به نام خد ا

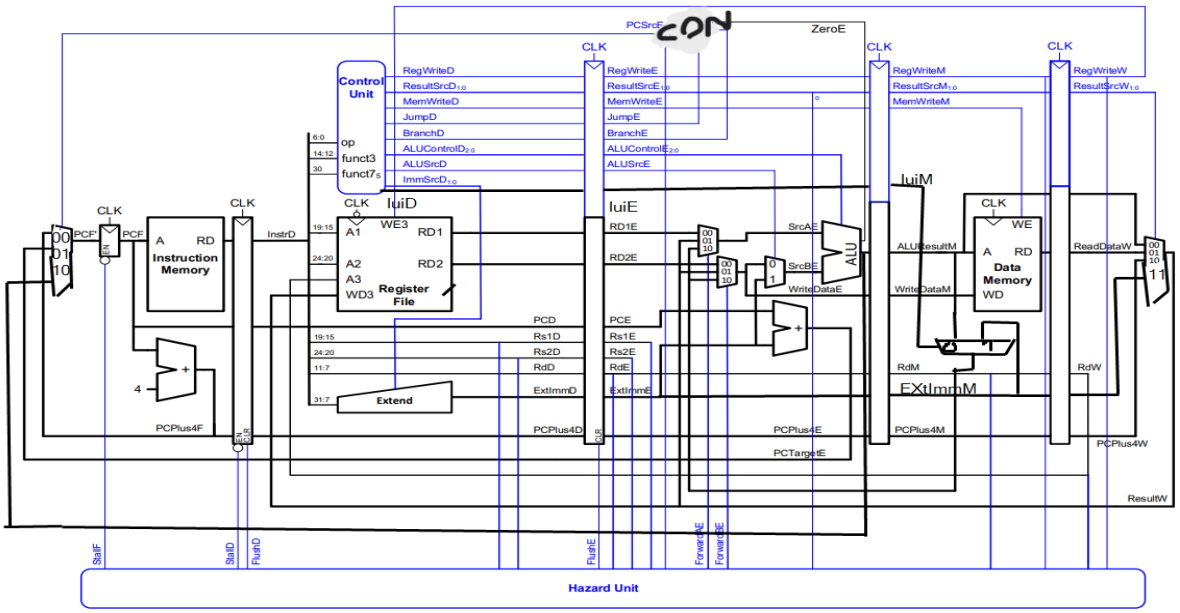
تمرین کامپیو تری چهارم درس سیستمهای دیجیتال 2

فرناز صداقتی 810100177

محمد مهدی دوست محمدی 810100142

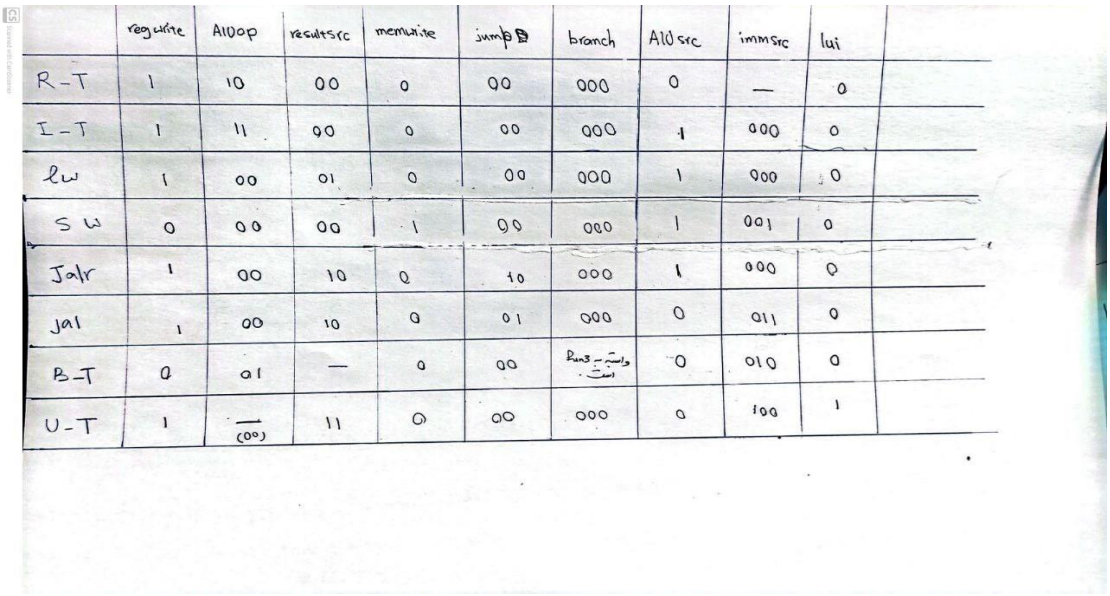
طراحی پردازنده ی RISC-V

مسیر داده :



27

سیگنال های کنترلی و کنترلر های پردازنده:



	regwrite	ALUop	resultsrc	memwrite	jump	branch	ALUsrc	immsrc	lui
R-T	1	10	00	0	00	000	0	—	0
I-T	1	11	00	0	00	000	1	000	0
lw	1	00	01	0	00	000	1	000	0
S W	0	00	00	1	00	000	1	001	0
Jalr	1	00	10	0	10	000	1	000	0
jal	1	00	10	0	01	000	0	011	0
B-T	0	01	—	0	00	Funct3 — واسمه	0	010	0
U-T	1	— (00)	11	0	00	000	0	100	1

Top module controller:

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

    wire AluSrcD, memWriteD, regWriteD, luiD, func7;
    wire [1:0] resultSrcD, JumpD;
    wire [2:0] immSrcD, BranchD;
    wire [2:0] func3, AluControlD;
    wire [6:0] op;

    RiscV_controller riscontrol(op, func3, func7, AluSrcD, memWriteD, regWriteD, luiD, resultSrcD, JumpD, AluControlD, immSrcD, BranchD);
    RiscV_datapath riscdatapath(clk, rst, regWriteD, memWriteD, AluSrcD, luiD, resultSrcD, JumpD, AluControlD, BranchD, immSrcD, op, func3, func7);


endmodule
```

Hazard Unit:

```
module hazard(input reg [4:0] Rs1D, Rs2D, Rs1E, Rs2E, RdE, RdM, RdW,
              input PCSrcE, ResultSrcE0,
              input RegWriteM, RegWriteW,
              output  reg [1:0] ForwardAE, ForwardBE,
              output  StallF, StallD, FlushD, FlushE);
              wire lwStallD;
              // forwarding reg
              always @(*) begin
                  ForwardAE = 2'b00;
                  ForwardBE = 2'b00;
                  if (Rs1E != 5'b0) begin
                      if ((Rs1E == RdM) && RegWriteM) ForwardAE = 2'b10;
                      else if ((Rs1E == RdW) && RegWriteW) ForwardAE = 2'b01;
                  end
                  if (Rs2E != 5'b0) begin
                      if ((Rs2E == RdM) && RegWriteM) ForwardBE = 2'b10;
                      else if ((Rs2E == RdW) && RegWriteW) ForwardBE = 2'b01;
                  end
              end


    reg lwStall;
    assign lwStallD = ResultSrcE0 && ((Rs1D == RdE) || (Rs2D == RdE));
    assign StallF = lwStallD;
     assign StallD = lwStallD;

     assign FlushD = (PCSrcE != 2'b00) ? 1'b1 : 1'b0 ;
     assign FlushE = lwStallD || (PCSrcE != 2'b00);

endmodule
```

conditionals (jump, branch):

```
module conditional(input [1:0] JumpE, input [2:0] BranchE, input zero, input b31, output reg [1:0] PCSrcE);
parameter [2:0] NotBranch = 3'b000, Beq = 3'b001, Bne = 3'b010, Blt = 3'b011, Bge = 3'b100;
parameter [1:0] JAL = 2'b01, JALR = 2'b10;
   always@(JumpE, BranchE, zero, b31)begin
      case(BranchE)
         NotBranch : PCSrcE <= (JumpE == 2'b01)  ? 2'b01 :
                               (JumpE == 2'b10) ? 2'b10 : 2'b00;

         Beq : PCSrcE <= (zero) ? 2'b01 : 2'b00;
         Bne : PCSrcE <= (~zero) ? 2'b01 : 2'b00;
         Blt : PCSrcE <= (b31) ? 2'b01 : 2'b00;
         Bge : PCSrcE <= (zero | ~b31) ? 2'b01 : 2'b00;
        endcase
    end

endmodule
```

Main controller:

```
module main_controller(input [2:0] func3, input [6:0] op, output reg memWriteD, regWriteD, AluSrcD, luiD, output reg [1:0] resultSrcD, JumpD, Aluop, output reg [2:0] BranchD, immSrcD);
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_T = 7'b0000011, JALR_T = 7'b1100111;
parameter [2:0] Beq = 3'b000, Bne = 3'b001, Blt = 3'b010, Bge = 3'b011;


   always@(func3, op)begin
      {Aluop, regWriteD, immSrcD, AluSrcD, memWriteD,
       resultSrcD, JumpD, BranchD, luiD} <= 16'b0;
    case (op)
         R_T: begin
            Aluop     <= 2'b10;
            regWriteD  <= 1'b1;
         end

         I_T: begin
            Aluop     <= 2'b11;
            regWriteD  <= 1'b1;
            immSrcD    <= 3'b000;
            AluSrcD    <= 1'b1;
            resultSrcD <= 2'b00;
         end

         S_T: begin
            Aluop     <= 2'b00;
            memWriteD  <= 1'b1;
            immSrcD    <= 3'b001;
            AluSrcD    <= 1'b1;
         end

         B_T: begin
            Aluop     <= 2'b01;
            immSrcD    <= 3'b010;
             case(func3)
                 Beq : BranchD <= 3'b001;
                 Bne : BranchD <= 3'b010;
                 Blt : BranchD <= 3'b011;
                 Bge : BranchD <= 3'b100;
                 default: BranchD <= 3'b000;
             endcase
         end



      U_T: begin
          resultSrcD <= 2'b11;
          immSrcD    <= 3'b100;
          regWriteD  <= 1'b1;
          luiD <= 1'b1;
      end

      J_T: begin
          resultSrcD <= 2'b10;
           immSrcD    <= 3'b011;
           JumpD      <= 2'b01;
           regWriteD  <= 1'b1;
      end

      LW_T: begin
         Aluop      <= 2'b00;
           regWriteD  <= 1'b1;
           immSrcD    <= 3'b000;
           AluSrcD    <= 1'b1;
           resultSrcD <= 2'b01;
      end

      JALR_T: begin
           Aluop      <= 2'b00;
           regWriteD  <= 1'b1;
           immSrcD    <= 3'b000;
           AluSrcD    <= 1'b1;
           JumpD      <= 2'b10;
           resultSrcD <= 2'b10;
      end
      default: begin
           regWriteD <= 1'b0;
           AluSrcD   <= 2'b00;
           Aluop     <= 3'b000;
      end

   endcase
nd
```

یافتن بزرگترین عضو یک آرایه ی 10 تایی از اعداد

صحیح :

طراحی شده عملیات یافتن بزرگترین عدد در یک آرایه 10 تایی از اعداد صحیح را RISC-V پردا زنده ی
:بصورت مجموعه دستورات زیر انجام میدهد

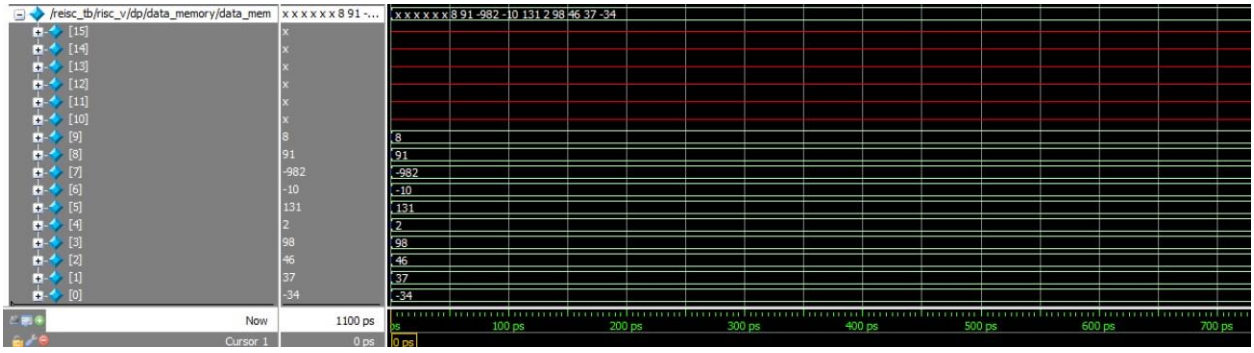
```
jal x0, ■axi■u■

   ■axi■u■:
       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 X 19 رجیستر فایل لود میشوند و هر بار با محتوای رجیستر X8 مقایسه میشوند و در صور تی که مقدار X19 بزرگتر از مقدار X8 باشد ، مقدار X19 در X8 ریخته میشود و حلقه تکرار میشود و محتوا ی اندیس بعدی آرایه در X19ریخته میشود و عملیات مقایسه تکرار میشود تا در آخر بزرگترین مقدار حاضر در آرایه در رجیستر X8 قابل مشاهده باشد .

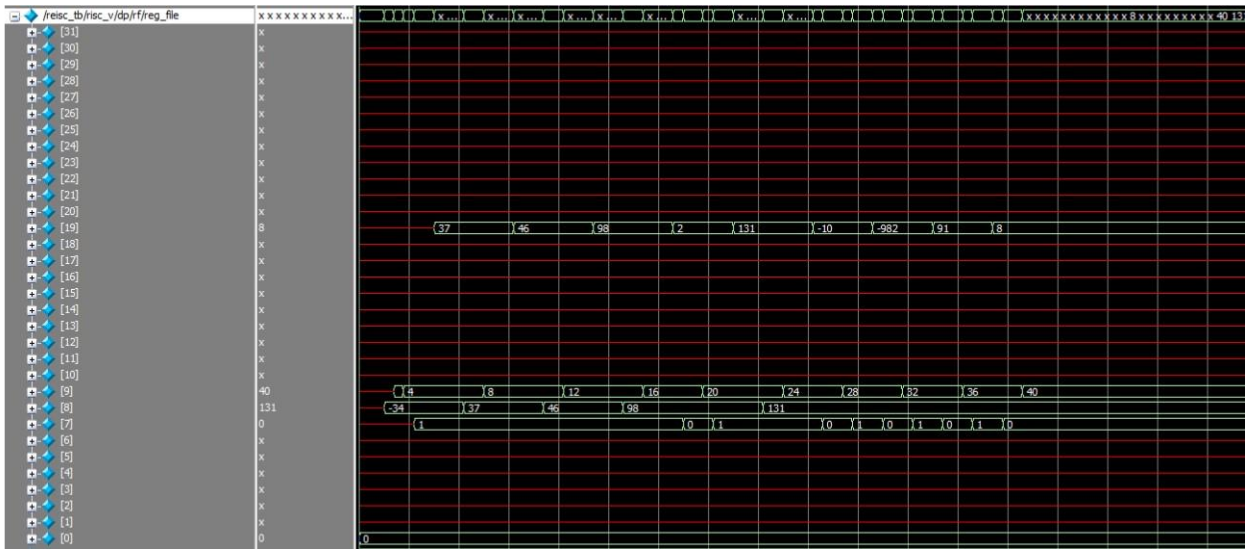
)در دستورات اسمبلی که در صفحه قبل آورده شدند ، S0 همان رجیستر X8 و S3 همان رجیستر X19 را بیان میکند.)

در ویوفرم زیر عملیات توضیح داده شده قابل مشاهده است:

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 131) را بیابیم :



در تصویر زیر تغییرات مقادیر رجیستر های X19 , X8 گویای عملیات یاد شده هستند:


/reisc_tb/risc_v/dp/rf/reg_file
[31]
[30]
[29]
[28]
[27]
[26]
[25]
[24]
[23]
[22]
[21]
[20]
[19]
[18]
[17]
[16]
[15]
[14]
[13]
[12]
[11]
[10]
[9]
[8]
[7]
[6]
[5]
[4]
[3]
[2]
[1]
[0]
37
46
98
2
131
-10
-982
91
8
4
8
12
16
20
24
28
32
36
40
-34
37
46
98
131
1
0
1
0
1
0
1
0
1
0
0