# 实验 6：完整流水线 CPU

张子康 PB22020660

2024 年 5 月 11 日

# 1 实验目的与内容

## 1.1 实验目的

将结合前递模块与段间寄存器控制模块，得到一个能正常运行的完整流水线 CPU。

## 1.2 实验内容

### 1.2.1 任务 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。

### 1.2.2 任务 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。

### 1.2.3 任务 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。

### 1.2.4 任务 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。

# 2 逻辑设计

## 2.1 任务 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 [0:0] rf_rd0_fe,
    output [0:0] rf_rd1_fe,
    output [31:0] rf_rd0_fd,
    output [31:0] rf_rd1_fd
);
    reg [31:0] rd0_fd,rd1_fd;
    reg [0:0] rd0_fe,rd1_fe;
    initial begin
        rd0_fd=0;
        rd0_fe=0;
        rd1_fd=0;
        rd1_fe=0;
    end

    assign rf_rd0_fe=rd0_fe;
    assign rf_rd1_fe=rd1_fe;
    assign rf_rd0_fd=rd0_fd;
    assign rf_rd1_fd=rd1_fd;

    always @(*) begin
        // 默认不前递
        rd0_fd=0;
        rd1_fd=0;
        rd0_fe=0;
        rd1_fe=0;
        // 如果发现写入的地址与读取的地址相同就前递
        if(rf_we_wb && rf_wa_wb!=5'b00000 && rf_wa_wb==rf_ra0_ex) begin
            rd0_fd=rf_wd_wb;
            rd0_fe=1;
        end
        if(rf_we_wb && rf_wa_wb!=5'b00000 && rf_wa_wb==rf_ra1_ex) begin
            rd1_fd=rf_wd_wb;
            rd1_fe=1;
        end
        if(rf_we_mem && rf_wa_mem!=5'b00000 && rf_wa_mem==rf_ra0_ex) begin
            rd0_fd=rf_wd_mem;
            rd0_fe=1;
        end
        if(rf_we_mem && rf_wa_mem!=5'b00000 && rf_wa_mem==rf_ra1_ex) begin
            rd1_fd=rf_wd_mem;
            rd1_fe=1;
        end
    end
endmodule
```

## 2.2 任务 2：加入前递的流水线

```
    // rf_rd0_ex_和rf_rd1_ex_为前递选择后的数据
    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(alu_res_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 mux3(
        .src1(rf_rd0_fd),
        .src0(rf_rd0_ex),
        .sel(rf_rd0_fe),
        .res(rf_rd0_ex_)
    );

    MUX1 mux4(
        .src1(rf_rd1_fd),
        .src0(rf_rd1_ex),
        .sel(rf_rd1_fe),
        .res(rf_rd1_ex_)
    );
```

将上述代码加入 CPU 模块，同时作出如下修改：

```
    MUX1 mux1(
        .src0(rf_rd0_ex_),
        .src1(pc_ex),
        .sel(alu_src0_sel_ex),
        .res(alu_src0_ex)
    );

    MUX1 mux2(
        .src0(rf_rd1_ex_),
        .src1(imm_ex),
        .sel(alu_src1_sel_ex),
        .res(alu_src1_ex)
    );

    EX_MEM ex_mem(
        .clk(clk),
        .rst(rst),
        .flush(flush_ex_mem),
        .stall(stall),
        .en(en),
        //PC
        .pc_add4_in(pcadd4_ex),
        .pc_in(pc_ex),
        .pc_add4_out(pcadd4_mem),
        .pc_out(pc_mem),
        //INST
        .inst_in(inst_ex),
        .inst_out(inst_mem),
        //DECODER
        .alu_op_in(alu_op_ex),
        .dmem_access_in(dmem_access_ex),
        .imm_in(imm_ex),
        .rf_ra0_in(rf_ra0_ex),
        .rf_ra1_in(rf_ra1_ex),
        .rf_wa_in(rf_wa_ex),
        .rf_we_in(rf_we_ex),
        .rf_wd_sel_in(rf_wd_sel_ex),
        .alu_src0_sel_in(alu_src0_sel_ex),
        .alu_src1_sel_in(alu_src1_sel_ex),
        .br_type_in(br_type_ex),
        .dmem_we_in(dmem_we_ex),
        .alu_op_out(alu_op_mem),
        .dmem_access_out(dmem_access_mem),
        .imm_out(imm_mem),
        .rf_ra0_out(rf_ra0_mem),
        .rf_ra1_out(rf_ra1_mem),
        .rf_wa_out(rf_wa_mem),
        .rf_we_out(rf_we_mem),
        .rf_wd_sel_out(rf_wd_sel_mem),
        .alu_src0_sel_out(alu_src0_sel_mem),
        .alu_src1_sel_out(alu_src1_sel_mem),
        .br_type_out(br_type_mem),
        .dmem_we_out(dmem_we_mem),
        //REG_FILE
        .rf_rd0_in(rf_rd0_ex_),
        .rf_rd1_in(rf_rd1_ex_),
        .rf_rd0_out(rf_rd0_mem),
        .rf_rd1_out(rf_rd1_mem),
        //MUX1
        .alu_src0_in(alu_src0_ex),
        .alu_src1_in(alu_src1_ex),
        .alu_src0_out(alu_src0_mem),
        .alu_src1_out(alu_src1_mem),
        //NPC
        .npc_in(npc_ex),
        .npc_out(npc_mem),
        //BRANCH
        .npc_sel_in(npc_sel_ex),
        .npc_sel_out(npc_sel_mem),
        //ALU
        .alu_res_in(alu_res_ex),
        .alu_res_out(alu_res_mem),
        //SLU
        .dmem_rd_out_in(0),
        .dmem_wdata_mem_in(0),
        .dmem_rd_out_out(),
        .dmem_wdata_mem_out(),
        //DM
        .dmem_rdata_mem_in(0),
        .dmem_rdata_mem_out(),
        .commit_in(commit_ex),
        .commit_out(commit_mem)
    );
```

## 2.3 任务 3：段间寄存器控制模块

```
`define DMEM_RDATA 2'b10
module SEG_CTRL (
    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
        // 默认不阻塞或冲刷流水线
        stall_pc=0;
        stall_if_id=0;
        flush_id_ex=0;
        flush_if_id=0;
        // 对于load-use，若写入地址和读取地址相同
        // 则阻塞流水线2周期
        if(rf_we_ex && rf_wd_sel_ex==`DMEM_RDATA && (
           rf_wa_ex==rf_ra0_id || rf_wa_ex==rf_ra1_id))
           begin
```

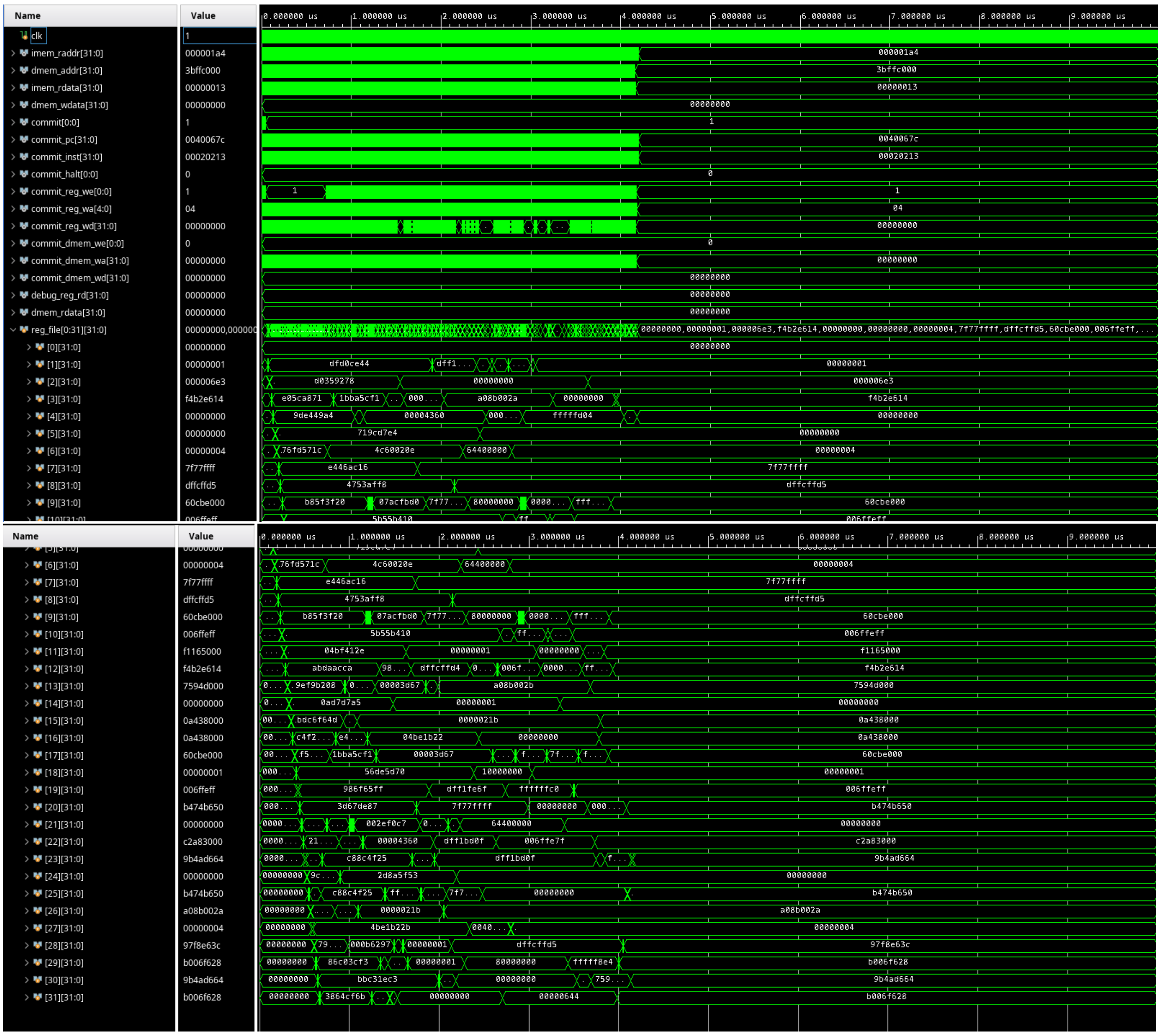
```
                flush_id_ex=1'b1;
                stall_pc=1'b1;
                stall_if_id=1'b1;
            end
            // 对于跳转指令，直接冲刷流水线
            if(npc_sel_ex!=2'b00) begin
                flush_id_ex=1'b1;
                flush_if_id=1'b1;
            end
        end
endmodule
```

## 2.4 任务 4：完整流水线 CPU

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

module CPU (input [0 : 0] clk,
            input [0 : 0] rst,
            input [0 : 0] global_en,
            output [31 : 0] imem_raddr,
            input [31 : 0] imem_rdata,
            input [31 : 0] dmem_rdata,       // Unused
            output [0 : 0] dmem_we,          // Unused
            output [31 : 0] dmem_addr,       // Unused
            output [31 : 0] dmem_wdata,      // Unused
            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,
                input [4 : 0] debug_reg_ra,
                output [31 : 0] debug_reg_rd);


    // TODO
    wire [0:0] commit_if;
    wire [31:0] pc_if;
    wire [31:0] pcadd4_if;
    wire [31:0] inst_if;

    wire [0:0] commit_id;
    wire [31:0] pc_id;
    wire [31:0] pcadd4_id;
    wire [31:0] inst_id;
    wire [4:0] alu_op_id;
    wire [3:0] dmem_access_id;
    wire [31:0] imm_id;
    wire [4:0] rf_ra0_id;
    wire [4:0] rf_ra1_id;
    wire [5:0] 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 [0:0] dmem_we_id;
    wire [31:0] rf_rd0_id;
    wire [31:0] rf_rd1_id;

    wire [0:0] commit_ex;
    wire [31:0] pc_ex;
    wire [31:0] pcadd4_ex;
    wire [31:0] inst_ex;
    wire [4:0] alu_op_ex;
    wire [3:0] dmem_access_ex;
    wire [31:0] imm_ex;
    wire [4:0] rf_ra0_ex;
    wire [4:0] rf_ra1_ex;
    wire [5:0] rf_wa_ex;
    wire [0:0] rf_we_ex;
    wire [1:0] rf_wd_sel_ex;
    wire [0:0] alu_src0_sel_ex;
    wire [0:0] alu_src1_sel_ex;
    wire [3:0] br_type_ex;
    wire [0:0] dmem_we_ex;
    wire [31:0] rf_rd0_ex;
    wire [31:0] rf_rd1_ex;
    wire [31:0] npc_ex;
    wire [31:0] pc_j_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 [0:0] commit_mem;
    wire [31:0] pc_mem;
    wire [31:0] pcadd4_mem;
    wire [31:0] inst_mem;
    wire [4:0] alu_op_mem;
    wire [3:0] dmem_access_mem;
    wire [31:0] imm_mem;
    wire [4:0] rf_ra0_mem;
    wire [4:0] rf_ra1_mem;
    wire [5:0] rf_wa_mem;
    wire [0:0] rf_we_mem;
    wire [1:0] rf_wd_sel_mem;
    wire [0:0] alu_src0_sel_mem;
    wire [0:0] alu_src1_sel_mem;
    wire [3:0] br_type_mem;
    wire [0:0] dmem_we_mem;
    wire [31:0] rf_rd0_mem;
    wire [31:0] rf_rd1_mem;
    wire [31:0] npc_mem;
    wire [31:0] pc_j_mem;
    wire [31:0] alu_src0_mem;
    wire [31:0] alu_src1_mem;
    wire [31:0] alu_res_mem;
    wire [31:0] dmem_rd_out_mem;
    wire [31:0] dmem_wdata_mem;
    wire [31:0] dmem_rdata_mem;
    wire [1:0] npc_sel_mem;

    wire [0:0] commit_wb;
    wire [31:0] pc_wb;
    wire [31:0] pcadd4_wb;
    wire [31:0] inst_wb;
    wire [4:0] alu_op_wb;
    wire [3:0] dmem_access_wb;
    wire [31:0] imm_wb;
    wire [4:0] rf_ra0_wb;
    wire [4:0] rf_ra1_wb;
    wire [5:0] rf_wa_wb;
    wire [0:0] rf_we_wb;
    wire [1:0] rf_wd_sel_wb;
    wire [0:0] alu_src0_sel_wb;
    wire [0:0] alu_src1_sel_wb;
    wire [3:0] br_type_wb;
    wire [0:0] dmem_we_wb;
    wire [31:0] rf_rd0_wb;
    wire [31:0] rf_rd1_wb;
    wire [31:0] npc_wb;
    wire [31:0] pc_j_wb;
    wire [31:0] alu_src0_wb;
    wire [31:0] alu_src1_wb;
    wire [31:0] alu_res_wb;
    wire [31:0] dmem_rd_out_wb;
    wire [31:0] dmem_wdata_wb;
    wire [31:0] dmem_rdata_wb;
    wire [31:0] rf_wd_wb;
    wire [1:0] npc_sel_wb;

    wire [0:0] rf_rd0_fe,rf_rd1_fe;
    wire [31:0] rf_rd0_fd,rf_rd1_fd;
    wire [31:0] rf_rd0_ex_,rf_rd1_ex_;

    wire flush,stall,stall_pc,stall_if_id,en;
    wire flush_if_id,flush_id_ex,flush_ex_mem,flush_mem_wb;

    assign commit_if = 1;
    assign stall = 0;
    assign en = global_en;
    assign global_en  = !(inst_wb == 32'H00100073);
    assign imem_raddr = (pc_if-32'h00400000)/'d4;
    assign inst_if    = imem_rdata;
    assign pc_j_ex       = alu_res_ex&~1;
    assign dmem_wd_in = rf_rd1_mem;
    assign dmem_we    = dmem_we_mem;
    assign dmem_addr  = (alu_res_mem-32'h10010000)/'d4;
    assign dmem_wdata = dmem_wdata_mem;
    assign dmem_rdata_mem = dmem_rdata;
    //assign flush_if_id=inst_if==32'h00000013 && inst_id
       ==32'h00000013;
    //assign flush_id_ex=inst_if==32'h00000013 && inst_id
       ==32'h00000013;
    assign flush_ex_mem=0;
    assign flush_mem_wb=0;

    // 前递模块
    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(alu_res_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 mux3(
        .src1(rf_rd0_fd),
        .src0(rf_rd0_ex),
        .sel(rf_rd0_fe),
        .res(rf_rd0_ex_)
    );

    MUX1 mux4(
        .src1(rf_rd1_fd),
        .src0(rf_rd1_ex),
        .sel(rf_rd1_fe),
        .res(rf_rd1_ex_)
    );

    // 段间寄存器控制器
    SEG_CTRL seg_ctrl(
        .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)
    );


    PC_PLUS4 pc_plus(
        .pc(pc_if),
        .pc_plus4(pcadd4_if)
    );

    PC 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    (npc_ex),
        .pc     (pc_if),
        .stall  (stall_pc),
        .flush(flush)
    );

    IF_ID if_id(
        .clk(clk),
        .rst(rst),
        .flush(flush_if_id),
        .stall(stall_if_id),
        .en(en),
        //PC
        .pc_add4_in(pcadd4_if),
        .pc_in(pc_if),
        .pc_add4_out(pcadd4_id),
        .pc_out(pc_id),
        //INST
        .inst_in(inst_if),
        .inst_out(inst_id),
        //DECODER
        .alu_op_in(0),
        .dmem_access_in(0),
        .imm_in(0),
        .rf_ra0_in(0),
        .rf_ra1_in(0),
        .rf_wa_in(0),
        .rf_we_in(0),
        .rf_wd_sel_in(0),
        .alu_src0_sel_in(0),
        .alu_src1_sel_in(0),
        .br_type_in(0),
        .dmem_we_in(0),
        .alu_op_out(),
        .dmem_access_out(),
        .imm_out(),
        .rf_ra0_out(),
        .rf_ra1_out(),
        .rf_wa_out(),
        .rf_we_out(),
        .rf_wd_sel_out(),
        .alu_src0_sel_out(),
        .alu_src1_sel_out(),
        .br_type_out(),
        .dmem_we_out(),
        //REG_FILE
        .rf_rd0_in(0),
        .rf_rd1_in(0),
        .rf_rd0_out(),
        .rf_rd1_out(),
        //MUX1
        .alu_src0_in(0),
        .alu_src1_in(0),
        .alu_src0_out(),
        .alu_src1_out(),
        //NPC
        .npc_in(0),
        .npc_out(),
        //BRANCH
        .npc_sel_in(0),
        .npc_sel_out(),
        //ALU
        .alu_res_in(0),
        .alu_res_out(),
        //SLU
        .dmem_rd_out_in(0),
        .dmem_wdata_mem_in(0),
        .dmem_rd_out_out(),
        .dmem_wdata_mem_out(),
        //DM
        .dmem_rdata_mem_in(0),
        .dmem_rdata_mem_out(),
        .commit_in(commit_if),
        .commit_out(commit_id)
    );

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

    ID_EX id_ex(
        .clk(clk),
        .rst(rst),
        .flush(flush_id_ex),
        .stall(stall),
        .en(en),
        //PC
        .pc_add4_in(pcadd4_id),
        .pc_in(pc_id),
        .pc_add4_out(pcadd4_ex),
        .pc_out(pc_ex),
        //INST
        .inst_in(inst_id),
        .inst_out(inst_ex),
        //DECODER
        .alu_op_in(alu_op_id),
        .dmem_access_in(dmem_access_id),
        .imm_in(imm_id),
        .rf_ra0_in(rf_ra0_id),
        .rf_ra1_in(rf_ra1_id),
        .rf_wa_in(rf_wa_id),
        .rf_we_in(rf_we_id),
        .rf_wd_sel_in(rf_wd_sel_id),
        .alu_src0_sel_in(alu_src0_sel_id),
        .alu_src1_sel_in(alu_src1_sel_id),
        .br_type_in(br_type_id),
        .dmem_we_in(dmem_we_id),
        .alu_op_out(alu_op_ex),
        .dmem_access_out(dmem_access_ex),
        .imm_out(imm_ex),
        .rf_ra0_out(rf_ra0_ex),
        .rf_ra1_out(rf_ra1_ex),
        .rf_wa_out(rf_wa_ex),
        .rf_we_out(rf_we_ex),
        .rf_wd_sel_out(rf_wd_sel_ex),
        .alu_src0_sel_out(alu_src0_sel_ex),
        .alu_src1_sel_out(alu_src1_sel_ex),
        .br_type_out(br_type_ex),
        .dmem_we_out(dmem_we_ex),
        //REG_FILE
        .rf_rd0_in(rf_rd0_id),
        .rf_rd1_in(rf_rd1_id),
        .rf_rd0_out(rf_rd0_ex),
        .rf_rd1_out(rf_rd1_ex),
        //MUX1
        .alu_src0_in(0),
        .alu_src1_in(0),
        .alu_src0_out(),
        .alu_src1_out(),
        //NPC
        .npc_in(0),
        .npc_out(),
        //BRANCH
        .npc_sel_in(0),
        .npc_sel_out(),
        //ALU
        .alu_res_in(0),
        .alu_res_out(),
        //SLU
        .dmem_rd_out_in(0),
        .dmem_wdata_mem_in(0),
        .dmem_rd_out_out(),
        .dmem_wdata_mem_out(),
        //DM
        .dmem_rdata_mem_in(0),
        .dmem_rdata_mem_out(),
        .commit_in(commit_id),
        .commit_out(commit_ex)
    );

    MUX1 mux1(
        .src0(rf_rd0_ex_),
        .src1(pc_ex),
        .sel(alu_src0_sel_ex),
        .res(alu_src0_ex)
    );

    MUX1 mux2(
        .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)
    );
    NPC npc(
        .pc_offset(alu_res_ex),
        .pc_add4(pcadd4_if),
        .pc_j(pc_j_ex),
        .npc_sel(npc_sel_ex),
        .npc(npc_ex)
    );

    EX_MEM ex_mem(
        .clk(clk),
        .rst(rst),
        .flush(flush_ex_mem),
        .stall(stall),
        .en(en),
        //PC
        .pc_add4_in(pcadd4_ex),
        .pc_in(pc_ex),
        .pc_add4_out(pcadd4_mem),
        .pc_out(pc_mem),
        //INST
        .inst_in(inst_ex),
        .inst_out(inst_mem),
        //DECODER
        .alu_op_in(alu_op_ex),
        .dmem_access_in(dmem_access_ex),
        .imm_in(imm_ex),
        .rf_ra0_in(rf_ra0_ex),
        .rf_ra1_in(rf_ra1_ex),
        .rf_wa_in(rf_wa_ex),
        .rf_we_in(rf_we_ex),
        .rf_wd_sel_in(rf_wd_sel_ex),
        .alu_src0_sel_in(alu_src0_sel_ex),
        .alu_src1_sel_in(alu_src1_sel_ex),
        .br_type_in(br_type_ex),
        .dmem_we_in(dmem_we_ex),
        .alu_op_out(alu_op_mem),
        .dmem_access_out(dmem_access_mem),
        .imm_out(imm_mem),
        .rf_ra0_out(rf_ra0_mem),
        .rf_ra1_out(rf_ra1_mem),
        .rf_wa_out(rf_wa_mem),
        .rf_we_out(rf_we_mem),
        .rf_wd_sel_out(rf_wd_sel_mem),
        .alu_src0_sel_out(alu_src0_sel_mem),
        .alu_src1_sel_out(alu_src1_sel_mem),
        .br_type_out(br_type_mem),
        .dmem_we_out(dmem_we_mem),
        //REG_FILE
        .rf_rd0_in(rf_rd0_ex_),
        .rf_rd1_in(rf_rd1_ex_),
        .rf_rd0_out(rf_rd0_mem),
        .rf_rd1_out(rf_rd1_mem),
        //MUX1
        .alu_src0_in(alu_src0_ex),
        .alu_src1_in(alu_src1_ex),
        .alu_src0_out(alu_src0_mem),
        .alu_src1_out(alu_src1_mem),
        //NPC
        .npc_in(npc_ex),
        .npc_out(npc_mem),
        //BRANCH
        .npc_sel_in(npc_sel_ex),
        .npc_sel_out(npc_sel_mem),
        //ALU
        .alu_res_in(alu_res_ex),
        .alu_res_out(alu_res_mem),
        //SLU
        .dmem_rd_out_in(0),
        .dmem_wdata_mem_in(0),
        .dmem_rd_out_out(),
        .dmem_wdata_mem_out(),
        //DM
        .dmem_rdata_mem_in(0),
        .dmem_rdata_mem_out(),
        .commit_in(commit_ex),
        .commit_out(commit_mem)
    );

    SLU slu(
        .addr(alu_res_mem),
        .dmem_access(dmem_access_mem),
        .rd_in(dmem_rdata_mem),
        .wd_in(rf_rd1_mem),
        .rd_out(dmem_rd_out_mem),
        .wd_out(dmem_wdata_mem)
    );

    MEM_WB mem_wb(
        .clk(clk),
        .rst(rst),
        .flush(flush_mem_wb),
        .stall(stall),
        .en(en),
        //PC
        .pc_add4_in(pcadd4_mem),
        .pc_in(pc_mem),
        .pc_add4_out(pcadd4_wb),
        .pc_out(pc_wb),
        //INST
        .inst_in(inst_mem),
        .inst_out(inst_wb),
        //DECODER
        .alu_op_in(alu_op_mem),
        .dmem_access_in(dmem_access_mem),
        .imm_in(imm_mem),
        .rf_ra0_in(rf_ra0_mem),
        .rf_ra1_in(rf_ra1_mem),
        .rf_wa_in(rf_wa_mem),
        .rf_we_in(rf_we_mem),
        .rf_wd_sel_in(rf_wd_sel_mem),
        .alu_src0_sel_in(alu_src0_sel_mem),
        .alu_src1_sel_in(alu_src1_sel_mem),
        .br_type_in(br_type_mem),
        .dmem_we_in(dmem_we_mem),
        .alu_op_out(alu_op_wb),
        .dmem_access_out(dmem_access_wb),
        .imm_out(imm_wb),
        .rf_ra0_out(rf_ra0_wb),
        .rf_ra1_out(rf_ra1_wb),
        .rf_wa_out(rf_wa_wb),
        .rf_we_out(rf_we_wb),
        .rf_wd_sel_out(rf_wd_sel_wb),
        .alu_src0_sel_out(alu_src0_sel_wb),
        .alu_src1_sel_out(alu_src1_sel_wb),
        .br_type_out(br_type_wb),
        .dmem_we_out(dmem_we_wb),
        //REG_FILE
        .rf_rd0_in(rf_rd0_mem),
        .rf_rd1_in(rf_rd1_mem),
        .rf_rd0_out(rf_rd0_wb),
        .rf_rd1_out(rf_rd1_wb),
        //MUX1
        .alu_src0_in(alu_src0_mem),
        .alu_src1_in(alu_src1_mem),
        .alu_src0_out(alu_src0_wb),
        .alu_src1_out(alu_src1_wb),
        //NPC
        .npc_in(npc_mem),
        .npc_out(npc_wb),
        //BRANCH
        .npc_sel_in(npc_sel_mem),
        .npc_sel_out(npc_sel_wb),
        //ALU
        .alu_res_in(alu_res_mem),
        .alu_res_out(alu_res_wb),
        //SLU
        .dmem_rd_out_in(dmem_rd_out_mem),
        .dmem_wdata_mem_in(dmem_wdata_mem),
        .dmem_rd_out_out(dmem_rd_out_wb),
        .dmem_wdata_mem_out(dmem_wdata_wb),
        //DM
        .dmem_rdata_mem_in(dmem_rdata_mem),
        .dmem_rdata_mem_out(dmem_rdata_wb),
        .commit_in(commit_mem),
        .commit_out(commit_wb)
    );

    MUX2 RF_MUX(
        .src0(pcadd4_wb),
        .src1(alu_res_wb),
        .src2(dmem_rd_out_wb),
        .src3(0),
        .sel(rf_wd_sel_wb),
        .res(rf_wd_wb)
    );

    /*
        ----------------------------------------------------------------------
        */
    /*                                      Commit
                                          */
    /*
        ----------------------------------------------------------------------
        */

    // wire [0 : 0] commit_if      ;
    // 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
            commit_reg          <= commit_wb;
            commit_pc_reg       <= pc_wb;
            commit_inst_reg     <= inst_wb;
            commit_halt_reg     <= inst_wb == `HALT_INST;
            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  <= alu_res_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;
endmodule
```

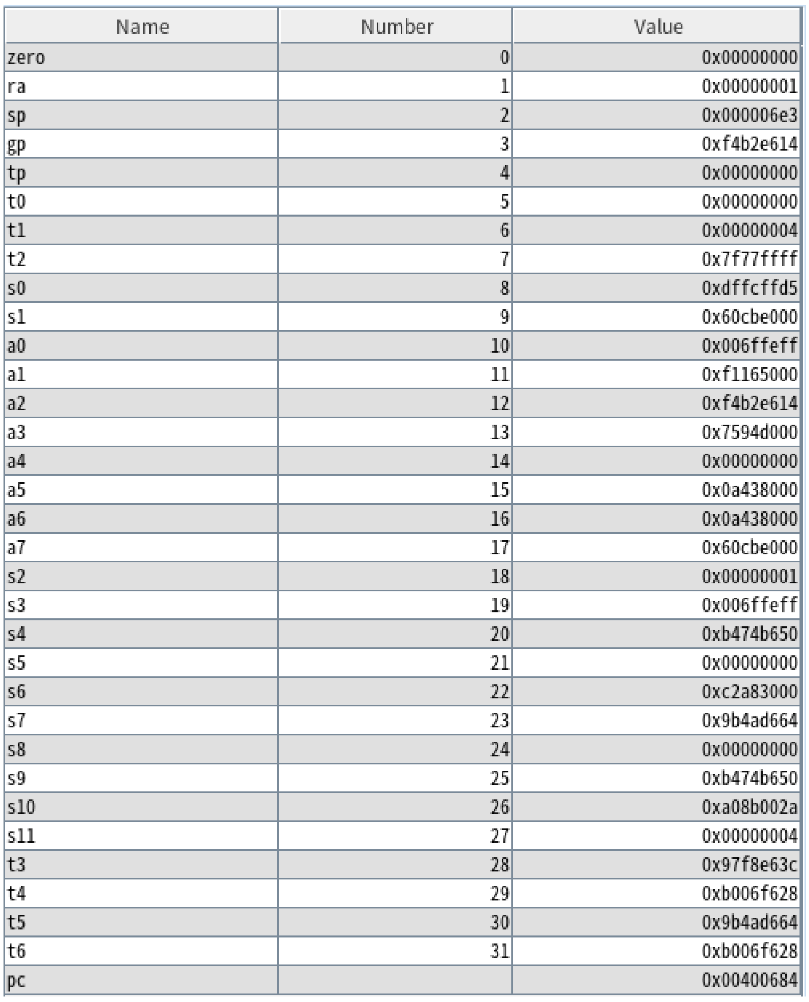
# 3 仿真结果与分析

## 3.1 Test1



| Name | Value |
|---|---|
| clk | 1 |
| imem_raddr[31:0] | 000001a4 |
| dmem_addr[31:0] | 3bffc000 |
| imem_rdata[31:0] | 00000013 |
| dmem_wdata[31:0] | 00000000 |
| commit[0:0] | 1 |
| commit_pc[31:0] | 0040067c |
| commit_inst[31:0] | 00020213 |
| commit_halt[0:0] | 0 |
| commit_reg_we[0:0] | 1 |
| commit_reg_wa[4:0] | 04 |
| commit_reg_wd[31:0] | 00000000 |
| commit_dmem_we[0:0] | 0 |
| commit_dmem_wa[31:0] | 00000000 |
| commit_dmem_wd[31:0] | 00000000 |
| debug_reg_rd[31:0] | 00000000 |
| dmem_rdata[31:0] | 00000000 |
| reg_file[0:31][31:0] | 00000000,000000 |
| [0][31:0] | 00000000 |
| [1][31:0] | 00000001 |
| [2][31:0] | 000006e3 |
| [3][31:0] | f4b2e614 |
| [4][31:0] | 00000000 |
| [5][31:0] | 00000000 |
| [6][31:0] | 00000004 |
| [7][31:0] | 7f77ffff |
| [8][31:0] | dffcffd5 |
| [9][31:0] | 60cbe000 |
| [10][31:0] | 006ffeff |
| [11][31:0] | f1165000 |
| [12][31:0] | f4b2e614 |
| [13][31:0] | 7594d000 |
| [14][31:0] | 00000000 |
| [15][31:0] | 0a438000 |
| [16][31:0] | 0a438000 |
| [17][31:0] | 60cbe000 |
| [18][31:0] | 00000001 |
| [19][31:0] | 006ffeff |
| [20][31:0] | b474b650 |
| [21][31:0] | 00000000 |
| [22][31:0] | c2a83000 |
| [23][31:0] | 9b4ad664 |
| [24][31:0] | 00000000 |
| [25][31:0] | b474b650 |
| [26][31:0] | a08b002a |
| [27][31:0] | 00000004 |
| [28][31:0] | 97f8e63c |
| [29][31:0] | b006f628 |
| [30][31:0] | 9b4ad664 |
| [31][31:0] | b006f628 |

图 1: 仿真结果

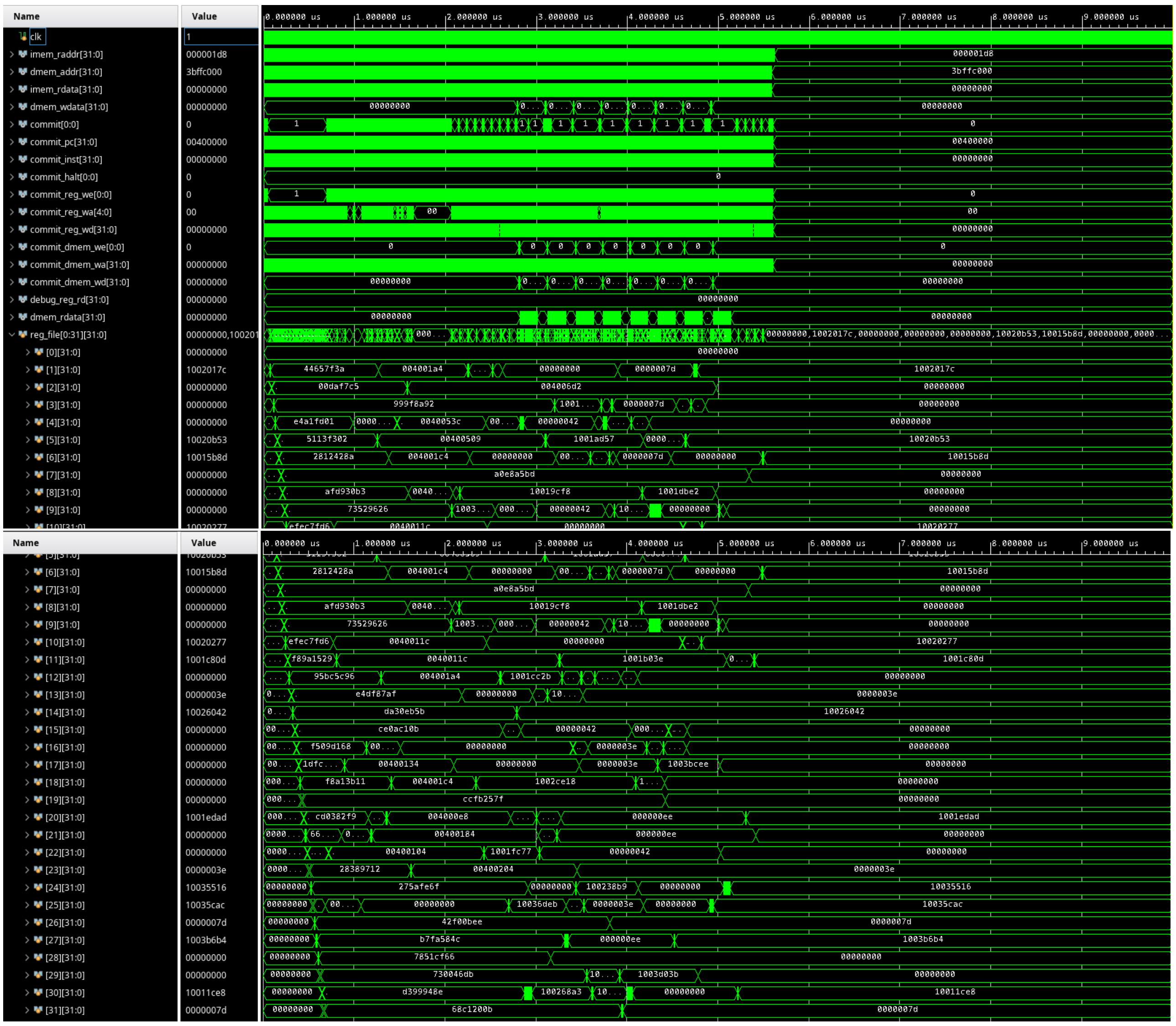


| Name | Number | Value |
|---|---|---|
| zero | 0 | 0x00000000 |
| ra | 1 | 0x00000001 |
| sp | 2 | 0x000006e3 |
| gp | 3 | 0xf4b2e614 |
| tp | 4 | 0x00000000 |
| t0 | 5 | 0x00000000 |
| t1 | 6 | 0x00000004 |
| t2 | 7 | 0x7f77ffff |
| s0 | 8 | 0xdffcffd5 |
| s1 | 9 | 0x60cbe000 |
| a0 | 10 | 0x006ffeff |
| 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 |  | 0x00400684 |

图 2: 运行结果

可以看到，仿真结果和运行结果吻合的很好。

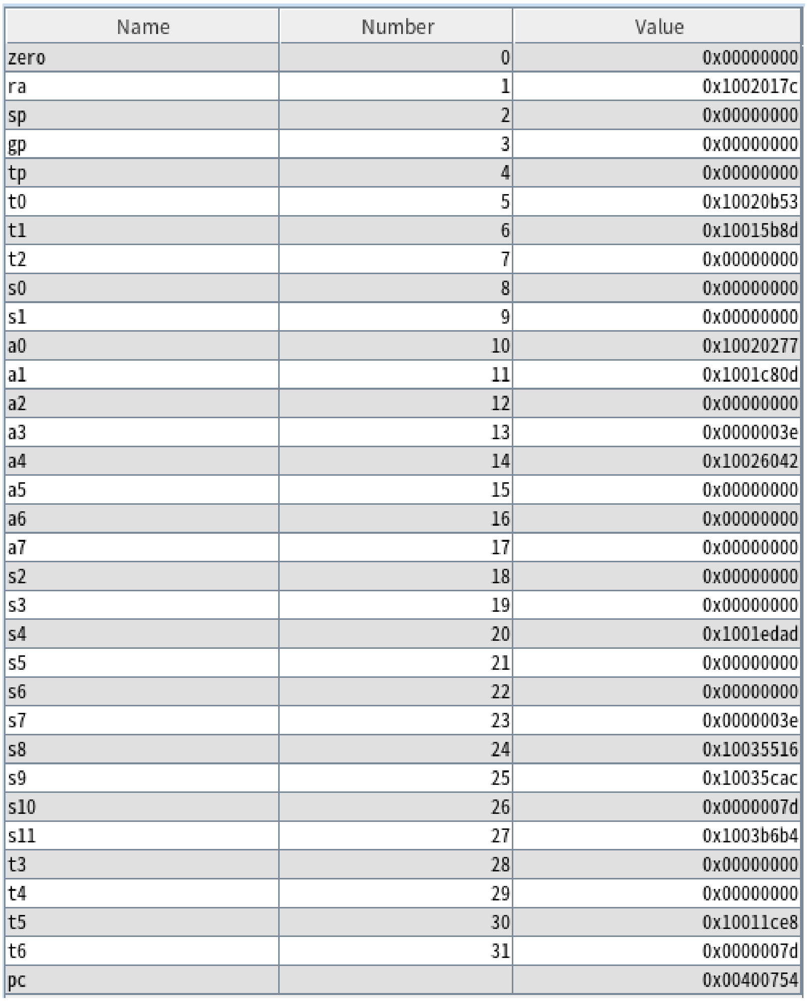
## 3.2 Test2



| Name | Value |
|---|---|
| clk | 1 |
| imem_raddr[31:0] | 000001d8 |
| dmem_addr[31:0] | 3bffc000 |
| imem_rdata[31:0] | 00000000 |
| dmem_wdata[31:0] | 00000000 |
| commit[0:0] | 0 |
| commit_pc[31:0] | 00400000 |
| commit_inst[31:0] | 00000000 |
| commit_halt[0:0] | 0 |
| commit_reg_we[0:0] | 0 |
| commit_reg_wa[4:0] | 00 |
| commit_reg_wd[31:0] | 00000000 |
| commit_dmem_we[0:0] | 0 |
| commit_dmem_wa[31:0] | 00000000 |
| commit_dmem_wd[31:0] | 00000000 |
| debug_reg_rd[31:0] | 00000000 |
| dmem_rdata[31:0] | 00000000 |
| reg_file[0:31][31:0] | 00000000,100201 |
| [0][31:0] | 00000000 |
| [1][31:0] | 1002017c |
| [2][31:0] | 00000000 |
| [3][31:0] | 00000000 |
| [4][31:0] | 00000000 |
| [5][31:0] | 10020b53 |
| [6][31:0] | 10015b8d |
| [7][31:0] | 00000000 |
| [8][31:0] | 00000000 |
| [9][31:0] | 00000000 |
| [10][31:0] | 10020277 |

| Name | Value |
|---|---|
| [6][31:0] | 10015b8d |
| [7][31:0] | 00000000 |
| [8][31:0] | 00000000 |
| [9][31:0] | 00000000 |
| [10][31:0] | 10020277 |
| [11][31:0] | 1001c80d |
| [12][31:0] | 00000000 |
| [13][31:0] | 0000003e |
| [14][31:0] | 10026042 |
| [15][31:0] | 00000000 |
| [16][31:0] | 00000000 |
| [17][31:0] | 00000000 |
| [18][31:0] | 00000000 |
| [19][31:0] | 00000000 |
| [20][31:0] | 1001edad |
| [21][31:0] | 00000000 |
| [22][31:0] | 00000000 |
| [23][31:0] | 0000003e |
| [24][31:0] | 10035516 |
| [25][31:0] | 10035cac |
| [26][31:0] | 0000007d |
| [27][31:0] | 1003b6b4 |
| [28][31:0] | 00000000 |
| [29][31:0] | 00000000 |
| [30][31:0] | 10011ce8 |
| [31][31:0] | 0000007d |

图 3: 仿真结果



| 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 | 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 |  | 0x00400754 |

图 4: 运行结果

可以看到，仿真结果和运行结果吻合的很好。

# 4 电路设计与分析

## 4.1 前递模块

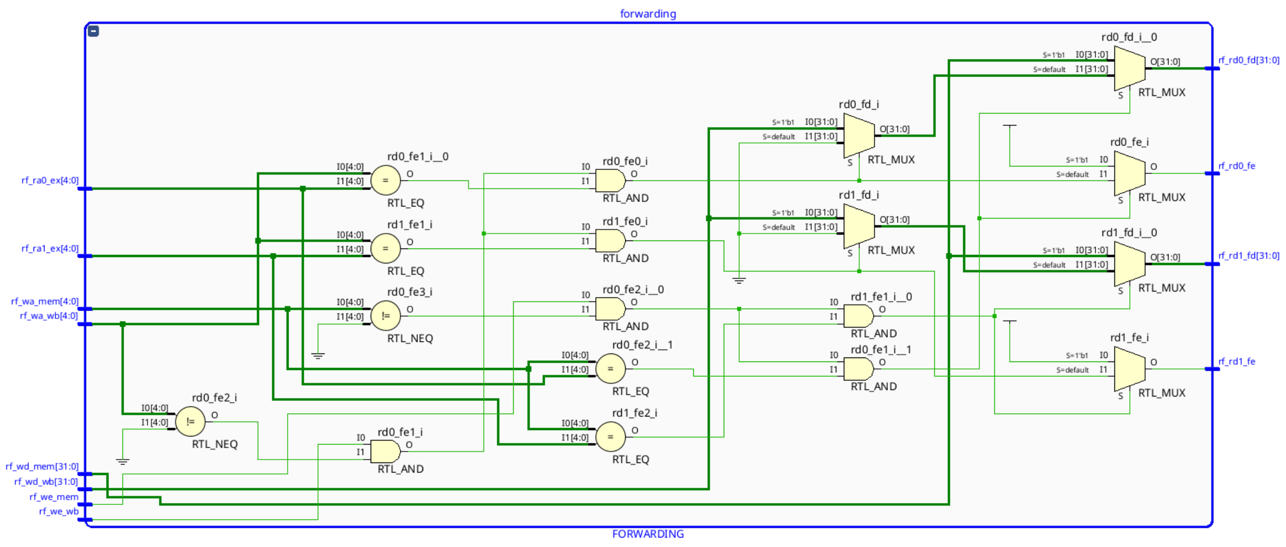


图 5: 前递模块

## 4.2 段间寄存器控制模块

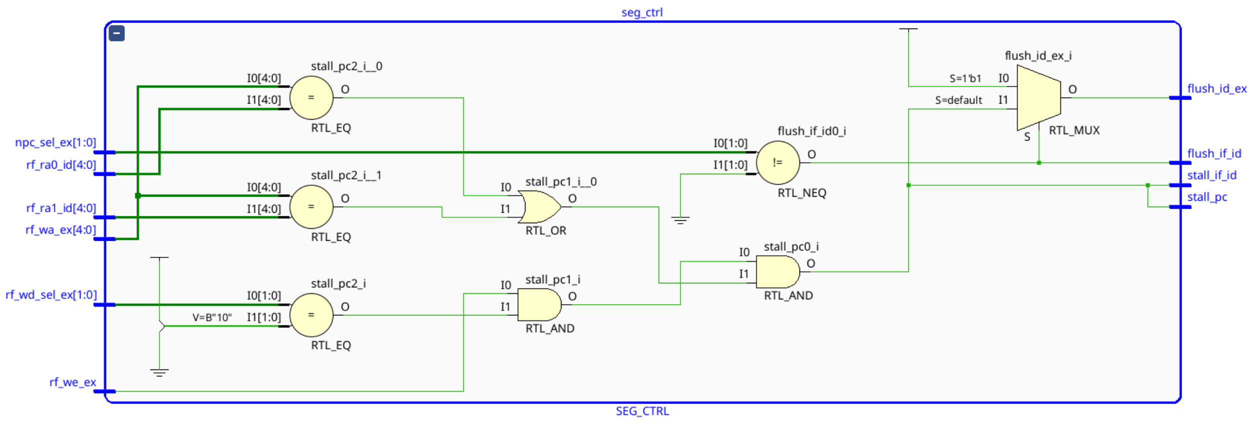


图 6: 段间寄存器控制模块

## 4.3 CPU 模块

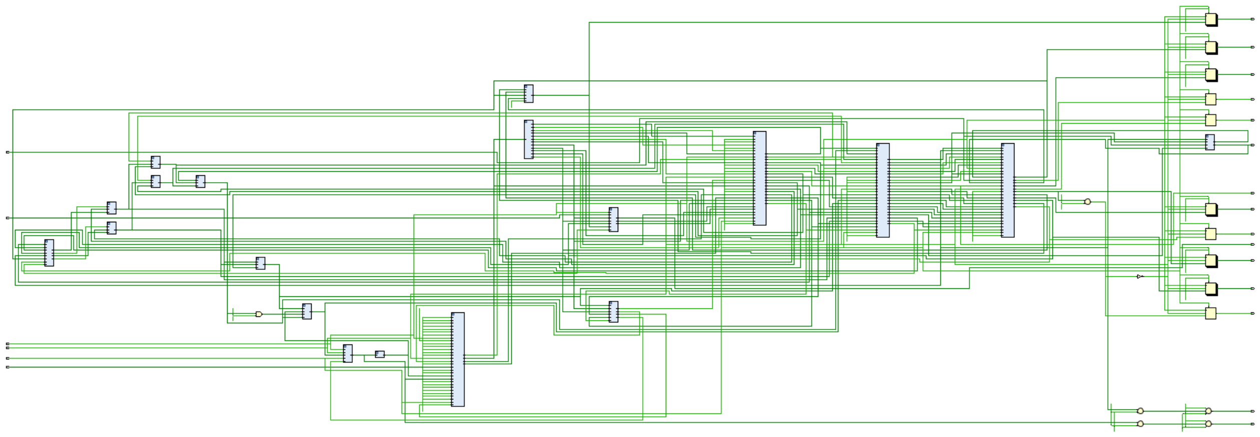
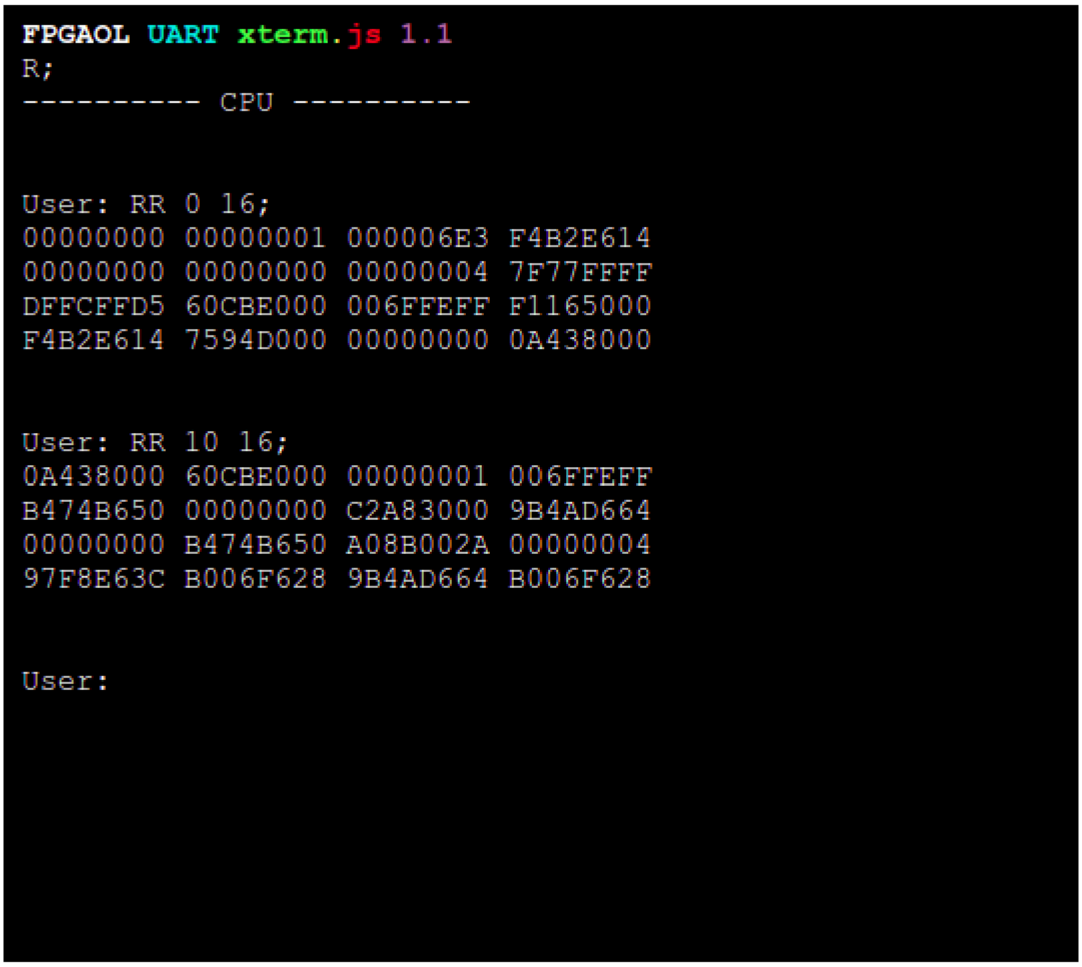


图 7: CPU 模块

# 5 测试结果与分析

## 5.1 Test1

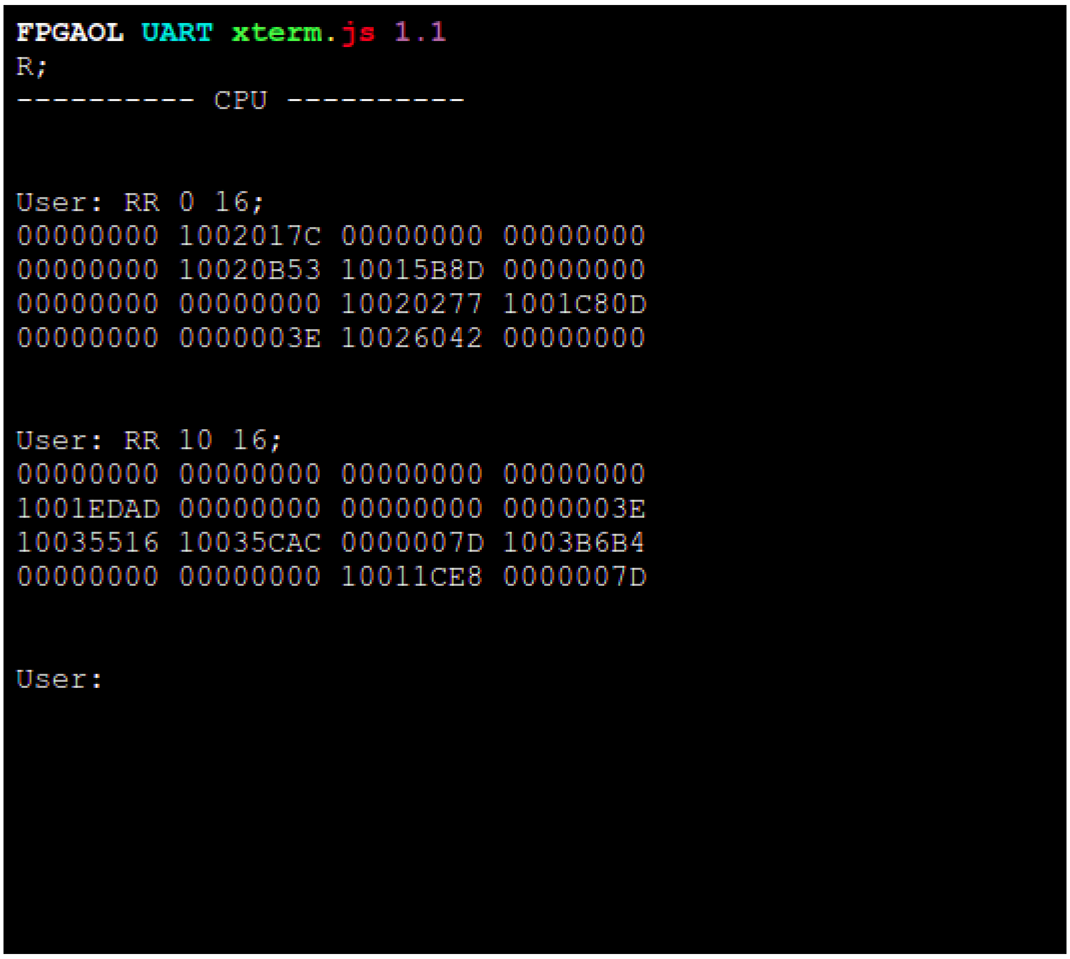


```
FPGAOL UART xterm.js 1.1
R;
---------- CPU ----------


User: RR 0 16;
00000000 00000001 000006E3 F4B2E614
00000000 00000000 00000004 7F77FFFF
DFFCFFD5 60CBE000 006FFEFF F1165000
F4B2E614 7594D000 00000000 0A438000


User: RR 10 16;
0A438000 60CBE000 00000001 006FFEFF
B474B650 00000000 C2A83000 9B4AD664
00000000 B474B650 A08B002A 00000004
97F8E63C B006F628 9B4AD664 B006F628


User:
```

图 8: 上板运行结果

可以看到，上板结果和运行结果吻合的很好。

## 5.2 Test2



```
FPGAOL UART xterm.js 1.1
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:
```

图 9: 上板运行结果

可以看到，上板结果和运行结果吻合的很好。

# 6 总结

本次实验实现了前递模块与段间寄存器控制模块，并将其加入 CPU 模块，从而实现了一个完整的五级流水线 CPU。