lab3 实验报告

李毅PB22051031

第一部分 实验内容

1.译码器设计

选择RV32I指令集，设计译码器 Decoder 模块，以正确生成控制信号。并结合仿真证明自己的设计。

2.搭建单周期CPU（部分）

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。能够在 FPGAOL完成上板运行，并通过助教给出的测试程序。

3.实验平台

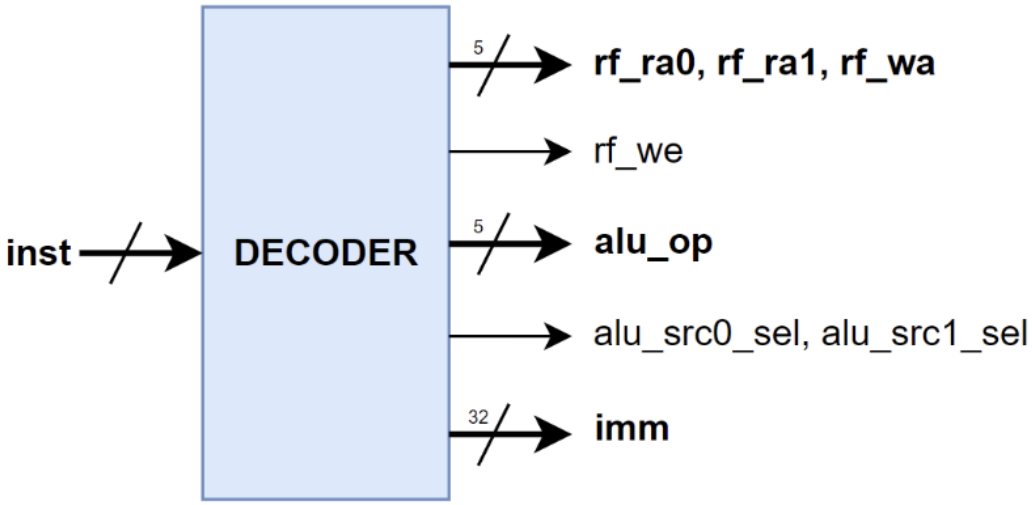
vivado，FPGAOL

第二部分 实验过程

2.1 实验设计

实验项目1：

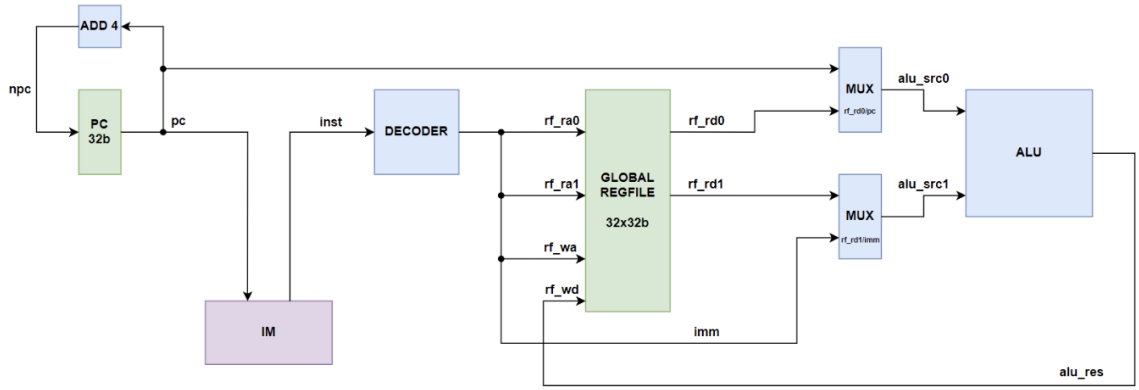
译码器数据通路如图：



输入32位指令码，根据输入的指令生成相应的控制与数据信号。

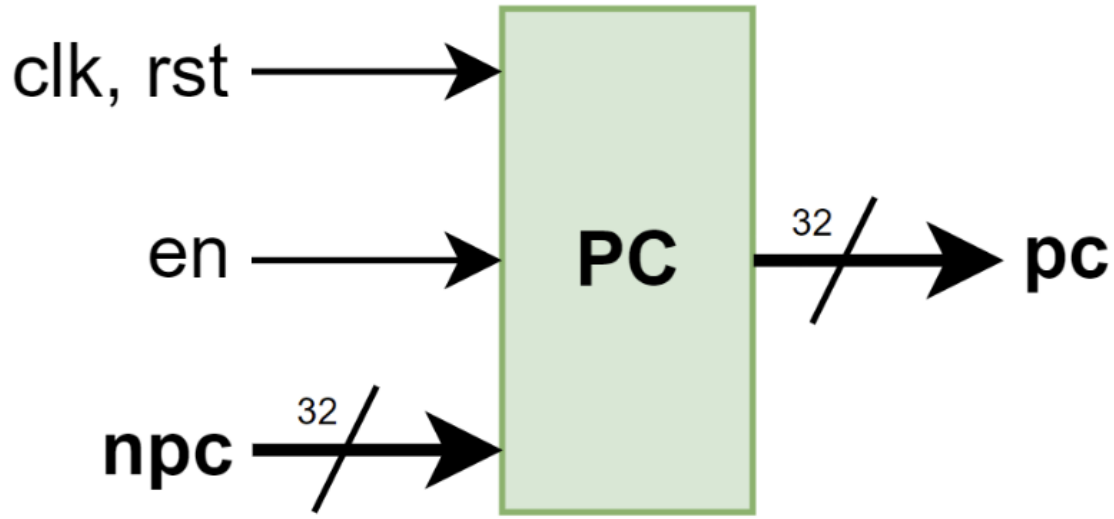
实验项目2：

CPU数据通路如图：



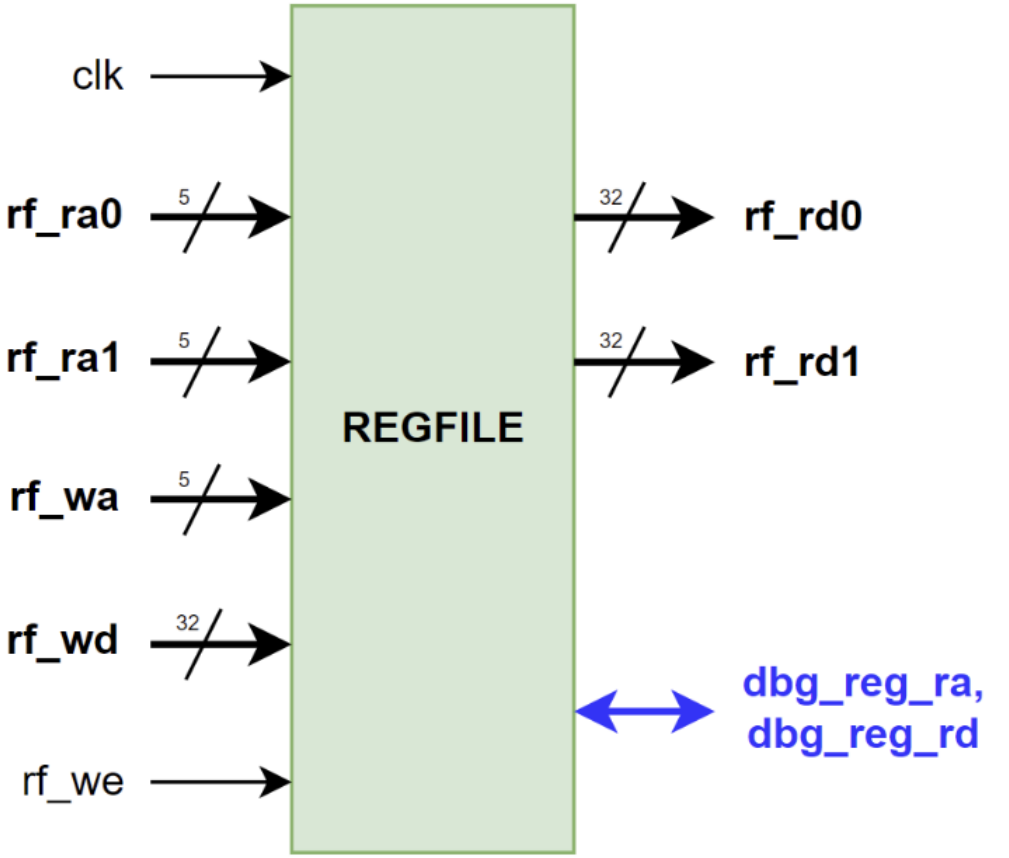
各模块数据通路如图：

PC模块：



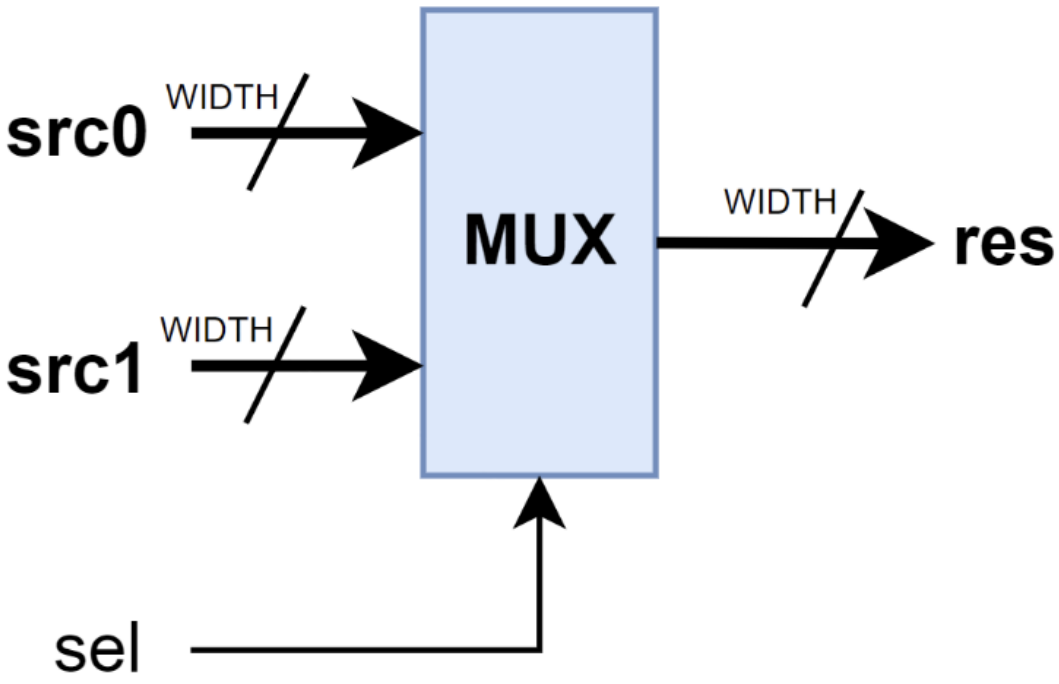
PC 寄存器时刻存储了正在执行的指令的地址。它的功能是将当前指令的地址传递给指令存储器，从而读出此时正确的指令内容。同时，它也需要能够接受下一条指令地址的输入，并在时钟上升沿到来时更新自己的值，从而实现了指令的连续运行。

寄存器堆：



本次实验我们需要在 Lab2 的寄存器堆基础上，为其添加一组 debug 接口 debug_reg_ra、debug_reg_rd。这一对端口与数据读端口功能一致，只用于仿真与上板时的调试。CPU 在正常运行时并不会用到这两个端口。

数据选择器：



ALU 的两个源操作数可能是寄存器堆的输出，也可能是立即数，还可能是当前 PC 的值。为此，我们需要为源操作数添加数据选择器，用于控制 ALU 的数据来源。

译码器：

见实验项目1.

ALU：

见lab2.

2.2 核心代码

译码器部分实现的核心代码如下：

```
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'b0010011:if (funct3==3'b101 || funct3==3'b001) begin
        imm={{27{inst[24]}},inst[24:20]};//srai
        end
        else begin
            imm={{20{inst[31]}},inst[31:20]};//else I type
        end
        7'b0110111: imm={inst[31:12],12'b0};//U type lui
        7'b0010111: imm={inst[31:12],12'b0};//U type auipc
        default: imm=32'b0;
    endcase
end

always @(*) begin
    if (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
    else if (opcode==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
    else if(opcode==7'b0110111)begin
        alu_op=5'B10010;//lui
    end
    else begin
        alu_op=5'B00000;//else to add
    end
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'b0110011: alu_src0_sel=0; //R-type
        7'b0010011: alu_src0_sel=0; //I-type
        7'b0010111: alu_src0_sel=1; //auipc
        7'b0110111: alu_src0_sel=0; //lui
        default: alu_src0_sel=0;
    endcase
end

always @(*) begin
    case (opcode)
        7'b0110011: alu_src1_sel=0;//R-type
        7'b0010011: alu_src1_sel=1;//I-type
        7'b0010111: alu_src1_sel=1;//auipc
        7'b0110111: alu_src1_sel=1;//lui
        default: alu_src1_sel=0;
    endcase
end

//rf_we
always @(*) begin
    case (opcode)
        7'b0110011: rf_we=1;//R-type
        7'b0010011: rf_we=1;//I-type
        7'b0010111: rf_we=1;//auipc
        7'b0110111: rf_we=1;//lui
        default: rf_we=1;
    endcase
end
```

PC部分实现的核心代码如下：

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

寄存器堆部分实现的核心代码如下：

```
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
```

CPU顶部模块实现的核心代码如下：

```
// 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;

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

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(cur_npc)
);
assign imem_raddr=cur_pc;
assign cur_inst=imem_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)
);

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)
);
```

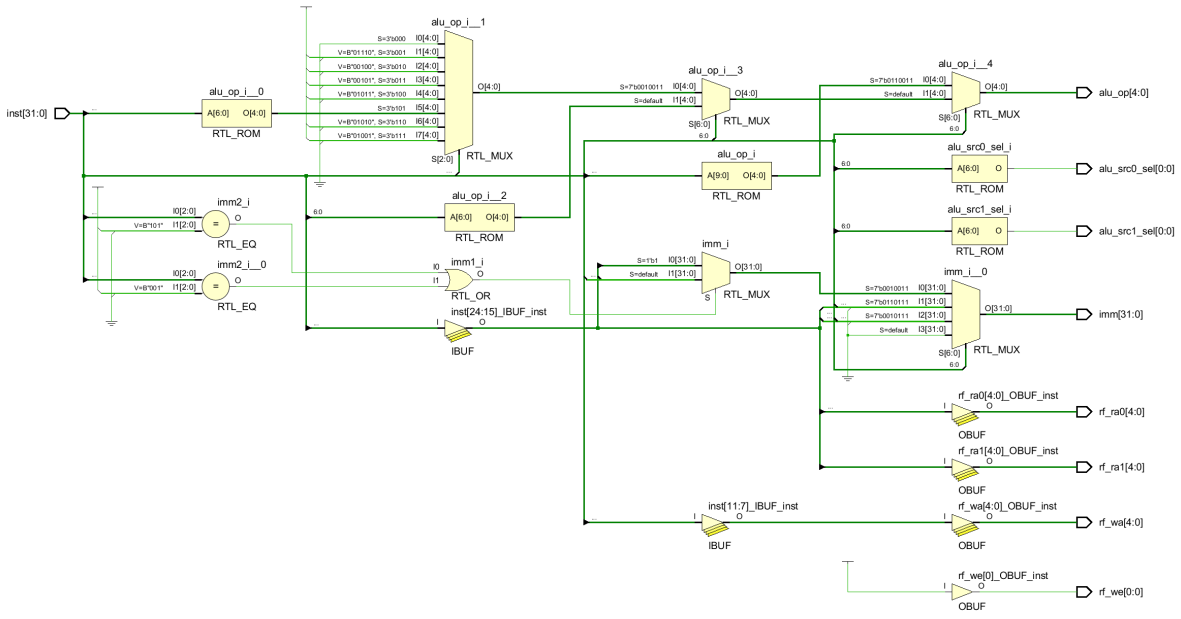
```
MUX1 rf_rd1_imm(
    .src0(rf_rd1),
    .src1(imm),
    .sel(alu_src1_sel),
    .res(alu_src1)
);

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

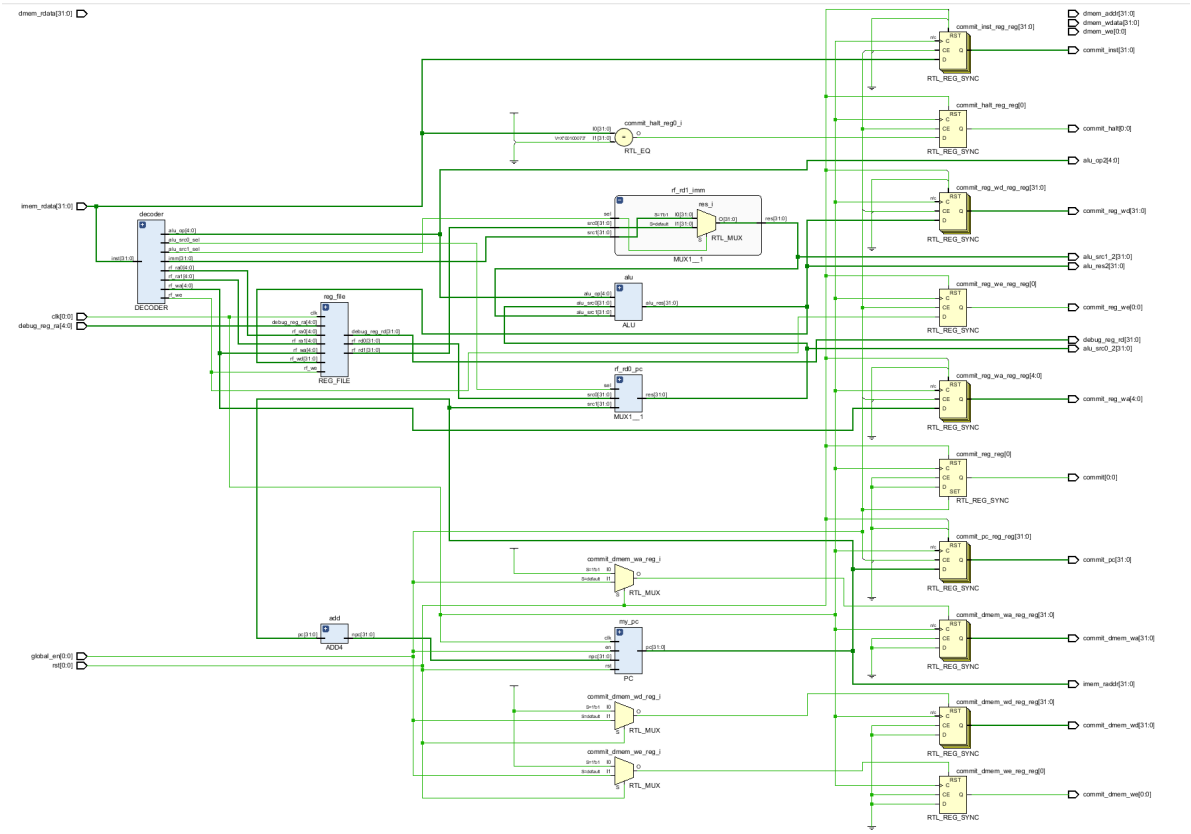
第三部分 实验结果

3.1 电路分析结果

3.1.1 译码器RTL电路



3.1.2 单周期CPU RTL电路



3.2 电路仿真结果

3.2.1 译码器仿真

仿真文件核心代码如下：

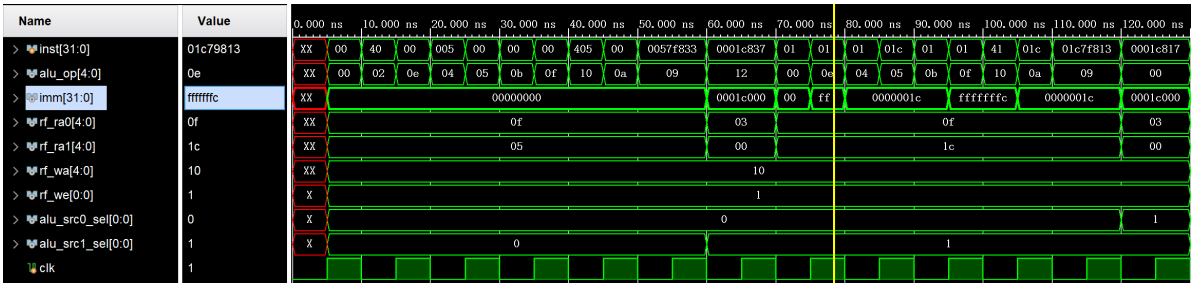
```
//rs2段为5，rs1段为15，rd段为16,imm段为28（1 1100）
initial begin
    clk = 0;

    #5
    inst=32'b0000000_00101_01111_000_10000_0110011;//add
    #5
    inst=32'b0100000_00101_01111_000_10000_0110011;//sub
    #5
    inst=32'b0000000_00101_01111_001_10000_0110011;//sll
    #5
    inst=32'b0000000_00101_01111_010_10000_0110011;//slt
    #5
    inst=32'b0000000_00101_01111_011_10000_0110011;//sltu
    #5
    inst=32'b0000000_00101_01111_100_10000_0110011;//xor
    #5
    inst=32'b0000000_00101_01111_101_10000_0110011;//srl
    #5
    inst=32'b0100000_00101_01111_101_10000_0110011;//sra
    #5
    inst=32'b0000000_00101_01111_110_10000_0110011;//or
    #5
    inst=32'b0000000_00101_01111_111_10000_0110011;//and
    #10
    inst=32'b0000000_00000_00011_100_10000_0110111;//lui
    #10
    inst=32'b0000000_11100_01111_000_10000_0010011;//addi
```

```
    #5
    inst=32'b0000000_11100_01111_001_10000_0010011;//slli
    #5
    inst=32'b0000000_11100_01111_010_10000_0010011;//slti
    #5
    inst=32'b0000000_11100_01111_011_10000_0010011;//sltiu
    #5
    inst=32'b0000000_11100_01111_100_10000_0010011;//xori
    #5
    inst=32'b0000000_11100_01111_101_10000_0010011;//srli
    #5
    inst=32'b0100000_11100_01111_101_10000_0010011;//srai
    #5
    inst=32'b0000000_11100_01111_110_10000_0010011;//ori
    #5
    inst=32'b0000000_11100_01111_111_10000_0010011;//andi
    #10
    inst=32'b0000000_00000_00011_100_10000_0010111;//auipc
    #10

    $finish;
end
always #5 clk = ~clk;
```

结果如下，均成功完成译码。



3.2.2 CPU仿真

导入指定的coe模块，仿真得到的运行结果如下：

RARS得到的结果如下：

Name	Number	Value
zero	0	0x00000000
ra	1	0x004c8688
sp	2	0x00369694
gp	3	0xfff8c000
tp	4	0x0039e664
t0	5	0x0045467c
t1	6	0x000f2000
t2	7	0x00000000
s0	8	0xfff69000
s1	9	0xfff9c000
a0	10	0x00000000
a1	11	0xfff73000
a2	12	0x00000000
a3	13	0xfff6d000
a4	14	0xfff9e000
a5	15	0x00000000
a6	16	0x00047000
a7	17	0x000c8000
s2	18	0x0002d000
s3	19	0x0038c6ac
s4	20	0xfff2c000
s5	21	0x00000000
s6	22	0x00000000
s7	23	0x00000000
s8	24	0xfff54000
s9	25	0x00037000
s10	26	0xfff9c000
s11	27	0x00000000
t3	28	0x00000000
t4	29	0x00000000
t5	30	0x00000000
t6	31	0x000c2000
pc		0x004006bc

仿真结果与RARS得到的结果一致

3.3 上板结果

```
User: RR 0 16
USTC COD Project
User: S;
Current PC is at 0x00400000


User: R;
---------- CPU ----------


00000000 004C8688 00369694 FFF8C000
0039E664 0045467C 000F2000 00000000
FFF69000 FFF9C000 00000000 FFF73000
00000000 FFF6D000 FFF9E000 00000000


00047000 000C8000 0002D000 0038C6AC
FFF2C000 00000000 00000000 00000000
FFF54000 00037000 FFF9C000 00000000
00000000 00000000 00000000 000C2000


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

与RARS，仿真结果完全一致。

第四部分 思考题

1. **本次实验的 CPU 中，哪些模块用到了时钟信号?**

   寄存器堆，PC，commit调试模块

2. **请分别给出一条指令，以符合下面的描述：**

   (1) alu_src0 选择 pc;

   (2) alu_src0 选择 rf_rd0;

   (3) alu_src1 选择 rf_rd1;

   (4) alu_src1 选择 imm;

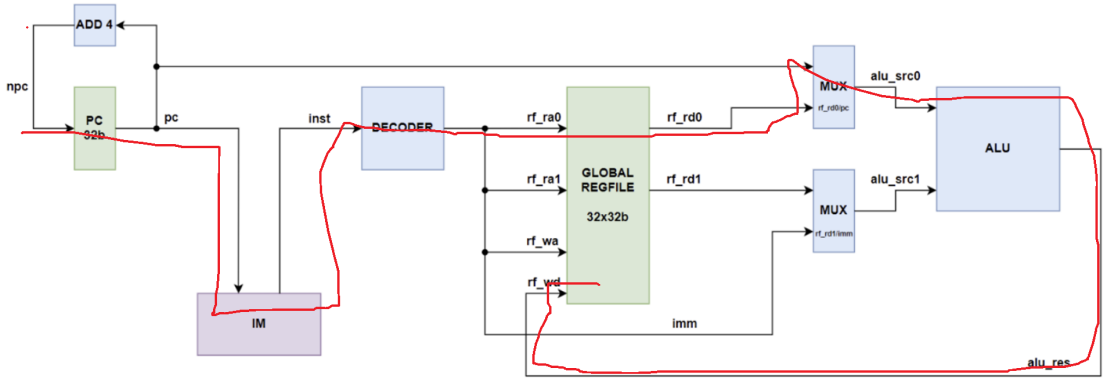
   (1) auipc

   (2)add

   (3)add

   (4)addi

3. **请指出本次实验的 CPU 中可能的关键路径；如果这条路径的延迟大于一个时钟周期，可能会带来什么影响?**



R型指令和I型指令执行为关键路径，如果这条路径的延迟大于一个时钟周期，会导致时序错误，如寄存器来不及写入就被错误的读出等。

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.decoder.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date: 2024/04/08 22:44:33
// Design Name:
// Module Name: Counter
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
```

```
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//

`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

`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 DECODER(
    input                   [31 : 0]            inst,

    output  reg             [ 4 : 0]            alu_op,
    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             [ 0 : 0]            alu_src0_sel,
    output  reg             [ 0 : 0]            alu_src1_sel
);

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'b0010011:if (funct3==3'b101 || funct3==3'b001) begin
        imm={{27{inst[24]}},inst[24:20]};//srai
        /*
            if (funct7==7'b0100000) begin
                imm={{27{inst[24]}},inst[24:20]};//srai
            end
            else begin
                imm={27'b0,inst[24:20]};//srli,slli
            end
        */
        end
        else begin
            imm={{20{inst[31]}},inst[31:20]};
        end
        /*
        else if(funct3==3'b011 || funct3==3'b100 || funct3==3'b110 ||
funct3==3'b111) begin
            imm={20'b0,inst[31:20]};//sltiu,xori,ori,andi
        end
        else begin
            imm={{20{inst[31]}},inst[31:20]};//addi,
        end
        */
        7'b0110111: imm={inst[31:12],12'b0};//U type lui
        7'b0010111: imm={inst[31:12],12'b0};//U type auipc
        default: imm=32'b0;
    endcase
end

always @(*) begin
    if (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
    else if (opcode==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
    else if(opcode==7'b0110111)begin
        alu_op=5'B10010;
    end
    else begin
        alu_op=5'B00000;//else to add
    end
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'b0110011: alu_src0_sel=0; //R-type
        7'b0010011: alu_src0_sel=0; //I-type
        7'b0010111: alu_src0_sel=1; //auipc
        7'b0110111: alu_src0_sel=0; //lui
        default: alu_src0_sel=0;
    endcase
end

always @(*) begin
    case (opcode)
        7'b0110011: alu_src1_sel=0;//R-type
        7'b0010011: alu_src1_sel=1;//I-type
        7'b0010111: alu_src1_sel=1;//auipc
        7'b0110111: alu_src1_sel=1;//lui
        default: alu_src1_sel=0;
    endcase
end

//rf_we
always @(*) begin
    case (opcode)
        7'b0110011: rf_we=1;//R-type
        7'b0010011: rf_we=1;//I-type
        7'b0010111: rf_we=1;//auipc
        7'b0110111: rf_we=1;//lui
        default: rf_we=1;
    endcase
end

endmodule
```

2.decoder_tb.v

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
//
// Company:
// Engineer:
//
// Create Date: 2024/04/09 15:57:32
// Design Name:
// Module Name: DECODER_tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
//

`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 DECODER_tb();
    reg [31:0]  inst;
    wire [4:0] alu_op;
    wire [31:0] imm;
    wire [4:0] rf_ra0;
    wire [4:0] rf_ra1;
    wire [4:0] rf_wa;
    wire [0:0] rf_we;
    wire [0:0] alu_src0_sel;
    wire [0:0] alu_src1_sel;
    reg  clk;

DECODER decoder_tb(
    .inst(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)
);
//rs2段为5，rs1段为15，rd段为16,imm段为28（1 1100）
initial begin
    clk = 0;

    #5
    inst=32'b0000000_00101_01111_000_10000_0110011;//add
    #5
    inst=32'b0100000_00101_01111_000_10000_0110011;//sub
    #5
    inst=32'b0000000_00101_01111_001_10000_0110011;//sll
    #5
    inst=32'b0000000_00101_01111_010_10000_0110011;//slt
    #5
    inst=32'b0000000_00101_01111_011_10000_0110011;//sltu
    #5
    inst=32'b0000000_00101_01111_100_10000_0110011;//xor
    #5
    inst=32'b0000000_00101_01111_101_10000_0110011;//srl
    #5
    inst=32'b0100000_00101_01111_101_10000_0110011;//sra
    #5
    inst=32'b0000000_00101_01111_110_10000_0110011;//or
    #5
    inst=32'b0000000_00101_01111_111_10000_0110011;//and
    #10
    inst=32'b0000000_00000_00011_100_10000_0110111;//lui
    #10
    inst=32'b0000000_11100_01111_000_10000_0010011;//addi
    #5
    inst=32'b0000000_11100_01111_001_10000_0010011;//slli
    #5
    inst=32'b0000000_11100_01111_010_10000_0010011;//slti
    #5
    inst=32'b0000000_11100_01111_011_10000_0010011;//sltiu
    #5
    inst=32'b0000000_11100_01111_100_10000_0010011;//xori
    #5
    inst=32'b0000000_11100_01111_101_10000_0010011;//srli
    #5
    inst=32'b0100000_11100_01111_101_10000_0010011;//srai
    #5
    inst=32'b0000000_11100_01111_110_10000_0010011;//ori
    #5
    inst=32'b0000000_11100_01111_111_10000_0010011;//andi
    #10
    inst=32'b0000000_00000_00011_100_10000_0010111;//auipc
    #10

    $finish;
end
always #5 clk = ~clk;

endmodule
```

3.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;

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

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(cur_npc)
);
assign imem_raddr=cur_pc;
assign cur_inst=imem_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)
);

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)
);
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  <= 0;                           // 不需要改动
            commit_dmem_wa_reg  <= 0;                           // 不需要改动
            commit_dmem_wd_reg  <= 0;                           // 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             [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             [ 0 : 0]            alu_src0_sel,
    output  reg             [ 0 : 0]            alu_src1_sel
);

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'b0010011:if (funct3==3'b101 || funct3==3'b001) begin
        imm={{27{inst[24]}},inst[24:20]};//srai
        /*
            if (funct7==7'b0100000) begin
                imm={{27{inst[24]}},inst[24:20]};//srai
            end
            else begin
                imm={27'b0,inst[24:20]};//srli,slli
            end
        */
        end
        else begin
            imm={{20{inst[31]}},inst[31:20]};
        end
        /*
        else if(funct3==3'b011 || funct3==3'b100 || funct3==3'b110 ||
funct3==3'b111) begin
            imm={20'b0,inst[31:20]};//sltiu,xori,ori,andi
        end
        else begin
            imm={{20{inst[31]}},inst[31:20]};//addi,
        end
        */
        7'b0110111: imm={inst[31:12],12'b0};//U type lui
        7'b0010111: imm={inst[31:12],12'b0};//U type auipc
        default: imm=32'b0;
    endcase
end
```

```
always @(*) begin
    if (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
    else if (opcode==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
    else if(opcode==7'b0110111)begin
        alu_op=5'B10010;
    end
    else begin
        alu_op=5'B00000;//else to add
    end
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'b0110011: alu_src0_sel=0; //R-type
        7'b0010011: alu_src0_sel=0; //I-type
        7'b0010111: alu_src0_sel=1; //auipc
        7'b0110111: alu_src0_sel=0; //lui
        default: alu_src0_sel=0;
    endcase
end

always @(*) begin
    case (opcode)
        7'b0110011: alu_src1_sel=0;//R-type
        7'b0010011: alu_src1_sel=1;//I-type
```

```
        7'b0010111: alu_src1_sel=1;//auipc
        7'b0110111: alu_src1_sel=1;//lui
        default: alu_src1_sel=0;
    endcase
end

//rf_we
always @(*) begin
    case (opcode)
        7'b0110011: rf_we=1;//R-type
        7'b0010011: rf_we=1;//I-type
        7'b0010111: rf_we=1;//auipc
        7'b0110111: rf_we=1;//lui
        default: rf_we=1;
    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;
            `SRL :
                alu_res = alu_src0 >> alu_src1;
            `SRA :
                alu_res = $signed(alu_src0) >>> $signed(alu_src1);
            `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'h0ffffffc) begin
        npc=pc+32'h4;
    end
end

endmodule
```

4.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;

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;

CPU cpu(
    .clk(clk),
    .rst(rst),
    .global_en(global_en),
    .imem_rdata(imem_rdata),
    .imem_raddr(imem_raddr),
    .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),
    .alu_res2(alu_res2),
    .alu_op2(alu_op2),
    .alu_src0_2(alu_src0_2),
    .alu_src1_2(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
);

initial begin
    clk = 0;
    global_en=1;
    #1
    rst = 1;
```

```
    #1
    rst = 0;
    we  = 0;
    d=32'b0;
    #10000
    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
```