浙江大学

本科实验报告

课程名称:	计算机组成与设计
姓　　名:	张晋恺
学　　院:	竺可桢学院
系:	所在系
专　　业:	计算机科学与技术
学　　号:	3230102400
指导教师:	刘海风

2024 年 9 月 28 日

浙江大学实验报告

课程名称：计算机组成与设计　实验类型：综合

实验项目名称：ALU，RegFile 与 FSM

学生姓名：张晋恺　专业：计算机科学与技术 学号：3230102400

同组学生姓名：　指导老师：刘海风

实验地点：东 4-512　实验日期：2024 年 9 月 28 日

一、操作方法与实验步骤

ALU 设计

源码

```
    module ALU (
    input  [31:0] A,
    input  [31:0] B,
    input  [3:0]  ALU_operation,
    output reg [31:0] res,
    output zero
);

// zero 信号在结果为 0 时为高电平
assign zero = (res == 32'd0) ? 1'b1 : 1'b0;

always @(*) begin
```

```
    case (ALU_operation)
        4'd0: res = A + B;           // ADD 操作
        4'd1: res = A - B;           // SUB 操作
        4'd2: res = A << B[4:0];     // SLL 操作，逻辑左移
        4'd3: res = ($signed(A) < $signed(B)) ? 32'd1 : 32'd0; // SLT 操作，有符号比较
        4'd4: res = (A < B) ? 32'd1 : 32'd0;  // SLTU 操作，无符号比较
        4'd5: res = A ^ B;           // XOR 操作
        4'd6: res = A >> B[4:0];     // SRL 操作，逻辑右移
        4'd7: res = $signed(A) >>> B[4:0]; // SRA 操作，算术右移
        4'd8: res = A | B;           // OR 操作
        4'd9: res = A & B;           // AND 操作
        default: res = 32'dx;        // 默认输出 x
    endcase
end

endmodule
```

仿真代码

```
module ALU_tb;

reg [31:0] A,B;
reg [3:0] ALU_operation;
wire [31:0] res;
wire zero;

ALU ALU_inst(
    .A(A),
    .B(B),
    .ALU_operation(ALU_operation),
    .res(res),
    .zero(zero)
);

integer i;

initial begin
    A=32'hA5A5A5A5;
    B=32'h5A5A5A5A;
    for(i=0;i<10;i=i+1) begin
        ALU_operation=i;
```

```
        #100;
    end
    //add overflow
    ALU_operation=4'd0;
    A=32'hFFFFFFFF;
    B=32'h00000001;
    #100;
    //sub need borrow
    ALU_operation=4'd1;
    A=32'h00000001;
    B=32'h00000002;
    #100;
    // logical shift left
    ALU_operation=4'd2;
    A=32'h00000001;
    B=32'd4;
    #100;
    // logical shift right
    ALU_operation=4'd6;
    A=32'h80000000;
    B=32'd4;
    #100;
    // arithmetic shift right
    ALU_operation=4'd7;
    A=32'hF0000000;
    B=32'd4;
    #100;
    //SLT
    ALU_operation=4'd3;
    A=32'hFFFFFFFF;
    B=32'h00000001;
    #100;
    //SLTU
    ALU_operation=4'd4;
    A=32'hFFFFFFFF;
    B=32'h00000001;
    #100;

    //Default output
    ALU_operation=4'hA;#10;
    ALU_operation=4'hB;#10;
    ALU_operation=4'hC;#10;
    ALU_operation=4'hD;#10;
    ALU_operation=4'hE;#10;
    ALU_operation=4'hF;#10;
```

```
68
69  end
70
71  endmodule
```

仿真波形以及分析见下一章

Register File 设计

源码

相当于一个二维数组，每个元素是一个 32 位的寄存器，通过地址选择器选择对应的寄存器，通过写使能信号和写数据信号写入数据，通过读地址信号读出数据。

```
1   module Regs(
2   input clk,
3   input rst,
4   input [4:0] Rs1_addr,
5   input [4:0] Rs2_addr,
6   input [4:0] Wt_addr,
7   input [31:0]Wt_data,
8   input RegWrite,
9   output [31:0] Rs1_data,
10  output [31:0] Rs2_data,
11  output [31:0] Reg00,
12  output [31:0] Reg01,
13  output [31:0] Reg02,
14  output [31:0] Reg03,
15  output [31:0] Reg04,
16  output [31:0] Reg05,
17  output [31:0] Reg06,
18  output [31:0] Reg07,
19  output [31:0] Reg08,
20  output [31:0] Reg09,
21  output [31:0] Reg10,
22  output [31:0] Reg11,
23  output [31:0] Reg12,
24  output [31:0] Reg13,
25  output [31:0] Reg14,
26  output [31:0] Reg15,
27  output [31:0] Reg16,
28  output [31:0] Reg17,
```

```
    output [31:0] Reg18,
    output [31:0] Reg19,
    output [31:0] Reg20,
    output [31:0] Reg21,
    output [31:0] Reg22,
    output [31:0] Reg23,
    output [31:0] Reg24,
    output [31:0] Reg25,
    output [31:0] Reg26,
    output [31:0] Reg27,
    output [31:0] Reg28,
    output [31:0] Reg29,
    output [31:0] Reg30,
    output [31:0] Reg31
);
// Your code here

    reg [31:0] Reg_file[31:1];

    assign Reg00 = 32'h00000000;
    assign Reg01 = Reg_file[1];
    assign Reg02 = Reg_file[2];
    assign Reg03 = Reg_file[3];
    assign Reg04 = Reg_file[4];
    assign Reg05 = Reg_file[5];
    assign Reg06 = Reg_file[6];
    assign Reg07 = Reg_file[7];
    assign Reg08 = Reg_file[8];
    assign Reg09 = Reg_file[9];
    assign Reg10 = Reg_file[10];
    assign Reg11 = Reg_file[11];
    assign Reg12 = Reg_file[12];
    assign Reg13 = Reg_file[13];
    assign Reg14 = Reg_file[14];
    assign Reg15 = Reg_file[15];
    assign Reg16 = Reg_file[16];
    assign Reg17 = Reg_file[17];
    assign Reg18 = Reg_file[18];
    assign Reg19 = Reg_file[19];
    assign Reg20 = Reg_file[20];
    assign Reg21 = Reg_file[21];
    assign Reg22 = Reg_file[22];
    assign Reg23 = Reg_file[23];
    assign Reg24 = Reg_file[24];
    assign Reg25 = Reg_file[25];
```

```
    assign Reg26 = Reg_file[26];
    assign Reg27 = Reg_file[27];
    assign Reg28 = Reg_file[28];
    assign Reg29 = Reg_file[29];
    assign Reg30 = Reg_file[30];
    assign Reg31 = Reg_file[31];

    assign Rs1_data = Rs1_addr? Reg_file[Rs1_addr] :0;
    assign Rs2_data = Rs2_addr? Reg_file[Rs2_addr] :0;

    integer i;
    always @(posedge clk or posedge rst) begin
        if (rst) begin
            for (i = 0; i < 32; i = i + 1) begin
                Reg_file[i] <= 32'h00000000;
            end
        end
        else begin
            if (RegWrite && Wt_addr) begin
                Reg_file[Wt_addr] <= Wt_data;
            end
        end
    end

endmodule
```

仿真代码

```
    `timescale 1ns / 1ns

module Regs_tb;
    reg clk;
    reg rst;
    reg [4:0] Rs1_addr;
    reg [4:0] Rs2_addr;
    reg [4:0] Wt_addr;
    reg [31:0]Wt_data;
    reg RegWrite;
    wire [31:0] Rs1_data;
    wire [31:0] Rs2_data;
    Regs Regs_U(
        .clk(clk),
```

```
        .rst(rst),
        .Rs1_addr(Rs1_addr),
        .Rs2_addr(Rs2_addr),
        .Wt_addr(Wt_addr),
        .Wt_data(Wt_data),
        .RegWrite(RegWrite),
        .Rs1_data(Rs1_data),
        .Rs2_data(Rs2_data)
    );

    always #10 clk = ~clk;

    initial begin
        clk = 0;
        rst = 1;
        RegWrite = 0;
        Wt_data = 0;
        Wt_addr = 0;
        Rs1_addr = 0;
        Rs2_addr = 0;
        #100
        rst = 0;
        RegWrite = 1;
        Wt_addr = 5'b00101;
        Wt_data = 32'ha5a5a5a5;
        #4
        Wt_data = 32'haaaa5555;
        #4
        Wt_data = 32'ha5a5a5a5;//only write on posedge clk
        #42
        Wt_addr = 5'b01010;
        Wt_data = 32'h5a5a5a5a;
        #50
        RegWrite = 0;
        Rs1_addr = 5'b00101;
        Rs2_addr = 5'b01010;

        Wt_addr = 5'b01010;
        Wt_data = 32'habcdedcb;//test when RegWrite = 0,cannot write
        #50
        RegWrite = 1;
        Wt_addr = 5'b00000;
        Wt_data = 32'hffffffff;//test cannot write reg00
        #50
        RegWrite = 0;
```

```
        Rs1_addr = 5'b00000;
        #100 $stop();
    end

endmodule

```

仿真波形以及分析见下一章

有限状态机 (FSM) 设计

源码

```
    module TruthEvaluator(
    input  clk,
    input  truth_detection,
    output trust_decision
);


// State definition
  localparam
    HIGHLY_TRUSTWORTHY=2'b00,
    TRUSTWORTHY=2'b01,
    SUSPICIOUS=2'b10,
    UNTRUSTWORTHY=2'b11;

    reg[1:0] curr_state,next_state;
    initial curr_state=HIGHLY_TRUSTWORTHY;


// First segment: state transfer
  always @(posedge clk) begin
       curr_state <= next_state;
  end

// Sencond segment: transfer condition
  always @(*) begin // combination logic
    case(curr_state)
    HIGHLY_TRUSTWORTHY: begin
        if(truth_detection==1'b1) next_state=HIGHLY_TRUSTWORTHY;
```

```
        else next_state=TRUSTWORTHY;
    end

    TRUSTWORTHY: begin
        if(truth_detection==1'b1) next_state=HIGHLY_TRUSTWORTHY;
        else next_state=SUSPICIOUS;
    end

    SUSPICIOUS: begin
        if(truth_detection==1'b1) next_state=TRUSTWORTHY;
        else next_state=UNTRUSTWORTHY;
    end

    UNTRUSTWORTHY:begin
        if(truth_detection==1'b1) next_state=SUSPICIOUS;
        else next_state=UNTRUSTWORTHY;
    end

    default:next_state=curr_state;

    endcase
  end

  assign trust_decision=((curr_state==HIGHLY_TRUSTWORTHY)||(curr_state==
   ↪  TRUSTWORTHY));


endmodule
```

仿真代码

```
module TruthEvaluator_tb;

reg clk;
reg truth_detection;
wire trust_decision;

always #5 clk=~clk;

TruthEvaluator uut(
    .clk(clk),
    .truth_detection(truth_detection),
```

```
    .trust_decision(trust_decision)
);

initial begin
    clk=0;
    truth_detection=0;
    #50
    truth_detection=1;
    #50
    $stop();
end

endmodule

```

具体仿真波形以及分析见下一章

二、实验结果与分析

ALU 设计

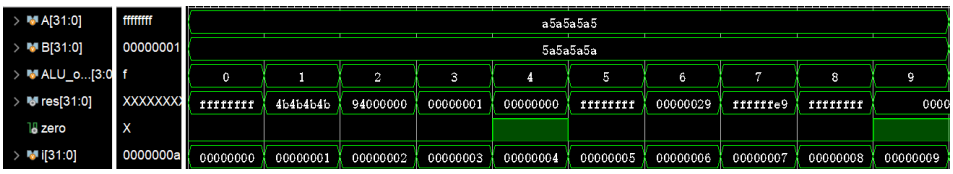

图 1: 常规仿真结果

接下来按 ALU 操作码的不同，依次验证 ALU 的功能：

$$A = (a5a5a5a5)_h = 1010\_0101\_1010\_0101\_1010\_0101\_1010\_0101_2$$
$$B = (5a5a5a5a)_h = 0101\_1010\_0101\_1010\_0101\_1010\_0101\_1010_2$$

$$A + B = (ffffffff)_h$$
$$A - B = (4b4b4b4b)_h$$
$$A << B[4:0] = 1001\_0100\_0000\_0000\_0000\_0000\_0000\_0000_2 = (94000000)_h$$
$$signed(A) < signed(B); res = 1$$
$$A > B; res = 0; zero = 1$$
$$A \oplus B = (ffff\_ffff)_h$$
$$A >> B[4:0] = 0000\_0000\_0000\_0000\_0000\_0000\_0010\_1001_2 = (00000029)_h$$
$$signed(A) >>> B[4:0] = 1111\_1111\_1111\_1111\_1111\_1111\_1110\_1001_2 = (ffffffe9)_h$$
$$A|B = (ffffffff)_h$$
$$A\&B = (0000\_0000)_h; zero = 1;$$

故基本操作正确

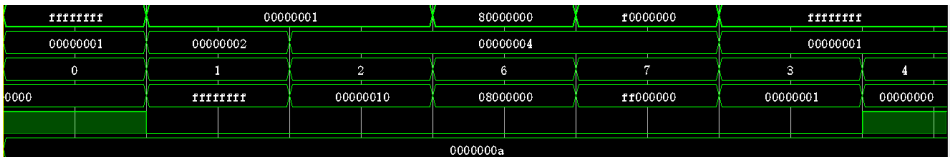

图 2: 加入边界测试

首先验证加法溢出的情况，可以看到，全 1 在加 1，会溢出，结果为 0，zero 信号为 1。再验证减法需要借位的情况，可以看到，A 是小于 B 的，减出来结果为全 1(-1 的补码)，zero 信号为 0。最后在验证一下逻辑移位与算数移位的区别，逻辑移位是在左边补 0，算数移位是在左边补符号位，结果也符合预期

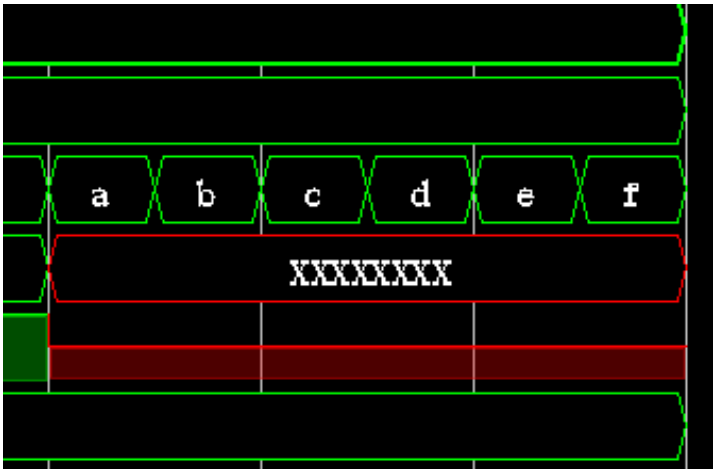

图 3: 默认输出

由于`ALU_operation[3:0]`是四位信号，最多会有 16 种操作，而我们只定义了 10 种操作，所以需要在其他情况下，输出为 x，这也是符合预期的。

Register file **设计**

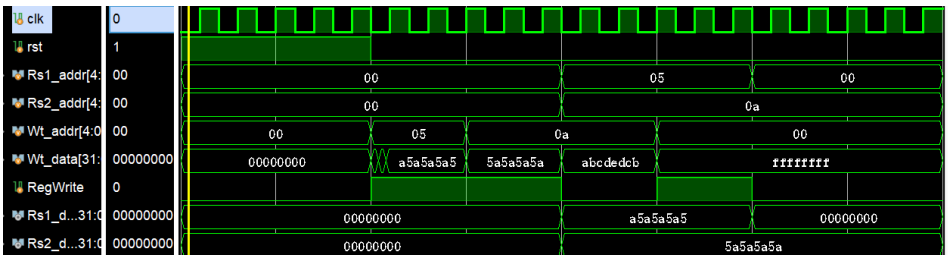

图 4: regfile 仿真结果

1. 首先 rst 信号为 1，所有寄存器的值都被清零
2. 然后将 rst 置为 0，将写入信号使能，向 5 号寄存器写入 `32'ha5a5a5a5`，**在这里，我设计了一个小脉冲，将写入信号短暂变为** `32'haaaa5555`，用于验证即使信号改变且写入型号使能，但是没有达到时钟上升沿，也不会写入数据
3. 改变写入地址，向 A 号寄存器写入 `32'h5a5a5a5a`
4. 写入使能信号失效，读取 5 号寄存器，读取 A 号寄存器，发现读取的数据是正确的，第 2 步中设置的小脉冲并没有干扰到数据
5. 当写入使能信号失效时，我们想要向 A 号写入 `32'habcdedcb`，无法写入，此时仍然读取 A 号寄存器，发现数据没有变化
6. 写入使能信号使能，向 0 号寄存器写入 `32'hffffffff`，读取 0 号寄存器，发现数据仍然是 0，验证了 0 号寄存器无法改变值

TruthEvaluator **设计**

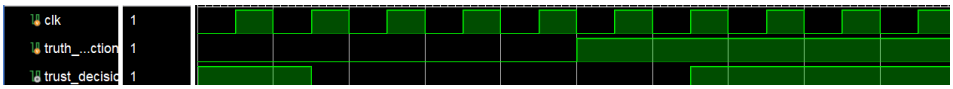

图 5: TruthEvaluator 仿真结果

一开始，状态为 HIGHLY_TRUSTWORTHY，输出结果为 1 可信，此时我们让输入为 0，一个时钟周期后，状态变为 TRUSTWORTHY，输出结果为 1，仍然可信，再过一个时钟周期，状态变为 SUSPICIOUS，输出结果为 0，不可信，再过一个时钟周期，状态变为 UNTRUSTWORTHY，输出结果为 0，不可信，再过一个时钟周期，仍然还是 UNTRUSTWORTHY，输出结果为 0，不可信；将输入变为 1 后，一开始为 UNTRUSTWORTHY，输出结果为 0，不可信，再过一个时钟周期，状态变为 SUSPICIOUS，输出结果为 0，不可信，再过一个时钟周期，状态变为 TRUSTWORTHY，输出结果为 1，可信，再过一个时钟周期，状态变为 HIGHLY_TRUSTWORTHY，输出结果为 1，可信，再过一个时钟周期，仍然为 HIGHLY_TRUSTWORTHY，输出结果为 1，可信，仿真结束。

三、讨论与实验心得

Lab1 的实验原理都比较简单, 代码的实现难度也不是很高，但是由于上学期计逻对于 Verilog 的一知半解，以及一个暑假以来没有动手写过 Verilog 的情况，在实验开始之前，我先整理了一下 Verilog 的基本语法，然后再开始写代码，比较顺利，但是在写 Regfile 的时候，一开始我想的是从 00 到 31 全部都用新定义的 Reg_file 来存储，所以一开始对于 Reg00 恒为 0 的情况，我写的是

```
assign Reg_file[0] = 32'h00000000;
assign Reg00 = Reg_file[0];
```

但是这样写会报错，因为 Reg00 是一个 reg 类型的变量，不能用连续赋值语句 assgin，所以要加上 always 块，但是这样又比较麻烦，所以我干脆直接去掉这一部分，直接将 0 接到 Reg00 上，让 Reg_file 从 1 开始，就会比较简洁；

在有限状态机的设计部分，一开始我觉得会比较难，因为我一开始想的是需要用到 D-flipflop 来设计状态再连线，但是后面我发现可以直接进行行为级的设计，再加上实验文档中提供的三段式的书写方法，让我无痛完成了这一部分。

总的来说，这次实验让我对于 Verilog 的语法有了更深的了解，也让锻炼了我自己设计仿真的能力，希望在以后的实验中能够更加顺利。

思考题

ALU 思考题

猜想造成不正常算数移位是因为三目运算符的存在, 一开始, 我的猜想是三目运算符中某个表达式的值是无符号的导致最后的结果会变成有符号的，所以，我一开始的修改是

```
assign res = //(ALU_operation == 4'd6) ? A >> B :
 ↪      // SRL Logic
(ALU_operation == 4'd7) ? ($signed(A) >>> $signed(B)) :         // SRA
 ↪  Arithmetic
32'd0;
```

将第一部分无符号的运算去掉，但是还是不行, 仍然会出现算术移位的错误，这时我发现 32'd0 是无符号的，所以我将 32'd0 改为 32'sd0，这样就正常了。

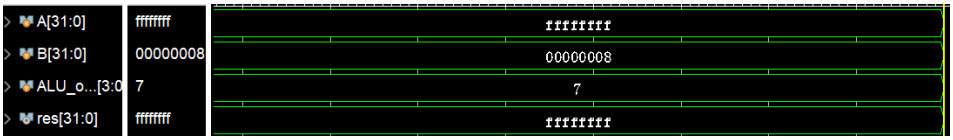

图 6: 测试结果

此时我再尝试加上第一部分无符号的运算

```
assign res = (ALU_operation == 4'd6) ? A >> B :
 ↪    // SRL Logic
(ALU_operation == 4'd7) ? ($signed(A) >>> $signed(B)) :         // SRA
 ↪  Arithmetic
32'sd0;
```

发现还是不行，所以我猜想三目运算符中对于有符号无符号的判断是只要其中一个是无符号的，那么结果就是无符号的，所以我将第一部分的无符号运算改为有符号的, 这样，结果终于正确了。

我本来想在 Verilog 文档中寻找有关这一部分的解释来加以验证，但是没有找到

状态转移图

Moore 状态图如下

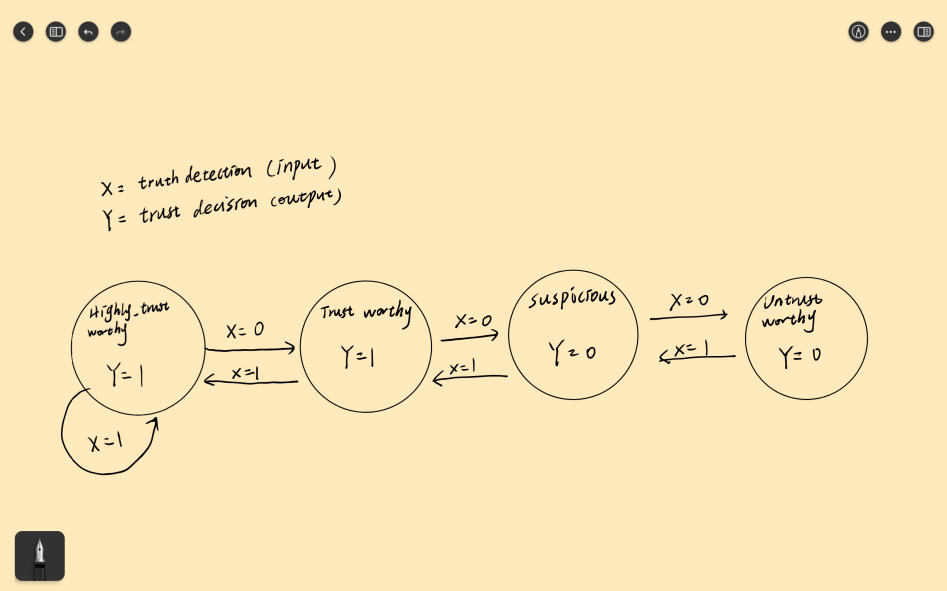

图 7: Moore 状态图