lab5 实验报告

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.写优先的寄存器堆

　根据实验文档中的介绍将寄存器堆改为写优先的模式，并仿真测试正确性。

2.无冒险流水线

　正确设计并例化四个段间寄存器，连线以实现无冒险流水线 CPU。最终，你需要在 FPGAOL 上上板运行，并通过我们给出的测试程序。

第二部分 实验过程

2.1 实验设计

2.1.1 写优先寄存器堆

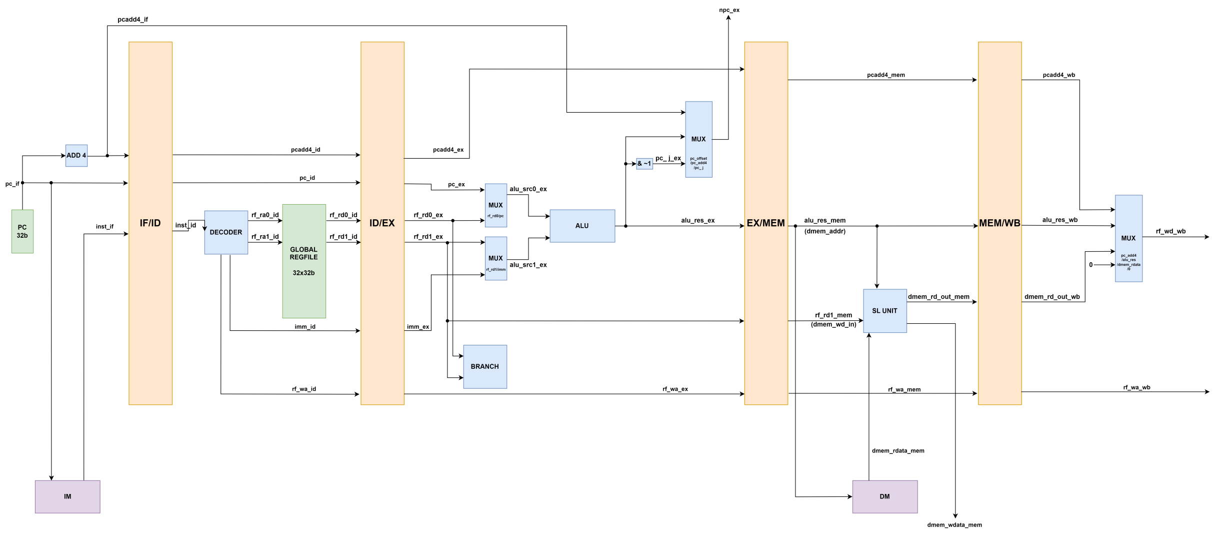
两层条件判断，若读0号寄存器，则输出0，此外，若写使能信号rf_we==1且读/写的寄存器编号相同，则输出将要写入的值。否则正常读寄存器堆中数据。

2.1.2 段间寄存器

段间寄存器模块，从最简单的角度来说，它只需要在每个时钟上升沿将输入传递到输出，不过，为了方便后续的操作，我们需要额外添加四个接口，即 rst、en、stall 和 flush：

- rst 的效果为同步清空，当此信号高电平时段间寄存器的所有寄存器都将被清空，与 CPU 的 rst 信号连接。
- en 的作用是让段间寄存器受到 PDU 的控制，保证其与 PC 寄存器 en 端口的同步。段间寄存器的 en 端口同样连接到 global_en。
- stall 的效果为停驻，若时钟上升沿此信号高电平，输出仍保持之前的值不变，而非接收输入（也即其是反向的写使能信号）。
- flush 的效果为同步清空，若时钟上升沿此信号高电平，段间寄存器的所有寄存器都将被清空。

2.1.3 CPU数据通路



2.2 核心代码

2.2.1 写优先寄存器堆

```
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
```

2.2.2 段间寄存器

```
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] 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] 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;

        alu_op_out=5'bz;
        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 (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;

            alu_op_out=5'bz;
            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;

            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
```

2.2.3 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 [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;
```

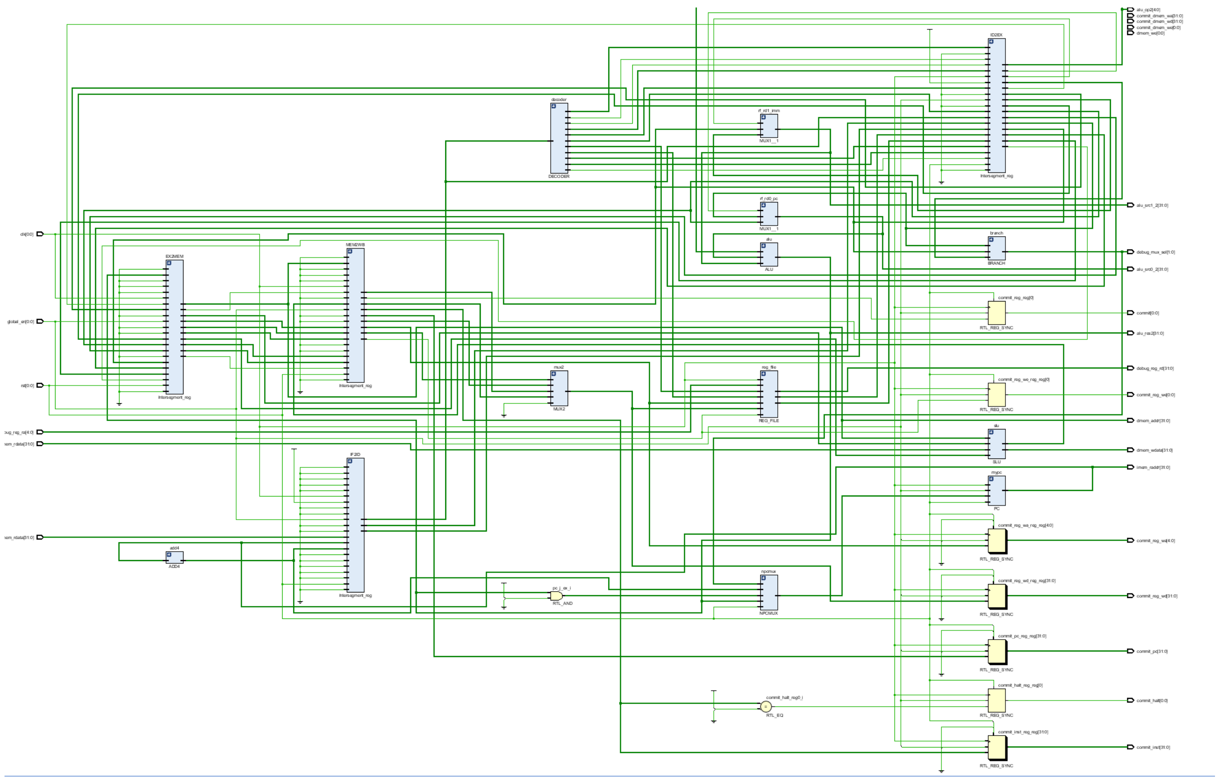
```
PC mypc(
    .clk(clk),
    .rst(rst),
    .en(global_en),
    .npc(npc_ex),
    .pc(pc_if)
);

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),
    .flush(flush),
    .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),

    .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 [31:0] imm_id;
wire [4:0] rf_ra0_id;
wire [4:0] rf_ra1_id;
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] 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] rd_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),

    .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),
```

```verilog
    .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_if),

    .pcadd4_out(pcadd4_ex),
    .inst_out(inst_ex),
    .pc_out(pc_ex),
    .rf_rd0_out(rf_rd0_ex),
    .rf_rd1_out(rf_rd1_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(rd_wd_sel_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(rd_wd_sel_ex),
    .commit_in(commit_ex),

    .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)
);

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;
SLU slu(
    .addr(dmem_addr),
    .wd_in(dmem_wd_in_mem),
```

```verilog
    .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;
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),

    .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)
);

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)
);

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;
```

第三部分 实验结果

3.1 电路分析结果（文档尾部有高清图）



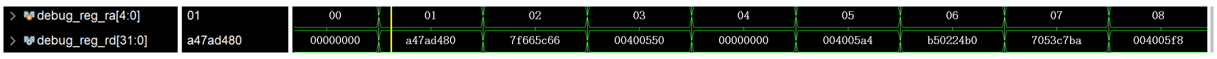
3.2 电路仿真结果

仿真文件如下(使用测试程序 2)：

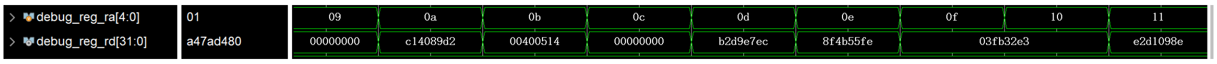
```
//////////////////////////////////////////////////////////////////////////////////
// Company: 
// Engineer: 
// 
// Create Date: 2024/04/09 17:02:48
// Design Name: 
// Module Name: CPU_tb
// Project Name: 
// Target Devices: 
// Tool Versions: 
// Description: 
// 
// Dependencies: 
// 
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
// 
//////////////////////////////////////////////////////////////////////////////////


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;

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[10:2]),      // 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)
);

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
```

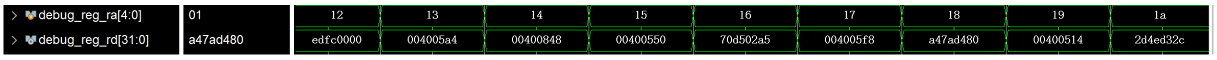
仿真结果如下：



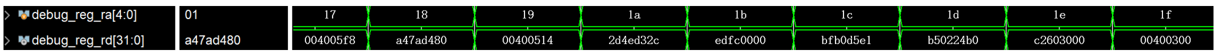
```
debug_reg_ra[4:0]    01         00        01        02        03        04        05        06        07        08
debug_reg_rd[31:0]   a47ad480   00000000  a47ad480  7f665c66  00400550  00000000  004005a4  b50224b0  7053c7ba  004005f8
```



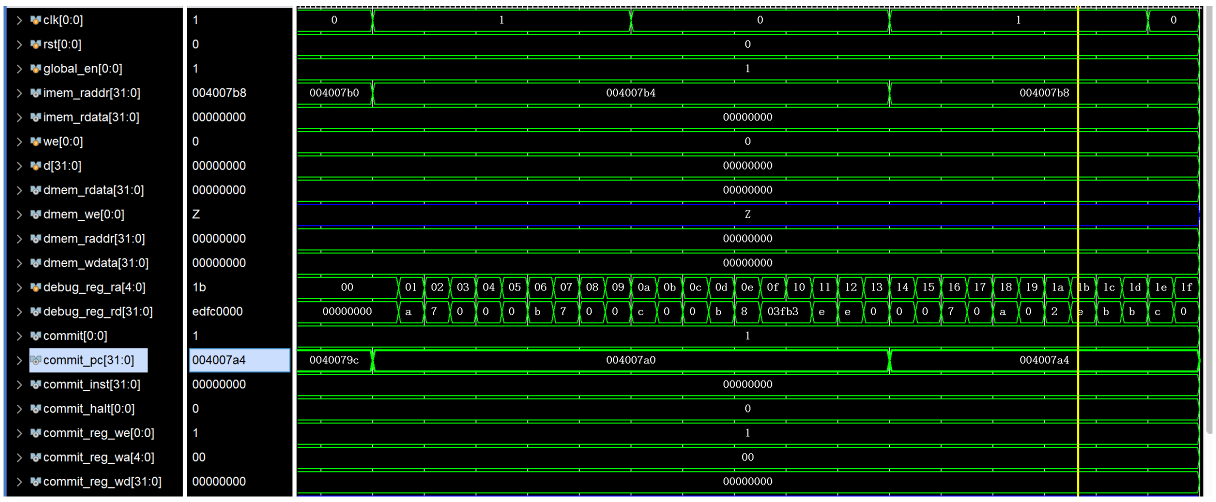
```
debug_reg_ra[4:0]    01         09        0a        0b        0c        0d        0e        0f        10        11
debug_reg_rd[31:0]   a47ad480   00000000  c14089d2  00400514  00000000  b2d9e7ec  8f4b55fe  03fb32e3            e2d1098e
```



```
debug_reg_ra[4:0]    01         12        13        14        15        16        17        18        19        1a
debug_reg_rd[31:0]   a47ad480   edfc0000  004005a4  00400848  00400550  70d502a5  004005f8  a47ad480  00400514  2d4ed32c
```

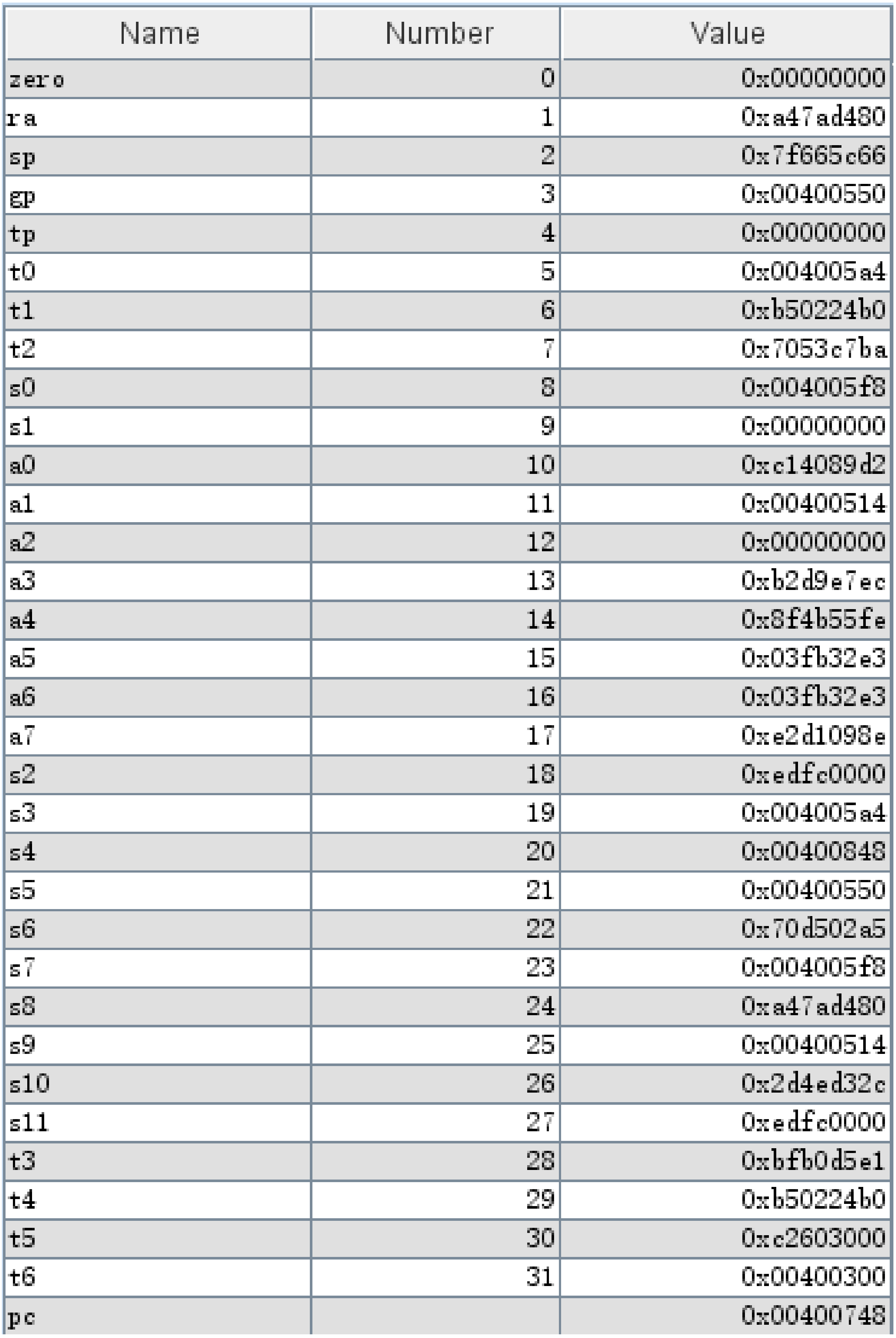


```
debug_reg_ra[4:0]    01         17        18        19        1a        1b        1c        1d        1e        1f
debug_reg_rd[31:0]   a47ad480   004005f8  a47ad480  00400514  2d4ed32c  edfc0000  bfb0d5e1  b50224b0  c2603000  00400300
```



```
clk[0:0]             1          0  1  0  1  0
rst[0:0]             0          0
global_en[0:0]       1          1
imem_raddr[31:0]     004007b8   004007b0  004007b4  004007b8
imem_rdata[31:0]     00000000   00000000
we[0:0]              0          0
d[31:0]              00000000   00000000
dmem_rdata[31:0]     00000000   00000000
dmem_we[0:0]         Z          Z
dmem_raddr[31:0]     00000000   00000000
dmem_wdata[31:0]     00000000   00000000
debug_reg_ra[4:0]    1b         00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f 10 11 12 13 14 15 16 17 18 19 1a 1b 1c 1d 1e 1f
debug_reg_rd[31:0]   edfc0000   00000000 a 7 0 0 0 b 7 0 0 c 0 0 b 8 03fb3 e e 0 0 0 7 0 a 0 2 e b b c 0
commit[0:0]          1          1
commit_pc[31:0]      004007a4   0040079c  004007a0  004007a4
commit_inst[31:0]    00000000   00000000
commit_halt[0:0]     0          0
commit_reg_we[0:0]   1          1
commit_reg_wa[4:0]   00         00
commit_reg_wd[31:0]  00000000   00000000
```

RARS运行结果如下：



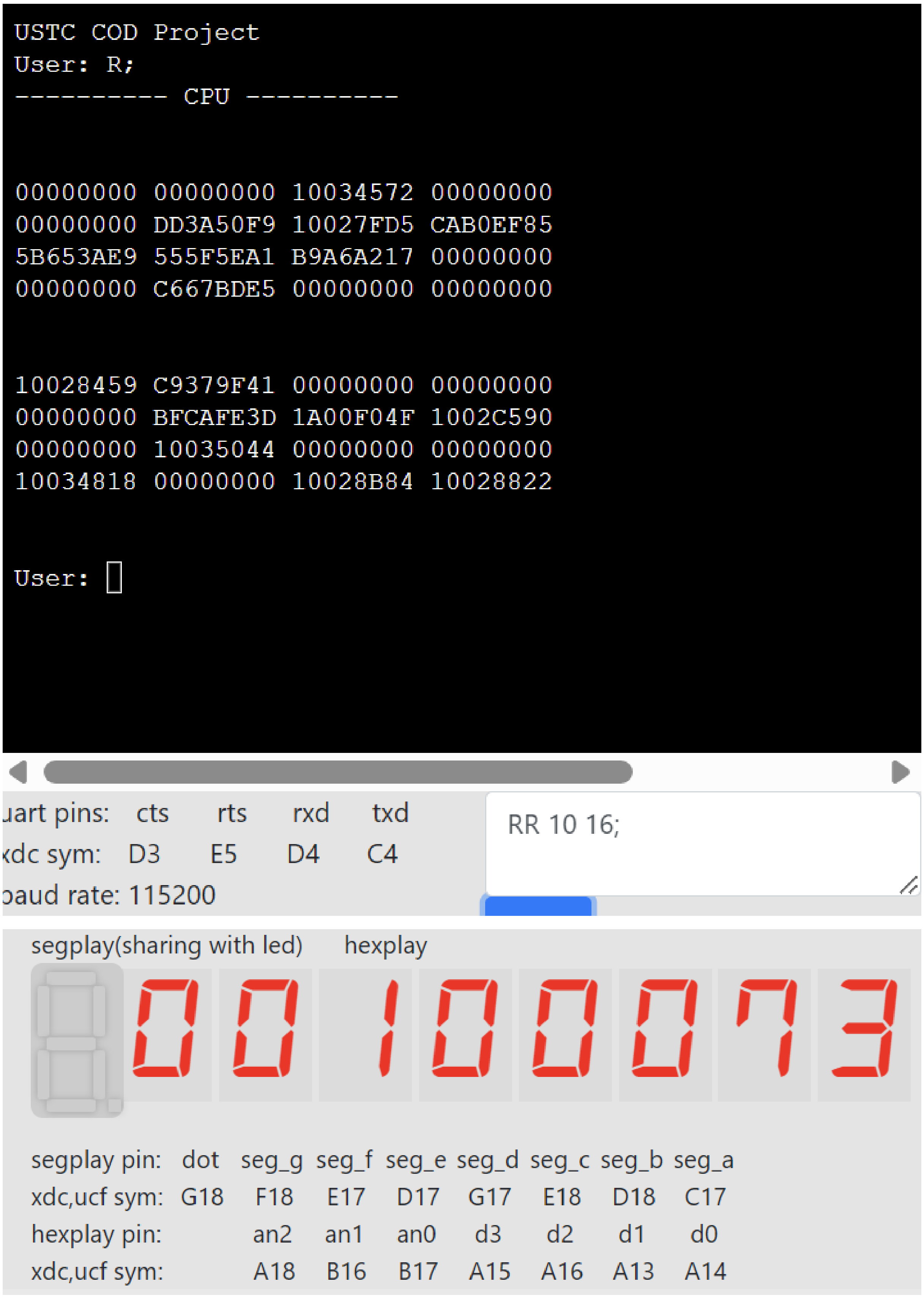
Name	Number	Value
zero	0	0x00000000
ra	1	0xa47ad480
sp	2	0x7f665c66
gp	3	0x00400550
tp	4	0x00000000
t0	5	0x004005a4
t1	6	0xb50224b0
t2	7	0x7053c7ba
s0	8	0x004005f8
s1	9	0x00000000
a0	10	0xc14089d2
a1	11	0x00400514
a2	12	0x00000000
a3	13	0xb2d9e7ec
a4	14	0x8f4b55fe
a5	15	0x03fb32e3
a6	16	0x03fb32e3
a7	17	0xe2d1098e
s2	18	0xedfc0000
s3	19	0x004005a4
s4	20	0x00400848
s5	21	0x00400550
s6	22	0x70d502a5
s7	23	0x004005f8
s8	24	0xa47ad480
s9	25	0x00400514
s10	26	0x2d4ed32c
s11	27	0xedfc0000
t3	28	0xbfb0d5e1
t4	29	0xb50224b0
t5	30	0xc2603000
t6	31	0x00400300
pc		0x00400748

保持一致。

3.3 上板结果

3.3.1 测试程序1

FPGAOL运行结果：



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


00000000 00000000 10034572 00000000
00000000 DD3A50F9 10027FD5 CAB0EF85
5B653AE9 555F5EA1 B9A6A217 00000000
00000000 C667BDE5 00000000 00000000


10028459 C9379F41 00000000 00000000
00000000 BFCAFE3D 1A00F04F 1002C590
00000000 10035044 00000000 00000000
10034818 00000000 10028B84 10028822


User: 
```

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

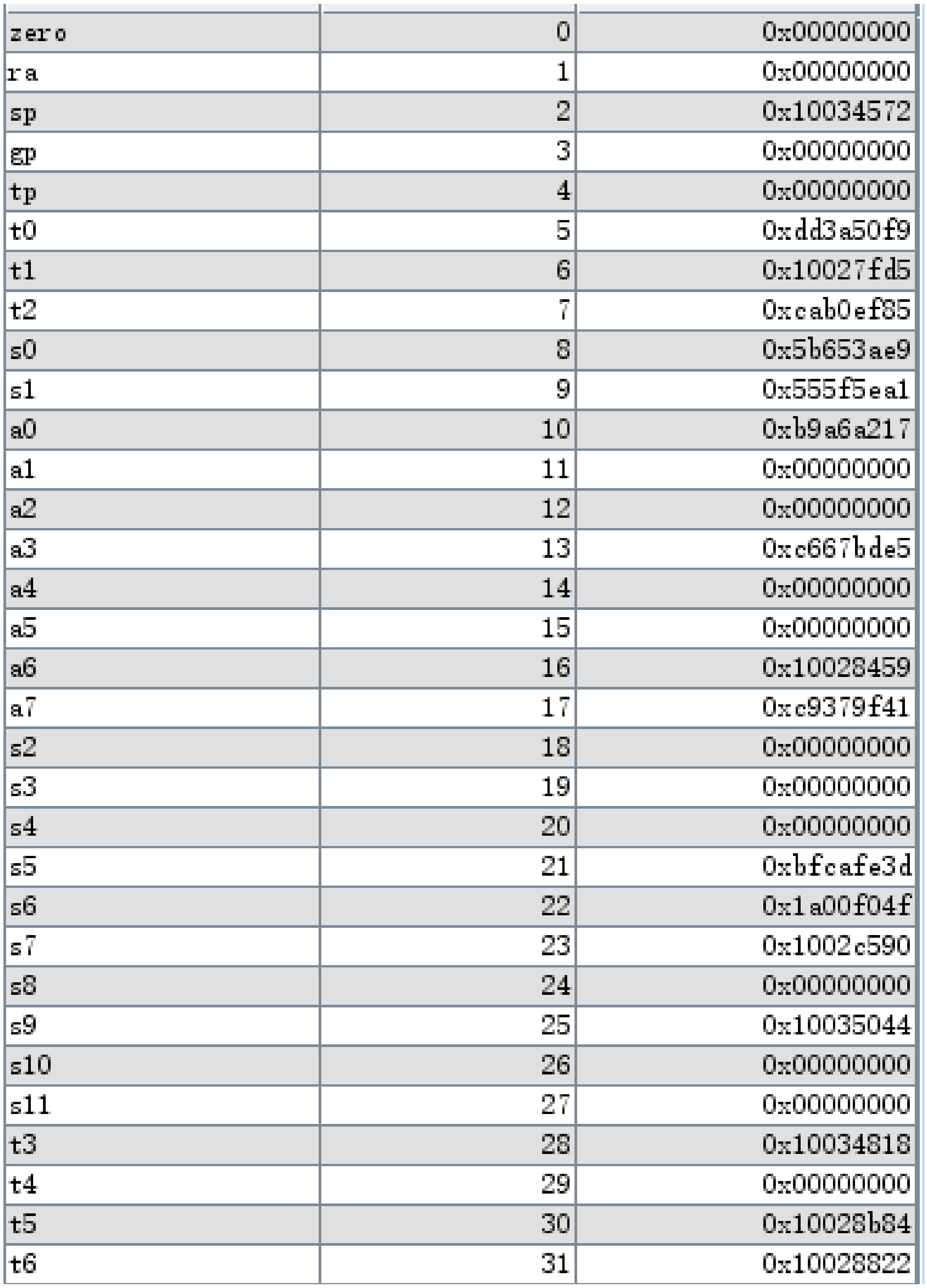
```
RR 10 16;
```

segplay(sharing with led)   hexplay

```
00 100073
```

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	G18	F18	E17	D17	G17	E18	D18	C17
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

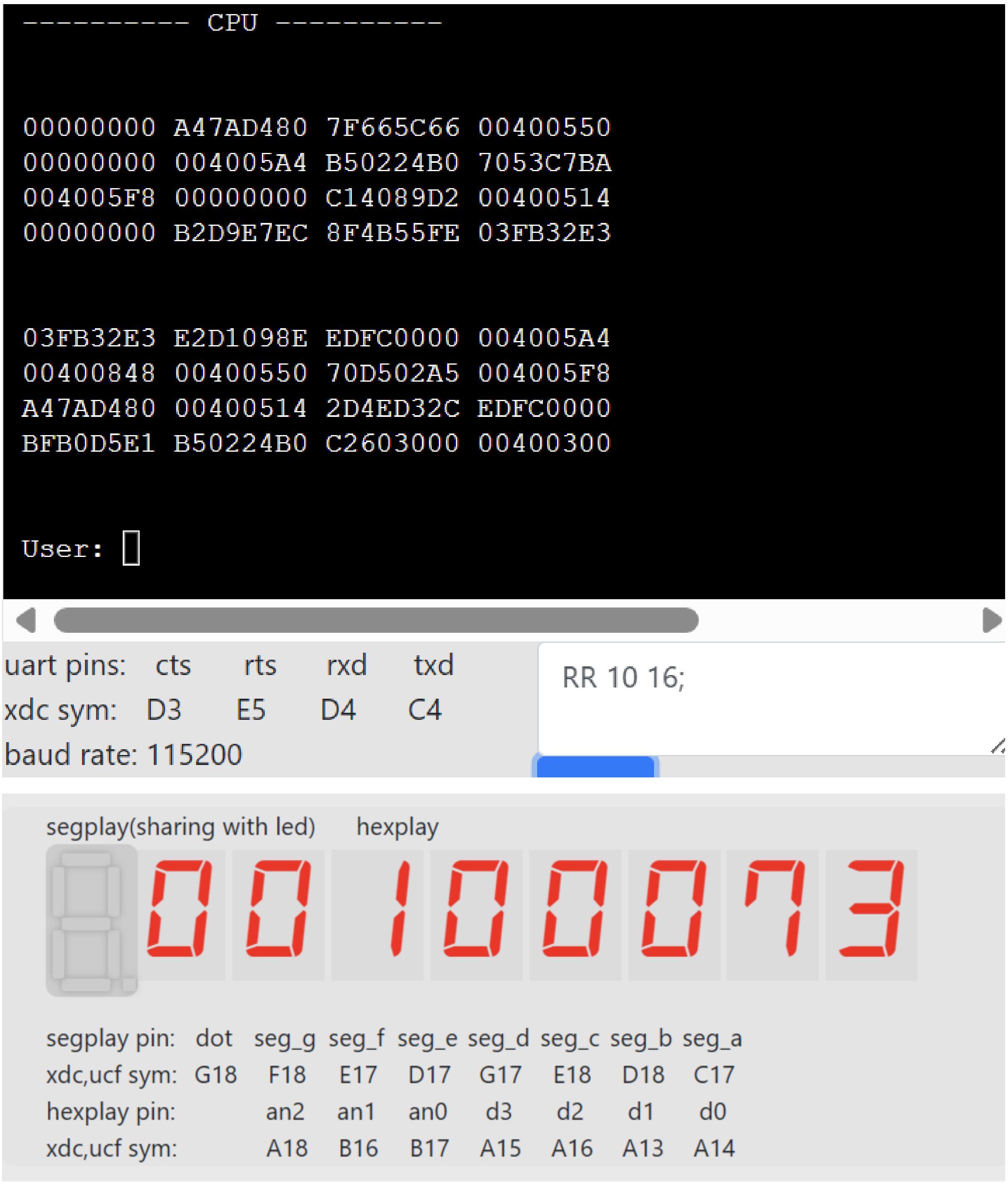
RARS结果：



zero	0	0x00000000
ra	1	0x00000000
sp	2	0x10034572
gp	3	0x00000000
tp	4	0x00000000
t0	5	0xdd3a50f9
t1	6	0x10027fd5
t2	7	0xcab0ef85
s0	8	0x5b653ae9
s1	9	0x555f5ea1
a0	10	0xb9a6a217
a1	11	0x00000000
a2	12	0x00000000
a3	13	0xc667bde5
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x10028459
a7	17	0xc9379f41
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0xbfcafe3d
s6	22	0x1a00f04f
s7	23	0x1002c590
s8	24	0x00000000
s9	25	0x10035044
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x10034818
t4	29	0x00000000
t5	30	0x10028b84
t6	31	0x10028822

3.3.2 测试程序2

FPGAOL运行结果：



```
---------- CPU ----------


00000000 A47AD480 7F665C66 00400550
00000000 004005A4 B50224B0 7053C7BA
004005F8 00000000 C14089D2 00400514
00000000 B2D9E7EC 8F4B55FE 03FB32E3


03FB32E3 E2D1098E EDFC0000 004005A4
00400848 00400550 70D502A5 004005F8
A47AD480 00400514 2D4ED32C EDFC0000
BFB0D5E1 B50224B0 C2603000 00400300


User:
```

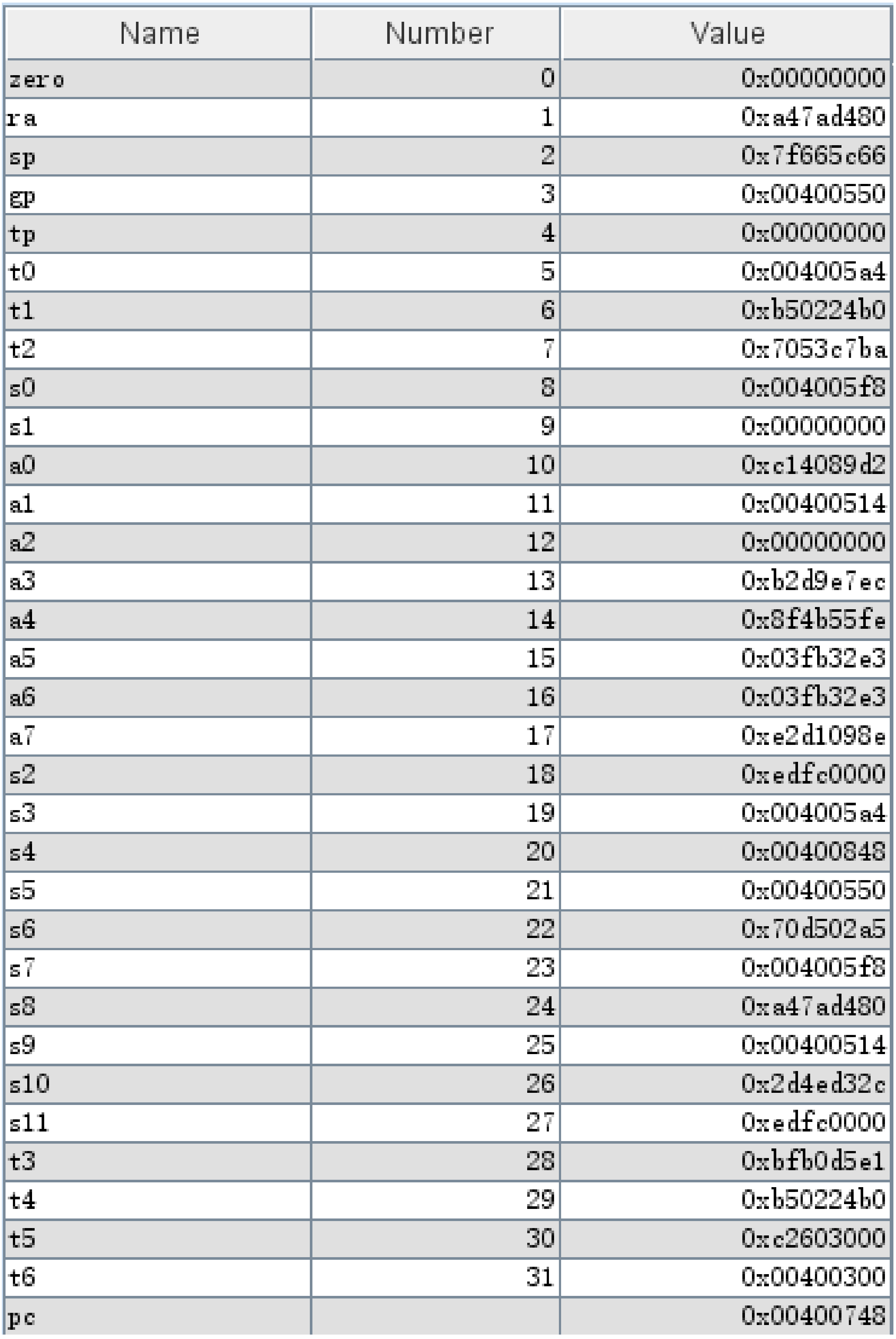
uart pins:	cts	rts	rxd	txd
xdc sym:	D3	E5	D4	C4

baud rate: 115200

RR 10 16;

segplay pin:	dot	seg_g	seg_f	seg_e	seg_d	seg_c	seg_b	seg_a
xdc,ucf sym:	G18	F18	E17	D17	G17	E18	D18	C17
hexplay pin:		an2	an1	an0	d3	d2	d1	d0
xdc,ucf sym:		A18	B16	B17	A15	A16	A13	A14

RARS结果：



Name	Number	Value
zero	0	0x00000000
ra	1	0xa47ad480
sp	2	0x7f665c66
gp	3	0x00400550
tp	4	0x00000000
t0	5	0x004005a4
t1	6	0xb50224b0
t2	7	0x7053c7ba
s0	8	0x004005f8
s1	9	0x00000000
a0	10	0xc14089d2
a1	11	0x00400514
a2	12	0x00000000
a3	13	0xb2d9e7ec
a4	14	0x8f4b55fe
a5	15	0x03fb32e3
a6	16	0x03fb32e3
a7	17	0xe2d1098e
s2	18	0xedfc0000
s3	19	0x004005a4
s4	20	0x00400848
s5	21	0x00400550
s6	22	0x70d502a5
s7	23	0x004005f8
s8	24	0xa47ad480
s9	25	0x00400514
s10	26	0x2d4ed32c
s11	27	0xedfc0000
t3	28	0xbfb0d5e1
t4	29	0xb50224b0
t5	30	0xc2603000
t6	31	0x00400300
pc		0x00400748

第四部分 思考题

1. **在实验报告中回答**，对于本次实验中的五级流水线 CPU，连续执行以下的指令序列后，**若寄存器堆没有使用写优先**，运行结束后 x4 与 x5 中的结果是什么？

```
addi x1, x0, 1
addi x2, x0, 2
addi x3, x0, 3
addi x1, x0, 10
addi x2, x0, 20
addi x3, x0, 30
addi x0, x0, 0
add  x4, x1, x2
add  x5, x2, x3
nop
nop
nop
```

addi x1, x0, 1	**IF**	**ID**	**EX**	**MEM**	**WB**								
addi x2, x0, 2		IF	ID	EX	MEM	WB							
addi x3, x0, 3			IF	ID	EX	MEM	WB						
addi x1, x0, 10				IF	ID	EX	MEM	WB					
addi x2, x0, 20					IF	ID	EX	MEM	WB				
addi x3, x0, 30						IF	ID	EX	MEM	WB			
addi x0, x0, 0							IF	ID	EX	MEM	WB		
add x4, x1, x2								IF	ID	EX	MEM	WB	
add x5, x2, x3									IF	ID	EX	MEM	WB
x1	-	-	-	-	1	1	1	10	10	10	10	10	10
x2	-	-	-	-	-	2	2	2	20	20	20	20	20
x3	-	-	-	-	-	-	3	3	3	30	30	30	30

对非写优先寄存器，x4=10+2=12，x5=20+3=23.

第五部分 心得体会

1.加深了单周期CPU原理的理解

2.对vivado，FPGAOL平台的使用（debug）熟练度进一步提升

第六部分 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                  [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 [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),
    .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),
    .flush(flush),
    .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),

    .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 [31:0] imm_id;
wire [4:0] rf_ra0_id;
wire [4:0] rf_ra1_id;
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] 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] rd_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),

    .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_if),

    .pcadd4_out(pcadd4_ex),
    .inst_out(inst_ex),
    .pc_out(pc_ex),
    .rf_rd0_out(rf_rd0_ex),
    .rf_rd1_out(rf_rd1_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(rd_wd_sel_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)
```

```verilog
);

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(rd_wd_sel_ex),
    .commit_in(commit_ex),

    .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)
);

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;
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)
```

```verilog
);

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;
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),

    .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)
);

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)
);

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;
/* ------------------------------------------------------------------------ */
/*                                  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,
```

```verilog
    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'b0,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
```

```verilog
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;//sltu
            10'b100_0000000: alu_op=5'B01011;//xor
            10'b101_0000000: alu_op=5'B01111;//srl
            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
    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

//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
        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]            en,
    input                   [31 : 0]            npc,

    output      reg         [31 : 0]            pc
);
```

```verilog
always @(posedge clk or posedge rst) begin
    if(rst)begin
        pc <= 32'h00400000;
    end
    else if(en)begin
        pc <= npc;
    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'h0ffffffc) begin
        npc=pc+32'h4;
    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
```

```verilog
    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


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]};
```

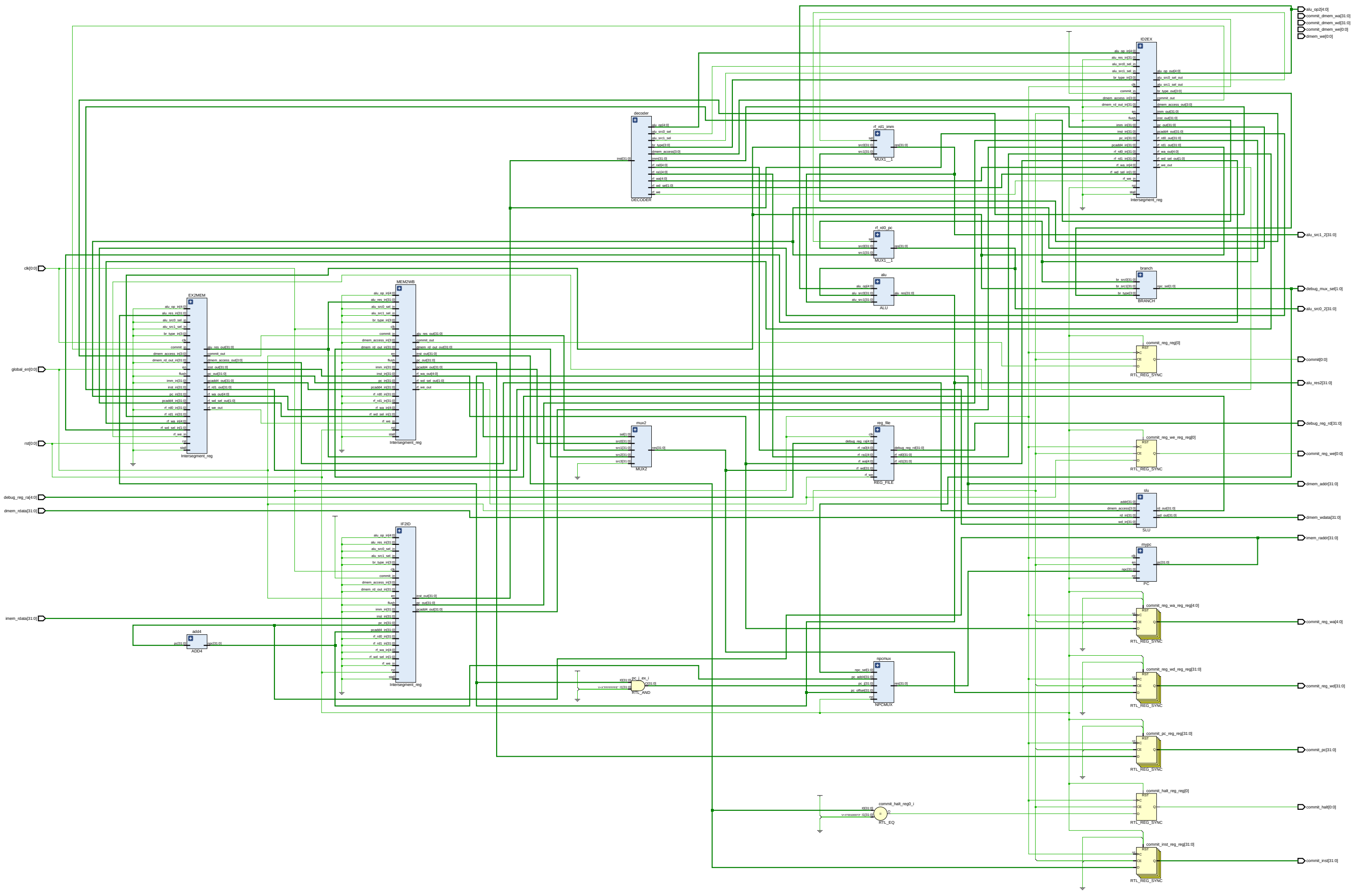
```
                            wd_out={rd_in[31:16],wd_in[7:0],rd_in[7:0]};
                        end
                2'b01: begin
                            rd_out={{24{rd_in[23]}},rd_in[23:16]};
                            wd_out={rd_in[31:24],wd_in[7:0],rd_in[15:0]};
                         end
                2'b01: 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'b01: 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'b01: 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
            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
            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] 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,
```

```verilog
    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] 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;

        alu_op_out=5'bz;
        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 (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;

            alu_op_out=5'bz;
            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;

            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
```

2.CPU_tb.v

```
/////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2024/04/09 17:02:48
// Design Name:
// Module Name: CPU_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////


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;

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[10:2]),      // 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)
);

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
```


clk[0:0]
global_en[0:0]
rst[0:0]
debug_reg_ra[4:0]
dmem_rdata[31:0]
imem_rdata[31:0]
EX2MEM
Intersegment_reg
MEM2WB
Intersegment_reg
IF2ID
Intersegment_reg
add4
ADD4
decoder
DECODER
ID2EX
Intersegment_reg
rf_rd1_imm
MUX1__1
rf_rd0_pc
MUX1__1
alu
ALU
branch
BRANCH
mux2
MUX2
reg_file
REG_FILE
slu
SLU
mypc
PC
npcmux
NPCMUX
RTL_AND
commit_halt_reg0_i
RTL_EQ
commit_reg_reg[0]
commit_reg_we_reg_reg[0]
commit_reg_wa_reg_reg[4:0]
commit_reg_wd_reg_reg[31:0]
commit_pc_reg_reg[31:0]
commit_halt_reg_reg[0]
commit_inst_reg_reg[31:0]
RTL_REG_SYNC
alu_op2[4:0]
commit_dmem_wa[31:0]
commit_dmem_wd[31:0]
commit_dmem_we[0:0]
dmem_we[0:0]
alu_src1_2[31:0]
debug_mux_sel[1:0]
alu_src0_2[31:0]
commit[0:0]
alu_res2[31:0]
debug_reg_rd[31:0]
commit_reg_we[0:0]
dmem_addr[31:0]
dmem_wdata[31:0]
imem_raddr[31:0]
commit_reg_wa[4:0]
commit_reg_wd[31:0]
commit_pc[31:0]
commit_halt[0:0]
commit_inst[31:0]