第一章 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。

第二章 实验任务与要求

2.1 试验任务

1. 学习 MIPS 指令集，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。
2. 归纳确定自己本次实验中准备实现的 ALU 运算，要求不实现定点乘除指令和浮点运算指令，要求至少实现 5 种 ALU 运算，其中要包含加减运算，其中减法在内部要转换为加法，与加法运算共同调用实验一里自己完成的加法模块去做。
3. 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。图 5.1 中的操作码位数和类型请自行设计，可以设计为独热码（一位有效编码）或二进制编码。比如，设计方案中预定实现 7 种 ALU 运算，则操作码采用独热码，则需 7bit 数据，每位单独指示一种运算；若采用二进制编码，则只用 3bit 数据位即可，但在需 ALU 内部先进行解码，才能确定 ALU 作何种运算。
4. 根据设计的实验方案，使用 verilog 编写相应代码。
5. 对编写的代码进行仿真，得到正确的波形图。
6. 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, 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。操作码可以考虑用 LCD 触摸屏输入，也可以用拨码开关输入。
7. 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。

2.2 实验要求

1. 实验箱结果截图（可按小组截图）
2. 代码注释（单独注释，着重说明 16 位所涉及的地方）

第三章 实验结果

3.1 实验代码

3.1.1 加法器

```
module adder(
    input  [31:0] operand1, // 32位输入
    input  [31:0] operand2, // 32位输入
    input         cin,      // 进位
    output [31:0] result,   // 32位输出
    output        cout      // 进位输出
    );
    assign {cout,result} = operand1 + operand2 + cin;

endmodule
```

3.1.2 ALU

```
module alu(
    input  [11:0] 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
```

```
18      wire alu_srl;    //逻辑右移
19      wire alu_sra;    //算术右移
20      wire alu_lui;    //高位加载
21
22      assign alu_add  = alu_control[11];
23      assign alu_sub  = alu_control[10];
24      assign alu_slt  = alu_control[ 9];
25      assign alu_sltu = alu_control[ 8];
26      assign alu_and  = alu_control[ 7];
27      assign alu_nor  = alu_control[ 6];
28      assign alu_or   = alu_control[ 5];
29      assign alu_xor  = alu_control[ 4];
30      assign alu_sll  = alu_control[ 3];
31      assign alu_srl  = alu_control[ 2];
32      assign alu_sra  = alu_control[ 1];
33      assign alu_lui  = alu_control[ 0];
34
35      wire [31:0] add_sub_result;
36      wire [31:0] slt_result;
37      wire [31:0] sltu_result;
38      wire [31:0] and_result;
39      wire [31:0] nor_result;
40      wire [31:0] or_result;
41      wire [31:0] xor_result;
42      wire [31:0] sll_result;
43      wire [31:0] srl_result;
44      wire [31:0] sra_result;
45      wire [31:0] lui_result;
46
47      assign and_result = alu_src1 & alu_src2;       // 与结果为两数按
            位与
48      assign or_result  = alu_src1 | alu_src2;       // 或结果为两数按
            位或
49      assign nor_result = ~or_result;                // 或非结果为或结
            果按位取反
50      assign xor_result = alu_src1 ^ alu_src2;       // 异或结果为两数
            按位异或
```

```
51      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
52
53  //-----{加法器}begin
54  //add,sub,slt,sltu均使用该模块
55      wire [31:0] adder_operand1;
56      wire [31:0] adder_operand2;
57      wire        adder_cin     ;
58      wire [31:0] adder_result  ;
59      wire        adder_cout    ;
60      assign adder_operand1 = alu_src1;
61      assign adder_operand2 = alu_add ? alu_src2 : ~alu_src2;
62      assign adder_cin      = ~alu_add; //减法需要cin
63      adder adder_module(
64      .operand1(adder_operand1),
65      .operand2(adder_operand2),
66      .cin     (adder_cin     ),
67      .result  (adder_result  ),
68      .cout    (adder_cout    )
69      );
70
71      //加减结果
72      assign add_sub_result = adder_result;
73
74      //slt结果
75      //adder_src1[31] adder_src2[31] adder_result[31]
76      //        0              1           X(0或1)        "正-负", 显然小于不成立
77      //        0              0             1            相减为负, 说明小于
78      //        0              0             0            相减为正, 说明不小于
79      //        1              1             1            相减为负, 说明小于
80      //        1              1             0            相减为正, 说明不小于
```

```
81      //        1              0            X(0或1)        "负-正", 显然小
            于成立
82      assign slt_result[31:1] = 31'd0;
83      assign slt_result[0]    = (alu_src1[31] & ~alu_src2[31]) | (~(
            alu_src1[31]^alu_src2[31]) & adder_result[31]);
84
85      //sltu结果
86      //对于32位无符号数比较, 相当于33位有符号数({1'b0,src1}和{1'b0,
            src2})的比较, 最高位0为符号位
87      //故, 可以用33位加法器来比较大小, 需要对{1'b0,src2}取反,即需要
            {1'b0,src1}+{1'b1,~src2}+cin
88      //但此处用的为32位加法器, 只做了运算:
                                          src1    +     ~src2    +cin
89      //32位加法的结果为{adder_cout,adder_result},则33位加法结果应该为
            {adder_cout+1'b1,adder_result}
90      //对比slt结果注释, 知道, 此时判断大小属于第二三种情况, 即源操作
            数1符号位为0, 源操作数2符号位为0
91      //结果的符号位为1, 说明小于, 即adder_cout+1'b1为2'b01, 即
            adder_cout为0
92      assign sltu_result = {31'd0, ~adder_cout};
93  //-----{加法器}end
94
95  //-----{移位器}begin
96      // 移位分三步进行,
97      // 第一步根据移位量低2位即[1:0]位做第一次移位,
98      // 第二步在第一次移位基础上根据移位量[3:2]位做第二次移位,
99      // 第三步在第二次移位基础上根据移位量[4]位做第三次移位。
100     wire [4:0] shf;
101     assign shf = alu_src1[4:0];
102     wire [1:0] shf_1_0;
103     wire [1:0] shf_3_2;
104     assign shf_1_0 = shf[1:0];
105     assign shf_3_2 = shf[3:2];
106
107      // 逻辑左移
108     wire [31:0] sll_step1;
109     wire [31:0] sll_step2;
```

```
110     assign sll_step1 = {32{shf_1_0 == 2'b00}} & alu_src2
                             // 若shf[1:0]="00",不移位
111                      | {32{shf_1_0 == 2'b01}} & {alu_src2[30:0], 1'
                           d0}      // 若shf[1:0]="01",左移1位
112                      | {32{shf_1_0 == 2'b10}} & {alu_src2[29:0], 2'
                           d0}      // 若shf[1:0]="10",左移2位
113                      | {32{shf_1_0 == 2'b11}} & {alu_src2[28:0], 3'
                           d0};     // 若shf[1:0]="11",左移3位
114     assign sll_step2 = {32{shf_3_2 == 2'b00}} & sll_step1
                           // 若shf[3:2]="00",不移位
115                      | {32{shf_3_2 == 2'b01}} & {sll_step1[27:0], 4'
                           d0}     // 若shf[3:2]="01",第一次移位结果左移
                           4位
116                      | {32{shf_3_2 == 2'b10}} & {sll_step1[23:0], 8'
                           d0}     // 若shf[3:2]="10",第一次移位结果左移
                           8位
117                      | {32{shf_3_2 == 2'b11}} & {sll_step1[19:0],
                           12'd0};  // 若shf[3:2]="11",第一次移位结果左
                           移12位
118     assign sll_result = shf[4] ? {sll_step2[15:0], 16'd0} :
        sll_step2;      // 若shf[4]="1",第二次移位结果左移16位
119
120     // 逻辑右移
121     wire [31:0] srl_step1;
122     wire [31:0] srl_step2;
123     assign srl_step1 = {32{shf_1_0 == 2'b00}} & alu_src2
                             // 若shf[1:0]="00",不移位
124                      | {32{shf_1_0 == 2'b01}} & {1'd0, alu_src2[31:1
                           ]}      // 若shf[1:0]="01",右移1位,高位补0
125                      | {32{shf_1_0 == 2'b10}} & {2'd0, alu_src2[31:2
                           ]}      // 若shf[1:0]="10",右移2位,高位补0
126                      | {32{shf_1_0 == 2'b11}} & {3'd0, alu_src2[31:3
                           ]};     // 若shf[1:0]="11",右移3位,高位补0
127     assign srl_step2 = {32{shf_3_2 == 2'b00}} & srl_step1
                           // 若shf[3:2]="00",不移位
128                      | {32{shf_3_2 == 2'b01}} & {4'd0, srl_step1[31:
                           4]}     // 若shf[3:2]="01",第一次移位结果右移
```

```
                        4位，高位补0
129                   | {32{shf_3_2 == 2'b10}} & {8'd0, srl_step1[31:
                        8]}     // 若shf[3:2]="10"，第一次移位结果右移
                        8位，高位补0
130                   | {32{shf_3_2 == 2'b11}} & {12'd0, srl_step1[31
                        :12]}; // 若shf[3:2]="11"，第一次移位结果右移
                        12位，高位补0
131   assign srl_result = shf[4] ? {16'd0, srl_step2[31:16]} :
        srl_step2;     // 若shf[4]="1"，第二次移位结果右移16位，高位补0
132
133   // 算术右移
134   wire [31:0] sra_step1;
135   wire [31:0] sra_step2;
136   assign sra_step1 = {32{shf_1_0 == 2'b00}} & alu_src2
                                        // 若shf[1:0]="00"，不移位
137                   | {32{shf_1_0 == 2'b01}} & {alu_src2[31],
                        alu_src2[31:1]}           // 若shf
                        [1:0]="01"，右移1位，高位补符号位
138                   | {32{shf_1_0 == 2'b10}} & {{2{alu_src2[31]}},
                        alu_src2[31:2]}       // 若shf[1:0]="10"，右移
                        2位，高位补符号位
139                   | {32{shf_1_0 == 2'b11}} & {{3{alu_src2[31]}},
                        alu_src2[31:3]};      // 若shf[1:0]="11"，右移
                        3位，高位补符号位
140   assign sra_step2 = {32{shf_3_2 == 2'b00}} & sra_step1
                                       // 若shf[3:2]="00"，不移位
141                   | {32{shf_3_2 == 2'b01}} & {{4{sra_step1[31]}},
                         sra_step1[31:4]}     // 若shf[3:2]="01"，第一
                        次移位结果右移4位，高位补符号位
142                   | {32{shf_3_2 == 2'b10}} & {{8{sra_step1[31]}},
                         sra_step1[31:8]}     // 若shf[3:2]="10"，第一
                        次移位结果右移8位，高位补符号位
143                   | {32{shf_3_2 == 2'b11}} & {{12{sra_step1[31]
                        }}, sra_step1[31:12]}; // 若shf[3:2]="11"，第
                        一次移位结果右移12位，高位补符号位
144   assign sra_result = shf[4] ? {{16{sra_step2[31]}}, sra_step2[31:
        16]} : sra_step2;     // 若shf[4]="1"，第二次移位结果右移16位，
```

```
        高位补符号位
//------{移位器}end

    // 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 :
                        32'd0;
endmodule
```

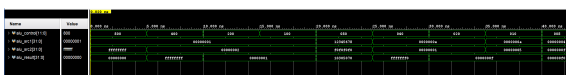
3.1.3 tb

```
module tb;

    reg   [11:0] alu_control;
    reg   [31:0] alu_src1;
    reg   [31:0] alu_src2;
    wire  [31:0] alu_result;
    alu alu_module(
        .alu_control(alu_control),
        .alu_src1   (alu_src1   ),
        .alu_src2   (alu_src2   ),
        .alu_result (alu_result )
    );

    initial begin
        //加法操作
        alu_control = 12'b1000_0000_0000;
        alu_src1 = 32'd1;
        alu_src2 = 32'hffffffff;
```

```
19
20          //减法操作
21          #5;
22          alu_control = 12'b0100_0000_0000;
23          alu_src1 = 32'd1;
24          alu_src2 = 32'd2;
25
26          //有符号比较
27          #5;
28          alu_control = 12'b0010_0000_0000;
29          alu_src1 = 32'd1;
30          alu_src2 = 32'd2;
31
32          //无符号比较
33          #5;
34          alu_control = 12'b0001_0000_0000;
35          alu_src1 = 32'd1;
36          alu_src2 = 32'd2;
37
38          //按位与
39          #5;
40          alu_control = 12'b0000_1000_0000;
41          alu_src1 = 32'h12345678;
42          alu_src2 = 32'hf0f0f0f0;
43
44          //按位或非
45          #5;
46          alu_control = 12'b0000_0100_0000;
47          alu_src1 = 32'he;
48          alu_src2 = 32'd1;
49
50          //按位或
51          #5;
52          alu_control = 12'b0000_0010_0000;
53          alu_src1 = 32'he;
54          alu_src2 = 32'd1;
55
```

```
56          //按位异或
57          #5;
58          alu_control = 12'b0000_0001_0000;
59          alu_src1 = 32'b1010;
60          alu_src2 = 32'b0101;
61
62          //逻辑左移
63          #5;
64          alu_control = 12'b0000_0000_1000;
65          alu_src1 = 32'd4;
66          alu_src2 = 32'hf;
67
68          //逻辑右移
69          #5;
70          alu_control = 12'b0000_0000_0100;
71          alu_src1 = 32'd4;
72          alu_src2 = 32'hf0;
73
74          //算术右移
75          #5;
76          alu_control = 12'b0000_0000_0010;
77          alu_src1 = 32'd4;
78          alu_src2 = 32'hf0000000;
79
80          //高位加载
81          #5;
82          alu_control = 12'b0000_0000_0001;
83          alu_src2 = 32'hbfc0;
84      end
85  endmodule
```

3.2 仿真波形



(a) 0~40ns

(b) 40~58ns

3.3 实验箱结果

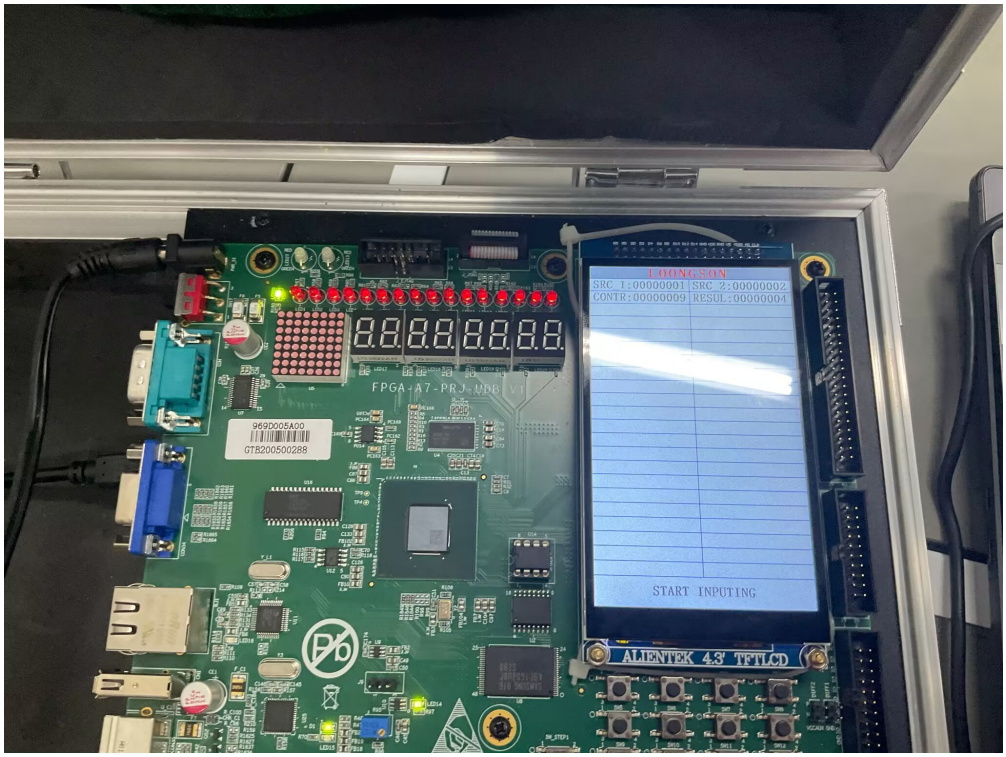


图 3.1 实验箱结果 (1«2=4)

附　　录

参考链接：https://github.com/zehua0417/ComputerOrganizationAndArchitecture_exp