计算机组成原理第 4 次实验报告

实验名称：ALU 模块实现

学号：2313546　　姓名：蒋枘言　　班次：1078

一、实验目的

1.熟悉 MIPS 指令集中的运算指令，学会对这些指令进行归纳分类。
2.了解 MIPS 指令结构。
3.熟悉并掌握 ALU 的原理、功能和设计。
4.进一步加强运用 verilog 语言进行电路设计的能力。
5.为后续设计 CPU 的实验打下基础。

二、实验内容说明

请根据实验指导手册中完成实验四 ALU 实验的代码功能验证，并做以下改进：

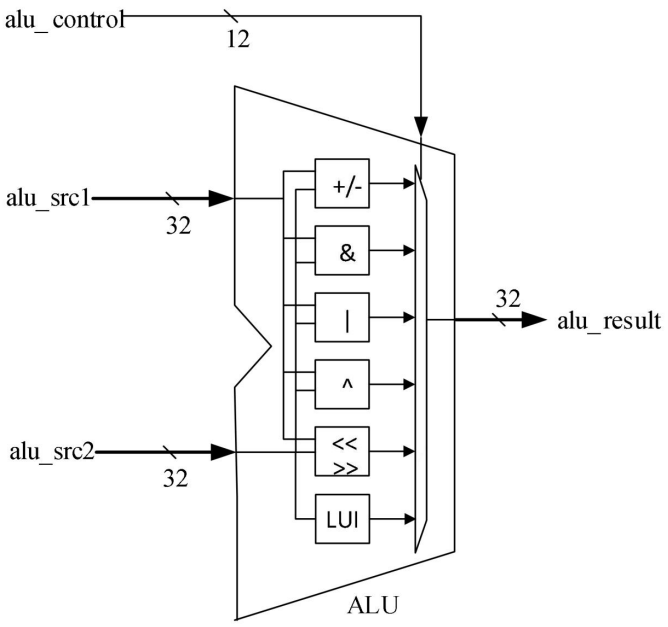
1.原有的操作码为 12 位，请压缩到操作码控制型号位宽为 4 位。

2.在操作码调整到 4 位之后，应该能支持 15 种不同运算，请添加至少三种运算功能（有符号数比较和无符号数比较的大于置位运算、一种未实现的位运算），然后上实验箱验证（可以不用仿真）。

3.实验报告中的原理图就用图 5.3 即可，不再是顶层模块图。

4.添加的三种运算上箱需要有验证照片。

三、实验原理图



四、实验步骤

首先我们需要把操作码压缩至 4 位，然后新增 3 种运算：①**有符号数比较的大于置位运算**；②**无符号数比较的大于置位运算**；③**逻辑左移取反**（自定义的）。需要对代码作出以下修改。

（1）alu.v 的代码如下：

```
`timescale 1ns / 1ps
module alu(
    input  [3:0]  alu_control,  // 4 位操作码，支持 16 种操作（0000 保留无操作）
    input  [31:0] alu_src1,     // 操作数 1（补码）
    input  [31:0] alu_src2,     // 操作数 2（补码）
    output [31:0] alu_result    // 计算结果
);

    // 定义操作码（4 位编码）
    localparam OP_ADD  = 4'b0001;  // 加法
    localparam OP_SUB  = 4'b0010;  // 减法
    localparam OP_SLT  = 4'b0011;  // 有符号小于置位
    localparam OP_SLTU = 4'b0100;  // 无符号小于置位
    localparam OP_AND  = 4'b0101;  // 按位与
    localparam OP_NOR  = 4'b0110;  // 按位或非
    localparam OP_OR   = 4'b0111;  // 按位或
    localparam OP_XOR  = 4'b1000;  // 按位异或
    localparam OP_SLL  = 4'b1001;  // 逻辑左移
    localparam OP_SRL  = 4'b1010;  // 逻辑右移
    localparam OP_SRA  = 4'b1011;  // 算术右移
    localparam OP_LUI  = 4'b1100;  // 高位加载
    localparam OP_SGT  = 4'b1101;  // 新增：有符号大于置位
    localparam OP_SGTU = 4'b1110;  // 新增：无符号大于置位
    localparam OP_SLLN = 4'b1111;  // 新增：逻辑左移后取反

    // 共用加法器模块
    wire [31:0] adder_operand2 = (alu_control == OP_ADD) ? alu_src2 : ~alu_src2;
    wire        adder_cin      = (alu_control == OP_ADD) ? 1'b0 : 1'b1;
    wire [32:0] adder_result   = alu_src1 + adder_operand2 + adder_cin;

    // 比较运算结果
    wire        slt  = (alu_src1[31] ^ alu_src2[31]) ? alu_src1[31] : adder_result[31];
    wire        sltu = ~adder_result[32];
    wire        sgt  = (alu_src1[31] ^ alu_src2[31]) ? ~alu_src1[31] : ~adder_result[31];
    wire        sgtu = adder_result[32];

    // 移位器逻辑
```

```
    wire [4:0]    shf = alu_src1[4:0];
    wire [31:0] sll_result = alu_src2 << shf;
    wire [31:0] srl_result = alu_src2 >> shf;
    wire [31:0] sra_result = $signed(alu_src2) >>> shf;
    wire [31:0] slln_result = ~sll_result;    // 新增：左移后取反

    // 根据操作码选择结果
    assign alu_result =
        (alu_control == OP_ADD)    ? adder_result[31:0] :
        (alu_control == OP_SUB)    ? adder_result[31:0] :
        (alu_control == OP_SLT)    ? {31'd0, slt} :
        (alu_control == OP_SLTU) ? {31'd0, sltu} :
        (alu_control == OP_AND)    ? (alu_src1 & alu_src2) :
        (alu_control == OP_NOR)    ? ~(alu_src1 | alu_src2) :
        (alu_control == OP_OR)     ? (alu_src1 | alu_src2) :
        (alu_control == OP_XOR)    ? (alu_src1 ^ alu_src2) :
        (alu_control == OP_SLL)    ? sll_result :
        (alu_control == OP_SRL)    ? srl_result :
        (alu_control == OP_SRA)    ? sra_result :
        (alu_control == OP_LUI)    ? {alu_src2[15:0], 16'd0} :
        (alu_control == OP_SGT)    ? {31'd0, sgt} :              // 新增：有符号大于
        (alu_control == OP_SGTU) ? {31'd0, sgtu} :             // 新增：无符号大于
        (alu_control == OP_SLLN) ? slln_result :                // 新增：左移取反
        32'd0;

endmodule
```

（2）alu_display.v 的代码如下（仅给出修改的关键部分）：

```
module alu_display(
    // ...（原有端口不变）
);

    // 修改控制信号位宽为 4 位
    reg     [3:0]    alu_control;     // 4 位操作码
    reg     [31:0] alu_src1;
    reg     [31:0] alu_src2;
    wire    [31:0] alu_result;

    // 输入处理逻辑修改（仅展示修改的关键部分）
    always @(posedge clk) begin
        if (!resetn) begin
            alu_control <= 4'd0;
        end else if (input_valid && input_sel == 2'b00) begin
            alu_control <= input_value[3:0];    // 仅取低 4 位
```

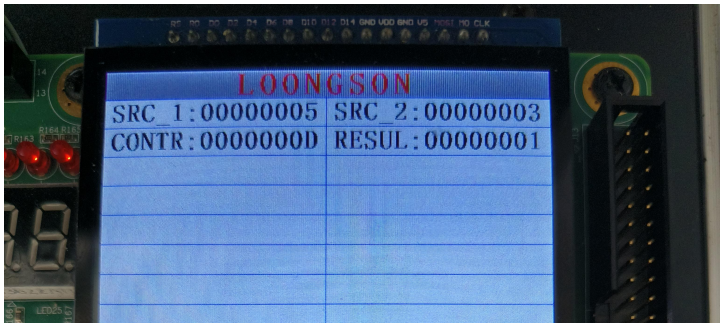
```
        end
    end

    // 显示部分修改（操作码扩展为 32 位显示）
    always @(posedge clk) begin
        case(display_number)
            6'd3 :  // 显示控制信号
            begin
                display_valid <= 1'b1;
                display_name  <= "CONTR";
                display_value <= {28'd0, alu_control};  // 4 位扩展为 32 位
            end
            // ...（其他显示逻辑不变）
        endcase
    end

endmodule
```

五、实验结果分析

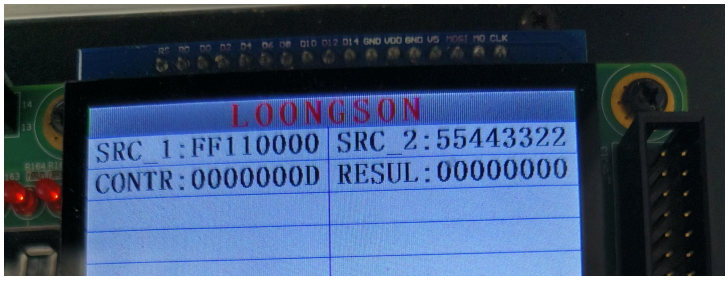
在实验箱上验证新增的 3 种运算：

（1）有符号数比较的大于置位运算：控制信号为 1101，高位补 0，八位十六进制的控制信号 CONTR 应为 **0000000D**。

取 alu_src1 = 32'h5，alu_src2 = 32'h3，alu_src1 与 alu_src2 均为有符号数，显然 alu_src1＞alu_src2，结果 RESUL 应为 1，验证正确。

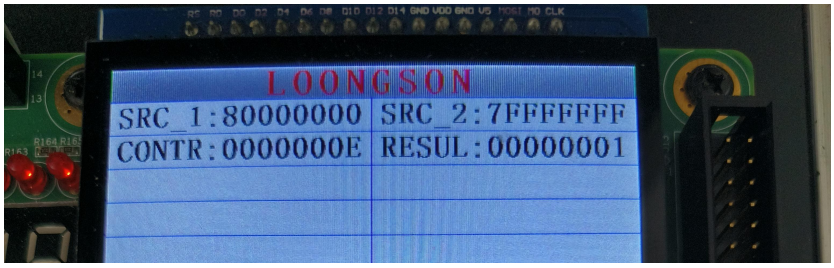


取 alu_src1 = 32'hFF110000，alu_src2 = 32'h55443322，alu_src1 与 alu_src2 均为有符号数，由于 alu_src1 为负数，alu_src2 为正数，显然 alu_src1＜alu_src2，结果 RESUL 应为 0，验证正确。



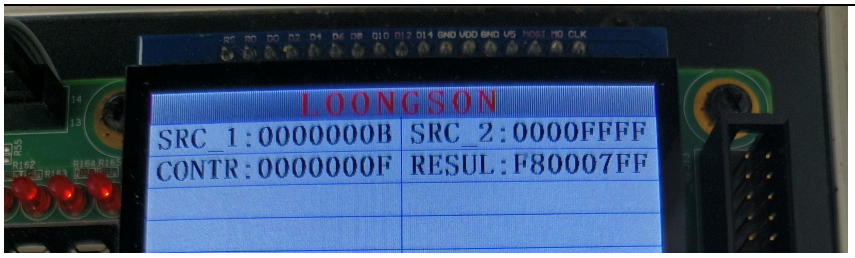
（2）无符号数比较的大于置位运算：控制信号为 1110，高位补 0，八位十六进制的控制信号 CONTR 应为 **0000000E**。

取 alu_src1 = 32'h80000000，alu_src2 = 32'h7FFFFFFF，alu_src1 与 alu_src2 均为无符号数，由于 alu_src1 的最高位为 8，alu_src2 的最高位为 7，显然 alu_src1＞alu_src2，结果 RESUL 应为 1，验证正确。



（3）逻辑左移取反运算：控制信号为 1111，高位补 0，八位十六进制的控制信号 CONTR 应为 **0000000F**。

取 alu_src1 = 32'h8000000B，alu_src2 = 32'h0000FFFF，alu_src1 表示的是左移的位数，这里左移 11 位（十六进制数 B），alu_src2 表示的是源操作数，这里的二进制表示是 0000 0000 0000 0000 1111 1111 1111 1111，左移 11 位得到 0000 0111 1111 1111 1111 1000 0000 0000，取反得到 1111 1000 0000 0000 0000 0111 1111 1111，十六进制为 F80007FF，验证结果正确。



六、总结感想

通过本次实验，我深入理解了 ALU 的功能设计与硬件实现逻辑，掌握了操作码压缩与功能扩展的方法。在 Verilog 代码重构过程中，进一步熟悉了硬件描述语言的模块化设计思维，并通过实验箱验证了新增运算的正确性，增强了硬件调试能力。新增的有符号/无符号大于比较及逻辑左移取反运算，让我对位运算和比较逻辑有了更直观的认识。此次实践为后续 CPU 设计奠定了扎实基础。