同济大学计算机系

数字逻辑课程实验报告

学　　号 2350752

姓　　名 田思宇

专　　业 计算机科学与技术

授课老师 张冬冬

一、实验内容

行为级 ALU 实验，在本次实验中讲使用行为级语言描述来设计 ALU，其中重点是 ALU 的四个标志位的判断逻辑，要深入理解 ALU 的实现原理以及 ALU 的电路图。

二、硬件逻辑图

（实验步骤中要求用 logisim 画图的实验，在该部分给出 logisim 原理图，否则该部分在实验报告中不用写）

三、模块建模

（该部分要求对实验中建模的所有模块进行功能描述，并列出各模块建模的 verilog 代码）

模块功能描述：首先本模块有两个 32 位的输入，一个 32 位输出，一个 4 位控制信号，四个标志位。本模块实现的功能是根据 4 位控制信号来确定对两个输入 a，b 的具体运算，分别有 add、addu（无符号加法）、sub、subu（无符号减法）、and（按位与运算）、or（按位或运算）、xor（按位异或运算）、nor（按位非或运算）、lui、slt、sltu（无符号比较）、sra、sla、sll、srl。具体进行哪一种运算我使用了 casex 来实现。然后四个标志位根据对应的逻辑来设置，在代码中用 if else 语言设置。四个标志位逻辑设置心得见实验报告最后面。

代码：

```
module alu(
input [31:0] a,     //32  位输入，操作数 1
input [31:0] b,     //32  位输入，操作数 2
input [3:0] aluc, //4 位输入，控制  alu  的操作
output reg [31:0] r, //32  位输出，由 a、b 经过 aluc 指定的操作生成
output reg zero,
output reg carry,
//0  标志位
   //  进位标志位
output reg negative,      //  负数标志位
output reg overflow      //  溢出标志位
);
reg signed [31:0] tempb;
reg signed [31:0] tempa;
always @(*)
begin
//初始化
tempb=b;
tempa=a;
zero = 1'b0;
```

```
carry = 1'b0;
negative = 1'b0;
overflow = 1'b0;
casex(aluc)
        4'b0000: r = a + b; // Addu
        4'b0010: r = a + b; // Add
        4'b0001: r = a - b; // Subu
        4'b0011: r = a - b; // Sub
        4'b0100: r = a & b; // And
        4'b0101: r = a | b; // Or
        4'b0110: r = a ^ b; // Xor
        4'b0111: r = ~(a | b); // Nor
        4'b100x: r = {b[15:0], 16'b0}; // Lui
        4'b1011: r = (tempa < tempb) ? 32'b1 : 32'b0; // Slt
        4'b1010: r = (a < b) ? 32'b1 : 32'b0; // Sltu
        4'b1100: r = tempb >>> a; // Sra
        4'b111x: r = b << a; // Sll/Sla
        4'b1101: r = b >> a; // Srl
        default: r = 32'b0;
endcase
if((aluc==4'b1010||aluc==4'b1011)&&(a==b))//零状态位
zero=1;
if(r==32'b0&&aluc!=4'b1010&&aluc!=4'b1011)
zero=1;

if(aluc==4'b0000&&((a[31]==1&&b[31]==1)||(a[31]==1&&b[31]==0&&r[31]==0)||(a[31]==0&
&b[31]==1&&r[31]==0)))
carry=1;
else if(aluc==4'b0001&&(a<b))//只要 a<b 就肯定溢出
carry=1;
else if(aluc==4'b1010&&(a<b))
carry=1;
else if(aluc==4'b1100&&a>=1)
carry=b[0+a-1];
else if(aluc==4'b1111&&a>=1)
carry=b[31-(a-1)];
else if(aluc==4'b1110&&a>=1)
carry=b[31-(a-1)];
else if(aluc==4'b1101&&a>=1)
carry=b[0+a-1];

if(aluc==4'b0010&&(a[31]==b[31]&&a[31]!=r[31]))
overflow=1;
else if(aluc==4'b0011&&(a[31]!=b[31]&&a[31]!=r[31]))
```

```
overflow=1;

if((aluc==4'b0010||aluc==4'b0011)&&r[31]==1)
negative=1;
else if(aluc==4'b1011&&(tempa<tempb))
negative=1;
else if(r[31]==1)
negative=1;

end
    endmodule
```

四、测试模块建模

（要求列写各建模模块的 test bench 模块代码）

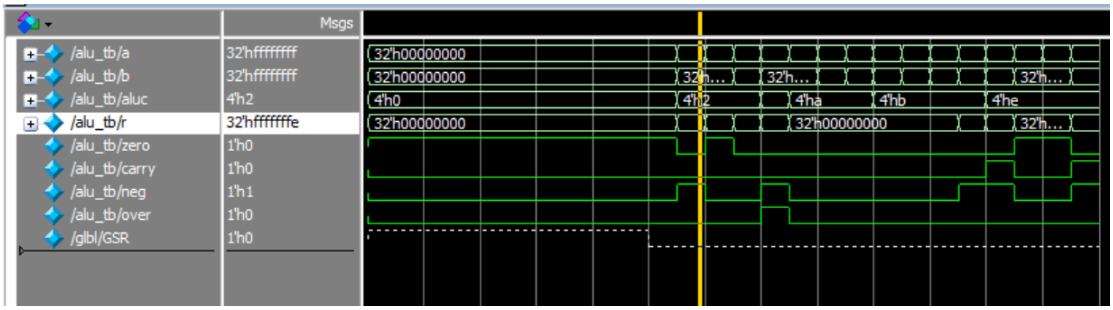
```
`timescale 1ns/1ns
module alu_tb;
reg [31:0] a,b;
reg [3:0] aluc;
wire [31:0] r;
wire zero,carry,neg,over;
initial
begin
        // Initialize Inputs
        a = 0; b = 0; aluc = 0;
        // Wait for global reset
        #100;
        // Add a few cycles to see the outputs
        #10;
        // Test case 1: Addition
        a = 32'hffffffff; b = 32'hffffffff; aluc = 4'b0010; #10;

        // Test case 2: Subtraction
        a = 32'h00000001; b = 32'hffffffff; aluc = 4'b0010; #10;

        // Test case 3: Logical AND
        a = 32'hffffffff; b = 32'h0000ffff; aluc = 4'b0010; #10;

        // Test case 4: Logical OR
        a = 32'h7fffffff; b = 32'h80000000; aluc = 4'b0011; #10;

        // Test case 5: Logical XOR
```

```
        a = 32'hffffffff; b = 32'h80000000; aluc = 4'b1010; #10;

        // Test case 6: Logical NOR
        a = 32'h0000ffff; b = 32'h00000000; aluc = 4'b1010; #10;

        // Test case 7: Logical LUI
        a = 32'hffffffff; b = 32'h0000ffff; aluc = 4'b1010; #10;

        // Test case 8: Set Less Than
        a = 32'h00000001; b = 32'hffffffff; aluc = 4'b1011; #10;

        // Test case 9: Set Less Than Unsigned
        a = 32'hffffffff; b = 32'h80000000; aluc = 4'b1011; #10;

        // Test case 10: Arithmetic Shift Right
        a = 32'h0000ffff; b = 32'h00000000; aluc = 4'b1011; #10;

        // Test case 11: Logical Shift Left
        a = 32'hffffffff; b = 32'h0000ffff; aluc = 4'b1011; #10;

        // Test case 12: Logical Shift Right
        a = 32'h00000008; b = 32'hffffffff; aluc = 4'b1110; #10;

        // Test case 13: Addition with overflow
        a = 32'h00000010; b = 32'h80000000; aluc = 4'b1110; #10;

        // Test case 14: Subtraction with overflow
        a = 32'h00000011; b = 32'h80000000; aluc = 4'b1110; #10;

        // Test case 15: Negative result
        a = 32'h0000001f; b = 32'h0000ffff; aluc = 4'b1110; #10;



        // End simulation
            $finish;
end
alu uut(a,b,aluc,r,zero,carry,neg,over);
endmodule
```

五、实验结果

（该部分可截图说明，要求 logisim 逻辑验证图、modelsim 仿真波形图、以及下板后的实验结果贴图（实验步骤中没有下板要求的实验，不需要下板贴图））

Modelsim 仿真波形图
如图是 0010 执行有符号加法，显然是负+负，但是没有溢出，结果正确



本次实验心得体会

ALU 是 CPU 中负责运算的模块，它有很多功能，然后有四个标志位，这在暑假的汇编都接触过。
其中运算功能用行为级描述很容易，比如有符号数加法和无符号数加法都是 a+b，因为计算机里面都是按位加，这个没区别。
但是在 Verilog 语言里面，a<b 的判断就有点区别了,因为如果是无符号数，a：1111 明显大于 b：0111，但是如果有符号数，a 是小于 b 的。所以想要在 Verilog 里面实现有符号数的比较，就要定义（中间）变量 just like：reg signed [31:0] a。以及想要实现>>>(算数右移，也要用 signed，否则和逻辑右移表现一样)

下面来讨论四个标志位的规则
一、首先 zero 位：对于所有运算都会影响 zero 位，对于有符号和无符号比较操作，如果 a=b，则把 zero 设置为 1（这样结合 result 的值就可以完全判断 a 和 b 之间的关系了），对于其他操作，只要 result=0，那么 zero=1。
二、然后对于 carry 位：
carry 只对无符号加减比较以及移位才会改变，有符号数的操作是不会改变的
carry 就是最高位更高的一位（虚拟的，原本是 0）变成了 1，那么 carry 就变成 1。
1、对于无符号数加法：如果最高位是都是 1、1，肯定就进，如果最高位 1、0 或 0、1，但是结果的最高位变成 0，说明进位
2、对于无符号数减法：如果最高位 1、1 但是结果最高位是 1，以及如果最高位是 0、0 但是结果最高位是 1、以及 0、1 一定会发生进位。（或者只要 a<b 就行，因为无符号数 a<b 就是单纯比大小，然后 a<b 那么减法就会借位）
3、移位的话：如果是右移 n（n>=1,不可以不移）位置，那么就是 b【0+n-1】，左移的话是 b【31-（n-1）】
三、对于 negtive 位
对于有符号比较，如果 a-b<0(也就是有符号数比较 a<b)，那么就设置为 1，其余情况如果 r[31]=1 代表结果是负数，那么就置 1。
四、对于 overflow 位
基础知识：首先所有数字都是补码这种编码方式存储，正数补码就是本身，负数

补码就是负数绝对值取反+1。由补码求真值：求反码（符号位不变其他反）+1，然后读取除符号位的值。
溢出的定义：对于无符号数来说，溢出就是十进制运算结果范围超过 0~(2^n)-1，有符号数来说，范围超过-2^(n-1)~2^(n-1)-1。例如一个八位的无符号数范围就是 0-255，对于八位有符号数来说就是-128-127
只有有符号数加减运算才会影响 overflow，如果溢出了，那么就置 1。
人眼判断只要看看符号位有没有变。
对于加法要知道只有正+正 or 负+负才会溢出。两个正数相加，数据位最高位进位而符号位没进位则溢出。两个负数相加，数据位最高位不进位，符号位进位。
对于减法要知道只有正-负 or 负-正才会溢出，也就是说两个输入符号位不一样，但是输出的符号位和第一个相反。