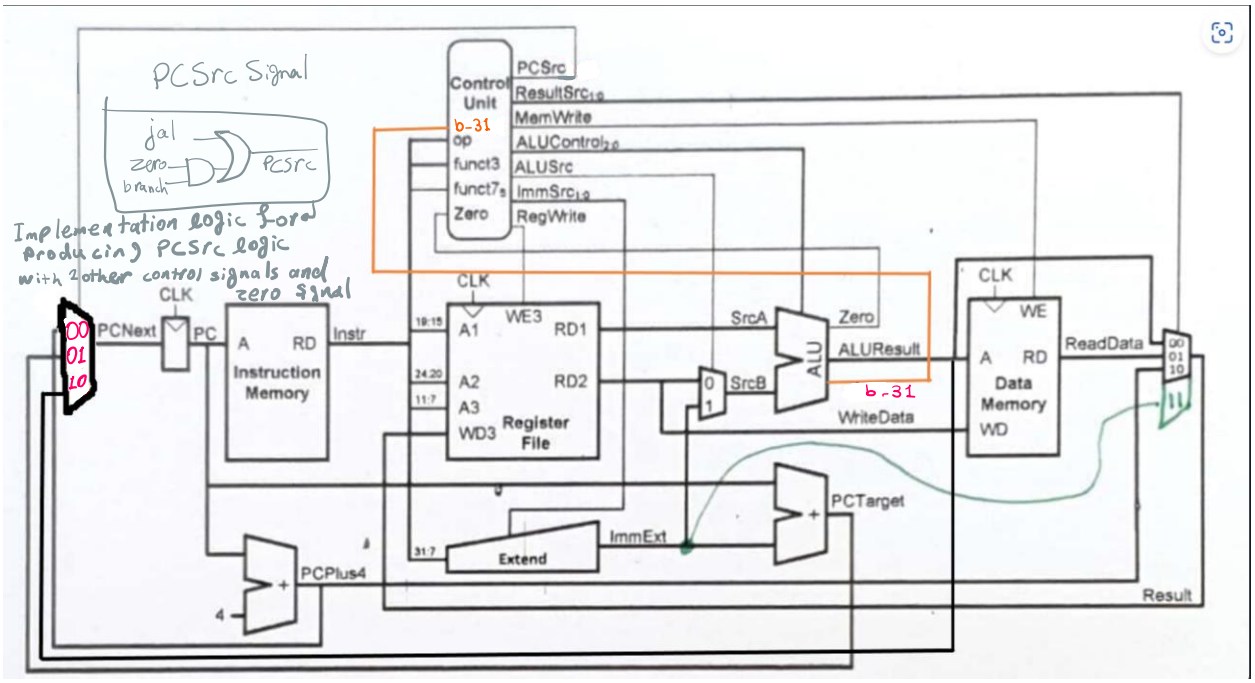
به نام خدا

تمرین کامپیوتری دوم درس سیستمهای دیجیتال ۲

فرناز صداقتی ۸۱۰۱۰۰۱۷۷ محمد مهدی دوست محمدی ۸۱۰۱۰۰۱۴۲

طراحی پردازنده ی RISC-V

مسیر داده :



سیگنال های کنترلی و کنترلرهای پردازنده:

Inst/Sig	PCsrc	Resultsrc	Mem Write	ALU controll	ALUOp	ALUSrc	ImmSrc	Regwrite	Branch	jalr	jal
R-Type	00	00	0	!?	10	0	xxx	1	0	0	0
I-Type	00	00	0	!?	11	1	000	1	0	0	0
LW	00	01	0	000(+)	00	1	000	1	0	0	0
SW	00	xx	1	000(+)	00	1	001	0	0	0	0
Jalr	10	10	0	000(+)	00	1	000	1	0	1	0
Jal	01	10	0	xxx	xx	x	011	1	0	0	1
B-Type	01	xx	0	001(-)	01	0	010	0	1	0	0
U-Tupe	00	11	0	xxx	xx	x	100	1	0	0	0

Top module controller:

```
module risc_v_controller(input [6:0] op, input [2:0] func3, input func7, input zero, b31,
output [1:0] pc_src,output [1:0] result_src, output mem_write, output [2:0] alu_control,output alu_src, output [2:0] imm_src,  output reg_write);


    wire [1:0] alu_op;
    wire branch, branch_res, jal, jalr;


    Alu_controller ac(func3, alu_op,func7,branch, zero, b31, alu_control, branch_res);

    Main_controller mc(op, zero, b31, result_src, mem_write, alu_op,alu_src, imm_src, reg_write, jal, jalr, branch);

    assign pc_src =  (jalr) ? 2'b10 : (jal | branch_res) ? 2'b01: 2'b00;

endmodule

```

ALU controller:

```
module Alu_controller(input [2:0] func3, input [1:0] alu_op, input func7,input branch, input zero, input b31, output reg [2:0] alu_control, output reg  branch_res);
parameter [1:0] R_T = 2'b10, I_T = 2'b11, S_T = 2'b00, B_T = 2'b01;
parameter [2:0] ADD = 3'b000, SUB = 3'b001, AND = 3'b010, OR = 3'b011, XOR =3'b100, SLT = 3'b101;
   always @(alu_op or func3 or func7)begin
        case (alu_op)
             R_T   : alu_control <= (func3 == 3'b000 & ~func7 ) ? ADD:
                                    (func3 == 3'b000 & func7 ) ? SUB:
                                    (func3 == 3'b111) ? AND:
                                    (func3 == 3'b110) ? OR:
                                    (func3 == 3'b010) ? SLT : 3'bxxx;
             I_T   : alu_control <= (func3 == 3'b000) ? ADD:
                                    (func3 == 3'b110) ? OR:
                                    (func3 == 3'b100) ? XOR:
                                    (func3 == 3'b010) ? SLT: 3'bxxx;
             S_T   : alu_control <= ADD;
             B_T   : alu_control <= SUB;
             default: alu_control <= ADD;

        endcase
    end

   always @(func3,branch, zero, b31) begin
        case(func3)
             3'b000   : branch_res <= branch & zero;
             3'b001   : branch_res <= branch & ~zero;
             3'b011   : branch_res <= branch & (~b31|zero);
             3'b010   : branch_res <= branch & b31;
             default: branch_res <= 1'b0;
        endcase
    end
endmodule


```

Risc-V top module:

```
module top_risc(input clk, rst);

    wire reg_write, mem_write, alu_src;
    wire [1:0] result_src, pc_src;
    wire [2:0] alu_control, imm_src, func3;
    wire [6:0] op;
    wire zero, b31, func7;

    risc_v_controller cu(op, func3, func7,zero, b31,
                                      pc_src,result_src, mem_write, alu_control,
                                      alu_src,imm_src,  reg_write);

    datapath dp(clk, rst, reg_write,alu_control,
                        mem_write,result_src,alu_src,
                        imm_src,pc_src,op,func3,
                        func7,  zero, b31);

endmodule

```

Main controller:

```
module Main_controller(input [6:0] op, input zero, input b31 , output reg [1:0] result_src, output reg mem_write,
                       output reg [1:0] alu_op, output reg alu_src, output reg [2:0] imm_src, output reg reg_write,
                       output reg jal, output reg jalr,output reg  branch);
parameter [6:0] R_T = 7'b0110011, I_T = 7'b0010011, S_T = 7'b0100011, B_T = 7'b1100011, U_T = 7'b0110111, J_T = 7'b1101111, LW = 7'b0000011, JALR = 7'b1100111;

    always @(op) begin
        {mem_write, reg_write, alu_src, jal, jalr, branch, imm_src, result_src, alu_op} <= 13'b0;
        case(op)
             R_T:begin
                reg_write <= 1'b1;
                imm_src <= 3'b000;
                alu_src <= 1'b0;
                mem_write <= 1'b0;
                result_src <= 2'b00;
                branch <= 1'b0;
                alu_op <= 2'b10;
                jal <= 1'b0;
                jalr <= 1'b0;
            end

             I_T:begin
                reg_write <= 1'b1;
                imm_src <= 3'b000;
                alu_src <= 1'b1;
                mem_write <= 1'b0;
                result_src <= 2'b00;
                branch <= 1'b0;
                alu_op <= 2'b11;
                jal <= 1'b0;
                jalr <= 1'b0;
            end
            S_T:begin
                reg_write <= 1'b0;
                imm_src <= 3'b001;
                alu_src <= 1'b1;
                mem_write <= 1'b1;
                result_src <= 2'b00;
                branch <= 1'b0;
                alu_op <= 2'b00;
                jal <= 1'b0;
                jalr <= 1'b0;
            end
            B_T:begin
                reg_write <= 1'b0;
                imm_src <= 3'b010;
                alu_src <= 1'b0;
                mem_write <= 1'b0;
                result_src <= 2'b00;
                branch <= 1'b1;
                alu_op <= 2'b01;
                jal <= 1'b0;
                jalr <= 1'b0;
            end
            U_T:begin
                reg_write <= 1'b1;
                imm_src <= 3'b100;
                alu_src <= 1'b0;
                mem_write <= 1'b0;
                result_src <= 2'b11;
                branch <= 1'b0;
                alu_op <= 2'b00;
                jal <= 1'b0;
                jalr <= 1'b0;
            end
            J_T:begin
                reg_write <= 1'b1;
                imm_src <= 3'b011;
                alu_src <= 1'b0;
                mem_write <= 1'b0;
                result_src <= 2'b10;
                branch <= 1'b0;
                alu_op <= 2'b00;
                jal <= 1'b1;
                jalr <= 1'b0;
            end

             LW:begin
                reg_write <= 1'b1;
                imm_src <= 3'b000;
                alu_src <= 1'b1;
                mem_write <= 1'b0;
                result_src <= 2'b01;
                branch <= 1'b0;
                alu_op <= 2'b00;
                jal <= 1'b0;
                jalr <= 1'b0;
            end

            JALR:begin
                reg_write <= 1'b1;
                imm_src <= 3'b000;
                alu_src <= 1'b1;
                 alu_src <= 1'b1;
                 mem_write <= 1'b0;
                 result_src <= 2'b10;
                 branch <= 1'b0;
                 alu_op <= 2'b00;
                 jal <= 1'b0;
                 jalr <= 1'b1;
             end
        endcase
    end
endmodule
```

یافتن بزرگترین عضو یک آرایه ی ۱۰ تایی از اعداد صحیح:

پردازنده ی RISC-Vطراحی شده عملیات یافتن بزرگترین عدد در یک آرایه ۱۰ تایی از اعداد صحیح را بصورت مجموعه دستورات زیر انجام میدهد:

```
jal x0, maximum
maximum:
   lw   s0, 0(zero)
   add  s1, zero, zero
   Loop:
      addi s1, s1, 4
      slti t2, s1, 40
      beq  t2, zero, EndLoop
      lw   s3, 0(s1)
      slt  t2, s0, s3
      beq  t2, zero, Loop
      add  s0, s3, zero
      jal  zero,Loop
   EndLoop:
      sw s0, 1000(zero)
      jal zero,End
End:
```

طی این دستورات مقدار اولین خانه از آرایه ی ما از حافظه داده ، در رجیستر X8 رجیستر فایل ذخیره میشود. سپس مقادیر ذخیره شده در دیگر اندیس های آرایه ،از حافظه ی داده طی رخداد یک حلقه ، در خانه ی X19 رجیستر فایل لود میشوند و هر بار با محتوای رجیستر X8 مقایسه میشوند و در صورتی که مقدار X19 بزرگتر از مقدار X8 باشد ، مقدار X19 در X8 ریخته میشود و حلقه تکرار میشود و محتوای اندیس بعدی آرایه در X19 ریخته میشود و عملیات مقایسه تکرار میشود تا در آخر بزرگترین مقدار حاضر در آرایه که در رجیستر X8 قرار دارد در خانه ۱۰۰۰ حافظه استور میشود.

(در دستورات اسمبلی که در صفحه قبل آورده شدند ، S0 همان رجیستر X8 و S3 همان رجیستر X19 را بیان میکند.)

در ویوفرم زیر عملیات توضیح داده شده قابل مشاهده است:

در پایین تصویر زیر مقادیر موجود در ده خانه ی آرایه را نشان میدهند که بصورت یک فایل متنی به برنامه ورودی داده شده اند و باید بزرگترین مقدار میان آنها (در این مورد عدد ۱۳۱) را بیابیم .همچنین تغییرات مقادیر رجیستر های X8 , X19 و خانه ۱۰۰۰ حافظه گویای عملیات یاد شده هستند: