组成原理实验课程第四次实验报告

实验名称	ALU 模块实验			班级	张金
学生姓名	冯思程	学号	2112213	指导老师	董前琨
实验地点	实验楼 A306		实验时间	2023.5.8 14：00	

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

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。

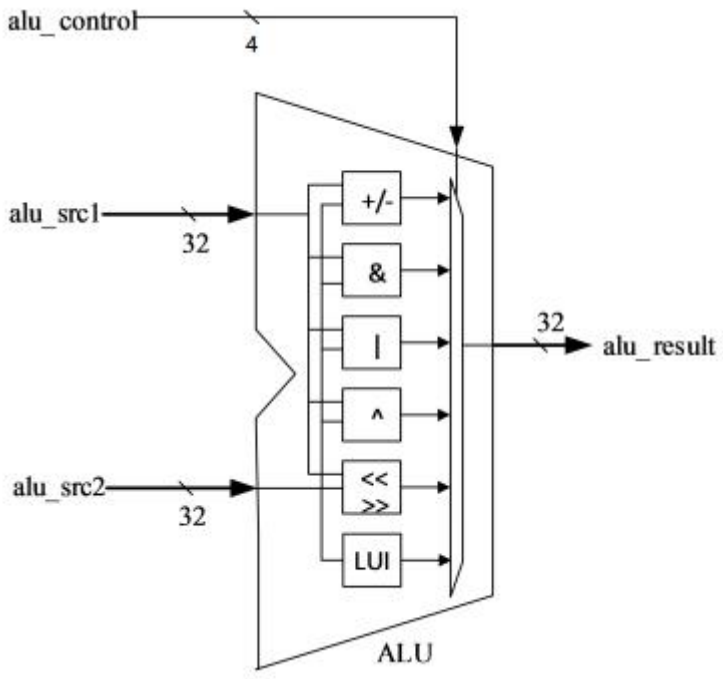
3、本次实验不用仿真波形，直接上实验箱验证即可。

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，并提交。

3、 实验原理图

修改后的 ALU 实验原理图如下： （将 alu_control 信号改成了 4 位）



原理图说明：共有三个输入，分别为控制信号、操作数 1、操作数 2，然后根据控制信

号选择进行运算的种类，然后通过源操作数进行运算，返回结果，中间聚合了多种运算的模块就是 ALU。

4、 实验步骤

1）对 alu.v 的修改：（用红色加粗部分表示修改或添加的语句）

i.对后三种运算的定义以及对控制信号的输入修改，代码如下：

```
module alu(
    // input    [11:0] alu_control,
    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;   //高低 16 位颠倒
    wire alu_nxor;    //按位同或
    wire alu_blt;     //有符号大于则置位
    assign alu_add   =(alu_control==4'b0001 ? 1 : 0);//等于 1 时为加法
    assign alu_sub   =(alu_control==4'b0010 ? 1 : 0);//等于 2 时为减法
    assign alu_slt   =(alu_control==4'b0011 ? 1 : 0);//以此类推
    assign alu_sltu =(alu_control==4'b0100 ? 1 : 0);
    assign alu_and   =(alu_control==4'b0101 ? 1 : 0);
    assign alu_nor   =(alu_control==4'b0110 ? 1 : 0);
    assign alu_or    =(alu_control==4'b0111 ? 1 : 0);
    assign alu_xor   =(alu_control==4'b1000 ? 1 : 0);
    assign alu_sll   =(alu_control==4'b1001 ? 1 : 0);
    assign alu_srl   =(alu_control==4'b1010 ? 1 : 0);
    assign alu_sra   =(alu_control==4'b1011 ? 1 : 0);
    assign alu_lui   =(alu_control==4'b1100 ? 1 : 0);
    assign alu_addi   =(alu_control==4'b1101 ? 1 : 0);//高 16 位和低 16 位颠倒
    assign alu_nxor   =(alu_control==4'b1110 ? 1 : 0);//按位同或
```

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

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;
```

li.添加针对按位同或和高低 16 位颠倒的运算结果，其中按位同或的运算结果就是按位异或的结果按位取反即可，颠倒高低 16 位运算就是将高低 16 位调换后拼接上即可，代码如下：

```
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]};//颠倒高低 16 位
```

lii.添加对有符号的大于则置位运算的运算结果，将两个操作数相减来验证结果，结果共有六种情况，如下。在这六种情况中，划线的三种为大于需要置位的情况，但需要注意，符号位为 0 包含了 0 的情况，故而需要在逻辑实现中去掉 SRC_1=SRC_2 即相减结果等于 0 的情况，代码如下：

```
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));
```

6 种情况说明简表:

```
//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, 说明小于
// 0                0                0 相减为正, 说明大于等于
// 1                1                1 相减为负, 说明小于
```

```
// 1              1              0 相减为正，说明大于等于
// 1              0              X(0 或 1) "负-正"，显然大于不成立
```

2）对 alu_display.v 的修改：（用红色加粗部分表示修改或添加的语句）

i.对 alu_control 信号长度定义进行修改：

```
reg    [3:0] alu_control;  // ALU 控制信号
```

li.对从触摸屏获取输入模块进行修改，主要修改控制信号的部分，修改原控制信号长度为 4 位，代码如下：

```
always @(posedge clk)
    begin
        if (!resetn)
        begin
            alu_control <= 4'd0;
        end
        else if (input_valid && input_sel==2'b00)
        begin
            alu_control <= input_value[3:0];
        end
    End
```

3）这里我们还有一个加法器模块 adder_module，这个模块会在进行 add、sub、slt、sltu、blt 运算时候被调用，源代码中进行书写了正确的代码，这里无需进行修改。

4）本实验进行仿真的意义不大而且仿真代码非常繁琐复杂，故不编写仿真文件来进行仿真，可以跳过仿真步骤直接将修改后的代码上箱验证。

5）将源码给出的约束文件导入和同时导入 lcd 屏模块，然后再进行上箱实验验证。其中约束文件可以直接使用，无需再进行修改。lcd 屏模块也在源码文件（source code）中给出，直接导入。然后分别依次跑综合、增强后确认无误，将电脑连接到实验箱后生成流文件到实验箱上，然后在实验箱上进行调试观察，验证是否完整的实现了目标功能。

5、 实验结果分析

原 12 种运算的功能说明与运算验证：

控制信号												ALU 操作	SRC1	SRC2	RESUL
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	无	无	无
1	0	0	0	0	0	0	0	0	0	0	0	加法	11111111	12345678	23456789
0	1	0	0	0	0	0	0	0	0	0	0	减法	11451400	11111100	00340300
0	0	1	0	0	0	0	0	0	0	0	0	有符号比较，小于置位	F0000000	00000001	00000001
0	0	0	1	0	0	0	0	0	0	0	0	无符号比较，小于置位	11000000	11114514	00000001
0	0	0	0	1	0	0	0	0	0	0	0	按位与	FFFFFFF1	12383126	12383120
0	0	0	0	0	1	0	0	0	0	0	0	按位或非	00000000	00000010	FFFFFFEF
0	0	0	0	0	0	1	0	0	0	0	0	按位或	00000001	00000010	00000011
0	0	0	0	0	0	0	1	0	0	0	0	按位异或	12340001	12340010	00000011
0	0	0	0	0	0	0	0	1	0	0	0	逻辑左移	00000001	00000002	00000004

0 0 0 0 0 0 0 0 0 0 1 0 0	逻辑右移	00000001	00000008	00000004
0 0 0 0 0 0 0 0 0 0 0 1 0	算术右移	00000001	F0000000	F8000000
0 0 0 0 0 0 0 0 0 0 0 0 1	高位加载	29384203	23849239	9239000

上箱结果验证：

1）按钮说明：

这里通过 input_sel 来控制输入的东西，input_sel 是两位的控制信号，分别对应按钮左起的第二个和最后一个，其中第二位是 input_sel[1],最后一个是 input_sel[0]。

2）功能说明：（共有 15 种功能，如下表中）

控制信号				ALU 操作
3	2	1	0	
0	0	0	0	无
0	0	0	1	加法
0	0	1	0	减法
0	0	1	1	有符号比较，小于置位
0	1	0	0	无符号比较，小于置位
0	1	0	1	按位与
0	1	1	0	按位或非
0	1	1	1	按位或
1	0	0	0	按位异或
1	0	0	1	逻辑左移
1	0	1	0	逻辑右移
1	0	1	1	算术右移
1	1	0	0	高位加载
1	1	0	1	高低位颠倒
1	1	1	0	按位同或
1	1	1	1	无符号大于则置位

3）验证说明：

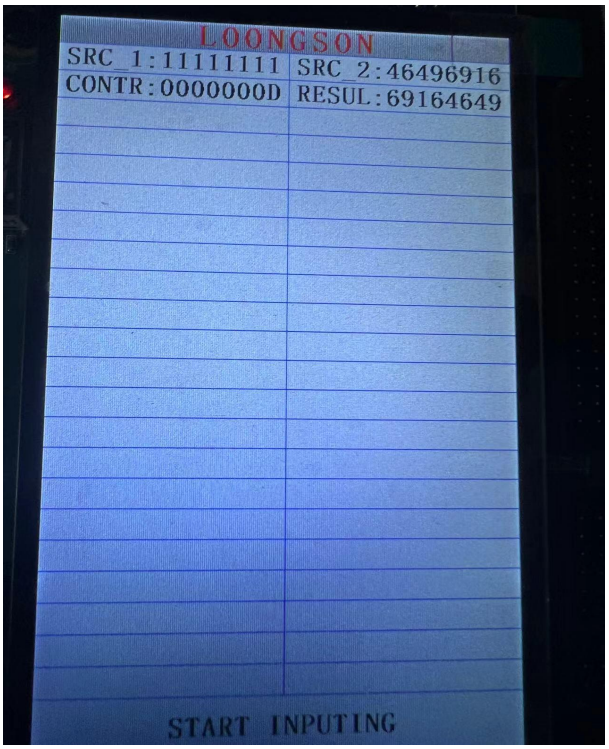
目标功能示例表：

ALU 操作	SRC1	SRC2	RESUL
加法	00000001	00000001	00000002
减法	00000001	00000001	00000000
有符号比较，小于置位	FF000002	00000001	00000001
无符号比较，小于置位	11000232	11239384	00000001
按位与	11111111	12345678	10101010
按位或非	00000000	EEEEEEEE	11111111
按位或	12345601	12345610	12345611
按位异或	00000001	00000001	00000000
逻辑左移	00000002	00000004	00000010

逻辑右移	00000002	00000004	00000001
算术右移	00000001	FF000000	FF800000
高位加载	00000000	21824912	49120000
高低位颠倒	11111111	46496916	69164649
按位同或	11111111	1111EEEE	FFFF0000
无符号大于则置位	11111111	00002323	00000001

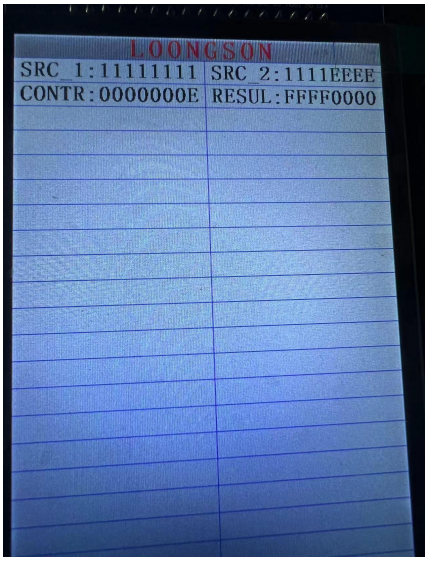
新增功能验证图片以及说明：

i.新增功能 1：高低 16 位调换，这个运算针对的是源操作数 2（SRC2）



如图中：SRC2 是 46496916，运算后变成了 69164649，成功实现了功能。

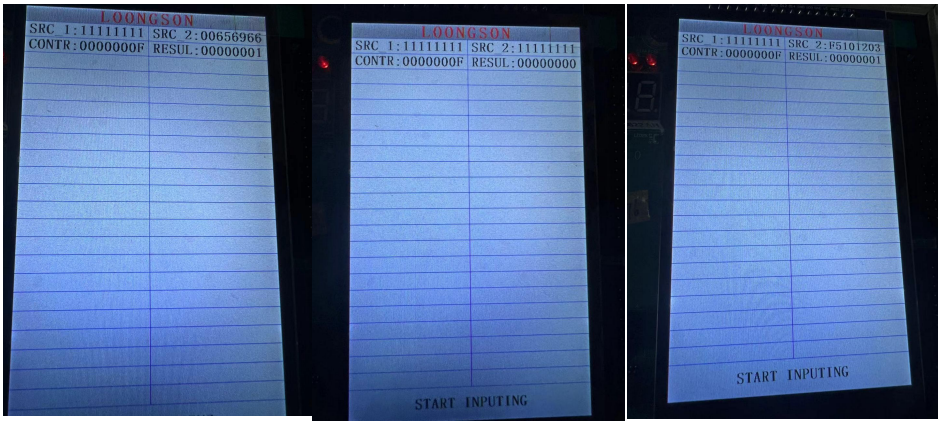
Ii.新增功能 2：按位同或，相同置 1，否则置 0。运算针对两个源操作数



上图中 SRC1 是 11111111，SRC2 是 1111EEEE，前 16 位是完全相同的所以都置 1，后面的 4

个字节 1 是 0001，E 是 1110，恰好都不同，所以都置 1，结果都是 0。所以最后的结果是 FFFF0000，结果正确，成功的实现了功能。

Iii.新增功能 3：有符号的大于则置位，这里用三张图片进行说明，分别是正数和负数和相等的情况。



正数　　相等　　负数

第一张图中，正数与正数比较，SRC1 大于 SRC2，所以大于置 1；
第二张图中，两个数相等，不符合条件，不置 1；
第三张图中，正数与负数比较，正数大于负数，所以大于置 1
综上，结果均正确，成功的实现了目标功能。

6、 总结感想

这个实验让我深入了解了 MIPS 指令集中的运算指令的原理和分类，掌握了 ALU 的原理、功能和设计，并加强了我的 verilog 语言电路设计能力。在实验中，我通过对原有的操作码进行位压缩和调整操作码控制信号位宽为 4 位，自行补充 3 种不同类型的运算，加深了对 ALU 的理解和应用。

在实验过程中，我遇到了一些挑战和困难，如操作码的调整和实验箱的验证。然而，通过与同学的交流和合作，我成功地克服了这些问题，并最终获得了满意的实验结果。

总之，这个实验让我获得了宝贵的学习经验和实践机会，让我对 verilog 语言有了更深入的理解和应用，同时也为我的未来学习和实践打下了坚实的基础，也为后续设计 cpu 的实验打下基础。