组成原理实验课程第_四_次实报告

实验名称	ALU 模块实现			班级	李涛老师
学生姓名	王昱	学号	2212046	指导老师	董前琨
实验地点	A306		实验时间	2024.5.16	

1、 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。

2、 实验内容说明

1、原始代码中只有表 5.1 中的 11 种运算，请另外补充至少三种不同类型运算（比较运算、位运算、数据加载运算等）。

2、实验报告中原理图参考 5.3，只画出补充的运算部分即可，报告中展示的结果应位补充运算的计算结果。

3、本次实验报告不需要有仿真结果，但需要有实验箱上箱验证的照片，同样，针对照片中的数据需要解释说明。

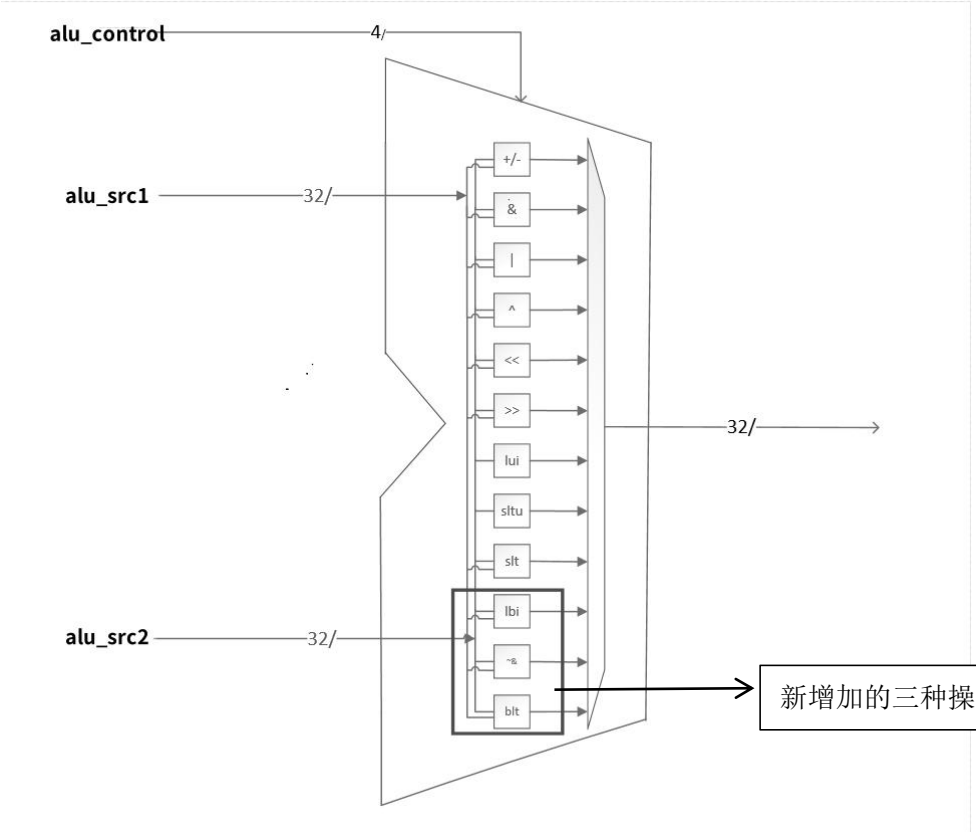
4、实验报告模板参考百度云盘文件，注意提交截至时间为 5 月 18 日下午 18：00。

5.具体说明：

完成对 ALU 运算器的改进，要求：

- 压缩 ALU 运算器的控制信号位宽 alu_control 调整到 4 位
- 扩展 ALU 运算器的运算数量补充三个：
    1. 无符号数比较，大于置位
    2. 按位与非
    3. 低位加载
- 上箱验证。

3、 实验原理图



实验原理图说明：

如图所示，实验原理图在原有的操作上新增加了三种框出来的操作：

- lbi（低位加载），对应 ALU 控制信号为 D，这个操作用于将立即数加载到寄存器的低位部分，而高位部分通常置为 0。
- ~&（按位与非），对应 ALU 控制信号为 E，对两个操作数按位与，然后对结果按位取反。
- Blt（无符号比较大于置位），对应 ALU 控制信号为 F，对两个操作数进行比较，若 alu_src1>alu_src2 则 result 为 1,否则为 0。

4、 实验步骤

（一）修改 alu.v 模块：

（1）控制关系

实验要求将控制信号缩减为四位，用四位的控制信号来控制执行什么样的输出。

首先将 alu_control 拓展为四位位宽：

```
module alu(
    input  [3:0] alu_control,  // ALU 控制信号
    input  [31:0] alu_src1,    // ALU 操作数 1,为补码
    input  [31:0] alu_src2,    // ALU 操作数 2，为补码
    output [31:0] alu_result   // ALU 结果
```

```
);
```

修改独热码，增加三个新的控制信号：

```
wire [31:0] add_sub_result;
wire [31:0] slt_result;
wire [31:0] sltu_result;
wire [31:0] and_result;
wire [31:0] nor_result;
wire [31:0] or_result;
wire [31:0] xor_result;
wire [31:0] sll_result;
wire [31:0] srl_result;
wire [31:0] sra_result;
wire [31:0] lui_result;
wire [31:0] hui_result;
wire [31:0] blt_result;
wire [31:0] bltu_result;
wire [31:0] nand_result;
```

用四位 alu_control 来控制输出什么样的结果,代码中通过一系列 wire 定义了不同操作的结果信号。每个操作都有一个独特的 4 位控制信号。当 alu_control 信号与某个操作的控制信号匹配时，对应的操作被执行，并计算结果。通过一系列 assign 语句，根据 alu_control 信号选择相应的结果并赋值给 alu_result:

```
assign alu_result = (alu_control==4'b0001|alu_control==4'b0010) ? add_sub_result[31:0] :
                        alu_control==4'b0011          ? slt_result :
                        alu_control==4'b0100         ? sltu_result :
                        alu_control==4'b0101          ? and_result :
                        alu_control==4'b0110          ? nor_result :
                        alu_control==4'b0111          ? or_result :
                        alu_control==4'b1000          ? xor_result :
                        alu_control==4'b1001          ? sll_result :
                        alu_control==4'b1010          ? srl_result :
                        alu_control==4'b1011          ? sra_result :
                        alu_control==4'b1100          ? lui_result :
                        alu_control==4'b1101         ? hui_result :
                        alu_control==4'b1110          ?nand_result :
                        alu_control==4'b1111          ?bltu_result :
                        32'd0;
```

控制信号与对应操作表如下所示：

CONTR	ALU操作
0	无
1	加法
2	减法
3	有符号比较，小于置位
4	无符号比较，小于置位
5	按位与
6	按位或非
7	按位或
8	按位异或
9	逻辑左移
A	逻辑右移
B	算术右移
C	高位加载
D	低位加载
E	按位与非
F	无符号比较，大于置位

（2）编写新增的操作：

- **编写低位加载**

```
assign hui_result = {16'd0,alu_src2[31:16]};
```

低位加载的结果 hui_result 是通过 assign hui_result = {16'd0, alu_src2[15:0]}; 实现的，这意味着将 alu_src2 的低 16 位加载到结果的低 16 位，高 16 位清零。

在最终结果选择时，通过 alu_control == 4'b1101 来选择 hui_result，实现低位加载操作。

- **编写按位与非**

```
assign nand_result = ~(alu_src1 & alu_src2);  // 按位与非结果
```

这样就实现了按位与非的操作，并且通过控制信号 alu_control == 4'b1110 来选择执行按位与非的操作。

- **编写无符号比较大于置位：**

```
assign bltu_result = {31'd0, adder_cout};//大于则置位
```

这里只需要把小于置位中的 adder_cout 取反即可。adder_cout 是通过 adder 模块中的加法操作实现的，其中 cout 表示在加法或减法运算中是否产生了进位。无符号比较大于置位就是利用了这一进位信息来判断 alu_src1 是否大于 alu_src2。

如果 adder_cout 为 1，则表示 alu_src1 大于 alu_src2，bltu_result 的最低位为 1，表示无符号数比较中 alu_src1 大于 alu_src2。

```
adder adder_module(
.operand1(adder_operand1),
.operand2(adder_operand2),
.cin     (adder_cin     ),
.result  (adder_result  ),
.cout    (adder_cout    )
);
```

如图，在这里会调用 adder 加法器模块，通过取反进行减法操作，输出 adder_cout。

（二）修改 display.v

(1)修改控制信号位宽

```
reg   [3:0] alu_control;  // ALU 控制信号   //ALU 控制信号
reg   [31:0] alu_src1;     // ALU 操作数 1
reg   [31:0] alu_src2;     // ALU 操作数 2
wire  [31:0] alu_result;   // ALU 结果
```

将控制信号从十二位修改为四位。

(2)修改控制信号写入

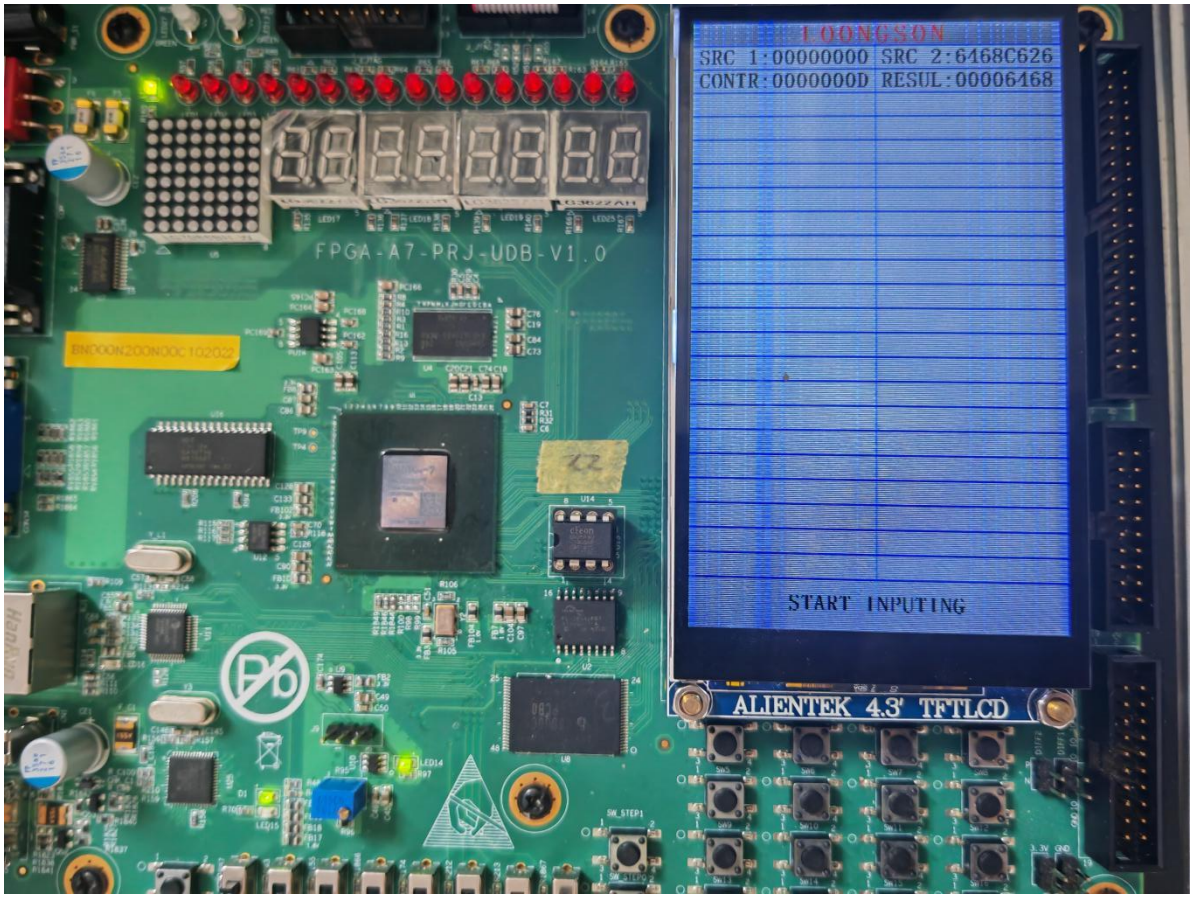
```
always @(posedge clk)
begin
    if (!resetn)
    begin
        alu_control <= 12'd0;
    end
    else if (input_valid && input_sel==2'b00)
    begin
        alu_control <= input_value[3:0];
    end
end
```

同样将这里修改为四位，前面用 28 个 0 填充

```
            6'd3 :
            begin
                display_valid <= 1'b1;
                display_name  <= "CONTR";
                display_value <={28'd0, alu_control};
            end
```

5.实验结果分析

（1）低位加载：



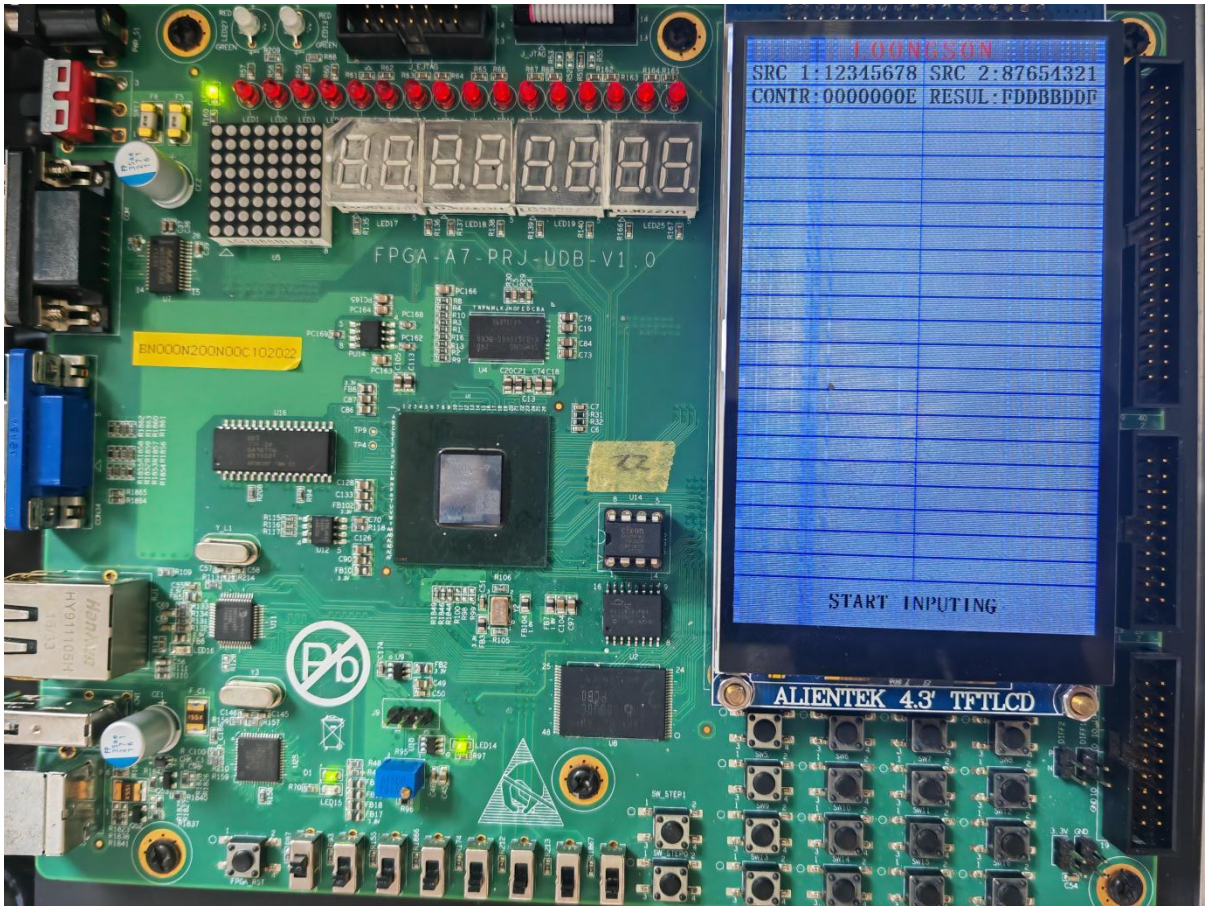
通过拨码开关，分别输入 CONTR，SRC1.,SRC2，其中 CONTR 为 D（1101），对应低位加载操作：

SRC1=00000000

SRC2=6468C626

输出结果 RESUL 为 00006468，把源操作数的高十六位加载到低位，符合低位加载，结果正确。

（2）按位与非



通过拨码开关，分别输入 CONTR，SRC1.,SRC2，其中 CONTR 为 E（1110），对应按位与非操作：

SRC1=12345678,SRC2=87654321,

SRC1=0001 0010 0011 0100 0101 0110 0111 1000

SRC2=1000 0111 0110 0101 0100 0011 0010 0001

按位与得 0000 0010 0010 0100 0100 0010 0010 0000

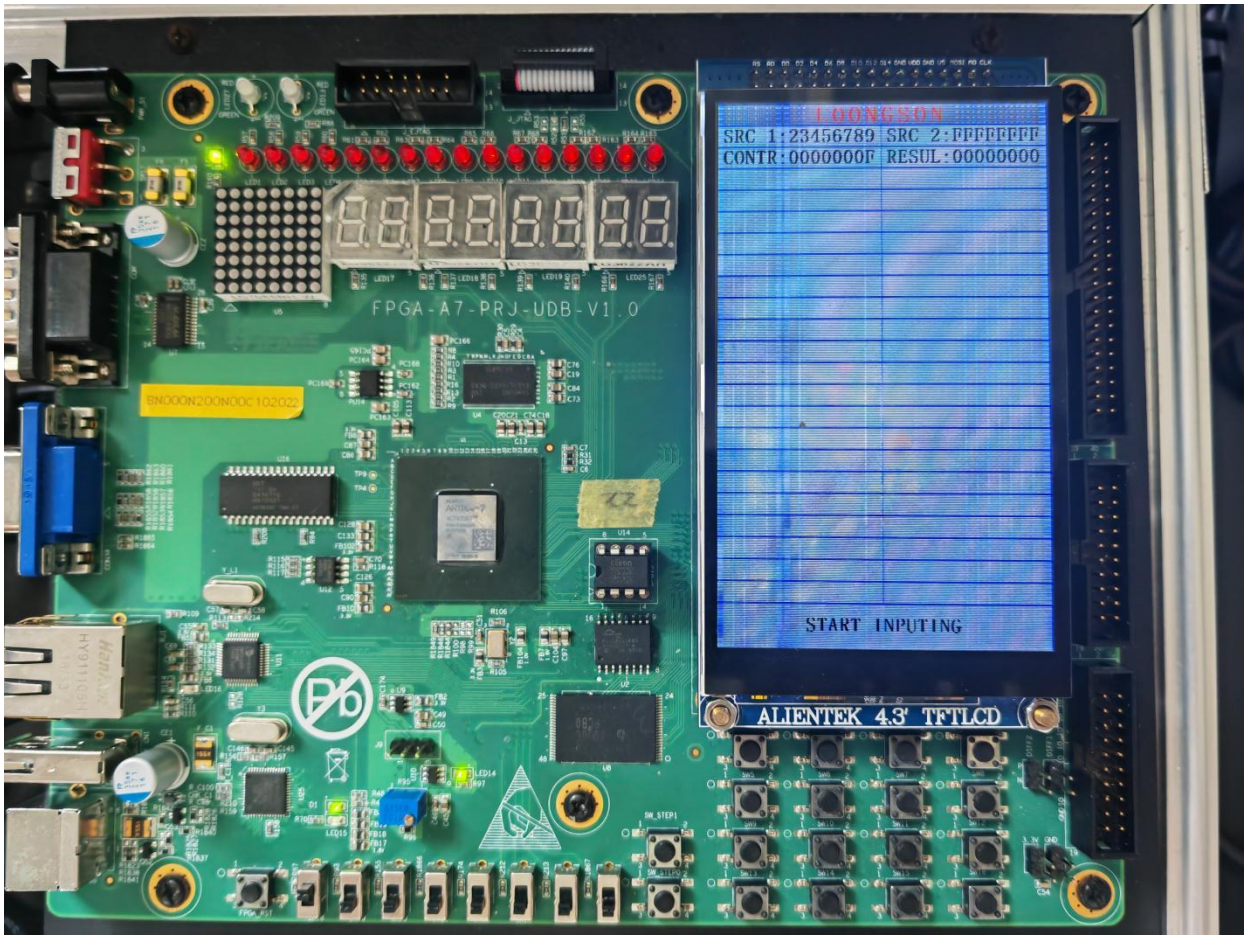
按位非得 RESULT=1111 1101 1101 1011 1011 1101 1101 1111

即 RESULT=FDDBBDDF

输出结果 RESUL 为 FDDBBDDF，说明验证结果正确。

（3）无符号比较大于置位

小于的情况：



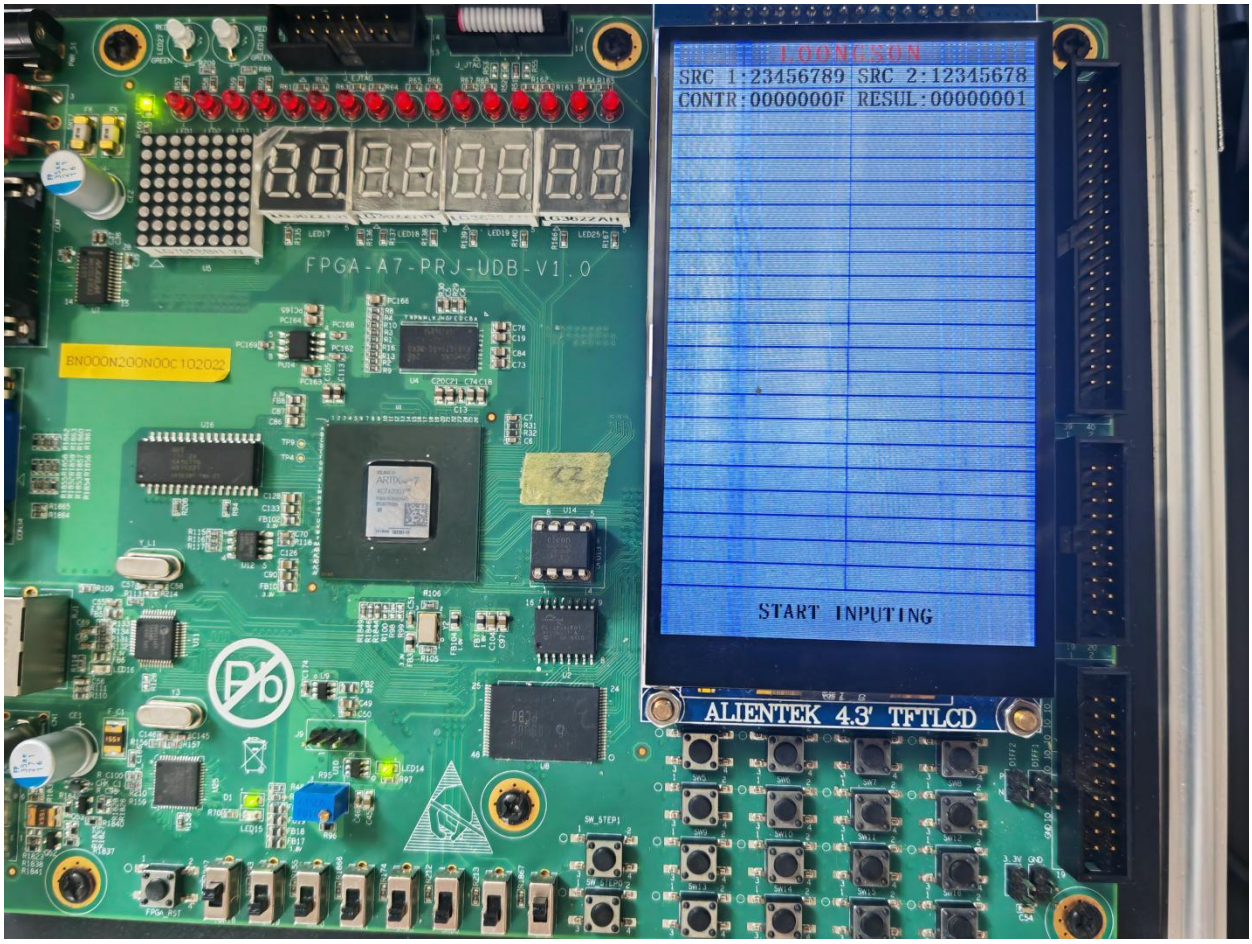
通过拨码开关，分别输入 CONTR，SRC1.,SRC2，其中 CONTR 为 F（1111），对应无符号比较大于置位操作：

SRC1=23456789,SRC2=FFFFFFFF,

由于是无符号比较，所以 SRC2 代表最大的数而不是负数，因此 SRC1<SRC2，即 RESULT=0.

输出结果 RESUL 为 00000000，说明验证结果正确。

大于的情况：



通过拨码开关，分别输入 CONTR，SRC1.,SRC2，其中 CONTR 为 F（1111），对应无符号比较大于置位操作：

SRC1=23456789,SRC2=12345678,

SRC1》SRC2，即 RESULT=1.

输出结果 RESUL 为 00000001，说明验证结果正确。

6.总结感想

1.难度与挑战

本次实验难度较高，需要对所给代码的各个部分进行认真阅读和理解。实验内容复杂，需要一定的耐心和细致的分析，才能顺利完成。

2.前后实验联系

本次实验与前面的实验联系紧密，完成本次实验的前提是已经认真完成了之前的实验。这种连贯性要求我们在每一步都认真对待，确保对每个环节都有充分的理解。

3.收获与提高

通过本次实验，我对整个程序的流程以及实验箱的结构有了更加深入的理解。同时，进一步熟悉了 Verilog 代码的编写。掌握了 ALU 模块的实现方式，并了解了不同运算的实现过程，这些知识为之后的 CPU 设计打下了坚实的基础。

总的来说，本次实验不仅提升了我的编程能力，也加深了我对计算机硬件设计的理解，为未来更复杂的实验做了良好的铺垫。