

lab4 实验报告

李毅PB22051031

第一部分 实验内容

1.访存控制单元设计

请根据自己选择的指令集，设计访存控制单元 SL_UNIT，以正确处理访存指令。你可以结合仿真证明自己的设计。

2.搭建单周期CPU

请正确实现 CPU 的各个功能模块，并根据数据通路将其正确连接。最终在 FPGAOL 上上板运行，并通过我们给出的测试程序。

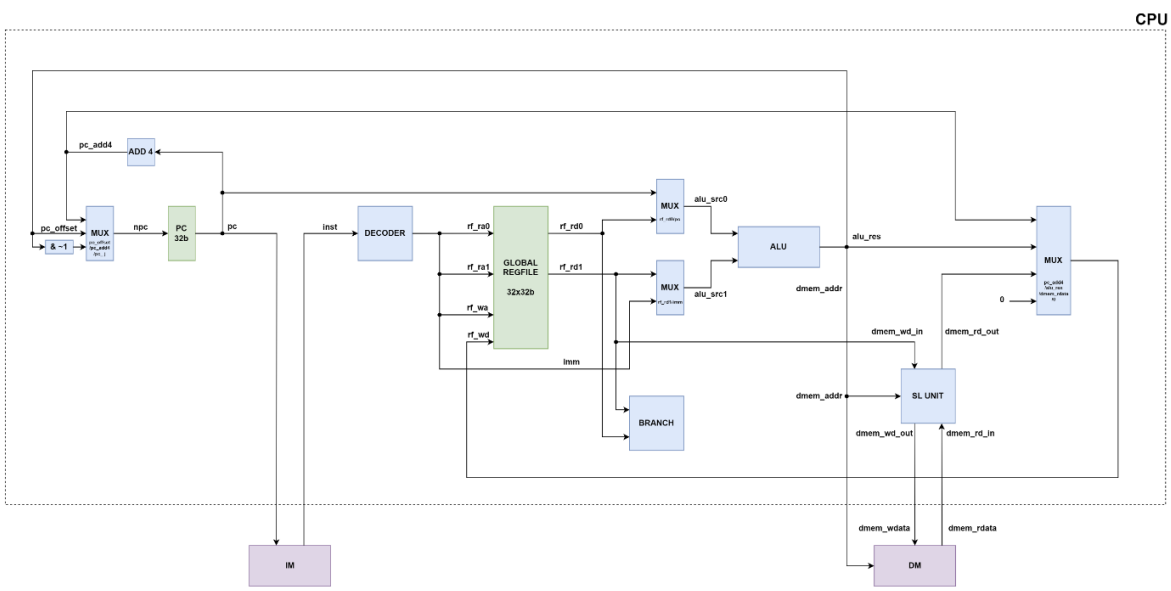
3.斐波那契数列

请将 Lab1 中编写的斐波那契数列程序（普通版本、大整数版本均可）导出为 COE 文件，在自己设计的 CPU 上运行。相关数据的输入、输出方式不限。

第二部分 实验过程

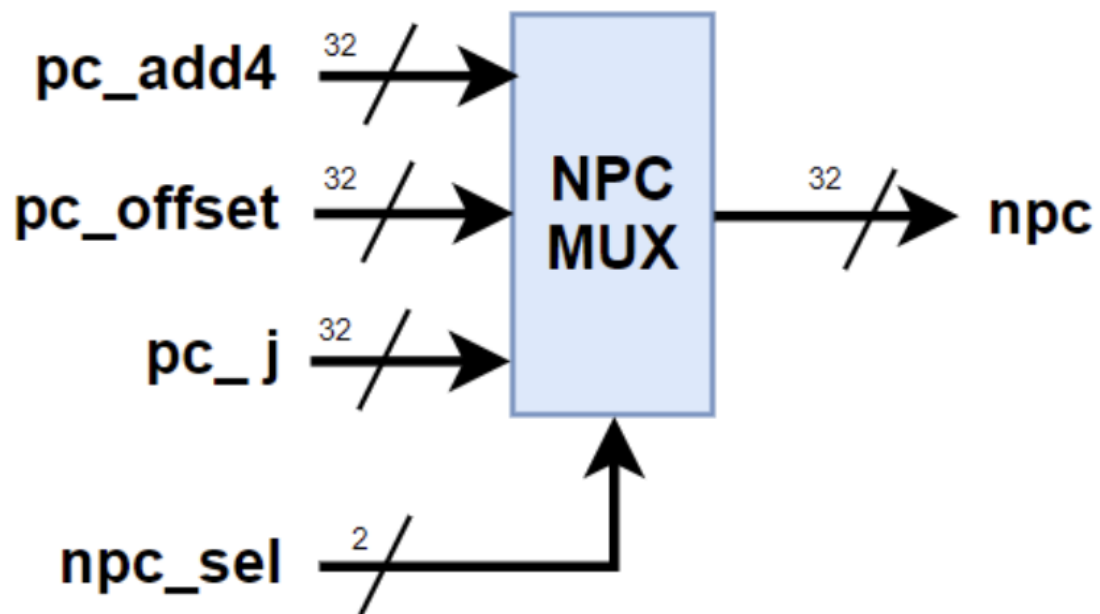
2.1 实验设计

2.1.1 CPU数据通路图



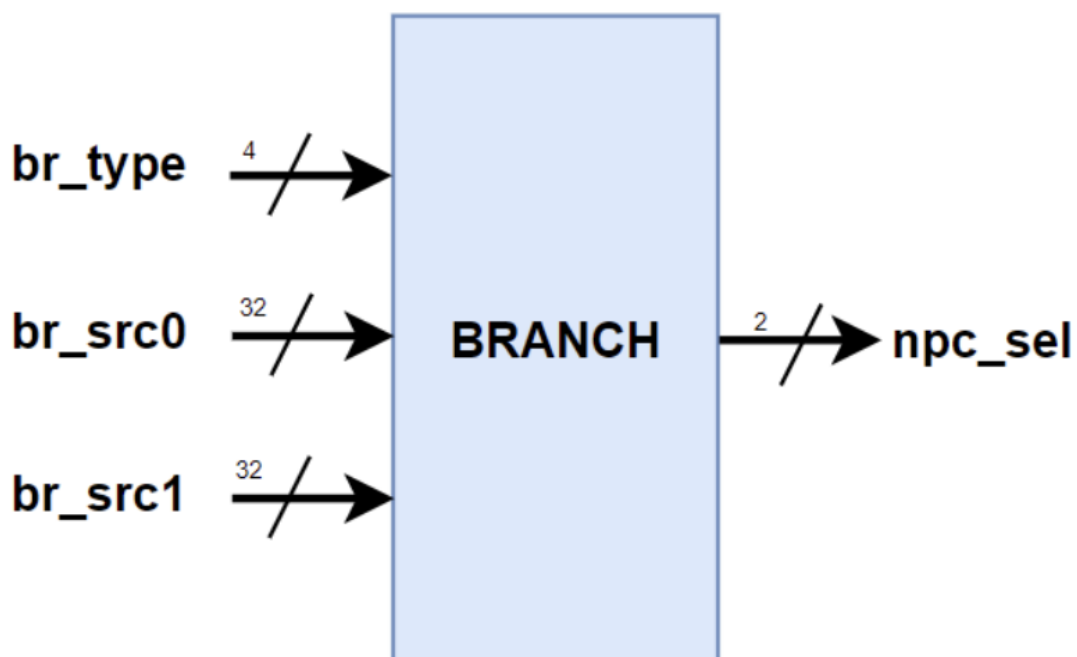
2.1.2 NPC选择器

单周期 CPU 需要能够执行分支指令，所以下一个周期的 PC 值就不一定是 $PC + 4$ 了。为此，我们需要一个数据选择器选择合适的值来传递给 PC 寄存器，下面是NPC选择器数据通路。



2.1.3 分支模块

与教材不同，我们把条件跳转指令的逻辑判断从 ALU 中移出，封装成了独立的模块。Branch 模块是专门用来处理分支指令的模块。它接收来自 Decoder 的控制信号，以及来自寄存器堆的两个数据。根据这两个数据之间的大小关系以及控制信号，Branch 模块最终产生相应的跳转信号。以下是本次实验中的分支模块结构数据通路图。

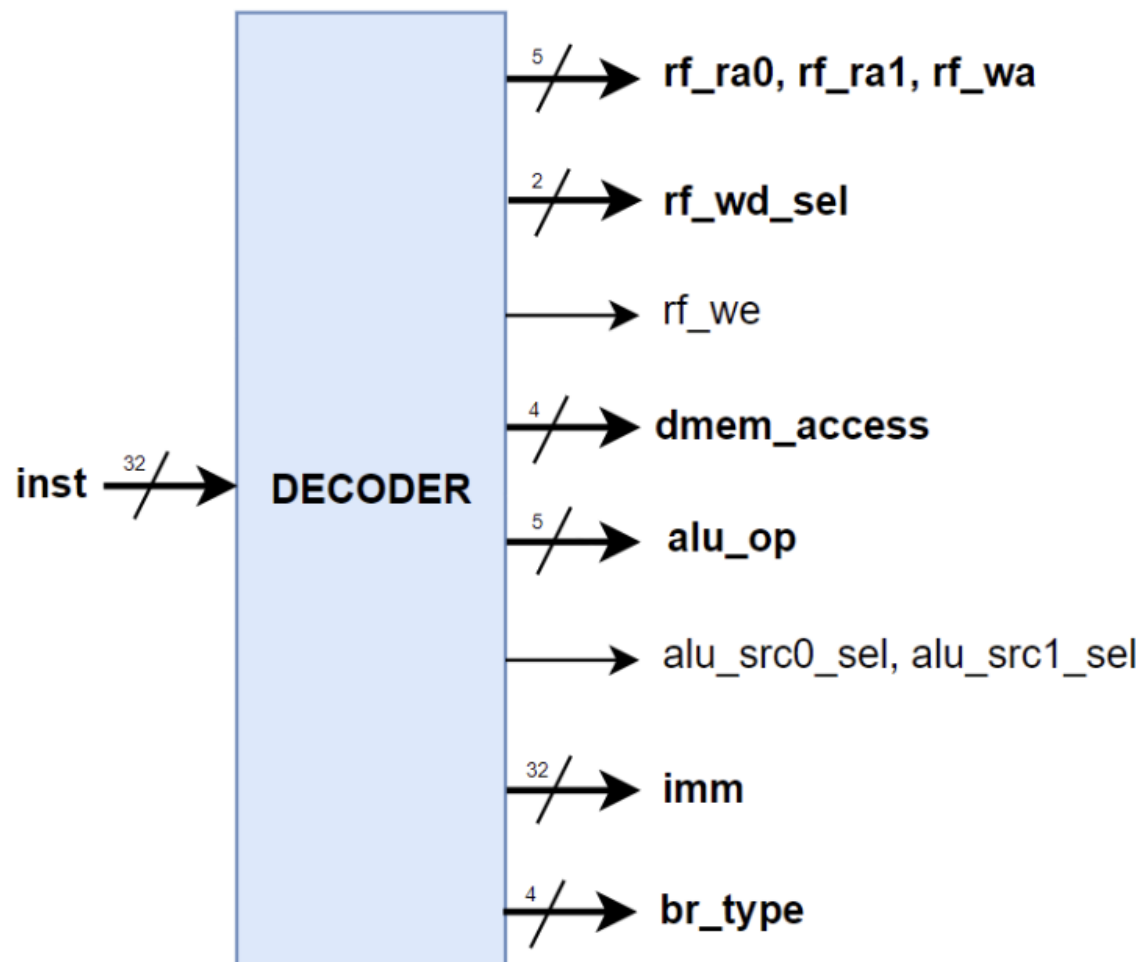


各个端口的作用是：

- br_type：分支跳转的类型；
- br_src0、br_src1：来自寄存器堆的两个数据；
- npc_sel：npc 选择器的控制信号；

2.1.4 译码器

为了实现完整的单周期 CPU，你需要修改你在 lab3 中的 Decoder 设计，处理简单单周期中没有实现的指令，以下是本次实验中的译码器的结构示意图。



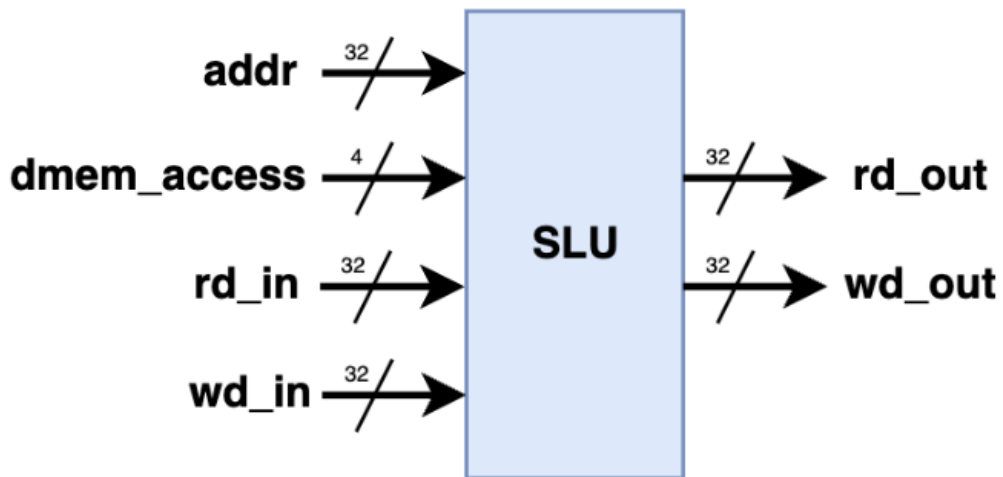
相较于 lab3，新增的各个输出端口的作用是：

- dmem_access：访存类型；
- rf_wd_sel：寄存器堆写回数据选择器的选择信号；
- br_type：分支跳转的类型；

这些信号的生成和 lab3 中描述的一样，都需要根据 inst 的数值，结合对应指令集的译码规则得出。

2.1.5 访存控制单元

访存控制单元主要是用于实现非整字访存指令的。

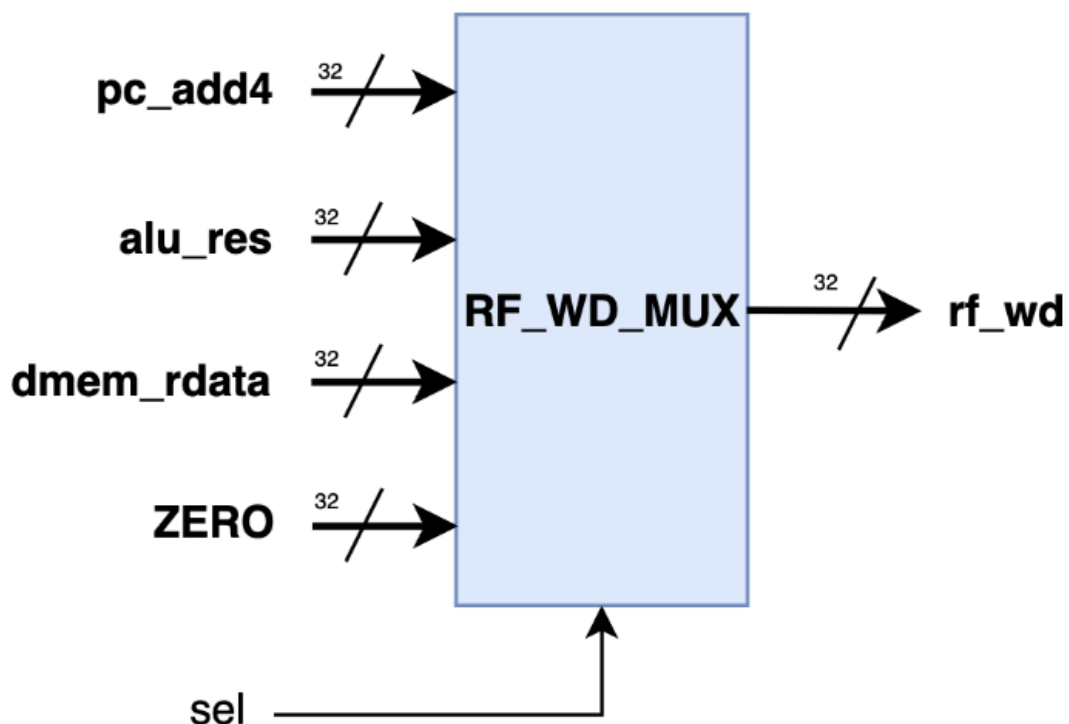


各个端口的作用是：

- addr：访存的地址；
- dmem_access：访存类型；
- rd_in：从存储器读到的原始数据；
- wd_in：要写入存储器的原始数据；
- rd_out：根据 dmem_access 处理过的读到的数据；
- wd_out：根据 dmem_access 处理过的要写入的数据；

2.1.6寄存器堆写回选择器

要写回寄存器堆的数据来源由你的 CPU 具体实现决定，在助教的实现中，要写回的数据有四个来源，PC 加四，ALU 的结果、从存储器读到的值和 0。实际上这里的 0 是没有实际作用的，但三选一和四选一选择器的资源开销基本一致，所以我们额外引入了一个输入端口，用一个四选一数据选择器用于控制写回的数据。



2.2 核心代码

2.2.1 NPC选择器

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

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

always @(*) 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=32'b0;
    endcase
end

endmodule
```

2.2.2 分支模块

```
module BRANCH(
    input               [ 3 : 0]       br_type,

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

    output reg         [ 1 : 0]       npc_sel
);

always @(*) begin
    case (br_type)
        4'b0000: 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
```

2.2.3 译码器

```
module DECODER (
    input                [31 : 0]    inst,

    output reg           [ 4 : 0]    alu_op,

    output reg           [ 3 : 0]    dmem_access, //访存
    output reg           [ 0 : 0]    dmem_we,
    output reg           [31 : 0]    imm,

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

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

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

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

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

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

//imm
always @(*) begin
    case (opcode)
        7'b0110111: imm={inst[31:12],12'b0}; //U type lui
        7'b0010111: imm={inst[31:12],12'b0}; //U type auipc
        7'b1101111: imm=
{{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0}; //jal
        7'b1100111: imm={{20{inst[31]}},inst[31:20]}; //jalr
        7'b1100011: if(funct3==3'b110 || funct3==3'b111) begin
            imm=
{{12{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //b type u
        end
        else begin
            imm=
{{12{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //b type
        end
        7'b0000011: if(funct3==3'b100 || funct3==3'b101) begin
            imm={20'b0,inst[31:20]}; //lbu, lhu
        end
    endcase
end
```

```

        end
    else begin
        imm={{20{inst[31]}},inst[31:20]}; //lb, lh, lw
    end
7'b0100011: imm={{20{inst[31]}},inst[31:25],inst[11:7]}; //sw sb sh
7'b0010011: if (funct3==3'b101 || funct3==3'b001) begin
    imm={{27{inst[24]}},inst[24:20]}; //srai
end
/*
else if(funct3==3'b011)begin
    imm={20'b0,inst[31:20]}; //sltiu
end
*/
else begin
    imm={{20{inst[31]}},inst[31:20]};
end
default: imm=32'b0;
endcase
end

//aluop
always @(*) begin
    case (opcode)
7'b0110011: begin
    case (funct)
        10'b000_000000: alu_op=5'B00000; //add
        10'b000_010000: alu_op=5'B00010; //sub
        10'b001_000000: alu_op=5'B01110; //sll
        10'b010_000000: alu_op=5'B00100; //slt
        10'b011_000000: alu_op=5'B00101; //sltiu
        10'b100_000000: alu_op=5'B01011; //xor
        10'b101_000000: alu_op=5'B01111; //srl
        10'b101_010000: alu_op=5'B10000; //sra
        10'b110_000000: alu_op=5'B01010; //or
        10'b111_000000: 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_src1_sel=1, 来自imm
always @(*) begin
    case (opcode)
        7'b0010111: alu_src0_sel=1; //auipc
        7'b0110111: alu_src0_sel=0; //lui
        7'b110111: 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'b110111: 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'b0000; //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

```

2.2.4 访存控制单元

```

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

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

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

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

```

```

        wd_out={wd_in[7:0],rd_in[23:0]};
    end
    default:begin
        rd_out=rd_in;
        wd_out=wd_in;
    end
endcase
end
4'b1000:begin
    case (addr[1:0])
        2'b00: begin
            rd_out={24'b0,rd_in[7:0]};
            wd_out={rd_in[31:8],wd_in[7:0]};
        end
        2'b01: begin
            rd_out={24'b0,rd_in[15:8]};
            wd_out={rd_in[31:16],wd_in[7:0],rd_in[7:0]};
        end
        2'b10: begin
            rd_out={24'b0,rd_in[23:16]};
            wd_out=
{rd_in[31:24],wd_in[7:0],rd_in[15:0]};
        end
        2'b11: begin
            rd_out={24'b0,rd_in[31:24]};
            wd_out={wd_in[7:0],rd_in[23:0]};
        end
    default:begin
        rd_out=rd_in;
        wd_out=wd_in;
    end
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
    endcase
end

```

```

        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

```

2.2.5寄存器堆写回选择器

```

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

    output [WIDTH-1 : 0] res
);

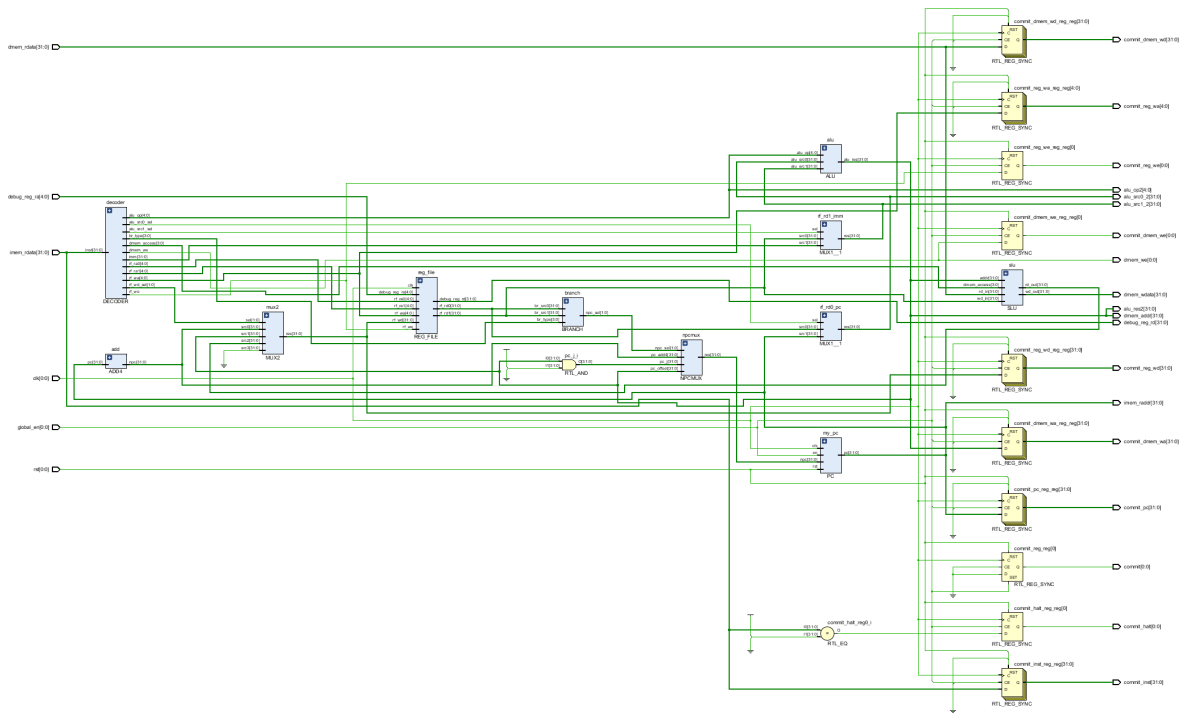
    assign res = sel[1] ? (sel[0] ? src3 : src2) : (sel[0] ? src1 : src0);

endmodule

```

第三部分 实验结果

3.1 电路分析结果



3.2 电路仿真结果

仿真文件如下(使用测试程序 2（分支与访存测试）):

```

////////////////////////////////////
/
// 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;

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)
    );

    initial begin
        clk = 0;
        global_en=1;
        #1
        rst = 1;
        #1
        rst = 0;
        we = 0;
        d=32'b0;
        #9000
        debug_reg_ra=0;
        #1
        repeat(32) begin
            #0.1
            debug_reg_ra=debug_reg_ra+1;
        end
        $finish;
    end

    always #10 clk = ~clk;
endmodule

```

仿真结果如下:



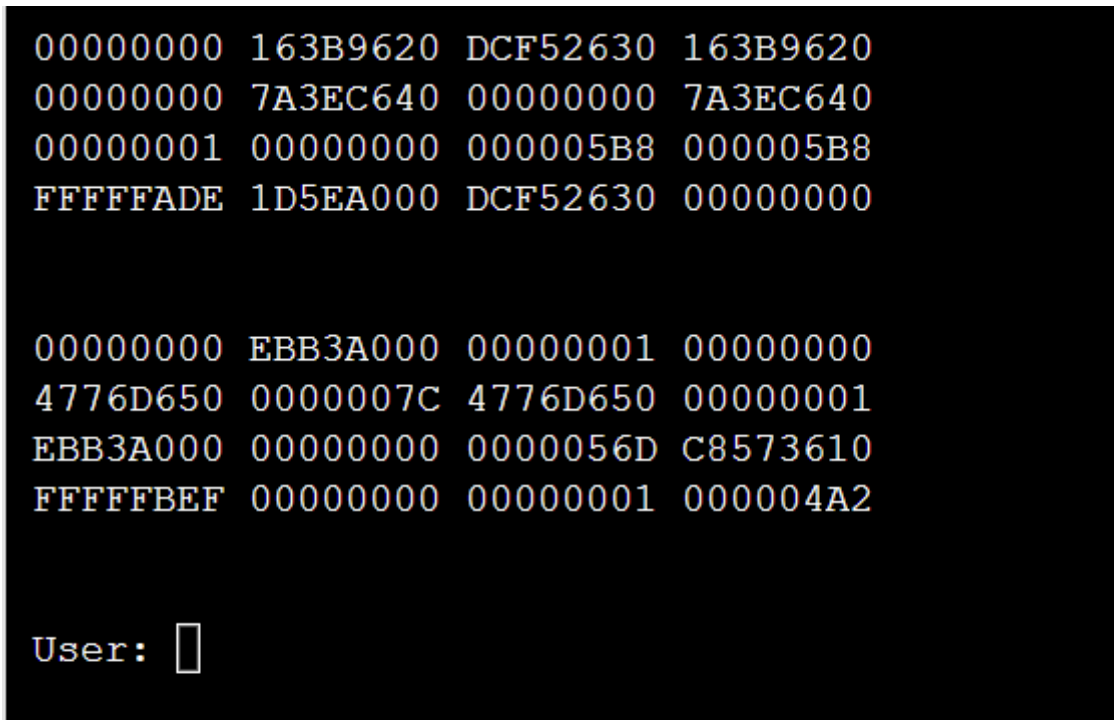
RARS运行结果如下:

Name	Number	Value
zero	0	0x00000000
ra	1	0x100193bf
sp	2	0x00000000
gp	3	0x10029317
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x10029ec8
t2	7	0x00000000
s0	8	0x1001f720
s1	9	0x1003e27a
a0	10	0x1001f860
a1	11	0x00000000
a2	12	0x10019699
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x10029afa
s3	19	0x00000000
s4	20	0x1001969a
s5	21	0x00000000
s6	22	0x10015f7a
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x1003947e
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x1002a163
t5	30	0x1001959c
t6	31	0x00000000
pc		0x00400724

保持一致。

3.3 上板结果

3.3.1 测试程序1



00000000 163B9620 DCF52630 163B9620
00000000 7A3EC640 00000000 7A3EC640
00000001 00000000 000005B8 000005B8
FFFFFFADE 1D5EA000 DCF52630 00000000

00000000 EBB3A000 00000001 00000000
4776D650 0000007C 4776D650 00000001
EBB3A000 00000000 0000056D C8573610
FFFFFFBEF 00000000 00000001 000004A2

User:

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

RR 10 16;

input

segplay(sharing with led) hexplay

00 100073

RARS运行结果:

Name	Number	Value
zero	0	0x00000000
ra	1	0x163b9620
sp	2	0xdcf52630
gp	3	0x163b9620
tp	4	0x00000000
t0	5	0x7a3ec640
t1	6	0x00000000
t2	7	0x7a3ec640
s0	8	0x00000001
s1	9	0x00000000
a0	10	0x000005b8
a1	11	0x000005b8
a2	12	0xffffffffade
a3	13	0x1d5ea000
a4	14	0xdcf52630
a5	15	0x00000000
a6	16	0x00000000
a7	17	0xebb3a000
s2	18	0x00000001
s3	19	0x00000000
s4	20	0x4776d650
s5	21	0x0000007c
s6	22	0x4776d650
s7	23	0x00000001
s8	24	0xebb3a000
s9	25	0x00000000
s10	26	0x0000056d
s11	27	0xc8573610
t3	28	0xffffffffbef
t4	29	0x00000000
t5	30	0x00000001
t6	31	0x000004a2
pc		0x00400684

3.3.2 测试程序2

```
00000000 100193BF 00000000 10029317
00000000 00000000 10029EC8 00000000
1001F720 1003E27A 1001F860 00000000
10019699 00000000 00000000 00000000
```

```
00000000 00000000 10029AFA 00000000
1001969A 00000000 10015F7A 00000000
00000000 1003947E 00000000 00000000
00000000 1002A163 1001959C 00000000
```

User:

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

RR 10 16;

input

segplay(sharing with led) hexplay

00 100073

RARS运行结果

Name	Number	Value
zero	0	0x00000000
ra	1	0x100193bf
sp	2	0x00000000
gp	3	0x10029317
tp	4	0x00000000
t0	5	0x00000000
t1	6	0x10029ec8
t2	7	0x00000000
s0	8	0x1001f720
s1	9	0x1003e27a
a0	10	0x1001f860
a1	11	0x00000000
a2	12	0x10019699
a3	13	0x00000000
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x10029afa
s3	19	0x00000000
s4	20	0x1001969a
s5	21	0x00000000
s6	22	0x10015f7a
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x1003947e
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x1002a163
t5	30	0x1001959c
t6	31	0x00000000
pc		0x00400724

3.3.3 斐波那契数列

00000000 00000000 00000000 00000000
00000000 0000000E 00000000 00000000
00000000 00000000 00000090 000000E9
0000000D 000000E9 00000000 00000000


00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000
00000000 00000000 00000000 00000000

User:

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

RR 10 16;
input

segplay(sharing with led) hexplay



00100073

RARS运行结果（结果在a1寄存器中）

Name	Number	Value
zero	0	0x00000000
ra	1	0x00000000
sp	2	0x7ffffefc
gp	3	0x00000000
tp	4	0x00000000
t0	5	0x0000000e
t1	6	0x00000000
t2	7	0x00000000
s0	8	0x00000000
s1	9	0x00000000
a0	10	0x00000090
a1	11	0x000000e9
a2	12	0x000000d
a3	13	0x000000e9
a4	14	0x00000000
a5	15	0x00000000
a6	16	0x00000000
a7	17	0x00000000
s2	18	0x00000000
s3	19	0x00000000
s4	20	0x00000000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0x00000000
s9	25	0x00000000
s10	26	0x00000000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x00000000
pc		0x00400038

第四部分 思考题

1.假设我们的存储器支持掩码访问，其对应接口如下：

```
module MEM (  
    input    [0:0]      clk,  
    input    [9:0]      a,  
    output   [31:0]     spo,  
    input    [0:0]      we,  
    input    [31:0]     d,  
    input    [3:0]      mask  
);
```

其中，mask 为高电平有效的控制信号，mask[0] 控制当前正在访问的字的最低字节是否有效；mask[3] 控制当前正在访问的字的最高字节是否有效。例如，如果 a = 0x4，we = 1，d = 0x12345678，mask=4'B0110，则数据存储器对应的操作为

```
M[0x4] <- M[0x4]  
M[0x5] <- 0x56  
M[0x6] <- 0x34  
M[0x7] <- M[0x7]
```

读操作则会将 mask 为 0 的字节置为 0。例如，如果 a = 0x4，we = 0，mask=4'B0110，则读出的结果为

```
{8'B0, M[0x6], M[0x5], 8'B0}
```

请重新设计 SL_UNIT 单元。你可以自行添加相关模块所需要的端口。注意：本题的前提是我们假设存在这样的存储器 IP 核，你并不需要实现这个 IP 核，也不需要仿真或上板，仅阐述 SL_UNIT 单元的设计方案即可。

```
module SLU (  
    input    [31 : 0]      addr,  
    input    [ 3 : 0]      dmem_access,  
  
    output   reg           [31 : 0]      mask  
);  
  
always @(*) begin  
    case (dmem_access)  
        4'b0001:begin  
            case (addr[1:0])  
                2'b00: begin  
                    mask=4'b0001;  
                end  
                2'b01: begin  
                    mask=4'b0010;  
                end  
                2'b01: begin  
                    mask=4'b0100;  
                end  
                2'b01: begin  
                    mask=4'b1000;  
                end  
                default:begin  
                    mask=4'b0000;  
                end  
            endcase  
        end  
    endcase  
end
```

```

4'b1000:begin
    case (addr[1:0])
        2'b00: begin
            mask=4'b0001;
        end
        2'b01: begin
            mask=4'b0010;
        end
        2'b01: begin
            mask=4'b0100;
        end
        2'b01: begin
            mask=4'b1000;
        end
        default:begin
            mask=4'b0000;
        end
    endcase
end
4'b1100:begin
    case (addr[1])
        1'b0:begin
            mask=4'b0011;
        end
        1'b1:begin
            mask=4'b1100;
        end
        default:begin
            mask=4'b0000;
        end
    endcase
end
4'b0011:begin
    case (addr[1])
        1'b0:begin
            mask=4'b0011;
        end
        1'b1:begin
            mask=4'b1100;
        end
        default:begin
            mask=4'b0000;
        end
    endcase
end
4'b1111:begin
    mask=4'b1111;
end
default:begin
    mask=4'b1111;
end
endcase
end

endmodule

```


2.请指出本次实验的 CPU 中可能的关键路径。

load类指令需要经历取指译码执行访存写回五步，时间最长，为关键路径

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        [31:0]      alu_src0_2,
        //output        [31:0]      alu_res2
    );

    // TODO
    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;

    /* ----- */
    /*                               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
```

```
        // !!!! 请注意根据自己的具体实现替换 <= 右侧的信号 !!!!
```

```
        commit_reg          <= 1'H1;           // 不需要改动
        commit_pc_reg       <= cur_pc;         // 需要为当前的 PC
        commit_inst_reg     <= cur_inst;       // 需要为当前的指令
        commit_halt_reg     <= cur_inst == 32'H00100073; // 注意! 请根据指
```

令集设置 HALT_INST!

```
        commit_reg_we_reg   <= rf_we;         // 需要为当前的寄存
```

器堆写使能

```
        commit_reg_wa_reg   <= rf_wa;         // 需要为当前的寄存
```

器堆写地址

```
        commit_reg_wd_reg   <= rf_wd;         // 需要为当前的寄存
```

器堆写数据

```
        commit_dmem_we_reg  <= dmem_we;       // 不需要改
```

动

```
        commit_dmem_wa_reg  <= dmem_addr;     // 不需要
```

改动

```
        commit_dmem_wd_reg  <= dmem_rdata;    // 不需要
```

改动

```
    end
```

```
end
```

```

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

```

```

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

endmodule

module DECODER (
    input                [31 : 0]      inst,

    output    reg        [ 4 : 0]      alu_op,

    output    reg        [ 3 : 0]      dmem_access, //访存
    output    reg        [ 0 : 0]      dmem_we,
    output    reg        [31 : 0]      imm,

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

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

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

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

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

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

//imm
always @(*) begin
    case (opcode)
        7'b0110111: imm={inst[31:12],12'b0}; //U type lui
        7'b0010111: imm={inst[31:12],12'b0}; //U type auipc
        7'b1101111: imm=
        {{12{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0}; //jal
        7'b1100111: imm={{20{inst[31]}},inst[31:20]}; //jalr
        7'b1100011: if(funct3==3'b110 || funct3==3'b111) begin
            imm=
            {{12{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0}; //b type u
        end
    endcase
end

```

```

        else begin
            imm=
{{12{inst[31]}}},inst[7],inst[30:25],inst[11:8],1'b0;//b type
        end
        7'b0000011: if(funcnt3==3'b100 || funcnt3==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 (funcnt3==3'b101 || funcnt3==3'b001) begin
            imm={{27{inst[24]}}},inst[24:20]};//srai
        end
        /*
        else if(funcnt3==3'b011)begin
            imm={20'b0,inst[31:20]};//sltiu
        end
        */
        else begin
            imm={{20{inst[31]}}},inst[31:20]};
        end
        default: imm=32'b0;
    endcase
end

//aluop
always @(*) begin
    case (opcode)
        7'b0110011: begin
            case (funct)
                10'b000_0000000: alu_op=5'B00000;//add
                10'b000_0100000: alu_op=5'B00010;//sub
                10'b001_0000000: alu_op=5'B01110;//sll
                10'b010_0000000: alu_op=5'B00100;//slt
                10'b011_0000000: alu_op=5'B00101;//slti
                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(funcnt7==7'b0000000)begin
                    alu_op=5'B01111;//srli
                end
                else begin
                    alu_op=5'B10000;//srai
                end
                3'b110: alu_op=5'B01010;//ori
            end
        end
    endcase
end

```

```

        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,ldu
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

```



```

    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

```

```

end

//br_type
always @(*) begin
    case (opcode)
        7'b1100011: begin
            case (funct3)
                3'b000:br_type=4'b0000; //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
);

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=reg_file[rf_ra0];
assign rf_rd1=reg_file[rf_ra1];
assign debug_reg_rd=reg_file[debug_reg_ra];

always @(posedge clk) begin
    if (rf_we) begin
        reg_file[rf_wa] <= rf_wd;
    end
    reg_file[5'b0] <= 32'b0;
end

endmodule

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

    output [WIDTH-1 : 0] res
);

    assign res = sel ? src1 : src0;

endmodule

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

module BRANCH(
    input [3 : 0] br_type,

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

    output reg [1 : 0] npc_sel
);

always @(*) begin
    case (br_type)
        4'b0000: 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;
    endcase
end
endmodule

```

```

        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,

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

always @(*) 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=32'b0;
    endcase
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
    end
endcase
end
4'b1100:begin
    case (addr[1])
        1'b0:begin
            rd_out={16'b0,rd_in[15:0]};
            wd_out={rd_in[31:16],wd_in[15:0]};

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

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

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

```

```

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

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

    output             [WIDTH-1 : 0]    res
);

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

endmodule

```

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;

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)
);

initial begin
    clk = 0;
    global_en=1;
    #1
    rst = 1;
    #1
    rst = 0;
    we = 0;
    d=32'b0;
    #9000
    debug_reg_ra=0;
    #1
    repeat(32) begin
        #0.1
        debug_reg_ra=debug_reg_ra+1;
    end
    $finish;
end

always #10 clk = ~clk;
endmodule

```

