同济大学计算机系

数字逻辑课程综合实验报告

学　　号 2352595

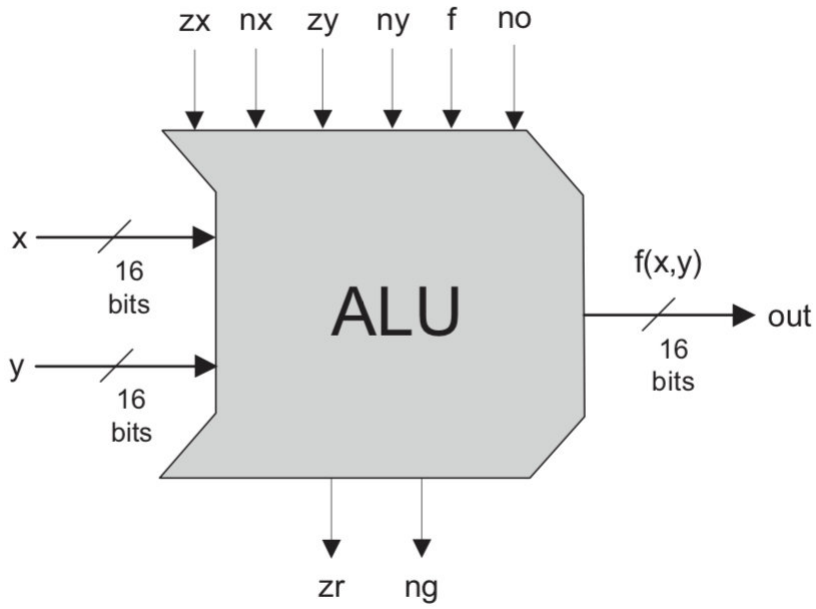
姓　　名 张嘉麟

专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

本实验旨在设计并实现一个简单的算术逻辑单元（ALU，Arithmetic Logic Unit），以掌握数字电路设计的基本原理及操作。ALU 负责执行算术运算（如加法、减法）和逻辑运算（如与、或、异或）。实验中，通过 HDL 语言描述 ALU 的功能，并进行电路设计、功能仿真和验证。通过选择控制信号确定输出结果。



These bits instruct how to preset the x input		These bits instruct how to preset the y input		This bit selects between + / And	This bit inst. how to postset out	Resulting ALU output
zx	nx	zy	ny	f	no	out=
if zx then x=0	if nx then x=!x	if zy then y=0	if ny then y=!y	if f then out=x+y else out=x&y	if no then out=!out	f(x,y)=
1	0	1	0	1	0	0
1	1	1	1	1	1	1
1	1	1	0	1	0	-1
0	0	1	1	0	0	x
1	1	0	0	0	0	y
0	0	1	1	0	1	!x
1	1	0	0	0	1	!y
0	0	1	1	1	1	-x
1	1	0	0	1	1	-y
0	1	1	1	1	1	x+1
1	1	0	1	1	1	y+1
0	0	1	1	1	0	x-1
1	1	0	0	1	0	y-1
0	0	0	0	1	0	x+y
0	1	0	0	1	1	x-y
0	0	0	1	1	1	y-x
0	0	0	0	0	0	x&y
0	1	0	1	0	1	x|y

（图片来源：从 0 到 1 构建计算机(3/12)--组合逻辑芯片：逻辑门、加法器、ALU_设计做 cpu 的逻辑芯片-CSDN 博客）

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）

略

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）

```
module alu(
    input [31:0] a,
    input [31:0] b,
    input [3:0] aluc,
    output reg [31:0] r,
    output reg zero,
    output reg carry,
    output reg negative,
    output reg overflow
    );
        always @(*) begin
        // Default values for flags and result
        zero = 0;
        carry = 0;
        negative = 0;
        overflow = 0;

        case(aluc)
            4'b0000: begin // ADDU: Unsigned addition
                r = a + b;
                carry = (a + b) < a; // Carry occurs if sum < a (overflow for unsigned)
            end
            4'b0010: begin // ADD: Signed addition
                r = $signed(a) + $signed(b);
                overflow = (($signed(a) > 0 && $signed(b) > 0 && $signed(r) < 0) ||
                            ($signed(a) < 0 && $signed(b) < 0 && $signed(r) > 0));
            end
            4'b0001: begin // SUBU: Unsigned subtraction
                r = a - b;
                carry = (a < b); // Borrow occurs if a < b (underflow for unsigned)
            end
            4'b0011: begin // SUB: Signed subtraction
                r = $signed(a) - $signed(b);
                overflow = (($signed(a) > 0 && $signed(b) < 0 && $signed(r) < 0) ||
```

```
                    ($signed(a) < 0 && $signed(b) > 0 && $signed(r) > 0));
end
4'b0100: begin // AND: Bitwise AND
    r = a & b;
end
4'b0101: begin // OR: Bitwise OR
    r = a | b;
end
4'b0110: begin // XOR: Bitwise XOR
    r = a ^ b;
end
4'b0111: begin // NOR: Bitwise NOR
    r = ~(a | b);
end
4'b1000: begin // LUI: Load Upper Immediate
    r = {b[15:0], 16'b0};
end
4'b1001: begin // LUI: Load Upper Immediate
    r = {b[15:0], 16'b0};
end
4'b1011: begin //SLT: Signed Less Than
    r = ($signed(a) < $signed(b)) ? 32'b1 : 32'b0;
    if (a == b)
        zero = 1;
    else
        zero = 0;
    if (r == 1)
        negative = 1;
    else
        negative = 0;
end
4'b1010: begin //SLTU: Unsigned Less Than
    r = (a < b) ? 32'b1 : 32'b0;
    carry = (a < b); // For SLTU, carry flag indicates a < b
    if (a == b)
        zero = 1;
    else
        zero = 0;
end
4'b1100: begin // SRA: Arithmetic Right Shift
    r = $signed(b) >>> a;
    if (a > 0)
        carry = (a < 32)?b[a - 1]:b[31];
    else
```

```
                carry = 0;
            end
            4'b1110: begin // SLL/SLA: Logical/Arithmetic Left Shift
                r = b << a;
                carry = b[31 - a]; // Carry out is the bit shifted out
            end
            4'b1111: begin // SLL/SLA: Logical/Arithmetic Left Shift
                r = b << a;
                carry = b[31 - a]; // Carry out is the bit shifted out
            end
            4'b1101: begin // SRL: Logical Right Shift
                r = b >> a;
                if (a > 0)
                    carry = b[a - 1]; // Carry out is the bit shifted out
                else
                    carry = 0;
            end
            default: begin
                r = 32'b0;
            end
        endcase

        // Set zero flag if result is zero
        if (aluc != 4'b1011 && aluc != 4'b1010)
            zero = (r == 32'b0);

        // Set negative flag for signed operations
        if (aluc != 4'b1011)
            negative = r[31];
    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 / 1ps
module alu_tb;
    // Inputs to the ALU
reg [31:0] a;
reg [31:0] b;
reg [3:0] aluc;

// Outputs from the ALU
wire [31:0] r;
```

```
wire zero;
wire carry;
wire negative;
wire overflow;

// Instantiate the ALU module
alu uut (
        .a(a),
        .b(b),
        .aluc(aluc),
        .r(r),
        .zero(zero),
        .carry(carry),
        .negative(negative),
        .overflow(overflow)
);

initial begin
        // Initialize Inputs
        a = 0;
        b = 0;
        aluc = 0;

        // Monitor the outputs
        $monitor("Time = %0t, a = %08h, b = %08h, aluc = %b, r = %08h, zero = %b, carry = %b,
negative = %b, overflow = %b",
                        $time, a, b, aluc, r, zero, carry, negative, overflow);

        // Test unsigned addition (ADDU)
        a = 32'h7FFFFFFF; b = 32'h00000001; aluc = 4'b0000;
        #10;

        // Test signed addition (ADD)
        a = 32'h7FFFFFFF; b = 32'h00000001; aluc = 4'b0010;
        #10;

        // Test unsigned subtraction (SUBU)
        a = 32'h00000002; b = 32'h00000001; aluc = 4'b0001;
        #10;

        // Test signed subtraction (SUB)
        a = 32'h80000000; b = 32'h00000001; aluc = 4'b0011;
        #10;
```

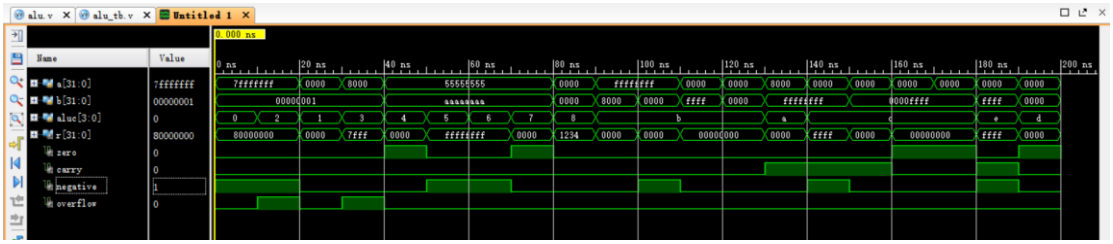
```
// Test bitwise AND (AND)
a = 32'h55555555; b = 32'hAAAAAAAA; aluc = 4'b0100;
#10;

// Test bitwise OR (OR)
a = 32'h55555555; b = 32'hAAAAAAAA; aluc = 4'b0101;
#10;

// Test bitwise XOR (XOR)
a = 32'h55555555; b = 32'hAAAAAAAA; aluc = 4'b0110;
#10;

// Test bitwise NOR (NOR)
a = 32'h55555555; b = 32'hAAAAAAAA; aluc = 4'b0111;
#10;

// Test Load Upper Immediate (LUI)
a = 0; b = 32'h00001234; aluc = 4'b1000; // Note: LUI ignores a, using b[15:0]
#10;

// Test Signed Less Than (SLT)
a = 32'hffffffff; b = 32'h80000000; aluc = 4'b1011;
#10;
a = 32'hffffffff; b = 32'h0000ffff;
#10;
a = 32'h00000001; b = 32'hffffffff;
#10;
a = 32'h0000ffff; b = 32'h00000000;
#10;
// Test Unsigned Less Than (SLTU)
a = 32'h80000000; b = 32'hFFFFFFFF; aluc = 4'b1010;
#10;

// Test Arithmetic Right Shift (SRA)
a = 8; b = 32'hffffffff; aluc = 4'b1100;
#10;
a = 1; b = 32'h0000ffff;
#10;
a = 17; b = 32'h0000ffff;
#10;
a = 31; b = 32'h0000ffff;
#10;

// Test Logical/Arithmetic Left Shift (SLL/SLA)
```

```
        a = 8; b = 32'hffffffff; aluc = 4'b1110;
        #10;

        // Test Logical Right Shift (SRL)
        a = 0; b = 32'h00000000; aluc = 4'b1101;
        #10;

        // End the simulation
        $stop;
end
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



0-40ns 为算术运算：

以 0-10ns 为例：

输入信号：

a = 0x7FFFFFFF（32 位最大正数）。

b = 0x00000001。

aluc = 4'b0000（也就是无符号加法 ADDU）。

计算结果：

r = a + b = 0x80000000（从无符号数角度是有效的结果，但符号位为 1）。

根据设计代码：

negative = r[31]（结果最高位决定符号），此时为 1

overflow 不会在无符号加法运算操作中发挥作用

40-80ns 为逻辑运算：

输入信号：
a = 0x55555555（二进制为交替的 0101）。
b = 0xAAAAAAAA（二进制为交替的 1010）。
运算逻辑：
按位执行 AND 操作：r = a & b。
输出结果：
r = 0x00000000（全 0）。
zero = 1（结果为 0）。
carry = 0（逻辑运算无进位）。
negative = 0（结果无符号）。
overflow = 0（逻辑运算无溢出）。

80-90ns 为加载高位立即数
（LUI，aluc = 4'b1000）
输入：
a = 0x00000000（在 LUI 操作中，a 通常忽略）。
b = 0x00001234（低 16 位作为加载的值）。
aluc = 4'b1000（LUI 操作代码）。
实现逻辑：
LUI 的操作是将 b 的低 16 位移至结果 r 的高 16 位，低 16 位补 0。
运算逻辑：r = {b[15:0], 16'b0}。
结果：0000 1234 0000 0000
输出：
r = 0x12340000（高 16 位为 b[15:0]，低 16 位补 0）。
zero = 0（结果非零）。
carry = 0（LUI 操作无进位）。
negative = 0（最高位为 0，结果非负）。
overflow = 0（LUI 操作无溢出）。

90-140ns 为比较运算：
以 90-100ns 为例：

aluc = 4'b1011（有符号比较，SLT）

输入信号：

a = 0xFFFFFFFF（有符号解释为 -1）。

b = 0x80000000（有符号解释为 -2147483648）。

运算逻辑：

比较 a < b（有符号数）。

-1 并不小于 -2147483648，因此 r = 0x00000000

输出结果：

r = 0x00000000（比较结果为假）。

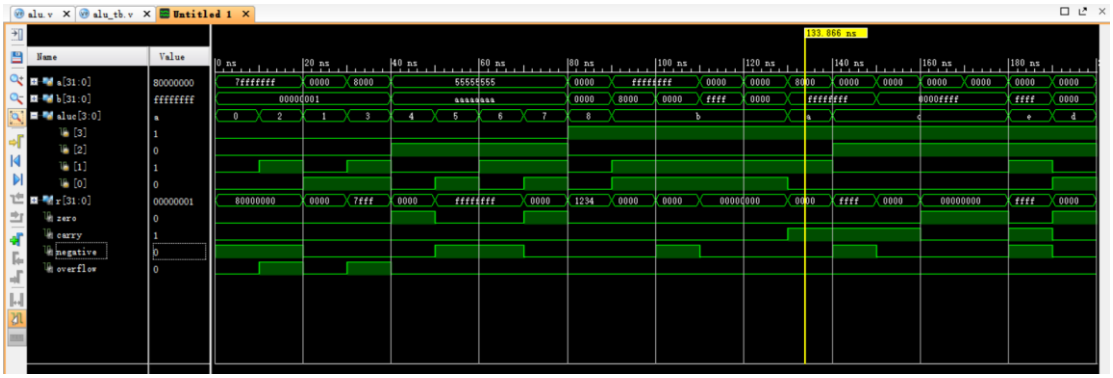
zero = 0（a != b）。

negative = 0（结果非负）。

overflow = 0（比较运算无溢出）。

140-200ns 为位移运算：

以 140-150ns 为例：



aluc = 4'b1100（算术右移，SRA）

输入信号：

a = 8（右移 8 位）。

b = 0xFFFFFFFF（有符号数 -1）。

运算逻辑：

算术右移将符号位填充到左侧空位：

输出结果：

r = 0xFFFFFFFF（保持 -1）。

carry = b[7] = 1（被移出的第 8 位是 1）。

zero = 0（结果非零）。

negative = 1（结果为负）。

overflow = 0（移位操作无溢出）。