**组成原理实验课程第 四 次实验报告**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 实验名称 | ALU模块实现 | | | 班级 | 李涛老师 |
| 学生姓名 | 张洋 | 学号 | 2111460 | 指导老师 | 董前琨老师 |
| 实验地点 | 津南实验楼A306 | | 实验时间 | 5月9日 18：30 – 20：30 | |

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。

1. **实验内容说明**

1. 学习MIPS指令集，熟知指令类型，了解指令功能和编码，归纳基础的ALU运算指令。

2. 归纳确定自己本次实验中准备实现的ALU运算，要求不实现定点乘除指令和浮点运算指令，要求至少实现5种ALU运算，其中要包含加减运算，其中减法在内部要转换为加法，与加法运算共同调用实验一里自己完成的加法模块去做。

3. 设计方案中实现11种ALU运算，操作码采用独热码。

32位源操作数1

32位源操作数2

运算结果

操作码

图1 ALU模块的大致框图

4. 根据设计的实验方案，使用verilog编写相应代码。

5. 将以上设计作为一个单独的模块，设计一个外围模块去调用该模块。外围模块中需调用封装好的LCD触摸屏模块，显示ALU的两个源操作数、操作码和运算结果，并且需要利用触摸功能输入源操作数。操作码可以考虑用LCD触摸屏输入，也可以用拨码开关输入。

6. 将编写的代码进行综合布局布线，并下载到试验箱中的FPGA板子上进行演示。

7. 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。

8. 操作码调整成4位之后，在原有11种运算的基础之上，自行补充3种不同类型的运算，操作码和运算自行选择，上实验箱验证计算结果。

1. **实验原理图**

ALU模块的原理图如下：



图2 ALU的原理图

1. 加减运算原理图

加减运算最后都是调用实验一中的加法器做的：



图3 加减法运算的原理图

1. 小于置位运算

对于有符号比较的小于置位，是利用减法结果比较的，其比较的真值表如下：

表1 有符号比较小于置位的真值表

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 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 |

由表1可得有符号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。对比表1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，如图4。故无符号32位比较小于置位运算结果表达式为：sltu\_result = ~adder\_cout.



图4 无符号32位数比较的原理图

1. **实验步骤**

(1) 验证实验初始的11种运算，表中列出操作码、操作数和运算结果。

(2) 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。

(3) 操作码调整成4位之后，在原有11种运算的基础之上，自行补充3种不同类型的运算。

(4) 验证改进实验后的11+3种运算的验证，表中列出操作码、操作数和运算结果。

**改进代码（改进处标红）**

**alu.v**

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

);

// ALU控制信号，独热码

wire alu\_add; //加法操作

wire alu\_sub; //减法操作

wire alu\_slt; //有符号比较，小于置位，复用加法器做减法

wire alu\_sltu; //无符号比较，小于置位，复用加法器做减法

wire alu\_and; //按位与

wire alu\_nor; //按位或非

wire alu\_or; //按位或

wire alu\_xor; //按位异或

wire alu\_sll; //逻辑左移

wire alu\_srl; //逻辑右移

wire alu\_sra; //算术右移

wire alu\_lui; //高位加载

//添加的三种运算

wire alu\_sgt; //有符号比较，大于置位，复用加法器做减法

wire alu\_nod; //按位与非

wire alu\_loi; //低位加载

//调整操作码控制信号位宽为4位

assign alu\_add = (alu\_control == 4'd11); //B

assign alu\_sub = (alu\_control == 4'd10); //A

assign alu\_slt = (alu\_control == 4'd9); //9

assign alu\_sltu = (alu\_control == 4'd8); //8

assign alu\_and = (alu\_control == 4'd7); //7

assign alu\_nor = (alu\_control == 4'd6); //6

assign alu\_or = (alu\_control == 4'd5); //5

assign alu\_xor = (alu\_control == 4'd4); //4

assign alu\_sll = (alu\_control == 4'd3); //3

assign alu\_srl = (alu\_control == 4'd2); //2

assign alu\_sra = (alu\_control == 4'd1); //1

assign alu\_lui = (alu\_control == 4'd0); //0

//添加的三种运算

assign alu\_sgt = (alu\_control == 4'd12); //C

assign alu\_nod = (alu\_control == 4'd13); //D

assign alu\_loi = (alu\_control == 4'd14); //E

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;

//添加的三种运算的结果

wire [31:0] sgt\_result;

wire [31:0] nod\_result;

wire [31:0] loi\_result;

assign and\_result = alu\_src1 & alu\_src2; // 与结果为两数按位与

assign or\_result = alu\_src1 | alu\_src2; // 或结果为两数按位或

assign nor\_result = ~or\_result; // 或非结果为或结果按位取反

assign xor\_result = alu\_src1 ^ alu\_src2; // 异或结果为两数按位异或

assign lui\_result = {alu\_src2[15:0], 16'd0}; // 立即数装载结果为立即数移位至高半字节

//添加按位与非运算

assign nod\_result = ~and\_result; // 与非结果为与结果按位取反

//添加低位加载运算

assign loi\_result = {16'd0, alu\_src2[15:0]}; // 立即数装载结果为立即数移位至低半字节

//-----{加法器}begin

//add,sub,slt,sltu均使用该模块

wire [31:0] adder\_operand1;

wire [31:0] adder\_operand2;

wire adder\_cin ;

wire [31:0] adder\_result ;

wire adder\_cout ;

assign adder\_operand1 = alu\_src1;

assign adder\_operand2 = alu\_add ? alu\_src2 : ~alu\_src2;

assign adder\_cin = ~alu\_add; //减法需要cin

adder adder\_module(

.operand1(adder\_operand1),

.operand2(adder\_operand2),

.cin (adder\_cin ),

.result (adder\_result ),

.cout (adder\_cout )

);

//加减结果

assign add\_sub\_result = adder\_result;

//slt结果

//adder\_src1[31] adder\_src2[31] adder\_result[31]

// 0 1 X(0或1) "正-负"，显然小于不成立

// 0 0 1 相减为负，说明小于

// 0 0 0 相减为正，说明不小于

// 1 1 1 相减为负，说明小于

// 1 1 0 相减为正，说明不小于

// 1 0 X(0或1) "负-正"，显然小于成立

assign slt\_result[31:1] = 31'd0;

assign slt\_result[0] = (alu\_src1[31] & ~alu\_src2[31]) | (~(alu\_src1[31]^alu\_src2[31]) & adder\_result[31]);

//添加有符号比较，大于置位运算

//sgt结果

//adder\_src1[31] adder\_src2[31] adder\_result[31]

// 0 1 X(0或1) "正-负"，显然大于成立

// 0 0 1 相减为负，说明不大于

// 0 0 0 相减为正，说明大于

// 1 1 1 相减为负，说明不大于

// 1 1 0 相减为正，说明大于

// 1 0 X(0或1) "负-正"，显然大于不成立

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]);

//sltu结果

//对于32位无符号数比较，相当于33位有符号数（{1'b0,src1}和{1'b0,src2}）的比较，最高位0为符号位

//故，可以用33位加法器来比较大小，需要对{1'b0,src2}取反,即需要{1'b0,src1}+{1'b1,~src2}+cin

//但此处用的为32位加法器，只做了运算: src1 + ~src2 +cin

//32位加法的结果为{adder\_cout,adder\_result},则33位加法结果应该为{adder\_cout+1'b1,adder\_result}

//对比slt结果注释，知道，此时判断大小属于第二三种情况，即源操作数1符号位为0，源操作数2符号位为0

//结果的符号位为1，说明小于，即adder\_cout+1'b1为2'b01，即adder\_cout为0

assign sltu\_result = {31'd0, ~adder\_cout};

//-----{加法器}end

//-----{移位器}begin

// 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。

wire [4:0] shf;

assign shf = alu\_src1[4:0];

wire [1:0] shf\_1\_0;

wire [1:0] shf\_3\_2;

assign shf\_1\_0 = shf[1:0];

assign shf\_3\_2 = shf[3:2];

// 逻辑左移

wire [31:0] sll\_step1;

wire [31:0] sll\_step2;

assign sll\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[30:0], 1'd0} // 若shf[1:0]="01",左移1位

| {32{shf\_1\_0 == 2'b10}} & {alu\_src2[29:0], 2'd0} // 若shf[1:0]="10",左移2位

| {32{shf\_1\_0 == 2'b11}} & {alu\_src2[28:0], 3'd0}; // 若shf[1:0]="11",左移3位

assign sll\_step2 = {32{shf\_3\_2 == 2'b00}} & sll\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {sll\_step1[27:0], 4'd0} // 若shf[3:2]="01",第一次移位结果左移4位

| {32{shf\_3\_2 == 2'b10}} & {sll\_step1[23:0], 8'd0} // 若shf[3:2]="10",第一次移位结果左移8位

| {32{shf\_3\_2 == 2'b11}} & {sll\_step1[19:0], 12'd0}; // 若shf[3:2]="11",第一次移位结果左移12位

assign sll\_result = shf[4] ? {sll\_step2[15:0], 16'd0} : sll\_step2; // 若shf[4]="1",第二次移位结果左移16位

// 逻辑右移

wire [31:0] srl\_step1;

wire [31:0] srl\_step2;

assign srl\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {1'd0, alu\_src2[31:1]} // 若shf[1:0]="01",右移1位,高位补0

| {32{shf\_1\_0 == 2'b10}} & {2'd0, alu\_src2[31:2]} // 若shf[1:0]="10",右移2位,高位补0

| {32{shf\_1\_0 == 2'b11}} & {3'd0, alu\_src2[31:3]}; // 若shf[1:0]="11",右移3位,高位补0

assign srl\_step2 = {32{shf\_3\_2 == 2'b00}} & srl\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {4'd0, srl\_step1[31:4]} // 若shf[3:2]="01",第一次移位结果右移4位,高位补0

| {32{shf\_3\_2 == 2'b10}} & {8'd0, srl\_step1[31:8]} // 若shf[3:2]="10",第一次移位结果右移8位,高位补0

| {32{shf\_3\_2 == 2'b11}} & {12'd0, srl\_step1[31:12]}; // 若shf[3:2]="11",第一次移位结果右移12位,高位补0

assign srl\_result = shf[4] ? {16'd0, srl\_step2[31:16]} : srl\_step2; // 若shf[4]="1",第二次移位结果右移16位,高位补0

// 算术右移

wire [31:0] sra\_step1;

wire [31:0] sra\_step2;

assign sra\_step1 = {32{shf\_1\_0 == 2'b00}} & alu\_src2 // 若shf[1:0]="00",不移位

| {32{shf\_1\_0 == 2'b01}} & {alu\_src2[31], alu\_src2[31:1]} // 若shf[1:0]="01",右移1位,高位补符号位

| {32{shf\_1\_0 == 2'b10}} & {{2{alu\_src2[31]}}, alu\_src2[31:2]} // 若shf[1:0]="10",右移2位,高位补符号位

| {32{shf\_1\_0 == 2'b11}} & {{3{alu\_src2[31]}}, alu\_src2[31:3]}; // 若shf[1:0]="11",右移3位,高位补符号位

assign sra\_step2 = {32{shf\_3\_2 == 2'b00}} & sra\_step1 // 若shf[3:2]="00",不移位

| {32{shf\_3\_2 == 2'b01}} & {{4{sra\_step1[31]}}, sra\_step1[31:4]} // 若shf[3:2]="01",第一次移位结果右移4位,高位补符号位

| {32{shf\_3\_2 == 2'b10}} & {{8{sra\_step1[31]}}, sra\_step1[31:8]} // 若shf[3:2]="10",第一次移位结果右移8位,高位补符号位

| {32{shf\_3\_2 == 2'b11}} & {{12{sra\_step1[31]}}, sra\_step1[31:12]}; // 若shf[3:2]="11",第一次移位结果右移12位,高位补符号位

assign sra\_result = shf[4] ? {{16{sra\_step2[31]}}, sra\_step2[31:16]} : sra\_step2; // 若shf[4]="1",第二次移位结果右移16位,高位补符号位

//-----{移位器}end

// 选择相应结果输出

assign alu\_result = (alu\_add|alu\_sub) ? add\_sub\_result[31:0] :

alu\_slt ? slt\_result :

alu\_sltu ? sltu\_result :

alu\_and ? and\_result :

1. **实验结果分析**

（1）验证实验初始的11种运算，表中列出操作码、操作数和运算结果。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **操作码**  **（十六进制）** | **ALU操作** | **操作数1** | **操作数2** | **运算结果** |
| 800 | 加法 | 0FFFFFFF | 00000001 | 10000000 |
| 00000020 | 00000002 | 00000022 |
| 400 | 减法 | 00000055 | 00000005 | 00000050 |
| 10000000 | 0FFFFFFF | 00000001 |
| 200 | 有符号比较，小于置位 | 00000009 | 00000005 | 00000000 |
| F0000000 | 00000001 | 00000001 |
| 100 | 无符号比较，小于置位 | 00000009 | 00000005 | 00000000 |
| F0000000 | 00000001 | 00000000 |
| 080 | 按位与 | FF00FF00 | 00FF00FF | 00000000 |
| 00001111 | 0000FFFF | 00001111 |
| 040 | 按位或非 | FF00FF00 | 000F000F | 00F000F0 |
| 00FF00FF | 11001100 | EE00EE00 |
| 020 | 按位或 | FF00FF00 | 000F000F | FF0FFF0F |
| 00FF00FF | 11001100 | 11FF11FF |
| 010 | 按位异或 | FF00FF00 | 00FF00FF | FFFFFFFF |
| 00001111 | 0000000F | 0000111E |
| 008 | 逻辑左移 | 00000004 | 0000FF00 | 000FF000 |
| 00000008 | 0000FF00 | 00FF0000 |
| 004 | 逻辑右移 | 00000004 | 000FF000 | 0000FF00 |
| 00000008 | 00FF0000 | 0000FF00 |
| 002 | 算术右移 | 00000008 | 00FF0000 | 0000FF00 |
| 00000008 | FFFF0000 | FFFFFF00 |
| 001 | 高位加载 |  | 12345678 | 56780000 |
|  | 87654321 | 12340000 |

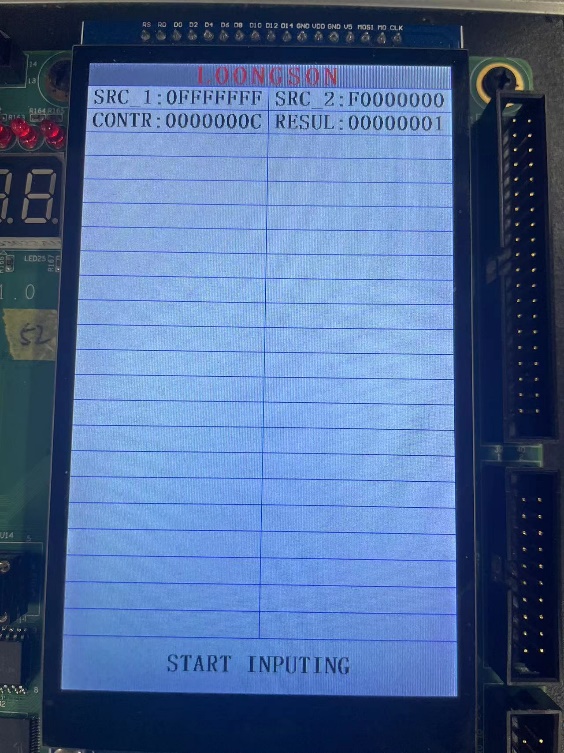
（2）验证实验改进后的11+3种运算，表中列出操作码、操作数和运算结果。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **操作码**  **（十六进制）** | **ALU操作** | **操作数1** | **操作数2** | **运算结果** |
| E | 低位加载 |  | 12345678 | 00005678 |
|  | 87654321 | 00004321 |
| D | 按位与非 | 00FF00FF | FF00FF00 | FFFFFFFF |
| 00FF00FF | 00FF00FF | FF00FF00 |
| C | 有符号比较，大于置位 | 00000050 | 00000020 | 00000001 |
| 0FFFFFFF | F0000000 | 00000001 |
| B | 加法 | 0FFFFFFF | 00000001 | 10000000 |
| 00000020 | 00000002 | 00000022 |
| A | 减法 | 00000055 | 00000005 | 00000050 |
| 10000000 | 0FFFFFFF | 00000001 |
| 9 | 有符号比较，小于置位 | 00000009 | 00000005 | 00000000 |
| F0000000 | 00000001 | 00000001 |
| 8 | 无符号比较，小于置位 | 00000009 | 00000005 | 00000000 |
| F0000000 | 00000001 | 00000000 |
| 7 | 按位与 | FF00FF00 | 00FF00FF | 00000000 |
| 00001111 | 0000FFFF | 00001111 |
| 6 | 按位或非 | FF00FF00 | 000F000F | 00F000F0 |
| 00FF00FF | 11001100 | EE00EE00 |
| 5 | 按位或 | FF00FF00 | 000F000F | FF0FFF0F |
| 00FF00FF | 11001100 | 11FF11FF |
| 4 | 按位异或 | FF00FF00 | 00FF00FF | FFFFFFFF |
| 00001111 | 0000000F | 0000111E |
| 3 | 逻辑左移 | 00000004 | 0000FF00 | 000FF000 |
| 00000008 | 0000FF00 | 00FF0000 |
| 2 | 逻辑右移 | 00000004 | 000FF000 | 0000FF00 |
| 00000008 | 00FF0000 | 0000FF00 |
| 1 | 算术右移 | 00000008 | 00FF0000 | 0000FF00 |
| 00000008 | FFFF0000 | FFFFFF00 |
| 0 | 高位加载 |  | 12345678 | 56780000 |
|  | 87654321 | 43210000 |

(3) 增加的三个运算实验箱验证照片

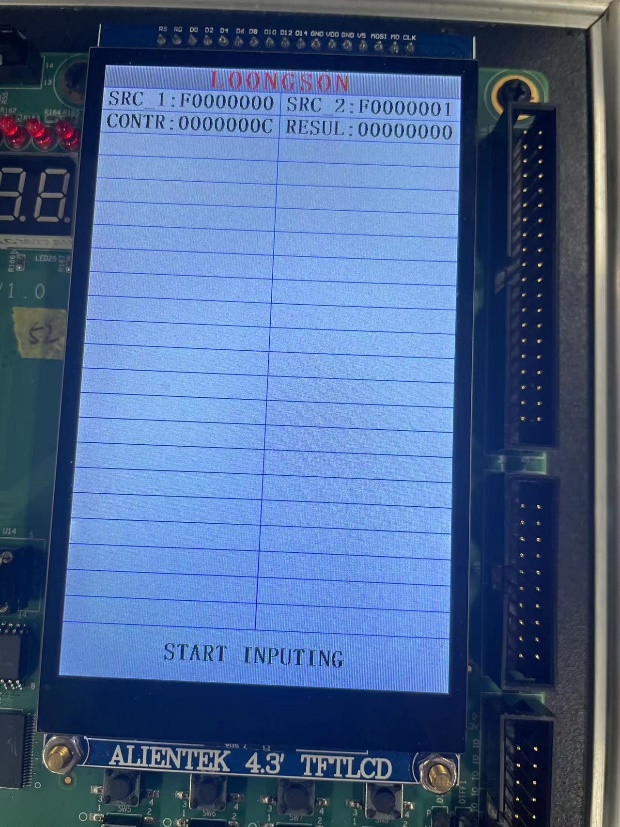
**有符号比较，大于置位：操作码为C（十六进制）**

a)正数-负数



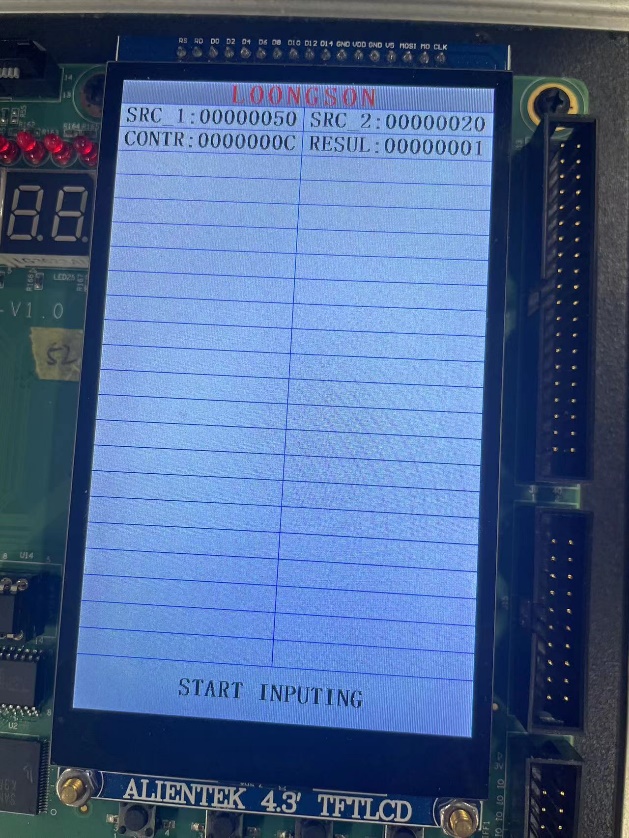
0FFFFFFF是正数，F0000000是负数，结果为1

b)负数-负数



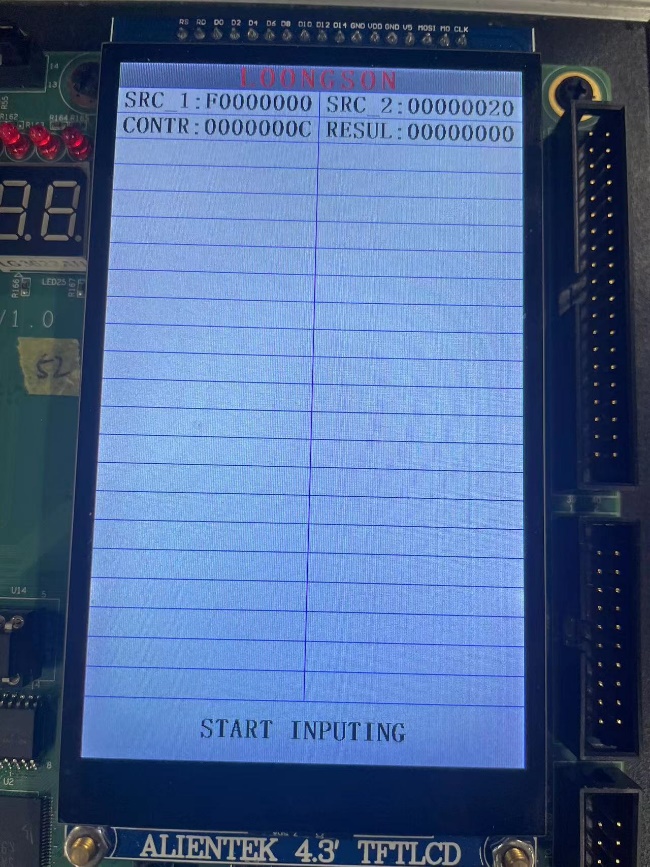
两个数都是负数，因为F0000001>F0000000，所以结果为0

c)正数-正数



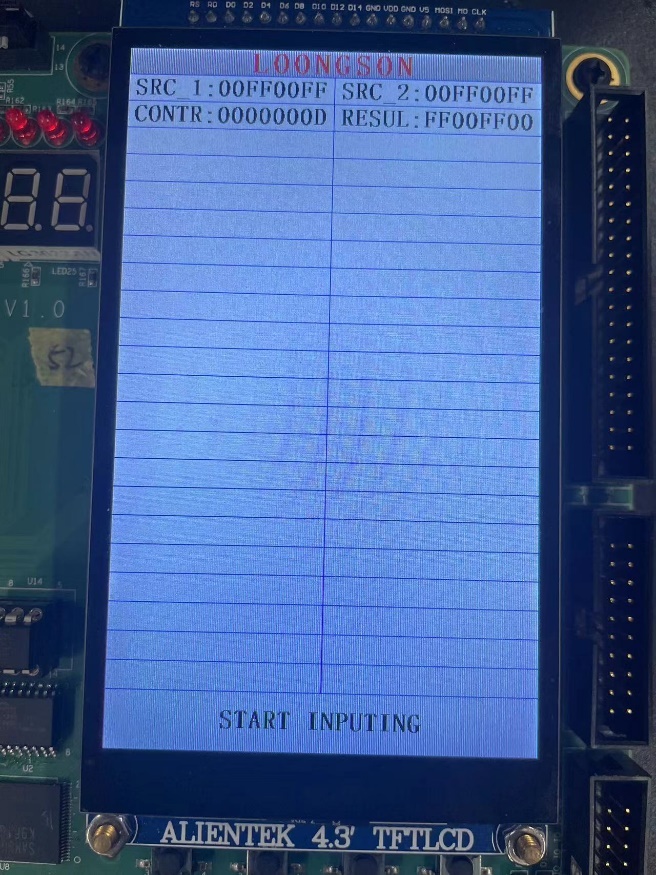
两个数都是正数，因为00000050>00000020，所以结果为1

d)负数-正数

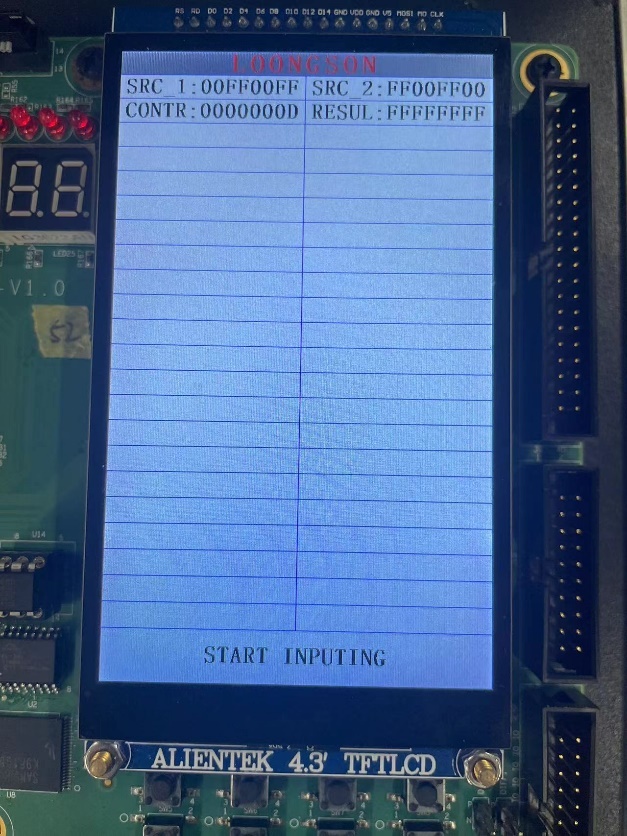


F0000000是负数，00000020是正数，结果为0

**按位与非：操作码为D（十六进制）**

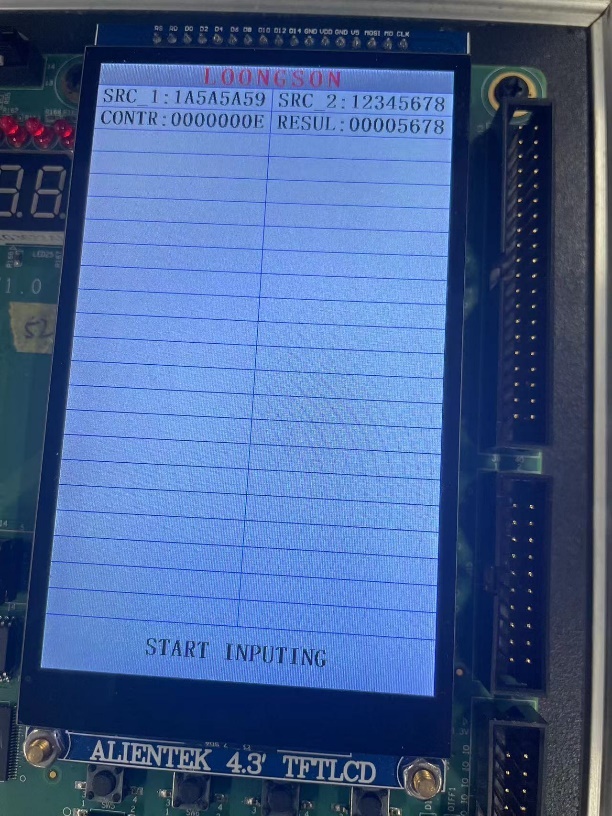
****

00FF00FF与00FF00FF进行按位与非运算，结果为FF00FF00，验证正确。



00FF00FF与FF00FF00进行按位与非运算，结果为FFFFFFFF，验证正确。

**低位加载：操作码为E（十六进制）**

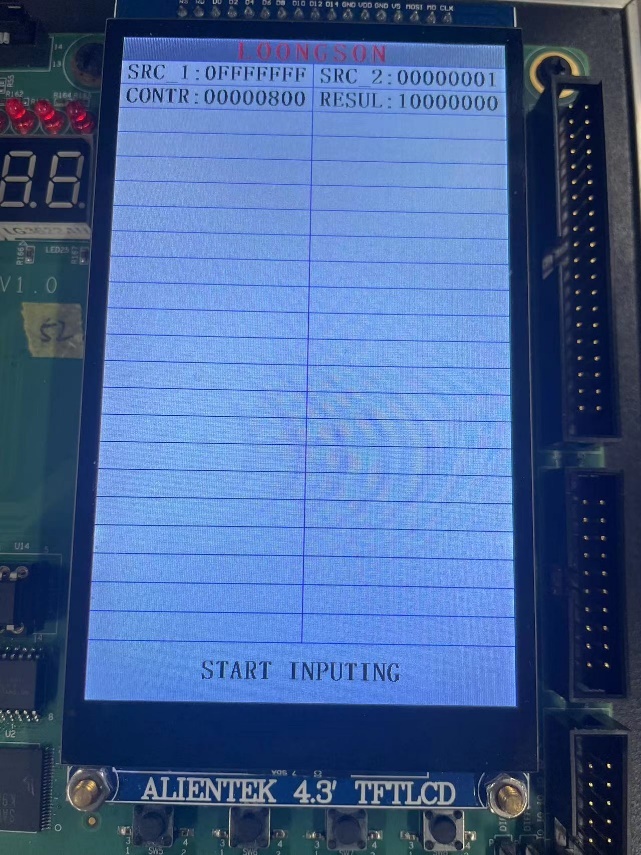
****

低位加载只和输入的第二个操作数有关，输入为12345678，低位加载的结果为00005678，验证正确。

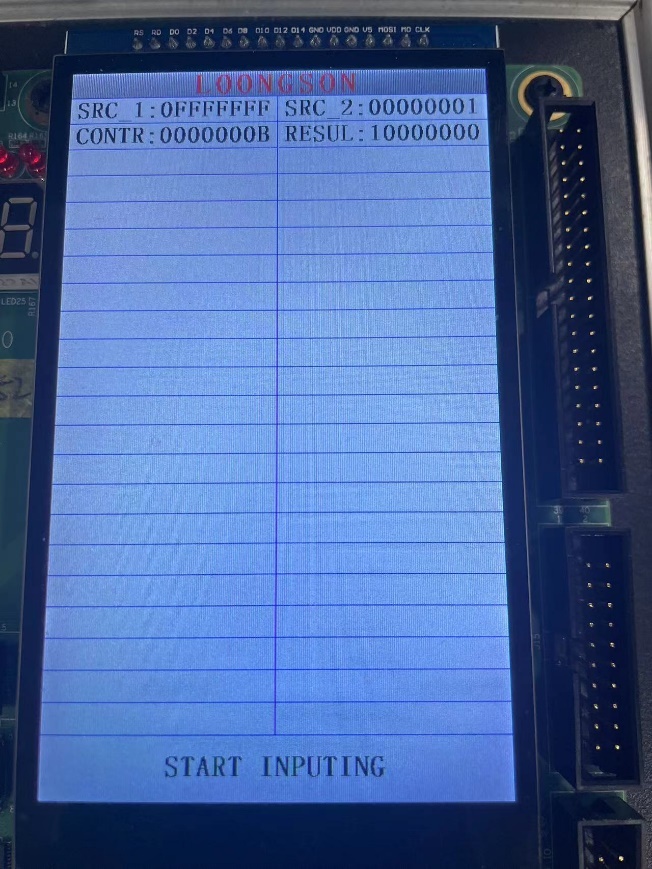
(4) 操作码更改前后照片对比

以加法为例，更改前加法的操作码是800（十六进制），更改后加法的操作码是B（十六进制）。

更改前：



更改后：



两次实验结果一致，成功更改操作码的位宽。

1. **总结感想**

在这次实验中，我学习了如何使用操作码进行运算，并且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。通过重新定义和压缩操作码，我可以使用更少的位数来控制运算，这样可以减少硬件成本和提高运算效率。

在实验过程中，我首先学习了11种基本的运算操作码，包括加减乘除、位运算和逻辑运算等。我了解了这些操作码的使用方法和运算规则，并且通过实验箱验证计算结果。在进行测试时，我准备了一些测试数据，以验证我的操作码的正确性。在实验过程中，我也遇到了一些问题，例如操作数超出范围等问题。通过仔细分析和查找资料，我成功解决了这些问题。

在掌握基本的操作码之后，我尝试自己定义了3种不同类型的运算，并且将操作码进行了位压缩。我通过实验箱验证了这些新的操作码，并且得到了正确的计算结果。这些操作码的定义和使用方法都比较简单，但是对于我来说，这是一次很好的实践和锻炼机会。

通过这个实验，我学到了很多知识和技能。我不仅学会了如何使用操作码进行运算，还了解了操作码的位压缩和控制信号位宽的调整方法。此外，我还学会了如何在实验箱上验证计算结果，并且提高了自己的实践能力。

在完成实验后，我认为这个实验对我的学习非常有帮助。通过实践，我巩固了理论知识，提高了自己的实践能力。此外，这个实验还让我认识到计算机组成原理的重要性，以及在计算机领域中深入学习的必要性。