同济大学计算机系

数字逻辑课程实验报告

学　　号 2154312

姓　　名 郑博远

专　　业 计算机科学与技术

授课老师 郭玉臣

一、实验内容

（1）ALU 实验：ALU 是负责运算的电路。ALU 必须实现以下几个运算：加（ADD）、减（SUB）、与（AND）、或（OR）、异或（XOR）、置高位立即数（LUI）、逻辑左移与算数左移（SLL）、逻辑右移（SRL）以及算数右移（SRA）、SLT、SLTU 等操作。输出 32 位计算结果、carry(借位进位标志位)、zero(零标志位)、negative(负数标志位)和 overflow(溢出标志位)。本实验实现 ALU 的基本思想是：在操作数输入之后将所有可能的结果都计算出来，通过操作符 aluc 的输入来判别需要执行的操作来选择需要的结果进行输出。

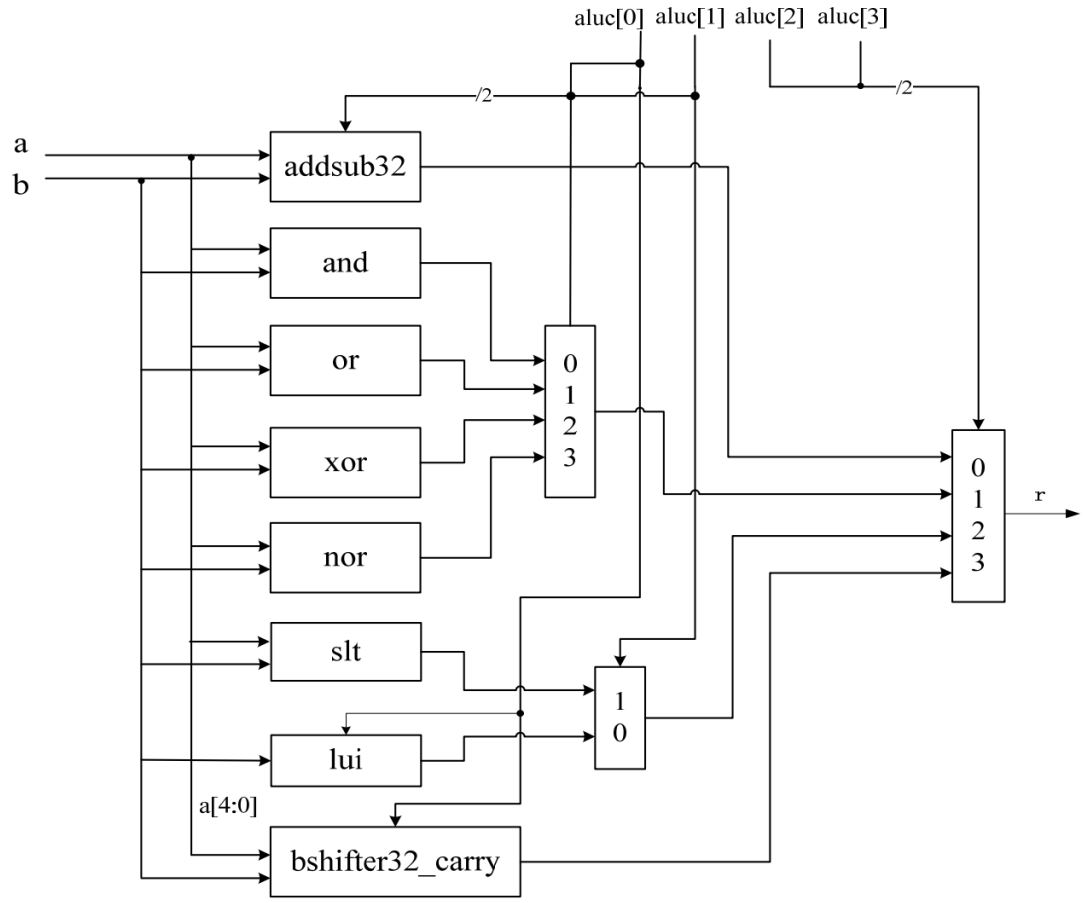


图 1　ALU 的原理图

二、模块建模

（1）ALU 实验：

ALU 实现的有符号数与无符号数的加法、减法、与、或、异或、或非、比较、移位等运算分别对应的 aluc 输入如下：

	aluc[3]	aluc[2]	aluc[1]	aluc[0]
Addu r=a+b 无符号	0	0	0	0
Add r=a+b 有符号	0	0	1	0
Subu r=a-b 无符号	0	0	0	1
Sub r=a-b 有符号	0	0	1	1
And r=a & b	0	1	0	0
Or r=a | b	0	1	0	1
Xor r=a ^ b	0	1	1	0
Nor r=~（a | b）	0	1	1	1
Lui r={b[15:0],16'b0}	1	0	0	X
Slt r=(a<b)?1:0 有符号	1	0	1	1
Sltu r=(a<b)?1:0 无符号	1	0	1	0
Sra r=b>>>a	1	1	0	0
Sll/Slr r=b<<a	1	1	1	X
Srl r=b>>a	1	1	0	1

```
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
    );

    reg [32:0] ae;
    reg [32:0] be;
    reg [32:0] ce;

    reg signed [31:0] sa;
```

```
    reg signed [31:0] sb;

    always@ (*)
    begin
    ae = {1'b0, a};
    be = {1'b0, b};
        casex(aluc)
            4'b0000:    //unsigned plus
            begin
                r <= a + b;
                zero = (r == 0) ? 1 : 0;
                ce = ae + be;
                carry <=  ce[32];
                negative <= r[31];
            end
            4'b0010:    //signed plus
            begin
                r <= a + b;
                zero = (r == 0) ? 1 : 0;
                negative <= r[31];
                if( a[31]== 1 && b[31]==1 && r[31] == 0)
                    overflow <= 1'b1;
                else if ( a[31]== 0 && b[31]==0 && r[31]
== 1)
                    overflow <= 1'b1;
                else
                    overflow <= 1'b0;
            end
            4'b0001:    //unsigned minus
            begin
                r <= a - b;
                zero = (r == 0) ? 1 : 0;
                ce = ae - be;
                carry <= ce[32];
                negative <= r[31];
            end
            4'b0011:    //signed minus
            begin
                r <= a - b;
                zero = (r == 0) ? 1 : 0;
                ce = ae - be;
                negative <= r[31];
                if( a[31]== 0 && b[31] == 1 && r[31] ==
```

```
1)
                  overflow <= 1'b1;
              else if ( a == 1 && b == 0 && r[31] == 0)
                  overflow <= 1'b1;
              else
                  overflow <= 1'b0;
          end
          4'b0100:    // AND
          begin
              r = a & b;
              zero = (r == 0) ? 1 : 0;
              negative <= r[31];
          end
          4'b0101:    // OR
          begin
              r = a | b;
              zero = (r == 0) ? 1 : 0;
              negative <= r[31];
          end
          4'b0110:    // XOR
          begin
              r = a ^ b;
              zero = (r == 0) ? 1 : 0;
              negative <= r[31];
          end
          4'b0111:    // NOR
          begin
              r = ~(a | b);
              zero = (r == 0) ? 1 : 0;
              negative <= r[31];
          end
          4'b100x:    // immediate number -> high
position
          begin
              r = {b[15:0], 16'b0};
              zero = (r == 0) ? 1 : 0;
              negative <= r[31];
          end
          4'b1011:    // signed compare
          begin
              zero = (a - b == 0) ? 1 : 0;
              sa = a;
              sb = b;
```

```
                negative = (sa < sb) ? 1 : 0;
            end
            4'b1010:    // unsigned compare
            begin
                zero = (a - b == 0) ? 1 : 0;
                carry = (a < b) ? 1 : 0;
            end
            4'b1100:    // >>>
            begin
                sb = b;
                r = sb >>> a;
                zero = (r == 0) ? 1 : 0;
                carry <= b[a - 1];
                negative <= r[31];
            end
            4'b111x:    // << / <<<
            begin
                r = b << a;
                zero = (r == 0) ? 1 : 0;
                carry <= b[32 - a];
                negative <= r[31];
            end
            4'b1101:    // >>
            begin
                r = b >> a;
                zero = (r == 0) ? 1 : 0;
                carry <= b[a - 1];
                negative <= r[31];
            end
        endcase
    end

endmodule
```

三、测试模块建模

（1）ALU 实验：

```
`timescale 1ns / 1ps
```

```
module alu_tb;
    reg [31:0] a;
    reg [31:0] b;
    reg [3:0] aluc;
    wire [31:0] r;
    wire zero, carry, negative, overflow;

    initial
    begin
    //unsigned plus
        aluc = 4'b0000;
            // carry
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
            // negative
        #5;
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b10000000_00111000_00001000_00000010;
        #5;
            // carry
        a = 32'b11111111_11000000_00001110_01100000;
        b = 32'b11111111_00111001_00001000_00011110;
        #5;

    //signed plus
        aluc = 4'b0010;
            // no overflow
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
            // negative
        #5;
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b10000000_00111000_00001000_00000010;
        #5;
            // no overflow
        a = 32'b11111111_11000000_00001110_01100000;
        b = 32'b11111111_00111001_00001000_00011110;
        #5;
            // overflow
        a = 32'b01111111_11000000_00001110_01100000;
        b = 32'b01111111_00111001_00001000_00011110;
        #5;
            // no overflow
```

```
        a = 32'b10110011_11000000_00001110_01100000;
        b = 32'b10001001_00111001_00001000_00011110;
        #5;

    //unsigned minus
        aluc = 4'b0001;
            // carry
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
            // zero
        #5;
        a = 32'b10000000_00111000_00001000_00000010;
        b = 32'b10000000_00111000_00001000_00000010;
        #5;
            // no carry
        a = 32'b11111111_11000000_00001110_01100000;
        b = 32'b11111111_00111001_00001000_00011110;
        #5;

    //signed minus
        aluc = 4'b0011;
            // no overflow
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
            // zero
        #5;
        a = 32'b10000000_00111000_00001000_00000010;
        b = 32'b10000000_00111000_00001000_00000010;
        #5;
            // no overflow
        a = 32'b11111111_11000000_00001110_01100000;
        b = 32'b11111111_00111001_00001000_00011110;
        #5;
            // overflow
        a = 32'b01111111_11000000_00001110_01100000;
        b = 32'b11111111_00111001_00001000_00011110;
        #5;


    //AND
        aluc = 4'b0100;
            // zero
        a = 32'b00011100_00000000_00001110_00000010;
```

```
        b = 32'b11100011_11111111_11110001_11111101;
        #5;

    //OR
        aluc = 4'b0101;
        #5;

    //XOR
        aluc = 4'b0110;
        #5;

    //NOR
        aluc = 4'b0111;
        #5;

    //LUI
        aluc = 4'b1001;
        #5;

    //signed compare
        aluc = 4'b1011;
        //no negative
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
        //negative
        #5;
        a = 32'b00000000_00111000_00001000_00000010;
        b =
32'b00100000_00111000_00001000_00000010;
        #5;

    //unsigned compare
        aluc = 4'b1010;
        //carry
        a = 32'b00011100_00000000_00001110_00000010;
        b = 32'b11111111_11111111_11111111_11111111;
        //no carry
        #5;
        a = 32'b11110000_00111000_00001000_00000010;
        b =
32'b11100000_00111000_00001000_00000010;
        //zero
        #5;
```

```
        a = 32'b11110000_00111000_00001000_00000010;
        b =
32'b11110000_00111000_00001000_00000010;

    // >>>
        aluc = 4'b1100;
        //carry
        a = 32'b00000000_00000000_00000000_00001000;
        b = 32'b11110000_00000000_00000000_10000000;
        #5;

    // >>
        aluc = 4'b1101;
        //no carry
        a = 32'b00000000_00000000_00000000_00001000;
        b = 32'b11110000_00000000_10000000_00000000;
        #5;

    // << / <<<
        aluc = 4'b1111;
        //no carry
        a = 32'b00000000_00000000_00000000_00001000;
        b = 32'b11111111_00001111_00000000_00000000;
        #5;

    end

    alu alu_inst(a, b, aluc, r, zero, carry, negative,
overflow);

endmodule
```

四、实验结果

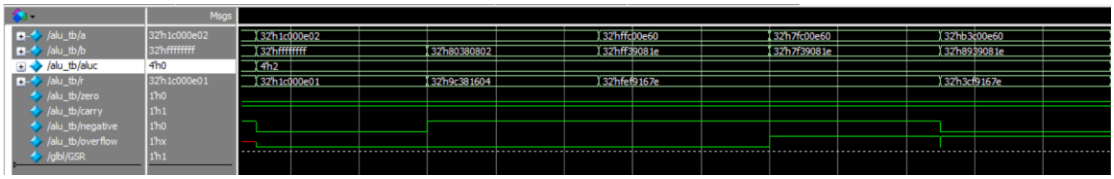
（1）ALU 实验：

a) modelsim 仿真波形图

首先测试无符号数加法：

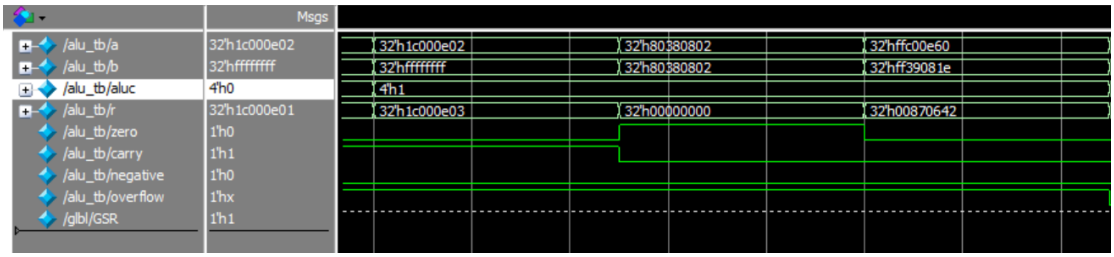
观察到，当两数相加产生进位时，carry 置 1；当最高位为 1 时，negative 置 1。

测试有符号数加法：



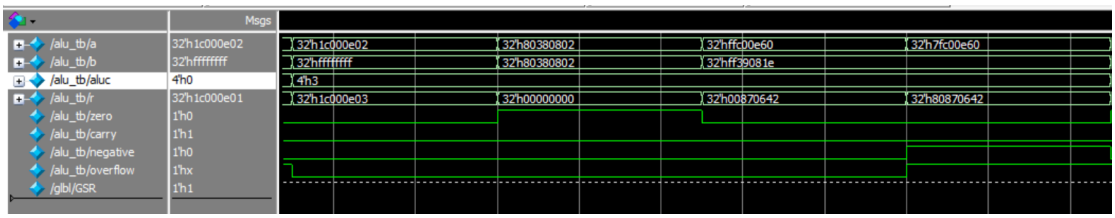
当最高位为 1 时，negative 置 1；overflow 是否置 1 不取决于是否进位，而取决于是否产生溢出（如 0xff39081e 与 0xffc00e60 相加不产生溢出）。其他标志位不发生变化。

测试无符号数减法：



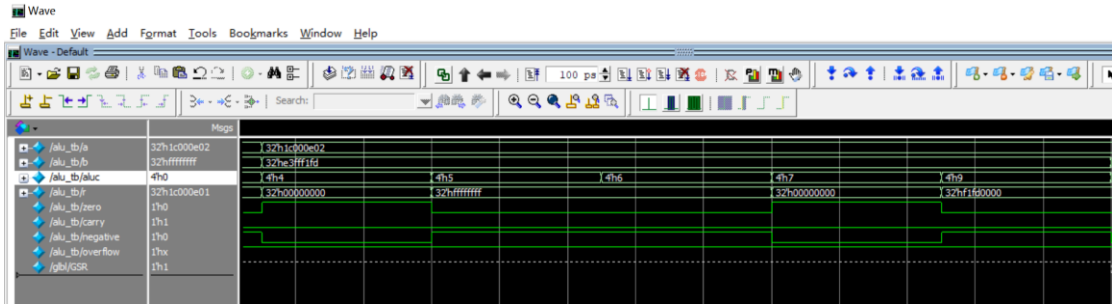
观察到，当两数相减产生借位时，carry 置 1；当最高位为 1 时，negative 置 1。

测试有符号数减法：



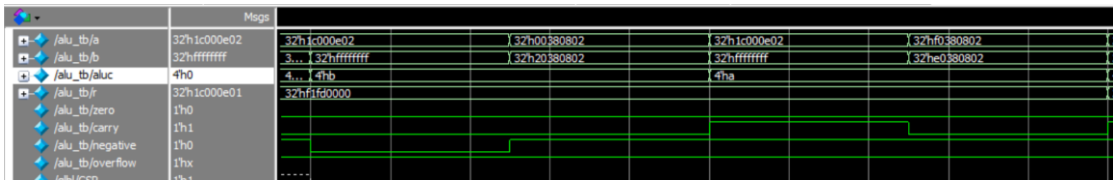
当最高位为 1 时，negative 置 1；overflow 是否置 1 不取决于是否借位，而取决于是否产生溢出（如 0xf1c000e02 与 0xffffffff 相减不产生溢出）。其他标志位不发生变化。

测试或、与、异或、或非、高位置立即数：



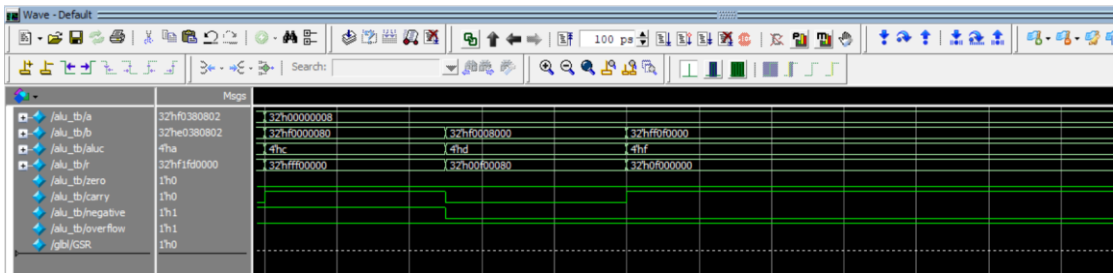
r 中结果对应 a 和 b 做对应运算所得到的结果。若结果为 0 则 zero 为 1；若最高位为 1，则 negative 为 1。

测试有符号、无符号比较：



均能得出正确的结果。若两数相等，则 zero 置 1。

测试左移、右移。



当算术右移时，高位补符号位；当逻辑右移时，高位补 0；当左移时，低位补 0。carry 位为最后移出的位值。