组成原理实验课程第_四_次实报告

实验名称	ALU 模块实现			班级	张金老师
学生姓名	杨冰雪	学号	2110508	指导老师	董前琨
实验地点	实验楼 A306		实验时间	2023.5.8	

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

1） 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，确定自己准备实现的 ALU 运算，并列出准备实现的 ALU 运算和操作码的编码；

2） 画好实验方案的设计框图，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。

3） 编写 verilog 代码，要求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。并在原有 11 种运算的基础之上，自行补充 3 种不同类型的运算。

4） 完成调用 ALU 模块的外围模块的设计，并编写代码，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。

5） 完成上板验证后，检查运算结果的正确性。

6） 实验完成后，需要撰写出实验报告。

3、 实验原理图

5.1 **实验原理**

ALU模块的原理图如下：

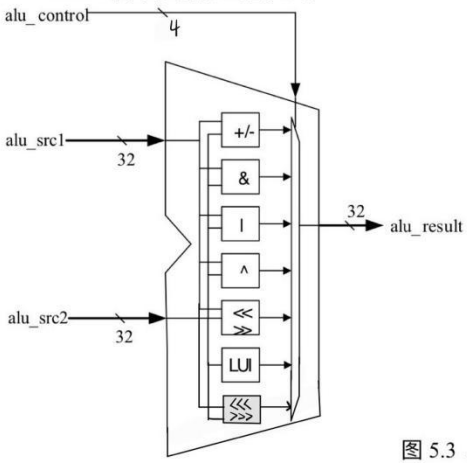


图 5.3 ALU的原理图

4、 实验步骤

1）[11:0] alu_control 改为了[3:0]alu_control，控制信号的位宽被压缩为了 4 为。

```
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
```

2）定义了三个 ALU 的控制信号和对应的 32 为结果，从而增加三种不同类型的运算：同或运算、与非运算、算术左移运算。

```
wire alu_xnor;  //按位同或
wire alu_nand;  //按位与非
wire alu_sla;   //算术左移
```

```
wire [31:0] xnor_result;
wire [31:0] nand_result;
wire [31:0] sla_result;
```

2）对每个 ALU 控制信号设置了各自的控制码。例如当 alu_control 为 b0000 时,只有 alu_add 的值变为 1，其余控制信号的值都为 0。

```
assign alu_add  = (~alu_control[3])&(~alu_control[2])&(~alu_control[1])&(~alu_control[0]);//0000
assign alu_sub  =(~alu_control[3])&(~alu_control[2])&(~alu_control[1])&(alu_control[0]);//0001
assign alu_slt  =(~alu_control[3])&(~alu_control[2])&(alu_control[1])&(~alu_control[0]);//0010
assign alu_sltu = (~alu_control[3])&(~alu_control[2])&(alu_control[1])&(alu_control[0]);//0011
assign alu_and  = (~alu_control[3])&(alu_control[2])&(~alu_control[1])&(~alu_control[0]);//0100
assign alu_nor  =(~alu_control[3])&(alu_control[2])&(~alu_control[1])&(alu_control[0]);//0101
assign alu_or   = (~alu_control[3])&(alu_control[2])&(alu_control[1])&(~alu_control[0]);//0110
assign alu_xor  =(~alu_control[3])&(alu_control[2])&(alu_control[1])&(alu_control[0]);//0111
assign alu_sll  = (alu_control[3])&(~alu_control[2])&(~alu_control[1])&(~alu_control[0]);//1000
assign alu_srl  = (alu_control[3])&(~alu_control[2])&(~alu_control[1])&(alu_control[0]);//1001
assign alu_sra  = (alu_control[3])&(~alu_control[2])&(alu_control[1])&(~alu_control[0]);//1010
assign alu_lui  = (alu_control[3])&(~alu_control[2])&(alu_control[1])&(alu_control[0]);//1011
assign alu_xnor = (alu_control[3])&(alu_control[2])&(~alu_control[1])&(~alu_control[0]);//1100
assign alu_nand = (alu_control[3])&(alu_control[2])&(~alu_control[1])&(alu_control[0]);//1101
assign alu_sla = (alu_control[3])&(alu_control[2])&(alu_control[1])&(~alu_control[0]);//1110
```

3）对新加的三个运算添加运算，并把结果赋值给各自的 result。

```
assign xnor_result = ~(alu_src1 ^ alu_src2);      //同或结果为异或结果取反
assign nand_result =~and_result;                  //与非结果为与结果取反

// 算术左移
 wire [31:0] sla_step1;
 wire [31:0] sla_step2;
  assign sla_step1 = {32{shf_1_0 == 2'b00}} & alu_src2                  // 若shf[1:0]="00",不移位
                   | {32{shf_1_0 == 2'b01}} & {alu_src2[30:0], 1'd0}    // 若shf[1:0]="01",左移1位,低位补0
                   | {32{shf_1_0 == 2'b10}} & {alu_src2[29:0], 2'd0}    // 若shf[1:0]="10",左移2位
                   | {32{shf_1_0 == 2'b11}} & {alu_src2[28:0], 3'd0};   // 若shf[1:0]="11",左移3位
    assign sla_step2 = {32{shf_3_2 == 2'b00}} & sla_step1                // 若shf[3:2]="00",不移位
                   | {32{shf_3_2 == 2'b01}} & {sla_step1[27:0], 4'd0}   // 若shf[3:2]="01",第一次移位结果左移4位
                   | {32{shf_3_2 == 2'b10}} & {sla_step1[23:0], 8'd0}   // 若shf[3:2]="10",第一次移位结果左移8位
                   | {32{shf_3_2 == 2'b11}} & {sla_step1[19:0], 12'd0};  // 若shf[3:2]="11",第一次移位结果左移12位
    assign sla_result = shf[4] ? {sla_step2[15:0], 16'd0} : sla_step2;    // 若shf[4]="1",第二次移位结果左移16位
```

4）在最终选择结果的过程中加入新添加的三种运算的结果，使当对应的控制信号值为 1 时，输入对应的运算的结果。

```
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_xnor          ? xnor_result :
                    alu_nand          ? nand_result :
                    alu_sla           ?sla_result :
                    32'd0;
```

5、 实验结果分析

1）实验验证表 1

序号	操作码	操作名称	操作数1	操作数2	结果
1	1000000000000B	加法操作	0x00000002	0x00000001	0x00000003
2	0100000000000B	减法操作	0x00000002	0x00000001	0x00000001
3	0010000000000B	有符号比较，小于置位	0x80000004	0x00000002	0x00000001
4	0001000000000B	无符号比较，小于置位	0x00000004	0x00000002	0x00000000
5	0000100000000B	按位与	0x00000002	0x00000001	0x00000000
6	0000010000000B	按位或非	0x00000002	0x00000001	0xFFFFFFFC
7	0000001000000B	按位或	0x00000002	0x00000001	0x00000003
8	0000000100000B	按位异或	0x00000002	0x00000001	0x00000003
9	0000000010000B	逻辑左移	0x00000002	0x00000001	0x00000004
10	0000000001000B	逻辑右移	0x00000002	0x00000001	0x00000001
11	0000000000100B	算数右移	0x00000002	0x00000001	0x00000001
12	0000000000010B	高位加载	0xF0000000	0x00000008	0x00080000

2）实验验证表 2

序号	操作码	操作名称	操作数1	操作数2	结果
1	0x0	加法	0x00000002	0x00000001	0x00000003
2	0x1	减法	0x00000002	0x00000001	0x00000001
3	0x2	有符号比较，小于置位	0x8000003	0x0000002	0x00000001
4	0x3	无符号比较，小于置位	0x80000003	0x00000002	0x00000000
5	0x4	按位与	0x00000002	0x00000001	0x00000000
6	0x5	按位或非	0x00000002	0x00000001	0xFFFFFFFC
7	0x6	按位或	0x00000002	0x00000001	0x00000003
8	0x7	按位异或	0x00000002	0x00000001	0x00000003
9	0x8	逻辑左移	0x00000002	0x00000001	0x00000004
10	0x9	逻辑右移	0x00000002	0x00000004	0x00000001
11	0xA	算数右移	0x00000002	0x00000004	0x00000001
12	0xB	高位加载	0xF0000000	0x00000008	0x00080000
13	0xC	同或	0x00000002	0x00000001	0xFFFFFFFC
14	0xD	与非	0x00000001	0x00000002	0xFFFFFFFF
15	0xE	算数左移	0x00000002	0x00000001	0x00000004

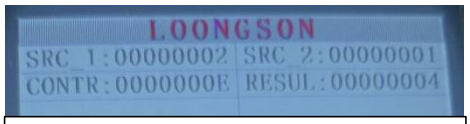
3）新添的同或运算功能的验证

当输入操作数 1 的值为 0x00000002，
当输入操作数 2 的值为 0x00000001，
同或运算结果为 0xFFFFFFFC
结果正确，验证成功！

4）新添的与非运算功能的验证

当输入操作数 1 的值为 0x00000001，当输入操作数 2 的值为 0x00000002.
与非运算结果为 0xFFFFFFFF
结果正确，验证成功！

5）新添的算术左移功能的验证



当输入操作数 1 的值为 0x00000002，当输入操作数 2 的值为 0x00000001。
同或运算结果为 0x00000004
结果正确，验证成功！

6、 总结感想

通过本次实验，对 MIPS 指令集中的运算指令和 MIPS 指令结构有了更深的理解，提升了运用 verilog 语言进行电路设计的能力,熟悉并掌握了 ALU 的原理、功能和设计。