



南京大學  
NANJING UNIVERSITY



# 数字逻辑电路基础

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 布尔代数

- **关于0和1的一套数学运算体系**起源于1850年前后英国数学家乔治·布尔 ( George Boole ) 的工作，因此称为布尔代数。
  - 0和1分别代表逻辑值 “假” 和 “真”
  - 通过逻辑关系可以构建基于0和1的布尔代数运算
  - 最基本的逻辑运算有：与 ( AND )、或 ( OR )、非 ( NOT )，运算符分别为 “ $\cdot$ ” ( “ $\wedge$ ” )、 “ $+$ ” ( “ $\vee$ ” )、 “ $\bar{\phantom{x}}$ ” ( “ $\neg$ ” )

**真值表：反映输入与输出之间的关系**

| A | B | $A \cdot B$ | $A + B$ | $\bar{A}$ | $A \oplus B$ |
|---|---|-------------|---------|-----------|--------------|
| 0 | 0 | 0           | 0       | 1         | 0            |
| 0 | 1 | 0           | 1       | 1         | 1            |
| 1 | 0 | 0           | 1       | 0         | 1            |
| 1 | 1 | 1           | 1       | 0         | 0            |

任何一种逻辑表达式都可写成这三种基本运算的逻辑组合。

例如，异或 ( XOR ) 运算的逻辑表达式为：

$$A \oplus B = \bar{A} \cdot B + A \cdot \bar{B}$$

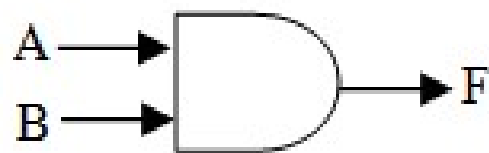
异或运算也称不等价运算。

# 一位逻辑门电路

- 可通过逻辑门电路来实现逻辑运算

- 三种基本门电路：与门、或门、非门

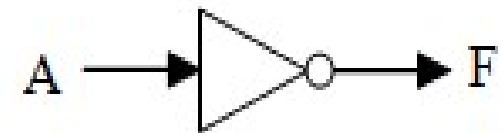
- 其他门电路可以由三种基本门电路组合形成（如异或门电路）



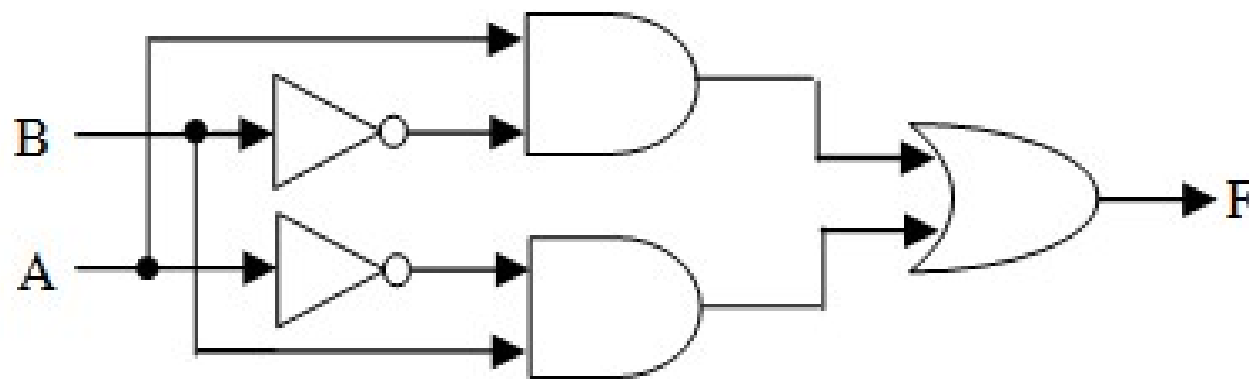
$$F=A \cdot B$$



$$F=A+B$$



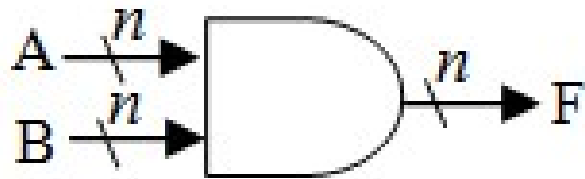
$$F=\bar{A}$$



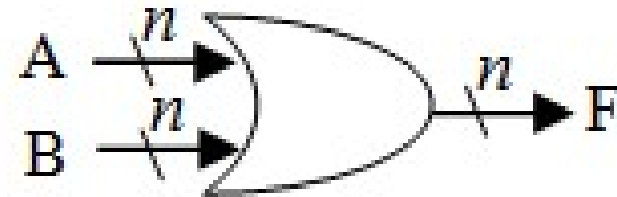
$$F=\bar{A} \cdot B + A \cdot \bar{B} = A \oplus B$$

# n位逻辑门电路

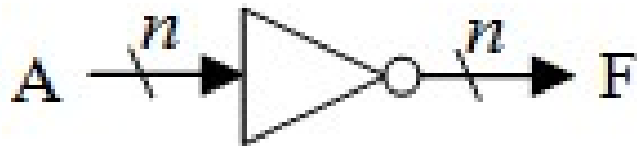
- 对于n位逻辑运算，只要重复使用n个相同的门电路即可
  - 例如，若 $A=A_{n-1}A_{n-2}\dots A_1 A_0$ ， $B=B_{n-1} B_{n-2}\dots B_1 B_0$ ，则与运算 $F=A\cdot B$ ，实际上是按位相与，即 $F_i=A_i\cdot B_i$  ( $0\leq i\leq n-1$ )
  - 假定逻辑值位数为n，则按位与、按位或、按位取反、按位异或的逻辑符号如图所示



$$F=A\cdot B$$



$$F=A+B$$



$$F=\overline{A}$$



$$F=A\oplus B$$

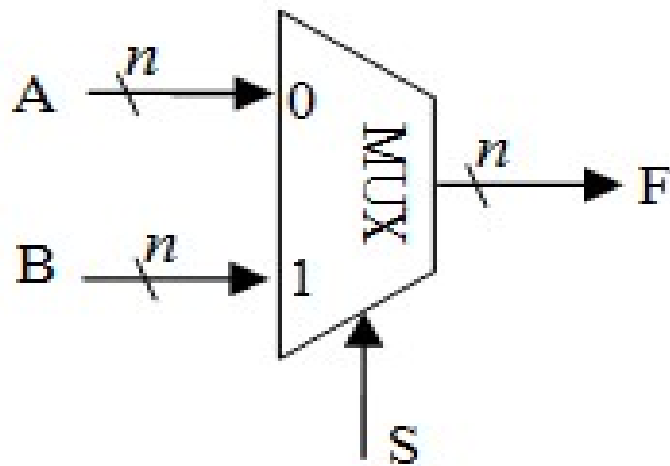
# 组合逻辑部件

---

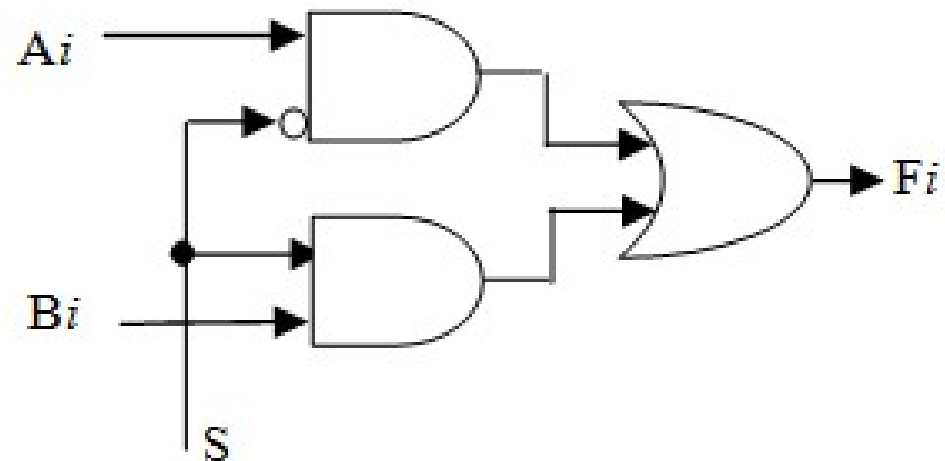
- 根据电路是否具有存储功能，将逻辑电路划分为两种类型
  - **组合逻辑电路**：没有存储功能，其输出仅依赖于当前输入
  - **时序逻辑电路**：具有存储功能，其输出不仅依赖于当前输入，还依赖于存储单元的当前状态
- 可以利用基本逻辑门电路构成一些具有特定功能的组合逻辑部件（**功能部件**）
  - **如译码器、编码器、多路选择器、加法器等**
- 实现一个功能部件的过程
  - 用一个**真值表**描述功能部件的输入和输出之间的关系
  - 根据真值表确定**逻辑表达式**
  - 根据逻辑表达式实现**逻辑电路**

# 多路选择器

- 最简单的多路选择器 ( MUX ) 是二路选择器
  - 有两个输入端A和B，一个输出端F，并有一个控制端S，其功能是：  
当S为0时， $F=A$ ；当S为1时， $F=B$ 。
- k路选择器应有k路输入，因而控制端S的位数应是 $\lceil \log_2 k \rceil$ 
  - 例如，三路或4路选择器，S有两位；5~8路选择器，S有3位。



二路选择器符号



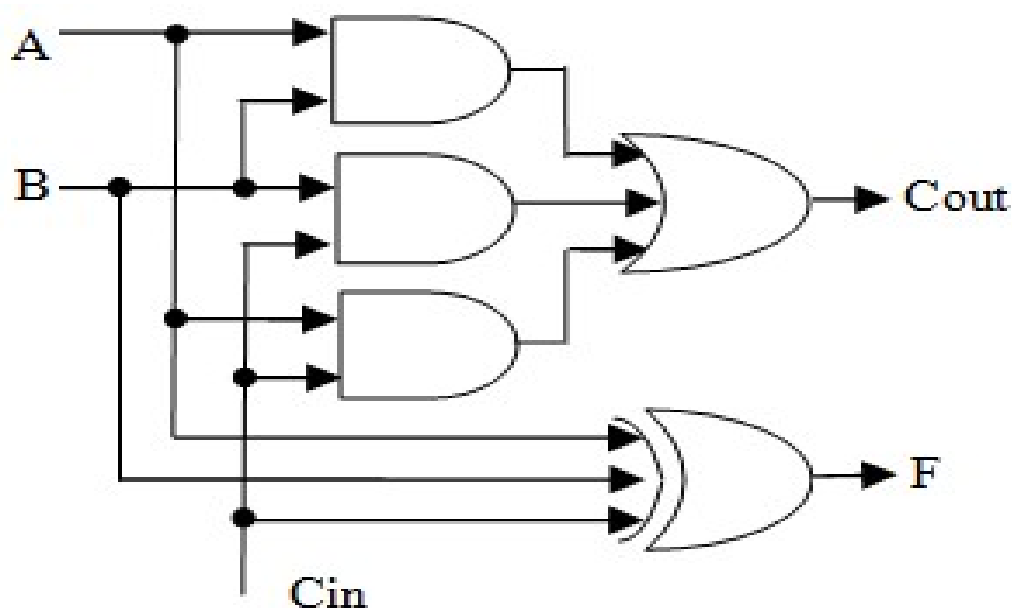
一位二路选择器逻辑电路

# 一位加法器（全加器）

- 一位加法器称为全加器
  - 两个加数为A和B，低位进位为Cin，和为F，向高位的进位为Cout
  - 化简后，逻辑表达式如下

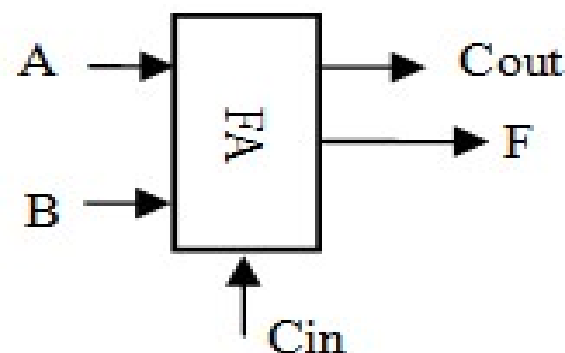
$$F = A \oplus B \oplus C_{in}$$

$$C_{out} = A \cdot B + A \cdot C_{in} + B \cdot C_{in}$$



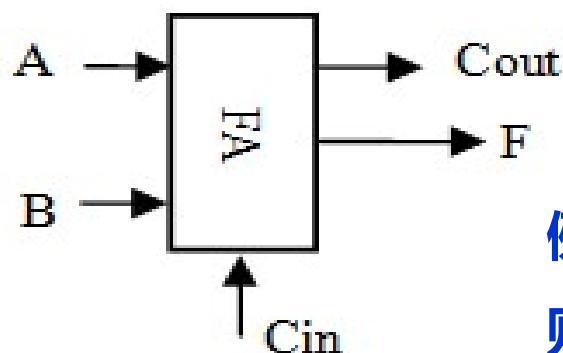
| A | B | Cin | F | Cout |
|---|---|-----|---|------|
| 0 | 0 | 0   | 0 | 0    |
| 0 | 0 | 1   | 1 | 0    |
| 0 | 1 | 0   | 1 | 0    |
| 0 | 1 | 1   | 0 | 1    |
| 1 | 0 | 0   | 1 | 0    |
| 1 | 0 | 1   | 0 | 1    |
| 1 | 1 | 0   | 0 | 1    |
| 1 | 1 | 1   | 1 | 1    |

全加器真值表



# n位加法器

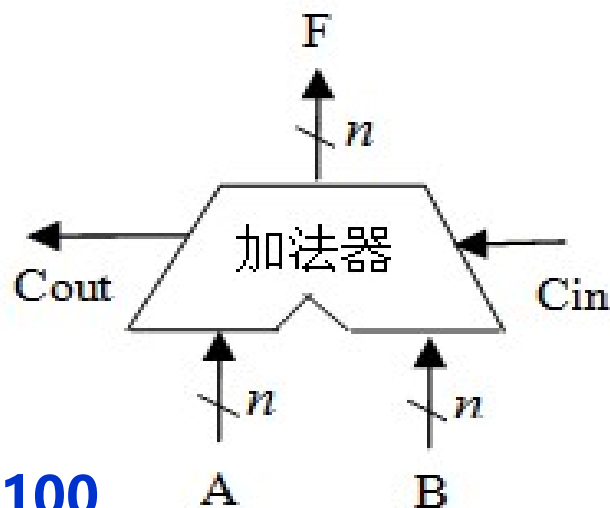
- n位加法器可用n个全加器实现



全加器符号

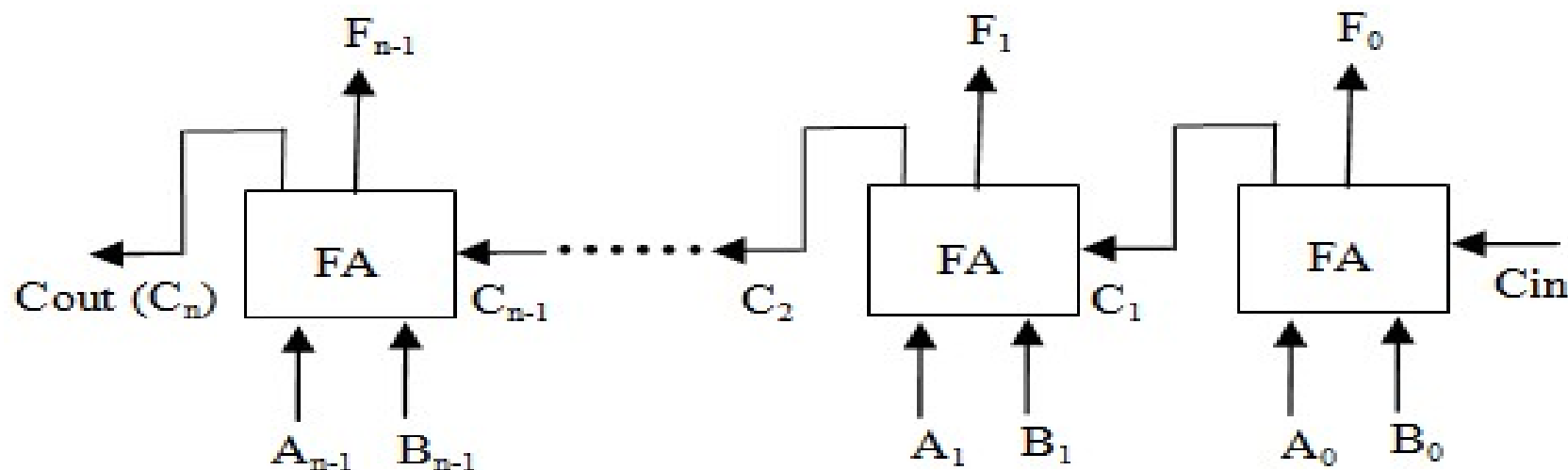
例：n=4，A=1001，B=1100

则：F=0101，Cout=1



加法器符号

无符号整数加

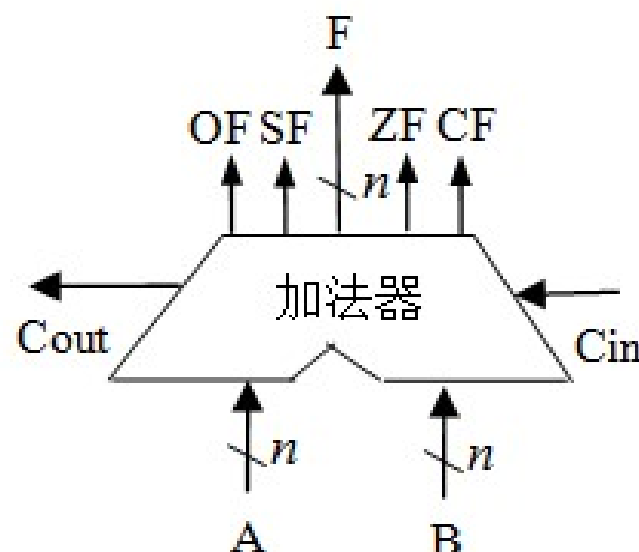


**重要认识：**加法由逻辑部件实现，而其他所有算术运算部件都基于加法器和逻辑运算实现，因此，所有算术运算是基于0和1以及逻辑运算实现的！

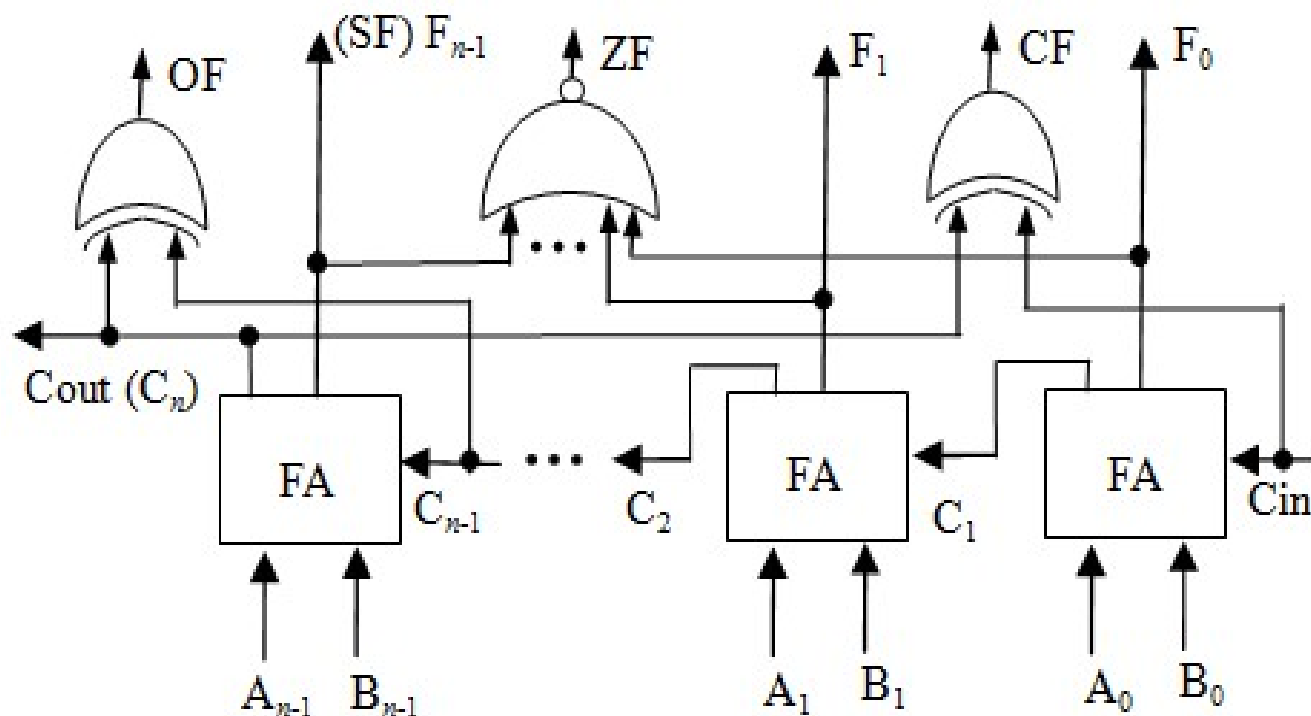


# n位带标志加法器

- n位加法器无法用于两个n位带符号整数（补码）相加，无法判断是否溢出
- 程序中经常需要比较大小，通过（在加法器中）做减法得到的标志信息来判断



带标志加法器符号



带标志加法器的逻辑电路

溢出标志OF：

$$OF = C_n \oplus C_{n-1}$$

符号标志SF：

$$SF = F_{n-1}$$

零标志ZF=1当且仅当F=0；

进位/借位标志CF：

$$CF = Cout \oplus Cin$$

# n位整数加/减运算器

先看一个C程序段：

```
int x=9, y=-6, z1, z2;  
z1=x+y;  
z2=x-y;
```

补码的定义 假定补码有n位，则：

$$[X]_{\text{补}} = 2^n + X \quad (-2^n \leq X < 2^n, \text{mod } 2^n)$$

问题：上述程序段中，x和y的机器数是什么？z1和z2的机器数是什么？

回答：x的机器数为 $[x]_{\text{补}}$ ，y的机器数为 $[y]_{\text{补}}$ ；

z1的机器数为 $[x+y]_{\text{补}}$ ；

z2的机器数为 $[x-y]_{\text{补}}$ 。

因此，计算机中需要有一个电路，能够实现以下功能：

已知 $[x]_{\text{补}}$ 和 $[y]_{\text{补}}$ ，计算 $[x+y]_{\text{补}}$ 和 $[x-y]_{\text{补}}$ 。

$$[-y]_{\text{补}} = \overline{[y]_{\text{补}}} + 1$$

根据补码定义，有如下公式：

$$[x+y]_{\text{补}} = 2^n + x + y = 2^n + x + 2^n + y = [x]_{\text{补}} + [y]_{\text{补}} \pmod{2^n}$$

$$[x-y]_{\text{补}} = 2^n + x - y = 2^n + x + 2^n - y = [x]_{\text{补}} + [-y]_{\text{补}} \pmod{2^n}$$

# n位整数加/减运算器

- 补码加减运算公式

$$[A+B]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}} \pmod{2^n}$$

$$[A-B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2^n}$$

— 实现减法的主要工作在于：求 $[-B]_{\text{补}}$

问题：如何求 $[-B]_{\text{补}}$ ？

$$[-B]_{\text{补}} = \overline{[B]_{\text{补}}} + 1$$

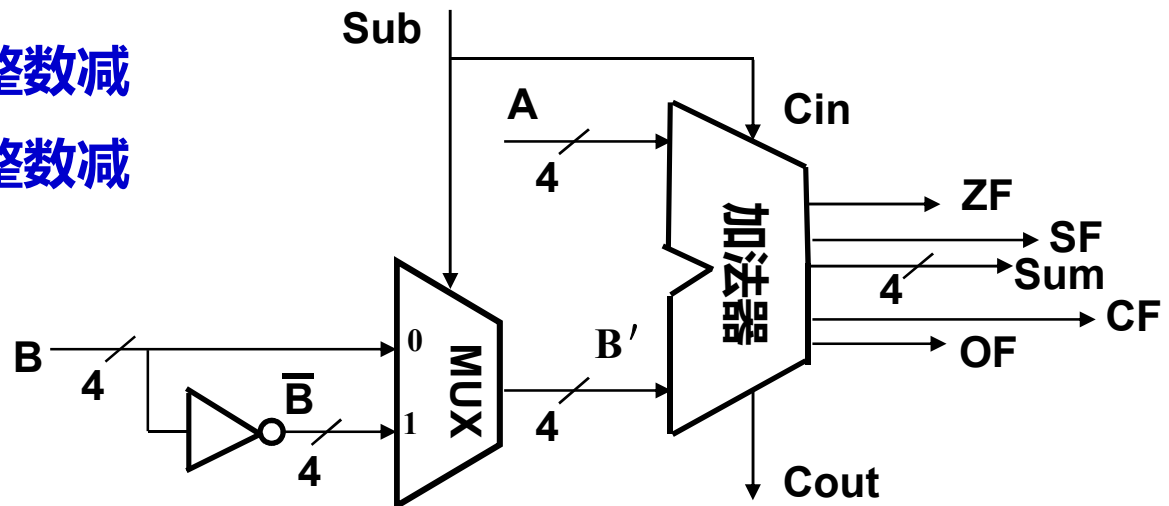
- 利用带标志加法器，可构造整数加/减运算器，进行以下运算：

无符号整数加、无符号整数减

带符号整数加、带符号整数减

在整数加/减运算部件基础上，加上寄存器、移位器以及控制逻辑，就可实现ALU、乘/除运算以及浮点运算电路

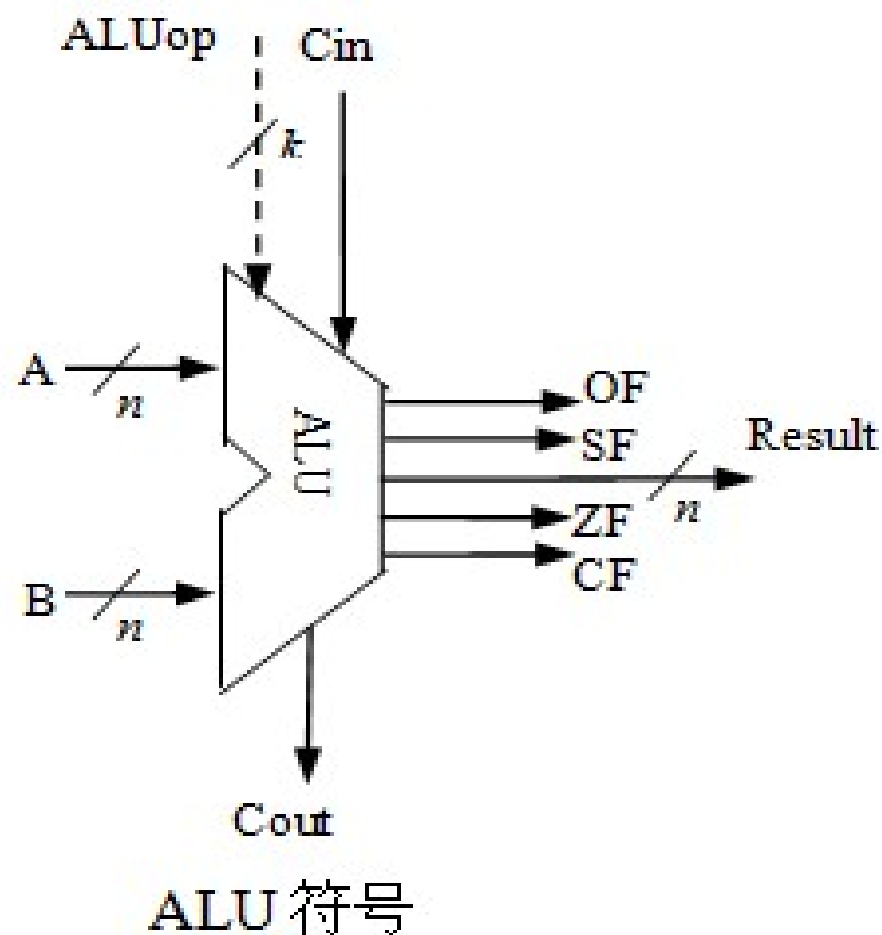
当Sub为1时，做减法  
当Sub为0时，做加法



整数加/减运算部件

# 算术逻辑部件 (ALU)

- 进行**基本**算术运算与逻辑运算
  - 无符号整数加、减
  - 带符号整数加、减
  - 与、或、非、异或等逻辑运算
- 核心电路是**带标志加法器**
- 输出除**和/差等**，还有**标志信息**
- 有一个**操作控制端** (ALUop)，用来决定ALU所执行的处理功能。ALUop的位数k决定了操作的种类，例如，当位数k为3时，ALU最多只有 $2^3=8$ 种操作。



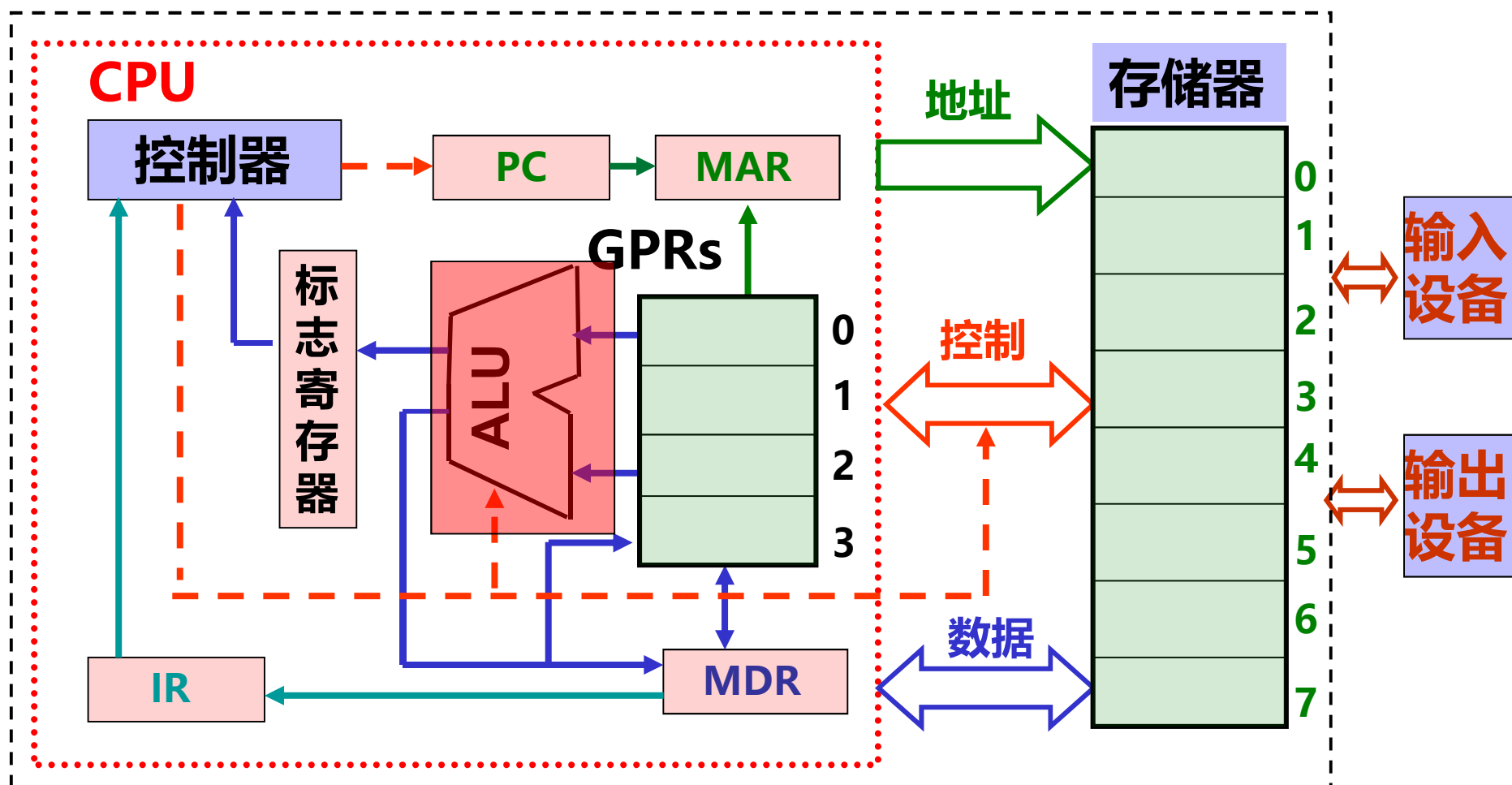
| ALUop | Result | ALUop | Result | ALUop | Result       | ALUop | Result |
|-------|--------|-------|--------|-------|--------------|-------|--------|
| 0 0 0 | A加B    | 0 1 0 | A与B    | 1 0 0 | A取反          | 1 1 0 | A      |
| 0 0 1 | A减B    | 0 1 1 | A或B    | 1 0 1 | $A \oplus B$ | 1 1 1 | 未用     |

# 回顾：认识计算机中最基本的部件

**CPU**：中央处理器；**PC**：程序计数器；**MAR**：存储器地址寄存器

**ALU**：算术逻辑部件；**IR**：指令寄存器；**MDR**：存储器数据寄存器

**GPRs**：通用寄存器组（由若干通用寄存器组成）





南京大學  
NANJING UNIVERSITY



# 从C表达式到逻辑电路

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 开场白

---

上一讲我们介绍了计算机中最基本的运算电路，我们知道，计算机中的所有运算都是由相应的运算电路完成的，而这些运算电路是由基本的逻辑门电路实现的。

那么，计算机是如何知道在运算电路中该执行什么操作？该对什么样的操作数进行运算的呢？

本讲主要介绍高级语言程序中的表达式、运算类指令和运算电路之间的关系。

# C语言支持的基本数据类型

| C语言声明                    | 操作数类型   | 存储长度（位） |
|--------------------------|---------|---------|
| (unsigned) char          | 整数 / 字节 | 8       |
| (unsigned) short         | 整数 / 字  | 16      |
| (unsigned) int           | 整数 / 双字 | 32      |
| (unsigned) long int      | 整数 / 双字 | 32      |
| (unsigned) long long int | -       | 2×32    |
| char *                   | 整数 / 双字 | 32      |
| float                    | 单精度浮点数  | 32      |
| double                   | 双精度浮点数  | 64      |
| long double              | 扩展精度浮点数 | 80 / 96 |

整数类型分：无符号整数和带符号整数



# 从C表达式到运算类指令

---

## C语言程序中的基本数据类型、基本运算类型

### –基本数据类型

- 无符号数(二进制位串)、带符号整数(补码)
- 浮点数(IEEE 754标准)
- 位串、字符(串)(ASCII码)

### –基本运算类型

- 算术(+ - \* / % > < >= <= == !=)
- 按位(| & ~ ^)
- 逻辑(|| && !)
- 移位(<< >>)
- 扩展和截断

例如：对于C赋值语句  $y=(x>>2)+k$ ，如何在计算机中实现呢？

转换为指令序列，通过执行运算类指令来实现！

# 从运算类指令到运算电路

---

计算机如何实现高级语言程序中的运算？

—将各类表达式编译（转换）为指令序列

例如： $y = (x \gg 2) + k$  转换为以下指令序列：

`sarw $2, %ax ; x >> 2`

`addw %bx, %ax ; (x >> 2) + k`

—计算机直接执行指令来完成运算

控制器对指令进行译码，产生控制信号送运算电路


—操作数在运算电路中运算

`sarw $2, %ax`：将操作数“2”和“R[ax]”送移位器运算

`addw %bx, %ax`：将R[ax]和R[bx]送整数加减器中运算

移位器和整数加减运算器都是由逻辑门电路构成的！

# 数据的运算

- 高级语言程序中涉及的运算（以C语言为例）
    - 整数算术运算、浮点数算术运算
    - 按位、逻辑、移位、位扩展和位截断等运算
  - 指令集中涉及到的运算
    - 涉及到的定点数运算
      - 算术运算
        - 带符号整数：取负 / 符号扩展 / 加 / 减 / 乘 / 除 / 算术移位
        - 无符号整数：0扩展 / 加 / 减 / 乘 / 除 / 逻辑左移 / 逻辑右移
      - 逻辑运算
        - 逻辑操作：与 / 或 / 非 / ...
    - 涉及到的浮点数运算：加、减、乘、除
  - 指令中的运算操作在运算电路中进行
    - 基本运算部件ALU、通用寄存器组，以及其他部件
- 
- The diagram consists of two large, light blue curved arrows. The first arrow starts at the top right of the first bullet point ('高级语言程序中涉及的运算') and points down to the second bullet point ('指令集中涉及到的运算'). The second arrow starts at the top right of the second bullet point and points down to the third bullet point ('指令中的运算操作在运算电路中进行').



南京大學  
NANJING UNIVERSITY



# C语言中的各类运算

南京大学

计算机科学与技术系

袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 开场白

---

上一讲谈到，在高级语言程序的表达式中的各类运算，会被编译器转换为相应的运算指令，程序运行时，**CPU**执行这些指令，控制操作数在运算电路中被处理。

本讲主要介绍**C**语言程序中涉及的运算，包括算术运算、按位运算、逻辑运算、移位运算等。

# C语言程序中涉及的运算

---

- 算术运算（最基本的运算）

- 无符号数、带符号整数、浮点数的+、-、\*、/、%运算等

- 按位运算

- 用途

- 对位串实现“掩码”（mask）操作或相应的其他处理  
（主要用于对多媒体数据或状态/控制信息进行处理）

- 操作

- 按位或：“|”
    - 按位与：“&”
    - 按位取反：“~”
    - 按位异或：“^”

如何从数据y中提取低位字节，并使高字节为0？

可用“&”实现“掩码”操作：`y & 0x00FF`

例如，当`y=0x0B2C`时，得到结果为：`0x002C`

# C语言程序中涉及的运算

- 移位运算

- 用途

- 提取部分信息
    - 扩大或缩小2、4、8...倍

- 操作

- 左移:  $x \ll k$ ; 右移:  $x \gg k$
    - 从运算符无法区分逻辑移位还是算术移位, 由x的类型确定
    - 若x为无符号数: 逻辑左(右)移

高(低)位移出, 低(高)位补0, 可能溢出!

问题: 何时可能发生溢出? 如何判断溢出?

若高位移出的是1, 则左移时发生溢出

- 若x为带符号整数: 算术左移、算术右移

左移: 高位移出, 低位补0。可能溢出!

溢出判断: 若移出的位不等于新的符号位, 则溢出。

右移: 低位移出, 高位补符, 可能发生有效数据丢失。

例: 某字长为8的机器中, x、y和z都是8位带符号整数, 已知 $x = -81$ , 则 $y = x/2 = ?$   $z = 2 * x = ?$

$-81 = -1010001B$ , 故x的机器数为10101111

$y = x/2 \rightarrow x \gg 1 : 11010111$

$z = 2 * x \rightarrow x \ll 1 : 01011110$

~~$y = -41$~~  ? 有效数据丢失  ~~$z = 94$~~  溢出

# C语言程序中涉及的运算举例

---

## 移位运算和按位运算举例

对于一个 $n$  ( $n \geq 8$ ) 位的变量 $x$ ，请根据C语言中按位运算的定义，写出满足下列要求的C语言表达式。

- (1)  $x$ 的最高有效字节不变，其余各位全变为0。
- (2)  $x$ 的最低有效字节不变，其余各位全变为0。
- (3)  $x$ 的最低有效字节全变为0，其余各位取反。
- (4)  $x$ 的最低有效字节全变1，其余各位不变。

参考答案：

- (1)  $(x >> (n-8)) << (n-8)$
- (2)  $x \& 0xFF$
- (3)  $((x \wedge \sim 0xFF) >> 8) << 8$
- (4)  $x | 0xFF$



# C语言程序中涉及的运算

---

- 逻辑运算

- 用途

- 用于关系表达式的运算

例如 , if ( x>y and i<100 ) then .....中的 “and” 运算

- 操作

- “||” 表示 “OR” 运算
    - “&&” 表示 “AND” 运算

例如 , if ((x>y) && (i<100)) then .....

- “!” 表示 “NOT” 运算

- 与按位运算的差别

- 符号表示不同 : & ~ && ; | ~ || ; .....
    - 运算过程不同 : 按位 ~ 整体
    - 结果类型不同 : 位串 ~ 逻辑值

# C语言程序中涉及的运算

---

- 位扩展和位截断运算

- 用途

- 类型转换时可能需要数据扩展或截断

- 操作

- 没有专门操作运算符，根据类型转换前、后数据长短确定是扩展还是截断

- 扩展：短转长

- 无符号数：0扩展（前面补0）

- 带符号整数：符号扩展（前面补符）

- 截断：长转短

- 强行将高位丢弃，故可能发生“溢出”

# C语言程序中涉及的运算

---

例1（扩展操作）：

在大端机上输出si, usi, i, ui的十进制和十六进制值是什么？

**short si = -32768;**

**unsigned short usi = si;**

**int i = si;**

**unsigned ui = usi;**

**si = -32768    80 00**

**usi = 32768    80 00**

**i = -32768    FF FF 80 00**

**ui = 32768    00 00 80 00**

**带符号整数：符号扩展**

**无符号整数：0扩展**

# C语言程序中涉及的运算

---

**例2（截断操作）：i 和 j 是否相等？**

```
int i = 32768;
```

```
short si = (short) i;
```

```
int j = si;
```

**不相等！**

**i = 32768    00 00 80 00**

**si = -32768    80 00**

**j = -32768    FF FF 80 00**

**原因：对i截断时发生了“溢出”，即：32768截断为16位数时，因其超出16位能表示的最大值，故无法截断为正确的16位数！**



南京大学  
NANJING UNIVERSITY



# 整数加减运算

南京大学

计算机科学与技术系

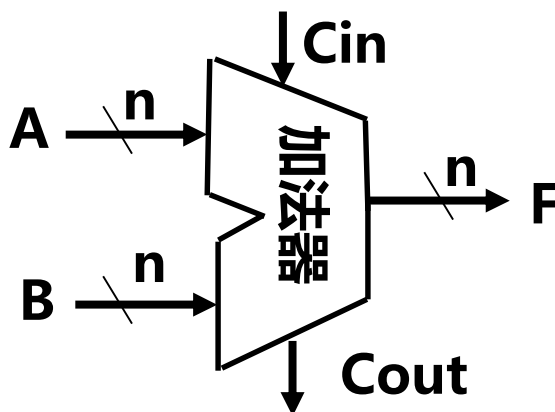
袁春风

email: [cfyuan@nju.edu.cn](mailto:cfyuan@nju.edu.cn)

2015.6

# 整数加、减运算

- C语言程序中的整数有
  - 带符号整数，如char、short、int、long型等
  - 无符号整数，如unsigned char、unsigned short、unsigned等
- 指针、地址等通常被说明为无符号整数，因而在进行指针或地址运算时，需要进行无符号整数的加、减运算
- 无符号整数和带符号整数的加、减运算电路完全一样，这个运算电路称为整数加减运算部件，基于带标志加法器实现
- 计算机中的加法器，因为只有n位，所以是一种模 $2^n$ 运算系统！



例：n=4, A=1001, B=1100

则：F=0101, Cout=1

# 回顾：整数加减运算部件

- 补码加减运算公式

$$[A+B]_{\text{补}} = [A]_{\text{补}} + [B]_{\text{补}} \pmod{2^n}$$

$$[A-B]_{\text{补}} = [A]_{\text{补}} + [-B]_{\text{补}} \pmod{2^n}$$

- 补码加减运算要点和运算部件

- 加、减法运算统一采用加法来处理
- 符号位(最高有效位MSB)和数值位一起参与运算
- 直接用Adder实现两个数的加运算（模运算系统）

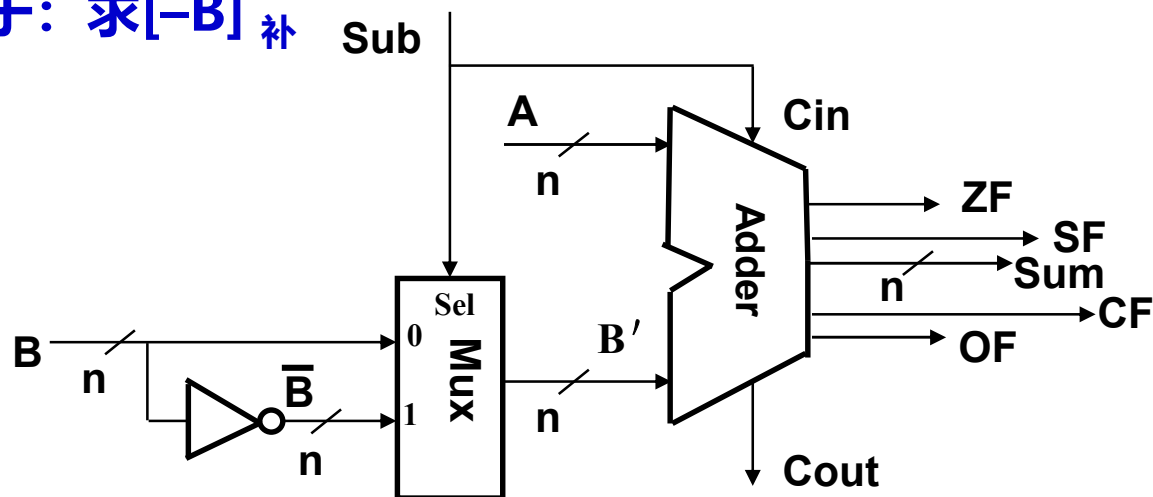
**问题：模是多少？** 运算结果高位丢弃，保留低 $n$ 位，相当于取模 $2^n$

- 实现减法的主要工作在于：求 $[-B]_{\text{补}}$

**问题：如何求 $[-B]_{\text{补}}$ ？**

$$[-B]_{\text{补}} = \overline{B} + 1$$

当Sub为1时，做减法  
当Sub为0时，做加法



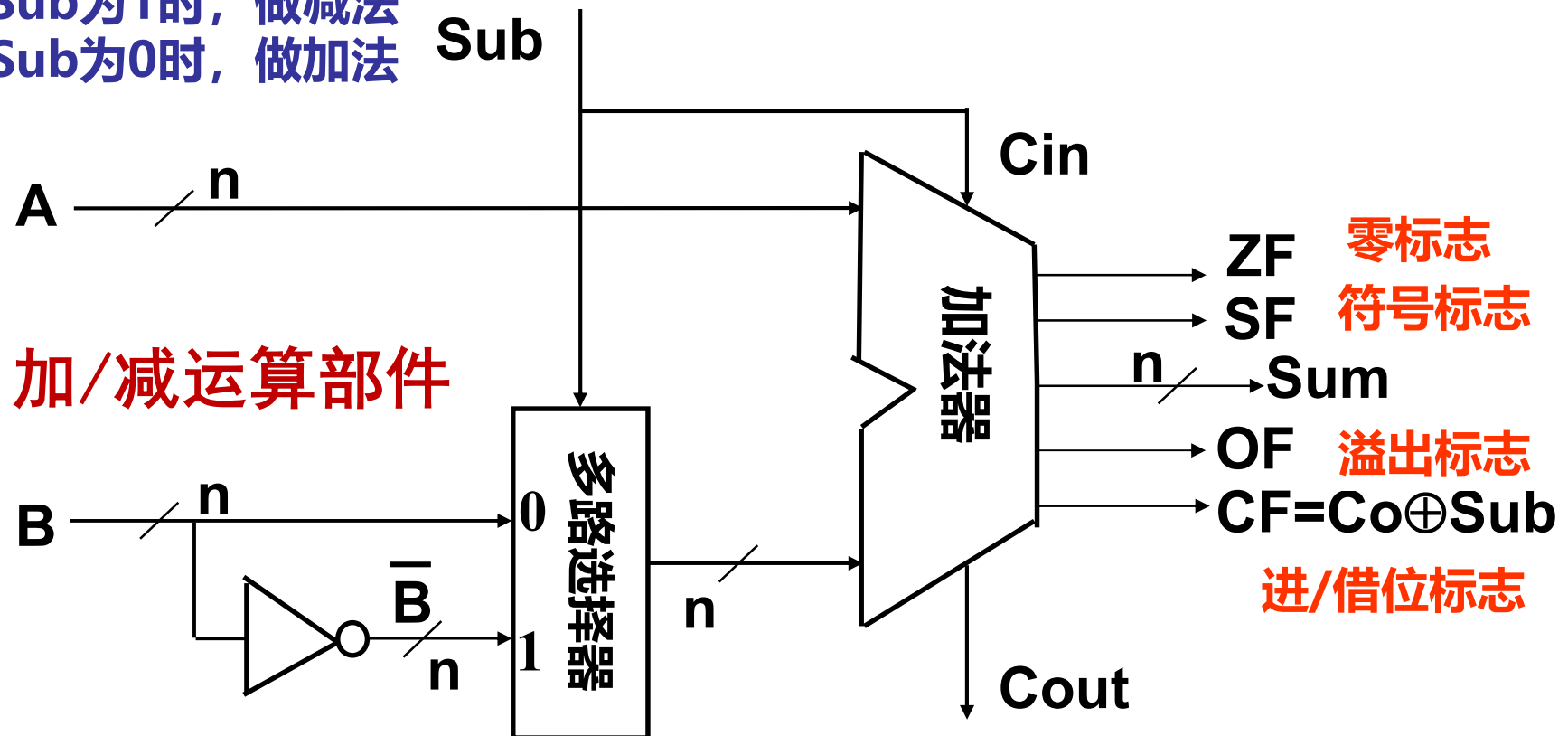
# 所有运算电路的核心

**重要认识1:** 计算机中所有运算都基于加法器实现!

**重要认识2:** 加法器不知道所运算的是带符号数还是无符号数。

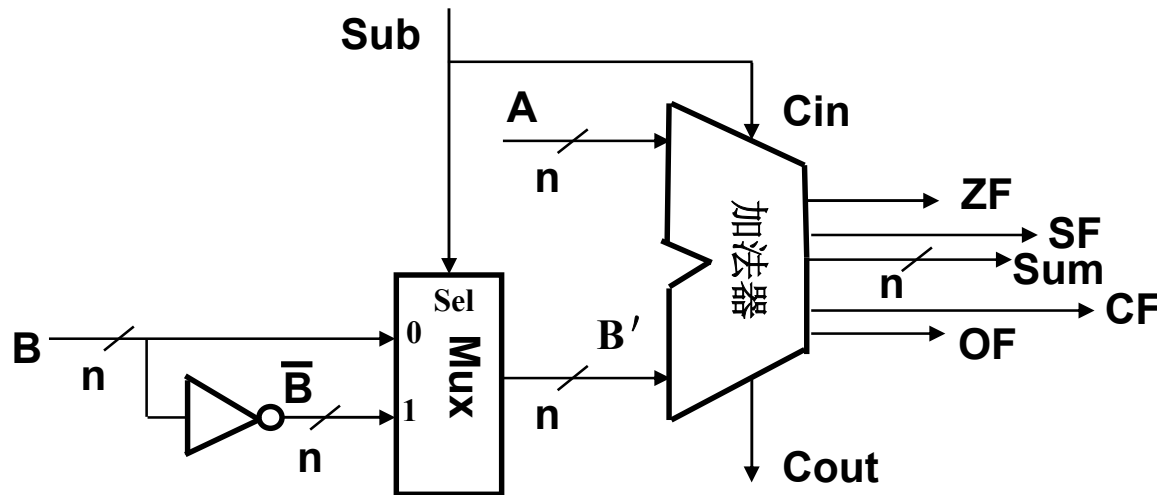
**重要认识3:** 加法器不判定对错，总是取低n位作为结果，并生成标志信息。

当Sub为1时，做减法  
当Sub为0时，做加法





# 条件标志位（条件码CC）



整数加/减运算部件

问题：为什么要生成并保存条件标志？

为了在分支指令（条件转移指令）中被用作是否转移执行的条件！

问题：OF=? ZF=?  
SF=? CF=?

```
if (i>j) {  
    ...  
}
```

还记得如何得到各个标志位吗？

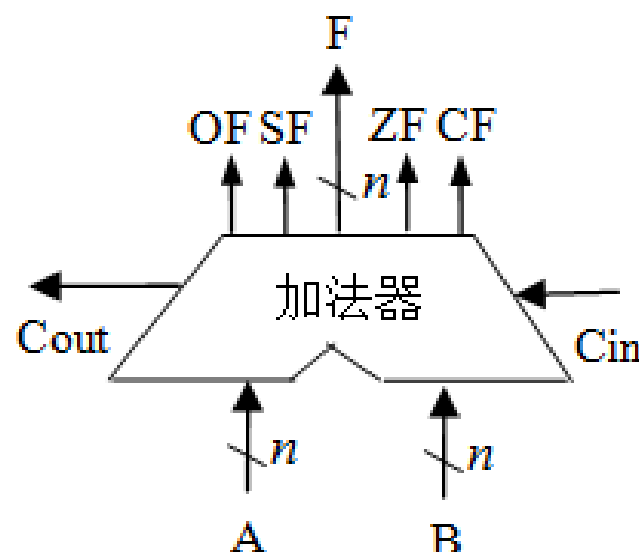
**OF**：若A与B' 同号但与Sum不同号，则1；否则0。 **SF**：sum符号

**ZF**：如Sum为0，则1，否则0。 **CF**： $Cout \oplus sub$

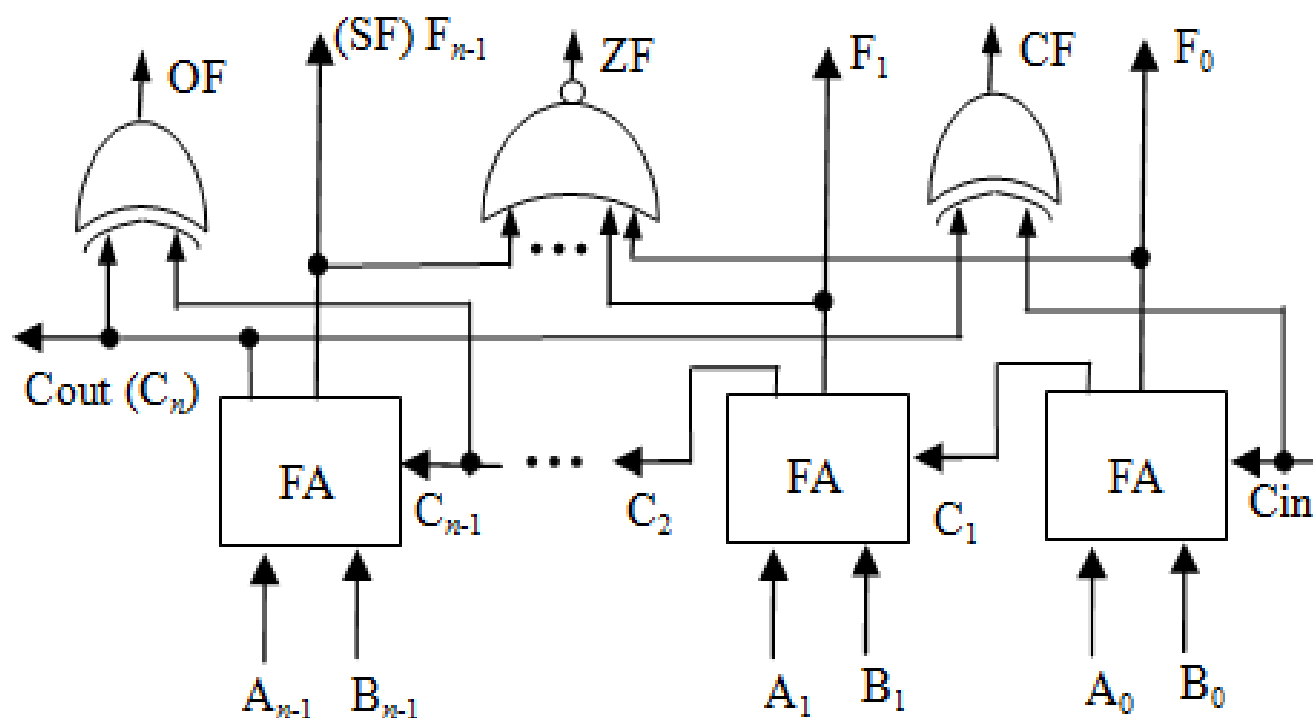
- 零标志**ZF**、溢出标志**OF**、进/借位标志**CF**、符号标志**SF**称为条件标志。
- **条件标志（Flag）**在运算电路中产生，被记录到专门的寄存器中
- 存放标志的寄存器通常称为**程序/状态字寄存器**或**标志寄存器**。每个标志对应标志寄存器中的一个标志位。 如，IA-32中的**EFLAGS寄存器**

# n位带标志加法器

- n位加法器无法用于两个n位带符号整数(补码)相加, 无法判断是否溢出
- 程序中经常需要比较大小, 通过(在加法器中)做减法得到的标志信息来判断



带标志加法器符号



带标志加法器的逻辑电路

溢出标志OF:

$$OF = C_n \oplus C_{n-1}$$

符号标志SF:

$$SF = F_{n-1}$$

零标志ZF=1当且仅当F=0;

进位/借位标志CF:

$$CF = Cout \oplus Cin$$

# 整数加法举例

做加法时，主要判断是否溢出

无符号加溢出条件：CF=1

带符号加溢出条件：OF=1

若n=8，计算107+46=?

$$107_{10} = 0110\ 1011_2$$

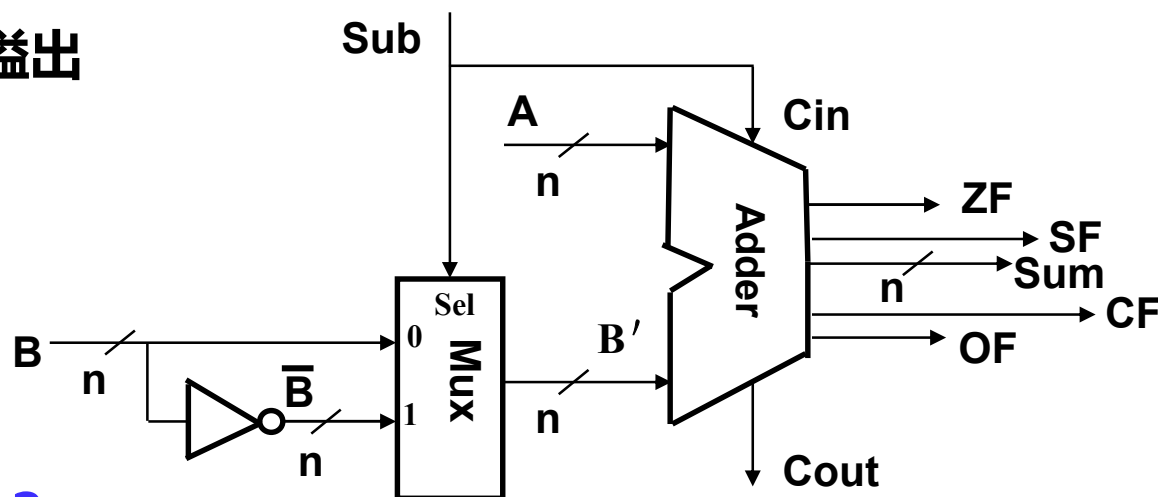
$$46_{10} = 0010\ 1110_2$$

$$\boxed{0}1001\ 1001$$

进位是真正的符号：+153

无符号：sum=153，因为CF=0，故未发生溢出，结果正确！

带符号：sum= -103，因为OF=1，故发生溢出，结果错误！



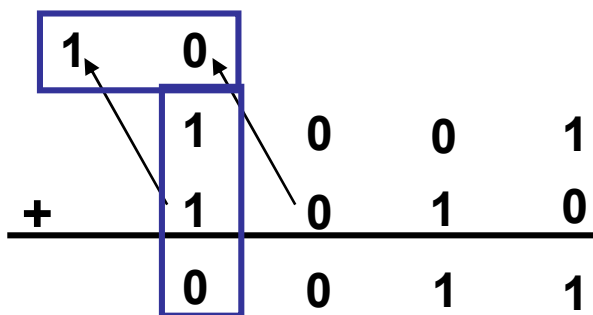
整数加/减运算部件

两个正数相加，结果为负数，  
故溢出！即OF=1

溢出标志OF=1、零标志ZF=0、  
符号标志SF=1、进位标志CF=0

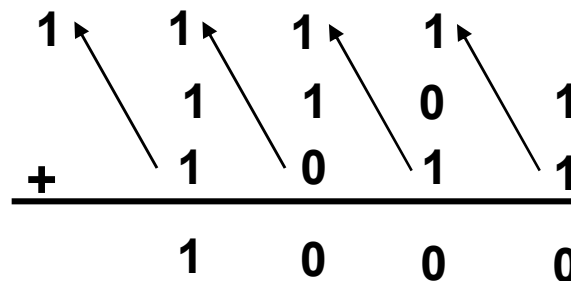
# 整数减法举例

$$\begin{aligned} -7 - 6 &= -7 + (-6) = +3 \text{ } \times \\ 9 - 6 &= 3 \text{ } \checkmark \end{aligned}$$



OF=1、ZF=0  
SF=0、借位CF=0

$$\begin{aligned} -3 - 5 &= -3 + (-5) = -8 \text{ } \checkmark \\ 13 - 5 &= 8 \text{ } \checkmark \end{aligned}$$



OF=0、ZF=0、  
SF=1、借位CF=0

带符号 (1) 最高位和次高位的进位不同  
溢出: (2) 和的符号位和加数的符号位不同

无符号减溢出: 差为负数, 即借位CF=1

做减法以比较大小, 规则:  
Unsigned: CF=0时, 大于  
Signed: OF=SF时, 大于

验证:  $9 > 6$ , 故CF=0;  $13 > 5$ , 故CF=0

验证:  $-7 < 6$ , 故OF≠SF  
 $-3 < 5$ , 故OF≠SF

# 整数减法举例

unsigned int x=134;

unsigned int y=246;

int m=x;

int n=y;

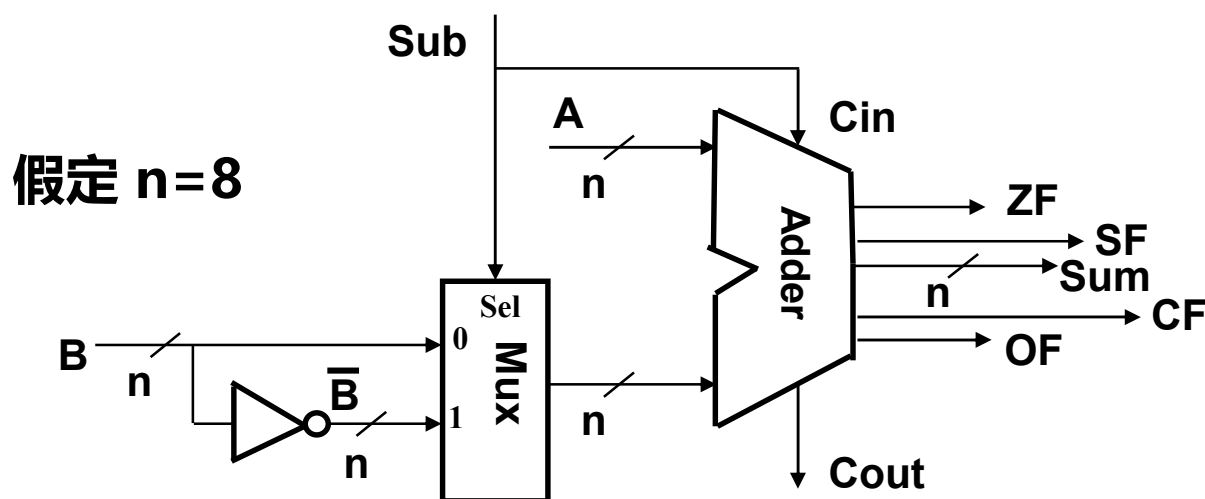
unsigned int **z1=x-y;**

unsigned int **z2=x+y;**

int **k1=m-n;**

int **k2=m+n;**

无符号和带符号加减运算都用该部件执行



x和m的机器数一样: 1000 0110, y和n的机器数一样: 1111 0110

z1和k1的机器数一样: 1001 0000, **CF=1**, **OF=0**, **SF=1**

z1的值为144 ( $=134-246+256$ ,  $x-y<0$ ), **k1的值为-112**。

无符号减公式:

$$\text{result} = \begin{cases} x-y & (x-y > 0) \\ x-y+2^n & (x-y < 0) \end{cases}$$

带符号减公式:

$$\text{result} = \begin{cases} x-y-2^n & (2^{n-1} \leq x-y) & \text{正溢出} \\ x-y & (-2^{n-1} \leq x-y < 2^{n-1}) & \text{正常} \\ x-y+2^n & (x-y < -2^{n-1}) & \text{负溢出} \end{cases}$$

# 整数加法举例

unsigned int x=134;

unsigned int y=246;

int m=x;

int n=y;

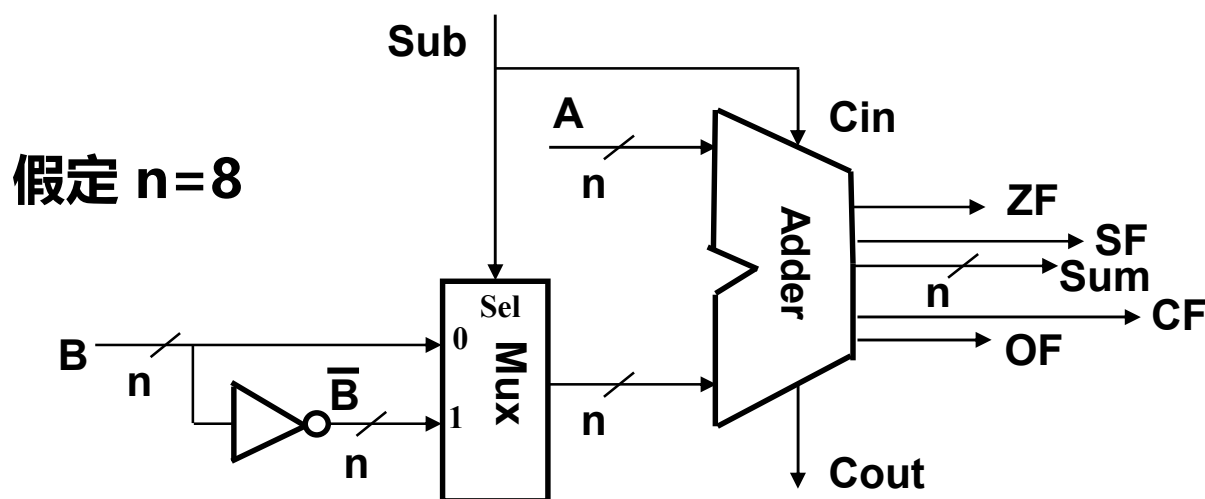
unsigned int z1=x-y;

unsigned int z2=x+y;

int k1=m-n;

int k2=m+n;

无符号和带符号加减运算都用该部件执行



x和m的机器数一样: 1000 0110, y和n的机器数一样: 1111 0110

z2和k2的机器数一样: 0111 1100, CF=1, OF=1, SF=0

z2的值为124 (=134+246-256, x+y>256)

k2的值为124 (= -122+(-10)+256, m+n=-132<-128, 即负溢出)

带符号加公式:

无符号加公式:

$$\text{result} = \begin{cases} x+y & (x+y < 2^n) \\ x+y-2^n & (2^n \leq x+y < 2^{n+1}) \end{cases}$$

$$\text{result} = \begin{cases} x+y-2^n & (2^{n-1} \leq x+y) & \text{正溢出} \\ x+y & (-2^{n-1} \leq x+y < 2^{n-1}) & \text{正常} \\ x+y+2^n & (x+y < -2^{n-1}) & \text{负溢出} \end{cases}$$

# 无符号整数加法溢出判断程序

如何用程序判断一个无符号数相加没有发生溢出

$$\text{result} = \begin{cases} x+y & (x+y < 2^n) \\ x+y-2^n & (2^n \leq x+y < 2^{n+1}) \end{cases}$$

发生溢出时，一定满足  $\text{result} < x$  and  $\text{result} < y$   
否则，若  $x+y-2^n \geq x$ ，则  $y \geq 2^n$ ，这是不可能的！

*/\* Determine whether arguments can be added  
without overflow \*/*

```
int uadd_ok(unsigned x, unsigned y)
{
    unsigned sum = x+y;
    return sum >= x;
}
```

# 带符号整数加法溢出判断程序

如何用程序判断一个带符号整数相加没有发生溢出

$$\text{result} = \begin{cases} x+y-2^n & (2^{n-1} \leq x+y) & \text{正溢出} \\ x+y & (-2^{n-1} \leq x+y < 2^{n-1}) & \text{正常} \\ x+y+2^n & (x+y < -2^{n-1}) & \text{负溢出} \end{cases}$$

**/\* Determine whether arguments can be added without overflow \*/**

```
int tadd_ok(int x, int y) {  
    int sum = x+y;  
    int neg_over = x < 0 && y < 0 && sum >= 0;  
    int pos_over = x >= 0 && y >= 0 && sum < 0;  
    return !neg_over && !pos_over;  
}
```



# 带符号整数减法溢出判断程序

以下程序检查带符号整数相减是否溢出有没有问题？

$$\text{result} = \begin{cases} x+y-2^n & (2^{n-1} \leq x+y) & \text{正溢出} \\ x+y & (-2^{n-1} \leq x+y < 2^{n-1}) & \text{正常} \\ x+y+2^n & (x+y < -2^{n-1}) & \text{负溢出} \end{cases} \quad \text{带符号整数加}$$

$$\text{result} = \begin{cases} x-y-2^n & (2^{n-1} \leq x-y) & \text{正溢出} \\ x-y & (-2^{n-1} \leq x-y < 2^{n-1}) & \text{正常} \\ x-y+2^n & (x-y < -2^{n-1}) & \text{负溢出} \end{cases} \quad \text{带符号整数减}$$

**/\* Determine whether arguments can be subtracted without overflow \*/**

**/\* WARNING: This code is buggy. \*/**

```
int tsub_ok(int x, int y) {  
    return tadd_ok(x, -y);  
}
```

当  $x=0$ ,  $y=0x80000000$  时，该函数判断错误

带符号减的溢出判断函数如何实现呢？

无符号减的溢出判断函数又如何实现呢？