组成原理实验课程第_4_次实验报告

实验名称	寄存器堆改进			班级	李涛
学生姓名	阿斯雅	学号	2210737	指导老师	董前琨
实验地点	实验楼 A 区 308		实验时间	5/9 日 中午	

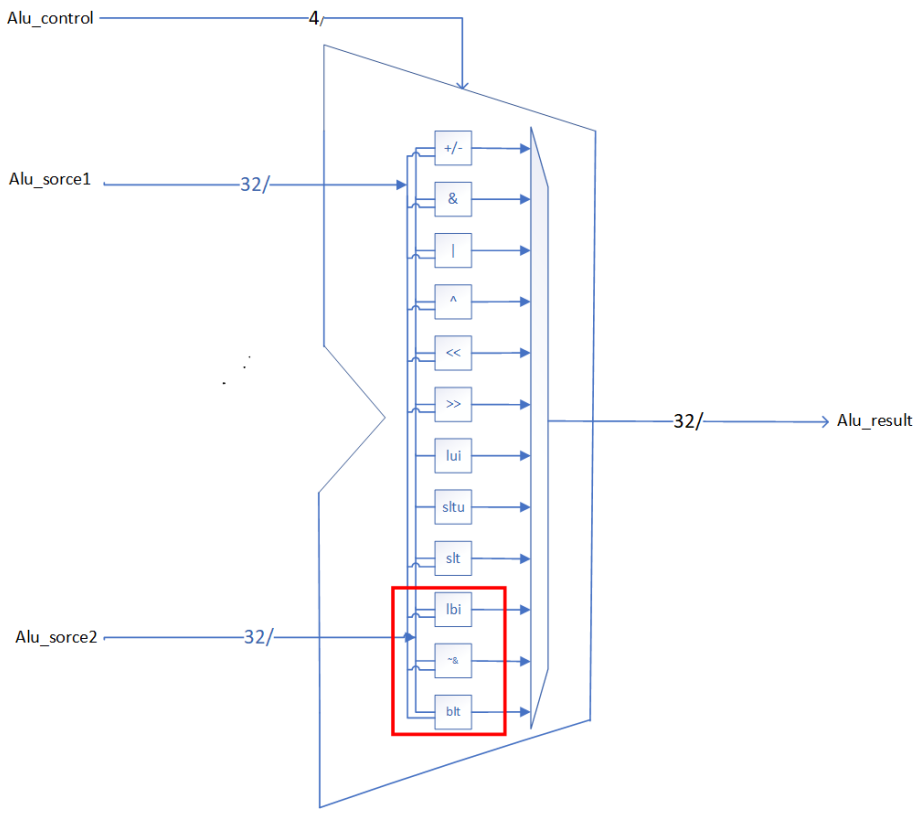
一、实验目的

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

结合实验指导手册中的实验四（ALU 模块实现实验）完成功能改进，实现一个能够完成更多运算的 ALU。

三、实验原理图



四、实验步骤

4.1、分析

因为我们要把原来的 12 位表示的 ALU 控制码变成四位的，所以我们可以直接判断当前的 ALU 控制码的数值是多少，而不是用当前为 1 的位数是多少。又因为四位最多能表示 16 种状态，除去零状态，我们最多能做 15 种运算，所以我们也需要在原来的 12 种运算上增加三种运算。本次实验我所加的三种运算是两个数与非，第二个操作数低位加载和无符号数大于则置位三种运算。

实现这三种运算并不难：实现与非我们可以在原来的与运算的结果前面加一个非符号，实现低位加载可以把第二个操作数的高 16 位加载到结果的低 16 位，高 16 位则置零。实现无符号数大于则置位则只需要把小于则置位中的最低一位取反就可。

4.2、修改代码

```
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] nand_result;
```

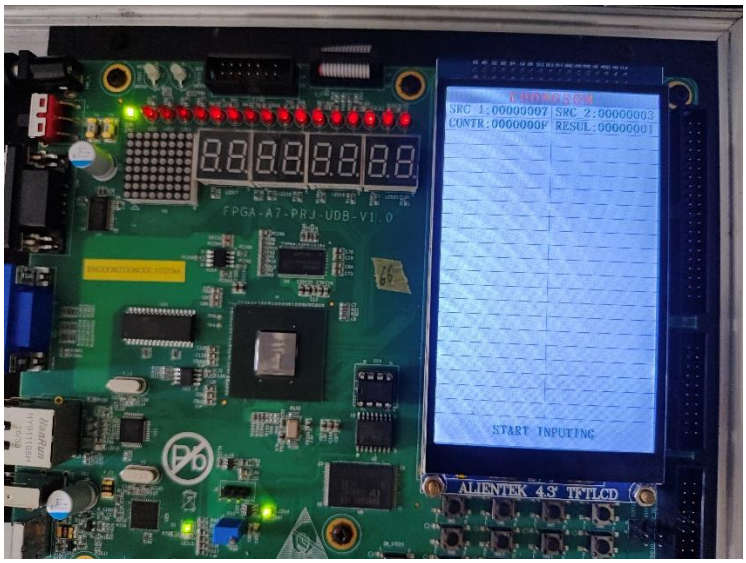
```
assign sltu_result = {31'd0, ~adder_cout};
assign bltu_result = {31'd0, adder_cout};//大于则置位
```

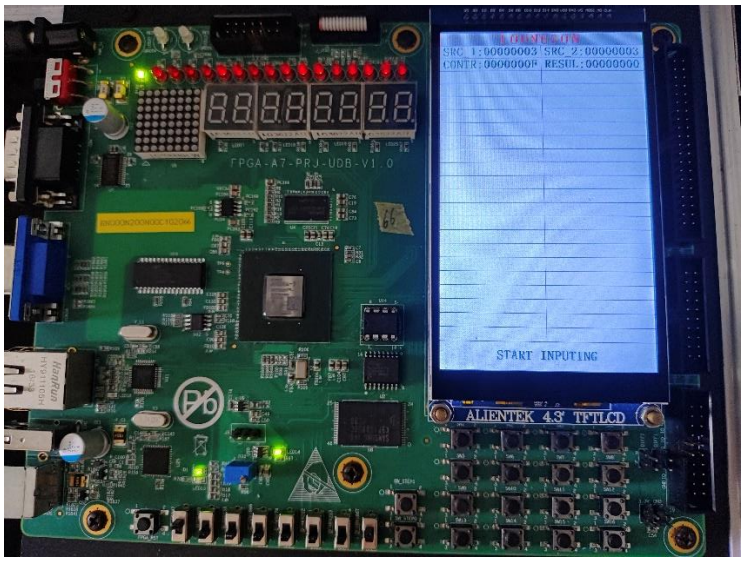
```
assign and_result = alu_src1 & alu_src2;      // 与结果为两数按位与
assign or_result  = alu_src1 | alu_src2;      // 或结果为两数按位或
assign nor_result = ~or_result;               // 或非结果为或结果按位取反
assign xor_result = alu_src1 ^ alu_src2;      // 异或结果为两数按位异或
assign nand_result = ~and_result;
assign lui_result = {alu_src2[15:0], 16'd0}; // 立即数装载结果为立即数移位至高半字节
assign hui_result = {16'd0,alu_src2[31:16]}; // 立即数装载结果为立即数移位至低半字节
```

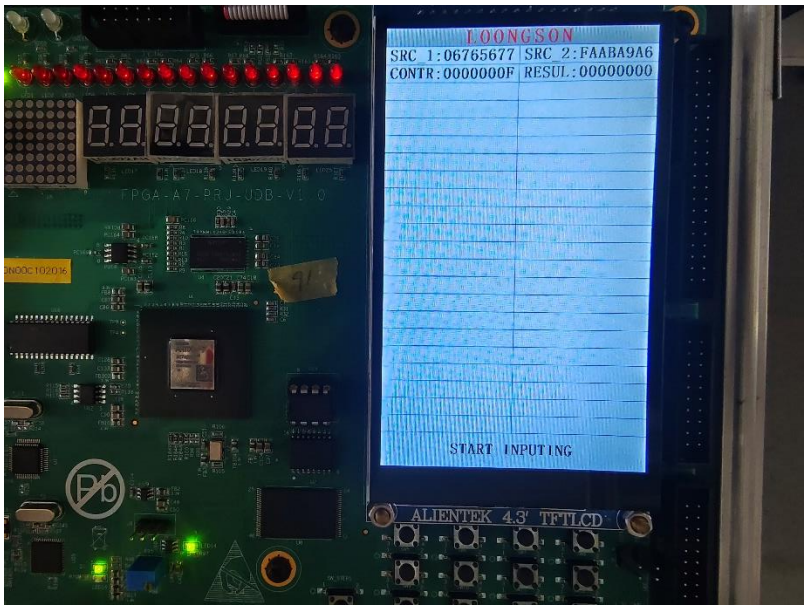
```
assign alu_result = (alu_control==4'b0001|alu_control==4'b0010) ? add_sub_result[31:0] :
                    alu_control==4'b0011          ? slt_result :
                    alu_control==4'b0100          ? sltu_result :
                    alu_control==4'b0101          ? and_result :
                    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 :
```

五、实验结果分析

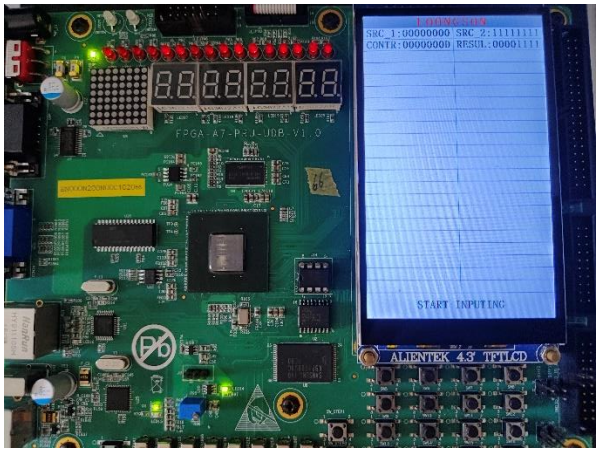
首先是大于则置位，控制这个运算的控制码是 1111，也就是十六进制的 F，我们可以 c 从第一二张图片看出，只有当第一个操作数大于第二个操作数时结果才会为 1，而从第三张图片也可以知道，我做的是两个无符号数的比较。







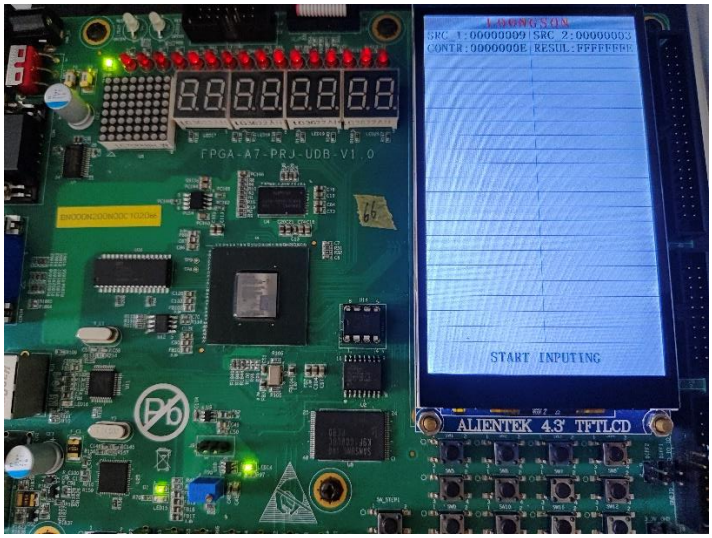
其次是低位加载，控制这个运算的控制码是 1101，也就是十六进制数 D。我们可以从图片看出最后结果是把第二个操作数的高 16 位加载到了结果的低 16 位。



最后一个是与非运算，相应的控制码是 1110，也就是十六进制数 E，我们可以从结果照片可以看出,操作数 1 是 9 和操作 2 是 3 的时候最后结果是 FFFFFFFE，我们可以验证这个结果对不对：

00000000000000000000000000001001
00000000000000000000000000000011

与结果：00000000000000000000000000000001
与非结果：11111111111111111111111111111110
可以看出结果是对的。



六、心得体会

通过本次 ALU 改进实验，我深入了解了在 Verilog 语句中实现无符号比较、有符号比较、左右移运算的方法。对于无符号比较运算，我了解到可以通过加法器输出的进位来判断，而有符号比较运算则需要根据真值表得出相应的表达式。左移和右移运算在 Verilog 中有对应的运算符，可以直接利用它们来实现。

除此之外，我还学到了如何通过增加新的运算功能来丰富 ALU 的功能。这些知识让我能够更加熟练地在 Verilog 语句中实现各种运算功能，不仅提升了 ALU 的性能，也增强了其功能性。这次实验对我的学习和技能提升都起到了重要作用。