组成原理实验课程第_4_次实验报告

实验名称	ALU 模块实现			班级	李涛老师
学生姓名	孙蕗	学号	2112060	指导老师	董前琨
实验地点	A308		实验时间	2023.5.9	

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、 实验改进内容

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，自行补充 3 种不同类型的运算，操作码和运算自行选择，需要上实验箱验证计算结果。

3） 本次实验不用仿真波形，直接上实验箱验证即可。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。实验报告中应该有两个表，第一个表为验证实验初始的 11 种运算，表中列出操作码、操作数和运算结果；第二个表是改进实验后的 11+3 种运算的验证，表中列出操作码、操作数和运算结果。注意自行添加的三种运算还需要附上实验箱验证照片。

3、 实验设备

（1） 装有 Xilinx Vivado 的计算机一台。

（2） LS-CPU-EXB-002 教学系统实验箱一套。

4、 实验任务

（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） 自行设计本次实验的方案，画出结构框图，大致结构框图如图 5.1。图 5.1 中的操作码位数和类型请自行设计，可以设计为独热码（一位有效编码）或二进制编码。比如，设计方案中预定实现 7 种 ALU 运算，则操作码采用独热码，则需 7bit 数据，每位单独指示一种运算；若采用二进制编码，则只用 3bit 数据位即可，但在需 ALU 内部先进行解码，才能确定 ALU 作何种运算。

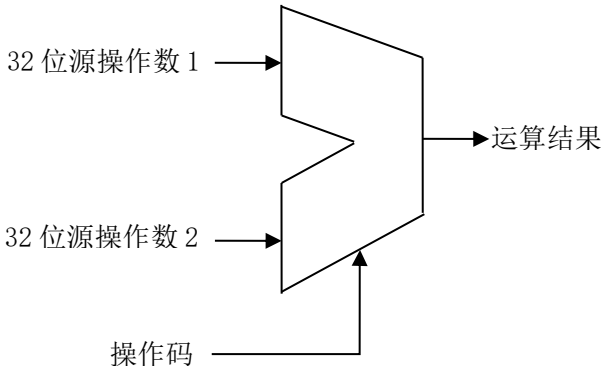


图 5.1　ALU **模块的大致框图**

（4） 根据设计的实验方案，使用 verilog 编写相应代码。

（5） 对编写的代码进行仿真，得到正确的波形图。

（6） 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，见图 5.2。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。

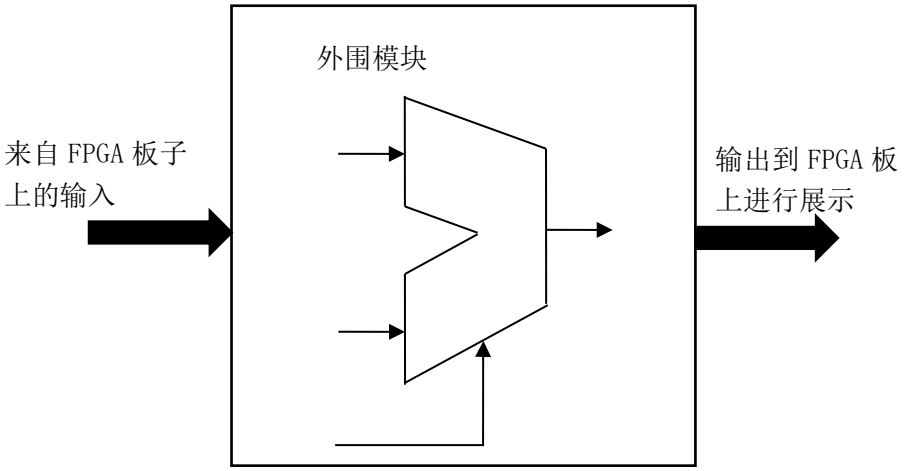


图 5.2　ALU 设计实验的顶层模块大致框图

（7） 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。

5、 实验原理图

ALU 模块的原理图如下：

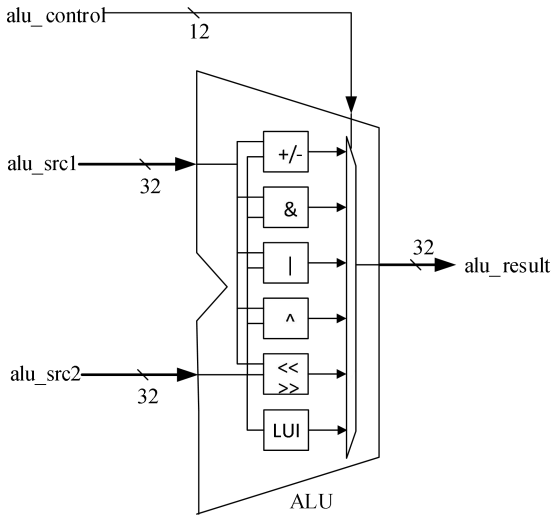


图 5.3　ALU 的原理图

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：

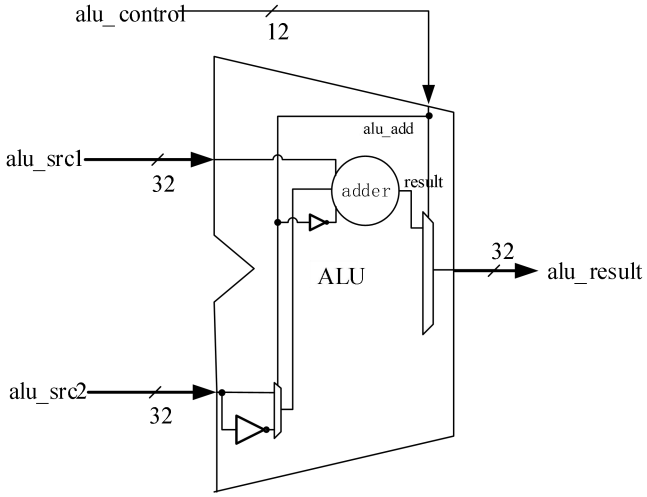


图 5.4　加减法运算的原理图

2）　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 5.2　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 5.2 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 5.2 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 5.5。故无符号 32 位比较小于置位运算结果表达式为：sltu_result = ~adder_cout.

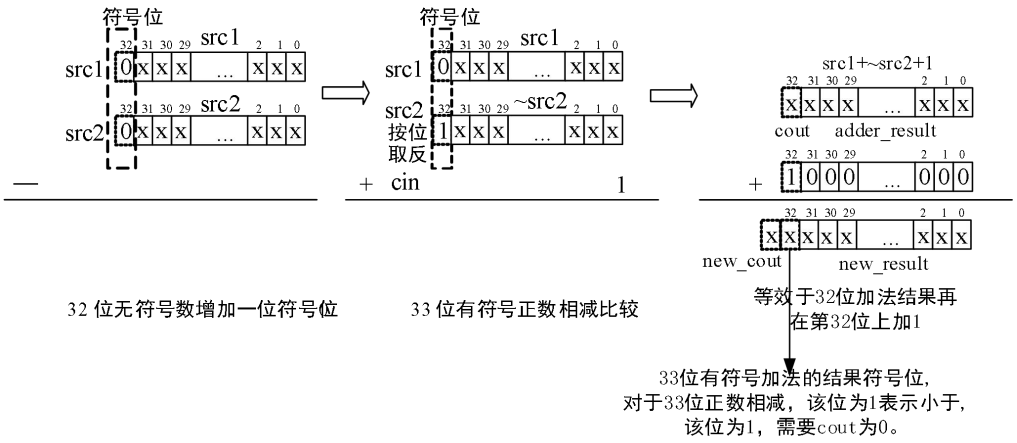


图 5.5　无符号 32 位数比较的原理图

6、 实验顶层模块框图

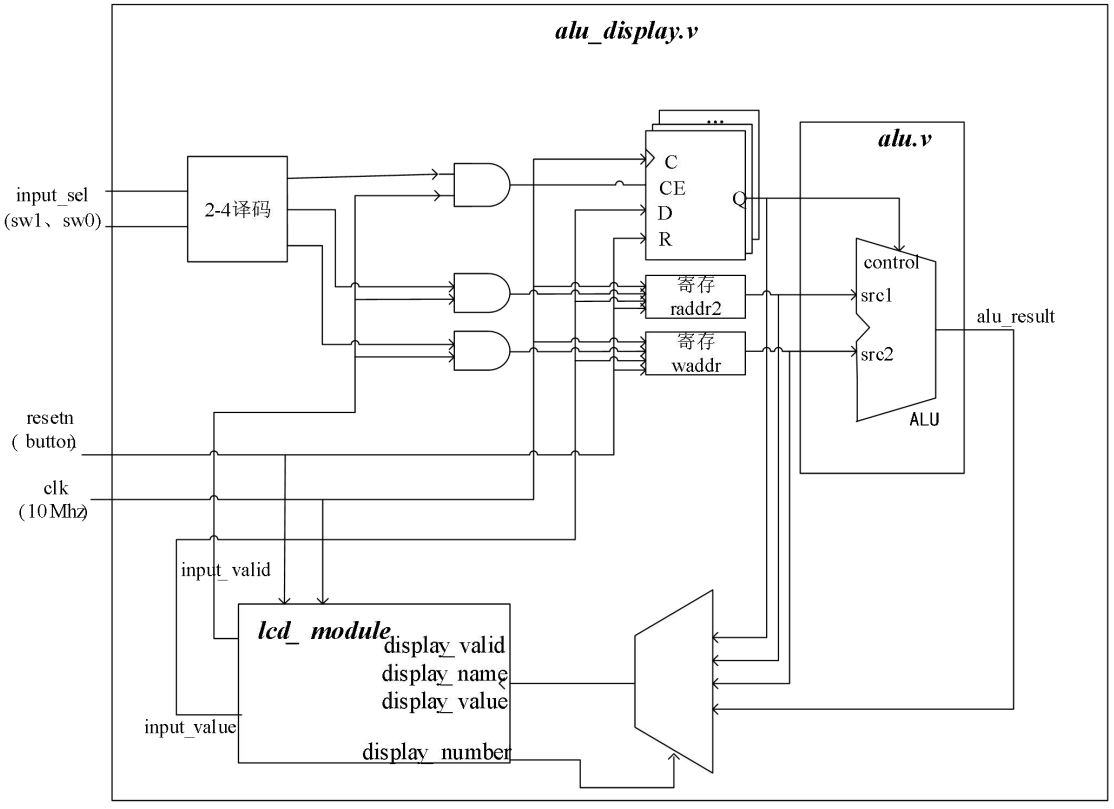


图 5.6　ALU 参考设计的顶层模块框图

7、 参考设计

（1） 做好预习：

1） 熟知指令类型，了解指令功能和编码；

2） 归纳基础的 ALU 运算指令，确定自己准备实现的 ALU 运算；

3） 设计本次实验的方案，列出准备实现的 ALU 运算和操作码的编码；

4） 在课前画好实验方案的设计框图，即补充完善图 5.1；

5） 如果对 FPGA 板了解的话，可确定设计中与 FPGA 板上交互的接口，画出包含外围模块的整体设计框图，即补充完善图 5.2。

（2） 实验实施：

1） 确认 ALU 模块的设计框图的正确性；

2） 编写 verilog 代码；

3） 对该模块进行仿真，得出正确的波形，截图作为实验报告结果一项的材料；

4） 完成调用 ALU 模块的外围模块的设计，并编写代码；

5） 对代码进行综合布局布线下载到试验箱里 FPGA 板上，进行上板验证。

（3） 实验检查：

1） 完成上板验证后，让指导老师或助教进行检查，进行现场演示。先说明自己实现的 ALU 运算类型，按照检查人员的要求，对特定源操作数进行特定运算操作，检查运算结果的正确性，可对演示结果进行拍照作为实验报告结果一项的材料。

（4） 实验报告的撰写：

1） 实验结束后，需按照规定的格式完成实验报告的撰写。

（5） 本部分给出的 ALU 实验的参考设计方案中，共实现了 12 种运算，采用独热编码操作码(即 ALU 的控制信号)，对应关系如下：

表 5.1 ALU **的控制信号**

控制信号												**ALU 操作**
11	**10**	**9**	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**	
0	0	0	0	0	0	0	0	0	0	0	0	无
1	0	0	0	0	0	0	0	0	0	0	0	加法
0	1	0	0	0	0	0	0	0	0	0	0	减法
0	0	1	0	0	0	0	0	0	0	0	0	有符号比较，小于置位
0	0	0	1	0	0	0	0	0	0	0	0	无符号比较，小于置位
0	0	0	0	1	0	0	0	0	0	0	0	按位与
0	0	0	0	0	1	0	0	0	0	0	0	按位或非
0	0	0	0	0	0	1	0	0	0	0	0	按位或
0	0	0	0	0	0	0	1	0	0	0	0	按位异或
0	0	0	0	0	0	0	0	1	0	0	0	逻辑左移
0	0	0	0	0	0	0	0	0	1	0	0	逻辑右移
0	0	0	0	0	0	0	0	0	0	1	0	算术右移
0	0	0	0	0	0	0	0	0	0	0	1	高位加载

8、 实验步骤

（1） 修改 alu_display.v 代码，实现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。

```
//-----{调用 ALU 模块}begin
    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
    wire    [31:0] alu_result;     // ALU 结果
```

（2） 修改 alu.v 代码，实现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。

```
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_nand;     //按位与非
    wire alu_lli;     //低位加载
    wire alu_not;     //按位取反

    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

    assign alu_nand = alu_control == 4'b1110 ? 1 : 0;
    assign alu_lli    = alu_control == 4'b1101 ? 1 : 0;
    assign alu_not    = alu_control == 4'b1100 ? 1 : 0;

    assign alu_add    = alu_control == 4'b1011 ? 1 : 0;
    assign alu_sub    = alu_control == 4'b1010 ? 1 : 0;
    assign alu_slt    = alu_control == 4'b1001 ? 1 : 0;
    assign alu_sltu = alu_control == 4'b1000 ? 1 : 0;
    assign alu_and    = alu_control == 4'b0111 ? 1 : 0;
    assign alu_nor    = alu_control == 4'b0110 ? 1 : 0;
    assign alu_or     = alu_control == 4'b0101 ? 1 : 0;
    assign alu_xor    = alu_control == 4'b0100 ? 1 : 0;
    assign alu_sll    = alu_control == 4'b0011 ? 1 : 0;
    assign alu_srl    = alu_control == 4'b0010 ? 1 : 0;
```

```
    assign alu_sra    = alu_control == 4'b0001 ? 1 : 0;
    assign alu_lui    = alu_control == 4'b0000 ? 1 : 0;

    wire[31:0] nand_result;
    wire[31:0] lli_result;
    wire[31:0] not_result;

    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;

    assign nand_result=~and_result;//按位与非
    assign lli_result = {16'd0,alu_src2[31:16]};//低位加载
    assign not_result=~alu_src2; //按位取反的结果

    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
```

（3） 修改 alu.v 代码，实现按位与非，低位加载，按位取反 3 种功能

```
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_nand;    //按位与非
    wire alu_lli;    //低位加载
    wire alu_not;    //按位取反

    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

assign alu_nand = alu_control == 4'b1110 ? 1 : 0;
assign alu_lli    = alu_control == 4'b1101 ? 1 : 0;
assign alu_not    = alu_control == 4'b1100 ? 1 : 0;

assign alu_add    = alu_control == 4'b1011 ? 1 : 0;
assign alu_sub    = alu_control == 4'b1010 ? 1 : 0;
assign alu_slt    = alu_control == 4'b1001 ? 1 : 0;
assign alu_sltu = alu_control == 4'b1000 ? 1 : 0;
assign alu_and    = alu_control == 4'b0111 ? 1 : 0;
assign alu_nor    = alu_control == 4'b0110 ? 1 : 0;
assign alu_or     = alu_control == 4'b0101 ? 1 : 0;
assign alu_xor    = alu_control == 4'b0100 ? 1 : 0;
assign alu_sll    = alu_control == 4'b0011 ? 1 : 0;
assign alu_srl    = alu_control == 4'b0010 ? 1 : 0;
assign alu_sra    = alu_control == 4'b0001 ? 1 : 0;
assign alu_lui    = alu_control == 4'b0000 ? 1 : 0;

wire[31:0] nand_result;
wire[31:0] lli_result;
wire[31:0] not_result;

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;

assign nand_result=~and_result;//按位与非
```

```
assign lli_result = {16'd0,alu_src2[31:16]};//低位加载
assign not_result=~alu_src2; //按位取反的结果

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
```

```
// 选择相应结果输出
        assign alu_result = alu_nand              ? nand_result :
                            alu_lli               ? lli_result :
                            alu_not               ? not_result :
                            (alu_add|alu_sub) ? add_sub_result[31:0] :
                            alu_slt               ? slt_result :
                            alu_sltu              ? sltu_result :
                            alu_and               ? and_result :
                            alu_nor               ? nor_result :
                            alu_or                ? or_result    :
                            alu_xor               ? xor_result :
                            alu_sll               ? sll_result :
                            alu_srl               ? srl_result :
                            alu_sra               ? sra_result :
                            alu_lui               ? lui_result :
                            32'd0;
```

9、 实验结果分析

（1） 实验手册上的代码

操作	ALU 操作数 1alu_src1	ALU 操作数 2alu_src2	ALU 控制信号 alu_control	ALU 结果 alu_result
加法	11111111	22222222	00000800	33333333
减法	22222222	11111111	00000400	11111111
有符号比较，小于置位（正数）	00000000	11111111	00000200	00000001
有符号比较，小于置位（负数）	00000002	FFFFFFFD	00000200	00000000
无符号比较，小于置位	00000000	11111111	00000100	00000001
按位与	10010011	11111111	00000080	10010011
按位或非	10010011	11111111	00000040	EEEEEEEE
按位或	10010011	11111111	00000020	11111111
按位异或	10010011	11111111	00000010	01101100
逻辑左移	00000004	00000004	00000008	00000040
逻辑右移	00000001	00000004	00000004	00000002

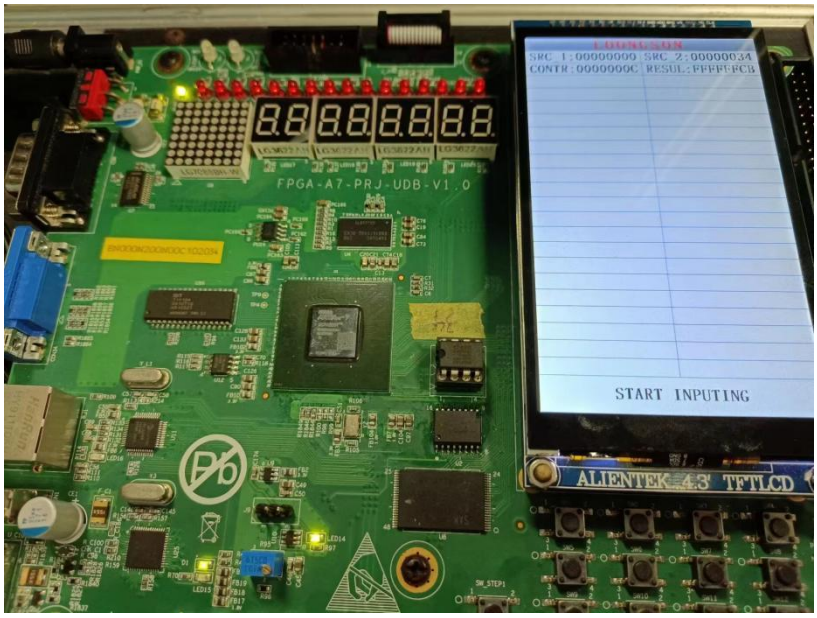
算术右移	00000002	FF000000	00000002	FFC00000
高位加载	22223333	00002222	00000001	22220000

（2） 改过的代码

操作	ALU 操作数 1alu_src1	ALU 操作数 2alu_src2	ALU 控制信号 alu_control	ALU 结果 alu_result
加法	11111111	22222222	0000000B	33333333
减法	22222222	11111111	0000000A	11111111
有符号比较，小于置位（正数）	00000000	11111111	00000009	00000001
有符号比较，小于置位（负数）	00000002	FFFFFFFD	00000009	00000000
无符号比较，小于置位	00000000	11111111	00000008	00000001
按位与	10010011	11111111	00000007	10010011
按位或非	10010011	11111111	00000006	EEEEEEEE
按位或	10010011	11111111	00000005	11111111
按位异或	10010011	11111111	00000004	01101100
逻辑左移	00000004	00000004	00000003	00000040
逻辑右移	00000001	00000004	00000002	00000002
算术右移	00000002	FF000000	00000001	FFC00000
高位加载	22223333	00002222	00000000	22220000
按位取反	00000000	00000034	0000000C	FFFFFFCB
低位加载	00000000	12345678	0000000D	00001234
按位与非	00001100	00001010	0000000E	FFFFEFFF

（3） 自行添加的三种运算附实验箱验证照片

A) 按位取反



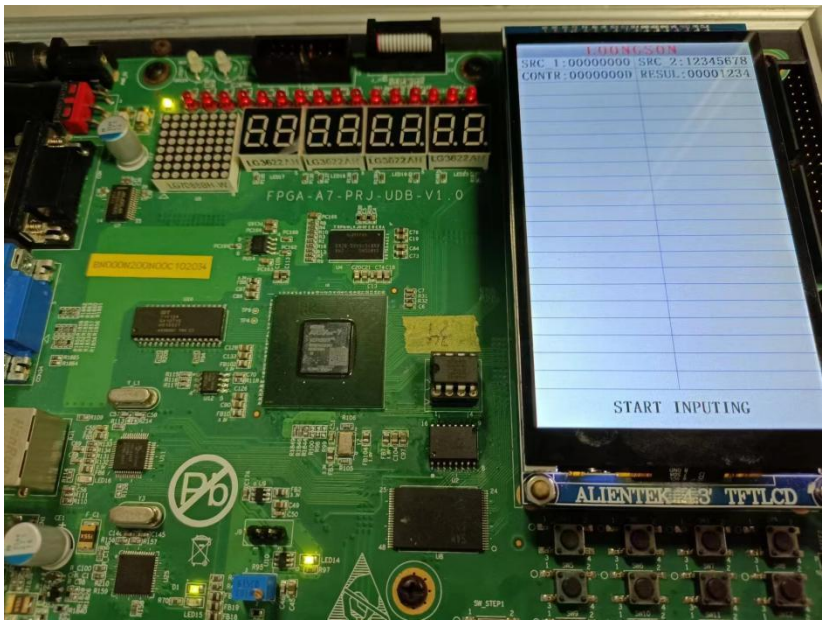
alu_src1=00000000，alu_src2=00000034。

alu_control=0000000C，按位取反。

16 进制 alu_src2 按位取反的结果是 FFFFFFCB。

显示 alu_result=FFFFFFCB。结果正确

B) 低位加载



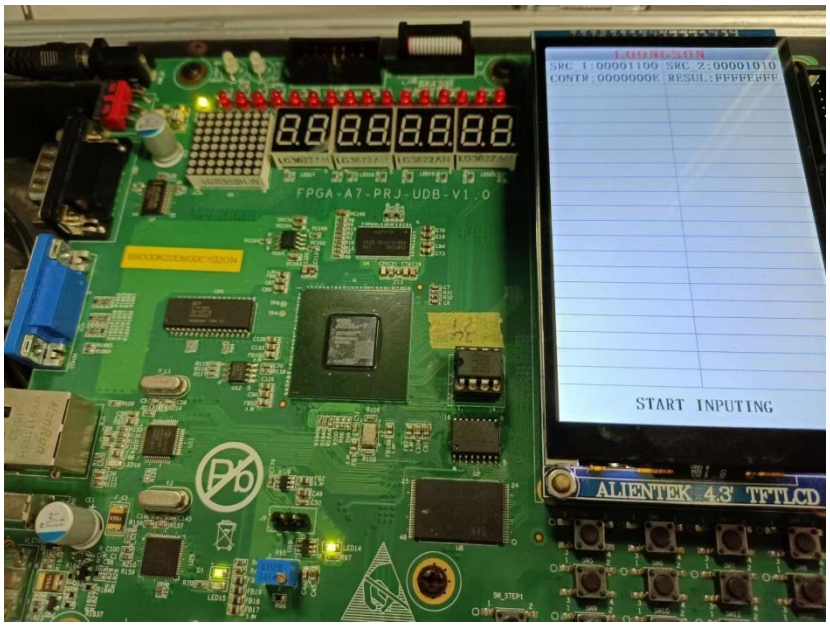
alu_src1=00000000，alu_src2=12345678。

alu_control=0000000D，低位加载。

alu_src2 低位加载的结果是 00001234。

显示 alu_result=00001234。结果正确

C) 按位与非



alu_src1=00001100，alu_src2=00001010。

alu_control=0000000E，按位与非。

16 进制 alu_src1 和 alu_src2 按位与非的结果是 FFFFEFFF。

显示 alu_result=FFFFEFFF。结果正确

10、 总结感想

（1） 通过本次实验，对 ALU 的指令集的功能与编码有了更深入的了解。在讨论、学

习中，学会了如何编写实现一条指令：如何定义、如何计算、如何显示结果等。

（2） 通过代码修改，实现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） 添加了按位与非，低位加载，按位取反 3 种功能，通过与、非运算并加入新指令，实现 3 种功能。