南京大学120周年校庆
120th ANNIVERSARY
NANJING UNIVERSITY
1902 - 2022

1902 2022

数据的运算

Data Operation

李杉杉

南京大学120周年校庆
120th ANNIVERSARY
NANJING UNIVERSITY
1902 - 2022

CONTENT

目录


南京大学120周年校庆
120th ANNIVERSARY
NANJING UNIVERSITY
1902 – 2022

01 逻辑运算

位组合

- 二进制是计算机编码存储和操作信息的核心
- 计算机要解决一个真正的问题，必须能唯一的识别出许多不同的数值，而不仅仅是0和1
- 唯一的识别多个数值，必须对多个位进行组合
- 如果用8位（对应8根线路上的电压）
  - 0100 1110，1110 0111……
  - 最多能区分出256（即2^8）个不同的值
- 如果有k位，最多能区分出2^k个不同的值
  - 每一种组合都是一个编码，对应着某个特定的值

位组合运算

- 除了需要表示出不同的数值之外，还需要对这种表示出来的信息进行运算
  - 1679年，德国数学家莱布尼茨发表了一篇关于二进制表示及算术运算的论文
  - 英国数学家乔治·布尔在1854年给出了二进制的逻辑运算，布尔代数即由此得名
  - 这些工作奠定了现代计算机工作的数学基础

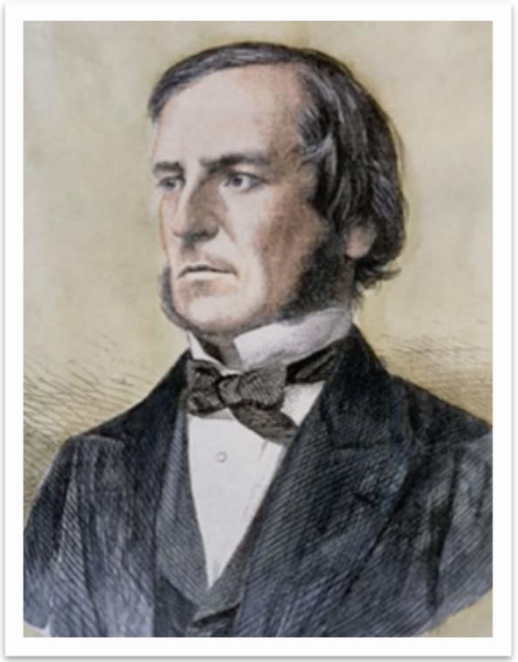
逻辑运算

- 又称布尔运算，用数学方法研究逻辑问题，等式表示判断，推理表示等式的变换
  - 布尔代数：变换的有效性依赖于符号的组合规律
  - 20世纪30年代，逻辑运算在电路系统上获得应用
  - 复杂电子计算机系统遵从布尔代数的变换规律
- 逻辑表达式：用逻辑运算符将关系表达式或逻辑量连接起来的有意义的式子
  - 运算结果是一个逻辑值，即“true”或“false”
  - C语言编译系统逻辑运算结果对应二进制的1和0
- 运算类型
  - 按位运算 “&” “|” “~” “^”
  - 移位运算 “<< ” “>> ”
  - 逻辑运算 “&&” “||” “!”

二进制逻辑运算

布尔代数

- 英国数学家George Boole
- 逻辑值True（真）和False(假）
- 编码为二进制位的1和0
- 逻辑推理的基本原则
  - 实现逻辑表达式
  - 描述逻辑函数
  - 表示逻辑运算



布尔代数运算符

- 数值取值非0即1
- 三种典型的运算符，分别表示与、或、非函数：
  - “与(AND)”运算符使用符号“•”表示
    - A•B，即A AND B，当且仅当二者取值都为1时，结果才为1
  - “或(OR)”运算符使用符号“+”表示
    - A+B，即A OR B，当其中至少有一个变量取值为1时，结果为1
  - “非(NOT)”运算符使用符号“¯”表示
    - $\overline{A}$，即NOT A，当A取值为1时，结果为0；当A取值为0时，结果为1

真值表

- 表示逻辑运算的简便方法
- *n*+1列
  - 前*n*列对应*n*个源操作数
  - 最后一列表示每种组合的输出
- 2^n行
  - 每个源操作数0或1，源操作数组合有2^n种可能
  - 每组值组合（输入组合）：真值表中的一行

A	B	AND/OR/NOT
0	0	
0	1	
1	0	
1	1	

与函数（AND）

- A•B，二元函数，需要两个源操作数
- 如果两个操作数中有一个为0，那么与函数的输出就为0
- 当且仅当两个操作数都为1时，与函数的输出为1
- 真值表2列源操作数，4种输入组合，输出为0001

A	B	**AND**
0	0	**0**
0	1	**0**
1	0	**0**
1	1	**1**

或函数（OR）

- A+B，二元函数，需要两个操作数
- 如果两个操作数中有一个为1，那么或函数的输出就为1
- 当且仅当两个操作数都为0时，或函数的输出为0
- 真值表2列源操作数，4种输入组合，输出为0111

A	B	**AND**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**1**

非函数（NOT）

- $\overline{A}$，一元函数，只作用于一个操作数
- 结果通过对输入进行补运算，即取反操作得到，也被称作补运算
- 当输入为1 时，输出为0；输入为0时，结果为1

A	NOT
0	**1**
1	**0**

逻辑完备性

- 其他任何逻辑函数都可以写成这三种基本逻辑运算符的逻辑组合
  - 异或函数，可以写成：$(\overline{A} \bullet B) + (A \bullet \overline{B})$
  - 可以利用该逻辑组合的真值表来证明

异或函数（XOR）

- A XOR B，二元函数，需要两个源操作数
- 若两个源操作数不同则异或运算输出为1，否则为0，即“相异为1，相同为0”

A	B	**XOR**
0	0	**0**
0	1	**1**
1	0	**1**
1	1	**0**

按位（逻辑）运算

- 对两个*m*位的位组合对应位上的数字按位运算
- 位组合的编号规则
  - 自右向左，顺序编号，最右边的一位是[0]，最左边是[*m*-1]
  - 32位组合A
  - 0001 0010 0011 0100 0101 0110 0111 1000
    - [31]位是0，[30]位是0，[29]位是0，[28]位是1
    - 记为A[31：0]

按位与运算

- 对两个*m*位的位组合对应位上的数字按位做与运算
  - *a*，*b*是8位的位组合，*c*是*a*和*b*做与运算的结果
  - *a*: 00111010
  - *b*: 11110000
  - *c*: 00110000
  - *c*的左边4位与*a*的左边4位相同，而*c*的右边4位均为0
- 与0做与运算结果为0，与1做与运算结果保持不变

按位与运算

- 假设有一个8位的位组合，称为*A*，最右边的两位有特殊的重要性。根据存储在*A*中的最右边两位数，要求计算机处理4个任务之一。如何把这两位孤立出来？
  - 位屏蔽为00000011
  - 和*A*做与运算，从7位到2位的位置上都为0，而0位和1位中的原来的值还在0和1的位置上
  - 屏蔽了7位到2位上的值，孤立出有重要性的0位和1位
  - 结果将是四种组合之一：00000000，00000001，00000010或00000011

按位或运算

- 对两个*m*位的位组合按位进行或运算，也被称为包含或运算（inclusive-OR）
  - *a*，*b*是8位的位组合，*c*是*a*和*b*做或运算的结果
  - *a*: 00111010
  - *b*: 11110000
  - *c*: 11111010
  - *c*的左边4位均为1，右边4位与*a*的右边4位相同
- 与1做或运算结果为1，与0做或运算结果保持不变

按位非运算

- 对一个*m*位的位组合按位进行非运算
  - *c*是对*a*进行非运算的结果
  - *a*: 00111010
  - *c*: 11000101

按位异或运算

- 对*m*位的位组合做异或运算
- 假如*a*和*b*是8位的位组合，*c*是*a*和*b*的异或运算：
  - *a*: 00111010
  - *b*: 11110000
  - *c*: 11001010
  - *c*的左边4位为*a*的左边4位按位取反的结果，而*c*的右边4位则与*a*的右边4位相同
- 与1做异或运算结果表示取反，与0做异或运算结果保持不变

按位取反

- 假如*a*是8位的位组合，*b*是11111111，*c*是*a*和*b*的异或运算：
  - *a*: 00111010
  - *b*: 11111111
  - *c*: 11000101
- c即为a取反的结果

C的位运算符

- C语言的低级语言特征之一
- 对数值进行位（布尔）运算的运算符集合
  - “&”：按位“与”运算
  - “|”：按位“或”运算
  - “~”：按位“非”
  - “^”：按位“异或”
  - “<<”：执行左移
  - “>>”：执行右移

二进制逻辑运算

示例

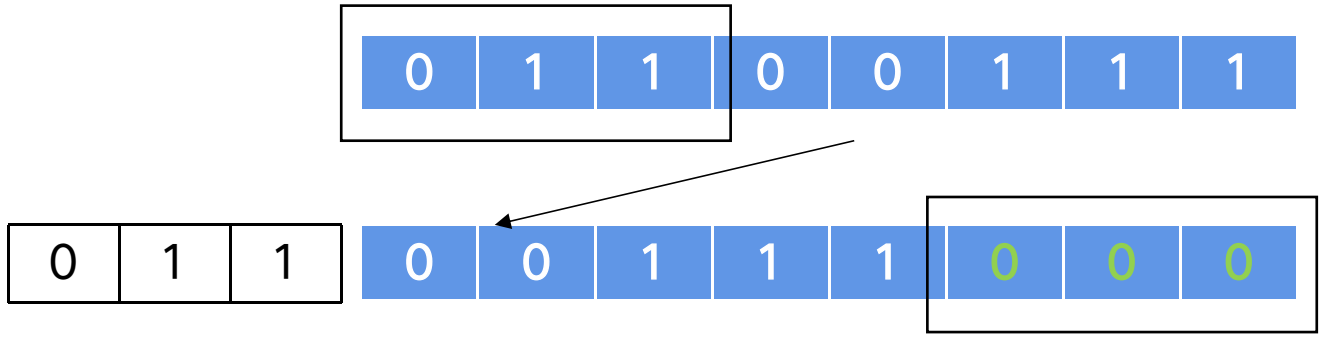
位运算表达式	二进制	位运算结果	十六进制
~0x41	~[0100 0001]	[1011 1110]	0xBE
~0x00	~[0000 0000]	[1111 1111]	0xFF
0x69 & 0x55	[0110 1001]&[0101 0101]	[0100 0001]	0x41
0x69 | 0x55	[0110 1001]|[0101 0101]	[0111 1101]	0x7D

移位运算符

- “<<”执行左移
- “>>”执行右移
- 第一个操作数是被移位的数值
- 第二个操作数是移动的位数
- 左移，零填充
- 右移，符号扩展

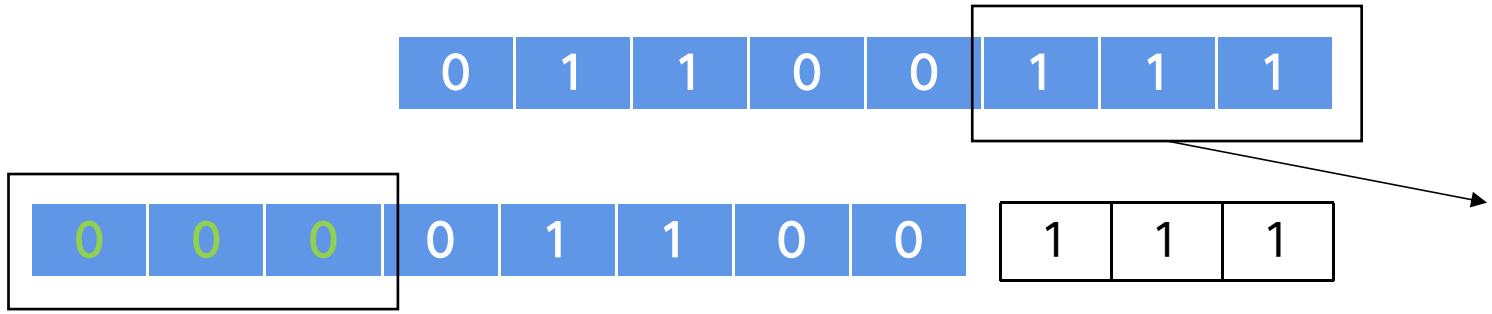
示例

- 8位整数：01100111 << 3 (算数/逻辑左移)



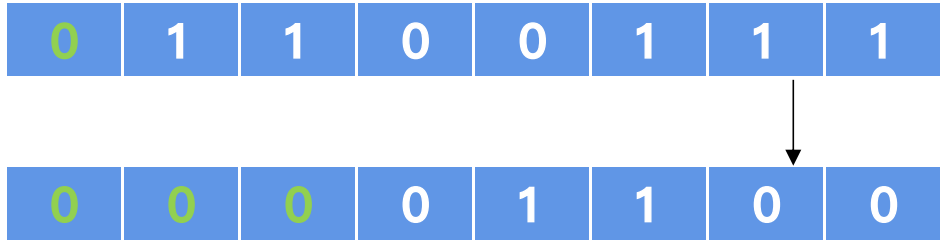
示例

- 8位整数：01100111 >> 3（逻辑右移）

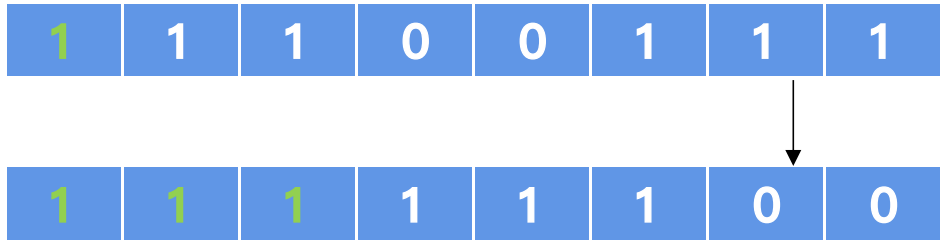


示例

- 8位整数：01100111 >> 3（算数右移）



- 8位整数：11100111 >> 3（算数右移）



二进制逻辑运算

示例

- 0x1234 << 3
- 0x1234 >> 2
- 1234 << 3
- 1234 >> 2
- 0x1234 << 5
- 0xFEDC >> 3

➢ /*等于0x91A0*/

➢ /*等于0x048D*/

➢ /*等于9872*/

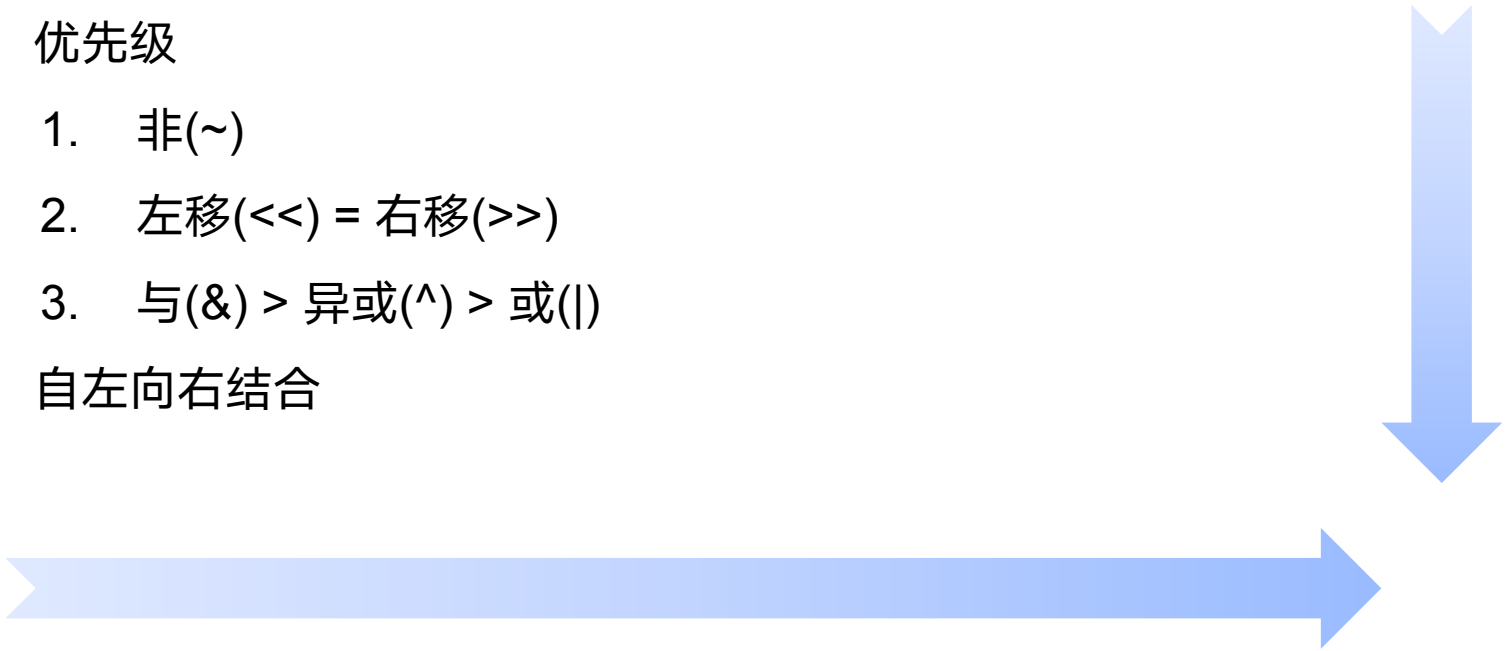
➢ /*等于308*/

➢ /*等于0x4680，结果仍是16位*/

➢ /*等于0xFFDB，需进行符号扩展*/

优先级及结合性

- 优先级
  1. 非(~)
  2. 左移(<<) = 右移(>>)
  3. 与(&) > 异或(^) > 或(|)
- 自左向右结合



优先级及结合性

- x = ~a | ~b;　　/*如果a=3，b=4*/
  - x = ~(0011) | ~(0100)
  - x = ~(0011) | ~(0100) = (1100) | ~(0100)
  - x = (1100) | ~(0100) = (1100) | (1011)
  - x = (1100) | (1011) = (1111)

- C语言的按位运算符，操作数都不能是浮点数，语句中的x、a、b都是整数。
- x = (1111) = -1，这里x是有符号整数

C的逻辑运算符

- 与位运算不同
- 所有非零的参数都为True，只有参数0为False
  - 位运算只有在特殊的数值条件下才得到0或1
- 逻辑运算的运算符集合
  - “&&”：“与”（AND）
  - “||”：“或”(OR)
  - “！”：“非”(NOT)

二进制逻辑运算

示例

逻辑运算表达式	结果
!0x41	0x00
!0x00	0x01
!!0x41	0x01
0x69 && 0x55	0x01
0x69 || 0x55	0x01
if(a && 5/a)	若a=0，则结果为0x00


南京大学120周年校庆
120th ANNIVERSARY
NANJING UNIVERSITY
1902 – 2022

02 算术运算

算数运算

- 用于数值计算
- 无符号/有符号（补码）整数的加减乘除运算
- 定点/浮点数加减乘除运算
- 变量与常数间的乘除运算

补码整数加法运算

- $[X+Y]_{补} = [X]_{补} + [Y]_{补}$
- 例：已知 X=+10010 Y= -10101（此处为原码表示） 求X+Y
  - $[X]_{补}$=010010 $[Y]_{补}$=101011
  - $[X+Y]_{补}=[X]_{补}+[Y]_{补}$=010010+101011=111101
  - 故X+Y= - 00011

补码整数减法运算

- $[X-Y]_{补} = [X]_{补} - [Y]_{补} = [X]_{补} + [-Y]_{补}$
  - $[-Y]_{补} = [[Y]_{补}]_{补}$
  - $[Y]_{补}$按位取反加1
- 例：已知 $[Y]_{补}$ =10011 求 $[-Y]_{补}$
  - $[Y]_{补}$ =10011　Y=－1101　-Y= 1101
  - $[-Y]_{补}$ = 01101　对比 $[Y]_{补}$ =10011
- 例：已知 X=+10101 Y=+10010 求X-Y
  - $[X]_{补}$=010101，$[Y]_{补}$=010010，$[-Y]_{补}$=101110
  - $[X-Y]_{补}=[X]_{补}+[-Y]_{补}$=010101+101110=1 000011
  - X－Y= + 00011

补码整数加减运算

X+X

- 与十进制运算十分相似
- 把一个数x加上它自身
- 每位上的数字都向左移了一位，乘以2
  - 61可以表示为 $0\times2^6+ 1\times2^5+ 1\times2^4+ 1\times2^3+ 1\times2^2+0\times2^1+ 1\times2^0$
  - $61+61 = 2\times61$，可以表示为 $2\times(0\times2^6+ 1\times2^5+ 1\times2^4+ 1\times2^3+ 1\times2^2+0\times2^1+ 1\times2^0)$
  - 也就是 $0\times2^7+ 1\times2^6+ 1\times2^5+ 1\times2^4+ 1\times2^3+0\times2^2+ 1\times2^1$

恰当的位数

- 为减少占用空间，会采用恰当的位数来表示数值
  - 6
    - 用4位（0110）
    - 用16位（0000 0000 0000 0110）
  - -6
    - 用4位（1010）
    - 用16位（1111 1111 1111 1010）

不同长度数值做加法

- 为了对两个具有不同长度的数值做加法，首先必须将它们表示为相同的长度

```
  0000000000001110
+             1100
------------------
              ?
```

补码整数加减运算

符号扩展

- 如果用0来扩展一个正数的左端，它的值不会改变
- 如果用1来扩展一个负数的左端，其值亦不会改变
- 在这两种情况中扩展的都是符号位，这种运算被称为符号扩展（Sign-EXTension，SEXT）
- 用于对不同长度的数值之间的运算

```
  0000 0000 0000 1110（14）
+ 0000 0000 0000 1100（12）
-------------------------
  0000 0000 0001 1010（26）
```

```
  0000 0000 0000 1110（14）
+ 1111 1111 1111 1100（-4）
-------------------------
  0000 0000 0000 1010（10）
```

补码整数加减运算

溢出

- 使用4位补码数据类型，计算2+6：

```
  0010
+ 0110
------
  1000
```

- 计算结果为-8，为什么会出现错误？
  - 4位表示的数值范围[-8,+7]
  - 2+6=8，大于+7，即大于0111
  - 0111是使用4位的补码数据能够表示的最大正数
  - 因此8不能用4位的补码表示出来

溢出的概念：
运算结果超出某种数据类型的表示范围。

补码整数加减运算

溢出

```
    0011   (3)
+   0110   (6)
    1001   (-7)
```

两个正数之和为负数!

```
    1101   (-3)
+   1010   (-6)
    0111   (7)
```

两个负数之和为正数!

补码整数加减运算

检测溢出

- 溢出只可能发生在同符号数相加时，包括$[X]_{补}$与$[Y]_{补}$； $[X]_{补}$ 与$[-Y]_{补}$同号；
- **方法1**：对操作数和运算结果的符号位进行检测，符号不相同时就表明发生了溢出
  - 设X0 ，Y0 为参加运算数的符号位， S0 为结果的符号位
  - $V = X_0Y_0\overline{S_0} + \overline{X_0}\,\overline{Y_0}\,S_0$ ，当V=1时，运算结果溢出
- **方法2**：对最高有效进位和符号进位进行检测，相异则发生溢出
  - 设运算时最高有效数据位产生的进位为C1，符号位产生的进位为C0
  - 溢出检测电路为： V= C0 ⊕ C1，当V=1时，运算结果溢出
  - 0.X1+0.Y1，C0=0，若C1=1；1.X1+1.Y1，C0=1，若C1=0，都改变了符号位，发生溢出
- **方法3**：用变型补码
  - $[X]_{补}= X_{f1}X_{f2}\,.\,X_1X_2\cdots\cdots X_n$ ，溢出判断：$V= X_{f1} \oplus X_{f2}$ ，当V=1时，运算结果溢出
  - X = - 10010 Y= - 10101 X+Y = 11 01110 + 11 01011 = 1 10 11001
  - V= 1 ⊕ 0 = 1 故发生溢出

检测溢出

- 溢出只可能发生在同符号数相加时，包括$[X]_{补}$与$[Y]_{补}$； $[X]_{补}$ 与$[-Y]_{补}$同号；
- **方法4**：溢出判断的软件方法

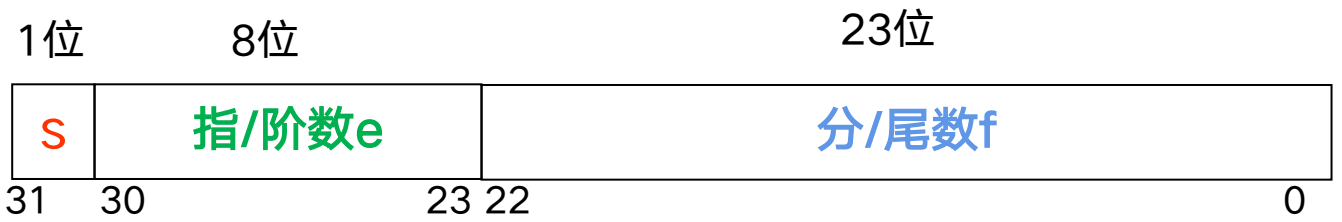
```
int tadd_ok(int x,int y) {
    int sum=x+y;
    int neg_over=x<0&&y<0&&sum>=0;
    int pos_over=x>=0&&y>=0&&sum<0;
    return !neg_over&&!pos_over;
}
```

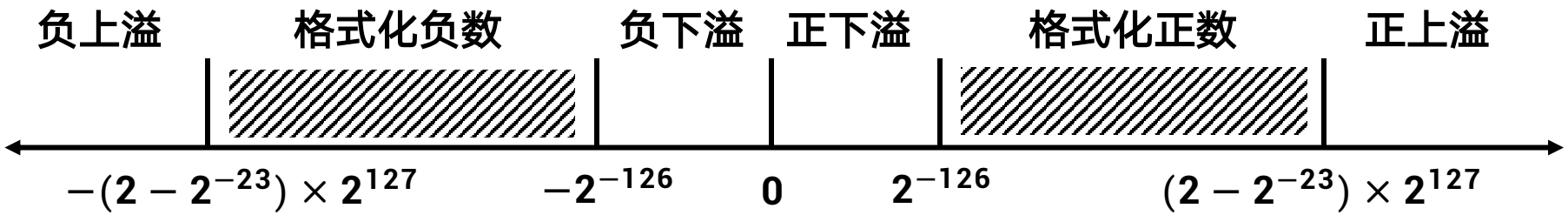
体会软/硬件功能的等效性和差异性！软/硬协同的系统观！

浮点数加减运算

IEEE-754浮点数表示

$$V = (-1)^S \times M \times 2^E$$





阶码的值	尾数的值	表示
0（全0）	0	+/- 0
0（全0）	非0	非规格化数
1~254	任意	规格化数
255（全1）	0	+/- ∞
255（全1）	非0	NaN

$$V_{\text{float}} = \begin{cases} (-1)^s \times 1.f \times 2^{e-127}, & 1 \le e \le 254 \\ (-1)^s \times 0.f \times 2^{-126}, & e = 0 \\ (-1)^s \times 1.f \times 2^{128} = Infinity(+and-), & e = 255, f = 0 \\ NaN(Not\ a\ Number), & e = 255, f \neq 0 \end{cases}$$

IEEE754浮点数标准规定的五种异常

- **无效运算（结果为NaN）**
  - 运算时有一个数是非有限数，如：加/减 ±∞，±∞/±∞，0×±∞，等
  - 结果无效，如：0/0，源操作数为NaN，一个数对0取余等
- **有限数除以0（结果为±∞）**
- **数太大**（阶上溢，**结果为±∞**）：如对于单精度，阶码 > 127
- **数太小**（阶下溢，**结果用非规格化数表示**）：如对于单精度，阶码 < -126
  - 注：IEEE754出现前阶下溢一般为0，换言之，IEEE754解决了这一问题
- **结果不精确**（舍入时引起）：如1/3不能精确表示为一个浮点数

浮点数加减运算方法及步骤

$$X+Y=(X_S\times B^{X_E-Y_E}+Y_S)\times B^{Y_E}$$
$$X-Y=(X_S\times B^{X_E-Y_E}-Y_S)\times B^{Y_E}$$
$$X_E\le Y_E$$

① 对阶：小阶向大阶看齐（右移阶码小的浮点数的尾数并同步增加其阶码），直至两数阶码相等

- $X_S\times B^{X_E}=X_S\times B^{X_E-Y_E}\times B^{Y_E}$，$X_S$尾数右移$(Y_E-X_E)$位，阶数增加$(Y_E-X_E)$位

② 尾数运算：对阶后的尾数做加（减）法运算

- 符号位+尾数的二进制数补码运算（隐藏位参与运算）
- 同号数相加，结果符号不同表示溢出；异号数相加不会溢出
- 求和（差）：
  - $[x+y]_{补}=[x]_{补}+[y]_{补}$（正数三码合一）
  - $[x-y]_{补}=[x]_{补}+[-y]_{补}$（$[-y]_{补}=[y]_{补}$按位取反加1）

浮点数加减运算

③ 结果规格化：对运算结果进行规格化处理

- 左规：0.0…01b…bb(k-1个0)则尾数左移k位， $Y_E = Y_E - k$
- 右规：11.bb…bb则尾数右移1位， $Y_E = Y_E + 1$

④ 舍入操作：右移规格化时可能丢失一些低位的数值位，为提高精度，可采取舍入的方法

•0 舍 1 入：若右移出的是1则在最低位加1

•恒置 1：只要数字位1被移掉,就将最后一位恒置成1

⑤ 溢出处理：浮点数溢出的标志，指数/阶码是否溢出

•阶码上溢：阶码的符号位为01，超出最大能表示的指数范围（单精度127，双1023)

•阶码下溢：阶码的符号位为10，超出最小能表示的指数范围（单精度-126，双-1022）

示例：2+6

- 转成二进制规格化小数
  - [X]补 =2=10=1.0×2^1 = 0 10000000 0000 0000 0000 0000 0000 000
  - [Y]补 =6=110=1.10×2^2 = 0 10000001 1000 0000 0000 0000 0000 000

① 对阶

- [ΔE]补 =[Ex]补 + [- Ey]补 = 00 10000000 + 11 0111 1111 =11 1111 1111 = $(-1)_{10}$
- X的阶码小于Y的阶码
- 将X的尾数向右移动1位，同时阶码加1，对阶后的X为：
- [X]补 = 0.10 ×2^2= 0 10000001 1000 0000 0000 0000 0000 000

示例：2+6

② 尾数加法运算：

- 2：00. 100 0000 0000 0000 0000 0000
- 6：01. 100 0000 0000 0000 0000 0000
- 尾数相加得到： 10. 00 000 0000 0000 0000 0000 0000（隐藏位进位）

③ 尾数规格化处理

- 右规（尾数右移，小数点左移）1位，阶码加1变为3（e=130），S=0

④ 舍入（0舍1入）

- 尾数最低位的0舍去

⑤ 溢出处理

- 阶码00 10000001 + 00 0000 0001 = 00 1000 0010无溢出

0 10000010 0000 0000 0000 0000 0000 000

浮点数加减运算

示例：8.25-1.25

- 转成二进制规格化小数
  - [X]补 =8.25=1000.01=1.00001×2^3 = 0 10000010 0000 1000 0000 0000 0000 000
  - [Y]补 =1.25=1.01=1.01×2^0 = 0 01111111 0100 0000 0000 0000 0000 000

① 对阶

- [ΔE]补 =[Ex]补 + [- Ey]补 = 00 10000010 + 11 10000001 =00 00000011 = $(3)_{10}$
- X的阶码大于Y 的阶码
- 将Y的尾数向右移动3位，同时阶码加3，对阶后的Y为：
- [Y]补 = 0.00101×2^3 = 0 10000010 0010 1000 0000 0000 0000 000

浮点数加减运算

示例：8.25-1.25

② 尾数加法运算：

- 8.25： 01. 0000 1000 0000 0000 0000 000
- 1.25： 00. 0010 1000 0000 0000 0000 000
- -1.25：11. 1101 1000 0000 0000 0000 000

- 8.25和-1.25尾数相加得到：00. 1110 0000 0000 0000 0000 0000 000（隐藏位借位）

③ 尾数规格化处理

- 左规（尾数左移，小数点右移）1位，阶码减1变为2（e=129），S=0

④ 舍入（0舍1入）

- 最低位补0

⑤ 溢出处理

- 阶码129无溢出

0 10000001 1100 0000 0000 0000 0000 000

浮点数运算

- $(3.14 + 1e^{10}) - 1e^{10} = 0.0$
- $3.14 + (1e^{10} - 1e^{10}) = 3.14$

加法不具备结合性

- $(1e^{20} \times 1e^{20}) \times 1e^{-20} = +\infty$
- $1e^{20} \times (1e^{20} \times 1e^{-20}) = 1e^{20}$

乘法不具备结合性

- $1e^{20} \times (1e^{20} - 1e^{20}) = 0.0$
- $(1e^{20} \times 1e^{20}) - (1e^{20} \times 1e^{20}) = NaN$

乘法在加法上不具备结合性

习题

- **书面作业**
  - 6.9
  - 6.16

谢　　谢

诚耀百廿 雄创一流