组成原理实验课程第4次实验报告

实验名称： ALU 模块实现

学生姓名：许洋 学号：2313721 班级：张金老师

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

针对组成原理第四次的ALU实验进行改进，要求：

1、将原有的操作码进行位压缩，调整操作码控制信号位宽为4位。

2、操作码调整成4位之后，在原有11种运算的基础之上，补充3种不同类型的运算（要求一种大于置位比较，一种位运算，一种自选），需要上实验箱或仿真验证计算结果。

3、注意改进实验上实验箱验证时，操作码应该已经压缩到4位位宽。

4、实验报告中的原理图就用图5.3即可，不再是顶层模块图。实验报告中讲清楚添加的三种运算过程，还需要附上实验箱验证照片。

三、实验原理图

ALU 模块的原理图如下：

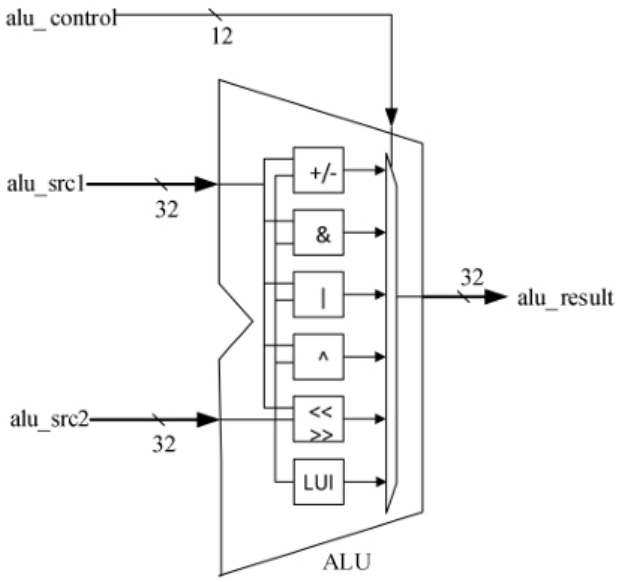


图 5.3 ALU 的原理图

实验原理

控制关系

由于我们要将控制信号缩减为4位，用以表示15种运算，因此需要重新设计控制信号及其ALU操作。改后的ALU控制关系及对应操作如下图（新增了三种操作：大于置位、算术左移、低位加载）：

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

CONTR	ALU操作
B	算数右移
C	高位加载
D	**有符号比较，大于置位**
E	**按位同或**
F	**低位加载**

实现原理（原12个）

- 加减法：

  调用实验一的加法器实现。

- 有符号比较的小于置位、无符号比较的小于置位：

源操作数 1 符号位		源操作数 2 符号位		结果符号位		判断	slt 结果
alu_src1[31]		alu_src2[31]		adder_result[31]			
0	正数	0	正数	0	正数	正>正	0
0	正数	0	正数	1	负数	正<正	1
0	正数	1	负数	X	无关	正>负	0
1	负数	0	正数	X	无关	负<正	1
1	负数	1	负数	0	正数	负>负	0
1	负数	1	负数	1	负数	负<负	1

根据真值表，得到有符号 32 位比较小于置位运算结果表达式：

```
slt_result = (alu_src1[31] & ~alu_src2[31]) | (~
(alu_src1[31]^alu_src2[31]) & adder_result[31])
```

对于 32 位无符号比较的小于置位，可在其高位前填 0 组合为 33 位正数的比较，即{1’b0, src1}和{1’b0, src2}的比较，最高位符号位为 0。对比上表可知，对于正数的比较，只要减法结果的符号位为 1，则表示小于。而 33 位正数相减，其结果的符号位最终可由 32 位加法的 cout+1’b1 得到。故无符号 32 位比较小于置位运算结果表达式为：sltu_result = ~adder_cout.

- 按位与、按位或非、按位或、按位异或：

  直接使用 verilog 运算函数（&、|、^）实现。

- 逻辑左移、逻辑右移、算数右移：

  移位分三步进行：

  第一步根据移位量低2位，即 [1:0] 位，做第一次移位；

第二步在第一次移位基础上根据移位量 [3:2] 位做第二次移位；

第三步在第二次移位基础上根据移位量[4]位做第三次移位。

- 高位加载：

  专门用于设置寄存器中常数的高16位置。将op2低16位存放到寄存器的高16位，结果的低16位用0填充。

实现原理（新增三个）

- 有符号比较，大于置位：

  相比于有符号比较的小于置位，在改为大于置位的时候，需要考虑等于的情况，因此我们增设一个adder_zero值，用于排除相等的情况。其对应真值表如下图。

源操作数1符号位		源操作数2符号位		结果符号位			判断	sgt结果
alu_src1[31]		alu_src1[31]		adder_result[31]		adder_zero		
0	正数	0	正数	0	正数	0	正=正	0
0	正数	0	正数	0	正数	1	正>正	1
0	正数	0	正数	1	负数	1	正<正	0
0	正数	1	负数	X	无关	1	正>负	1
1	负数	0	正数	X	无关	1	负<正	0
1	负数	1	负数	0	正数	0	负=负	0
1	负数	1	负数	0	正数	1	负>负	1
1	负数	1	负数	1	负数	1	负<负	0

  其中，adder_zero是对结果 adder_result 进行或缩位运算，当源操作数1=源操作数2时，adder_zero为0，不置位。因此，大于置位的实现是：~（小于置位表达式）& adder_zero，即不满足小于置位且两操作数不相等的就是大于置位。

- 按位同或：

  按位异或的取反，我们只需要在我们前面的代码上加上一个取反符号即可。

- 低位加载：

  是将低16位存放到寄存器的低16位，结果的高16位用0填充。

四、实验步骤

- **修改 alu.v 文件：**

1. ALU控制信号改成4位

```
input [3:0] alu_control
```

2. 将原来的独热码改成4位二进制编码，并新增三个控制信号：

alu_sgt（有符号比较，大于置位）、alu_xnor（按位同或）、alu_hui（低位加载）

3. 有符号比较，大于则置位（sgt）：

```
wire adder_zero;
assign adder_zero=|adder_result;
assign sgt_result[31:1] = 31'd0;
assign sgt_result[0] = ~((alu_src1[31] & ~alu_src2[31]) | (~
(alu_src1[31]^alu_src2[31]) & adder_result[31])) & adder_zero;
```

4. 按位同或（xnor）

```
assign xnor_result = ~(alu_src1 ^ alu_src2);
```

在原来按位异或的基础上，进行取反。

5. 低位加载（hui）

```
assign hui_result = { 16'd0,alu_src2[15:0]};
```

- 自写 **testbench**

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
// Engineer:
//
// Create Date: 2025/04/08 10:25:45
// Design Name:
// Module Name: tb
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
```

```
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////////
///////////////


 module tb;
    // Inputs
    reg   [3:0] alu_control;  // 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
    // Outputs
    wire  [31:0] alu_result;   // ALU结果
    // Instantiate the Unit Under Test (UUT)
    alu uut (
        .alu_control(alu_control),
        .alu_src1   (alu_src1   ),
        .alu_src2   (alu_src2   ),
        .alu_result (alu_result )
    );

    initial begin
        // Initialize Inputs
        alu_control=0;
        alu_src1 = 0;
        alu_src2 = 0;
        // Wait 100 ns for global reset to finish
        #100;
        alu_control = 4'b0001;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0010;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0011;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0100;
```

```
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0101;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0110;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b0111;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1000;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1001;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1010;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1011;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1100;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1101;
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
        alu_control = 4'b1110;
        alu_src1 = $random;
        alu_src2 = $random;
```

```
        #100;
        alu_control = 4'b1111;
        //低位加载
        alu_src1 = $random;
        alu_src2 = $random;
        #100;
    end
endmodule
```

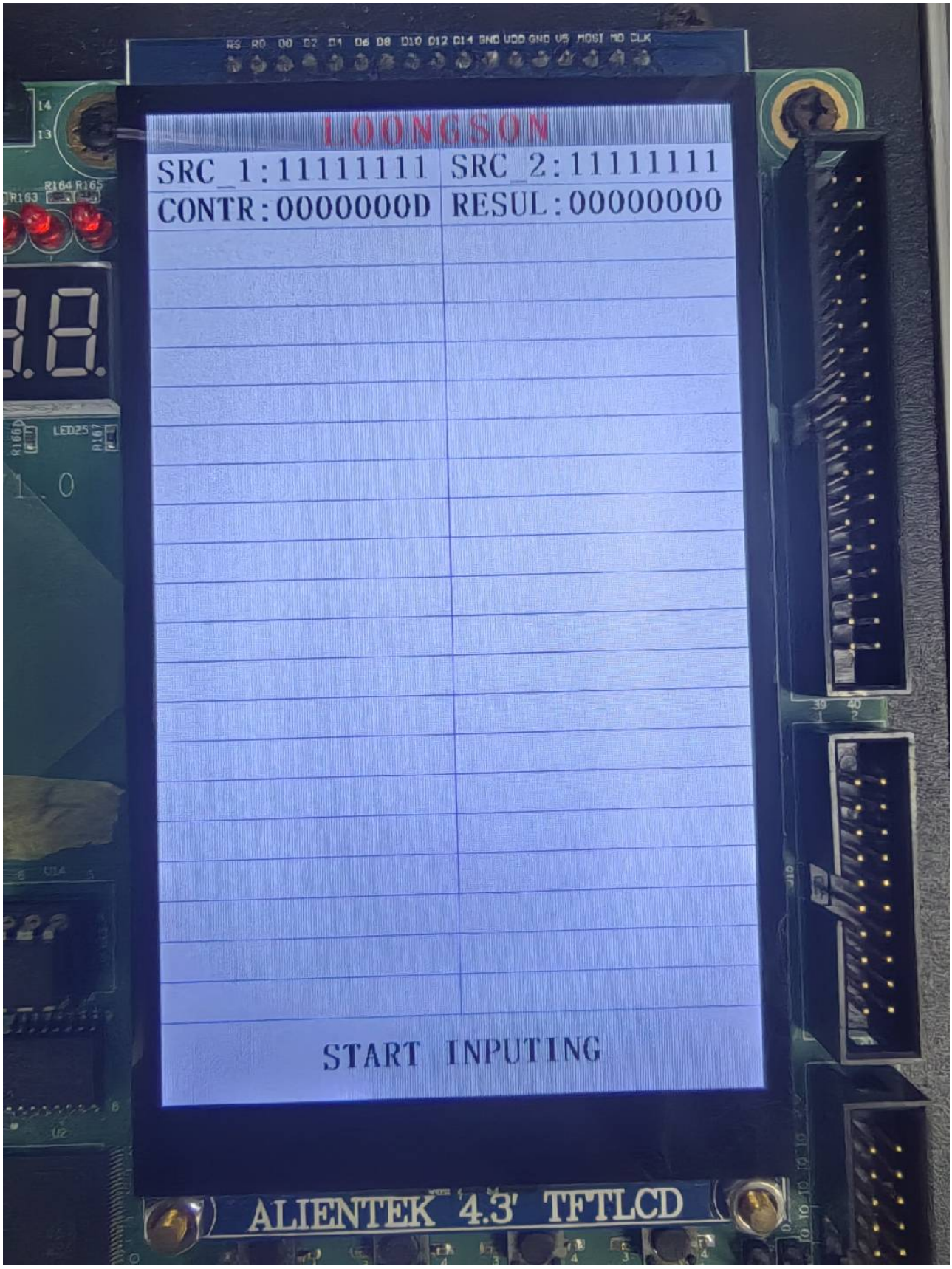
1. 两个操作数：随机获取
2. 每隔100ns，控制信号加1，进行写一个对应的ALU运算。

五、实验结果分析

原实验经过验证均没有问题，只展示新添加功能的验证。

- 有符号比较，大于置位（d）

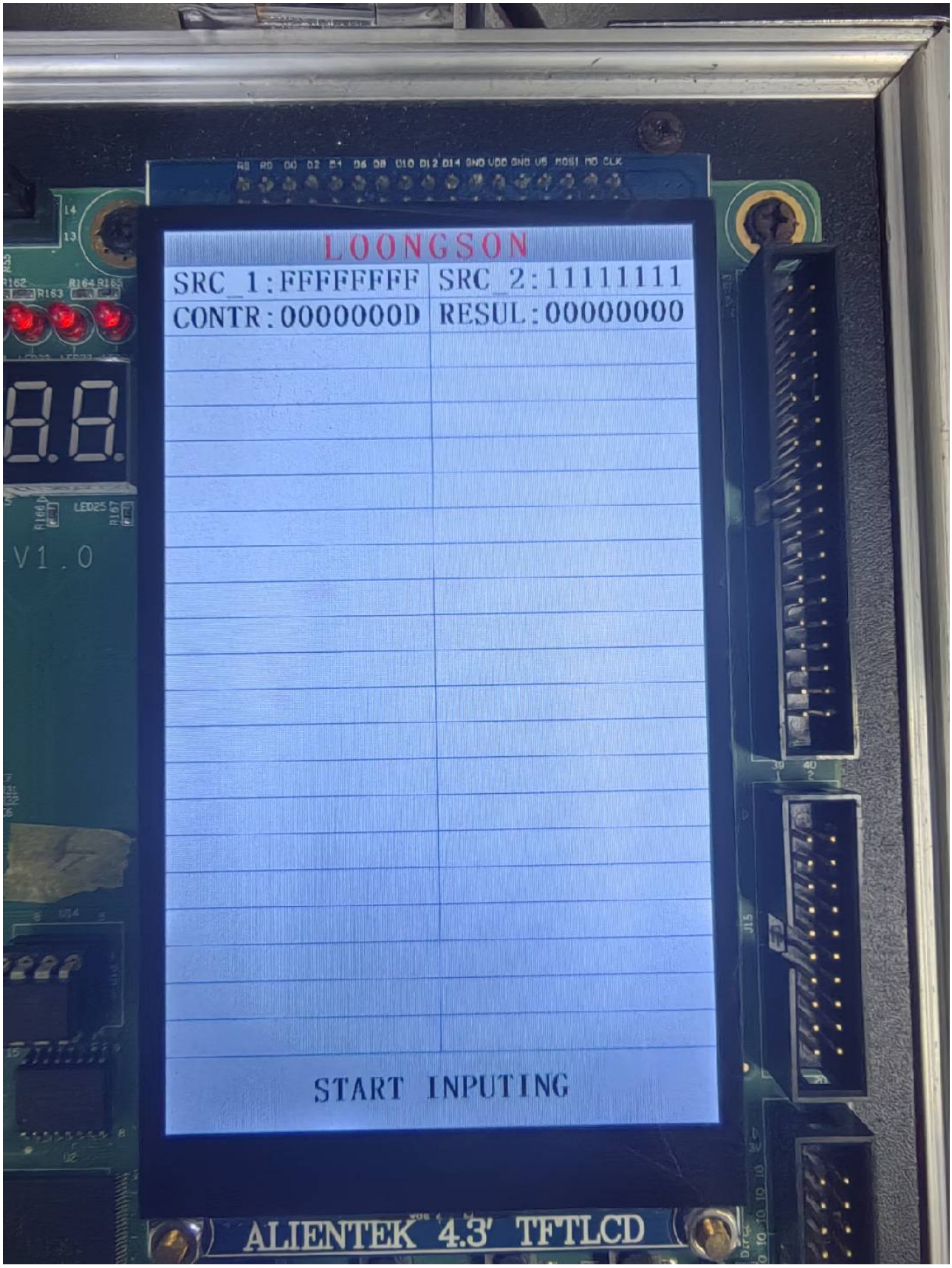
为了使得验证更加充分，我们选取了上述四个情况：



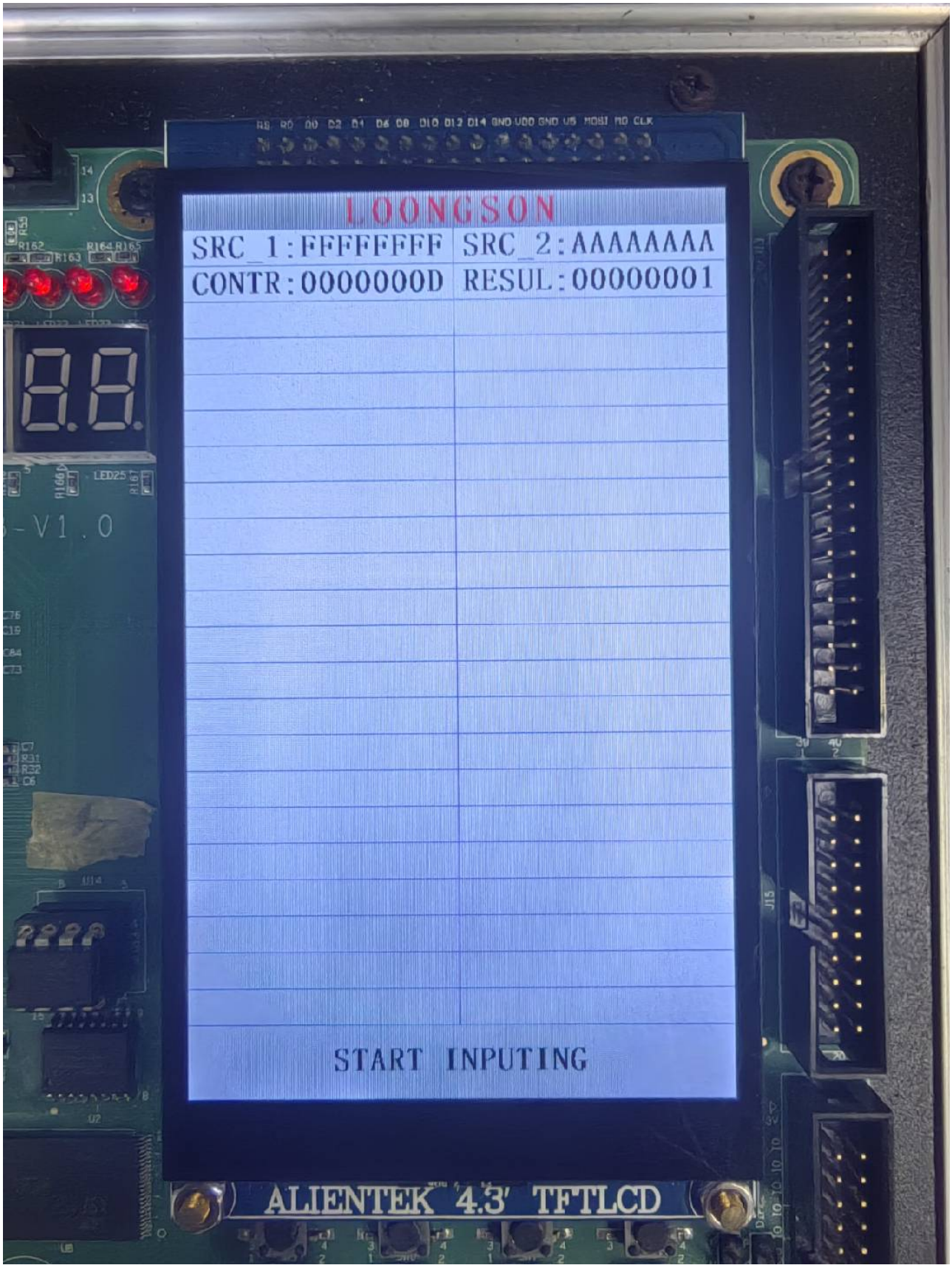
两个数相同，输出0


LOONGSON
SRC_1:22222222
SRC_2:11111111
CONTR:0000000D
RESUL:00000001
START INPUTING

两个正数比较，SRC_1>SRC_2，输出1



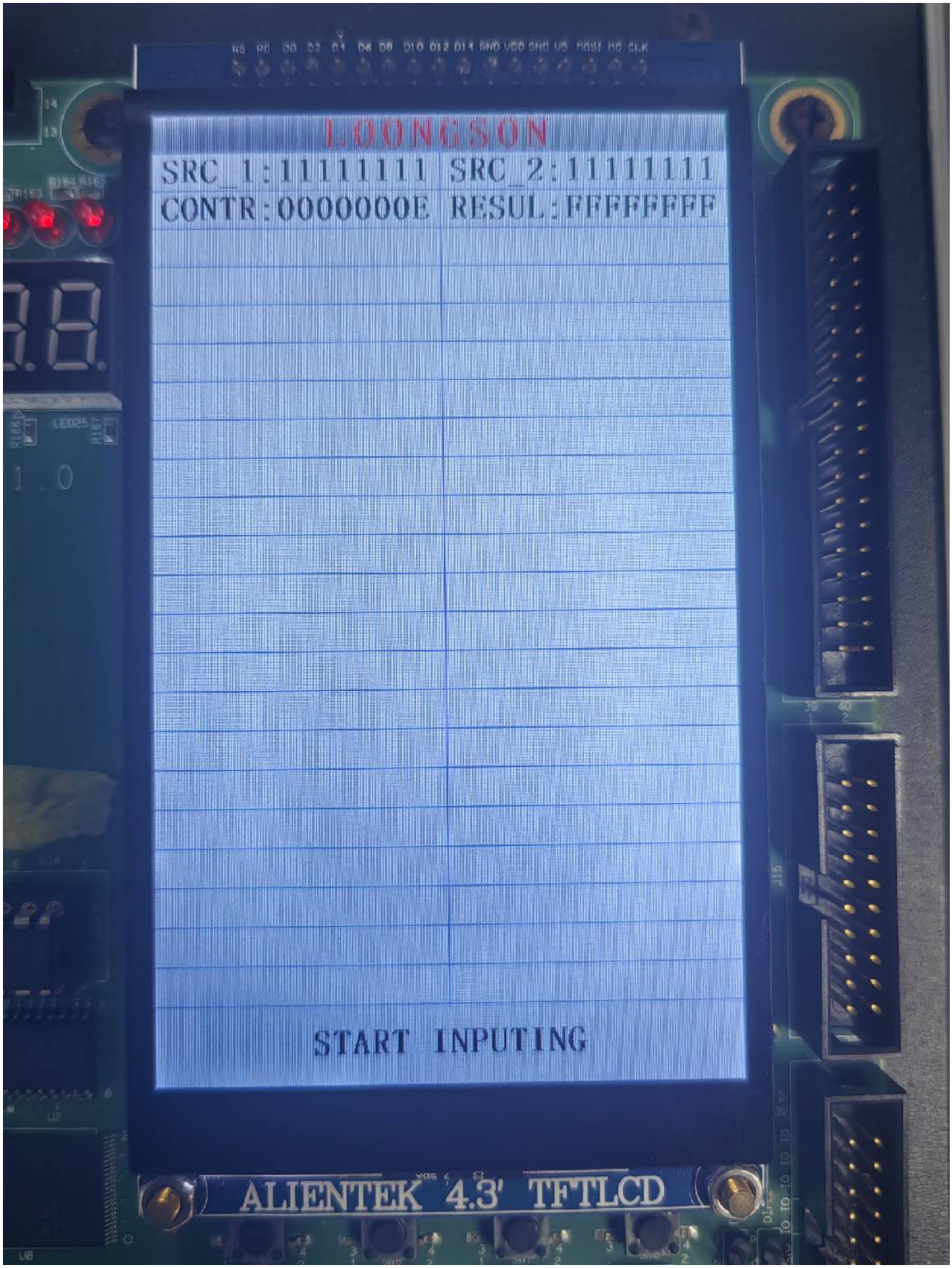
负数肯定小于正数，输出0



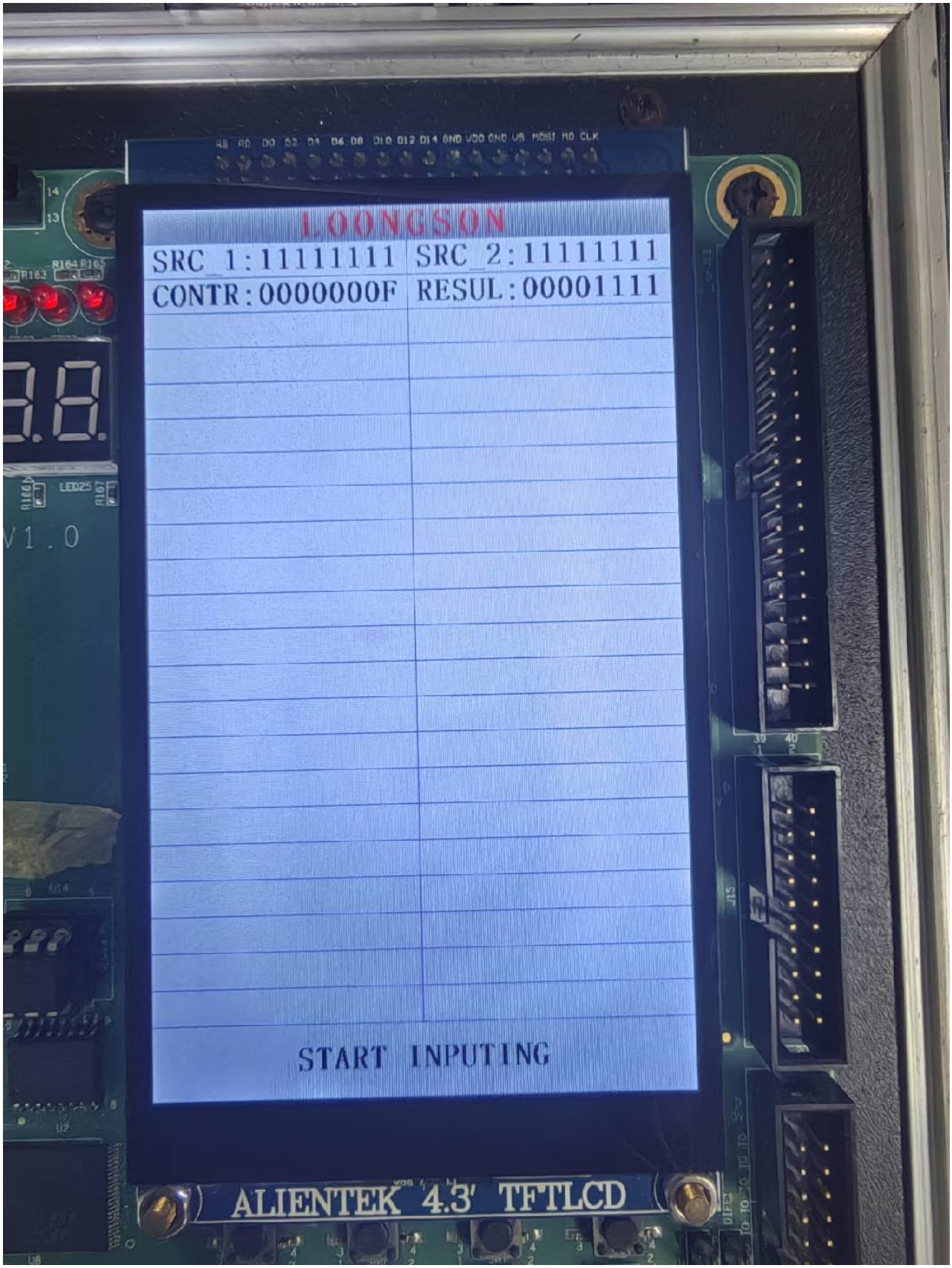
输入两个负数，一个是FFFFFFFF，一个是AAAAAAAA，应该输出的是1。

- 按位同或（e）

  我们输入SRC_1=11111111，SRC_2=11111111时，将其转换为二进制数，按位同或后，转换成十六进制数，可以得到RESUL=FFFFFFFF。



- 低位加载（f）


LOONGSON
SRC 1:11111111
SRC 2:11111111
CONTR:0000000F
RESUL:00001111
START INPUTING
ALIENTEK 4.3' TFTLCD

六、总结感想

在实验过程中，将操作码控制信号从多位压缩至 4 位，这一要求让我深刻体会到硬件资源优化的重要性。通过重新设计控制信号与 ALU 操作的对应关系，我学会了如何在有限的编码空间内合理分配功能，这种对资源利用效率的思考，在数字电路设计中至关重要。新增的三种运算功能 —— 有符号比较大于置位、按位同或和低位加载，每一种都需要深入理解其运算原理，并转化为 Verilog 代码。特别是在实现有符号比较大于置位功能时，通过分析真值表、引入辅助信号 adder_zero 来处理相等情况，这一过程让我对数字逻辑的严谨性有了更深刻的认识。

通过本次实验，我不仅掌握了 ALU 模块的设计与实现方法，而且让我对计算机组成原理有了更深入的理解。这些知识和技能将成为我未来学习和研究 CPU 设计的重要基石。