实验二：ALU

设计思路

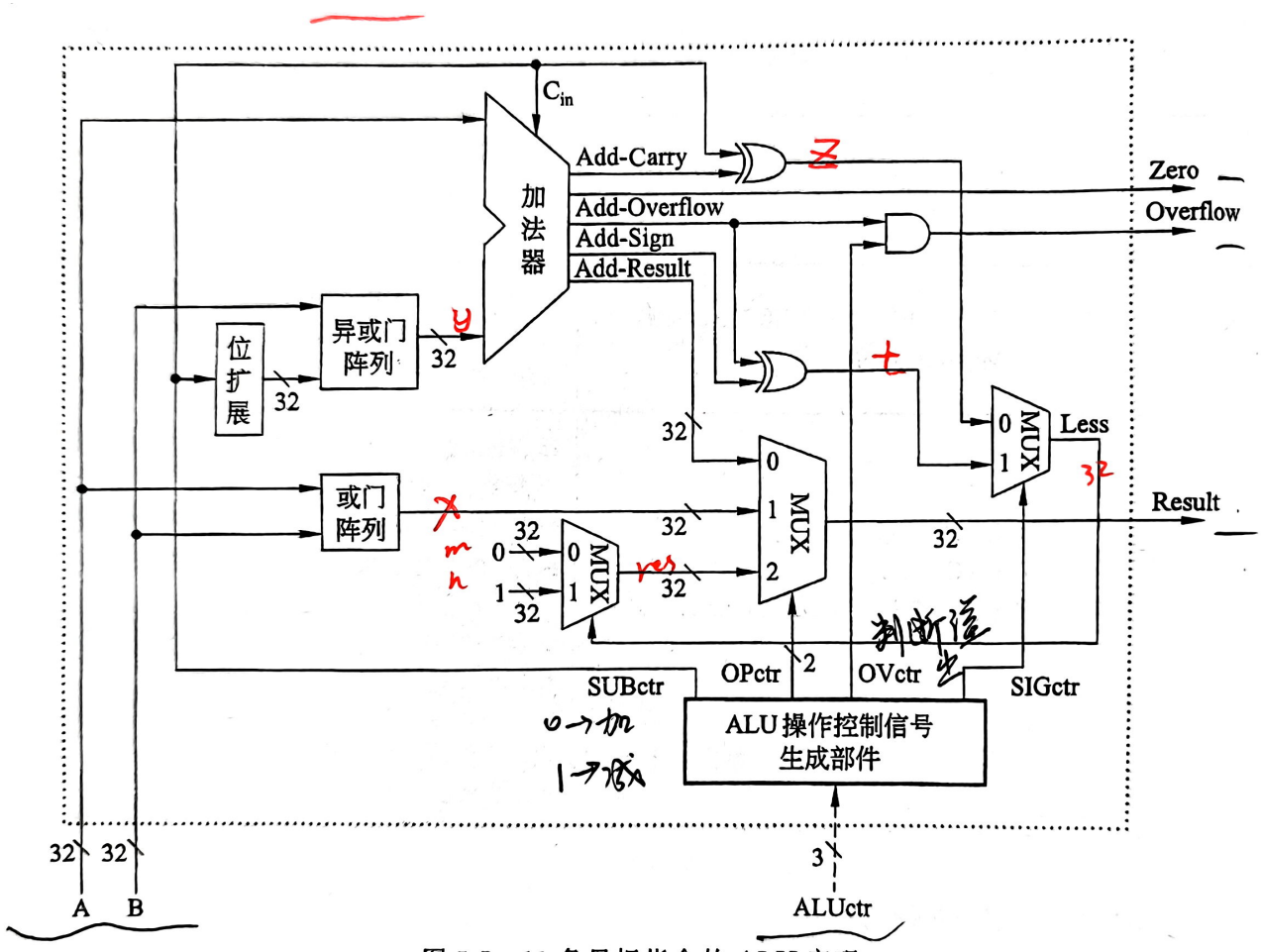


图 5.7　11 条目标指令的 ALU 实现

ALU模块：

1. 输入声明：

- 输入 A 和 B 表示两个32位的操作数。
- 输入 ALUctr[2:0] 表示控制信号，可以根据其编码进行相应的操作。其输出四个控制信号：1.SUBctr用于控制ALU执行加法还是减法运算，SUBctr=1时做减法，反之做加法。2.OPctr用来控制选择哪种运算的结果作为Result输出。3.OVctr用于控制是否进行溢出判断，当OVctr=1时，进行溢出判断，此时若发生溢出，Overflow=1，若OVctr=0,此时若发生溢出，Overflow也不为1。4.SIGctr=1时，ALU执行“带符号整数比较小于置1”，SIGctr=0时，ALU执行“无符号数比较小于置1”。

2. 输出声明：

- 输出 Overflow 表示溢出标志。
- 输出 Zero 表示零标志。

- 输出 Result 表示ALU的计算结果。

3. **中间信号声明：**
   - 子模块输入输出的中间信号，如 SUBctr、OVctr、SIGctr、Add_Carry、Add_Overflow、Add_Sign、z、t、Cin、x、y、m、n。
4. **信号赋值：**
   - 根据 ALUctr 的不同位，计算出 SUBctr、OVctr、SIGctr、OPctr，并赋值给相应的中间信号。
5. **MUX2to1模块调用：**
   - 调用 MUX2to1 模块，根据 Less 选择 Add_Carry 或 Add_Overflow，并将结果赋值给中间信号 res。
6. **MUX3to1模块调用：**
   - 调用 MUX3to1 模块，根据 OPctr 选择输入 x、y 或 res，并将结果赋值给 Result。
7. **Adder 模块调用：**
   - 调用 Adder 模块，计算加法的结果和相关标志，并将结果赋值给中间信号 Add_Carry、Add_Overflow、Add_Sign 和 Add_Result。

MUX2to1模块：

1. **输入声明：**
   - 输入 x 和 y 表示两个32位的操作数。
   - 输入 ctr 表示选择信号。
2. **输出声明：**
   - 输出 res 表示选择后的结果。
3. **逻辑：**
   - 根据 ctr 的值选择 x 或 y 作为输出 res。

MUX3to1模块：

1. **输入声明：**
   - 输入 a、b、c表示三个32位的操作数。
   - 输入 ctr 表示选择信号。
2. **输出声明：**
   - 输出 res 表示选择后的结果。
3. **逻辑：**
   - 使用 `case` 语句根据 ctr 的值选择输入 a、b 或 c，并将结果赋值给 res。

Adder 模块：

1. **输入声明：**
   - 输入 A 和 x 表示两个32位的加法操作数。
   - 输入 Cin 表示加法的进位。
2. **输出声明：**
   - 输出 Add_Carry 表示进位标志。
   - 输出 Add_Overflow 表示溢出标志。
   - 输出 Add_Sign 表示符号标志。
   - 输出 Add_Result 表示计算结果。
   - 输出 Zero 表示零标志。
3. **逻辑：**
   - 使用 `always` 块计算加法，同时计算进位、溢出、符号和零标志。

这样，整个ALU的设计思路涵盖了模块的层次结构，输入输出的定义，以及中间信号的传递和计算。

代码展示

```
// ALU Model
module ALU(input[31: 0] A,
           input[31: 0] B,      //   两个输入的32位操作数
           input[2: 0] ALUctr,  //   控制信号编码
           output zero,         //   零标志
           output overflow,     //   溢出标志
           output[31: 0] Result);   //  32位输出操作数

    // 加法器
    wire Cin;    //进位
    //加法器的输出结果（是中间结果）
    wire Add_Carry, Add_Overflow, Add_Sign;
    wire[31: 0] Add_Result;


    // 控制信号
    wire SUBctr, OVctr, SIGctr;
    // SUBctr = 1做减法运算，  为0 做加法运算
    // OVCtr 控制是否需要溢出判断，OVCtr = 1时要进行溢出判断
    // 如果溢出，则overflow置为1；否则就算溢出overflow也不为1
    // SIGctr = 1执行带符号整数比较小于置1，=0无符号比较小于置为1

    wire[1: 0] OPctr;
```

```
    // OPctr = 1 控制选择三种运算作为运算结果（加减、按位或、小于置1），所以是两位

    //中间结果
    wire[31: 0] m, n, r, x, y, Less;
    wire z, t;


    // 根据表格5.3（137页）赋值
    assign SUBctr = ALUctr[2];
    assign OVctr = !ALUctr[1] & ALUctr[0];
    assign SIGctr = ALUctr[0];
    assign OPctr[0] = ALUctr[2] & ALUctr[1];
    assign OPctr[1] = !ALUctr[2] & ALUctr[1] & !ALUctr[0];


    // 0 和 1位扩展
    assign m = {32{1'b0}};
    assign n = {32{1'b1}};

    // 中间结果赋值
    assign y = A | B;
    assign x = {32{SUBctr}} ^ B;
    assign z = Cin ^ Add_Carry;
    assign t = Add_Overflow ^ Add_Sign;
    assign Cin = SUBctr;


    //输出结果
    assign overflow = Add_Overflow & OVctr;

    // 加法器输入和输出
    Adder adder(Cin, A, x, Add_Carry, zero, Add_Overflow, Add_Sign,
Add_Result);


    // 二选一选择器输入输出
    MUX2to1 m1(z, t, SIGctr, Less);
    MUX2to1 m2(m, n, Less, r);

    //三选一选择器输入和输出
    MUX3to1 m3(Add_Result, r, y, OPctr, Result);
endmodule
```

```
module MUX2to1(input [31:0] x,
               input [31:0] y,
               input ctr,
```

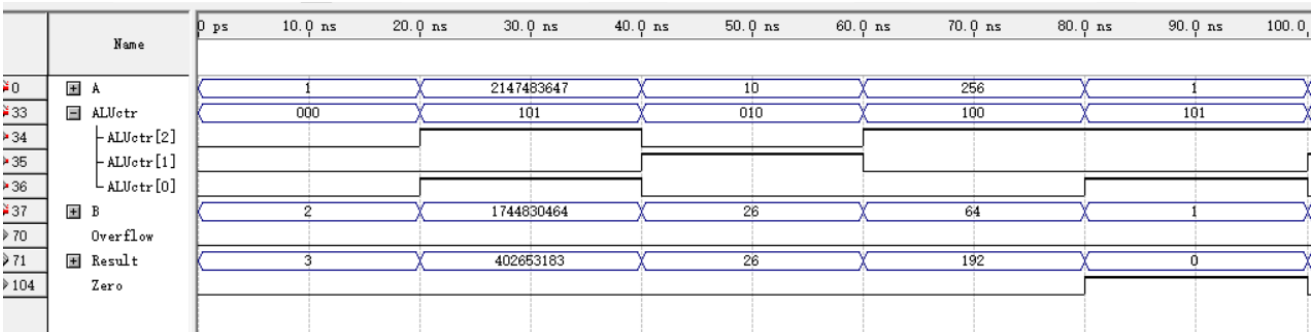
```
                output reg [31:0] res
                );

    always@(ctr)
    begin
        case(ctr)
            1'b0:res=x;
            1'b1:res=y;
        endcase
    end

endmodule
```

```
module MUX3to1(input [31:0] a,
                input [31:0] b,
                input [31:0] c,
                input [1:0] ctr,
                output reg [31:0] res
                );

    always@(ctr)
    begin
        if(ctr == 2'b00) res=a;
        else if(ctr == 2'b01) res=b;
        else if(ctr == 2'b10) res=c;
    end

endmodule
```

```
module Adder(Cin, A, x, Add_Carry, Zero, Add_Overflow, Add_Sign, Add_Result);
    input[31: 0] A, x;
    input Cin;
    output reg[31: 0] Add_Result;
    output reg Add_Carry;    // 进位标志

    output Add_Overflow, Add_Sign, Zero;

    always @(A or x or Cin)
    begin
        {Add_Carry, Add_Result} = A + x + Cin;
    end

    //全0为1，把所有位或起来再取反
    assign Zero = ~|Add_Result;
```

```
    //符号位是最高位
    assign Add_Sign = Add_Result[31];

    //溢出标志
    assign Add_Overflow = Add_Carry ^ Add_Sign ^ x[31] ^ A[31];
endmodule
```

波形分析



如上图所示，

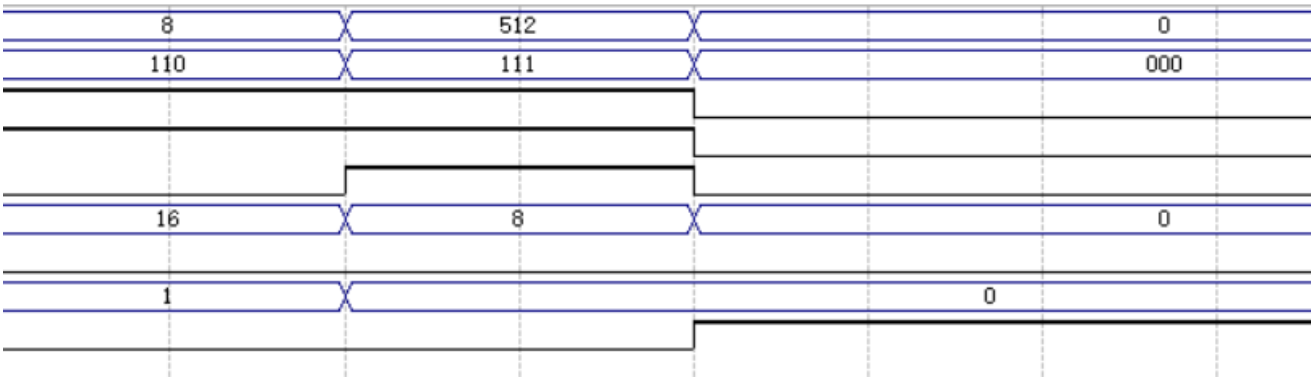
当ALUctr=000时，表示addu（加法，不判溢出），此时A=1，B=3，得到Result=3，Zero=0。

当ALUctr=101时，表示sub（减法，判溢出），此时A=2147483647，B=1744830464，得到Result=402653183，Zero=0，Overflow=0。

当ALUctr为010时，表示or（按位或，不判溢出），此时A=10，B=26，得到按位或的Result=26，Zero=0。

当ALUctr为100时，表示sub（减，判溢出），此时A=256，B=64，得到Result=194，Zero=0，Overflow=0。

当ALUctr=101时，表示sub（减法，判溢出），此时A=1，B=1，得到Result=0，Zero=1（结果为0），Overflow=0。



当ALUctr=110时，表示sltu（减，不判溢出，无符号数比较大小），此时A=8，B=16，得到Result=1(A小于B)，Zero=0,Overflow=0。

当ALUctr=111时，表示slt（减，不判溢出，带符号数比较大小），此时A=512，B=8，得到Result=0(A大于B)，Zero=0,Overflow=0。