

# 实验一 运算器与排序

**zjx@ustc.edu.cn**

**2020.4.22**

---

2020-4-22

2020春\_计算机组成原理实验\_CS-USTC

1

实验目的

熟练Vivado和N4的设计实现流程

模块化、层次化、参数化设计方法

组合逻辑电路和寄存器的描述方法

## 实验目标

- 掌握算术逻辑单元（ALU）的功能，加/减运算时溢出、进位/借位、零标志的形成及其应用
- 掌握数据通路和控制器的设计和描述方法

2020-4-22

2020春\_计算机组成原理实验\_CS-USTC

2

1. 设计一ALU：根据功能选择s，对a和b进行算术（加、减）或者逻辑（与、或、非、异或）运算，产生运算结果y和相应标志f（进位/借位、溢出、零标志）。对于算术运算，影响进位/借位、溢出、零标志；对于逻辑运算，仅零标志有效。

f: 标志位，包括进位/借位(CF)，溢出位(OF)，零标志(ZF)

为了便于下载测试，可以选取 $n = 6$ ， $m = k = 3$

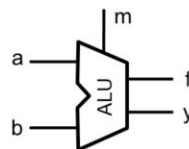
2. ALU应用设计：利用上述ALU模块和适当逻辑电路，分别实现如下功能

- (1) 比较两无符号数或者有符号数的大小关系
- (2) 求多个数的累加和
- (3) 求补码代表的实际值
- (4) 求给定两个初始整数的配波拉契数列
- (5) .....

# 实验内容

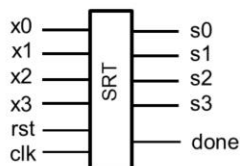
## 1. 算术逻辑单元 (ALU)

- m: 操作类型, 加、减、与、或、异或等运算
- a, b: 操作数, 对于减运算, a是被减数
- y: 运算结果, 和、差 .....
- f: 标志, 进位/借位、溢出、零标志



## 2. 排序 (Sort)

- x0 ~ x3: 4个输入数据
- s0 ~ s3: 递增排序后的结果
- done: 排序结束标志
- clk, rst: 时钟, 复位信号



2020-4-22

2020春\_计算机组成原理实验\_CS-USTC

3

1. 设计一ALU: 根据功能选择s, 对a和b进行算术(加、减)或者逻辑(与、或、非、异或)运算, 产生运算结果y和相应标志f(进位/借位、溢出、零标志)。对于算术运算, 影响进位/借位、溢出、零标志; 对于逻辑运算, 仅零标志有效。

f: 标志位, 包括进位/借位(CF), 溢出位(OF), 零标志(ZF)

为了便于下载测试, 可以选取  $n = 6$ ,  $m = k = 3$

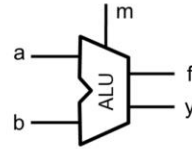
2. ALU应用设计: 利用上述ALU模块和适当逻辑电路, 分别实现如下功能

- (1) 比较两无符号数或者有符号数的大小关系
- (2) 求多个数的累加和
- (3) 求补码代表的实际值
- (4) 求给定两个初始整数的配波拉契数列
- (5) .....

# ALU


```

module alu
    #(parameter WIDTH = 32) //数据宽度
    (output [WIDTH-1:0] y, //运算结果
    output zf, //零标志
    output cf, //进位/借位标志
    output of, //溢出标志
    input [WIDTH-1] a, b, //两操作数
    input [2:0] m //操作类型
    );
    .....
endmodule
    
```



ALU 模块功能表

m	y	cf	of	zf
000	$a + b$	*	*	*
001	$a - b$	*	*	*
010	$a \& b$	x	x	*
011	$a   b$	x	x	*
100	$a \wedge b$	x	x	*
其他	x	x	x	x



CF Carry Flag: Set on high-order bit carry or borrow; cleared otherwise. This flag is used by instructions that add or subtract multi-byte numbers. Rotate instructions can also isolate a bit in memory or a register by placing it in the Carry Flag.

ZF Zero Flag: Set if result is zero; cleared otherwise.

SF Sign Flag: Set equal to high-order bit of result (0 if positive, 1 if negative).

OF Overflow Flag: Set if the signed result cannot be expressed within the number of bits in the destination operand; cleared otherwise. A significant digit has been lost because the size of the result exceeded the capacity of its destination location.

## 示例：ALU标志设置

- 已知：[X] = 1000 0111B, [Y] = 0111 1001B  
求[X±Y]后的状态标志？

[X+Y]:

$$\begin{array}{r} \text{OF}=0 \\ \text{CF}=1 \quad \begin{array}{r} \overline{11} \\ 1000 \ 0111 \\ + \ 0111 \ 1001 \\ \hline \overline{0000} \ 0000 \end{array} \\ \text{SF}=0 \quad \text{ZF}=1 \end{array}$$

[X-Y]:

$$\begin{array}{r} \text{OF}=1 \\ \text{CF}=0 \quad \begin{array}{r} \overline{10} \\ 1000 \ 0111 \\ + \ 1000 \ 0111 \\ \hline \overline{0000} \ 1110 \end{array} \\ \text{SF}=0 \quad \text{ZF}=0 \end{array}$$

# ALU标志应用

- 有符号数运算
  - 利用CF实现多精度计算
  - 利用OF判断溢出
  - 利用OF、SF和ZF判断两数大小
- 无符号数运算
  - 利用CF实现多精度计算
  - 利用CF和ZF判断两数大小

有符号数比较

X与Y 关系	X-Y后标志 OF SF ZF
X=Y	0 0 1
X>Y	0 0 0 1 1 0
X<Y	0 1 0 1 0 0

无符号数比较

X与Y 关系	X-Y后标志 CF ZF
X=Y	0 1
X>Y	0 0
X<Y	1 0

## 示例：标志应用

- 前例中,  $[X]=10000111B$ ,  $[Y]=01111001B$   
     $[X + Y]$ :  $CF=1, OF=0, ZF=1, SF=0$   
     $[X - Y]$ :  $CF=0, OF=1, ZF=0, SF=0$
- 无符号数:  $X = 135, Y = 121, CF, ZF$   
     $X + Y = 0 \rightarrow$  有进位, 结果为零  
     $X - Y = 14 \rightarrow$  无借位,  $X > Y$
- 有符号数(补码):  $X = -121, Y = 121, OF, SF, ZF$   
     $X + Y = 0 \rightarrow$  正确  
     $X - Y = 14 \rightarrow$  溢出,  $X < Y$



# ALU 标志位

- 进位/借位 (**cf**)

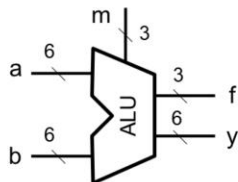
- ADD: {cf, y} = a + b
- SUB: {cf, y} = a - b

- 溢出 (**of**)

- ADD:  $of = (\sim a[5] \& \sim b[5] \& y[5]) \mid (a[5] \& b[5] \& \sim y[5])$
- SUB:  $of = (\sim a[5] \& b[5] \& y[5]) \mid (a[5] \& \sim b[5] \& \sim y[5])$

- 零 (**zf**)

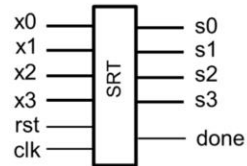
- $zf = \sim |y|$



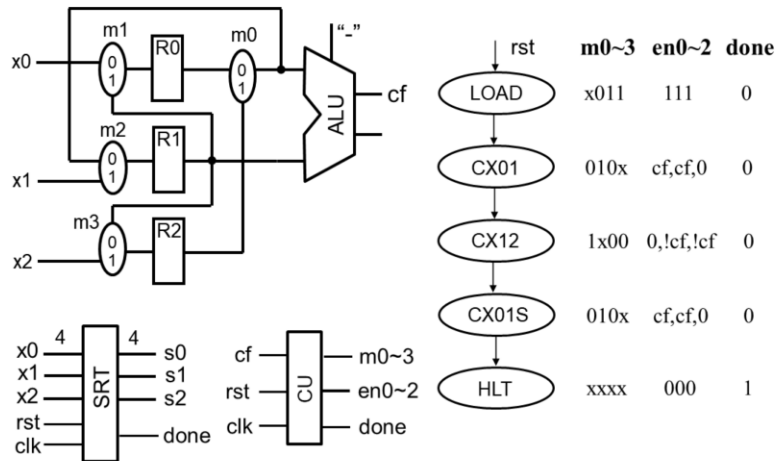
m	a[5]	b[5]	y[5]	of
+	0	0	1	1
	1	1	0	1
	其他			0
-	0	1	1	1
	1	0	0	1
	其他			0

## 排序电路

```
module sort
    #(parameter N = 4)           //数据宽度
    (output [N-1:0] s0, s1, s2, s3, //排序后的四个数据（递增）
    output done,                 //排序结束标志
    input [N-1] x0, x1, x2, x3,  //原始输入数据
    input clk, rst               //时钟（上升沿有效）、复位（高电平有效）
    );
    .....
endmodule
```



# 示例：三个数排序



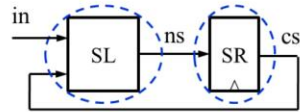
## 示例：三个数排序 (续1)

```

wire [4:0] i0, i1, i2, r0, r1, r2, a;

// Data Path
register #(4) R0 (clk, rst, en0, i0, r0),
          #(4) R1 (clk, rst, en1, i1, r1),
          #(4) R2 (clk, rst, en2, i2, r2);
alu #(4) ALU (, cf,,, a, r1, SUB);
mux2 #(4) M0 (a, r0, r2, m0),
        #(4) M1 (i0, x0, r1, m1),
        #(4) M2 (i1, a, x1, m2),
        #(4) M3 (i2, r1, x2, m3);

// Control Unit
always @(posedge clk, posedge rst)
    if (rst) current_state <= LOAD;
    else
        current_state <= next_state;
    
```



```

always @(*) begin
    case (current_state)
        LOAD: next_state = CX01;
        CX01: next_state = CX12;
        CX12: next_state = CX01S;
        CX01S: next_state = HLT;
        HLT: next_state = HLT;
        default: next_state = HLT;
    endcase
end
    
```

## 示例：三个数排序 (续2)

// 两段式：控制信号--组合输出

```
always @(*) begin
    {m0, m1, m2, m3, en0, en1, en2, done} = 8'h0;
    case (current_state)
        LOAD: {m2, m3, en0, en1, en2} = 5b'11111;
        CX01, CX01S: begin m1 = 1; en0 = cf; en1 = cf; end
        CX12: begin m0 = 1; en1 = ~cf; en2 = ~cf; end
        HLT: done = 1;
    endcase
end
```



# The End