计算机组成原理 实验报告

实验题目：综合设计
学生姓名：阿非提
学生学号：PB20111633
完成日期：2022.5.26

实验目的

- 理解计算机硬件系统的组成结构和工作原理
- 掌握软硬件综合系统的设计和调试方法

实验平台

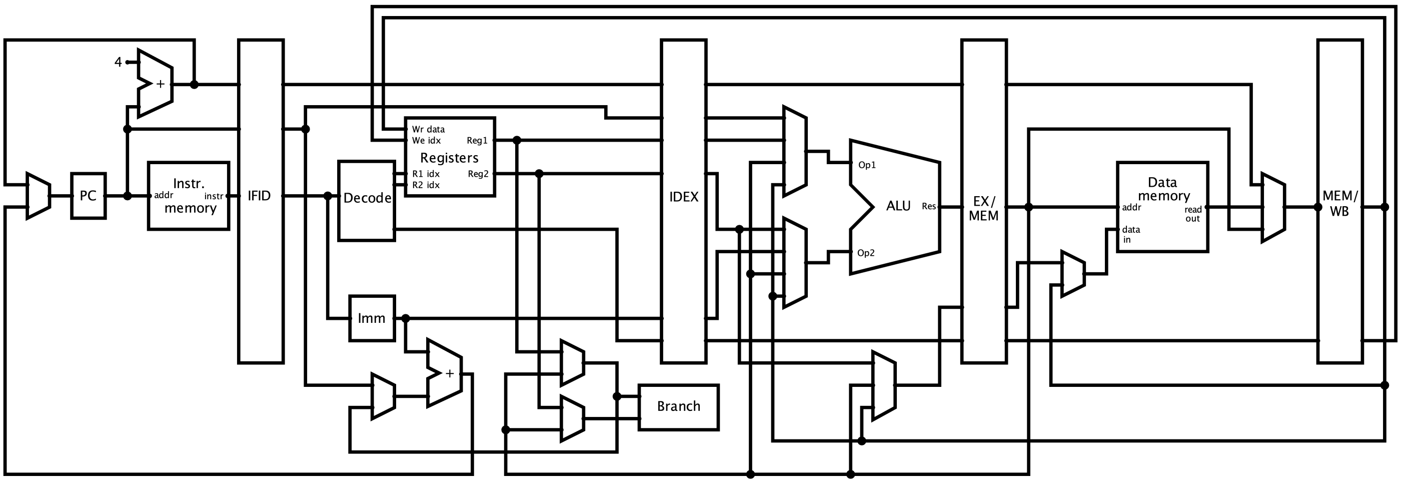
- Vivado
- Rars

实验练习

- **cpu设计**

  支持的指令： add, addi, sub, and, andi, or, ori, xor, xori, srl, srli, sll, slli, beq, bne, bge, blt, jal, jalr, auipc, lui, lw, sw, lh, sw, lb, sb 以及6个 ecall (print int, read int, print char, read char, print string, read string) 指令，共33个指令。

  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

二进制转BCD码

```
module binaryToBCD(
    input [31:0] int,
    output reg [39:0] bcd
    );
    integer i;

    always@(int)begin
        bcd = 0;
        for(i = 31; i >= 0; i = i - 1)begin
            if(bcd[3:0] > 4)
                bcd[3:0] = bcd[3:0] + 3;
            if(bcd[7:4] > 4)
                bcd[7:4] = bcd[7:4] + 3;
            if(bcd[11:8] > 4)
                bcd[11:8] = bcd[11:8] + 3;
            if(bcd[15:12] > 4)
                bcd[15:12] = bcd[15:12] + 3;
            if(bcd[19:16] > 4)
                bcd[19:16] = bcd[19:16] + 3;
            if(bcd[23:20] > 4)
                bcd[23:20] = bcd[23:20] + 3;
            if(bcd[27:24] > 4)
                bcd[27:24] = bcd[27:24] + 3;
            if(bcd[31:28] > 4)
                bcd[31:28] = bcd[31:28] + 3;
            if(bcd[35:32] > 4)
                bcd[35:32] = bcd[35:32] + 3;
            if(bcd[39:36] > 4)
                bcd[39:36] = bcd[39:36] + 3;
            bcd = {bcd[38:0],int[i]};
        end
    end

endmodule
```

BCD码转二进制

```
module BCDToBinary(
    input [39:0] bcd,
    output [31:0] int
    );

    wire [3:0] a0 = bcd[3:0];
    wire [6:0] a1 = (bcd[7:4] << 3) + (bcd[7:4] << 1);
    wire [9:0] a2 = (bcd[11:8] << 6) + (bcd[11:8] << 5) + (bcd[11:8] << 2) ;
    wire [13:0] a3 = (bcd[15:12] << 9) + (bcd[15:12] << 8) + (bcd[15:12] << 7) +
(bcd[15:12] << 6) + (bcd[15:12] << 5) + (bcd[15:12] << 3);
    wire [16:0] a4 =  (bcd[19:16] << 13) + (bcd[19:16] << 10) + (bcd[19:16] << 9)
+ (bcd[19:16] << 8) + (bcd[19:16] << 4);
    wire [19:0] a5 =  (bcd[23:20] << 16) + (bcd[23:20] << 15) + (bcd[23:20] <<
10) + (bcd[23:20] << 9) + (bcd[23:20] << 7) + (bcd[23:20] << 5);
```

```
    wire [23:0] a6 = (bcd[27:24] << 19) + (bcd[27:24] << 18) + (bcd[27:24] << 17) + (bcd[27:24] << 16) + (bcd[27:24] << 14) + (bcd[27:24] << 9) + (bcd[27:24] << 6);
    wire [26:0] a7 = (bcd[31:28] << 23) + (bcd[31:28] << 20) + (bcd[31:28] << 19) + (bcd[31:28] << 15) + (bcd[31:28] << 12) + (bcd[31:28] << 10) + (bcd[31:28] << 9) + (bcd[31:28] << 7);
    wire [29:0] a8 = (bcd[35:32] << 26) + (bcd[35:32] << 24) + (bcd[35:32] << 23) + (bcd[35:32] << 22) + (bcd[35:32] << 21) + (bcd[35:32] << 20) + (bcd[35:32] << 18) + (bcd[35:32] << 16) + (bcd[35:32] << 15) + (bcd[35:32] << 14) + (bcd[35:32] << 13) + (bcd[35:32] << 8);
    wire [33:0] a9 = (bcd[39:36] << 29) + (bcd[39:36] << 28) + (bcd[39:36] << 27) + (bcd[39:36] << 25) + (bcd[39:36] << 24) + (bcd[39:36] << 23) + (bcd[39:36] << 20) + (bcd[39:36] << 19) + (bcd[39:36] << 17) + (bcd[39:36] << 15) + (bcd[39:36] << 14) + (bcd[39:36] << 11) + (bcd[39:36] << 9);
    wire [33:0] a = a0 + a1 + a2 + a3 + a4 + a5 + a6 + a7 + a8 + a9;
    assign int = a[31:0];

endmodule
```

ecall

其中的 rx 和 tx 模块由数字电路实验 “实验10_综合试验” 提供。

```
module ecall(
    input clk,rst,
    input interrupt,
    output reg continue,
    input [31:0] a7,a0,
    output reg [31:0] reg_a0_data,
    output reg reg_a0_write,
    output reg [9:0] mem_read_addr,
    input [7:0] mem_read_data,
    output reg [9:0] mem_write_addr,
    output reg [7:0] mem_write_data,
    output reg mem_write_enable,
    input uart_rx,
    output uart_tx
    );

    reg [7:0] buffer[0:255];

    reg [1:0] current_state;
    reg [1:0] next_state;

    reg finish_loding;
    reg finish_flushing;

    parameter IDLE = 2'h0;
    parameter LOAD_BUFFER = 2'h1;
    parameter FLUSH_BUFFER = 2'h2;
```

```verilog
always@(*)begin
    case (current_state)
    IDLE:begin
        if(interrupt)
            next_state = LOAD_BUFFER;
        else
            next_state = IDLE;
    end
    LOAD_BUFFER:begin
        if(finish_loding)
            next_state = FLUSH_BUFFER;
        else
            next_state = LOAD_BUFFER;
    end
    FLUSH_BUFFER:begin
        if(finish_flushing)
            next_state = IDLE;
        else
            next_state = FLUSH_BUFFER;
    end
    default: next_state = IDLE;
    endcase
end

always@(posedge clk or posedge rst)begin
    if (rst)
        continue <= 0;
    else if ((current_state == FLUSH_BUFFER) && (next_state == IDLE))
        continue <= 1;
    else
        continue <= 0;
end


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

//printInt
wire isNegative = a0[31];
wire [31:0] int = isNegative? (~a0[31:0] + 1): a0[31:0];
wire [39:0] bcd;
wire [3:0] bcds[0:9];
assign bcds[0] = bcd[39:36];
assign bcds[1] = bcd[35:32];
assign bcds[2] = bcd[31:28];
assign bcds[3] = bcd[27:24];
assign bcds[4] = bcd[23:20];
assign bcds[5] = bcd[19:16];
assign bcds[6] = bcd[15:12];
assign bcds[7] = bcd[11:8];
assign bcds[8] = bcd[7:4];
assign bcds[9] = bcd[3:0];

parameter  SIGN = 8'd45;
parameter CHAR_ZERO = 8'd48;

binaryToBCD binaryToBCD(
    .int(int),
    .bcd(bcd)
);

//readInt
wire [31:0] IntTemp;
reg [39:0] Bcd;
reg IsNegative;
reg [7:0] temp;
wire [31:0] bcd_to_int = IsNegative ? (~IntTemp + 1): IntTemp;
reg [31:0] Int;

BCDToBinary BCDToBinary(
    .int(IntTemp),
    .bcd(Bcd)
);


//load buffer
integer i;
reg [7:0] index1;
reg [3:0] index2;
reg isLeadingZero;

wire rx_vld;
wire [7:0] rx_data;

rx rx(
    .clk(clk),
    .rst(rst),
    .rx(uart_rx),
    .rx_vld(rx_vld),
    .rx_data(rx_data)
);

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

```verilog
        if(rst)begin
            for(i = 0; i < 256; i = i + 1)begin
                buffer[i] <= 0;
            end
            finish_loding <= 0;
            index1 <= 0;
            index2 <= 0;
            mem_read_addr <= a0;
            isLeadingZero <= 1;
            IsNegative <= 0;
            Bcd <= 0;
        end
        else if (current_state == LOAD_BUFFER)begin
            if (a7 == 1)begin //printInt
                if(index1 == 8'h0 && isNegative)begin
                    buffer[index1] <= SIGN;
                    index1 <= index1 + 1;
                end
                else if (index2 < 4'ha)begin
                    if (index2 < 4'h9 && bcds[index2] == 0 && isLeadingZero)begin
                        index2 = index2 + 1;
                    end
                    else begin
                        isLeadingZero <= 0;
                        buffer[index1] <= bcds[index2] + CHAR_ZERO;
                        index1 <= index1 + 1;
                        index2 <= index2 + 1;
                    end
                end
                else begin
                    buffer[index1] <= 0;
                    finish_loding <= 1;
                end
            end
            else if(a7 == 4)begin//printString
                if(mem_read_data && index1 < 255)begin
                    buffer[index1] <= mem_read_data;
                    index1 <= index1 + 1;
                    mem_read_addr <= mem_read_addr + 1;
                end
                else begin
                    buffer[index1] <= 0;
                    finish_loding <= 1;
                end
            end
            else if (a7 == 11)begin//printChar
                buffer[0] <= a0[7:0];
                buffer[1] <= 0;
                finish_loding <= 1;
            end
            else if (a7 == 5)begin//readInt
                if (index1 > 10)begin
                    Int <= bcd_to_int;
                    finish_loding <= 1;
                end
                else if (rx_vld)begin
                    if(index1 == 0 && rx_data == SIGN)begin
                        IsNegative <= 1;
                    end
                    else if(rx_data >= 8'h30 && rx_data <= 8'h39)begin
                        temp = rx_data - CHAR_ZERO;
                        Bcd <= {Bcd[35:0],temp[3:0]};
                        index1 <= index1 + 1;
                    end
                    else begin
                        Int <= bcd_to_int;
                        finish_loding <= 1;
                    end
                end
            end
            else if (a7 == 8 || a7 == 12) begin//readChar,readString
                if (index1 >= 255)begin
                    buffer[255] <= 0;
                    finish_loding <= 1;
                end
                else if (rx_vld)begin
                    if(rx_data == 8'ha)begin
                        buffer[index1] <= 0;
                        finish_loding <= 1;
                    end
                    else begin
                        buffer[index1] <= rx_data;
                        index1 = index1 + 1;
                    end
                end
            end
            else begin //error
                buffer[0] <= 0;
                finish_loding <= 1;
            end
        end
        else begin
            finish_loding <= 0;
            index1 <= 0;
            index2 <= 0;
            mem_read_addr <= a0;
            isLeadingZero <= 1;
            IsNegative <= 0;
            Bcd <= 0;
        end
end

//flush buffer
reg [7:0] index3;

reg tx_ready;
reg [7:0] tx_data;
wire tx_rd;

tx tx(
    .clk(clk),
    .rst(rst),
    .tx(uart_tx),
    .tx_ready(tx_ready),
    .tx_rd(tx_rd),
    .tx_data(tx_data)
);

always@(posedge clk or posedge rst)begin
    if (rst)begin
        finish_flushing <= 0;
        index3 <= 0;
        tx_ready <= 0;
        mem_write_enable <= 0;
        mem_write_addr <= 0;
        mem_write_data <= 0;
        reg_a0_data <= 0;
        reg_a0_write <= 0;
        tx_data <= 0;
    end
    else if (current_state == FLUSH_BUFFER)begin
        if (a7 == 5)begin
            reg_a0_data <= Int;
            reg_a0_write <= 1;
            finish_flushing <= 1;
        end
        else if (a7 == 8)begin
            if(buffer[index3] && index3 < 255)begin
                mem_write_addr <= a0 + index3;
                mem_write_data <= buffer[index3];
                mem_write_enable <= 1;
                index3 = index3 + 1;
            end
            else begin
                mem_write_addr <= a0 + index3;
                mem_write_data <= 0;
                mem_write_enable <= 1;
                finish_flushing <= 1;
            end
        end
        else if (a7 == 12)begin
            reg_a0_data <= {24'h0,buffer[0]};
            reg_a0_write <= 1;
            finish_flushing <= 1;
        end
        else if (a7 == 1 || a7 == 4 || a7 == 11)begin
            if (buffer[index3] == 0 || index3 >= 255)begin
                tx_ready <= 0;
                finish_flushing <= 1;
            end
            else begin
                tx_ready <= 1;
                tx_data <= buffer[index3];
                if (tx_rd)begin
                    index3 = index3 + 1;
                end
            end
        end
        else begin
            finish_flushing <= 1;
        end
    end
    else begin
        finish_flushing <= 0;
        index3 <= 0;
        tx_ready <= 0;
        mem_write_enable <= 0;
        mem_write_addr <= 0;
        mem_write_data <= 0;
        reg_a0_data <= 0;
        reg_a0_write <= 0;
        tx_data <= 0;
    end
end

endmodule
```

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,
```

```verilog
    output reg [31:0] read_data1,
    output reg [31:0] read_data2,
    output [31:0] a0,
    output [31:0] a7,
    input reg_a0_write,
    input [31:0] reg_a0_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

    assign a0 = (write_register == 5'd10)? write_data : registers[10];
    assign a7 = (write_register == 5'd17)? write_data : registers[17];

    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;
        else if(reg_a0_write)
            registers[10] <= reg_a0_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;
        3'h7: result = signed_operant2;
        default: result = 0;
        endcase
    end

endmodule
```

立即数扩展

```
module imme(
    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
        7'b0110111: 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] addressing,
    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_ecall,
    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_lh = (opcode == 7'b0000011) && (funct3 == 3'b001);
    wire is_sh = (opcode == 7'b0100011) && (funct3 == 3'b001);
    wire is_lb = (opcode == 7'b0000011) && (funct3 == 3'b000);
    wire is_sb = (opcode == 7'b0100011) && (funct3 == 3'b000);
    wire is_lui = (opcode == 7'b0110111);
    assign is_ecall = (instruction == 32'h00000073);

    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 | is_lh | is_lb | is_lui;;
    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 | is_sh | is_lh | is_sb | is_lb | is_lui;
    assign mem_write = is_sw | is_sh | is_sb;
    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
| is_lh | is_sh | is_lb | is_sb;
    assign is_r2_EX_needed = is_add | is_sub | is_and  | is_or  | is_xor  |
is_sll | is_srl | is_sw | is_sh | is_sb;
    assign is_r2_MEM_needed = is_sw | is_sh | is_sb;

     always@(*)begin
        if (is_lw | is_sw)
            addressing = 2'h0;
        else if (is_lh | is_sh)
            addressing = 2'h1;
        else if (is_lb | is_sb)
            addressing = 2'h2;
        else
            addressing = 2'h0;
    end

    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 | is_lh | is_lb)
            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 if(is_lui)
            alu_opcode = 3'h7;
        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 is_ecall,
    input continue,
    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,
    output interrupt
    );

    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 is_ecall_dependent =  (ID_EX_reg_w_addr == 5'd10) || (ID_EX_reg_w_addr
== 5'd17) || (EX_MEM_reg_w_addr == 5'd10) || (EX_MEM_reg_w_addr == 5'd17);
    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 | (is_ecall
& (~continue));
    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;
    assign interrupt = is_ecall & (~is_ecall_dependent) & (~continue);


    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

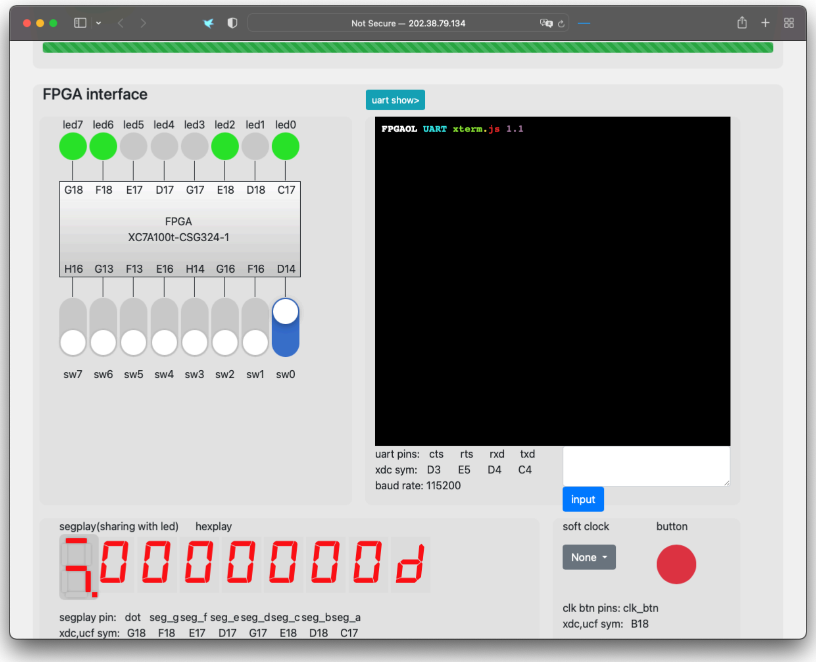
```
module data_io_memory(
    input clk,
    input write_enable,
    input [1:0] addressing, //0: word addressing, 1: half word addressing, 2:
byte addressing
    input [31:0] addr,
    input [31:0] write_data,
    output [31:0] read_data,
    input [9:0] mem_write_addr,
    input [7:0] mem_write_data,
    input mem_write_enable,
    input [9:0] mem_read_addr,
    output [7:0] mem_read_data
    );

    parameter mask = 32'hffffffff;
    parameter mask1 = 32'h0000ffff;
    parameter mask2 = 32'h000000ff;

    //
    reg [31:0] data_ram_read_data;
    reg [31:0] data_ram_write_data;
    wire [31:0] data_ram_read_data_temp;


    wire [31:0] mem_write_data_temp = (data_ram_read_data & ( mask ^ (mask2 <<
(mem_write_addr[1:0] * 8)))) | (mem_write_data << (mem_write_addr[1:0] * 8));
    wire [31:0] mem_read_data_temp;
    assign read_data =  data_ram_read_data;
    assign mem_read_data = mem_read_data_temp[(mem_read_addr[1:0] * 8)+:8];


    //addressing mode
    always@(*)begin
        case(addressing)
        2'h0:begin
            data_ram_read_data = data_ram_read_data_temp;
            data_ram_write_data = write_data;
        end
        2'h1:begin
            data_ram_read_data = {16'h0, data_ram_read_data_temp[(addr[1] *
16)+:16]};
            data_ram_write_data = (data_ram_read_data_temp & ( mask ^ (mask1 <<
(addr[1] * 16)))) | ((write_data & mask1) << (addr[1] * 16));
        end
        2'h2:begin
            data_ram_read_data = {24'h0, data_ram_read_data_temp[(addr[1:0] *
8)+:8]};
            data_ram_write_data = (data_ram_read_data_temp & ( mask ^ (mask2 <<
(addr[1:0] * 8)))) | ((write_data & mask2) << (addr[1:0] * 8));
        end
        default:begin
            data_ram_read_data = data_ram_read_data_temp;
            data_ram_write_data = write_data;
        end
        endcase
    end

    //data_mem parameters
    wire [31:0] Data = mem_write_enable ? mem_write_data_temp :
data_ram_write_data;
    wire [7:0] Addr = mem_write_enable ? mem_write_addr[9:2] : addr[9:2];
    wire Write = write_enable | mem_write_enable;

    data_mem data_mem(
        .clk(clk),
        .a(Addr),
        .d(Data),
        .we(Write),
        .spo(data_ram_read_data_temp),
        .dpra(mem_read_addr[9:2]),
        .dpo(mem_read_data_temp)
    );

endmodule
```

cpu

```
module cpu(
    input clk,rst,
    input uart_rx,
    output uart_tx,
    output ecall_write_ready
);
```

```verilog
wire [31:0] pc;
wire interrupt;
wire continue;



//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;
reg [1:0]  ID_EX_addressing;
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;
reg [1:0]  EX_MEM_addressing;

//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;
    wire [31:0] a0;
    wire [31:0] a7;
    wire reg_a0_write;
    wire [31:0] reg_a0_data;

    register_file register_file(
        .clk(clk),
        .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),
```

```verilog
        .read_data2(reg_data2),
        .a0(a0),
        .a7(a7),
        .reg_a0_write(reg_a0_write),
        .reg_a0_data(reg_a0_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_ecall;
    wire is_jal;
    wire is_jalr;
    wire is_branch;
    wire [1:0] addressing;

    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),
        .addressing(addressing),
        .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_ecall(is_ecall),
        .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),
        .is_ecall(is_ecall),
        .continue(continue),
        .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),
        .interrupt(interrupt)
    );

    //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;
            ID_EX_addressing <= 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;
```


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
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
button
None
clk btn pins: clk_btn
xdc,ucf sym: B18

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

```
                                                                ID_EX_addressing
<=                                                              addressing;
                                                                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;
            EX_MEM_addressing <= 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;
            EX_MEM_addressing <= ID_EX_addressing;
        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;
    wire [9:0] mem_write_addr;
    wire [7:0] mem_write_data;
    wire mem_write_enable;
    wire [9:0] mem_read_addr;
    wire [7:0] mem_read_data;

    data_io_memory data_io_memory(
        .clk(clk),
        .write_enable(EX_MEM_mem_write),
        .addr(EX_MEM_alu_result),
        .addressing(EX_MEM_addressing),
        .write_data(write_data),
        .read_data(read_data),
        .mem_write_addr(mem_write_addr),
        .mem_write_data(mem_write_data),
        .mem_write_enable(mem_write_enable),
        .mem_read_addr(mem_read_addr),
        .mem_read_data(mem_read_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

//ecalls
    assign ecall_write_ready = is_ecall &  (a7 == 5 || a7 == 8 || a7 == 12);

    ecall ecall(
        .clk(clk),
        .rst(rst),
        .interrupt(interrupt),
        .continue(continue),
        .a7(a7),
        .a0(a0),
        .reg_a0_data(reg_a0_data),
        .reg_a0_write(reg_a0_write),
        .mem_read_addr(mem_read_addr),
        .mem_read_data(mem_read_data),
        .mem_write_addr(mem_write_addr),
        .mem_write_data(mem_write_data),
        .mem_write_enable(mem_write_enable),
        .uart_rx(uart_rx),
        .uart_tx(uart_tx)
    );


endmodule
```

- **cpu功能仿真**

运行使用的汇编代码

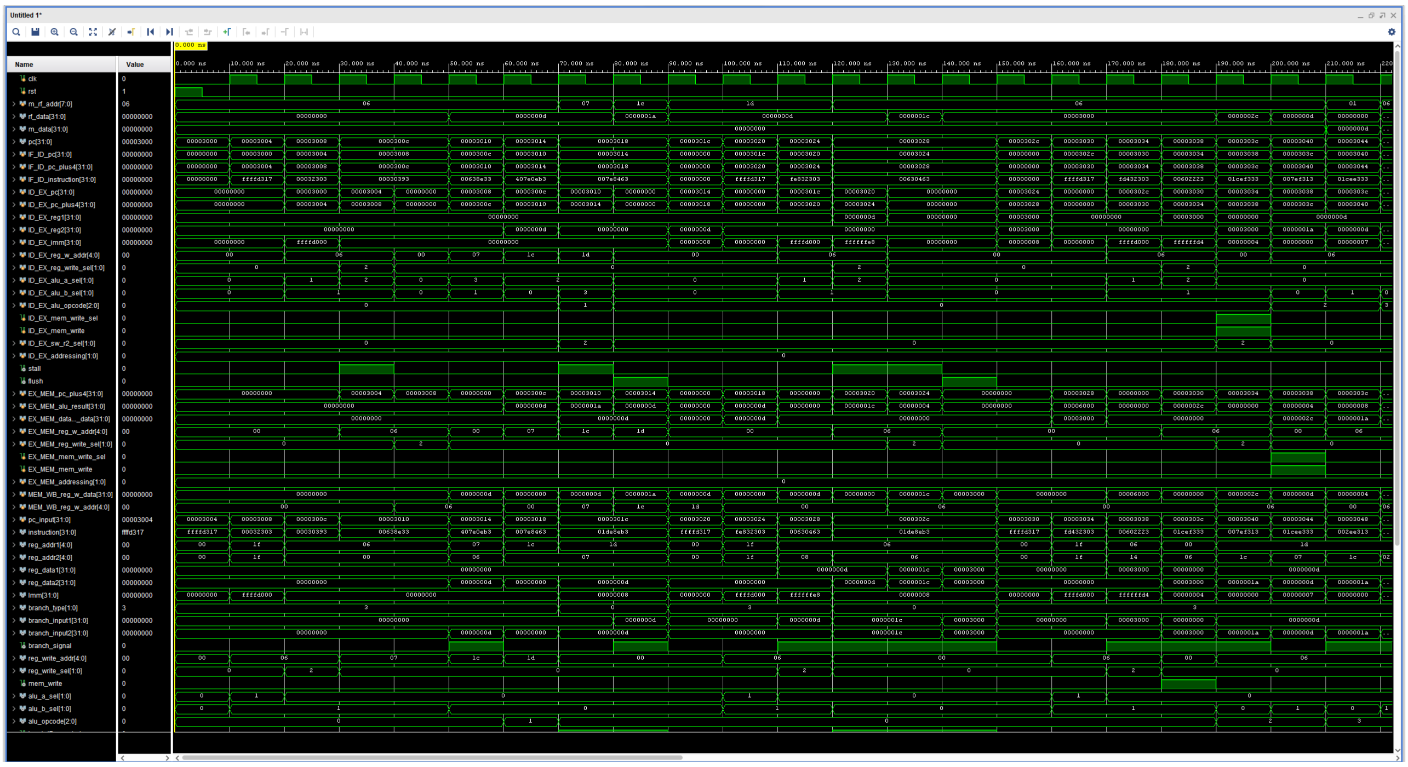
```
.data
    data1: .word 0xd
    data2: .word 0x3000
    data3: .word 0x12345678
.text
    lw t1,data1     #t1 = 0xd
    addi t2,t1,0        #t2 = 0xd   //EX data hazards stalling
    add t3,t2,t1        #t3 = 0x1a  //EX data hazards forwarding from EX/MEM
    sub t4,t3,t2        #t4 = 0xd   //EX data 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,Branch3     #branch taken
    add t4,t4,t4
Branch3:bge t2,zero,Branch4     #branch taken
    add t4,t4,t4
Branch4:bne t2,zero,Branch5 #branch taken
    add t4,t4,t4
Branch5:jal t1,Jump
    add t4,t4,t4
Jump:   li t4,305419896     #t4 = 0x12345678
    lb t2,data3     #t2 = 0x78
    sb t4,(zero)        #data1 = 0x00000078
    lh t2,data3     #t2 = 0x5678
    sh t4,2(zero)       #data1 = 0x56780078
```

rars 生成的代码以及流水线执行图

Lab6 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	420	430	440	450	460	470	480	490
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							
3084	lui x29,0x00012345																																						IF	ID	EX	MEM	WB						
3088	addi x29,x29,0x00000678																																							IF	ID	EX	MEM	WB					
308c	auipc x7,0xfffffffd																																								IF	ID	EX	MEM	WB				
3090	lb x7,0xffffff7c(x7)																																									IF	ID	EX	MEM	WB			
3094	sb x29,0(x0)																																										IF	ID	EX	MEM	WB		
3098	auipc x7,0xfffffffd																																											IF	ID	EX	MEM	WB	
309c	lh x7,0xffffff70(x7)																																												IF	ID	EX	MEM	WB	
30a0	sh x29,2(x0)																																													IF	ID	EX	MEM	WB

仿真结果

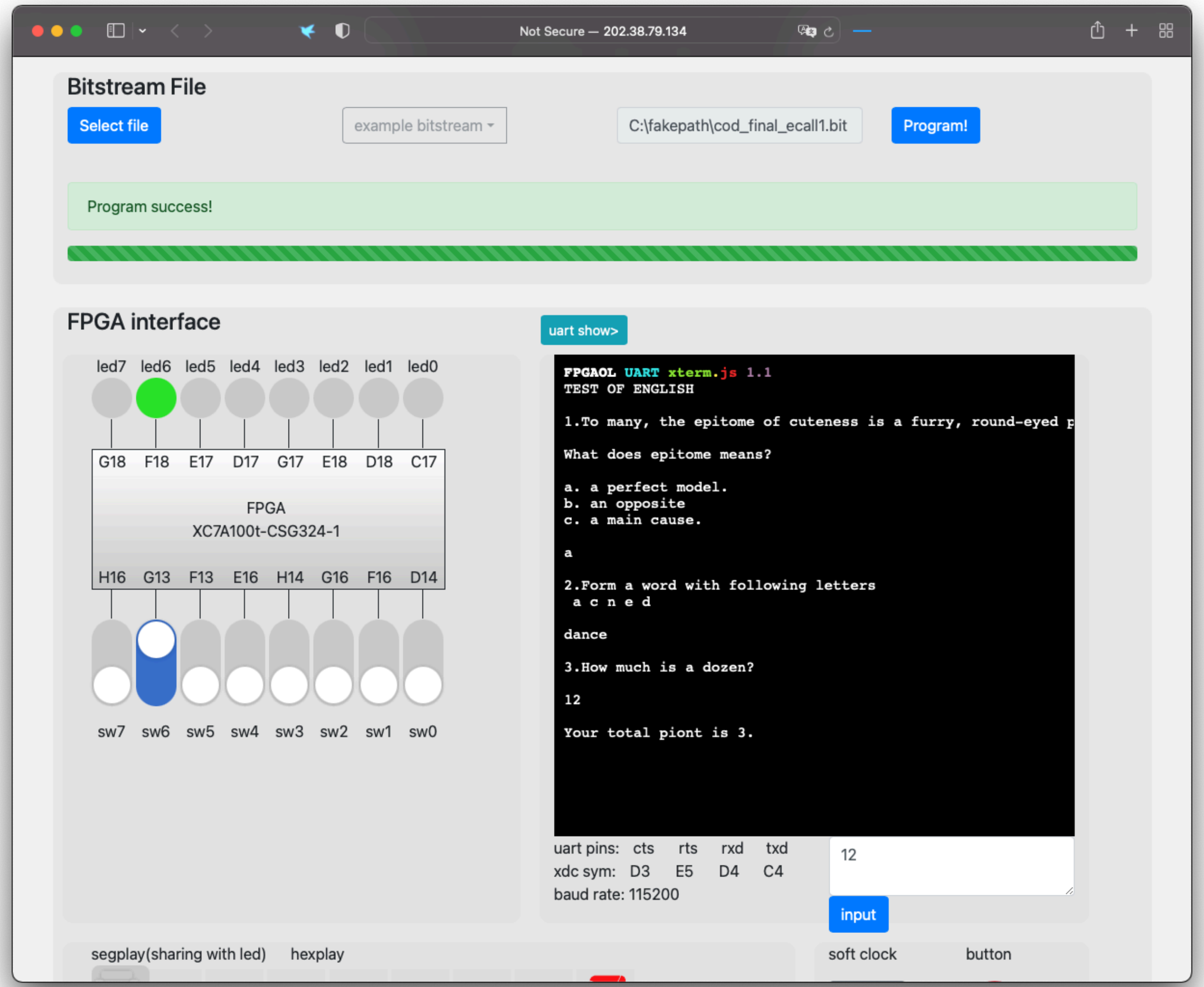


Name	Value
clk	0
rst	1
m_rf_addr[7:0]	06
rf_data[31:0]	00000000
m_data[31:0]	00000000
pc[31:0]	00003000
IF_ID_pc[31:0]	00000000
IF_ID_pc_plus4[31:0]	00000000
IF_ID_instruction[31:0]	00000000
ID_EX_pc[31:0]	00000000
ID_EX_pc_plus4[31:0]	00000000
ID_EX_reg1[31:0]	00000000
ID_EX_reg2[31:0]	00000000
ID_EX_imm[31:0]	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
ID_EX_addressing[1:0]	0
stall	0
flush	0
EX_MEM_pc_plus4[31:0]	00000000
EX_MEM_alu_result[31:0]	00000000
EX_MEM_data..._data[31:0]	00000000
EX_MEM_reg_w_addr[4:0]	00
EX_MEM_reg_write_sel[1:0]	0
EX_MEM_mem_write_sel	0
EX_MEM_mem_write	0
EX_MEM_addressing[1:0]	0
MEM_WB_reg_w_data[31:0]	00000000
MEM_WB_reg_w_addr[4:0]	00
pc_input[31:0]	00003004
instruction[31:0]	ffffd317
reg_addr1[4:0]	00
reg_addr2[4:0]	00
reg_data1[31:0]	00000000
reg_data2[31:0]	00000000
Imm[31:0]	00000000
branch_type[1:0]	3
branch_input1[31:0]	00000000
branch_input2[31:0]	00000000
branch_signal	0
reg_write_addr[4:0]	00
reg_write_sel[1:0]	0
mem_write	0
alu_a_sel[1:0]	0
alu_b_sel[1:0]	0
alu_opcode[2:0]	0

- **ecall 测试汇编代码1**

```
.data
    guess:      .word 0x00000000
    guess1:     .word 0x00000000
    answer:     .word 0x636e6164
    answer1:    .word 0x00000065
    greeting:   .string "TEST OF ENGLISH\n\n"
    question1:  .string "1.To many, the epitome of cuteness is a furry, round-
eyed puppy\n\nWhat does epitome means?\n\na. a perfect model.\nb. an opposite\nc.
a main cause.\n\n"
    question2:  .string "\n\n2.Form a word with following letters\n a c n e
d\n\n"
    question3:  .string "\n\n3.How much is a dozen?\n\n"
    ending:     .string "\n\nYour total piont is "
.text
    #initialize score, answers ...
    li t0,0
    li t1,97
    lw t2,answer
    lw t3,answer1
    li t4,12

    #greet
    la a0,greeting
    li a7,4
    ecall

    #first question
    la a0,question1
    ecall

    #answer
    li a7,12
    ecall
    li a7,11
    ecall
    bne a0,t1,Wrong
    addi t0,t0,1

Wrong:  #second question
    la a0,question2
    li a7,4
    ecall

    #answer
    li a0,0
    li a7,8
    ecall
    li a7,4
    ecall

    lw t5,guess
    lw t6,guess1
    bne t2,t5,Wrong1
    bne t3,t6,Wrong1
    addi t0,t0,1

Wrong1: #third queston
    la a0,question3
    li a7,4
    ecall

    #answer
    li a7,5
    ecall
    li a7,1
    ecall

    bne a0,t4,Wrong2
    addi t0,t0,1

Wrong2: #final result
    la a0,ending
    li a7,4
    ecall
    addi a0,t0,0
    li a7,1
    ecall
    li a0,46
    li a7,11
    ecall

Loop:   jal Loop
```

FPGAOL测试结果


Not Secure — 202.38.79.134
Bitstream File
Select file
example bitstream
C:\fakepath\cod_final_ecall1.bit
Program!
Program success!
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
12
input
segplay(sharing with led)
hexplay
soft clock
button

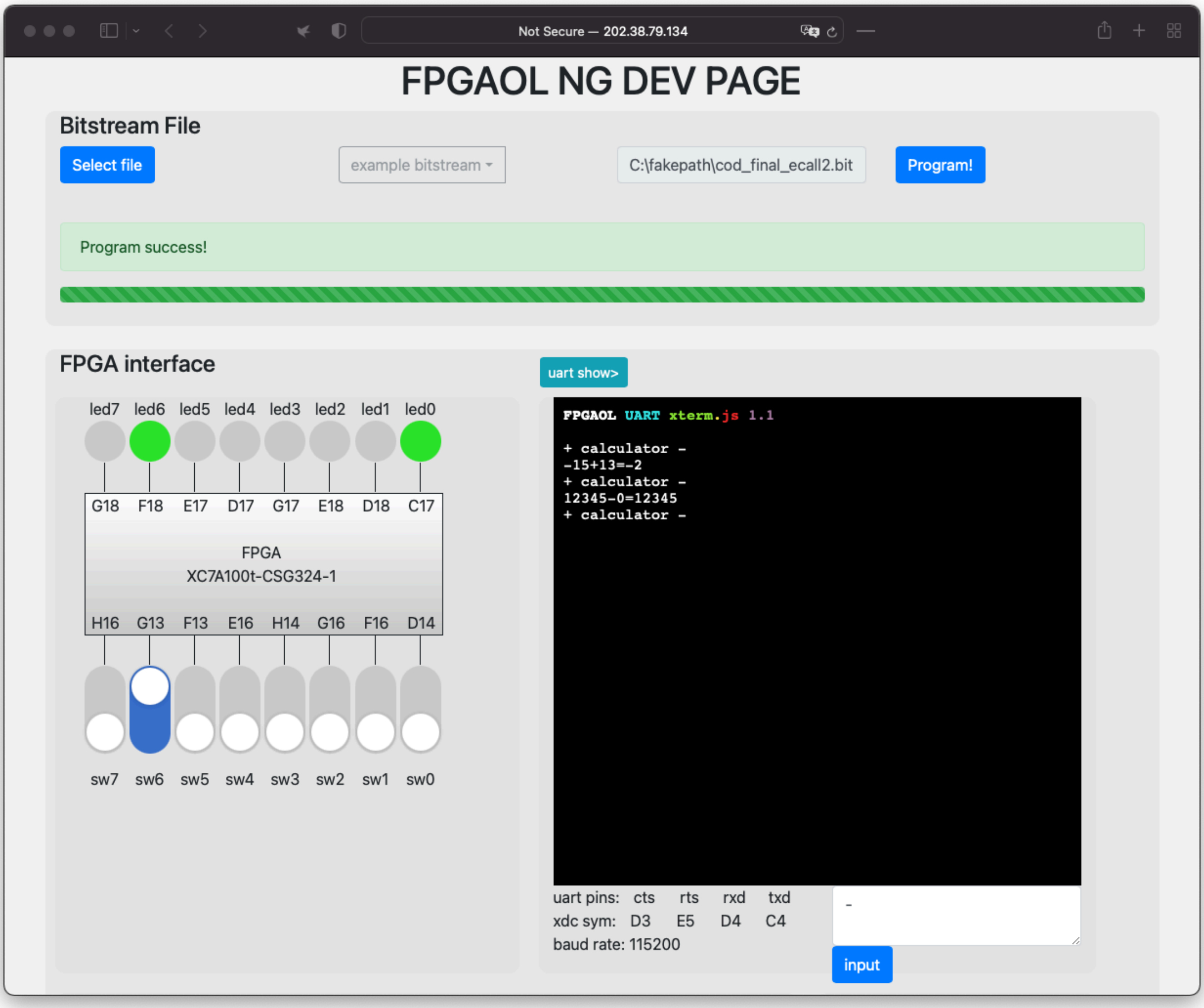
```
FPGAOL UART xterm.js 1.1
TEST OF ENGLISH

1.To many, the epitome of cuteness is a furry, round-eyed p

What does epitome means?

a. a perfect model.
b. an opposite
c. a main cause.

a

2.Form a word with following letters
 a c n e d

dance

3.How much is a dozen?

12

Your total piont is 3.
```

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

- **ecall 测试汇编代码2**

```
.data
    prompt:      .string "\n+ calculator - \n"
    prompt1:     .string "\ninvalid operator!\n"

.text
    #initialize
    li t3,43     #t3 = '+'
    li t4,45     #t4 = '-'

    #prompt
start:  la a0, prompt
    li a7,4
    ecall

    #first operant
    li a7,5
    ecall
    li a7,1
    ecall
    addi t0,a0,0
    #operator
    li a7,12
    ecall
    li a7,11
    ecall
    addi t1,a0,0

    #second operator
    li a7,5
    ecall
    li a7,1
    ecall
    addi t2,a0,0

    #if operator is addition
    bne t1,t3,NotADD
    add t0,t0,t2
    jal Print

    #if operator is subtraction
NotADD: bne t1,t4,NotSUB
    sub t0,t0,t2

    #print the result
Print:  li a0,61
    li a7,11
    ecall
    addi a0,t0,0
    li a7,1
    ecall
    li a0,10
    li a7,11
    jal start

    #if oprator is invalid
NotSUB: la a0,prompt1
    li a7,4
    ecall
    jal start
```

FPGAOL测试结果


Not Secure — 202.38.79.134
FPGAOL NG DEV PAGE
Bitstream File
Select file
example bitstream
C:\fakepath\cod_final_ecall2.bit
Program!
Program success!
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

```
FPGAOL UART xterm.js 1.1

+ calculator -
-15+13=-2
+ calculator -
12345-0=12345
+ calculator -
```

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

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。