00000000	00000000	1002017c	00000000	00000000	10020b53	10015b8d	00000000		
> debug_reg_ra[4:0]	07	09	0a	0b	0c	0d	0e	0f	10	11
> debug_reg_rd[31:0]	00000000	00000000	10020277	1001c80d	00000000	0000003e	10026042	00000000		



>  debug_reg_ra[4:0]	07	12	13	14	15	16	17	18	19	1a
>  debug_reg_rd[31:0]	00000000	00000000	1001edad	00000000	0000003e	10035516	10035cac	0000007d		

>  debug_reg_ra[4:0]	07	17	18	19	1a	1b	1c	1d	1e	1f
>  debug_reg_rd[31:0]	00000000	0000003e	10035516	10035cac	0000007d	1003b6b4	00000000	1001ce8	0000007d	

RARS运行结果如下：

Name	Number	Value
zero	0	0x00000000
ra	1	0x1002017c
sp	2	0x00000000
gp	3	0x00000000
tp	4	0x00000000
t0	5	0x10020b53
t1	6	0x10015b8d
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x10020277
a1	11	0x1001c80d
a2	12	0x00000000
a3	13	0x0000003e
a4	14	0x10026042
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x1001edad
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x0000003e
s8	24	0x10035516
s9	25	0x10035cac
s10	26	0x0000007d
s11	27	0x1003b6b4
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x10011ce8
t6	31	0x0000007d
pc		0x00400758

保持一致。

### 3.3 上板结果

#### 3.3.1 测试程序2

FPGAOL运行结果:

![image-20240430172642563](C:\Users\Liyi\AppData\Roaming\Typora\typora-user-images\image-20240430172642563.png)

```
USTC COD Project
User: RR;
R;

USTC COD Project
User: R;
----- CPU -----

User: RR 0 16;
00000000 1002017C 00000000 00000000
00000000 10020B53 10015B8D 00000000
00000000 00000000 10020277 1001C80D
00000000 0000003E 10026042 00000000

User: RR 10 16;
00000000 00000000 00000000 00000000
1001EDAD 00000000 00000000 0000003E
10035516 10035CAC 0000007D 1003B6B4
00000000 00000000 10011CE8 0000007D

User:

uart pins:  cts    rts    rxd    txd
xdc sym:   D3     E5     D4     C4
baud rate: 115200
```

RARS结果:

Name	Number	Value
zero	0	0x00000000
ra	1	0x1002017c
sp	2	0x00000000
gp	3	0x00000000
tp	4	0x00000000
t0	5	0x10020b53
t1	6	0x10015b8d
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x10020277
a1	11	0x1001e80d
a2	12	0x00000000
a3	13	0x0000003e
a4	14	0x10026042
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x1001edad
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x0000003e
s8	24	0x10035516
s9	25	0x10035cac
s10	26	0x0000007d
s11	27	0x1003b6b4
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x10011ce8
t6	31	0x0000007d
pc		0x00400758

## 第四部分 心得体会

- 1.加深了流水线CPU原理的理解
- 2.对vivado, FPGAOI平台的使用 (debug) 熟练度进一步提升

## 第五部分 附件

### 1.CPU.v

```
`include "../include/config.v"

`define ADD          5'B00000
`define SUB          5'B00010
`define SLT          5'B00100
`define SLTU         5'B00101
`define AND          5'B01001
`define OR           5'B01010
`define XOR          5'B01011
`define SLL          5'B01110
`define SRL          5'B01111
`define SRA          5'B10000
`define SRC0         5'B10001
`define SRC1         5'B10010

module CPU (
    input          [ 0 : 0]      clk,
    input          [ 0 : 0]      rst,

    input          [ 0 : 0]      global_en,

    /* ----- Memory (inst) ----- */
    output         [31 : 0]      imem_raddr,
    input          [31 : 0]      imem_rdata,

    /* ----- Memory (data) ----- */
    input          [31 : 0]      dmem_rdata, // Unused
    output         [ 0 : 0]      dmem_we,    // Unused
    output         [31 : 0]      dmem_addr,  // Unused
    output         [31 : 0]      dmem_wdata, // Unused

    /* ----- Debug ----- */
    output         [ 0 : 0]      commit,
    output         [31 : 0]      commit_pc,
    output         [31 : 0]      commit_inst,
    output         [ 0 : 0]      commit_halt,
    output         [ 0 : 0]      commit_reg_we,
    output         [ 4 : 0]      commit_reg_wa,
    output         [31 : 0]      commit_reg_wd,
    output         [ 0 : 0]      commit_dmem_we,
    output         [31 : 0]      commit_dmem_wa,
    output         [31 : 0]      commit_dmem_wd,

    output         [31:0]      alu_res2,
    output         [4:0]      alu_op2,
    output         [31:0]      alu_src0_2,
    output         [31:0]      alu_src1_2,

    input          [ 4 : 0]      debug_reg_ra,
    output         [31 : 0]      debug_reg_rd,

    output         [1:0]      debug_mux_sel,
```

```

output          [0:0]    stall_pc_1,
output          [0:0]    rf_we_ex_2,
output          [1:0]    rf_wd_sel_ex_2,
output          [4:0]    rf_wa_ex_2,
output          [4:0]    rf_ra0_id_2,
output          [4:0]    rf_ra1_id_2,
output          [0:0]    rf_rd0_fe_2,
output          [0:0]    rf_rd1_fe_2,
output          [31:0]   rf_rd0_ex_2,
output          [31:0]   rf_rd1_ex_2,
output          [3:0]    br_type_ex_2,
output          [1:0]    npc_sel_ex_2,
output          [31:0]   dmem_rd_out_mem_2,
output          [31:0]   dmem_wd_out_mem_2
//output        [31:0]    alu_src0_2,
//output        [31:0]    alu_res2
);

/* ----- */
/*
wire [31:0] cur_npc;
wire [31:0] cur_pc;
wire [31:0] cur_inst;
wire [4:0] rf_ra0;
wire [4:0] rf_ra1;
wire [4:0] rf_wa;
wire [31:0] rf_wd;
wire [31:0] rf_rd0;
wire [31:0] rf_rd1;
wire [0:0] rf_we;
wire [4:0] alu_op;
wire [31:0] alu_src0;
wire [31:0] alu_src1;
wire [31:0] imm;
wire [ 0 : 0] alu_src0_sel;
wire [ 0 : 0] alu_src1_sel;
wire [ 4 : 0] debug_reg_ra;
wire [31 : 0] debug_reg_rd;
wire [0:0] we;
wire [8:0] a;
wire [31:0] d;
wire [31:0] alu_res;
wire [3:0] dmem_access;
wire [1:0] rf_wd_sel;
wire [3:0] br_type;
wire [1:0] npc_sel;
wire [31:0] pc_add4;
wire [31:0] dmem_wd_in;
wire [31:0] dmem_rd_out;
wire [31:0] dmem_wd_out;
wire [31:0] dmem_rd_in;
//wire [31:0] dmem_wa;
//wire [31:0] dmem_wd;
wire [31:0] pc_j;

assign we=0;
assign d=32'b0;

```

```

//assign dmem_wa=dmem_addr;
//assign dmem_wd=dmem_rdata;

PC my_pc (
    .clk      (clk      ),
    .rst      (rst      ),
    .en       (global_en ),    // 当 global_en 为高电平时，PC 才会更新，CPU 才会执
行指令。
    .npc      (cur_npc   ),
    .pc       (cur_pc    )
);

ADD4 add(
    .pc(cur_pc),
    .npc(pc_add4)
);

assign dmem_wd_in=rf_rd1;
assign imem_raddr=cur_pc;
assign cur_inst=imem_rdata;
assign dmem_addr=alu_res;
assign dmem_wdata=dmem_wd_out;
assign dmem_rd_in=dmem_rdata;
//assign alu_src0_2=alu_src0;
//assign alu_res2=alu_res;
DECODER decoder(
    .inst(cur_inst),
    .alu_op(alu_op),
    .imm(imm),
    .rf_ra0(rf_ra0),
    .rf_ra1(rf_ra1),
    .rf_wa(rf_wa),
    .rf_we(rf_we),
    .alu_src0_sel(alu_src0_sel),
    .alu_src1_sel(alu_src1_sel),
    .dmem_access(dmem_access),
    .rf_wd_sel(rf_wd_sel),
    .br_type(br_type),
    .dmem_we(dmem_we)
);

REG_FILE reg_file(
    .clk(clk),
    .rf_ra0(rf_ra0),
    .rf_ra1(rf_ra1),
    .rf_wa(rf_wa),
    .rf_we(rf_we),
    .rf_wd(rf_wd),
    .rf_rd0(rf_rd0),
    .rf_rd1(rf_rd1),
    .debug_reg_ra(debug_reg_ra),
    .debug_reg_rd(debug_reg_rd)
);

MUX1 rf_rd0_pc(
    .src0(rf_rd0),
    .src1(cur_pc),
    .sel(alu_src0_sel),

```

```

        .res(alu_src0)
    );

MUX1 rf_rd1_imm(
    .src0(rf_rd1),
    .src1(imm),
    .sel(alu_src1_sel),
    .res(alu_src1)
);

ALU alu(
    .alu_src0(alu_src0),
    .alu_src1(alu_src1),
    .alu_op(alu_op),
    .alu_res(alu_res)
);

BRANCH branch(
    .br_src0(rf_rd0),
    .br_src1(rf_rd1),
    .br_type(br_type),
    .npc_sel(npc_sel)
);

NPCMUX npcmux(
    .pc_add4(pc_add4),
    .pc_offset(alu_res),
    .pc_j(pc_j),
    .npc_sel(npc_sel),
    .res(cur_npc)
);

SLU slu(
    .addr(dmem_addr),
    .wd_in(dmem_wd_in),
    .rd_out(dmem_rd_out),
    .rd_in(dmem_rd_in),
    .wd_out(dmem_wd_out),
    .dmem_access(dmem_access)
);

MUX2 mux2(
    .src0(pc_add4),
    .src1(alu_res),
    .src2(dmem_rd_out),
    .src3(32'b0),
    .res(rf_wd),
    .sel(rf_wd_sel)
);

assign pc_j=alu_res & (~32'b1);
//assign rf_wd=alu_res;
assign alu_res2=alu_res;
assign alu_src0_2=alu_src0;
assign alu_src1_2=alu_src1;
assign alu_op2=alu_op;

```

```
// TODO
wire [ 0 : 0] commit_if;
wire [ 0 : 0] commit_id;
wire [ 0 : 0] commit_ex;
wire [ 0 : 0] commit_mem;
wire [ 0 : 0] commit_wb;
```

```
wire [0:0] stall;
wire [0:0] flush;
wire [0:0] stall_pc;
wire [0:0] stall_if_id;
wire [0:0] flush_if_id;
wire [0:0] flush_id_ex;
wire [31:0] pcadd4_if;
wire [31:0] inst_if;
wire [31:0] pc_if;
wire [31:0] pcadd4_id;
wire [31:0] pc_id;
wire [31:0] pc_mem;
wire [31:0] pc_wb;
wire [31:0] inst_id;
wire [31:0] inst_ex;
wire [31:0] inst_mem;
wire [31:0] inst_wb;
wire [31:0] npc_ex;
```

```
assign stall=0;
assign flush=0;
```

```
PC mypc(
    .clk(clk),
    .rst(rst),
    .stall(stall_pc),
    .flush(flush),
    .en(global_en),
    .npc(npc_ex),
    .pc(pc_if)
);
```

```
ADD4 add4(
    .pc(pc_if),
    .npc(pcadd4_if)
);
```

```
assign imem_raddr=pc_if;
assign inst_if=imem_rdata;
```

```
Intersegment_reg IF2ID(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .stall(stall_if_id),
    .flush(flush_if_id),
    .pcadd4_in(pcadd4_if),
    .inst_in(inst_if),
    .pc_in(pc_if),
    .rf_rd0_in(32'b0),
```



```

        .rf_rd1_in(32'b0),
        .imm_in(32'b0),
        .rf_wa_in(5'b0),
        .alu_res_in(32'b0),
        .dmem_rd_out_in(32'b0),
        .alu_op_in(5'b0),
        .alu_src0_sel_in(1'b0),
        .alu_src1_sel_in(1'b0),
        .rf_we_in(1'b0),
        .br_type_in(4'b0),
        .dmem_access_in(4'b0),
        .rf_wd_sel_in(2'b0),
        .commit_in(commit_if),
        .dmem_we_in(1'b0),

        .rf_ra0_in(5'b0),
        .rf_ra1_in(5'b0),

        .pcadd4_out(pcadd4_id),
        .pc_out(pc_id),
        .inst_out(inst_id),
        .commit_out(commit_id)
    );

```

```

wire [4:0] alu_op_id;
wire [3:0] dmem_access_id;
wire [0:0] dmem_we_id;
wire [0:0] dmem_we_ex;
wire [0:0] dmem_we_mem;
wire [31:0] imm_id;
wire [4:0] rf_ra0_id;
wire [4:0] rf_ra1_id;
wire [4:0] rf_ra0_ex;
wire [4:0] rf_ra1_ex;
wire [4:0] rf_wa_id;
wire [0:0] rf_we_id;
wire [1:0] rf_wd_sel_id;
wire [0:0] alu_src0_sel_id;
wire [0:0] alu_src1_sel_id;
wire [3:0] br_type_id;
wire [4:0] rf_wa_wb;
wire [0:0] rf_we_wb;
wire [31:0] rf_wd_wb;
wire [31:0] rf_rd0_id;
wire [31:0] rf_rd1_id;
wire [31:0] pcadd4_ex;
wire [31:0] pc_ex;
wire [31:0] rf_rd0_ex;
wire [31:0] rf_rd1_ex;
wire [31:0] rf_rd0_raw_ex;
wire [31:0] rf_rd1_raw_ex;
wire [31:0] imm_ex;
wire [4:0] rf_wa_ex;
wire [4:0] alu_op_ex;
wire [0:0] alu_src0_sel_ex;
wire [0:0] alu_src1_sel_ex;
wire [0:0] rf_we_ex;

```

```

wire [3:0] br_type_ex;
wire [3:0] dmem_access_ex;
wire [1:0] rf_wd_sel_ex;

```

```

DECODER decoder(
    .inst(inst_id),
    .alu_op(alu_op_id),
    .dmem_access(dmem_access_id),
    .dmem_we(dmem_we_id),
    .imm(imm_id),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_id),
    .rf_we(rf_we_id),
    .rf_wd_sel(rf_wd_sel_id),
    .alu_src0_sel(alu_src0_sel_id),
    .alu_src1_sel(alu_src1_sel_id),
    .br_type(br_type_id)
);

```

```

REG_FILE reg_file(
    .clk(clk),
    .rf_ra0(rf_ra0_id),
    .rf_ra1(rf_ra1_id),
    .rf_wa(rf_wa_wb),
    .rf_we(rf_we_wb),
    .rf_wd(rf_wd_wb),
    .rf_rd0(rf_rd0_id),
    .rf_rd1(rf_rd1_id),
    .debug_reg_ra(debug_reg_ra),
    .debug_reg_rd(debug_reg_rd)
);

```

```

Intersegment_reg ID2EX(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .stall(stall),
    .flush(flush_id_ex),

    .pcadd4_in(pcadd4_id),
    .inst_in(inst_id),
    .pc_in(pc_id),
    .rf_rd0_in(rf_rd0_id),
    .rf_rd1_in(rf_rd1_id),
    .imm_in(imm_id),
    .rf_wa_in(rf_wa_id),
    .alu_res_in(32'b0),
    .dmem_rd_out_in(32'b0),
    .alu_op_in(alu_op_id),
    .alu_src0_sel_in(alu_src0_sel_id),
    .alu_src1_sel_in(alu_src1_sel_id),
    .rf_we_in(rf_we_id),
    .br_type_in(br_type_id),
    .dmem_access_in(dmem_access_id),
    .rf_wd_sel_in(rf_wd_sel_id),
    .commit_in(commit_id),

```

```

        .dmem_we_in(dmem_we_id),

        .rf_ra0_in(rf_ra0_id),
        .rf_ra1_in(rf_ra1_id),

        .rf_ra0_out(rf_ra0_ex),
        .rf_ra1_out(rf_ra1_ex),

        .pcadd4_out(pcadd4_ex),
        .inst_out(inst_ex),
        .pc_out(pc_ex),
        .rf_rd0_out(rf_rd0_raw_ex),
        .rf_rd1_out(rf_rd1_raw_ex),
        .imm_out(imm_ex),
        .rf_wa_out(rf_wa_ex),
        .alu_op_out(alu_op_ex),
        .alu_src0_sel_out(alu_src0_sel_ex),
        .alu_src1_sel_out(alu_src1_sel_ex),
        .rf_we_out(rf_we_ex),
        .br_type_out(br_type_ex),
        .dmem_access_out(dmem_access_ex),
        .rf_wd_sel_out(rf_wd_sel_ex),
        .dmem_we_out(dmem_we_ex),
        .commit_out(commit_ex)
    );

    wire [31:0] alu_src0_ex;
    wire [31:0] alu_src1_ex;
    wire [31:0] alu_res_ex;
    wire [1:0] npc_sel_ex;
    wire [31:0] pc_j_ex;

    MUX1 rf_rd0_pc(
        .src0(rf_rd0_ex),
        .src1(pc_ex),
        .sel(alu_src0_sel_ex),
        .res(alu_src0_ex)
    );

    MUX1 rf_rd1_imm(
        .src0(rf_rd1_ex),
        .src1(imm_ex),
        .sel(alu_src1_sel_ex),
        .res(alu_src1_ex)
    );

    ALU alu(
        .alu_src0(alu_src0_ex),
        .alu_src1(alu_src1_ex),
        .alu_op(alu_op_ex),
        .alu_res(alu_res_ex)
    );

    BRANCH branch(
        .br_type(br_type_ex),
        .br_src0(rf_rd0_ex),
        .br_src1(rf_rd1_ex),
        .npc_sel(npc_sel_ex)
    );

```

```

);

assign pc_j_ex=alu_res_ex & (~32'b1);

NPCMUX npcmux(
    .pc_add4(pcadd4_if),
    .pc_offset(alu_res_ex),
    .pc_j(pc_j_ex),
    .npc_sel(npc_sel_ex),
    .res(npc_ex),
    .rst(rst)
);

wire [31:0] pcadd4_mem;
wire [31:0] alu_res_mem;
wire [31:0] rf_rd1_mem;
wire [4:0] rf_wa_mem;
wire [0:0] rf_we_mem;
wire [3:0] dmem_access_mem;
wire [1:0] rf_wd_sel_mem;

Intersegment_reg EX2MEM(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .stall(stall),
    .flush(flush),

    .pcadd4_in(pcadd4_ex),
    .inst_in(inst_ex),
    .pc_in(pc_ex),
    .rf_rd0_in(32'b0),
    .rf_rd1_in(rf_rd1_ex),
    .imm_in(32'b0),
    .rf_wa_in(rf_wa_ex),
    .alu_res_in(alu_res_ex),
    .dmem_rd_out_in(32'b0),
    .alu_op_in(5'b0),
    .alu_src0_sel_in(1'b0),
    .alu_src1_sel_in(1'b0),
    .rf_we_in(rf_we_ex),
    .br_type_in(4'b0),
    .dmem_access_in(dmem_access_ex),
    .rf_wd_sel_in(rf_wd_sel_ex),
    .commit_in(commit_ex),
    .dmem_we_in(dmem_we_ex),

    .rf_ra0_in(5'b0),
    .rf_ra1_in(5'b0),

    .pcadd4_out(pcadd4_mem),
    .inst_out(inst_mem),
    .pc_out(pc_mem),
    .rf_rd1_out(rf_rd1_mem),
    .rf_wa_out(rf_wa_mem),
    .alu_res_out(alu_res_mem),
    .rf_we_out(rf_we_mem),
    .dmem_access_out(dmem_access_mem),

```

```

        .rf_wd_sel_out(rf_wd_sel_mem),
        .commit_out(commit_mem),
        .dmem_we_out(dmem_we_mem)
    );

    wire [31:0] dmem_wd_in_mem;
    wire [31:0] dmem_rd_out_mem;
    wire [31:0] dmem_rd_in_mem;
    wire [31:0] dmem_wd_out_mem;

    assign dmem_addr=alu_res_mem;
    assign dmem_wd_in_mem=rf_rd1_mem;
    assign dmem_wdata=dmem_wd_out_mem;
    assign dmem_rd_in_mem=dmem_rdata;
    assign dmem_we=dmem_we_mem;
    SLU slu(
        .addr(dmem_addr),
        .wd_in(dmem_wd_in_mem),
        .rd_out(dmem_rd_out_mem),
        .rd_in(dmem_rd_in_mem),
        .wd_out(dmem_wd_out_mem),
        .dmem_access(dmem_access_mem)
    );

    wire [31:0] pcadd4_wb;
    wire [31:0] alu_res_wb;
    wire [1:0] rf_wd_sel_wb;
    wire [31:0] dmem_rd_out_wb;
    wire [0:0] dmem_we_wb;
    Intersegment_reg MEM2WB(
        .clk(clk),
        .rst(rst),
        .en(global_en),
        .stall(stall),
        .flush(flush),

        .pcadd4_in(pcadd4_mem),
        .inst_in(inst_mem),
        .pc_in(pc_mem),
        .rf_rd0_in(32'b0),
        .rf_rd1_in(32'b0),
        .imm_in(32'b0),
        .rf_wa_in(rf_wa_mem),
        .alu_res_in(alu_res_mem),
        .dmem_rd_out_in(dmem_rd_out_mem),
        .alu_op_in(5'b0),
        .alu_src0_sel_in(1'b0),
        .alu_src1_sel_in(1'b0),
        .rf_we_in(rf_we_mem),
        .br_type_in(4'b0),
        .dmem_access_in(4'b0),
        .rf_wd_sel_in(rf_wd_sel_mem),
        .commit_in(commit_mem),

        .rf_ra0_in(5'b0),
        .rf_ra1_in(5'b0),

        .dmem_we_in(dmem_we_mem),

```

```

        .pcadd4_out(pcadd4_wb),
        .inst_out(inst_wb),
        .pc_out(pc_wb),
        .alu_res_out(alu_res_wb),
        .rf_wa_out(rf_wa_wb),
        .rf_we_out(rf_we_wb),
        .rf_wd_sel_out(rf_wd_sel_wb),
        .dmem_rd_out_out(dmem_rd_out_wb),
        .commit_out(commit_wb),
        .dmem_we_out(dmem_we_wb)
    );

    MUX2 mux2(
        .src0(pcadd4_wb),
        .src1(alu_res_wb),
        .src2(dmem_rd_out_wb),
        .src3(32'b0),
        .res(rf_wd_wb),
        .sel(rf_wd_sel_wb)
    );

    wire [0:0] rf_rd0_fe;
    wire [0:0] rf_rd1_fe;
    wire [31:0] rf_rd0_fd;
    wire [31:0] rf_rd1_fd;

    Forwarding forwarding(
        .rf_we_mem(rf_we_mem),
        .rf_we_wb(rf_we_wb),
        .rf_wa_mem(rf_wa_mem),
        .rf_wa_wb(rf_wa_wb),
        .rf_wd_mem(alu_res_mem),
        .rf_wd_wb(rf_wd_wb),
        .rf_ra0_ex(rf_ra0_ex),
        .rf_ra1_ex(rf_ra1_ex),
        .rf_rd0_fe(rf_rd0_fe),
        .rf_rd1_fe(rf_rd1_fe),
        .rf_rd0_fd(rf_rd0_fd),
        .rf_rd1_fd(rf_rd1_fd)
    );

    MUX1 rf_rd0(
        .src0(rf_rd0_raw_ex),
        .src1(rf_rd0_fd),
        .sel(rf_rd0_fe),
        .res(rf_rd0_ex)
    );

    MUX1 rf_rd1(
        .src0(rf_rd1_raw_ex),
        .src1(rf_rd1_fd),
        .sel(rf_rd1_fe),
        .res(rf_rd1_ex)
    );

    Segctr segctr(
        .rf_we_ex(rf_we_ex),
        .rf_wd_sel_ex(rf_wd_sel_ex),

```

```

        .rf_wa_ex(rf_wa_ex),
        .rf_ra0_id(rf_ra0_id),
        .rf_ra1_id(rf_ra1_id),
        .npc_sel_ex(npc_sel_ex),
        .stall_pc(stall_pc),
        .stall_if_id(stall_if_id),
        .flush_if_id(flush_if_id),
        .flush_id_ex(flush_id_ex)
    );

    assign debug_mux_sel=npc_sel_ex;
    assign alu_res2=alu_res_ex;
    assign alu_src0_2=alu_src0_ex;
    assign alu_src1_2=alu_src1_ex;
    assign alu_op2=alu_op_ex;

    assign rf_rd0_ex_2=rf_rd0_ex;
    assign rf_rd1_ex_2=rf_rd1_ex;

    assign stall_pc_1=stall_pc;
    assign rf_wd_sel_ex_2=rf_wd_sel_ex;
    assign rf_we_ex_2=rf_we_ex;
    assign rf_wa_ex_2=rf_wa_ex;
    assign rf_ra0_id_2=rf_ra0_id;
    assign rf_ra1_id_2=rf_ra1_id;

    assign rf_rd0_fe_2=rf_rd0_fe;
    assign rf_rd1_fe_2=rf_rd1_fe;

    assign br_type_ex_2=br_type_ex;
    assign npc_sel_ex_2=npc_sel_ex;

    assign dmem_rd_out_mem_2=dmem_addr;
    assign dmem_wd_out_mem_2=dmem_rdata;
    /* ----- */
    /*                               Commit                               */
    /* ----- */

    reg [ 0 : 0]    commit_reg          ;
    reg [31 : 0]    commit_pc_reg       ;
    reg [31 : 0]    commit_inst_reg     ;
    reg [ 0 : 0]    commit_halt_reg     ;

    reg [ 0 : 0]    commit_reg_we_reg   ;
    reg [ 4 : 0]    commit_reg_wa_reg   ;
    reg [31 : 0]    commit_reg_wd_reg   ;
    reg [ 0 : 0]    commit_dmem_we_reg  ;
    reg [31 : 0]    commit_dmem_wa_reg  ;
    reg [31 : 0]    commit_dmem_wd_reg  ;

    assign commit_if = 1'H1;    // 这个信号需要经过 IF/ID、ID/EX、EX/MEM、MEM/WB 段间寄存器，最终连接到 commit_reg 上

    always @(posedge clk) begin
        if (rst) begin
            commit_reg          <= 1'H0;
            commit_pc_reg       <= 32'H0;
            commit_inst_reg     <= 32'H0;

```

```

        commit_halt_reg    <= 1'H0;
        commit_reg_we_reg  <= 1'H0;
        commit_reg_wa_reg  <= 5'H0;
        commit_reg_wd_reg  <= 32'H0;
        commit_dmem_we_reg <= 1'H0;
        commit_dmem_wa_reg <= 32'H0;
        commit_dmem_wd_reg <= 32'H0;
    end
    else if (global_en) begin
        // 这里右侧的信号都是 MEM/WB 段间寄存器的输出
        commit_reg          <= commit_wb;
        commit_pc_reg       <= pc_wb;
        commit_inst_reg     <= inst_wb;
        commit_halt_reg     <= inst_wb == 32'H00100073;
        commit_reg_we_reg   <= rf_we_wb;
        commit_reg_wa_reg   <= rf_wa_wb;
        commit_reg_wd_reg   <= rf_wd_wb;
        commit_dmem_we_reg  <= dmem_we_wb;
        //      commit_dmem_wa_reg <= dmem_addr_wb;
        //      commit_dmem_wd_reg <= dmem_wdata_wb;
    end
end

assign commit          = commit_reg;
assign commit_pc       = commit_pc_reg;
assign commit_inst     = commit_inst_reg;
assign commit_halt     = commit_halt_reg;
assign commit_reg_we   = commit_reg_we_reg;
assign commit_reg_wa   = commit_reg_wa_reg;
assign commit_reg_wd   = commit_reg_wd_reg;
assign commit_dmem_we  = commit_dmem_we_reg;
//assign commit_dmem_wa = commit_dmem_wa_reg;
//assign commit_dmem_wd = commit_dmem_wd_reg;

/* ----- */

/*单周期
assign commit_if = 1'H1;

reg  [ 0 : 0]  commit_reg          ;
reg  [31 : 0]  commit_pc_reg       ;
reg  [31 : 0]  commit_inst_reg     ;
reg  [ 0 : 0]  commit_halt_reg     ;

reg  [ 0 : 0]  commit_reg_we_reg   ;
reg  [ 4 : 0]  commit_reg_wa_reg   ;
reg  [31 : 0]  commit_reg_wd_reg   ;
reg  [ 0 : 0]  commit_dmem_we_reg  ;
reg  [31 : 0]  commit_dmem_wa_reg  ;
reg  [31 : 0]  commit_dmem_wd_reg  ;

always @(posedge clk) begin
    if (rst) begin
        commit_reg          <= 1'H0;

```



```

        commit_pc_reg      <= 32'H0;
        commit_inst_reg    <= 32'H0;
        commit_halt_reg    <= 1'H0;

        commit_reg_we_reg  <= 1'H0;
        commit_reg_wa_reg  <= 5'H0;
        commit_reg_wd_reg  <= 32'H0;
        commit_dmem_we_reg <= 1'H0;
        commit_dmem_wa_reg <= 32'H0;
        commit_dmem_wd_reg <= 32'H0;

    end
    else if (global_en) begin
        // !!!! 请注意根据自己的具体实现替换 <= 右侧的信号 !!!!
        commit_reg          <= 1'H1;                // 不需要改动
        commit_pc_reg       <= cur_pc;                // 需要为当前的 PC
        commit_inst_reg     <= cur_inst;              // 需要为当前的指令
        commit_halt_reg     <= cur_inst == 32'H00100073; // 注意! 请根据指令集设置 HALT_INST!

        commit_reg_we_reg   <= rf_we;                // 需要为当前的寄存器堆写使能
        commit_reg_wa_reg   <= rf_wa;                // 需要为当前的寄存器堆写地址
        commit_reg_wd_reg   <= rf_wd;                // 需要为当前的寄存器堆写数据
        commit_dmem_we_reg  <= dmem_we;              // 不需要改动
        commit_dmem_wa_reg  <= dmem_addr;            // 不需要改动
        commit_dmem_wd_reg  <= dmem_rdata;           // 不需要改动

    end
end

assign commit          = commit_reg;
assign commit_pc       = commit_pc_reg;
assign commit_inst     = commit_inst_reg;
assign commit_halt     = commit_halt_reg;

assign commit_reg_we   = commit_reg_we_reg;
assign commit_reg_wa   = commit_reg_wa_reg;
assign commit_reg_wd   = commit_reg_wd_reg;
assign commit_dmem_we  = commit_dmem_we_reg;
assign commit_dmem_wa  = commit_dmem_wa_reg;
assign commit_dmem_wd  = commit_dmem_wd_reg;
*/

endmodule

module DECODER (
    input          [31 : 0]      inst,

    output reg      [ 4 : 0]     alu_op,

    output reg      [ 3 : 0]     dmem_access, //访存
    output reg      [ 0 : 0]     dmem_we,

```

```

        output      reg      [31 : 0]      imm,

        output      [ 4 : 0]      rf_ra0,
        output      [ 4 : 0]      rf_ra1,
        output      [ 4 : 0]      rf_wa,
        output      reg      [ 0 : 0]      rf_we,
        output      reg      [ 1 : 0]      rf_wd_sel, //寄存器堆写回选择

        output      reg      [ 0 : 0]      alu_src0_sel,
        output      reg      [ 0 : 0]      alu_src1_sel,

        output      reg      [ 3 : 0]      br_type //分支跳转类型
    );

    wire [6:0] opcode;
    wire [6:0] funct7;
    wire [2:0] funct3;
    wire [9:0] funct;

    assign opcode=inst[6:0];
    assign funct3=inst[14:12];
    assign funct7=inst[31:25];
    assign funct={funct3[2:0],funct7[6:0]};

    //register
    assign rf_wa=inst[11:7];
    assign rf_ra0=inst[19:15];
    assign rf_ra1=inst[24:20];

    //imm
    always @(*) begin
        case (opcode)
            7'b0110111: imm={inst[31:12],12'b0}; //U type lui
            7'b0010111: imm={inst[31:12],12'b0}; //U type auipc
            7'b1101111: imm=
            {{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0}; //jal
            7'b1100111: imm={{20{inst[31]}},inst[31:20]}; //jalr
            7'b1100011: if(funct3==3'b110 || funct3==3'b111) begin
                imm=
            {{12{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //b type u
            end
            else begin
                imm=
            {{12{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //b type
            end
            7'b0000011: if(funct3==3'b100 || funct3==3'b101) begin
                imm={{20{inst[31]}},inst[31:20]}; //lbu, lhu
            end
            else begin
                imm={{20{inst[31]}},inst[31:20]}; //lb, lh, lw
            end
            7'b0100011: imm={{20{inst[31]}},inst[31:25],inst[11:7]}; //sw sb sh
            7'b0010011: if (funct3==3'b101 || funct3==3'b001) begin
                imm={{27{inst[24]}},inst[24:20]}; //srai
            end
            /*
            else if(funct3==3'b011)begin

```

```

        imm={20'b0,inst[31:20]};//sltiu
    end
    */
    else begin
        imm={{20{inst[31]}},inst[31:20]};
    end
    default: imm=32'b0;
endcase
end

//aluop
always @(*) begin
    case (opcode)
        7'b0110011: begin
            case (funct)
                10'b000_0000000: alu_op=5'B00000;//add
                10'b000_0100000: alu_op=5'B00010;//sub
                10'b001_0000000: alu_op=5'B01110;//sll
                10'b010_0000000: alu_op=5'B00100;//slt
                10'b011_0000000: alu_op=5'B00101;//sltiu
                10'b100_0000000: alu_op=5'B01011;//xor
                10'b101_0000000: alu_op=5'B01111;//srli
                10'b101_0100000: alu_op=5'B10000;//sra
                10'b110_0000000: alu_op=5'B01010;//or
                10'b111_0000000: alu_op=5'B01001;//and
                default: alu_op=5'b11111;
            endcase
        end
        7'b0010011: begin
            case (funct3)
                3'b000: alu_op=5'B00000;//addi
                3'b001: alu_op=5'B01110;//slli
                3'b010: alu_op=5'B00100;//slti
                3'b011: alu_op=5'B00101;//sltiu
                3'b100: alu_op=5'B01011;//xori
                3'b101: if(funct7==7'b0000000)begin
                    alu_op=5'B01111;//srli
                end
                else begin
                    alu_op=5'B10000;//srai
                end
                3'b110: alu_op=5'B01010;//ori
                3'b111: alu_op=5'B01001;//andi
                default: alu_op=5'b11111;
            endcase
        end
        7'b0110111: begin
            alu_op=5'B10010;//lui
        end
        7'b0010111: begin
            alu_op=5'b00000;//auipc
        end
        7'b0000011: begin
            alu_op=5'b00000;//lb, lh, lw, lbu, lhu
        end
        7'b1100011: begin
            alu_op=5'b00000;//b type to add
        end
    end
end

```

```

7'b110111:begin
    alu_op=5'b00000;//jal
end
7'b1100111:begin
    alu_op=5'b00000;//jalr
end
7'b0100011:begin
    alu_op=5'b00000;//sb,sh,sw
end
    default: alu_op=5'b00000;//else to add
endcase
end

//alu_src0_sel=0,来自ra0,alu_src0_sel=1,来自PC。alu_src1_sel=0,来自
ra1,alu_src0_sel=1,来自imm
always @(*) begin
    case (opcode)
        7'b0010111: alu_src0_sel=1; //auipc
        7'b0110111: alu_src0_sel=0; //lui
        7'b1101111: alu_src0_sel=1; //jal
        7'b1100111: alu_src0_sel=0; //jalr
        7'b1100011: alu_src0_sel=1; //B-type
        7'b0000011: alu_src0_sel=0; //I-type,load
        7'b0100011: alu_src0_sel=0; //S-type
        7'b0110011: alu_src0_sel=0; //R-type
        7'b0010011: alu_src0_sel=0; //I-type,imm
        default: alu_src0_sel=0;
    endcase
end

always @(*) begin
    case (opcode)
        7'b0010111: alu_src1_sel=1; //auipc
        7'b0110111: alu_src1_sel=1; //lui
        7'b1101111: alu_src1_sel=1; //jal
        7'b1100111: alu_src1_sel=1; //jalr
        7'b1100011: alu_src1_sel=1; //B-type
        7'b0000011: alu_src1_sel=1; //I-type,load
        7'b0100011: alu_src1_sel=1; //S-type
        7'b0110011: alu_src1_sel=0; //R-type
        7'b0010011: alu_src1_sel=1; //I-type,imm
        default: alu_src1_sel=0;
    endcase
end

//rf_we
always @(*) begin
    case (opcode)
        7'b0010111: rf_we=1; //auipc
        7'b0110111: rf_we=1; //lui
        7'b1101111: rf_we=1; //jal
        7'b1100111: rf_we=1; //jalr
        7'b1100011: rf_we=0; //B-type
        7'b0000011: rf_we=1; //I-type,load
        7'b0100011: rf_we=0; //S-type
        7'b0110011: rf_we=1; //R-type
        7'b0010011: rf_we=1; //I-type,imm
        default: rf_we=1;
    endcase
end

```

```

    endcase
end

//dmem_access
always @(*) begin
    case (opcode)
        7'b0000011:begin
            case (funct3)
                3'b000: dmem_access=4'b0001; //lb
                3'b001: dmem_access=4'b0011; //lh
                3'b010: dmem_access=4'b1111; //lw
                3'b100: dmem_access=4'b1000; //lbu
                3'b101: dmem_access=4'b1100; //lhu
                default: dmem_access=4'b0000;
            endcase
        end
        7'b0100011:begin
            case (funct3)
                3'b000: dmem_access=4'b0001; //sb
                3'b001: dmem_access=4'b0011; //sh
                3'b010: dmem_access=4'b1111; //sw
                default: dmem_access=4'b0000;
            endcase
        end
        default: dmem_access=4'b0000;
    endcase
end

//rf_wd_sel pc_add4:00 alu_res:01 dmem_rdata:10 ZERO:11
always @(*) begin
    case (opcode)
        7'b0010111: rf_wd_sel=2'b01; //auipc
        7'b0110111: rf_wd_sel=2'b01; //lui
        7'b1101111: rf_wd_sel=2'b00; //jal
        7'b1100111: rf_wd_sel=2'b00; //jalr
        7'b1100011: rf_wd_sel=2'b11; //B-type*not essential
        7'b0000011: rf_wd_sel=2'b10; //I-type,load
        7'b0100011: rf_wd_sel=2'b11; //S-type*not essential
        7'b0110011: rf_wd_sel=2'b01; //R-type
        7'b0010011: rf_wd_sel=2'b01; //I-type,imm
        default:rf_wd_sel=2'b11;
    endcase
end

//br_type
always @(*) begin
    case (opcode)
        7'b1100011: begin
            case (funct3)
                3'b000:br_type=4'b0100; //beq
                3'b001:br_type=4'b0001; //bne
                3'b100:br_type=4'b0010; //blt
                3'b101:br_type=4'b0011; //bge
                3'b110:br_type=4'b0110; //bltu
                3'b111:br_type=4'b0111; //bgeu
                default:br_type=4'b1111;
            endcase
        end
    end
end

```

```

        7'b1101111:br_type=4'b1000; //jal
        7'b1100111:br_type=4'b1000; //jalr
        default: br_type=4'b1111;
    endcase
end
//dmem_we
always @(*) begin
    case (opcode)
        7'b0100011: dmem_we=1;
        default: dmem_we=0;
    endcase
end

endmodule

module PC (
    input                [ 0 : 0]      clk,
    input                [ 0 : 0]      rst,
    input                [ 0 : 0]      stall,
    input                [ 0 : 0]      flush,
    input                [ 0 : 0]      en,
    input                [31 : 0]      npc,

    output reg           [31 : 0]      pc
);

always @(posedge clk or posedge rst) begin
    if(rst)begin
        pc <= 32'h00400000;
    end
    else if (en) begin
        if(flush)begin
            pc <= 32'h00400000;
        end
        else if (stall==1'b0)begin
            pc <= npc;
        end
        // flush 和 stall 操作的逻辑, flush 的优先级更高
    end
end

endmodule

module ALU (
    input                [31 : 0]      alu_src0,
    input                [31 : 0]      alu_src1,
    input                [ 4 : 0]      alu_op,

    output reg           [31 : 0]      alu_res
);
always @(*) begin
    case(alu_op)
        `ADD :
            alu_res = alu_src0 + alu_src1;
        `SUB :
            alu_res = alu_src0 - alu_src1;
        `SLT :

```

```

        alu_res = ($signed(alu_src0) < $signed(alu_src1)) ? 32'b1 :
32'b0;

        `SLTU :
            alu_res = (alu_src0 < alu_src1) ? 32'b1 : 32'b0;
        `AND :
            alu_res = alu_src0 & alu_src1;
        `OR :
            alu_res = alu_src0 | alu_src1;
        `XOR :
            alu_res = alu_src0 ^ alu_src1;
        `SLL :
            alu_res = alu_src0 << alu_src1[4:0];
        `SRL :
            alu_res = alu_src0 >> alu_src1[4:0];
        `SRA :
            alu_res = $signed(alu_src0) >>> $signed(alu_src1[4:0]);
        `SRC0 :
            alu_res = alu_src0;
        `SRC1 :
            alu_res = alu_src1;
        default :
            alu_res = 32'H0;
    endcase
end
endmodule

module REG_FILE (
    input                [ 0 : 0]      clk,

    input                [ 4 : 0]      rf_ra0,
    input                [ 4 : 0]      rf_ra1,
    input                [ 4 : 0]      rf_wa,
    input                [ 0 : 0]      rf_we,
    input                [31 : 0]      rf_wd,

    output               [31 : 0]      rf_rd0,
    output               [31 : 0]      rf_rd1,

    input                [ 4 : 0]      debug_reg_ra,
    output               [31 : 0]      debug_reg_rd
);

    reg [31 : 0] reg_file [0 : 31];

    // 用于初始化寄存器
    integer i;
    initial begin
        for (i = 0; i < 32; i = i + 1)
            reg_file[i] = 0;
        end

    assign rf_rd0= (rf_ra0 == 0) ? 0 : ( (rf_we && (rf_ra0 == rf_wa)) ? rf_wd :
reg_file[rf_ra0] );
    assign rf_rd1= (rf_ra1 == 0) ? 0 : ( (rf_we && (rf_ra1 == rf_wa)) ? rf_wd :
reg_file[rf_ra1] );
    assign debug_reg_rd=(debug_reg_ra == 0) ? 0 : ( (rf_we && (debug_reg_ra==
rf_wa)) ? rf_wd : reg_file[debug_reg_ra] );

```

```

        always@(negedge clk) begin
            if(rf_we && rf_wa!=0) begin
                reg_file[rf_wa]<=rf_wd;
            end
        end

endmodule

module MUX1 # (
    parameter WIDTH = 32
)(
    input [WIDTH-1 : 0] src0, src1,
    input [0 : 0] sel,

    output [WIDTH-1 : 0] res
);

    assign res = sel ? src1 : src0;

endmodule

module ADD4 (
    input [31:0] pc,
    output reg [31:0] npc
);
    always @(*) begin
        if(pc<=32'h0fffffff) begin
            npc=pc+32'h4;
        end
    end
end

endmodule

module BRANCH(
    input [3 : 0] br_type,

    input [31 : 0] br_src0,
    input [31 : 0] br_src1,

    output reg [1 : 0] npc_sel
);

    always @(*) begin
        case (br_type)
            4'b0100: npc_sel= (br_src0 == br_src1) ? 2'b01 : 2'b00;
            4'b0001: npc_sel= (br_src0 != br_src1) ? 2'b01 : 2'b00;
            4'b0010: npc_sel= ($signed(br_src0) < $signed(br_src1)) ? 2'b01 :
2'b00;
            4'b0011: npc_sel= ($signed(br_src0) >= $signed(br_src1)) ? 2'b01 :
2'b00;
            4'b0110: npc_sel= (br_src0 < br_src1) ? 2'b01 : 2'b00;
            4'b0111: npc_sel= (br_src0 >= br_src1) ? 2'b01 : 2'b00;
            4'b1000: npc_sel=2'b10;
            default: npc_sel=2'b00;
        endcase
    end
end

```



```

endmodule

module NPCMUX # (
    parameter WIDTH = 32
) (
    input [WIDTH-1 : 0] pc_add4, pc_offset, pc_j,
    input [ 1 : 0] npc_sel,
    input [0:0] rst,

    output reg [WIDTH-1 : 0] res
);

always @(*) begin
    if(rst==0)begin
        case (npc_sel)
            2'b00: res=pc_add4;
            2'b01: res=pc_offset;
            2'b10: res=pc_j;
            2'b11: res=32'b0;
            default: res=pc_add4;
        endcase
    end
    else begin
        res=pc_add4;
    end
end

endmodule

module SLU (
    input [31 : 0] addr,
    input [ 3 : 0] dmem_access,

    input [31 : 0] rd_in,
    input [31 : 0] wd_in,

    output reg [31 : 0] rd_out,
    output reg [31 : 0] wd_out
);

always @(*) begin
    case (dmem_access)
        4'b0001:begin
            case (addr[1:0])
                2'b00: begin
                    rd_out={{24{rd_in[7]}},rd_in[7:0]};
                    wd_out={rd_in[31:8],wd_in[7:0]};
                end
                2'b01: begin
                    rd_out={{24{rd_in[15]}},rd_in[15:8]};
                    wd_out={rd_in[31:16],wd_in[7:0],rd_in[7:0]};
                end
                2'b10: begin
                    rd_out={{24{rd_in[23]}},rd_in[23:16]};
                    wd_out=
{rd_in[31:24],wd_in[7:0],rd_in[15:0]};
                end
                2'b11: begin

```

```

        rd_out={{24{rd_in[31]}},rd_in[31:24]};
        wd_out={wd_in[7:0],rd_in[23:0]};

    end
    default:begin
        rd_out=rd_in;
        wd_out=wd_in;
    end
endcase
end
4'b1000:begin
    case (addr[1:0])
        2'b00: begin
            rd_out={24'b0,rd_in[7:0]};
            wd_out={rd_in[31:8],wd_in[7:0]};

        end
        2'b01: begin
            rd_out={24'b0,rd_in[15:8]};
            wd_out={rd_in[31:16],wd_in[7:0],rd_in[7:0]};

        end
        2'b10: begin
            rd_out={24'b0,rd_in[23:16]};
            wd_out=
{rd_in[31:24],wd_in[7:0],rd_in[15:0]};

        end
        2'b11: begin
            rd_out={24'b0,rd_in[31:24]};
            wd_out={wd_in[7:0],rd_in[23:0]};

        end
        default:begin
            rd_out=rd_in;
            wd_out=wd_in;
        end
    end
endcase
end
4'b1100:begin
    case (addr[1])
        1'b0:begin
            rd_out={16'b0,rd_in[15:0]};
            wd_out={rd_in[31:16],wd_in[15:0]};

        end
        1'b1:begin
            rd_out={16'b0,rd_in[31:16]};
            wd_out={wd_in[15:0],rd_in[15:0]};

        end
        default:begin
            rd_out=rd_in;
            wd_out=wd_in;
        end
    end
endcase
end
4'b0011:begin
    case (addr[1])
        1'b0:begin
            rd_out={{16{rd_in[15]}},rd_in[15:0]};
            wd_out={rd_in[31:16],wd_in[15:0]};

        end
        1'b1:begin
            rd_out={{16{rd_in[31]}},rd_in[31:16]};

```

```

        wd_out={wd_in[15:0],rd_in[15:0]};
    end
    default:begin
        rd_out=rd_in;
        wd_out=wd_in;
    end
endcase
end
4'b1111:begin
    rd_out=rd_in;
    wd_out=wd_in;
end
default:begin
    rd_out=rd_in;
    wd_out=wd_in;
end
endcase
end
endmodule

module MUX2 # (
    parameter WIDTH = 32
)(
    input [WIDTH-1 : 0] src0, src1, src2, src3,
    input [1 : 0] sel,

    output [WIDTH-1 : 0] res
);

    assign res = sel[1] ? (sel[0] ? src3 : src2) : (sel[0] ? src1 : src0);

endmodule

module Intersegment_reg(
    input [0:0] clk,
    input [0:0] rst,
    input [0:0] en,
    input [0:0] stall,
    input [0:0] flush,

    input [31:0] pcadd4_in,
    input [31:0] inst_in,
    input [31:0] pc_in,
    input [31:0] rf_rd0_in,
    input [31:0] rf_rd1_in,
    input [31:0] imm_in,
    input [4:0] rf_wa_in,
    input [31:0] alu_res_in,
    input [31:0] dmem_rd_out_in,

    input [4:0] rf_ra0_in,
    input [4:0] rf_ra1_in,

    input [0:0] dmem_we_in,

    input [4:0] alu_op_in,
    input [0:0] alu_src0_sel_in,

```

```

input [0:0] alu_src1_sel_in,
input [0:0] rf_we_in,
input [3:0] br_type_in,
input [3:0] dmem_access_in,
input [1:0] rf_wd_sel_in,

input [0:0] commit_in,

output reg [31:0] pcadd4_out,
output reg [31:0] inst_out,
output reg [31:0] pc_out,
output reg [31:0] rf_rd0_out,
output reg [31:0] rf_rd1_out,
output reg [31:0] imm_out,
output reg [4:0] rf_wa_out,
output reg [31:0] alu_res_out,
output reg [31:0] dmem_rd_out_out,

output reg [4:0] rf_ra0_out,
output reg [4:0] rf_ra1_out,

output reg [0:0] dmem_we_out,

output reg [4:0] alu_op_out,
output reg [0:0] alu_src0_sel_out,
output reg [0:0] alu_src1_sel_out,
output reg [0:0] rf_we_out,
output reg [3:0] br_type_out,
output reg [3:0] dmem_access_out,
output reg [1:0] rf_wd_sel_out,

output reg [0:0] commit_out

);

always @(posedge clk or posedge rst) begin
    if (rst) begin
        pcadd4_out=32'b0;
        inst_out=32'b0;
        pc_out=32'b0;
        rf_rd0_out=32'b0;
        rf_rd1_out=32'b0;
        imm_out=32'b0;
        rf_wa_out=5'b0;
        alu_res_out=32'b0;
        dmem_rd_out_out=32'b0;
        commit_out=1'b0;

        rf_ra0_out=5'b0;
        rf_ra1_out=5'b0;

        dmem_we_out=1'b0;

        alu_op_out=5'b0;
        alu_src0_sel_out=1'b0;
        alu_src1_sel_out=1'b0;
        rf_we_out=1'b0;
        br_type_out=4'b0;
    end
end

```

```

        dmem_access_out=4'b0;
        rf_wd_sel_out=2'b0;
    end
    else if (en) begin
        if(flush)begin
            pcadd4_out=32'b0;
            inst_out=32'b0;
            pc_out=32'b0;
            rf_rd0_out=32'b0;
            rf_rd1_out=32'b0;
            imm_out=32'b0;
            rf_wa_out=5'b0;
            alu_res_out=32'b0;
            dmem_rd_out_out=32'b0;
            commit_out=1'b0;

            rf_ra0_out=5'b0;
            rf_ra1_out=5'b0;

            dmem_we_out=1'b0;

            alu_op_out=5'b0;
            alu_src0_sel_out=1'b0;
            alu_src1_sel_out=1'b0;
            rf_we_out=1'b0;
            br_type_out=4'b0;
            dmem_access_out=4'b0;
            rf_wd_sel_out=2'b0;
        end
        else if (stall==1'b0)begin
            pcadd4_out=pcadd4_in;
            inst_out=inst_in;
            pc_out=pc_in;
            rf_rd0_out=rf_rd0_in;
            rf_rd1_out=rf_rd1_in;
            imm_out=imm_in;
            rf_wa_out=rf_wa_in;
            alu_res_out=alu_res_in;
            dmem_rd_out_out=dmem_rd_out_in;
            commit_out=commit_in;

            rf_ra0_out=rf_ra0_in;
            rf_ra1_out=rf_ra1_in;

            dmem_we_out=dmem_we_in;

            alu_op_out=alu_op_in;
            alu_src0_sel_out=alu_src0_sel_in;
            alu_src1_sel_out=alu_src1_sel_in;
            rf_we_out=rf_we_in;
            br_type_out=br_type_in;
            dmem_access_out=dmem_access_in;
            rf_wd_sel_out=rf_wd_sel_in;
        end
        // flush 和 stall 操作的逻辑, flush 的优先级更高
    end
end
endmodule

```

```

module Forwarding(
    input [0:0] rf_we_mem,
    input [0:0] rf_we_wb,

    input [4:0] rf_wa_mem,
    input [4:0] rf_wa_wb,

    input [31:0] rf_wd_mem,
    input [31:0] rf_wd_wb,

    input [4:0] rf_ra0_ex,
    input [4:0] rf_ra1_ex,

    output reg [0:0] rf_rd0_fe,
    output reg [0:0] rf_rd1_fe,

    output reg [31:0] rf_rd0_fd,
    output reg [31:0] rf_rd1_fd
);

always @(*) begin
    if(rf_we_mem && rf_wa_mem!=0 && (rf_wa_mem==rf_ra0_ex))begin
        rf_rd0_fe=1'b1;
        rf_rd0_fd=rf_wd_mem;
    end
    else if(rf_we_wb && rf_wa_wb!=0 && (rf_wa_wb==rf_ra0_ex))begin
        rf_rd0_fe=1'b1;
        rf_rd0_fd=rf_wd_wb;
    end
    else begin
        rf_rd0_fe=1'b0;
    end

    if(rf_we_mem && rf_wa_mem!=0 && (rf_wa_mem==rf_ra1_ex))begin
        rf_rd1_fe=1'b1;
        rf_rd1_fd=rf_wd_mem;
    end
    else if(rf_we_wb && rf_wa_wb!=0 && (rf_wa_wb==rf_ra1_ex))begin
        rf_rd1_fe=1'b1;
        rf_rd1_fd=rf_wd_wb;
    end
    else begin
        rf_rd1_fe=1'b0;
    end
end

endmodule

module Segctr(
    input [0:0] rf_we_ex,
    input [1:0] rf_wd_sel_ex,
    input [4:0] rf_wa_ex,
    input [4:0] rf_ra0_id,
    input [4:0] rf_ra1_id,
    input [1:0] npc_sel_ex,

```



```

module CPU_tb();

reg [0:0] clk;
reg [0:0] rst;
reg [0:0] global_en;
wire [31:0] imem_raddr;
wire [31:0] imem_rdata;
reg [0:0] we;
reg [31:0] d;

wire [31 : 0] dmem_rdata; // Unused
wire [ 0 : 0] dmem_we;    // Unused
wire [31 : 0] dmem_raddr; // Unused
wire [31 : 0] dmem_wdata; // Unused

reg [ 4 : 0] debug_reg_ra;
wire [31 : 0] debug_reg_rd;

wire [ 0 : 0]      commit;
wire [31 : 0]      commit_pc;
wire [31 : 0]      commit_inst;
wire [ 0 : 0]      commit_halt;
wire [ 0 : 0]      commit_reg_we;
wire [ 4 : 0]      commit_reg_wa;
wire [31 : 0]      commit_reg_wd;
wire [ 0 : 0]      commit_dmem_we;
wire [31 : 0]      commit_dmem_wa;
wire [31 : 0]      commit_dmem_wd;

wire [31:0]      alu_res2;
wire [4:0]       alu_op2;
wire [31:0]      alu_src0_2;
wire [31:0]      alu_src1_2;
wire [1:0]       debug_mux_sel;

wire [0:0] stall_pc_1;
wire [1:0] rf_wd_sel_ex_2;
wire [4:0] rf_wa_ex_2;
wire [4:0] rf_ra0_id_2;
wire [4:0] rf_ra1_id_2;

wire [0:0] rf_rd0_fe_2;
wire [0:0] rf_rd1_fe_2;
wire [0:0] rf_we_ex_2;

wire [31:0] rf_rd0_ex_2;
wire [31:0] rf_rd1_ex_2;

wire [3:0] br_type_ex_2;
wire[1:0] npc_sel_ex_2;

wire [31:0] dmem_rd_out_mem_2;
wire [31:0] dmem_wd_out_mem_2;

INST_MEM mem (
    .a(imem_raddr[10:2]), // input wire [8 : 0] a
    .d(d), // input wire [31 : 0] d
    .clk(clk), // input wire clk

```



```

        .we(we),      // input wire we
        .spo(imem_rdata) // output wire [31 : 0] spo
    );

DATA_MEM mem2 (
    .a(dmem_raddr >> 2),      // input wire [8 : 0] a
    .d(dmem_wdata),          // input wire [31 : 0] d
    .clk(clk), // input wire clk
    .we(dmem_we),           // input wire we
    .spo(dmem_rdata) // output wire [31 : 0] spo
);

CPU cpu(
    .clk(clk),
    .rst(rst),
    .global_en(global_en),
    .imem_rdata(imem_rdata),
    .imem_raddr(imem_raddr),
    .dmem_addr(dmem_raddr),
    .dmem_rdata(dmem_rdata),
    .dmem_wdata(dmem_wdata),
    .dmem_we(dmem_we),
    .debug_reg_ra(debug_reg_ra),
    .debug_reg_rd(debug_reg_rd),
    .commit(commit),
    .commit_pc(commit_pc),
    .commit_inst(commit_inst),
    .commit_halt(commit_halt),
    .commit_reg_we(commit_reg_we),
    .commit_reg_wa(commit_reg_wa),
    .commit_reg_wd(commit_reg_wd),
    .commit_dmem_wa(commit_dmem_wa),
    .commit_dmem_wd(commit_dmem_wd),
    .commit_dmem_we(commit_dmem_we),
    .alu_res2(alu_res2),
    .alu_op2(alu_op2),
    .alu_src0_2(alu_src0_2),
    .alu_src1_2(alu_src1_2),
    .debug_mux_sel(debug_mux_sel),
    .stall_pc_1(stall_pc_1),
    .rf_wd_sel_ex_2(rf_wd_sel_ex_2),
    .rf_we_ex_2(rf_we_ex_2),
    .rf_wa_ex_2(rf_wa_ex_2),
    .rf_ra0_id_2(rf_ra0_id_2),
    .rf_ra1_id_2(rf_ra1_id_2),
    .rf_rd0_fe_2(rf_rd0_fe_2),
    .rf_rd1_fe_2(rf_rd1_fe_2),
    .rf_rd0_ex_2(rf_rd0_ex_2),
    .rf_rd1_ex_2(rf_rd1_ex_2),
    .br_type_ex_2(br_type_ex_2),
    .npc_sel_ex_2(npc_sel_ex_2),
    .dmem_rd_out_mem_2(dmem_rd_out_mem_2),
    .dmem_wd_out_mem_2(dmem_wd_out_mem_2)
);

initial begin
    clk = 0;
    global_en=1;

```

```
#1
rst = 1;
#1
rst = 0;
we = 0;
d=32'b0;
#900
debug_reg_ra=0;
#1
repeat(32) begin
    #0.1
    debug_reg_ra=debug_reg_ra+1;
end
$finish;
end

always #1 clk = ~clk;
endmodule
```

