中国科学技术大学计算机学院

《计算机组成原理实验报告》

实验题目：流水线CPU设计

学生姓名：林宸昊

学生学号：PB20000034

完成日期：2022.5.1

【实验题目】流水线CPU设计

【实验目的】

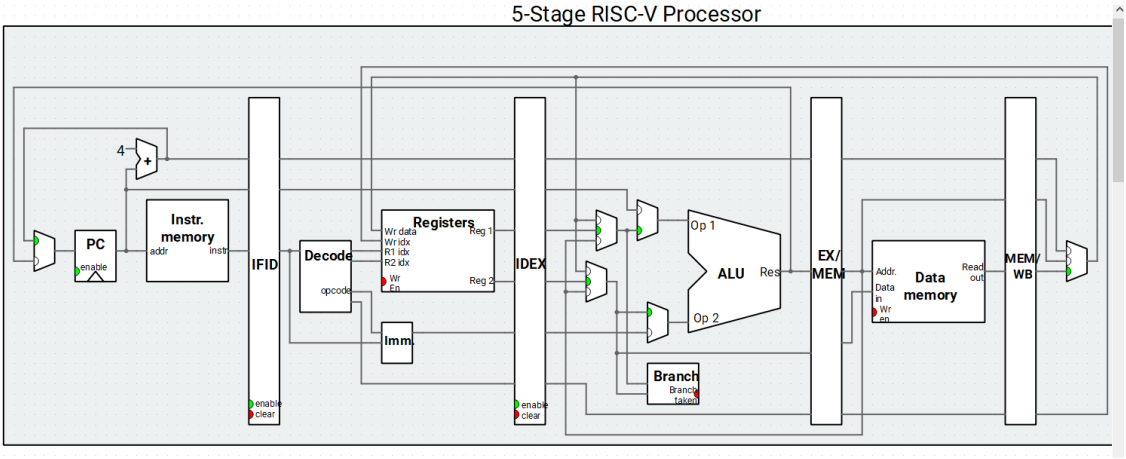
- 理解流水线CPU结构和工作原理；
- 掌握流水线CPU设计和调试方法，尤其是其中数据相关和控制相关的处理；
- 熟练掌握数据通路及控制器的设计和描述方法。

【实验环境】VIVADO FPGAOL

【实验步骤】

【一、整体数据通路】

- 大体框架采用Ripes上所提供的五级流水线数据通路：

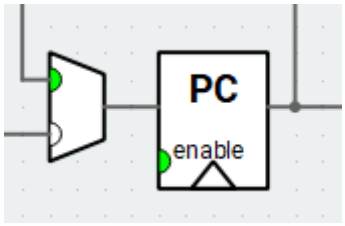


- 其中未详细标示的各数据段寄存器以及前递、冒险处理单元参照ppt内容进行整合。

【二、CPU设计】

【PC】

- 在数据通路中的位置



- 模块设计

```
module pc5(
    input [31:0] in,
    input pc_en,
    input rst,
    input clk,
    output reg [31:0] out
);
    always @ (posedge clk or posedge rst)
    begin
        if(rst)
            out <= 32'h0000_3000;
        else if(pc_en)
            out <= in;
        else
            out <= out;
    end
endmodule
```

- 模块例化

```
pc5 pc(
    .in(pc_in),
    .pc_en(pc_en),
    .rst(rst),
    .clk(clk),
    .out(pc)
);
```

事实上此处的pc_in由图可知，由一个二选一选择器产生，在设计文件中如此表现：

```
wire [31:0] pc_1, pc_0;
assign pc_0 = pc + 4;
assign pc_1 = (imm << 1) + pce; //pce and imm are from ID stage
wire pcSrc; //decided by jump or not

mux2_1 MUX_2_1_pc_in(
    .a(pc_0),
    .b(pc_1),
    .sel(pcSrc),
    .o(pc_in)
);
```

【INS】

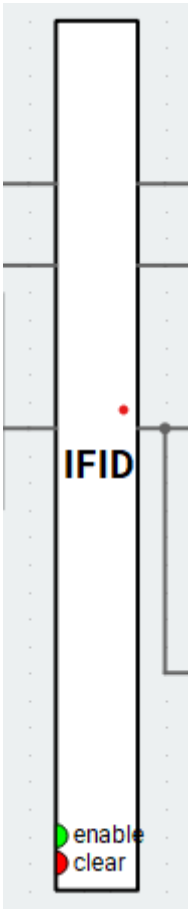
- 数据通路中的位置

- 模块例化（同单周期CPU采用分布式只读存储器）

```
dist_mem_gen_0 ins_mem(
    .a(pc[9:2]),
    .spo(INS)
);
```

【IFID】

- 数据通路中的位置



涉及传递信号：pcd, pcin, ir

- 模块设计

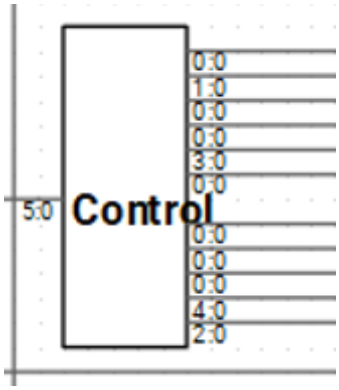
```
module IFID(
    input clk,
    input rs1_en, rs1_clr,
    input [31:0] pc, pc_0,
    input [31:0] INS,
    output reg [31:0] pcd, pcin, ir
    );
    always @(posedge clk)
    begin
        if (rs1_clr)    //clear if needed
        begin
            ir <= 0;
            pcd <= 0;
            pcin <= 0;
        end
        else if(rs1_en)     //else herit from last stage
        begin
            ir <= INS;
            pcd <= pc;
            pcin <= pc_0;
        end
        else begin          //else keep
            ir <= ir;
            pcd <= pcd;
            pcin <= pcin;
        end
    end
endmodule
```

- 模块例化

```
IFID IFID(
    .clk(clk),
    .rs1_en(rs1_en),    //enable signal from hazard
    .rs1_clr(rs1_clr),  //clear signal from hazard
    .pc(pc),
    .pc_0(pc_0),
    .INS(INS),
    .pcin(pcin),        //pc_0 to pcin
    .pcd(pcd),          //pc to pcd
    .ir(ir)             //INS to ir
);
```

【controler——即control信号产生器（对应ppt）】

- 数据通路中的位置



- 模块设计

```
module controler(
    input [6:0] ins,            //ir signal
    input [1:0] a_fwd, b_fwd,   //from forward
    output reg [31:0] control   //control signal
);
    parameter ALUresult = 2'b01;
    parameter DataMem = 2'b10;
    parameter PCadd4 = 2'b11;
    parameter IDIE = 2'b00;
    always @ (*)
    begin
        //首先处理默认信号以及输入信号（非自我产生）
        control[31:26] = 0;
        control[23:22] = 0;
        control[19] = 0;
        control[15:14] = 0;
        control[11:10] = 0;
        control[7:6] = 0;
        control[25:24] = a_fwd;
        control[21:20] = b_fwd;
        case(ins)
            7'b0110011:
            begin   //add
                control[3:0] = 4'b0000; //alu_op
                control[5:4] = 2'b11;   //a_sel && b_sel
                control[9:8] = 2'b00;   //jal && br
```

```
                control[13:12] = 2'b00; //m_rd && m_wr
                control[17:16] = 2'b01; //wb_sel, 01 means choosing ALUres
                control[18] = 1;        //rf_wr
            end
            7'b0010011:
            begin   //addi
                control[3:0] = 4'b0000;
                control[5:4] = 2'b10;
                control[9:8] = 2'b00;
                control[13:12] = 2'b00;
                control[17:16] = 2'b01; //choose ALUres
                control[18] = 1;
            end
            7'b0000011:
            begin   //lw
                control[3:0] = 4'b0000;
                control[5:4] = 2'b10;
                control[9:8] = 2'b00;
                control[13:12] = 2'b10;
                control[17:16] = 2'b10; //choose data from memory
                control[18] = 1;
            end
            7'b0100011:
            begin   //sw
                control[3:0] = 4'b0000;
                control[5:4] = 2'b10;
                control[9:8] = 2'b00;
                control[13:12] = 2'b01;
                control[17:16] = 2'b00; //choose data from IDIE
                control[18] = 0;
            end
            7'b1100011:
            begin   //beq
                control[3:0] = 4'b0001;
                control[5:4] = 2'b11;
                control[9:8] = 2'b01;
                control[13:12] = 2'b00;
                control[17:16] = 2'b00; //choose data from IDIE
                control[18] = 0;
            end
            7'b1101111:
            begin   //jal
                control[3:0] = 4'b0000;
                control[5:4] = 2'b00;
                control[9:8] = 2'b10;
                control[13:12] = 2'b00;
                control[17:16] = 2'b11; //choose pc + 4
                control[18] = 1;
            end
            default:
            begin
                control[3:0] = 4'b0000;
                control[5:4] = 2'b00;
                control[9:8] = 2'b00;
                control[13:12] = 2'b00;
                control[17:16] = 2'b00;
                control[18] = 0;
            end
```

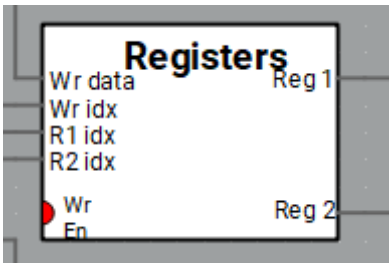
```
        endcase
    end
endmodule
```

- 模块例化

```
controler controler(
    .ins(ir[6:0]),
    .a_fwd(a_fwd),
    .b_fwd(b_fwd),
    .control(control)   //original signal
);
```

【registers】

- 数据通路中的位置



- 模块设计

```
module registers # (parameter WIDTH = 32) (
    input clk,
    input [4:0] ra0,
    output reg [WIDTH - 1:0] rd0,
    input [4:0] ra1,
    output reg [WIDTH - 1:0] rd1,
    input [4:0] wa,
    input we,
    input [WIDTH - 1:0] wd,
    input [7:0] ra2,
    output reg [WIDTH - 1:0] rd_debug
);

    reg [WIDTH - 1:0] regfile [31:0];
    initial
    begin
    regfile[0] = 0;
    regfile[1] = 0;
    regfile[2] = 0;
    regfile[3] = 0;
    regfile[4] = 0;
    regfile[5] = 0;
    regfile[6] = 0;
    regfile[7] = 0;
    regfile[8] = 0;
    regfile[9] = 0;
    regfile[10] = 0;
    regfile[11] = 0;
    regfile[12] = 0;
    regfile[13] = 0;
```

```
    regfile[14] = 0;
    regfile[15] = 0;
    regfile[16] = 0;
    regfile[17] = 0;
    regfile[18] = 0;
    regfile[19] = 0;
    regfile[20] = 0;
    regfile[21] = 0;
    regfile[22] = 0;
    regfile[23] = 0;
    regfile[24] = 0;
    regfile[25] = 0;
    regfile[26] = 0;
    regfile[27] = 0;
    regfile[28] = 0;
    regfile[29] = 0;
    regfile[30] = 0;
    regfile[31] = 0;
    end

    always @ (*) begin  //基本与单周期一致，此处额外处理以满足写优先
        if(we)
        begin
                if(wa == ra0)
                    rd0 = wd;
                else
                    rd0 = regfile[ra0];
                if(wa == ra1)
                    rd1 = wd;
                else
                    rd1 = regfile[ra1];
                if(wa == ra2)
                    rd_debug = wd;
                else
                    rd_debug = regfile[ra2];
         end
         else
         begin
                rd0 = regfile[ra0];
                rd1 = regfile[ra1];
                rd_debug = regfile[ra2];
            end
     end

    always  @ (posedge  clk)
        if (we)
            if(wa == 0)
            begin
                regfile[wa] <= 0;
            end
            else
            begin
                regfile[wa] <= wd;
            end
endmodule
```

- 模块例化

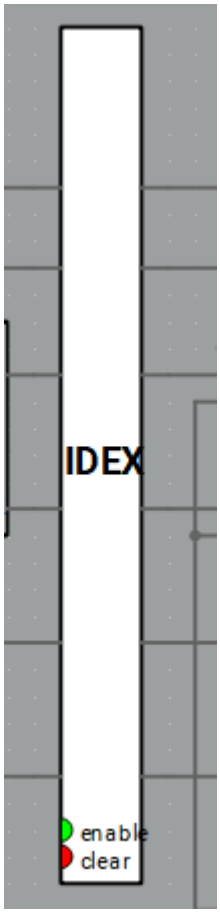
```
register_file Registers(
    .clk(clk),
    .ra0(ir[19:15]),            //rs1
    .rd0(ReadData1_reg),
    .ra1(ir[24:20]),            //rs2
    .rd1(ReadData2_reg),
    .wa(rdw),                   //dest from MEM stage
    .we(ctrlw[18]),
    .wd(WriteData),             //write
    .ra2(m_rf_addr[7:0]),       //used for debug
    .rd_debug(rf_data[31:0])    //used for debug
);
```

【immediate generator】

- 与单周期一致。

【IDEX】

- 数据通路中的位置



涉及传递信号：ctrl, pce, pcin2, a, b, imm, ra1, ra2, rd

- 模块设计

```
module IDEX(
    input clk,
    input rs2_en, rs2_clr,
    input [31:0] control,
    input [31:0] pcin, pcd,
```

```
    input [31:0] rs1, rs2,
    input [31:0] ir, Immgen,
    output reg[31:0] ctrl,
    output reg[31:0] pce, pcin2,
    output reg[31:0] a, b,
    output reg[31:0] imm,
    output reg[4:0] rd, ra1, ra2
    );
    initial begin
        ctrl = 0;
        a = 0;
        b = 0;
    end
    always @(posedge clk)
    begin
        if(rs2_clr)
        begin
            ctrl <= 0;
            pcin2 <= 0;
            pce <= 0;
            a <= 0;
            b <= 0;
            imm <= 0;
            rd <= 0;
            ra1 <= 0;
            ra2  <= 0;
        end
        else if(rs2_en)
        begin
            ctrl <= control;
            pcin2 <= pcin;
            pce <= pcd;
            a <= rs1;
            b <= rs2;
            imm <= Immgen;
            rd <= ir[11:7];
            ra1 <= ir[19:15];
            ra2 <= ir[24:20];
        end
        else
        begin
            ctrl <= ctrl ;
            pcin2 <= pcin2;
            pce <= pce  ;
            a <= a    ;
            b <= b    ;
            imm <= imm  ;
            rd <= rd   ;
            ra1 <= ra1  ;
            ra2 <= ra2  ;
        end
    end
endmodule
```

- 模块例化

```
IDEX IDEX(
```

```
    .clk(clk),
    .control(control),
    .ctrl(ctrl),              //control to ctrl
    .rs1(ReadData1_reg),
    .rs2(ReadData2_reg),
    .a(a),                    //rs1 to a
    .b(b),                    //rs2 to b
    .Immgen(Immgen),
    .imm(imm),                //Immgen to imm
    .ir(ir),
    .rd(rd),                  //ir[11:7]
    .ra1(ra1),                //ir[19:15]
    .ra2(ra2),                //ir[24:20]
    .pcin(pcin),
    .pcin2(pcin2),            //pcin to pcin2
    .pcd(pcd),
    .pce(pce),                //pcd to pce
    .rs2_en(rs2_en),          //from hazard
    .rs2_clr(rs2_clr)         //from hazard
);
```

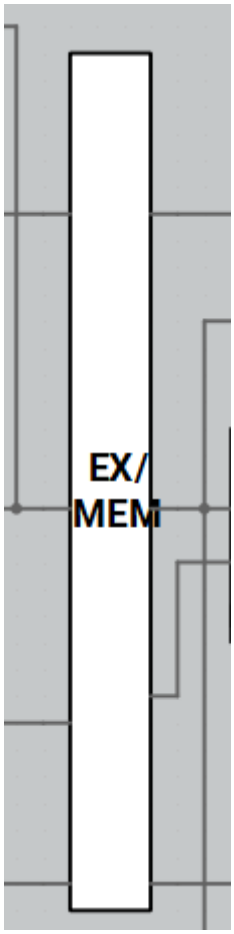
【alu】

- 与单周期一致，直接沿用；
- 模块例化

```
alu alu(
    .a(op1),
    .b(op2),
    .f(ctrl[3:0]),
    .y(ALU_result),
    //.blt(blt),
    .z(ALU_zero)
    //.less(ALU_less)
);
```

【EXMEM】

- 数据通路中的位置



涉及传递的信号：ctrlm, y, bm, pcin3, rdm, zero

- 模块设计

```
module EXMEM(
    input clk,
    input [31:0] ALU_result, op2, ctrl, pcin2, zero_n, ALU_zero,
    input [4:0] rd,
    output reg [31:0] ctrlm, y, bm, pcin3,
    output reg [4:0] rdm,
    output reg zero
);
    always @ (posedge clk)
    begin
        y <= ALU_result;
        zero <= ALU_zero;
        bm <= op2;
        ctrlm <= ctrl;
        pcin3 <= pcin2;
        rdm <= rd;
    end
endmodule
```

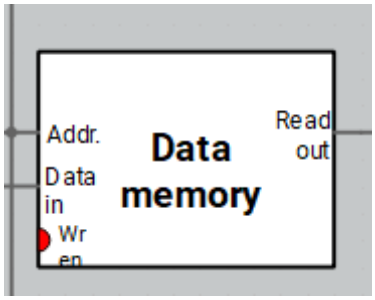
- 模块例化

```
EXMEM EXMEM(
    .clk(clk),
    .ALU_result(ALU_result),
    .y(y),                      //ALU_result to y
    .ALU_zero(ALU_zero),
    .zero(zero),                //ALU_zero to zero
```

```
    .op_r(op_r),
    .bm(bm),                        //op_r to bm
                                    //事实上此处的op_r并非直接进入alu的op2，还需要与
                                    //imm竞争
    .ctrl(ctrl),
    .ctrlm(ctrlm),                  //ctrl to ctrlm
    .rd(rd),
    .rdm(rdm),                      //rd to rdm
    .pcin2(pcin2),
    .pcin3(pcin3)                   //pcin2 to pcin3
);
```

【data memory】

- 数据通路中的位置

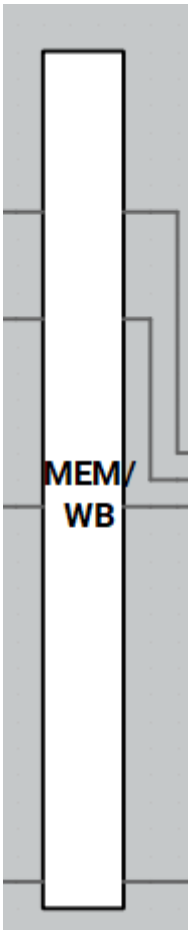


- 模块例化（采用同单周期一致的双端口分布式存储器）

```
assign we = ctrlm[12] && (~y[10]);  //write or not

dist_mem_gen_1 data_mem(
    .a(y[9:2]),
    .d(bm),
    .dpra(m_rf_addr[7:0]),  //second address
    .dpo(m_data[31:0]),     //ReadData2
    .clk(clk),
    .we(we),
    .spo(ReadData1_data)    //ReadData1
);
```

【MEMWB】

- 数据通路中的位置



涉及传递的信号：rdw, yw, mdr, ctrlw, pcin4

- 模块设计

```
module MEMWB(
    input clk,
    input [4:0] rdm,
    input [31:0] y, ReadData, ctrlm, pcin3,
    output reg [31:0] yw, mdr, ctrlw, pcin4,
    output reg [4:0] rdw
    );
    always @(posedge clk)
    begin
        rdw <= rdm;
        yw <= y;
        mdr <= ReadData;
        ctrlw <= ctrlm;
        pcin4 <= pcin3;
    end
endmodule
```

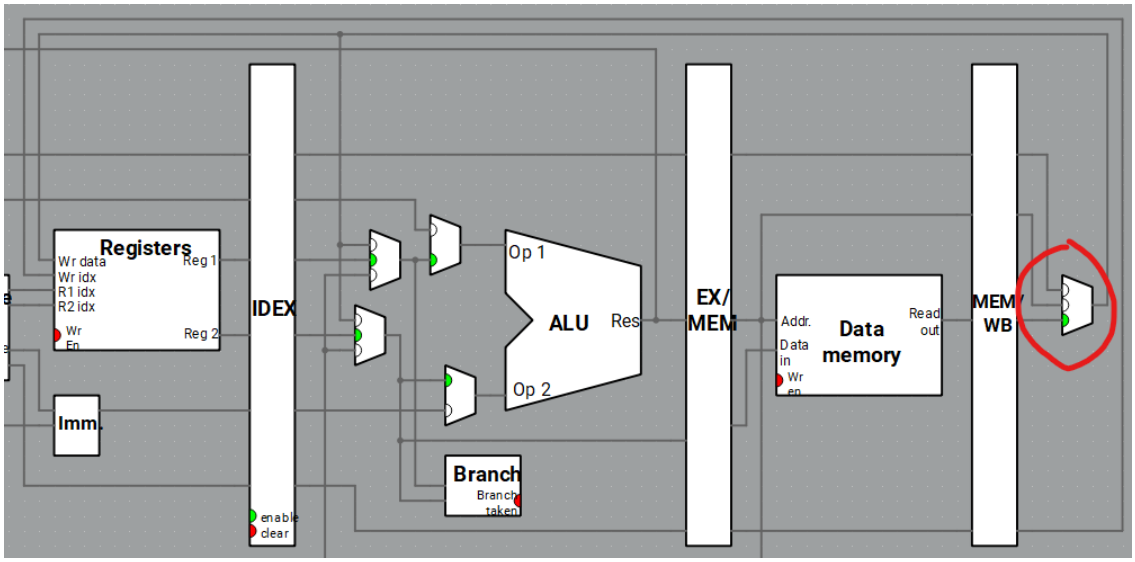
- 模块例化

```
mux2_1 MUX_2_1_data_mem(
    .a(ReadData1_data),
    .b(io_din[31:0]),
    .sel(y[10]),
    .o(MUX_2_1_out_data)
);
```

```
MEMWB MEMWB(
    .clk(clk),
    .rdm(rdm),
    .rdw(rdw),                          //rdm to rdw
    .y(y),
    .yw(yw),                            //y to yw
    .ctrlm(ctrlm),
    .ctrlw(ctrlw),                      //ctrlm to ctrlm
    .ReadData(MUX_2_1_out_data),
    .mdr(mdr),                          //io_din or read from data memory
    .pcin3(pcin3),
    .pcin4(pcin4)                       //pcin3 to pcin4
);
```

【WriteData——寄存器写入数据的处理】

- 数据通路中的位置

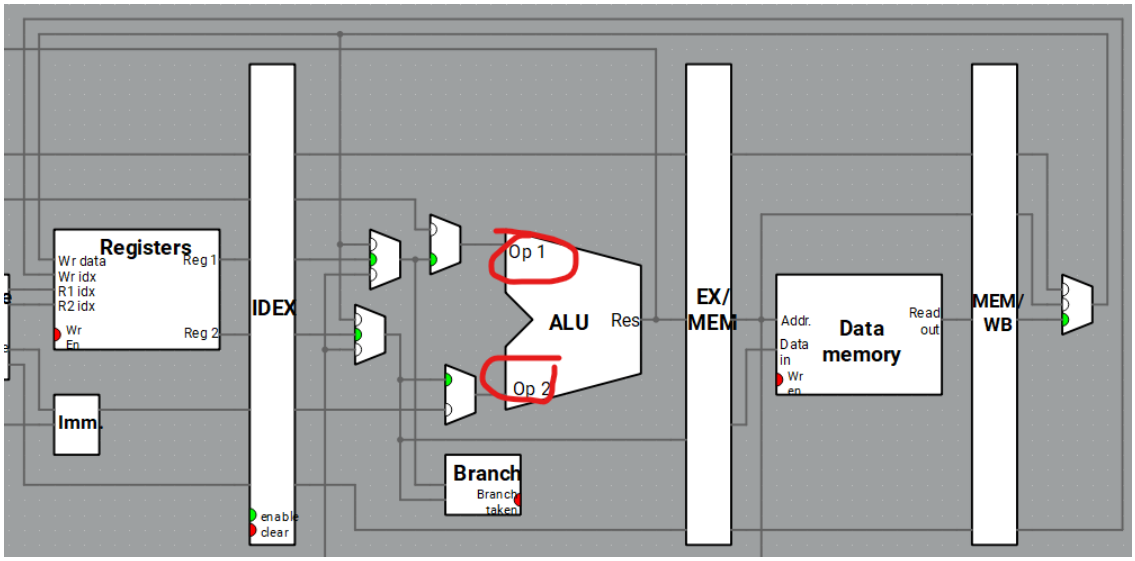


- 模块设计

```
always @ (*) begin
    case(ctrlw[17:16])
        2'b01:WriteData = yw;
        2'b10:WriteData = mdr;
        2'b11:WriteData = pcin4;
        default:WriteData = 0;
    endcase
end
```

【op1 && op2】

- 数据通路中的位置



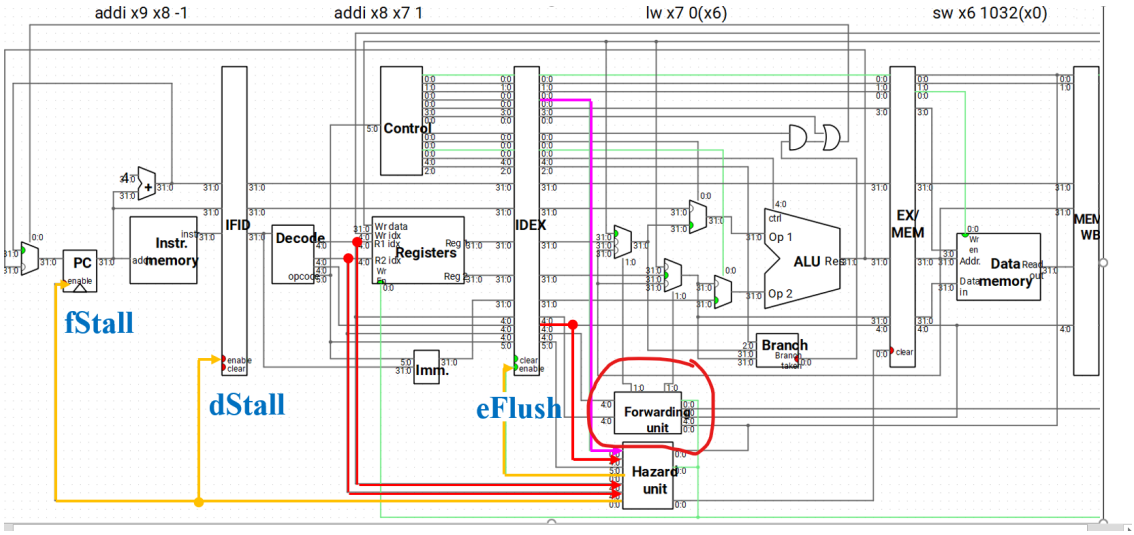
- 模块设计

```
always @(*) begin
    case (control[25:24])       //a_fwd
        2'b00:op1 = a;
        2'b01:op1 = y;
        2'b10:op1 = WriteData;
    endcase
end

always @(*) begin
    case (control[21:20])       //b_fwd
        2'b00:op_r = b;
        2'b01:op_r = y;
        2'b10:op_r = WriteData;
        default:op_r = LastWriteData;
    endcase
end

mux2_1 MUX_2_1_pc(
    .a(imm),
    .b(op_r),
    .sel(ctrl[4]),
    .o(op2)
);
```

值得注意的是，不同处选择的是不同阶段传递得到的control信号，如是否选择立即数取决于EXMEM阶段所产生的ctrl信号而非直接产生的control信号。

【forwarding unit】

- 数据通路中的位置



- 模块设计

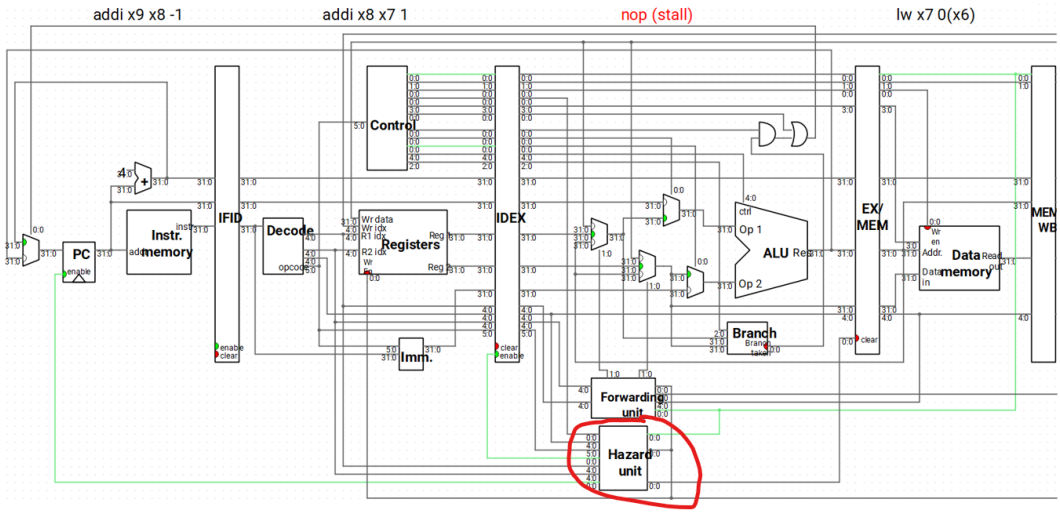
```
module forward(
    input [4:0] ra1, ra2, rdm, rdw,
    input wr_m, wr_w,
    output reg [1:0] a_fwd, b_fwd
    );
    always @ (*) begin
        if(wr_m && ra1 == rdm)          //寄存器写入且需要读取的寄存器与EXMEM阶段
                                        //写入的目标相同，进行数据前递
            a_fwd = 2'b01;
        else if(wr_w && ra1 == rdw)     //寄存器写入且需要读取的寄存器与MEMWB阶段
                                        //写入的目标相同，进行数据前递
            a_fwd = 2'b10;
        else a_fwd = 2'b00;
    end
    always @ (*) begin
        if(wr_m && ra2 == rdm)
            b_fwd = 2'b01;
        else if(wr_w && ra2 == rdw)
            b_fwd = 2'b10;
        else b_fwd = 2'b00;
    end
endmodule
```

- 模块例化

```
forward forwarding_unit(
    .a_fwd(a_fwd),
    .b_fwd(b_fwd),
    .ra1(ra1),
    .ra2(ra2),
    .rdm(rdm),
    .rdw(rdw),
    .rdf(rdf),
    .wr_m(ctrlm[18]),
    .wr_w(ctrlw[18]),
);
```

【hazard unit】

- 数据通路中的位置



- 模块设计

```
module hazard(
    input rd_idex, wr_ifid,
    input [4:0] rd, ra1, ra2,
    input zero, beq, jal,
    input a_sel, b_sel,
    output reg pc_en, rs1_en, rs2_en, rs1_clr, rs2_clr
    );
    always @(*) begin
        if (jal || (zero && beq))    //若发生跳转，则写入新pc，同时清空IF及ID（已进
//入的ir）
        begin
            pc_en = 1;
            rs1_clr = 1;
            rs1_en = 0;
            rs2_clr = 1;
            rs2_en = 0;
        end
        else if( rd_idex                        //IDEX阶段需要读取datamem
            && ( (a_sel && rd == ra1)           //此处亦有两种情况：
                                                //与ra1冲突或者与ra2冲突
                || ((b_sel || wr_ifid )         //对应lw或sw
                        && rd == ra2) ) )
        begin
            pc_en = 0;
            rs1_clr = 0;
            rs1_en = 0;
            rs2_clr = 1;                        //需要等待一个周期以进行存储器的读写
            rs2_en = 0;
        end
        else begin
            pc_en = 1;
            rs1_clr = 0;
            rs1_en = 1;
            rs2_clr = 0;
            rs2_en = 1;
        end
    end
```

```
endmodule
```

- 模块例化

```
hazard hazard_unit(
    .rd_idex(ctrl[13]),         //from ID stage
    .wr_ifid(control[12]),      //from IF stage
    .zero(ALU_zero),
    .beq(ctrl[8]),
    .jal(ctrl[9]),
    .a_sel(control[5]),
    .b_sel(control[4]),
    .rd(rd),
    .ra1(ir[19:15]),
    .ra2(ir[24:20]),
    .pc_en(pc_en),
    .rs1_en(rs1_en),
    .rs1_clr(rs1_clr),
    .rs2_en(rs2_en),
    .rs2_clr(rs2_clr)
);
```

【io处理】

- 模块设计

  不同于单周期，多周期中的io需要连接在EXMEM的输出中，以保证取得的是需要的结果而不是有可能被临时更改或者刷洗的值。

```
assign io_addr = y[7:0];
assign io_dout = bm;
assign io_we = ctrlm[12];
```

【cpu】

- 模块例化（包括pdu调试）

```
module CPU5_one_cycle(
    input clk,rst,
    input run,step,
    input valid,
    input [4:0] in,
    output [1:0] check,
    output [4:0] out0,
    output [2:0] an,
    output [3:0] seg,
    output ready
    );

    //IO_BUS
    wire [7:0] io_addr;
    wire [31:0] io_dout;
    wire io_we;
    wire [31:0] io_din;

    //Debug_BUS
```

```
    wire [7:0] m_rf_addr;
    wire [31:0] rf_data;
    wire [31:0] m_data;
    wire [31:0] pc;

    wire [31:0] pcin, pcd, pce;
    wire [31:0] ir, imm, mdr;
    wire [31:0] a, b, y, bm, yw;
    wire [4:0]  rd, rdm, rdw;
    wire [31:0] ctrl, ctrlm, ctrlw;

    wire clk_cpu;

    cpu5 cpu(
        .clk(clk_cpu),
        .rst(rst),

        .io_addr(io_addr),
        .io_dout(io_dout),
        .io_we(io_we),
        .io_din(io_din),

        .m_rf_addr(m_rf_addr),
        .rf_data(rf_data),
        .m_data(m_data),
        .pc(pc),

        .pcin(pcin),
        .pcd(pcd),
        .pce(pce),
        .ir(ir),
        .imm(imm),
        .mdr(mdr),
        .a(a),
        .b(b),
        .y(y),
        .bm(bm),
        .yw(yw),
        .rd(rd),
        .rdm(rdm),
        .rdw(rdw),
        .ctrl(ctrl),
        .ctrlm(ctrlm),
        .ctrlw(ctrlw)
    );

    pdu5 debug(
        .clk(clk),
        .rst(rst),

        .run(run),
        .step(step),
        .clk_cpu(clk_cpu),

        .valid(valid),
        .in(in),

        .check(check),
```

```
        .out0(out0),
        .an(an),
        .seg(seg),
        .ready(ready),

        .io_addr(io_addr),
        .io_dout(io_dout),
        .io_we(io_we),
        .io_din(io_din),

        .m_rf_addr(m_rf_addr),
        .rf_data(rf_data),
        .m_data(m_data),
        .pc(pc),

        .pcin(pcin),
        .pcd(pcd),
        .pce(pce),
        .ir(ir),
        .imm(imm),
        .mdr(mdr),
        .a(a),
        .b(b),
        .y(y),
        .bm(bm),
        .yw(yw),
        .rd(rd),
        .rdm(rdm),
        .rdw(rdw),
        .ctrl(ctrl),
        .ctrlm(ctrlm),
        .ctrlw(ctrlw)
    );

endmodule
```

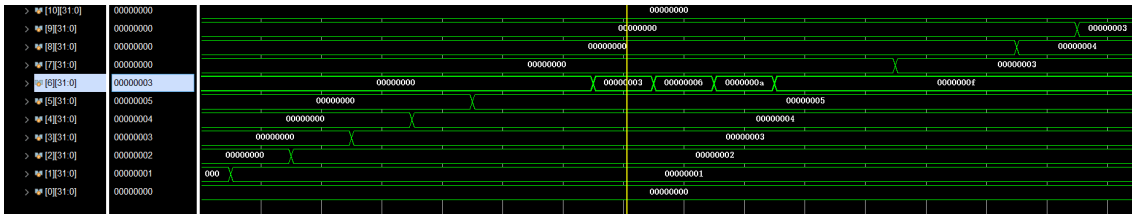
【二、hazard_test.coe仿真测试】

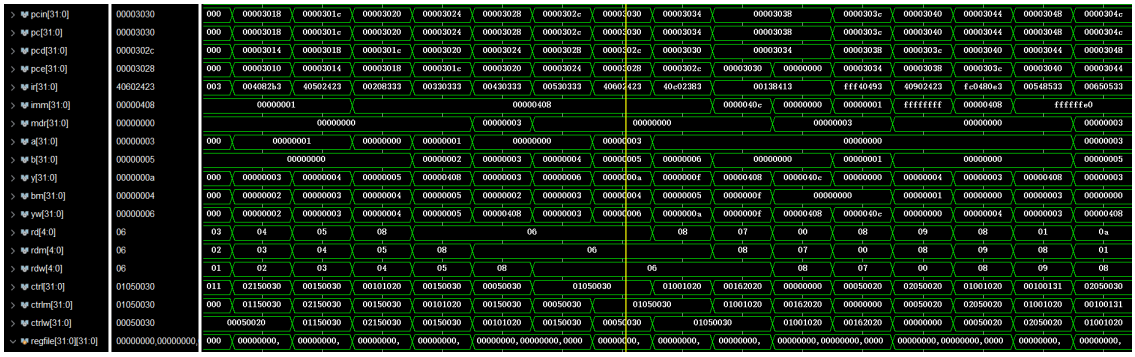
- 汇编代码

```
start:
sw x0, 0x408(x0)    #out1=0

#test data hazards
addi x1, x0, 1  #x1=1
addi x2, x1, 1  #x2=2
add x3, x1, x2  #x3=3
add x4, x1, x3  #x4=4
add x5, x1, x4  #x5=5
sw x5, 0x408(x0) #out1=5

add x6, x1, x2  #x6=3
add x6, x6, x3  #x6=6
add x6, x6, x4  #x6=10
add x6, x6, x5  #x6=15
sw x6, 0x408(x0)    #out1=15
```

```
#test load-use hazard
lw x7, 0x410(x0) #x7=in
addi x8, x7, 1   #x8=in+1
addi x9, x8, -1  #x9=in
sw x9, 0x408(x0)    #out1=in
```

- 仿真结果

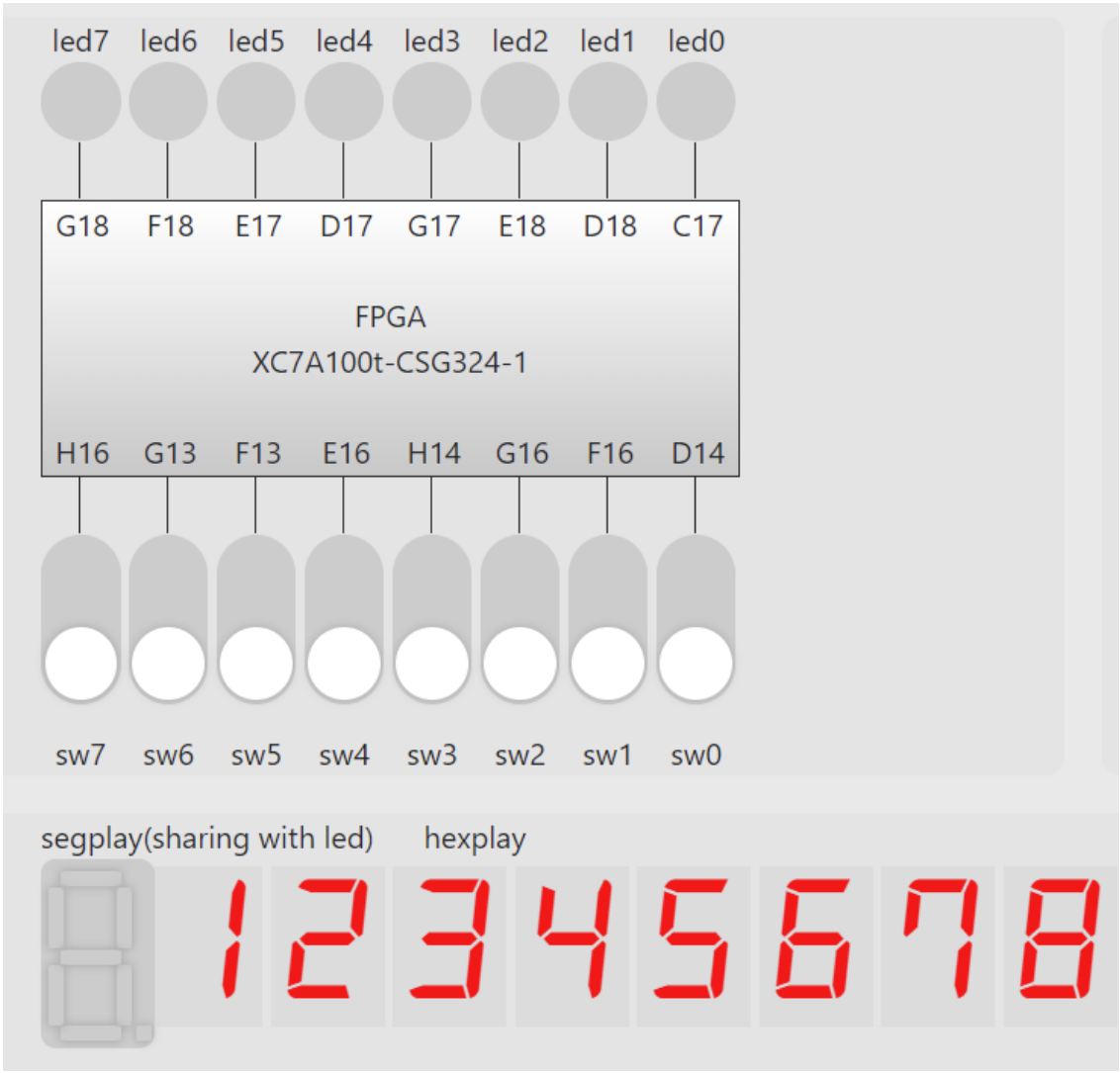




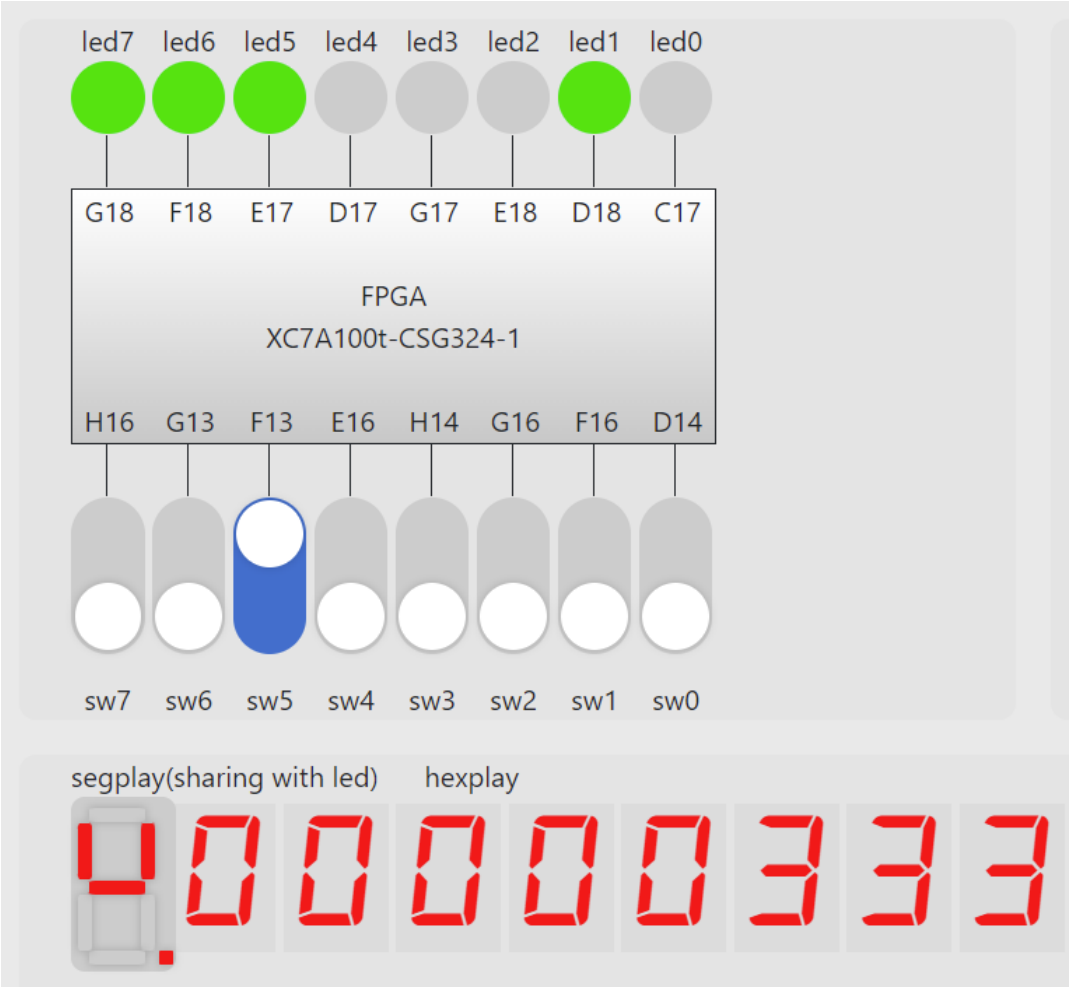
与代码注释应得结果一致。

【三、斐波那契coe烧写】

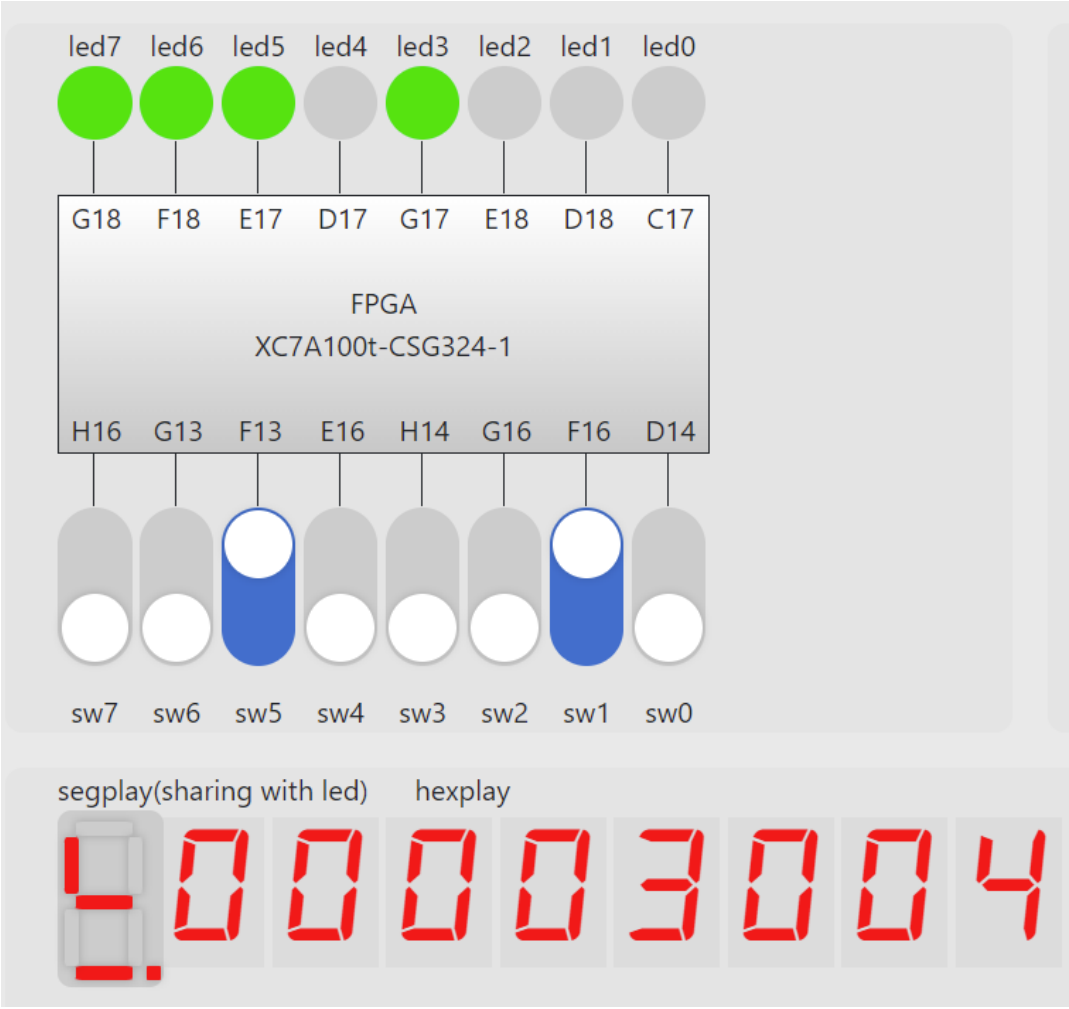
- 初始化


led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
1 2 3 4 5 6 7 8

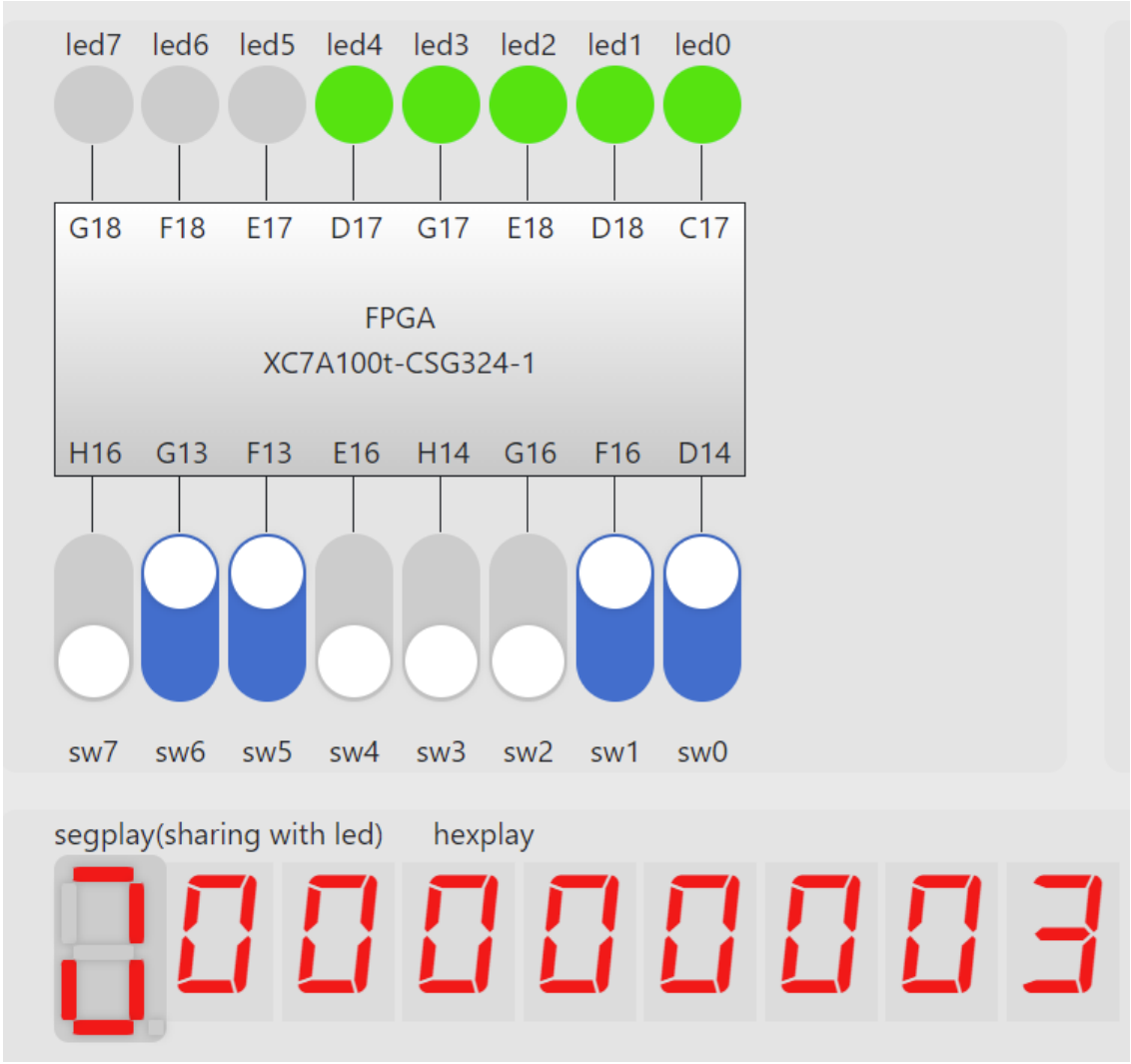
- 单步运行两次
  - 查看ir


led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
40000333

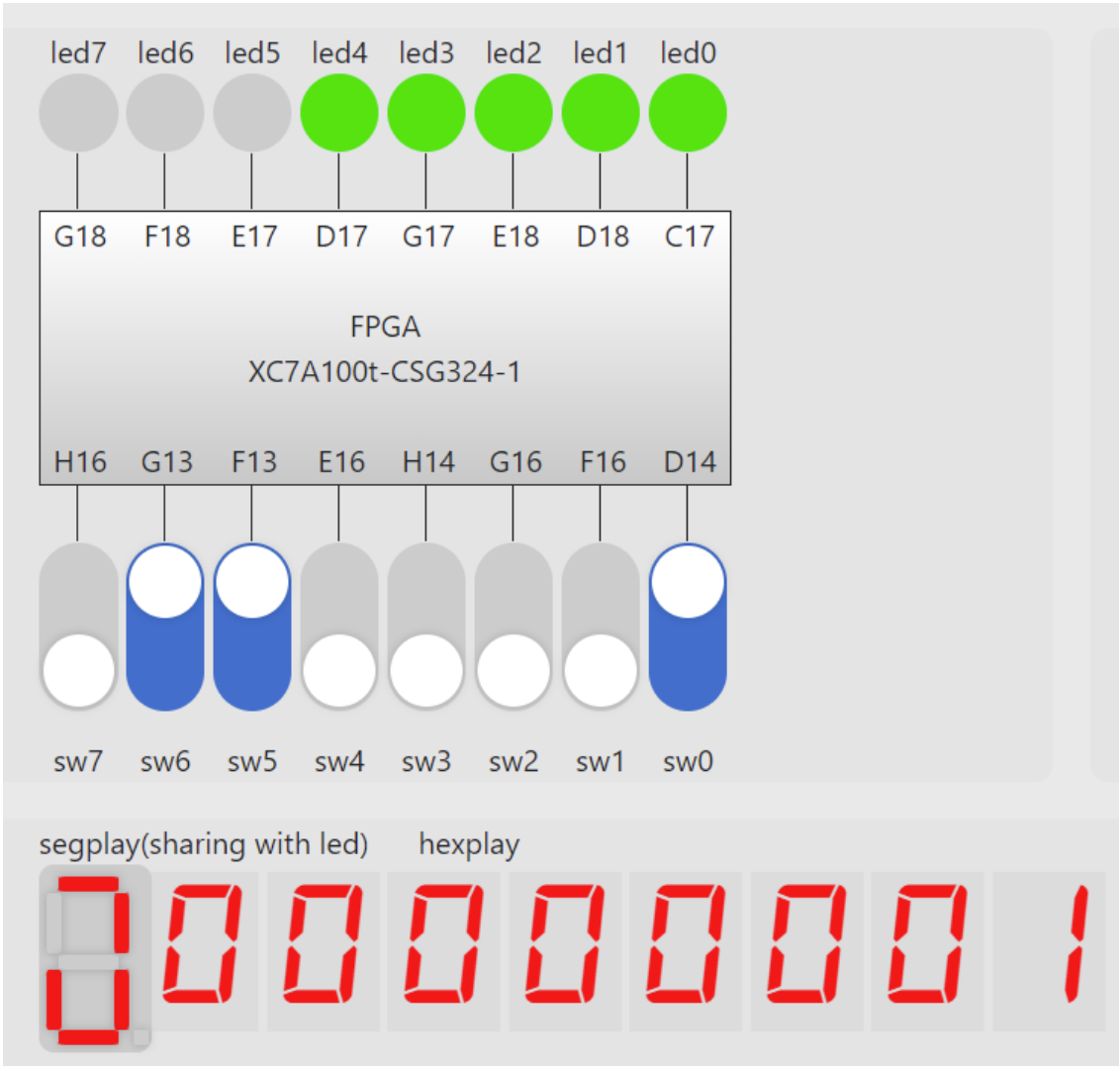
- 查看pce


led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
40003004

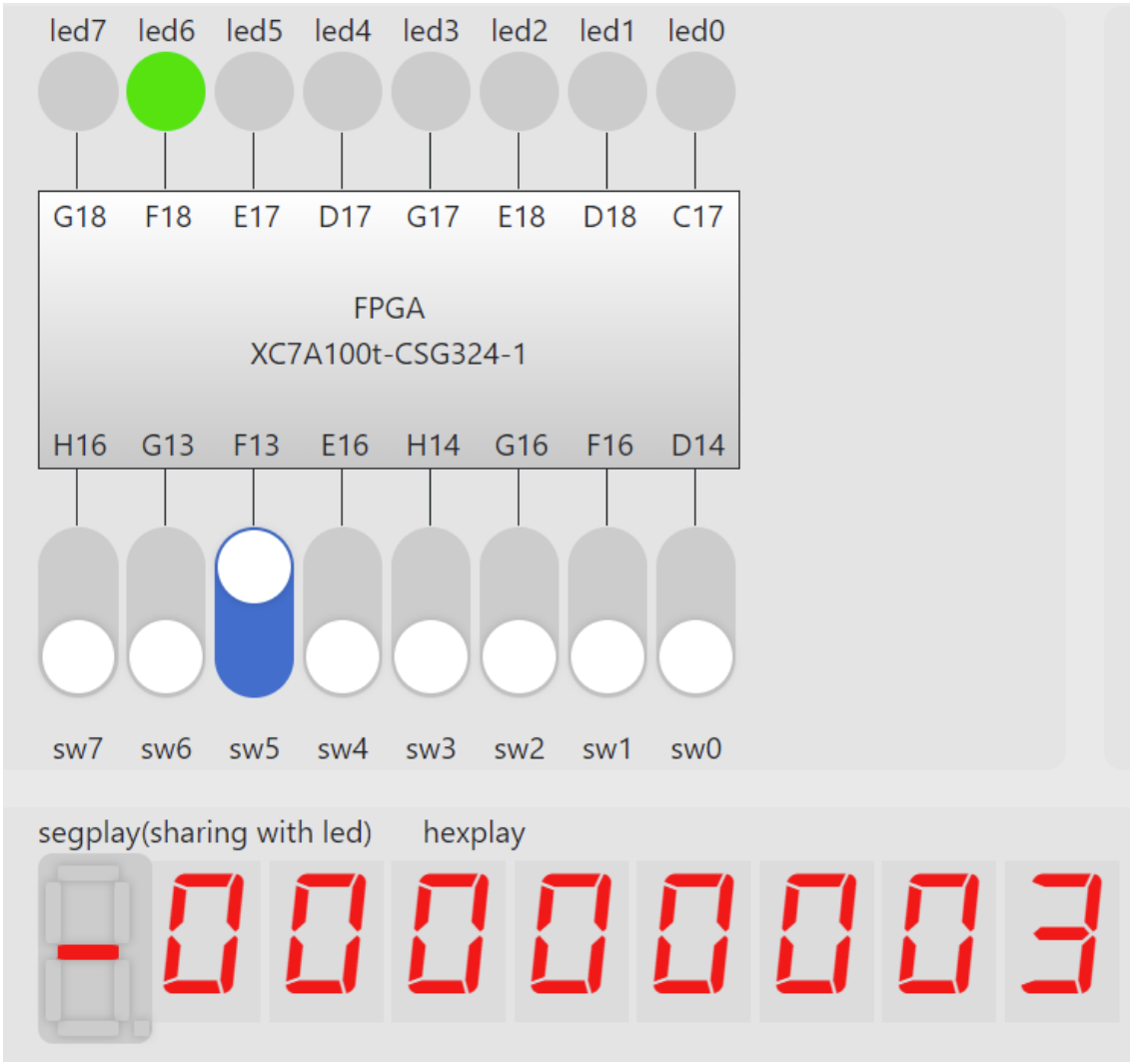
- 连续运行，输入f0

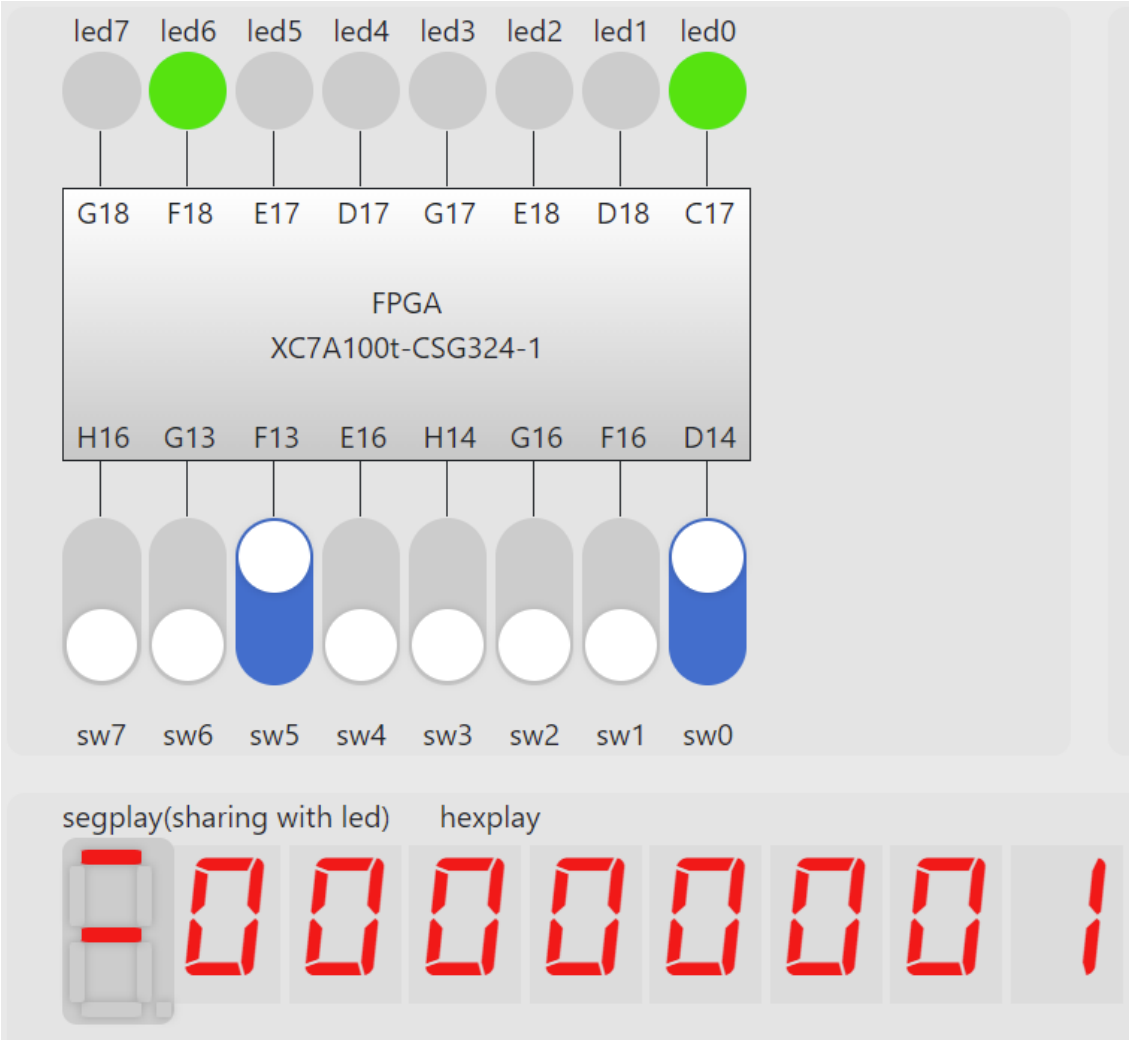


- 连续运行，输入f1

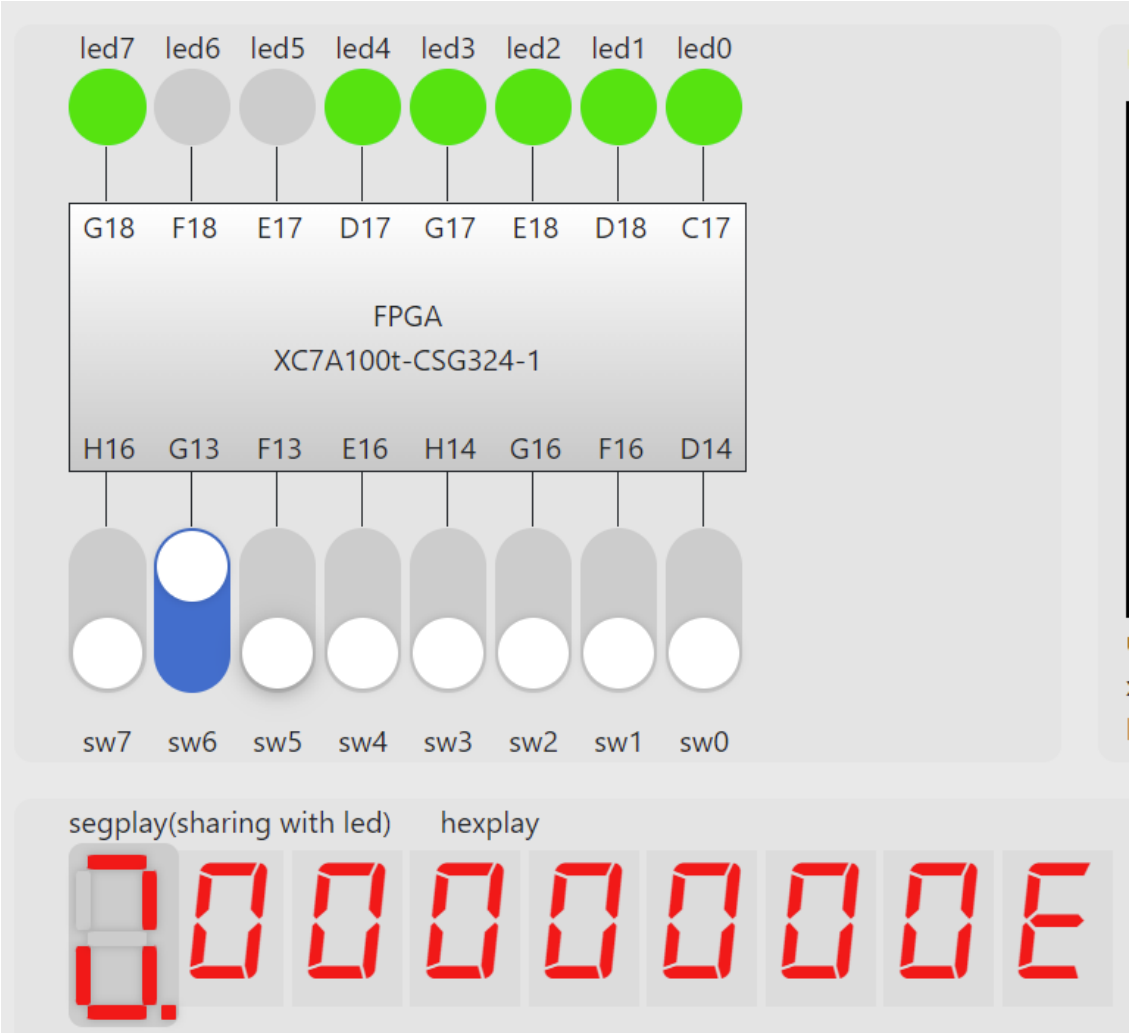

led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
0 0 0 0 0 0 0 1

- 查看存储器


led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
segplay(sharing with led)
hexplay
00000003

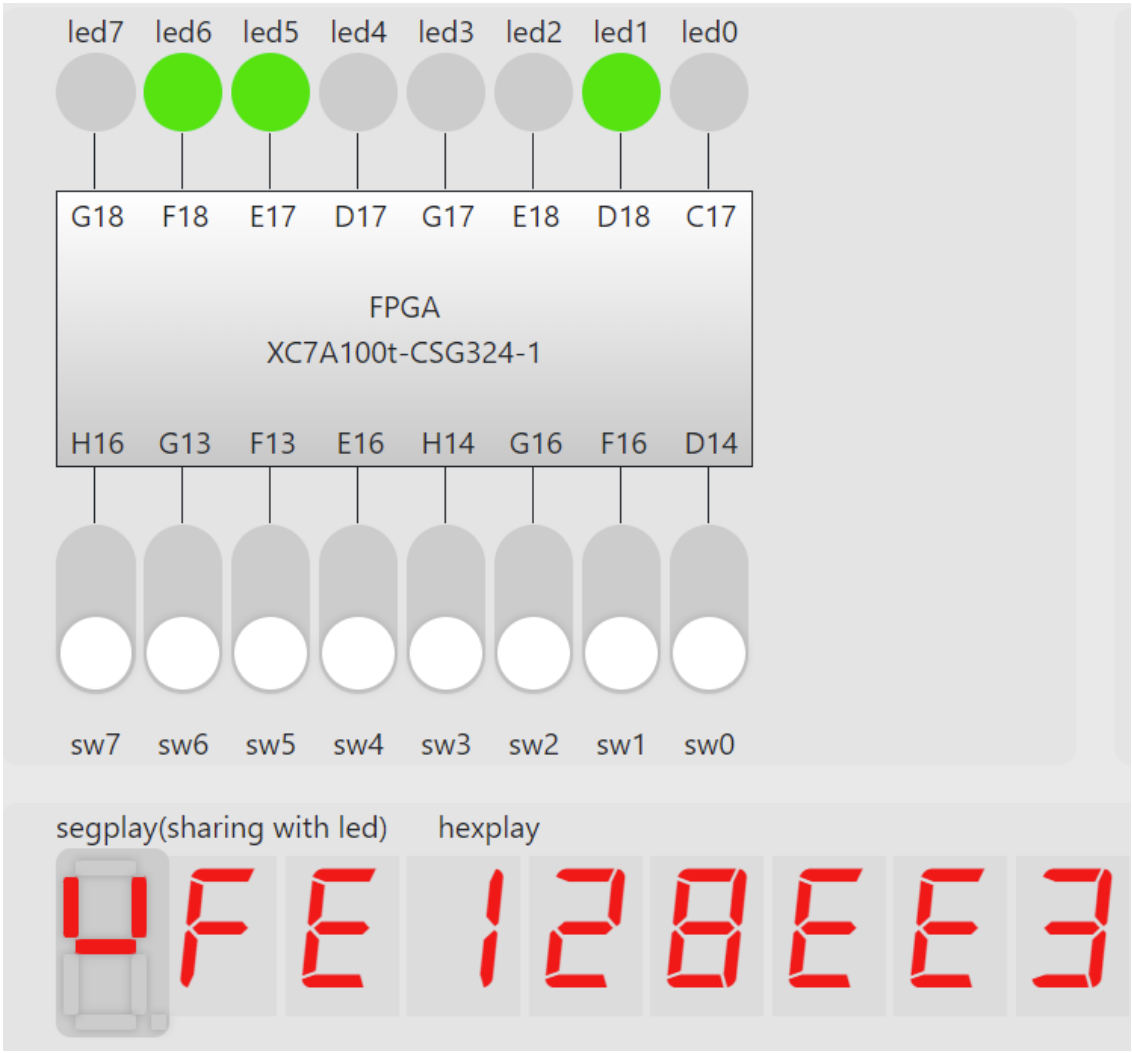


- 连续运行，按动valid进行计算




```
3 + 1 = 4; 4 + 1 = 5; 5 + 4 = 9; 9 + 5 = 14 = e
```

- beq跳转处理检查



事实上，两个周期内ir保持为此值，认为处理正确，等待跳转。

【总结与思考】

- 实验总结
  - 虽然也可以说成是和单周期一致的看图连线，但多了不少难点：
    - io的连接；
    - hazard冒险处理的判断；
    - 莫名其妙的无尽的bug……
  - 事实上，原先设计的五级流水线CPU在处理跳转指令时需要多等待一个周期，在跳转结束前会读入不该涉及的上一条指令，在仿真结果中发现虽然读入但不会产生影响，然而在实际烧写时很可能由于时钟周期与仿真相差太大导致会产生影响。除了本报告中的写法外，还可以通过在寄存器的读取出口直接进行判断取0，用这个0信号代替原本的0信号，可以不用额外等待一个时钟周期，同时还能避免产生未及时清空的问题。
  - 除此之外中规中矩。
- 实验建议

  还是那句话，实验文档真的可以详细一点，开局一张图，意思全靠猜（