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：阿非提
学生学号：PB20111633
完成日期：2022.5.18

实验目的

- 理解流水线CPU的结构和工作原理
- 掌握流水线CPU的设计和调试方法，特别是流水线中数据相关和控制相关的处理
- 熟练掌握数据通路和控制器的设计和描述方法

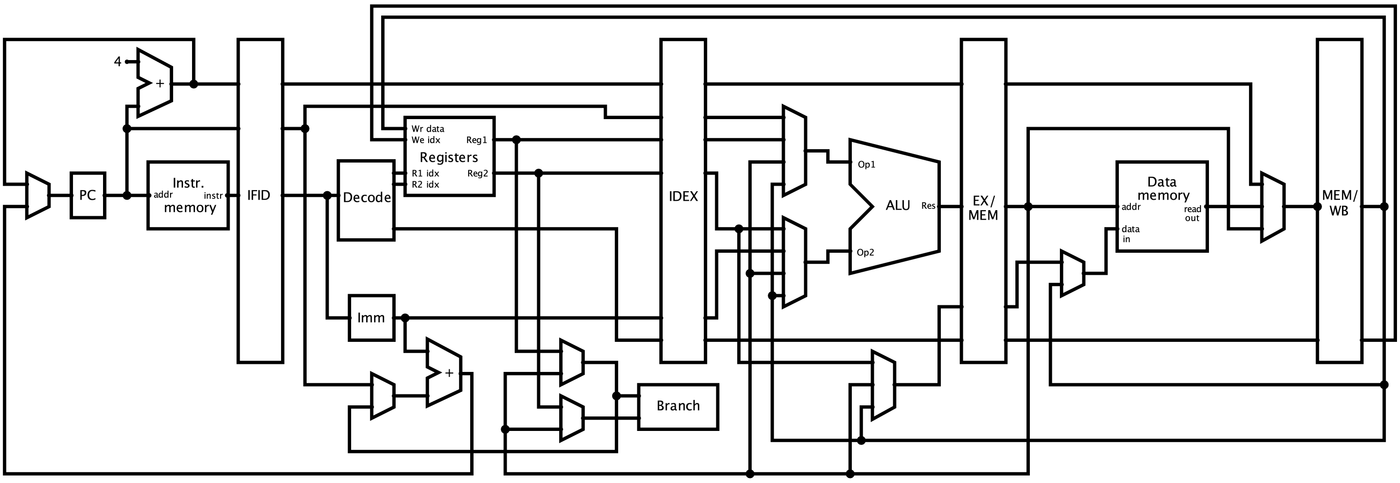
实验平台

- Vivado
- Rars

实验练习

- **cpu设计**

  在Ripes 提供的流水线处理器的数据通路的基础上，将控制冒险前移到 ID 阶段得到如下数据通路。



设计文件

寄存器堆

经过修改，寄存器堆会在写地址端口 `write_register` 不为零时进行写入操作。

```
module register_file(
    input clk,rst,
    input [4:0] read_register1,
    input [4:0] read_register2,
    input [4:0] write_register,
    input [31:0] write_data,
    output reg [31:0] read_data1,
    output reg [31:0] read_data2,
    //debug port
    input [4:0] debug_register,
    output reg [31:0] debug_register_data
    );

    reg [31:0] registers[0:31];
    integer  i;

    always@(*)begin
        if(read_register1)begin
            if(read_register1 == write_register)
                read_data1 = write_data;
            else
                read_data1 = registers[read_register1];
        end
        else
            read_data1 = 0;
    end

    always@(*)begin
        if(read_register2)begin
            if(read_register2 == write_register)
                read_data2 = write_data;
            else
                read_data2 = registers[read_register2];
        end
        else
            read_data2 = 0;
    end

    always@(*)begin
        if(debug_register)begin
            if(debug_register == write_register)
                debug_register_data = write_data;
            else
                debug_register_data = registers[debug_register];
        end
        else
            debug_register_data = 0;
    end


    always@(posedge clk or posedge rst)begin
        if(rst)
```

```
            for(i = 0; i < 32; i = i + 1)
                registers[i] <= 0;
        else if(write_register != 5'b0)
            registers[write_register] <= write_data;
    end

endmodule
```

alu

```
module alu(
    input [31:0] operant1,operant2,
    input [2:0] operation_code,
    output reg [31:0] result
    );

    wire signed [31:0] signed_operant1 = operant1;
    wire signed [31:0] signed_operant2 = operant2;

    always@(*)begin
        case(operation_code)
        3'h0: result = signed_operant1 + signed_operant2;
        3'h1: result = signed_operant1 - signed_operant2;
        3'h2: result = signed_operant1 & signed_operant2;
        3'h3: result = signed_operant1 | signed_operant2;
        3'h4: result = signed_operant1 ^ signed_operant2;
        3'h5: result = signed_operant1 << signed_operant2;
        3'h6: result = signed_operant1 >> signed_operant2;
        default: result = 0;
        endcase
    end

endmodule
```

立即数扩展

```
module immediate_Generator(
    input [31:0] inst,
    output reg [31:0] imme
);
always@(*)begin
    case(inst[6:0])
    7'b0000011: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b0010011: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b1100111: imme = {{21{inst[31]}},inst[30:20]};//I-type
    7'b0100011: imme = {{21{inst[31]}},inst[30:25],inst[11:7]};//S-type
    7'b1100011: imme = {{20{inst[31]}},inst[7],inst[30:25],inst[11:8],1'b0};//SB-type
    7'b1101111: imme = {{13{inst[31]}},inst[19:12],inst[20],inst[30:21],1'b0};//UJ-type
    7'b0010111: imme = {inst[31:12],12'h0};//U-type
    default: imme = 32'h0;
    endcase
end

endmodule
```

pc

```
module PC(
    input clk,rst,en,
    input [31:0] pc_input,
    output reg [31:0] pc
    );

    always@(posedge clk or posedge rst)begin
        if(rst)
            pc <= 32'h00003000;
        else if(en)
            pc <= pc_input;
    end
endmodule
```

decode

```
module decode(
    input [31:0] instruction,
    output [4:0] reg_addr1,
    output [4:0] reg_addr2,
    output [4:0] reg_write_addr,
    output reg [1:0] reg_write_sel,
    output mem_write,
    output reg [1:0] alu_a_sel,
    output reg [1:0] alu_b_sel,
    output reg [2:0] alu_opcode,
    output is_r1_ID_needed,
    output is_r2_ID_needed,
    output is_r1_EX_needed,
    output is_r2_EX_needed,
    output is_r2_MEM_needed,
    output is_jal,
    output is_jalr,
    output is_branch,
    output reg [1:0] branch_type
    );
    assign reg_addr1 = instruction[19:15];
    assign reg_addr2 = instruction[24:20];

    wire [6:0] opcode = instruction[6:0];
    wire [6:0] funct7 = instruction[31:25];
    wire [2:0] funct3 = instruction[14:12];

    wire is_add = (opcode == 7'b0110011) && (funct3 == 3'b000) && (funct7 ==
7'b0000000);
    wire is_addi = (opcode == 7'b0010011) && (funct3 == 3'b000);
    wire is_sub = (opcode == 7'b0110011) && (funct3 == 3'b000) && (funct7 ==
7'b0100000);
    wire is_and = (opcode == 7'b0110011) && (funct3 == 3'b111) && (funct7 ==
7'b0000000);
    wire is_andi = (opcode == 7'b0010011) && (funct3 == 3'b111);
    wire is_or = (opcode == 7'b0110011) && (funct3 == 3'b110) && (funct7 ==
7'b0000000);
    wire is_ori = (opcode == 7'b0010011) && (funct3 == 3'b110);
    wire is_xor = (opcode == 7'b0110011) && (funct3 == 3'b100) && (funct7 ==
7'b0000000);
    wire is_xori = (opcode == 7'b0010011) && (funct3 == 3'b100);
    wire is_sll = (opcode == 7'b0110011) && (funct3 == 3'b001) && (funct7 ==
7'b0000000);
    wire is_slli = (opcode == 7'b0010011) && (funct3 == 3'b001) && (funct7 ==
7'b0000000);
    wire is_srl = (opcode == 7'b0110011) && (funct3 == 3'b101) && (funct7 ==
7'b0000000);
    wire is_srli = (opcode == 7'b0010011) && (funct3 == 3'b101) && (funct7 ==
7'b0000000);
    wire is_auipc = (opcode == 7'b0010111);
    wire is_beq = (opcode == 7'b1100011) && (funct3 == 3'b000);
    wire is_bne = (opcode == 7'b1100011) && (funct3 == 3'b001);
    wire is_blt = (opcode == 7'b1100011) && (funct3 == 3'b100);
    wire is_bge = (opcode == 7'b1100011) && (funct3 == 3'b101);
    assign is_jal = (opcode == 7'b1101111);
    assign is_jalr = (opcode == 7'b1100111);
    wire is_lw = (opcode == 7'b0000011) && (funct3 == 3'b010);
    wire is_sw = (opcode == 7'b0100011) && (funct3 == 3'b010);
    wire is_regWrite = is_add | is_addi | is_sub | is_and | is_andi | is_or |
is_ori | is_xor | is_xori | is_sll | is_slli | is_srl | is_srli | is_auipc |
is_jal | is_jalr | is_lw;
    assign is_branch = is_beq | is_bge | is_blt | is_bne;
    wire is_imm = is_addi | is_andi | is_ori | is_xori | is_slli | is_srli |
is_auipc | is_sw | is_lw;
    assign mem_write = is_sw;
    assign reg_write_addr = is_regWrite ? instruction[11:7] : 5'h0;
    assign is_r1_ID_needed = is_beq | is_bge | is_blt | is_bne | is_jalr;
    assign is_r2_ID_needed = is_beq | is_bge | is_blt | is_bne;
    assign is_r1_EX_needed = is_add | is_addi | is_sub | is_and | is_andi | is_or
| is_ori | is_xor | is_xori | is_sll | is_slli | is_srl | is_srli | is_lw |
is_sw;
    assign is_r2_EX_needed = is_add | is_sub | is_and  | is_or  | is_xor  |
is_sll | is_srl | is_sw;
    assign is_r2_MEM_needed = is_sw;

    always@(*)begin
        if(is_beq)
            branch_type = 2'h0;
        else if(is_blt)
            branch_type = 2'h1;
        else if(is_bge)
            branch_type = 2'h2;
        else
            branch_type = 2'h3;
    end

    always@(*)begin
        if(is_jal | is_jalr)
            reg_write_sel = 2'h1;
        else if(is_lw)
            reg_write_sel = 2'h2;
        else
            reg_write_sel = 2'h0;
    end
    always@(*)begin
        if(is_auipc)
            alu_a_sel = 2'h1;
        else
            alu_a_sel = 2'h0;
    end
    always@(*)begin
        if(is_imm)
            alu_b_sel = 2'h1;
        else
            alu_b_sel = 2'h0;
    end
    always@(*) begin
        if(is_add | is_addi)
            alu_opcode = 3'h0;
        else if(is_sub)
            alu_opcode = 3'h1;
        else if(is_and | is_andi)
            alu_opcode = 3'h2;
        else if(is_or | is_ori)
            alu_opcode = 3'h3;
        else if(is_xor | is_xori)
            alu_opcode = 3'h4;
        else if(is_sll | is_slli)
            alu_opcode = 3'h5;
        else if(is_srl | is_srli)
            alu_opcode = 3'h6;
        else
            alu_opcode = 0;
    end

endmodule
```

hazard

```
module hazerd_detection(
    input [4:0] ID_EX_reg_w_addr,
    input [4:0] EX_MEM_reg_w_addr,
    input [1:0] ID_EX_reg_write_sel,
    input [1:0] EX_MEM_reg_write_sel,
    input [4:0] reg_addr1,
    input [4:0] reg_addr2,
    input is_r1_ID_needed,
    input is_r2_ID_needed,
    input is_r1_EX_needed,
    input is_r2_EX_needed,
    input is_r2_MEM_needed,
    input [1:0] alu_a_sel,
    input [1:0] alu_b_sel,
    input is_branch,
    input is_jump,
    input branch_signal,
    input [31:0] reg_data1,
    input [31:0] reg_data2,
    input [31:0] EX_MEM_alu_result,
    output reg [1:0]  ID_EX_alu_a_sel,
    output reg [1:0]  ID_EX_alu_b_sel,
    output reg [1:0]  ID_EX_sw_r2_sel,
    output [31:0] branch_input1,
    output [31:0] branch_input2,
    output [31:0] jalr_reg,
    output ID_EX_mem_write_sel,
    output stall,
    output flush
    );

    wire is_r1_EX_dependent = (ID_EX_reg_w_addr != 0) && (reg_addr1 ==
ID_EX_reg_w_addr);
    wire is_r2_EX_dependent = (ID_EX_reg_w_addr != 0) && (reg_addr2 ==
ID_EX_reg_w_addr);
```

```
    wire is_r1_MEM_dependent = (EX_MEM_reg_w_addr != 0) && (reg_addr1 ==
EX_MEM_reg_w_addr);
    wire is_r2_MEM_dependent = (EX_MEM_reg_w_addr != 0) && (reg_addr2 ==
EX_MEM_reg_w_addr);
    wire ID_stall_by_EX = (is_r1_ID_needed && is_r1_EX_dependent) ||
(is_r2_ID_needed && is_r2_EX_dependent);
    wire ID_stall_by_MEM = ((is_r1_ID_needed && is_r1_MEM_dependent) ||
(is_r2_ID_needed && is_r2_MEM_dependent)) && (EX_MEM_reg_write_sel == 2'h2);
    wire EX_stall_by_MEM = ((is_r1_EX_needed && is_r1_EX_dependent) ||
(is_r2_EX_needed && is_r2_EX_dependent && (~is_r2_MEM_needed))) &&
(ID_EX_reg_write_sel == 2'h2);
    wire r1_ID_need_forward = is_r1_ID_needed && is_r1_MEM_dependent &&
(EX_MEM_reg_write_sel != 2'h2);
    wire r2_ID_need_forward = is_r2_ID_needed && is_r2_MEM_dependent &&
(EX_MEM_reg_write_sel != 2'h2);
    wire r1_EX_need_forward_EX_MEM = is_r1_EX_needed && is_r1_EX_dependent &&
(ID_EX_reg_write_sel != 2'h2);
    wire r2_EX_need_forward_EX_MEM = is_r2_EX_needed && is_r2_EX_dependent &&
(ID_EX_reg_write_sel != 2'h2);
    wire r1_EX_need_forward_MEM_WB = is_r1_EX_needed && is_r1_MEM_dependent;
    wire r2_EX_need_forward_MEM_WB = is_r2_EX_needed && is_r2_MEM_dependent;
    wire r2_MEM_need_forward = is_r2_MEM_needed && is_r2_EX_dependent &&
(ID_EX_reg_write_sel == 2'h2);
    assign stall = ID_stall_by_EX | ID_stall_by_MEM | EX_stall_by_MEM;
    assign flush = ~(ID_stall_by_EX | ID_stall_by_MEM ) & ((is_branch &
branch_signal) | is_jump);
    assign branch_input1 = r1_ID_need_forward ? EX_MEM_alu_result : reg_data1;
    assign branch_input2 = r2_ID_need_forward ? EX_MEM_alu_result : reg_data2;
    assign jalr_reg = r1_ID_need_forward ? EX_MEM_alu_result : reg_data1;
    assign ID_EX_mem_write_sel = r2_MEM_need_forward;

    always@(*)begin
        if(r1_EX_need_forward_EX_MEM)
            ID_EX_alu_a_sel = 2'h2;
        else if(r1_EX_need_forward_MEM_WB)
            ID_EX_alu_a_sel = 2'h3;
        else
            ID_EX_alu_a_sel = alu_a_sel;
    end

    always@(*)begin
        if(~is_r2_MEM_needed & r2_EX_need_forward_EX_MEM)
            ID_EX_alu_b_sel = 2'h2;
        else if(~is_r2_MEM_needed & r2_EX_need_forward_MEM_WB)
            ID_EX_alu_b_sel = 2'h3;
        else
            ID_EX_alu_b_sel = alu_b_sel;
    end
    always@(*)begin
        if(r2_EX_need_forward_EX_MEM)
            ID_EX_sw_r2_sel <= 2'h1;
        else if(r2_EX_need_forward_MEM_WB)
            ID_EX_sw_r2_sel <= 2'h2;
        else
            ID_EX_sw_r2_sel <= 2'h0;
    end

endmodule
```

branch

```
module branch(
    input [1:0] branch_type, //0: beq, 1: blt
    input [31:0] input1,
    input [31:0] input2,
    output reg branch_signal
    );
    wire signed [31:0] signed_input1 = input1;
    wire signed [31:0] signed_input2 = input2;

    always@(*)begin
        case(branch_type)
        2'h0:branch_signal = (signed_input1 == signed_input2)? 1:0;
        2'h1:branch_signal = (signed_input1 < signed_input2)? 1:0;
        2'h2:branch_signal = (signed_input1 >= signed_input2)? 1:0;
        2'h3:branch_signal = (signed_input1 != signed_input2)? 1:0;
        default:;
        endcase
    end
endmodule
```

data memory

```
module data_io_memory(
    input clk,
    input write_enable,
    input [31:0] addr,
    input [31:0] write_data,
    output [31:0] read_data,
    //io memory ports
    output io_we,
    output [7:0] io_addr,
    output [31:0] io_dout,
    input [31:0] io_din,
    //debug ports
    input [7:0] m_rf_addr,
    output [31:0] m_data
    );

    // is address in 0x0000_0000 ~ 0x0000_03ff range
    wire is_data_ram = (addr[31:10] == 22'b0)? 1:0;

    //if address is I/O memory address
    assign io_we = (is_data_ram == 0)? write_enable : 0;
    assign io_dout = write_data;
    assign io_addr = addr[7:0];

    //if address is memory address
    wire data_ram_write = (is_data_ram == 1)? write_enable : 0;
    wire [31:0] data_ram_data;

    assign read_data = (is_data_ram == 1)? data_ram_data : io_din;

    data_mem data_mem(
        .clk(clk),
        .a(addr[9:2]),
        .d(write_data),
        .we(data_ram_write),
        .spo(data_ram_data),
        //debug port
        .dpra(m_rf_addr),
        .dpo(m_data)
    );

endmodule
```

cpu

```
module cpu(
    input clk,rst,
    //IO_BUS
    output [7:0] io_addr,
    output [31:0] io_dout,
    output io_we,
    input [31:0] io_din,
    //Debug_BUS
    input [7:0] m_rf_addr,
    output [31:0] rf_data,
    output [31:0] m_data,
    //PC/IF/ID register
    output [31:0] pc,
    output [31:0] pcd, //pc_id
    output [31:0] ir,  //inst_id
    output [31:0] pcin,
    //ID/EX register
    output [31:0] pce, //pc_ex
    output [31:0] a,   //rf_rdata1_ex
    output [31:0] b,   //rf_rdata2_ex
    output [31:0] imm,
    output [4:0] rd,
    output [31:0] ctrl,
    output [31:0] y,
    output [31:0] bm,
    output [4:0] rdm,
    output [31:0] ctrlm,
    //MEM/WB register
    output reg [31:0] yw,
    output reg [31:0] mdr,
    output [4:0] rdw,
    output [31:0] ctrlw
);

//IF/ID registers
reg [31:0] IF_ID_pc;
reg [31:0] IF_ID_pc_plus4;
reg [31:0] IF_ID_instruction;

//ID/EX registers
reg [31:0] ID_EX_pc;
reg [31:0] ID_EX_pc_plus4;
reg [31:0] ID_EX_reg1;
reg [31:0] ID_EX_reg2;
reg [31:0] ID_EX_imm;
reg [4:0]  ID_EX_reg_w_addr;
reg [1:0]  ID_EX_reg_write_sel; // 0: alu result, 1: pc+4, 2: mem
reg [1:0]  ID_EX_alu_a_sel; // 0: reg1, 1: pc, 2: EX/MEM, 3: MEM/WB
reg [1:0]  ID_EX_alu_b_sel; // 0: reg2, 1: imm, 2: EX/MEM, 3: MEM/WB
reg [2:0]  ID_EX_alu_opcode; //0: add, 1: sub, 2: and, 3: or, 4: xor, 5: shift left, 6: shift right
reg        ID_EX_mem_write_sel; //0: data, 1: MEM/WB
reg        ID_EX_mem_write;
reg [1:0]  ID_EX_sw_r2_sel;
wire stall;
wire flush;


//EX/MEM registers
reg [31:0] EX_MEM_pc_plus4;
reg [31:0] EX_MEM_alu_result;
reg [31:0] EX_MEM_data_mem_data;
reg [4:0]  EX_MEM_reg_w_addr;
reg [1:0]  EX_MEM_reg_write_sel; // 0: alu result, 1: pc+4, 2: mem
reg        EX_MEM_mem_write_sel; //0: data, 1: MEM/WB
reg        EX_MEM_mem_write;

//MEM/WB registers
reg [31:0] MEM_WB_reg_w_data;
reg [4:0]  MEM_WB_reg_w_addr;


//IF stage

    //pc
    reg [31:0] pc_input;
    PC PC(
        .clk(clk),
        .rst(rst),
        .en(~stall),
        .pc_input(pc_input),
        .pc(pc)
    );

    //instruction memory
    wire [31:0] instruction;
    instruction_mem instruction_mem(
        .a(pc[9:2]),
        .spo(instruction)
    );

    always@(posedge clk or posedge rst)begin
        if(rst | flush)begin
            IF_ID_pc <= 0;
            IF_ID_pc_plus4 <= 0;
            IF_ID_instruction <= 0;
        end
        else if(~stall)begin
            IF_ID_pc <= pc;
            IF_ID_pc_plus4 <= pc + 4;
            IF_ID_instruction <= instruction;
        end
    end

//ID stage

    //register file
    wire [4:0] reg_addr1;
    wire [4:0] reg_addr2;
    wire [31:0] reg_data1;
    wire [31:0] reg_data2;
    register_file register_file(
        .clk(clk),
```

```verilog
    .rst(rst),
    .read_register1(reg_addr1),
    .read_register2(reg_addr2),
    .write_register(MEM_WB_reg_w_addr),
    .write_data(MEM_WB_reg_w_data),
    .read_data1(reg_data1),
    .read_data2(reg_data2),
    .debug_register(m_rf_addr[4:0]),
    .debug_register_data(rf_data)
);

//imm
wire [31:0] Imm;
imme imme(.inst(IF_ID_instruction),.imme(Imm));

//branch
wire [1:0] branch_type;
wire [31:0] branch_input1;
wire [31:0] branch_input2;
wire branch_signal;
branch branch(
    .branch_type(branch_type),
    .input1(branch_input1),
    .input2(branch_input2),
    .branch_signal(branch_signal)
);

//decode
wire [4:0] reg_write_addr;
wire [1:0] reg_write_sel;
wire mem_write;
wire [1:0] alu_a_sel;
wire [1:0] alu_b_sel;
wire [2:0] alu_opcode;
wire is_r1_ID_needed;
wire is_r2_ID_needed;
wire is_r1_EX_needed;
wire is_r2_EX_needed;
wire is_r2_MEM_needed;
wire is_jal;
wire is_jalr;
wire is_branch;

decode decode(
    .instruction(IF_ID_instruction),
    .reg_addr1(reg_addr1),
    .reg_addr2(reg_addr2),
    .reg_write_addr(reg_write_addr),
    .reg_write_sel(reg_write_sel),
    .mem_write(mem_write),
    .alu_a_sel(alu_a_sel),
    .alu_b_sel(alu_b_sel),
    .alu_opcode(alu_opcode),
    .is_r1_ID_needed(is_r1_ID_needed),
    .is_r2_ID_needed(is_r2_ID_needed),
    .is_r1_EX_needed(is_r1_EX_needed),
    .is_r2_EX_needed(is_r2_EX_needed),
    .is_r2_MEM_needed(is_r2_MEM_needed),
    .is_jal(is_jal),
    .is_jalr(is_jalr),
    .is_branch(is_branch),
    .branch_type(branch_type)
);

//hazerds detection
wire [1:0] id_ex_alu_a_sel;
wire [1:0] id_ex_alu_b_sel;
wire [1:0] id_ex_sw_r2_sel;
wire id_ex_mem_write_sel;
wire [31:0] jalr_reg;
hazerd_detection hazerd_detection(
    .ID_EX_reg_w_addr(ID_EX_reg_w_addr),
    .EX_MEM_reg_w_addr(EX_MEM_reg_w_addr),
    .ID_EX_reg_write_sel(ID_EX_reg_write_sel),
    .EX_MEM_reg_write_sel(EX_MEM_reg_write_sel),
    .reg_addr1(reg_addr1),
    .reg_addr2(reg_addr2),
    .is_r1_ID_needed(is_r1_ID_needed),
    .is_r2_ID_needed(is_r2_ID_needed),
    .is_r1_EX_needed(is_r1_EX_needed),
    .is_r2_EX_needed(is_r2_EX_needed),
    .is_r2_MEM_needed(is_r2_MEM_needed),
    .alu_a_sel(alu_a_sel),
    .alu_b_sel(alu_b_sel),
    .is_branch(is_branch),
    .is_jump(is_jal | is_jalr),
    .branch_signal(branch_signal),
    .reg_data1(reg_data1),
    .reg_data2(reg_data2),
    .EX_MEM_alu_result(EX_MEM_alu_result),
    .ID_EX_alu_a_sel(id_ex_alu_a_sel),
    .ID_EX_alu_b_sel(id_ex_alu_b_sel),
    .ID_EX_sw_r2_sel(id_ex_sw_r2_sel),
    .branch_input1(branch_input1),
    .branch_input2(branch_input2),
    .jalr_reg(jalr_reg),
    .ID_EX_mem_write_sel(id_ex_mem_write_sel),
    .stall(stall),
            .flush(flush)
        );

        //pc

        always@(*)begin
            if((is_branch & branch_signal) | is_jal)
                pc_input = IF_ID_pc + Imm;
            else if(is_jalr)
                pc_input = jalr_reg + Imm;
            else
                pc_input = pc + 4;
        end

        //ID->EX
        always@(posedge clk or posedge rst)begin
            if(rst | stall)begin
                ID_EX_pc <= 0;
                ID_EX_pc_plus4 <= 0;
                ID_EX_reg1 <= 0;
                ID_EX_reg2 <= 0;
                ID_EX_imm <= 0;
                ID_EX_reg_w_addr <= 0;
                ID_EX_reg_write_sel <= 0;
                ID_EX_alu_a_sel <= 0;
                ID_EX_alu_b_sel <= 0;
                ID_EX_alu_opcode <= 0;
                ID_EX_mem_write_sel <= 0;
                ID_EX_mem_write <= 0;
                ID_EX_sw_r2_sel <= 0;
            end
            else begin
                ID_EX_pc <= IF_ID_pc;
                ID_EX_pc_plus4 <= IF_ID_pc_plus4;
                ID_EX_reg1 <= reg_data1;
                ID_EX_reg2 <= reg_data2;
                ID_EX_imm <= Imm;
                ID_EX_reg_w_addr <= reg_write_addr;
                ID_EX_reg_write_sel <= reg_write_sel;
                ID_EX_alu_a_sel <= id_ex_alu_a_sel;
                ID_EX_alu_b_sel <= id_ex_alu_b_sel;
                ID_EX_alu_opcode <= alu_opcode;
                ID_EX_mem_write_sel <= id_ex_mem_write_sel;
                ID_EX_mem_write <= mem_write;
                ID_EX_sw_r2_sel <= id_ex_sw_r2_sel;
            end
        end

    //EX stage
```

```
//alu
reg [31:0] operant1;
reg [31:0] operant2;
wire [31:0] alu_result;
alu alu(
    .operant1(operant1),
    .operant2(operant2),
    .operation_code(ID_EX_alu_opcode),
    .result(alu_result)
);

always@(*)begin
    case(ID_EX_alu_a_sel)
    2'h0:operant1 = ID_EX_reg1;
    2'h1:operant1 = ID_EX_pc;
    2'h2:operant1 = EX_MEM_alu_result;
    2'h3:operant1 = MEM_WB_reg_w_data;
    endcase
end

always@(*)begin
    case(ID_EX_alu_b_sel)
    2'h0:operant2 = ID_EX_reg2;
    2'h1:operant2 = ID_EX_imm;
    2'h2:operant2 = EX_MEM_alu_result;
    2'h3:operant2 = MEM_WB_reg_w_data;
    endcase
end

//EX->MEM
always@(posedge clk or posedge rst)begin
    if(rst)begin
        EX_MEM_pc_plus4 <= 0;
        EX_MEM_alu_result <= 0;
        EX_MEM_data_mem_data <= 0;
        EX_MEM_reg_w_addr <= 0;
        EX_MEM_reg_write_sel <= 0;
        EX_MEM_mem_write_sel <= 0;
        EX_MEM_mem_write <= 0;
    end
    else begin
        EX_MEM_pc_plus4 <= ID_EX_pc_plus4;
        EX_MEM_alu_result <= alu_result;
        case(ID_EX_sw_r2_sel)
        2'h0:EX_MEM_data_mem_data <= ID_EX_reg2;
        2'h1:EX_MEM_data_mem_data <= EX_MEM_alu_result;
        2'h2:EX_MEM_data_mem_data <= MEM_WB_reg_w_data;
        default:;
```

```
            endcase
            EX_MEM_reg_w_addr <= ID_EX_reg_w_addr;
            EX_MEM_reg_write_sel <= ID_EX_reg_write_sel;
            EX_MEM_mem_write_sel <= ID_EX_mem_write_sel;
            EX_MEM_mem_write <= ID_EX_mem_write;
        end
    end

//MEM stage

    //data memory
    wire [31:0] write_data = EX_MEM_mem_write_sel ? MEM_WB_reg_w_data :
EX_MEM_data_mem_data;
    wire [31:0] read_data;
    data_io_memory data_io_memory(
        .clk(clk),
        .write_enable(EX_MEM_mem_write),
        .addr(EX_MEM_alu_result),
        .write_data(write_data),
        .read_data(read_data),
        .io_we(io_we),
        .io_addr(io_addr),
        .io_dout(io_dout),
        .io_din(io_din),
        .m_rf_addr(m_rf_addr),
        .m_data(m_data)
    );

    always@(posedge clk or posedge rst)begin
        if(rst)begin
            MEM_WB_reg_w_data <= 0;
            MEM_WB_reg_w_addr <= 0;
        end
        else begin
            case(EX_MEM_reg_write_sel)
            2'h0:MEM_WB_reg_w_data <= EX_MEM_alu_result;
            2'h1:MEM_WB_reg_w_data <= EX_MEM_pc_plus4;
            2'h2:MEM_WB_reg_w_data <= read_data;
            default:MEM_WB_reg_w_data <= 0;
            endcase
            MEM_WB_reg_w_addr <= EX_MEM_reg_w_addr;
        end
    end

//output modules
    assign pcd = IF_ID_pc;
    assign ir = IF_ID_instruction;
    assign pcin = pc_input;
    assign pce = ID_EX_pc;
    assign a = ID_EX_reg1;
    assign b = ID_EX_reg2;
    assign imm = ID_EX_imm;
    assign rd = ID_EX_reg_w_addr;
    assign ctrl = {stall,stall,(~stall &
flush),flush,2'b0,ID_EX_alu_a_sel,2'b0,ID_EX_alu_b_sel,1'b0,(ID_EX_reg_w_addr !=
5'h0),ID_EX_reg_write_sel,2'b0,(ID_EX_reg_write_sel ==
2'h2),ID_EX_mem_write,2'b0,(is_jal | is_jalr),(is_branch &
branch_signal),2'b0,alu_a_sel[0],alu_b_sel[0],ID_EX_alu_opcode};
    assign y = EX_MEM_alu_result;
    assign bm = EX_MEM_mem_write_sel ? MEM_WB_reg_w_data : EX_MEM_data_mem_data;
    assign rdm = EX_MEM_reg_w_addr;
    assign ctrlm =
{26'b0,EX_MEM_reg_write_sel,2'b0,EX_MEM_mem_write,EX_MEM_mem_write_sel};
    always@(posedge clk or posedge rst)begin
        if(rst)begin
            yw <= 0;
            mdr <= 0;
        end
        else begin
            yw <= (EX_MEM_reg_write_sel == 2'h0) ? EX_MEM_alu_result :
EX_MEM_pc_plus4;
            mdr <= read_data;
        end
    end
    assign rdw = MEM_WB_reg_w_addr;
    assign ctrlw = {31'b0,(MEM_WB_reg_w_addr != 5'h0)};


endmodule
```

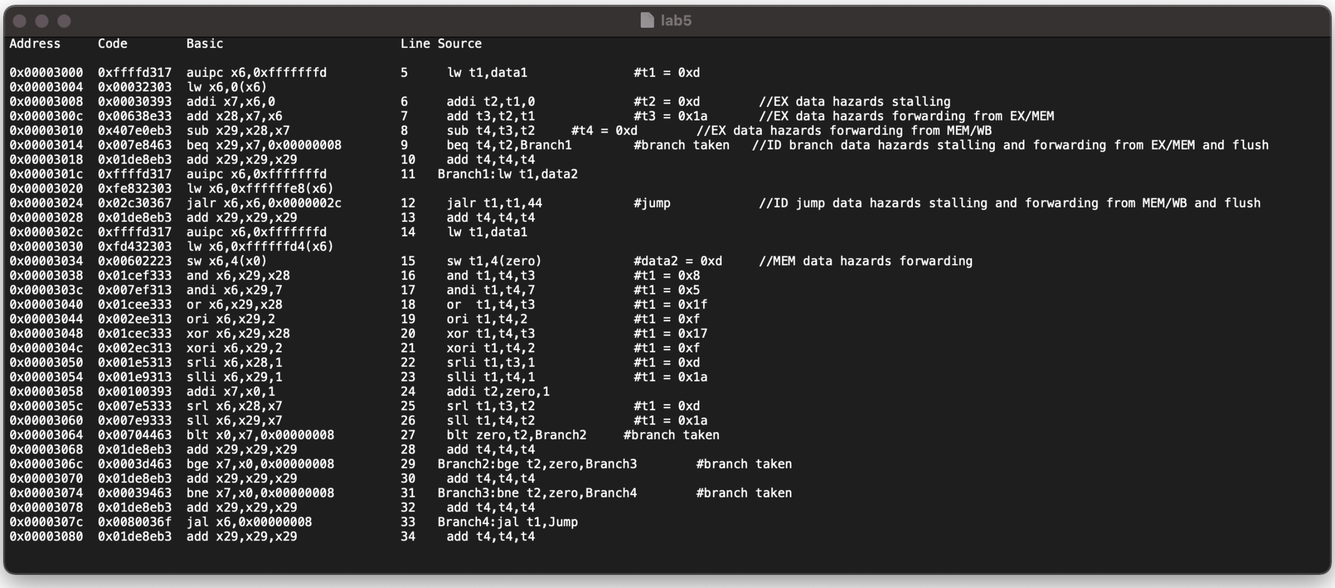
- **cpu功能仿真**

运行使用的汇编代码

```
.data
    data1: .word 0xd
    data2: .word 0x3000
.text
    lw t1,data1      #t1 = 0xd
    addi t2,t1,0     #t2 = 0xd   //EX data hazards stalling
    add t3,t2,t1     #t3 = 0x1a  //EX data hazards forwarding from EX/MEM
    sub t4,t3,t2     #t4 = 0xd   //EX data hazards forwarding from MEM/WB

    beq t4,t2,Branch1    #branch taken   //ID branch data hazards stalling and
forwarding from EX/MEM and flush
```

```
    add t4,t4,t4

Branch1:lw t1,data2
    jalr t1,t1,44       #t1 = 0x3028    //ID jump data hazards stalling and forwarding from MEM/WB and flush

    add t4,t4,t4
    lw t1,data1
    sw t1,4(zero).     #data2 = 0xd    //MEM data hazards forwarding

    and t1,t4,t3        #t1 = 0x8
    andi t1,t4,7        #t1 = 0x5
    or  t1,t4,t3        #t1 = 0x1f
    ori t1,t4,2         #t1 = 0xf
    xor t1,t4,t3        #t1 = 0x17
    xori t1,t4,2        #t1 = 0xf
    srli t1,t3,1        #t1 = 0xd
    slli t1,t4,1        #t1 = 0x1a
    addi t2,zero,1
    srl t1,t3,t2        #t1 = 0xd
    sll t1,t4,t2        #t1 = 0x1a
    blt zero,t2,Branch2     #branch taken
    add t4,t4,t4
Branch2:bge t2,zero,Branch3     #branch taken
    add t4,t4,t4
Branch3:bne t2,zero,Branch4     #branch taken
    add t4,t4,t4
Branch4:jal t1,Jump
    add t4,t4,t4
Jump:
```

rars 生成的代码



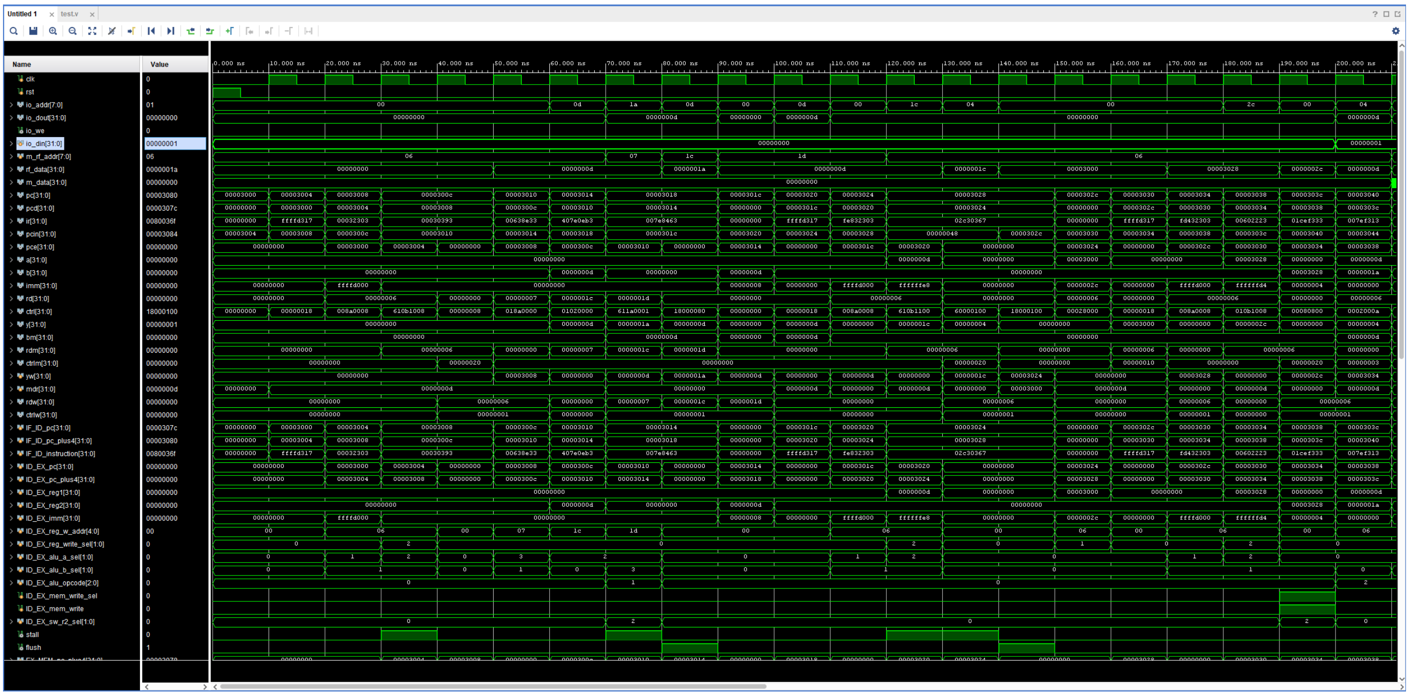
```
Address     Code        Basic                        Line Source

0x00003000  0xffffd317  auipc x6,0xfffffffd          5     lw t1,data1              #t1 = 0xd
0x00003004  0x00032303  lw x6,0(x6)
0x00003008  0x00030393  addi x7,x6,0                 6     addi t2,t1,0             #t2 = 0xd       //EX data hazards stalling
0x0000300c  0x00638e33  add x28,x7,x6                7     add t3,t2,t1             #t3 = 0x1a      //EX data hazards forwarding from EX/MEM
0x00003010  0x407e0eb3  sub x29,x28,x7               8     sub t4,t3,t2     #t4 = 0xd       //EX data hazards forwarding from MEM/WB
0x00003014  0x007e8463  beq x29,x7,0x00000008        9     beq t4,t2,Branch1        #branch taken   //ID branch data hazards stalling and forwarding from EX/MEM and flush
0x00003018  0x01de8eb3  add x29,x29,x29              10    add t4,t4,t4
0x0000301c  0xffffd317  auipc x6,0xfffffffd          11   Branch1:lw t1,data2
0x00003020  0xfe832303  lw x6,0xffffffe8(x6)
0x00003024  0x02c30367  jalr x6,x6,0x0000002c        12    jalr t1,t1,44            #jump           //ID jump data hazards stalling and forwarding from MEM/WB and flush
0x00003028  0x01de8eb3  add x29,x29,x29              13    add t4,t4,t4
0x0000302c  0xffffd317  auipc x6,0xfffffffd          14    lw t1,data1
0x00003030  0xfd432303  lw x6,0xffffffd4(x6)
0x00003034  0x00602223  sw x6,4(x0)                  15    sw t1,4(zero)            #data2 = 0xd    //MEM data hazards forwarding
0x00003038  0x01cef333  and x6,x29,x28               16    and t1,t4,t3             #t1 = 0x8
0x0000303c  0x007ef313  andi x6,x29,7                17    andi t1,t4,7             #t1 = 0x5
0x00003040  0x01cee333  or x6,x29,x28                18    or  t1,t4,t3             #t1 = 0x1f
0x00003044  0x002ee313  ori x6,x29,2                 19    ori t1,t4,2              #t1 = 0xf
0x00003048  0x01cec333  xor x6,x29,x28               20    xor t1,t4,t3             #t1 = 0x17
0x0000304c  0x002ec313  xori x6,x29,2                21    xori t1,t4,2             #t1 = 0xf
0x00003050  0x001e5313  srli x6,x28,1                22    srli t1,t3,1             #t1 = 0xd
0x00003054  0x001e9313  slli x6,x29,1                23    slli t1,t4,1             #t1 = 0x1a
0x00003058  0x00100393  addi x7,x0,1                 24    addi t2,zero,1
0x0000305c  0x007e5333  srl x6,x28,x7                25    srl t1,t3,t2             #t1 = 0xd
0x00003060  0x007e9333  sll x6,x29,x7                26    sll t1,t4,t2             #t1 = 0x1a
0x00003064  0x00704463  blt x0,x7,0x00000008         27    blt zero,t2,Branch2     #branch taken
0x00003068  0x01de8eb3  add x29,x29,x29              28    add t4,t4,t4
0x0000306c  0x0003d463  bge x7,x0,0x00000008         29   Branch2:bge t2,zero,Branch3       #branch taken
0x00003070  0x01de8eb3  add x29,x29,x29              30    add t4,t4,t4
0x00003074  0x00039463  bne x7,x0,0x00000008         31   Branch3:bne t2,zero,Branch4       #branch taken
0x00003078  0x01de8eb3  add x29,x29,x29              32    add t4,t4,t4
0x0000307c  0x0080036f  jal x6,0x00000008            33   Branch4:jal t1,Jump
0x00003080  0x01de8eb3  add x29,x29,x29              34    add t4,t4,t4
```

流水线执行图

Lab5 pipeline graph

		10	20	30	40	50	60	70	80	90	100	110	120	130	140	150	160	170	180	190	200	210	220	230	240	250	260	270	280	290	300	310	320	330	340	350	360	370	380	390	400	410
3000	auipc x6,0xfffffffd	IF	ID	EX	MEM	WB																																				
3004	lw x6,0(x6)		IF	ID	EX	MEM	WB																																			
3008	addi x7,x6,0			IF	ID	*	EX	MEM	WB																																	
300C	add x28,x7,x6				IF	*	ID	EX	MEM	WB																																
3010	sub x29,x28,x7						IF	ID	EX	MEM	WB																															
3014	beq x29,x7,0x00000008							IF	*	ID	EX	MEM	WB																													
3018	add x29,x29,x29									IF	NOP	NOP	NOP	NOP																												
301C	auipc x6,0xfffffffd										IF	ID	EX	MEM	WB																											
3020	lw x6,0xffffffe8(x6)											IF	ID	EX	MEM	WB																										
3024	jalr x6,x6,0x0000002c												IF	*	*	ID	EX	MEM	WB																							
3028	add x29,x29,x29															IF	NOP	NOP	NOP	NOP																						
302C	auipc x6,0xfffffffd																IF	ID	EX	MEM	WB																					
3030	lw x6,0xffffffd4(x6)																	IF	ID	EX	MEM	WB																				
3034	sw x6,4(x0)																		IF	ID	EX	MEM	WB																			
3038	and x6,x29,x28																			IF	ID	EX	MEM	WB																		
303C	andi x6,x29,7																				IF	ID	EX	MEM	WB																	
3040	or x6,x29,x28																					IF	ID	EX	MEM	WB																
3044	ori x6,x29,2																						IF	ID	EX	MEM	WB															
3048	xor x6,x29,x28																							IF	ID	EX	MEM	WB														
304C	xori x6,x29,2																								IF	ID	EX	MEM	WB													
3050	srli x6,x28,1																									IF	ID	EX	MEM	WB												
3054	slli x6,x29,1																										IF	ID	EX	MEM	WB											
3058	addi x7,x0,1																											IF	ID	EX	MEM	WB										
305C	srl x6,x28,x7																												IF	ID	EX	MEM	WB									
3060	sll x6,x29,x7																													IF	ID	EX	MEM	WB								
3064	blt x0,x7,0x00000008																														IF	ID	EX	MEM	WB							
3068	add x29,x29,x29																															IF	NOP	NOP	NOP	NOP						
306C	bge x7,x0,0x00000008																																IF	ID	EX	MEM	WB					
3070	add x29,x29,x29																																	IF	NOP	NOP	NOP	NOP				
3074	bne x7,x0,0x00000008																																		IF	ID	EX	MEM	WB			
3078	add x29,x29,x29																																			IF	NOP	NOP	NOP	NOP		
307C	jal x6,0x00000008																																				IF	ID	EX	MEM	WB	
3080	add x29,x29,x29																																					IF	NOP	NOP	NOP	NOP

仿真结果



Name	Value
clk	0
rst	0
io_addr[7:0]	01
io_dout[31:0]	00000000
io_we	0
io_din[31:0]	00000001
m_rf_addr[7:0]	06
rf_data[31:0]	0000001a
m_data[31:0]	00000000
pc[31:0]	00003080
pcd[31:0]	0000307c
ir[31:0]	0080036f
pcin[31:0]	00003084
pce[31:0]	00000000
a[31:0]	00000000
b[31:0]	00000000
imm[31:0]	00000000
rd[31:0]	00000000
ctrl[31:0]	18000100
y[31:0]	00000001
bm[31:0]	00000000
rdm[31:0]	00000000
ctrlm[31:0]	00000000
yw[31:0]	00000000
mdr[31:0]	0000000d
rdw[31:0]	00000000
ctrlw[31:0]	00000000
IF_ID_pc[31:0]	0000307c
IF_ID_pc_plus4[31:0]	00003080
IF_ID_instruction[31:0]	0080036f
ID_EX_pc[31:0]	00000000
ID_EX_pc_plus4[31:0]	00000000
ID_EX_reg1[31:0]	00000000
ID_EX_reg2[31:0]	00000000
ID_EX_imm[31:0]	00000000
ID_EX_reg_w_addr[4:0]	00
ID_EX_reg_write_sel[1:0]	0
ID_EX_alu_a_sel[1:0]	0
ID_EX_alu_b_sel[1:0]	0
ID_EX_alu_opcode[2:0]	0
ID_EX_mem_write_sel	0
ID_EX_mem_write	0
ID_EX_sw_r2_sel[1:0]	0
stall	0
flush	1

- **测试I/O版斐波拉契数列**

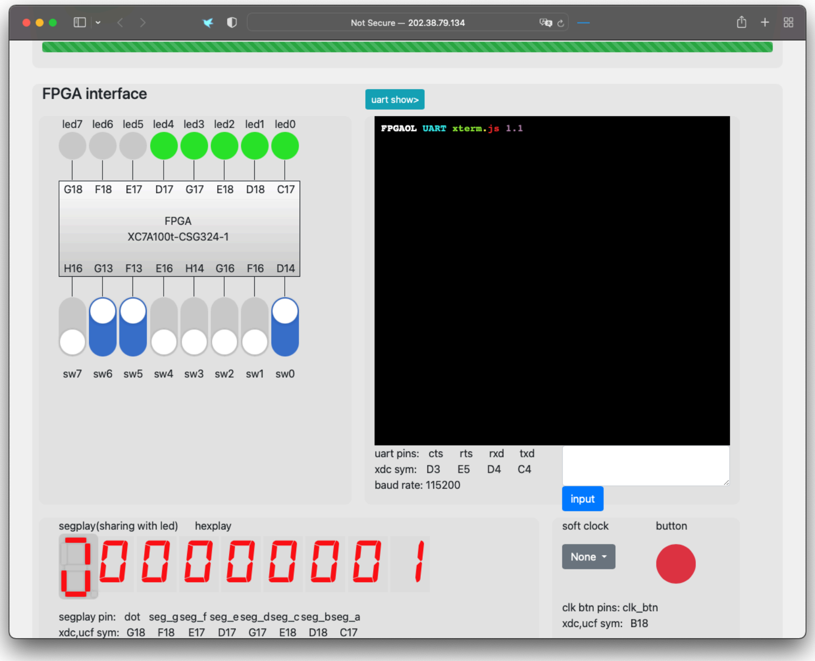
测试代码由群文件提供。

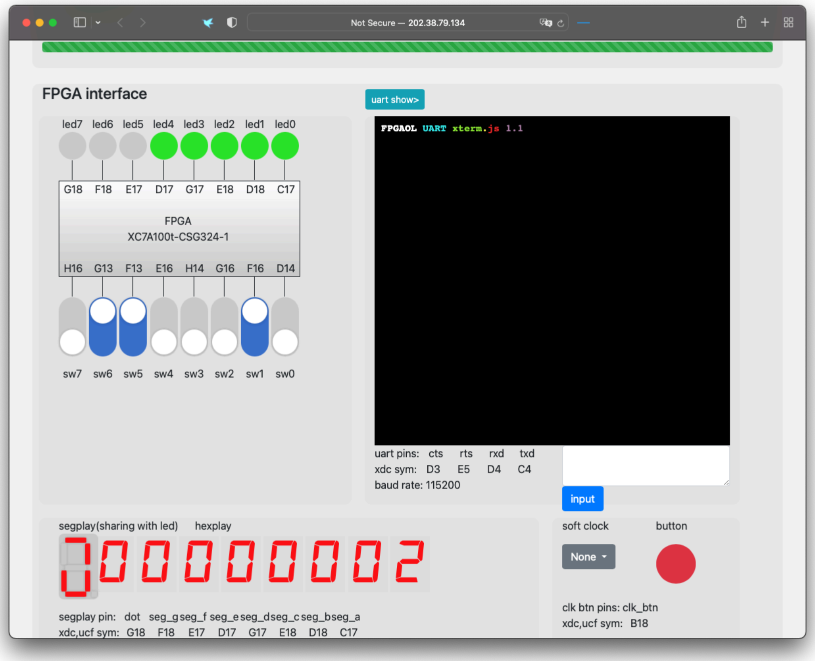
设计文件（其中pdu为群文件提供的模块）

```
module top(
    input clk,rst,      //clk,sw7
    input run,          //sw6
    input step,         //button
    input valid,        //sw5
    input [4:0] in,     //sw4-0
    output [1:0] check, //led6-5
    output [4:0] out0,  //led4-0
    output [2:0] an,    //an
    output [3:0] seg,   //seg
    output ready       //led7
);

wire cpu_clk;
wire [7:0] io_addr;
wire [31:0] io_dout;
wire io_we;
wire [31:0] io_din;
wire [7:0] m_rf_addr;
wire [31:0] rf_data;
wire [31:0] m_data;
wire [31:0] pc;
wire [31:0] pcd;
wire [31:0] ir;
wire [31:0] pcin;
wire [31:0] pce;
wire [31:0] a;
wire [31:0] b;
wire [31:0] imm;
wire [4:0] rd;
wire [31:0] ctrl;
wire [31:0] y;
wire [31:0] bm;
wire [4:0] rdm;
wire [31:0] ctrlm;
wire [31:0] yw;
wire [31:0] mdr;
wire [4:0] rdw;
wire [31:0] ctrlw;

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

pdu pdu(
    .clk(clk),
    .rst(rst),
    .run(run),
    .step(step),
    .clk_cpu(cpu_clk),
    .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),
    .pcd(pcd),
    .ir(ir),
    .pcin(pcin),
    .pce(pce),
    .a(a),
    .b(b),
    .imm(imm),
    .rd(rd),
    .ctrl(ctrl),
    .y(y),
    .bm(bm),
    .rdm(rdm),
    .ctrlm(ctrlm),
    .yw(yw),
    .mdr(mdr),
    .rdw(rdw),
    .ctrlw(ctrlw)
);

endmodule
```

运行结果

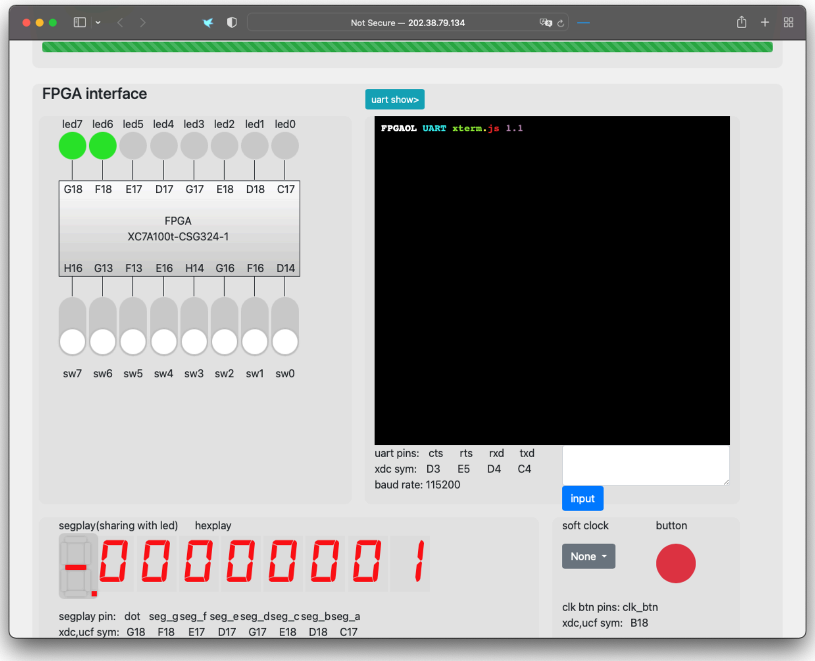
（前两项分别输入1，2）


Not Secure — 202.38.79.134
FPGA interface
uart show>
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
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led) hexplay
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock
None
button
clk btn pins: clk_btn
xdc,ucf sym: B18

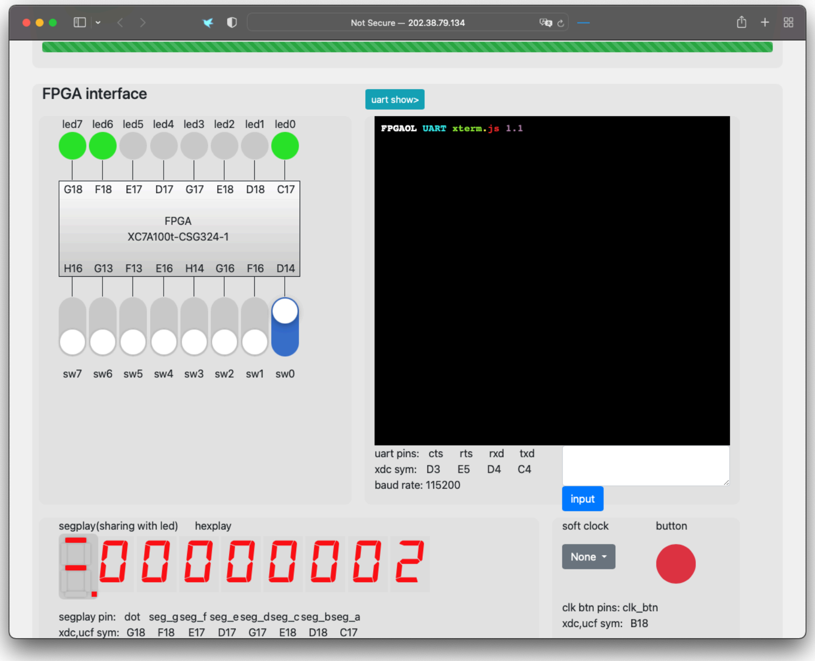


Not Secure — 202.38.79.134
FPGA interface
uart show>
led7 led6 led5 led4 led3 led2 led1 led0
G18 F18 E17 D17 G17 E18 D18 C17
FPGA
XC7A100t-CSG324-1
H16 G13 F13 E16 H14 G16 F16 D14
sw7 sw6 sw5 sw4 sw3 sw2 sw1 sw0
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led) hexplay
00000002
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

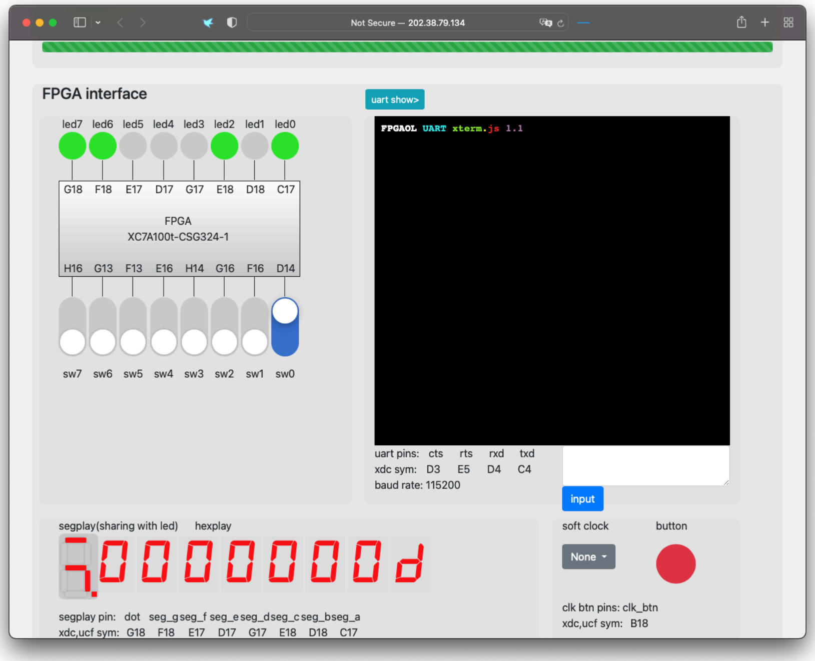
当停止运行时数据内存的状态



Not Secure — 202.38.79.134
FPGA interface
uart show>
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
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led) hexplay
00000001
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18



Not Secure — 202.38.79.134
FPGA interface
uart show>
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
FPGAOL UART xterm.js 1.1
uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200
input
segplay(sharing with led) hexplay
00000002
segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17
soft clock button
None
clk btn pins: clk_btn
xdc,ucf sym: B18


Not Secure — 202.38.79.134
FPGA interface
uart show>
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
input
segplay(sharing with led)
hexplay
0000000d
soft clock
button
None

```
FPGAOL UART xterm.js 1.1
```

uart pins: cts rts rxd txd
xdc sym: D3 E5 D4 C4
baud rate: 115200

segplay pin: dot seg_g seg_f seg_e seg_d seg_c seg_b seg_a
xdc,ucf sym: G18 F18 E17 D17 G17 E18 D18 C17

clk btn pins: clk_btn
xdc,ucf sym: B18

总结与思考

- 通过本次试验我更深入的了解了risc指令集下的流水线cpu的实现以及需要注意的事项 。
- 本次试验难易程适中。
- 本次试验任务量适中。