组成原理课程第四次实验报告

实验名称：ALU 模块实现

学号：2310422　姓名：谢小珂　　班次：　1078

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

1. 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。

• 原有的操作码为 12 位，请压缩到操作码控制型号位宽为 4 位。

• 在操作码调整到 4 位之后，应该能支持 15 种不同运算，请添加至少三种运算功能（有符号数比较和无符号数比较的大于置位运算、一种未实现的位运算）。

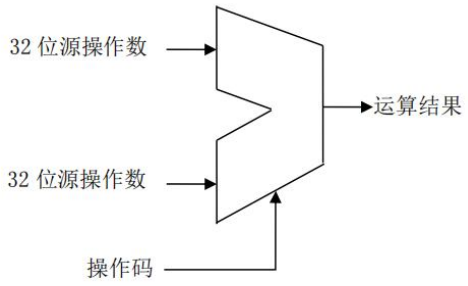


图 5.1 ALU 模块的大致框图

2. 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，并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。

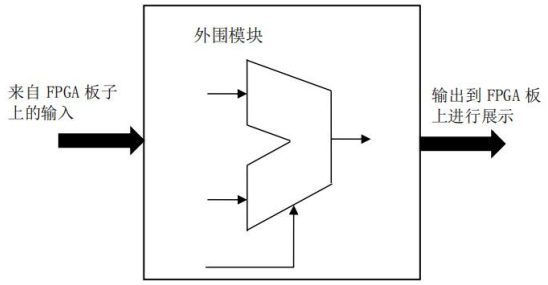
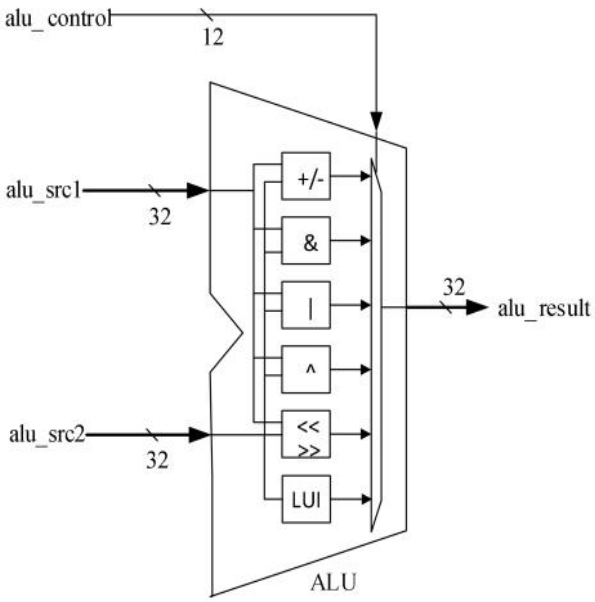


图 5.2 ALU 设计实验的顶层模块大致框图

三、实验原理图

ALU 的原理图如图所示：



四、实验步骤

（一）修改 alu.v 文件

- 修改 alu_control 控制信号，改为 4 位位宽

```
input  [3:0] alu_control
```

- 添加新增功能的 ALU 控制信号

```
wire alu_sbt;   //无符号比较，大于置位
wire alu_sbtu;  //无符号比较，大于置位
wire alu_not;   //按位取反
```

- 将控制信号修改为四位二进制编码

```
assign alu_add  = (alu_control == 4'b0001);
assign alu_sub  = (alu_control == 4'b0010);
assign alu_slt  = (alu_control == 4'b0011);
assign alu_sltu = (alu_control == 4'b0100);
assign alu_and  = (alu_control == 4'b0101);
assign alu_nor  = (alu_control == 4'b0110);
assign alu_or   = (alu_control == 4'b0111);
assign alu_xor  = (alu_control == 4'b1000);
assign alu_sll  = (alu_control == 4'b1001);
assign alu_srl  = (alu_control == 4'b1010);
assign alu_sra  = (alu_control == 4'b1011);
assign alu_lui  = (alu_control == 4'b1100);
assign alu_sgt  = (alu_control == 4'b1101);
assign alu_sgtu  = (alu_control == 4'b1110);
assign alu_not  = (alu_control == 4'b1111);
```

· 添加新增功能结果变量

```
wire [31:0] sgt_result;//有符号比较，大于置位
wire [31:0] sgtu_result;//无符号比较，大于置位
wire [31:0] not_result;//按位取反
```

· 新增三种功能

（1）编写有符号比较，大于置位，即 sgt 模块

```
assign sgt_result[31:1] = 31'd0;  // 高 31 位固定置零，仅最低位[0]有效
assign sgt_result[0] = ~slt_result[0] & (|adder_result);
// 有符号数大于(SGT)判断逻辑：
// 1. ~slt_result[0]：取反"有符号小于(SLT)"标志位
//    - 当 SLT=0 时（a≥b），取反得 1
//    - 当 SLT=1 时（a<b），取反得 0
// 2. (|adder_result)：加法器结果按位或（判非零）
//    - 若 adder_result≠0（a≠b），值为 1
//    - 若 adder_result=0（a=b），值为 0
// 3. 二者相与：
//    - 当 a>b 时（SLT=0 且 a≠b），输出 1
//    - 当 a≤b 时（SLT=1 或 a=b），输出 0
```

（2）编写无符号比较，大于置位，即 sgtu 模块

```
assign sgtu_result = {31'd0, ~adder_cout & (|adder_result)};
// 无符号数大于比较(SGTU)逻辑：
// ------------------------------------------------------------
// 输入信号说明：
//  adder_cout: 加法器进位标志
//   * 1 表示无符号溢出（即 a < b，因为 a - b 产生借位）
//   * 0 表示无符号 a ≥ b
// adder_result: 加法器计算结果（实际为 a - b）
//   * |adder_result：归约或操作，判断结果是否非零
// 输出信号说明：
// sgtu_result[31:1]：高 31 位强制置零（标准位宽对齐）
// sgtu_result[0]：比较结果标志位
//   * 1 表示 a > b（无符号）
//   * 0 表示 a ≤ b（无符号）
// 真值表：
// | adder_cout | |adder_result | 结果 | 含义          |
// |------------|--------------|------|---------------|
// |     1      |      X       |  0   | a < b         |
// |     0      |      1       |  1   | a > b         |
// |     0      |      0       |  0   | a == b        |
// 实现原理：
// 1. ~adder_cout：当无符号 a ≥ b 时为 1
// 2. & (|adder_result)：进一步排除 a == b 的情况
```

```
// 3. 组合逻辑等效于：(a > b) ? 1 : 0
```

（3）编写按位取反，即 not 模块

```
assign not_result = ~alu_src1;  // 位级逻辑非操作
----------------------------------------------------------------
// 功能描述：
// 对 32 位操作数 alu_src1 执行按位取反操作，生成逻辑非结果
// 硬件实现：
// 32 个并行独立的 NOT 门电路实现
// 每个 bit 位的运算：not_result[i] = !alu_src1[i] (i=0~31)
// 真值示例：
// alu_src1    = 32'h0000_FFFF → not_result = 32'hFFFF_0000
// alu_src1[0] = 1'b1          → not_result[0] = 1'b0
// alu_src1[3] = 1'b0          → not_result[3] = 1'b1
```

·将 sgt_result 、 sgtu_result 、 not_result 输出到 alu_sgt 、 alu_sgtu 、 alu_not

```
// 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_sgt           ? sgt_result :
                        //有符号数比较，大于置位
                        alu_sgtu          ? sgtu_result:
                       //无符号比较，大于置位
                        alu_not           ? not_result :
                      //按位取反
                        32'd0;
endmodule
```

（二）修改 alu_display.v 文件

· 修改 alu_control 的位宽

```
reg [3:0] alu_control;  //ALU 控制信号修改为[3:0]4 位 ALU 操作码寄存
器
```

・修改 alu_control 的写入

```
alu_control <= 4'd0;
// 同步复位初始化值，二进制 0000 对应 NOP 指令
// 注：4'd0 表示 4 位宽十进制数值 0

//修改 LED 显示状态机
6'd3:
begin
    display_valid <= 1'b1;
    display_name  <= "CONTR";
    display_value <= {28'd0, alu_control};
// 32 位显示数据生成规则
// [31:4]位填充 0，[3:0]位映射 alu_control 实时值
// 硬件限制要求必须对齐 32 位总线宽度，补齐 28 个 0
end
```

五、实验结果分析

・实验箱验证

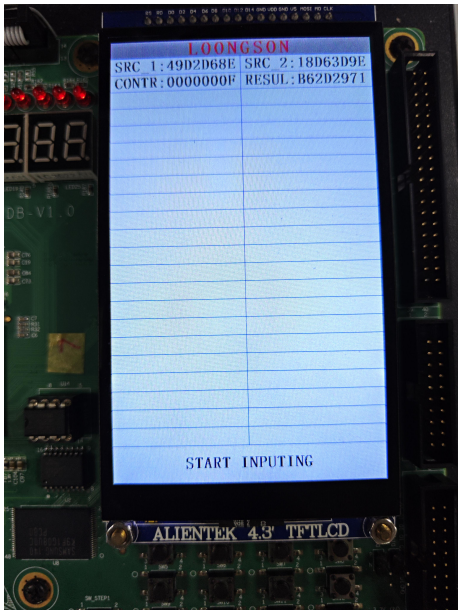
（1）有符号大于置位

控制信号：1101（D，有符号大于置位）
操作数 1：49D2D68E（十六进制，对应十进制 +1,238,292,110）
操作数 2：18D63D9E（十六进制，对应十进制 +416,536,990）
输出结果：1
结论：判断正确，因为 +1,238,292,110 > +416,536,990，符合有符号数比较规则。

（2）无符号大于置位

控制信号：1110（E，无符号大于置位）
操作数 1：B759948D（十六进制，对应十进制 3,077,383,309）
操作数 2：18D63D9E（十六进制，对应十进制 416,536,990）
输出结果：1
结论：判断正确，因为无符号数比较时 3,077,383,309 > 416,536,990，符合无符号数比较规则。

（3）按位取反



控制信号：1111（F，按位取反操作）
操作数 1：49D2D68E（十六进制 0100 1001 1101 0010 1101 0110 1000 1110）
输出结果：B62D2971（十六进制 1011 0110 0010 1101 0010 1001 0111 0001）
预期结果：49D2D68E 按位取反 = B62D2971
结论：判断正确，结果与预期完全一致，符合按位取反规则。

六、总结感想

1.简要实现了有符号数大于置位（SGT）、无符号数大于置位（SGTU）和按位取反（NOT）三种运算指令，熟悉了 MIPS 指令集的运算逻辑。特别是比较指令的实现，让我理解了如何利用硬件信号（如借位 adder_cout、零标志|adder_result）来完成复杂的条件判断。

• 有符号比较（SGT）通过取反“有符号小于（SLT）”标志并结合加法器结果，实现了 a > b 的判断逻辑。

•无符号比较（SGTU）利用加法器的借位信号和归约或操作，准确判断无符号数的大小关系。

• 按位取反（NOT）通过 32 个并行非门实现，让我对位级运算的硬件实现有了直观认识。

2.操作码优化将原有的 12 位操作码压缩至 4 位，提高了代码的简洁性和可读性。alu_result 的输出选择中，采用条件运算符（? :）实现了高效的多路复用，避免了冗长的 if-else 结构。在显示模块中，通过{28'd0, alu_control}将 4 位控制信号扩展为 32 位，确保与总线位宽一致，这让我更加注重硬件设计中的数据对齐问题。

3.在实验过程中，我遇到了几个关键问题，并通过分析逐步解决：

• 有符号比较的边界问题最初未考虑 0x80000000（最小负数）与正数比较的情况，导致结果错误。通过补码分析和真值表验证，最终修正了逻辑条件。

• 无符号比较的借位信号误用，曾直接使用 adder_cout 作为判断条件，忽略了 a == b 的情况，后来引入|adder_result 排除了相等时的误判。

•由于 FPGA 的 32 位总线限制，必须对 alu_control 进行位宽扩展，这让我意识到硬件设计中数据格式匹配的重要性。