浙江大学

本科实验报告

课程名称：计算机组成

姓　　名：胡亮泽

学　　院：计算机科学与技术学院

　　系：计算机科学与技术系

专　　业：计算机科学与技术

学　　号：3120102116

指导教师：姜晓红

2014 年　3　月　14　日

浙江大学实验报告

课程名称：Computer Organization 实验类型：综合

实验项目名称：Lab3： multi function ALU

学生姓名：胡亮泽 专业：计算机科学与技术

学号：3120102116

同组学生姓名：王艺 指导老师：姜晓红

实验地点：东 4-509 实验日期：2014 年 3 月 11 日

一、 实验目的和要求

1. Master the principle and design method of ALU.
2. Master the principle and design method of ALU Controller.

二、 实验内容和原理

ALU 由两部分组成，分别是 ALU 和 ALU control，其中 ALU 负责运算，而 ALU control 负责解码的功能，即将相应的输入码转换成对应的指令码传输给 ALU，从而实现对应的运算功能。

ALU control 的解码对应真值表如下：

ALUop		Func（from R instruction）						operation
ALUop1	ALUop2	F5	F4	F3	F2	F1	F0	
0	0	×	×	×	×	×	×	010:ADD
0	1	×	×	×	×	×	×	110:SUB
1	×	×	×	0	0	0	0	010:ADD
1	×	×	×	0	0	1	0	110:SUB
1	×	×	×	0	1	0	0	000:AND
1	×	×	×	0	1	0	1	001: OR
1	×	×	×	1	0	1	0	111:SLT

其中 ALUop1 和 ALUop2 由两个开关实现，F0~F3 由 4 个开关实现。

其对应代码如下：

```
module aluc(input wire[1:0] button,
          input wire[3:0] switch,
              output wire[2:0]control
    );

wire ALUop1,ALUop2;
wire result1,result2,result3,result4,result5,result6;
wire or_result1,or_result2;

assign ALUop1 = button[1];
assign ALUop2 = button[0];


and

and1(result1,~switch[3],~switch[2],~switch[1],~switch[0],ALUop1),
    and2(result2,~switch[3],~switch[2],switch[1],~switch[0],ALUop1
),
    and3(result3,~switch[3],switch[2],~switch[1],~switch[0],ALUop1
),
    and4(result4,~switch[3],switch[2],~switch[1],switch[0],ALUop1)
,
    and5(result5,switch[3],~switch[2],switch[1],~switch[0],ALUop1)
,
    and6(result6,~ALUop1,ALUop2);
or
  or1(or_result1,result4,result5),
  or2(control[1],result1,result2,result5,result6),
  or3(control[2],result6,result2,result5);
and
   and7(control[0],ALUop1,or_result1);



endmodule
```

如代码所示，本次实验采用结构化描述的方法实现 ALU control 的功能。以输出相应的指令码。具体对应关系不做赘述。

ALU 的代码如下所示：

```
module alu(input wire[15:0] op1,

          input wire[15:0] op2,

             input wire[2:0]  control,

             output wire o_zf,

             output reg[15:0] disp_code

    );
```

```
wire switch;

wire co;

wire[15:0] add_sub_result;

assign switch = (control==3'b110);



assign o_zf = (disp_code==0);

adder_16bits m0(op1, op2, switch, add_sub_result, co);


always @* begin


case(control)

3'b010: disp_code= add_sub_result;

3'b110: disp_code= add_sub_result;

3'b010: disp_code= add_sub_result;

3'b110: disp_code= add_sub_result;

3'b000: disp_code= op1 & op2;

3'b001: disp_code= op1 | op2;

3'b111: disp_code= (op1<op2?1:0);

endcase

end


endmodule
```

其中 adder_16bits m0(op1, op2, switch, add_sub_result, co) 模块用于进行加减法的运算，switch 用于表示加法或减法，在 switch 为 0 时进行加法运算，否则进行减法运算。根据指令真值表的关系，采用 assign 赋值语句：

```
assign switch = (control==3'b110);
```

进行赋值，而 control 即为 ALU control 传输的指令序号。

之后便是根据不同的指令进行不同的运算了，通过 case 语句实现。

```
3'b010: disp_code= add_sub_result;

3'b110: disp_code= add_sub_result;

3'b010: disp_code= add_sub_result;

3'b110: disp_code= add_sub_result;

3'b000: disp_code= op1 & op2;

3'b001: disp_code= op1 | op2;

3'b111: disp_code= (op1<op2?1:0);
```

另外，16 位加法器的功能代码如下：

```
module adder_16bits(A, B, Ctr, S, Co);
    parameter size=16;
   input [size:1] A;
   input [size:1] B;
   input Ctr;
   output [size:1] S;
   output Co;
    wire[size-1:1]  Ctemp;
    wire[size:1]    Bo;

    assign Bo={16{Ctr}}^B;
adder_1bit
    A1(.a(A[1]),.b(Bo[1]),.ci(Ctr),.s(S[1]),.co(Ctemp[1])),
        A2(A[2],Bo[2],Ctemp[1],S[2],Ctemp[2]),
        A3(A[3],Bo[3],Ctemp[2],S[3],Ctemp[3]),
        A4(A[4],Bo[4],Ctemp[3],S[4],Ctemp[4]),
        A5(A[5],Bo[5],Ctemp[4],S[5],Ctemp[5]),
        A6(A[6],Bo[6],Ctemp[5],S[6],Ctemp[6]),
        A7(A[7],Bo[7],Ctemp[6],S[7],Ctemp[7]),
        A8(A[8],Bo[8],Ctemp[7],S[8],Ctemp[8]),
        A9(A[9],Bo[9],Ctemp[8],S[9],Ctemp[9]),
        A10(A[10],Bo[10],Ctemp[9],S[10],Ctemp[10]),
        A11(A[11],Bo[11],Ctemp[10],S[11],Ctemp[11]),
        A12(A[12],Bo[12],Ctemp[11],S[12],Ctemp[12]),
        A13(A[13],Bo[13],Ctemp[12],S[13],Ctemp[13]),
        A14(A[14],Bo[14],Ctemp[13],S[14],Ctemp[14]),
        A15(A[15],Bo[15],Ctemp[14],S[15],Ctemp[15]),
        A16(A[16],Bo[16],Ctemp[15],S[16],Co);
endmodule

module adder_1bit(a, b, ci, s, co);
input wire a,b,ci;
    output wire s,co;
    and (c1,a,b), (c2,b,ci), (c3,a,ci);
    xor (s1,a,b), (s,s1,ci);
    or (co,c1,c2,c3);
endmodule
```

加法器原理为串行加法进位原理，这里不多做赘述。

另外，还通过 assign 赋值语句，使得在运算结果为 0 时，LED 变亮，具体方法简单，不做赘述。

三、 实验过程和数据记录

将代码下载到开发板上之后，将所有开关置于低电平，发现初始数码管显示“1122”，即 op1 的值。将最右端的开关上拨，显示“3344”，即 op2 的值。将 switch[1]开关上拨后，显示 4466，即加法（初始状态下两个按钮均未按下）运算得到的结果。按下 ALUop2 后，显示 ddde（减法）的运算结果。接着，在按下 ALUop1 的情况下，分别将四个控制开关拨到 0000,0010,0100，0101,1010 的状态，分别显示 4466（加），ddde（减）,1100（与）,3366（或）,0001（比较判断）的结果，与结果相符。

另外，在输入两个完全相同的数字时，利用减法得到 0 的运算结果后 LED 灯会变亮。

四、 实验结果

实验过程中的结果完全与预期符合。但过程中通过门级机构化描述的方法实现功能需要极大的耐心与细心，并且在发现错误的时候较难发现并纠正。通常可能出现的情况为线路的错误逻辑关系，遗漏部分逻辑关系，以及同一点高低电平出现矛盾的情况。

五、 讨论与心得

本次实验在纠错后，主要让我掌握了 ALU 的功能实现方法，以及进一步加深了结构化描述语言的实现方法。另外，为了防止错误的产生，一定要在描述之前清晰的列出逻辑关系，并且代码书写时一定要清晰，有条理，否则会对后期的调试造成致命性的难度！