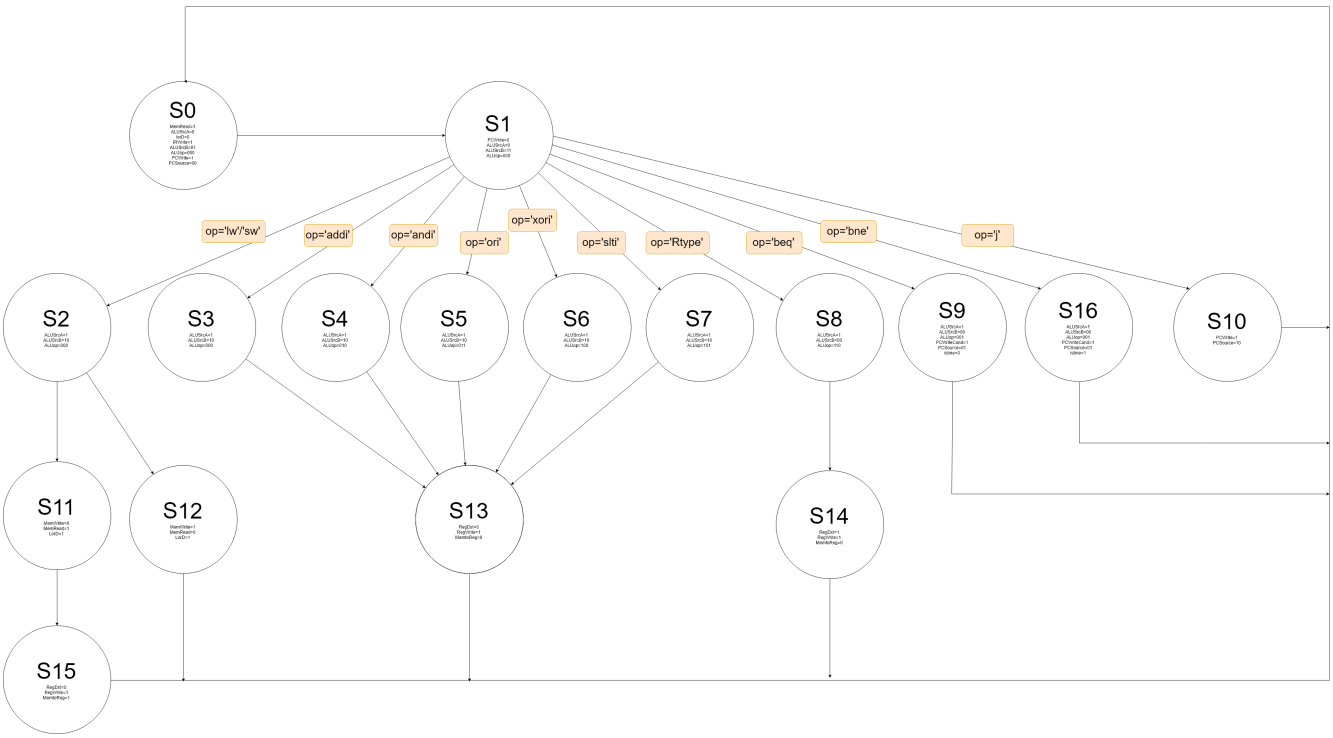
Lab5 多周期MIPS-CPU

PB17111623 范睿

逻辑设计

状态机



- 状态解释

  S0: Fetch,取指

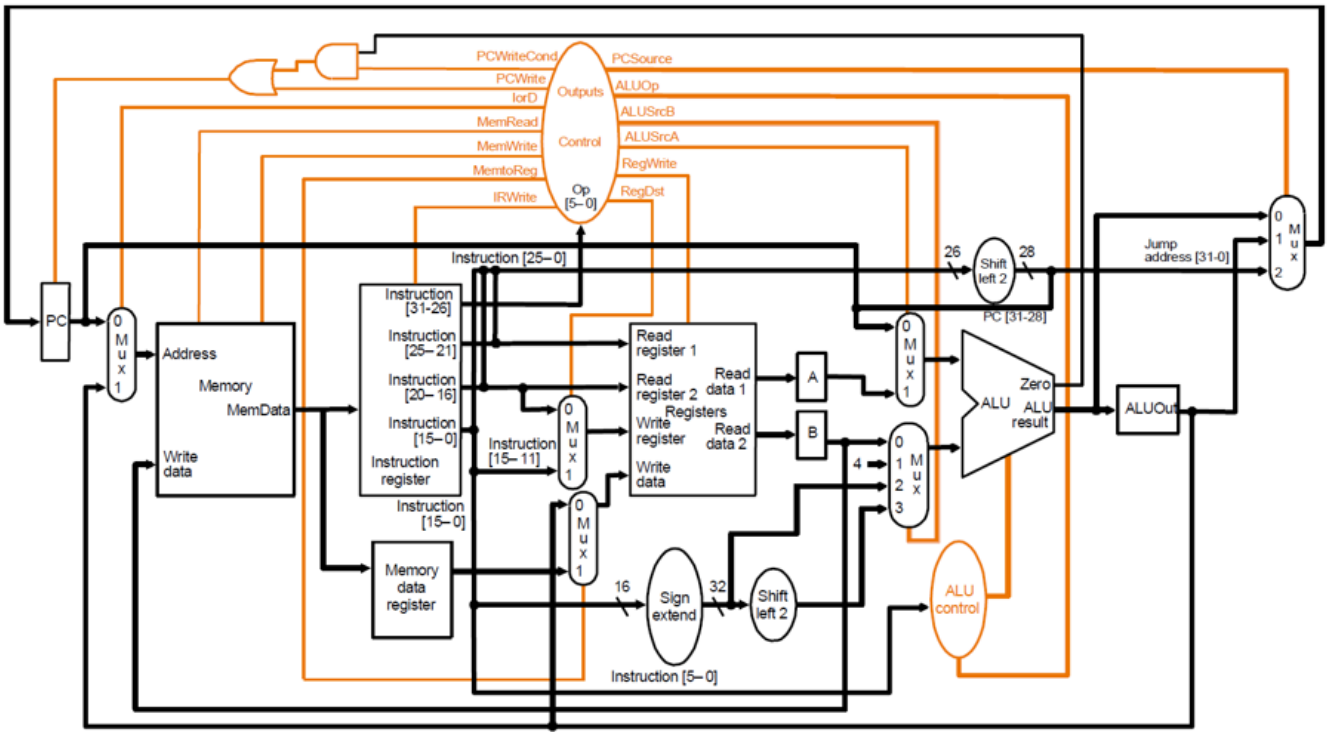
  S1: Decode,译码

  S2~S10: Excute,执行

  S11~S14: Mem,访存

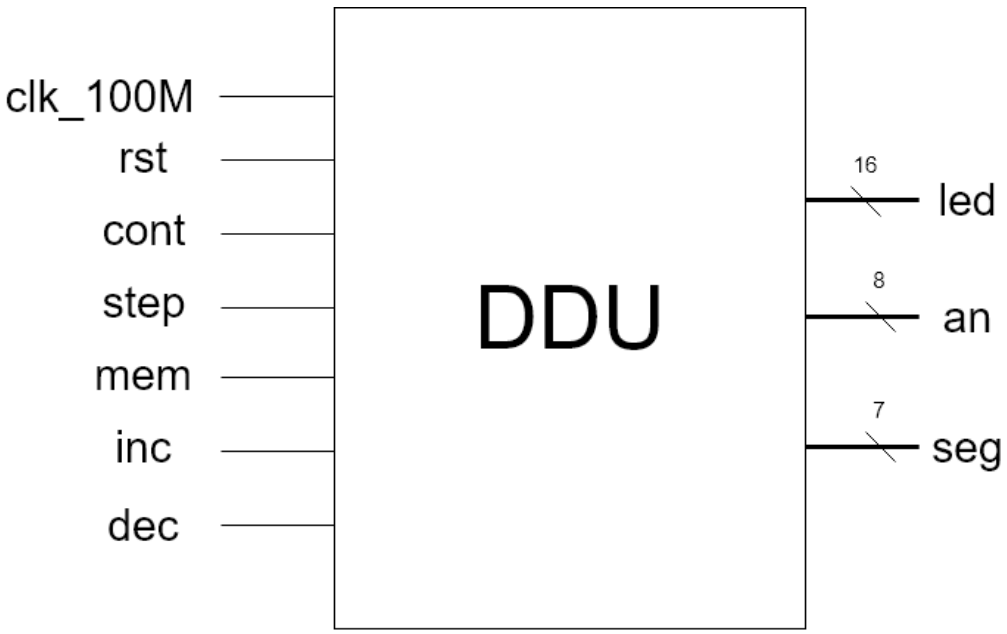
  S15: Write Back,写回

数据通路



核心代码

DDU 模块



- 端口：
  - cont：1：CPU连续执行；0：单步执行
  - step：单步执行脉冲
  - mem：1：查看存储器；0：查看寄存器
  - inc：1：查看地址+1；0：查看地址-1
- 功能：
  - 生成100HZ的时钟clk
  - 根据cont，step生成run信号，输出到cpu实例中作为cpu的时钟

- 根据inc和dec生成mem_addr和reg_addr信号，进行存储器和寄存器的读取

CPU模块

- 端口：
  - clk
  - rst
  - mem_addr_ddu：8位，读取的mem地址
  - reg_addr_ddu：5位， 读取的reg地址
  - reg_data_ddu：32位，读到的reg数据（返回给DDU）
  - mem_data_ddu：32位，读到的mem数据（返回给DDU）
  - PC：32位，PC值（返回给DDU）
- 功能：
  - 给各个module之间连线

```
module cpu(
    input clk,
    input rst,
    input [7:0]mem_addr_ddu,
    input [4:0]reg_addr_ddu,

    output [31:0]reg_data_ddu,
    output [31:0]mem_data_ddu,
    output [31:0]PC
    );
    wire [5:0]op;
    wire [31:0]A;
    wire [31:0]B;
    wire [31:0]ALUresult;
    wire [31:0]ALUout;
    wire Zero;
    wire [31:0]Jump_addr;
    wire [31:0]Instruction;

    wire PCWriteCond,PCWrite,IorD,MemRead,MemWrite,MemtoReg,IRWrite,ALUSrcA,RegWrite,RegDst,isbne;
    wire [1:0]ALUSrcB, PCSource;
    wire [2:0]ALUop;
//contrul unit生成各个信号（状态机代码也在里面）
    control_unit cu(.clk(clk),
        .rst(rst),
        .op(op),

        .PCWriteCond(PCWriteCond),
        .PCWrite(PCWrite),
        .IorD(IorD),
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .MemtoReg(MemtoReg),
        .IRWrite(IRWrite),
```

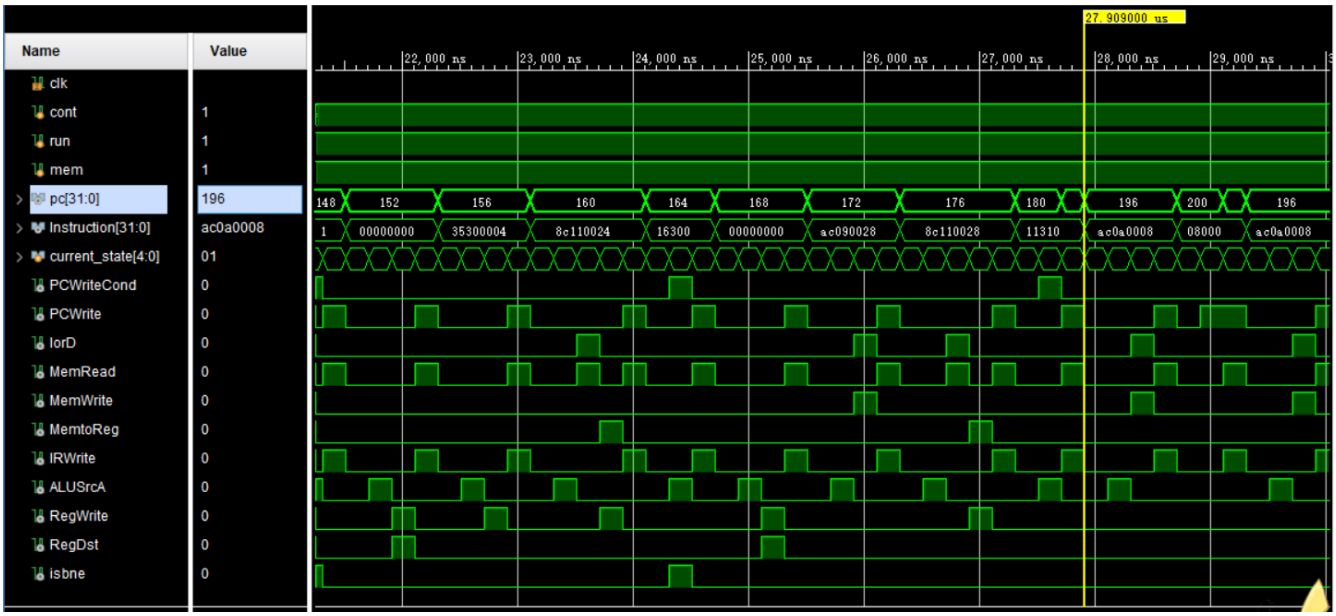
```
        .PCSource(PCSource),
        .ALUop(ALUop),
        .ALUSrcB(ALUSrcB),
        .ALUSrcA(ALUSrcA),
        .RegWrite(RegWrite),
        .RegDst(RegDst),
        .isbne(isbne)
        );

    assign Jump_addr = {PC[31:28],Instruction[25:0],2'b00};
    wire [31:0]next_pc;
//计算PC的下一个值
    next_PC next_pc_dut(
        .PCSource(PCSource),
        .ALUresult(ALUresult),
        .ALUout(ALUout),
        .Jump_addr(Jump_addr),

        .next_pc(next_pc)
        );
//给PC赋值
    wire opisbne;
    PC1 PC_dut(
            .clk(clk),
            .rst(rst),
            .PCWrite(PCWrite),
            .Zero(Zero),
            .PCWriteCond(PCWriteCond),
            .next_pc(next_pc),
            .PC(PC),
            .isbne(isbne)
            );

    wire [7:0]rw_addr;
    wire [31:0]WriteData_mem;
    assign WriteData_mem=B;

//计算读写memory的地址rw_addr
    MEM mem_dut(
        .MemRead(MemRead),
        .MemWrite(MemWrite),
        .lorD(lorD),
        .PC(PC[7:0]),
        .ALUout(ALUout[7:0]),

        .rw_addr(rw_addr)
        );

    wire [31:0]MemData;
//ip核
//mem_addr-mem_data-DDU
//rd_addr-writedata_mem-memdata
    dist_mem_gen_0 MEM_dut(
```

```
        .a(rw_addr),
        .d(WriteData_mem),
        .dpra(mem_addr_ddu),
        .clk(clk),
        .we(MemWrite),

        .spo(MemData),
        .dpo(mem_data_ddu)
        );

    wire [31:0]Memery_data_register;
    wire [4:0] ReadRegAddr_1;
    wire [4:0] ReadRegAddr_2;
    wire [15:0]instant;
//写IR与Memory Data Register
    IRnMDR IRnMDR_dut(
        .clk(clk),
        .rst(rst),
        .MemData(MemData),
        .IRWrite(IRWrite),

        .op(op),
        .Memery_data_register(Memery_data_register),
        .Instruction(Instruction),
        .ReadRegAddr_1(ReadRegAddr_1),
        .ReadRegAddr_2(ReadRegAddr_2),
        .instant(instant)
        );

    wire [31:0]WriteData_reg;
    wire [4:0]Write_reg_dst;
    wire [31:0]ReadReg_1;
    wire [31:0]ReadReg_2;
//IR和RF之间的连线
    con_IR_RF IR_RF_dut(
        .MemtoReg(MemtoReg),
        .RegDst(RegDst),
        .Memery_data_register(Memery_data_register),
        .ALUout(ALUout),
        .instant(instant[15:11]),
        .ReadRegAddr_2(ReadRegAddr_2),

        .WriteData_reg(WriteData_reg),
        .Write_reg_dst(Write_reg_dst)
        );
//RF寄存器堆
    RF RF_dut(
        .ra0(ReadRegAddr_1),//直接从IR连过来
        .ra1(ReadRegAddr_2),
        .we(RegWrite),
        .wa(Write_reg_dst),
        .wd(WriteData_reg),
        .clk(clk),
```

```
        .rst(rst),
        .ddu_out(reg_data_ddu),

        .rd0(ReadReg_1),
        .rd1(ReadReg_2),
        .ddu_rd(reg_addr_ddu)
        );
//写数据通路中的AB
    RFout_AB AB_dut(.clk(clk),
            .rst(rst),
            .ReadReg_1(ReadReg_1),
            .ReadReg_2(ReadReg_2),

            .A(A),
            .B(B)
            );

    wire [31:0]ALU_a;
    wire [31:0]ALU_b;

    wire [2:0]ALUoperation;
    wire [2:0]flag;
//根据ALUop计算ALUoperation，决定ALU做什么操作
    ALU_control ALU_control_dut(
        .ALUop(ALUop),
        .func(instant[5:0]),

        .ALUoperation(ALUoperation)
        );
//计算ALU两操作数
    ALU_Operands ALU_AnB_dut(
        .instant(instant),
        .PC(PC),
        .A(A),
        .B(B),
        .ALUSrcA(ALUSrcA),
        .ALUSrcB(ALUSrcB),

        .ALU_a(ALU_a),
        .ALU_b(ALU_b)
        );
//调用ALU进行运算
    ALU alu_dut(.a(ALU_a),.b(ALU_b),.s(ALUoperation),.y(ALUresult),.f(flag));
    assign Zero = flag[0];
//ALU输出
    ALU_out ALU_out_dut(
        .clk(clk),
        .rst(rst),
        .ALUresult(ALUresult),

        .ALUout(ALUout)
        );

```

```
194 endmodule
```

仿真



下载

此下载结果为运行结束之后mem中0x02地址中的值

实验总结

在本次试验中，我实现了一个mips指令集的cpu。其中遇到了很多bug。

- always敏感变量中若用到了多个信号，比如posedge inc or posedge rst，但在块中没有用到inc，会报错ambiguous clock。
- 写到后面发现ALU中没有nor操作
- 计算rw_addr时，需要当MemRead或MemWrite其一有效就对它赋值。我开始只令MemRead有效时对它赋值，sw就出了问题。
- 本来我没有将bne与beq分成两个状态，这样就使bne失效了，实际上bne是zero等于0的时候跳转，beq是zero等于1的时候跳转，两者还是有区别的
- …

总之，本次实验中我愈发体会到把各个功能分到各个模块去做是多么重要。否则debug效率极低。

代码

DDU

```
module DDU(
    input clk_100M,
```

```
    input rst,
    input cont,
    input step,
    input mem,
    input inc,
    input dec,

    output reg [15:0]led,
    output [7:0]an,
    output [6:0]seg
    );
    wire locked;
    wire clk;//1hz
    wire [31:0]pc;
    wire [31:0]mem_data;
    wire [31:0]reg_data;
    wire [7:0]mem_addr;
    wire [4:0]reg_addr;
    wire [31:0]pc_temp = {2'B00,pc[31:2]};
    reg [7:0]mem_addr_dec;
    reg [7:0]mem_addr_inc;
    reg [4:0]reg_addr_dec;
    reg [4:0]reg_addr_inc;
    reg run;
    reg [31:0]cnt;

    always@(posedge clk_100M or posedge rst)begin
        if(rst) cnt<=32'b0;
        else begin
            if(cnt == (32'd1000000-32'd1))cnt<=0;
            else cnt <= cnt + 1;
        end
    end

    assign clk = (cnt<32'd500000) ? 0 : 1;

    always@(*)begin//run
        if(cont)begin
            run = clk;
        end
        else run = step;
    end
    always@(posedge clk or posedge rst)begin//led
        if(rst)begin
            led <= {8'b0,pc[7:0]};
        end
        else begin
            led <= mem ? {mem_addr,pc[7:0]} : {reg_addr,3'b0,pc[7:0]};
        end
    end

    always@(posedge inc or posedge rst)begin//men_addr & reg_addr
        if(rst)begin
```

```
            mem_addr_inc<=0;
            reg_addr_inc<=0;
        end
        else begin
            mem_addr_inc <= mem_addr_inc+1;
            reg_addr_inc <= reg_addr_inc+1;
        end
    end
    always@(posedge dec or posedge rst)begin//men_addr & reg_addr
        if(rst)begin
            mem_addr_dec<=0;
            reg_addr_dec<=0;
        end
        else begin
            mem_addr_dec <= mem_addr_dec+1;
            reg_addr_dec <= reg_addr_dec+1;
        end
    end

    assign mem_addr = mem_addr_inc-mem_addr_dec;
    assign reg_addr = reg_addr_inc-reg_addr_dec;

    seg seg_dut(.clk(clk),.rst(rst),.mem_data(mem_data),.reg_data(reg_data),.mem(mem),.an(an),.seg(seg));
    cpu cpu_dut(.clk(run),.rst(rst),.mem_addr_ddu(mem_addr),.reg_addr_ddu(reg_addr),.reg_data_ddu(reg_data),.mem_data_ddu(mem_data),.PC(pc));
endmodule

```

数码管显示模块

```
module seg(
    input clk,
    input rst,
    input [31:0]mem_data,
    input [31:0]reg_data,
    input mem,

    output reg [7:0]an,
    output reg [6:0]seg
    );

    reg [2:0] count;    //即将显示的位
    always@(posedge clk)
    begin
        count <= count + 1'b1;
    end

     always@(count)
    begin

```

```
        case(count)
        0:begin
            an = 8'b1111_1110;
        end
        1:begin
            an = 8'b1111_1101;
        end
        2:begin
            an = 8'b1111_1011;
        end
        3:begin
            an = 8'b1111_0111;
        end
        4:begin
            an = 8'b1110_1111;
        end
        5:begin
            an = 8'b1101_1111;
        end
        6:begin
            an = 8'b1011_1111;
        end
        7:begin
            an = 8'b0111_1111;
        end
        default: an = 8'b0000_0000;
        endcase
    end

    wire [31:0]display_data = mem ? mem_data : reg_data;
    reg [3:0]seg_temp;

    always@(*)begin
        case(an)
            8'b11111110:seg_temp=display_data[3:0];
            8'b11111101:seg_temp=display_data[7:4];
            8'b11111011:seg_temp=display_data[11:8];
            8'b11110111:seg_temp=display_data[15:12];
            8'b11101111:seg_temp=display_data[19:16];
            8'b11011111:seg_temp=display_data[23:20];
            8'b10111111:seg_temp=display_data[27:24];
            8'b01111111:seg_temp=display_data[31:28];
            default seg_temp = 4'd0;
       endcase
    end

    always@(*)begin
        case(seg_temp)
            4'd0:seg = 7'b1000000;
            4'd1:seg = 7'b1111001;
            4'd2:seg = 7'b0100100;
            4'd3:seg = 7'b0110000;
            4'd4:seg = 7'b0011001;
```

```
            4'd5:seg = 7'b0010010;
            4'd6:seg = 7'b0000010;
            4'd7:seg = 7'b1111000;
            4'd8:seg = 7'b0000000;
            4'd9:seg = 7'b0010000;
            4'd10:seg = 7'b0001000;
            4'd11:seg = 7'b0000011;
            4'd12:seg = 7'b1000110;
            4'd13:seg = 7'b0100001;
            4'd14:seg = 7'b0000110;
            4'd15:seg = 7'b0001110;
            default:seg = 7'b0000000;
        endcase
    end
endmodule
```

Control Unit

```
module control_unit(
    input clk,
    input rst,
    input [5:0]op,

    output reg PCWriteCond,
    output reg PCWrite,
    output reg IorD,
    output reg MemRead,
    output reg MemWrite,
    output reg MemtoReg,
    output reg IRWrite,
    output reg [1:0]PCSource,
    output reg [2:0]ALUop,
    output reg [1:0]ALUSrcB,
    output reg ALUSrcA,
    output reg RegWrite,
    output reg RegDst,
    output reg isbne
    );
    parameter Rtype = 6'b000000;
    parameter addi = 6'b001000;
    parameter andi = 6'b001100;
    parameter ori = 6'b001101;
    parameter xori = 6'b001110;
    parameter slti = 6'b001010;
    parameter lw = 6'b100011;
    parameter sw = 6'b101011;
    parameter beq = 6'b000100;
    parameter bne = 6'b000101;
    parameter j = 6'b000010;

    parameter S0 = 5'b0000;
    parameter S1 = 5'b0001;
```

```
    parameter S2 = 5'b0010;
    parameter S3 = 5'b0011;
    parameter S4 = 5'b0100;
    parameter S5 = 5'b0101;
    parameter S6 = 5'b0110;
    parameter S7 = 5'b0111;
    parameter S8 = 5'b1000;
    parameter S9 = 5'b1001;
    parameter S10 =5'b1010;
    parameter S11 = 5'b1011;
    parameter S12 = 5'b1100;
    parameter S13 = 5'b1101;
    parameter S14 = 5'b1110;
    parameter S15 = 5'b1111;
    parameter S16 = 5'b10000;

    reg [4:0]current_state;
    reg [4:0]next_state;

    always@(posedge clk or posedge rst)begin
        if(rst)current_state <= S0;
        else current_state<=next_state;
    end

    always@(*)begin//next_state
        case(current_state)
            S0:next_state=S1;
            S1:begin
                if(op == lw || op == sw)next_state=S2;
                else if(op == addi)next_state=S3;
                else if(op == andi)next_state=S4;
                else if(op == ori)next_state=S5;
                else if(op == xori)next_state=S6;
                else if(op == slti)next_state=S7;
                else if(op == Rtype)next_state=S8;
                else if(op == beq)next_state=S9;
                else if(op == j)next_state=S10;
                else if(op == bne)next_state=S16;
                else next_state=S0;
            end
            S2:begin if(op==lw)next_state=S11;else next_state=S12;end
            S3,S4,S5,S6,S7:next_state=S13;
            S8:next_state=S14;
            S9,S10,S12,S13,S14,S15,S16:next_state=S0;
            S11:next_state=S15;
        endcase
    end

    always@(op or current_state)begin//ALUop
        case(current_state)
            S0,S1,S2,S3:ALUop=3'b000;
            S4:ALUop=3'b010;//Rtype
            S5:ALUop=3'b011;//bne/beq
```

```
            S6:ALUop=3'b100;
            S7:ALUop=3'b101;
            S8:ALUop=3'b110;
            S9,S16:ALUop=3'b001;
            default:ALUop=3'b111;
        endcase
    end
    always@(op or current_state)begin//ALUSrcA&ALUSrcB
        case(current_state)
            S0:begin ALUSrcA=0;ALUSrcB=2'b01;end
            S1:begin ALUSrcA=0;ALUSrcB=2'b11;end
            S2,S3,S4,S5,S6,S7:begin ALUSrcA=1;ALUSrcB=2'b10;end
            S8,S9,S16:begin ALUSrcA=1;ALUSrcB=2'b00;end
            default:begin ALUSrcA=0;ALUSrcB=2'b01;end
        endcase
    end
    always@(current_state)begin//RegWrite&RegDst
        case(current_state)
            S13:begin RegWrite<=1;RegDst<=0;end
            S14:begin RegWrite<=1;RegDst<=1;end
            S15:begin RegWrite<=1;RegDst<=0;end
            default:begin RegWrite<=0;RegDst<=0;end
        endcase
    end
    always@(current_state)begin//PCSource&PCWriteCond
        case(current_state)
            S0:begin PCSource=2'b00;PCWriteCond=0;end
            S9,S16:begin PCSource=2'b01;PCWriteCond=1;end
            S10:begin PCSource=2'b10;PCWriteCond=0;end
            default:begin PCSource=2'b00;PCWriteCond=0;end
        endcase
    end
    always@(current_state)begin//PCWrite
        case(current_state)
            S0,S10:PCWrite=1;
            default:PCWrite=0;
        endcase
    end
    always@(current_state)begin//IorD
        case(current_state)
            S11,S12:IorD=1;
            default:IorD=0;
        endcase
    end
    always@(current_state)begin//MemRead
        case(current_state)
            S0,S11:MemRead=1;
            default:MemRead=0;
        endcase
    end
    always@(current_state)begin//MemRead
        case(current_state)
            S12:MemWrite=1;
```

```
            default:MemWrite=0;
        endcase
    end
    always@(current_state)begin//MemtoReg
        case(current_state)
            S15:MemtoReg=1;
            default:MemtoReg=0;
        endcase
    end
    always@(current_state)begin//IRWrite
        case(current_state)
            S0:IRWrite=1;
            default:IRWrite=0;
        endcase
    end

    always@(current_state)begin
        case(current_state)
            S16:isbne=1;
            default:isbne=0;
    endcase
    end

endmodule
```

计算下一个PC值

```
module next_PC(
    input [1:0]PCSource,
    input [31:0]ALUresult,
    input [31:0]ALUout,
    input [31:0]Jump_addr,
    output reg [31:0]next_pc
    );
    always@(*)begin
        case(PCSource)
            0:next_pc=ALUresult;
            1:next_pc=ALUout;
            2:next_pc=Jump_addr;
       endcase
    end
endmodule
```

更新PC

```
module PC1(
    input clk,
    input rst,
    input PCWrite,
    input Zero,
    input PCWriteCond,
```

```
    input [31:0]next_pc,
    input isbne,
    output reg [31:0]PC
    );
   wire isbne_cond = (~Zero && PCWriteCond);
   wire notbne_cond = (Zero && PCWriteCond);
   always@(posedge clk or posedge rst)begin
         if(rst)PC<=0;
         else begin
            if(PCWrite) PC = next_pc;
            else begin
                if((isbne&&isbne_cond)||(~isbne && notbne_cond)) PC = next_pc;
            end
         end
     end

endmodule
```

计算读写Memory的地址rw_addr

```
module MEM(
    input MemRead,
    input MemWrite,
    input lorD,
    input [7:0]PC,
    input [7:0]ALUout,
    output [7:0]rw_addr
    );

    assign rw_addr = (MemRead || MemWrite) ? (lorD ? {2'b0,ALUout[7:2]} :
{2'b0,PC[7:2]}) : 8'b0;

endmodule
```

写IR和MDR

```
module IRnMDR(
    input clk,
    input rst,
    input [31:0]MemData,
    input IRWrite,
    output reg [31:0]Memery_data_register,
    output reg [31:0]Instruction,
    output [5:0]op,
    output [4:0] ReadRegAddr_1,
    output [4:0] ReadRegAddr_2,
    output [15:0]instant
    );

    always@(posedge clk or posedge rst)begin
        if(rst)Memery_data_register<=0;
```

```
        else Memery_data_register<=MemData;
    end

    always@(posedge clk or posedge rst)begin
        if(rst) Instruction=0;
        else if(IRWrite) Instruction=MemData;
    end

    assign op = Instruction[31:26];
    assign ReadRegAddr_1 = Instruction[25:21];
    assign ReadRegAddr_2 = Instruction[20:16];
    assign instant = Instruction[15:0];

endmodule
```

IR和RF的连线

```
module con_IR_RF(
    input MemtoReg,
    input RegDst,
    input [31:0]Memery_data_register,
    input [31:0]ALUout,
    input [4:0]instant,
    input [4:0]ReadRegAddr_2,

    output [31:0]WriteData_reg,
    output [4:0]Write_reg_dst
    );

    assign WriteData_reg = MemtoReg ? Memery_data_register : ALUout;
    assign Write_reg_dst = RegDst ? instant :ReadRegAddr_2;

endmodule
```

RF

```
module RF #(parameter reg_num=5,width=32) (
    input [reg_num-1:0] ra0,//读的地址
    input [reg_num-1:0] ra1,//读的地址
    input [reg_num-1:0] wa,//写的地址
    input [width-1:0] wd,//写的数据
    input we,
    input clk,
    input [reg_num-1:0] ddu_rd,
    input rst,
    output [width-1:0] ddu_out,
    output [width-1:0] rd0,//被读的数据
    output [width-1:0] rd1//被读的数据
    );
    reg [reg_num:0] RF [0:width-1];
    always@(posedge clk or posedge rst)
```

```
    begin
        if(rst)
            begin
            RF[0]<=0;RF[1] <=0;RF[2] <=0;RF[3] <=0;
            RF[4] <=0;RF[5] <=0;RF[6] <=0;RF[7] <=0;
            RF[8] <=0;RF[9] <=0;RF[10] <=0;RF[11] <=0;
            RF[12] <=0;RF[13] <=0;RF[14] <=0;RF[15] <=0;
            RF[16] <=0;RF[17] <=0;RF[18] <=0;RF[19] <=0;
            RF[20] <=0;RF[21] <=0;RF[22] <=0;RF[23] <=0;
            RF[24] <=0;RF[25] <=0;RF[26] <=0;RF[27] <=0;
            RF[28] <=0;RF[29] <=0;RF[30] <=0;RF[31] <=0;
            end
        else if(we)
            RF[wa] <= wd;
    end

    assign rd0=RF[ra0];
    assign rd1=RF[ra1];
    assign ddu_out=RF[ddu_rd];

endmodule
```

写AB寄存器

```
module RFout_AB(
    input clk,
    input rst,
    input [31:0]ReadReg_1,
    input [31:0]ReadReg_2,
    output reg [31:0]A,
    output reg [31:0]B
    );
    always@(posedge clk or posedge rst)begin
        if(rst)begin A<=0;B<=0;end
        else begin
            A<=ReadReg_1;
            B<=ReadReg_2;
        end
    end
endmodule
```

ALU control

```
module ALU_control(
    input [2:0]ALUop,
    input [5:0]func,
    output reg [2:0]ALUoperation
    );

    always@(*)begin
        case(ALUop)
```

```
 3'b000,3'b001,3'b010,3'b011,3'b100,3'b101:ALUoperation=ALUop;//+,-,and,or,xor,slt
            3'b110:begin
                case(func)
                    6'b100000:ALUoperation=3'b000;//+
                    6'b100010:ALUoperation=3'b001;//-
                    6'b100100:ALUoperation=3'b010;//and
                    6'b100101:ALUoperation=3'b011;//or
                    6'b100110:ALUoperation=3'b100;//xor
                    6'b101010:ALUoperation=3'b101;//slt
                    6'b100111:ALUoperation=3'b110;//nor
                    default:;
                endcase
            end
            3'b111:;
        endcase
    end

endmodule
```

ALU操作数

```
module ALU_Operands(
    input [15:0]instant,
    input [31:0]PC,
    input [31:0]A,
    input [31:0]B,
    input ALUSrcA,
    input [1:0]ALUSrcB,
    output  reg [31:0]ALU_a,
    output reg [31:0]ALU_b
    );

    wire [31:0]sign_extend = instant[15] ? {16'b1111111111111111, instant} : {16'b0,
instant};
    wire [31:0]left_shift = {sign_extend[29:0] ,2'b00};

    always@(*)begin
        case(ALUSrcA)
            0:ALU_a=PC;
            1:ALU_a=A;
        endcase
        case(ALUSrcB)
            0:ALU_b=B;
            1:ALU_b=32'd4;
            2:ALU_b=sign_extend;
            3:ALU_b=left_shift;
        endcase
    end

endmodule
```

ALU

```
module ALU(
    input [31:0]a,
    input [31:0]b,
    input [2:0]s,
    output reg [31:0]y,
    output reg [2:0]f//[进位/借位，溢出，零标志]
    );
    initial begin
        f = 3'b000;
    end
    reg [32:0]y1;
    reg [31:0]_y;
    reg [31:0]b1;
    reg c;
    always@(*)begin
        f = 3'b000;
        case(s)
            3'b000://add
                begin
                y1 = a + b;
                //判断进位
                   {f[2], y} = y1;
                //判断溢出
                   if(a[31]==0 && b[31]==0)begin //ab均正
                        if(y[31]==1) f[1] = 1;//溢出
                        else f[1] = 0;
                   end
                   else if(a[31]==1 && b[31]==1) //ab均负
                        if(y[31]==0) f[1] = 1;
                        else f[1]=0;
                 end
            3'b001://sub
                begin
                   b1 = ~b + 1;//b1=-b
                   y1 = a + ~b + 6'b000001;
                   {f[2], y} = y1;//判断借位
                   //判断溢出
                   if(a[31]==0 && b1[31]==0)begin //a,-b均正
                        if(y[31]==1) f[1] = 1;//溢出
                        else f[1] = 0;
                   end
                   else if(a[31]==1 && b1[31]==1) //a,-b均负
                        if(y[31]==0) f[1] = 1;
                        else f[1]=0;
                   //由于100000求反+1后仍为100000，因此单独考虑
                   if(b==32'b10000000000000000000000000000000)begin
                        if(a[31]==0) f[1]=1;
                        else f[1]=0;
                   end
                end
            3'b010://and
```

```
                begin
                    y <= a&b;
                end
            3'b011://or
                begin
                    y <= a|b;
                end
           /* 4'b0101://not
                begin
                    y <= ~a;
                end*/
            3'b100://xor
                begin
                    y <= a^b;
                end
            3'b110://nor
                begin
                    y <= ~(a|b);
                end
            3'b101://slt
                begin
                    if(a<b)y<=1;
                    else y<=0;
                end
            default:;
        endcase
        f[0] = (&(~y));//zero flag
    end
endmodule
```

ALU输出

```
module ALU_out(
    inout clk,
    input rst,
    input [31:0]ALUresult,
    output reg [31:0]ALUout
    );

    always@(posedge clk or posedge rst)begin
        if(rst)ALUout<=0;
        else ALUout<=ALUresult;
    end

endmodule
```