

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
学生姓名	齐明杰	学号	2113997	指导老师	董前琨
实验地点	津南实验楼 A306		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、实验内容说明

在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。
- 5、按实验报告模板要求完成实验报告，以附件形式提交，提交时文件名格式为“学号_姓名_组成原理第四次实验.pdf”。注意实验报告中要有介绍分析的内容，针对实验箱照片，要解释图中信息，是否验证成功。

3、实验原理图

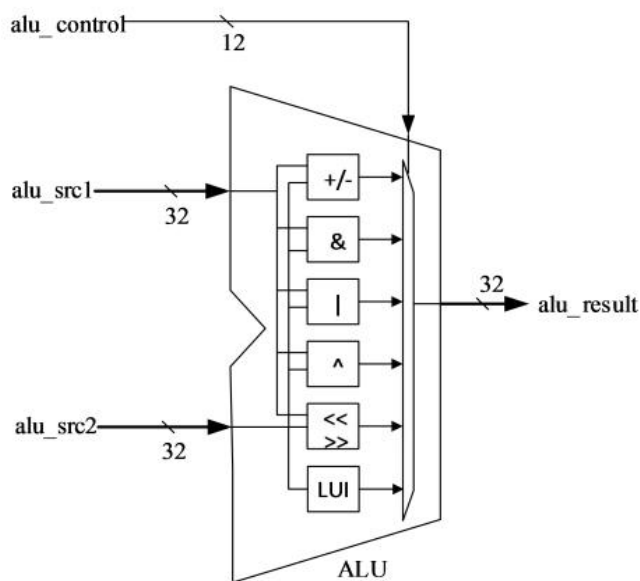


图 5.3 ALU 的原理图

4、实验步骤

项目代码(改动处标红):

1. 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
```

```
wire alu_addi;    //低半部分和高半部分颠倒  
wire alu_nxor;    //按位同或  
wire alu_blt;     //无符号大于则置位
```

注: 添加的三种运算分别为: ① 将第二个源操作数的高 16 位与低 16 位交换
② 两个源操作数进行按位同或操作
③ 将两个源操作数进行无符号大于置位比较

```
assign alu_add  =(alu_control==4'b0001 ? 1 : 0); //1 为加法  
assign alu_sub  =(alu_control==4'b0010 ? 1 : 0); //2 为减法  
assign alu_slt  =(alu_control==4'b0011 ? 1 : 0); //3 为有符号比较  
assign alu_sltu =(alu_control==4'b0100 ? 1 : 0); //4 为无符号比较  
assign alu_and  =(alu_control==4'b0101 ? 1 : 0); //5 为按位与  
assign alu_nor  =(alu_control==4'b0110 ? 1 : 0); //6 为按位或非  
assign alu_or   =(alu_control==4'b0111 ? 1 : 0); //7 为按位或  
assign alu_xor  =(alu_control==4'b1000 ? 1 : 0); //8 为按位异或  
assign alu_sll  =(alu_control==4'b1001 ? 1 : 0); //9 为逻辑左移  
assign alu_srl  =(alu_control==4'b1010 ? 1 : 0); //A 为逻辑右移  
assign alu_sra  =(alu_control==4'b1011 ? 1 : 0); //B 为算术右移  
assign alu_lui  =(alu_control==4'b1100 ? 1 : 0); //C 为高位加载
```

```
assign alu_addi  =(alu_control==4'b1101 ? 1 : 0);//D 为低半部分和高半部分颠倒
```

```
assign alu_nxor  =(alu_control==4'b1110 ? 1 : 0);//E 为按位同或
```

```
assign alu_blt   =(alu_control==4'b1111 ? 1 : 0);//F 为无符号大于则置位
```

注：以 `alu_add=(alu_control==4'b0001)?1:0` 为例，意思是若控制信号为 1 则将 `alu_add` 置为 1，即进行加法操作，依此类推。另外，我添加的颠倒，同或，无符号大于置为操作的控制信号依次为 D, E, F.

```
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] addi_result;
```

```
wire[31:0] nxor_result;
```

```
wire[31:0] blt_result;
```

注：分别用于存储三个运算的结果。

```
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节
```

```
assign nxor_result=~xor_result; //同或结果为异或取反
```

```
assign addi_result={alu_src2[15:0],alu_src2[31:16]};//颠倒
```

注：同或和异或是相反操作，因此直接把已经算好的异或取反即可。

高低 16 位互换只需要使用花括号进行拼接操作即可。

```
.....
```

```
.....(省略不变代码)
```

```
//slt 结果
```

```
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]);
```

```
//blt 结果
```

```
assign blt_result[31:1]=31'd0;
```

```
assign blt_result[0]=(~alu_src1[31] & alu_src2[31])|(~(alu_src1[31]^alu_src2[31]) &
```

`~adder_result[31]&(adder_result!=32'b0));`

注：此处为无符号大于置位比较操作，使用真值表，利用逻辑代数运算知识可得出上述公式。

```
.....
.....(省略不变代码)
// 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 :
                    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 :
                    alu_addi          ? addi_result :
                    alu_nxor          ? nxor_result :
                    alu_blt           ? blt_result  :
                    32'd0;

endmodule
```

2. alu_display.v:

```
.....
.....(省略不变代码)
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

.....
.....(省略不变代码)
//当 input_sel 为 00 时，表示输入数控制信号，即 alu_control
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
end
```

注：同上，压缩后的控制信号为 4 位。

.....

.....(省略不变代码)

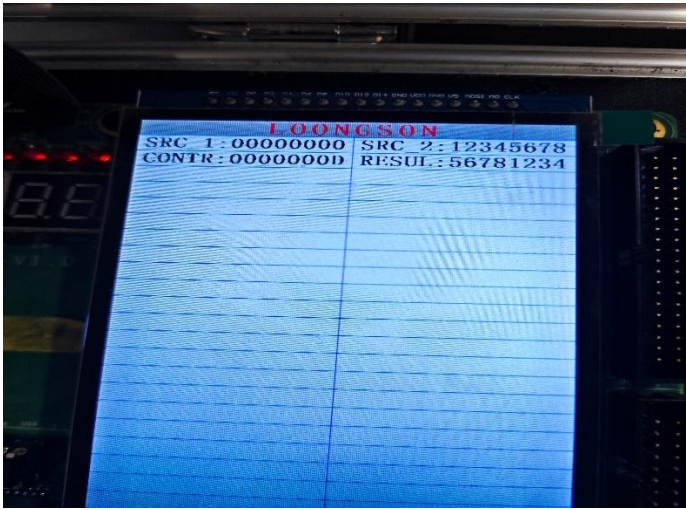
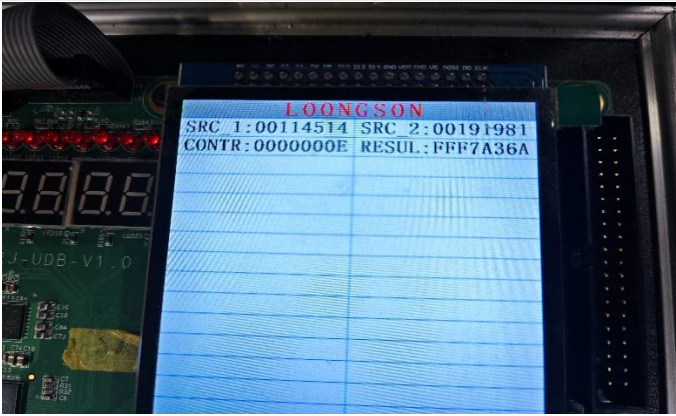

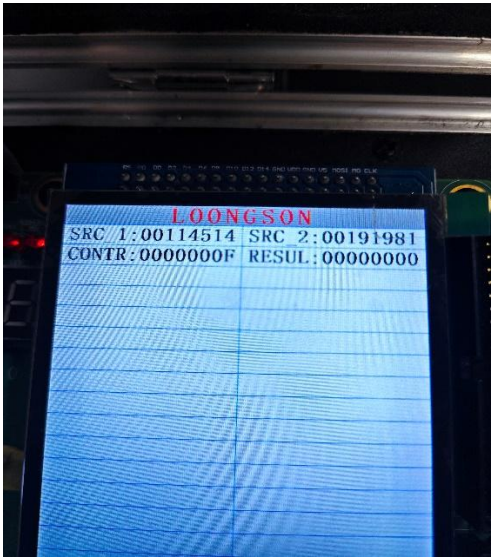
5、实验结果分析

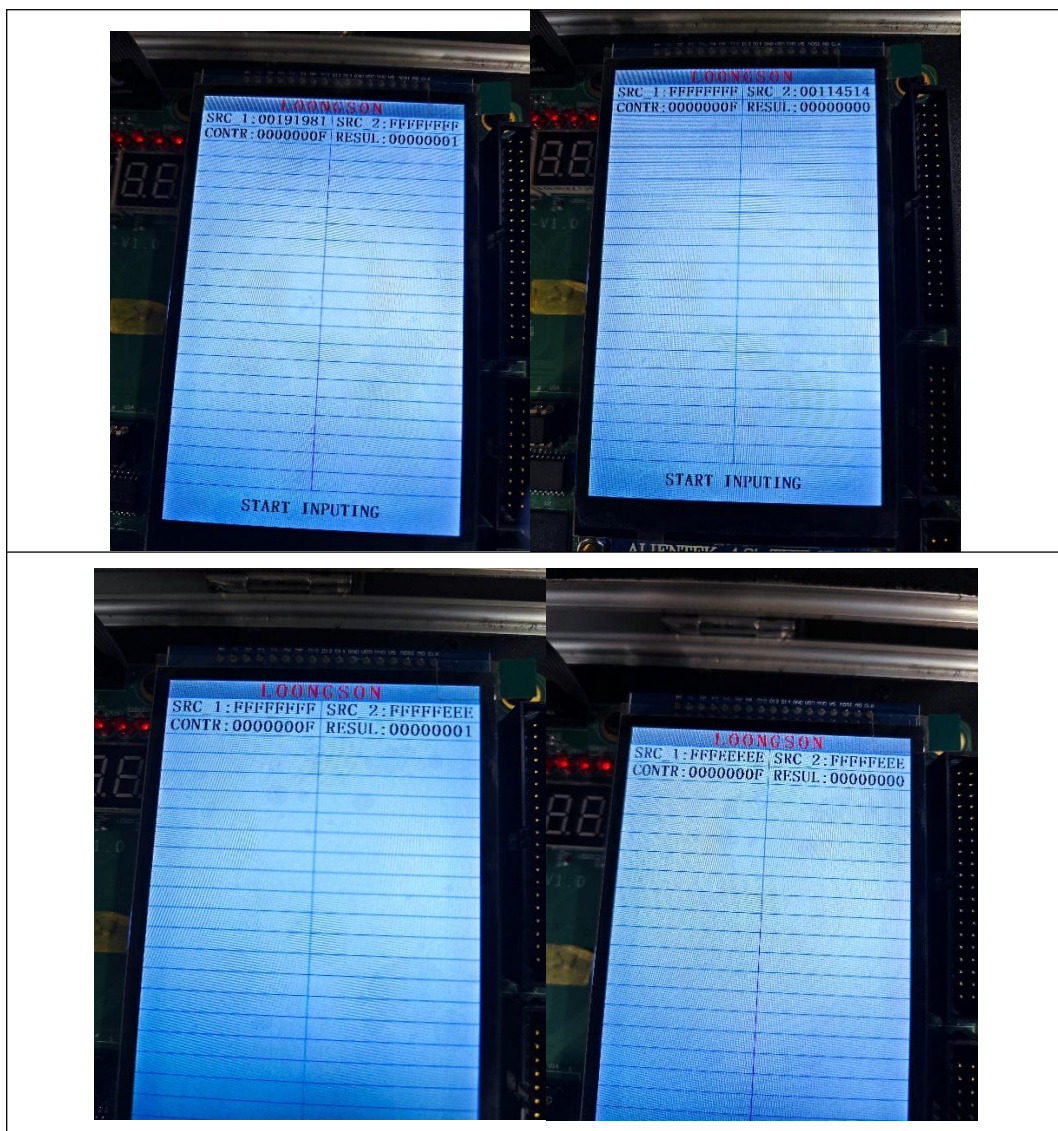
一、验证实验初始的 12 种运算

控制信号	ALU 操作	Src1	Src2	result
000000000000	无	\	\	\
100000000000	加法	FFFFFFFF	0000000F	0000000E
010000000000	减法	0000FFFF	0000EEEE	00001111
001000000000	有符号比较, 小于置位	FFFFFFFF	7FFFFFFF	00000001
000100000000	无符号比较, 小于置位	7FFFFFFF	FFFFFFFF	00000001
000010000000	按位与	87654321	12345678	02244220
000001000000	按位或非	87654321	12345678	688AA886
000000100000	按位或	87654321	12345678	97755779
000000010000	按位异或	87654321	12345678	95511559
000000001000	逻辑左移	00000005	00000001	00100000
000000000100	逻辑右移	00000005	10000000	00000001
000000000010	算数右移	00000005	10000000	11111100
000000000001	高位加载	\	12345678	56780000

二、改进实验后的 12+3 种运算的验证

控制信号	ALU 操作	Src1	Src2	result
0000	无	\	\	\
0001	加法	FFFFFFFF	0000000F	0000000E
0010	减法	0000FFFF	0000EEEE	00001111
0011	有符号比较, 小于置位	FFFFFFFF	7FFFFFFF	00000001
0100	无符号比较, 小于置位	7FFFFFFF	FFFFFFFF	00000001
0101	按位与	87654321	12345678	02244220
0110	按位或非	87654321	12345678	688AA886
0111	按位或	87654321	12345678	97755779
1000	按位异或	87654321	12345678	95511559
1001	逻辑左移	00000005	00000001	00100000
1010	逻辑右移	00000005	10000000	00000001
1011	算数右移	00000005	10000000	11111100
1100	高位加载	\	12345678	56780000

1101	低 16 位和高 16 位交换	\	12345678	56781234
				
1110	按位同或	00114514	00191981	FFF7A36A
				
1111	无符号比较，大于置位	如下各图	如下各图	如下各图
<div></div>				



6、 总结感想

在完成这次 ALU 实验改进后，我深感收获颇丰。首先，我通过对操作码进行位压缩以及扩展运算种类的操作，学会了如何在有限的位宽内灵活调整运算功能，并在实际硬件中验证其可行性。这对于我在之后的数字电路设计中，能够更好地利用有限的资源来提高硬件性能具有极大的帮助。

其次，我通过直接在实验箱上验证实验结果，学会了将理论知识与实际操作相结合的能力。在实际操作中，我对原理图和 verilog 代码有了更深刻的理解，同时也锻炼了我分析和解决问题的能力。