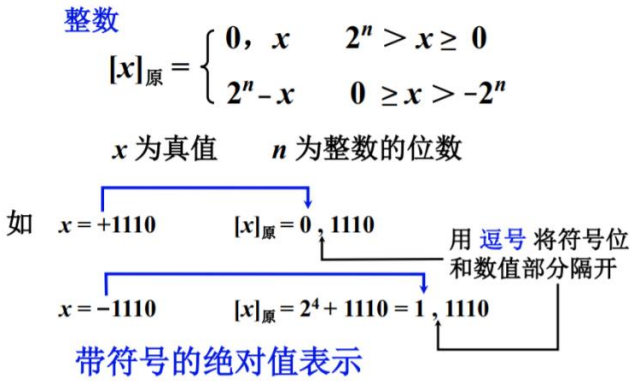
第二章常用知识：

Part01：整数

1.有符号数的表示（原码、补码和移码）：

（1）整数原码的表示（补充知识）



（2）原码的特点（补充知识）

但是用原码作加法时，会出现如下问题：

要求	数1	数2	实际操作	结果符号
加法	正	正	加	正
加法	正	负	减	可正可负
加法	负	正	减	可正可负
加法	负	负	加	负

能否 只作加法？

找到一个与负数等价的正数 来代替这个负数

就可使 减 ⟶ 加

（3）引入补码的意义：可以将减法转换为加法，这样计算机系统内的加减计算可以统一到加法。

（4）求补码的快捷方式：

当这个数为正数时，补码和原码一样。

当这个数为负数时，补码可用原码除符号位外每位取反，末位加一得到。

Eg：原码 1001,0101➔补码：1110,1011

（5）用移码表示浮点数阶码的意义：由于移码的二进制表示的值是随着真值递增的，所以能够方便地判断浮点数阶码的大小（只需从高位开始按位比较每一位），从而使得浮点数的比较可以通过整数的比较来实现。

（6）真值、补码、移码对照表：

真值 x (n=5)	$[x]_{补}$	$[x]_{移}$	$[x]_{移}$对应的十进制整数
-100000	100000	000000	0
- 11111	100001	000001	1
- 11110	100010	000010	2
⋮	⋮	⋮	⋮
- 00001	111111	011111	31
± 00000	000000	100000	32
+ 00001	000001	100001	33
+ 00010	000010	100010	34
⋮	⋮	⋮	⋮
+ 11110	011110	111110	62
+ 11111	011111	111111	63

注意：补码的二进制表示的值是不随着真值的递增而递增的，但是移码是的，所以补码用于计算，移码用于比较。

2.有符号数与无符号数的表示范围（以 32 位的数为例）：

无符号数：Umax =0，Umax = 2^{32}-1

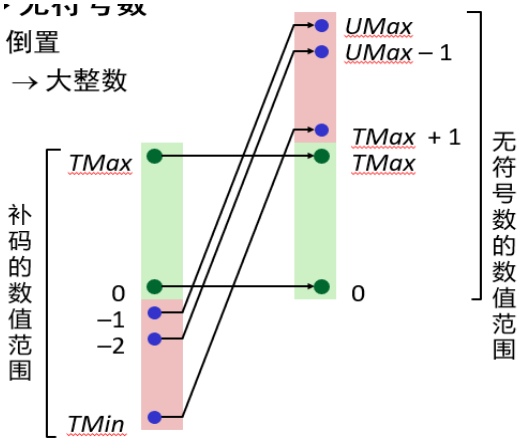
有符号数（补码表示）：Tmax=-2^{31}，Tmax= 2^{31}-1

3.有符号数和无符号数转换的基本原则：

（1）位模式不变

（2）重新解读（按目标编码的规则解读）

（3）负的有符号转为无符号数会加 2^w,大于 Tmax 的无符号数转为有符号数会减 2^w



注意：当表达式含有无符号数和有符号数时，有符号数会隐式转换为无符号数。

4.整数的加法、乘法、移位，以及溢出

（1）w 位无符号数加法溢出判断

运算结果超过 w 位则溢出，对结果取 mod2^w，即只保留低 w 位。

（2）w 位有符号数加法和溢出判断

有符号数加法和无符号数加法有完全相同的位级表现。

溢出判断：两个正数相加结果为负数则溢出，或者两个负数相加结果为整数则溢出，其他情况皆没有发生溢出。

$$TAdd(x,y)=\begin{cases} x+y-2^w, & TMax_w < x+y & \text{正溢出} \\ x+y, & TMin_w \le x+y \le TMax_w & \text{正常} \\ x+y+2^w, & x+y < TMin_w & \text{负溢出} \end{cases}$$

（3）w 位无符号数乘法

结果最多可能为 2w 位，对结果取 mod2^w，即只保留低 w 位。

（4）w 位有符号数乘法

结果最多可能为 2w 位，对结果取 mod2^w，即只保留低 w 位。

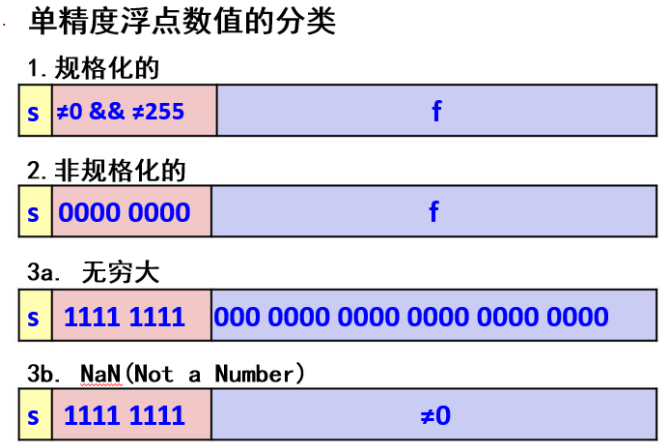
与有符号数乘法不同的是，会对高位进行符号扩展，然后在截断保留低 w 位。

5.关于除以 2 的幂以及舍入的问题

当被除数位正数时是向另舍入的，当被除数位负数时，会向下舍入，为了达到不论正负数都是向 0 舍入的要求，会在除之前加上一个偏置（这个偏置为除数-1）。

Part02：浮点数

1.规格化数和非规格化数，无穷的表示以及 NaN

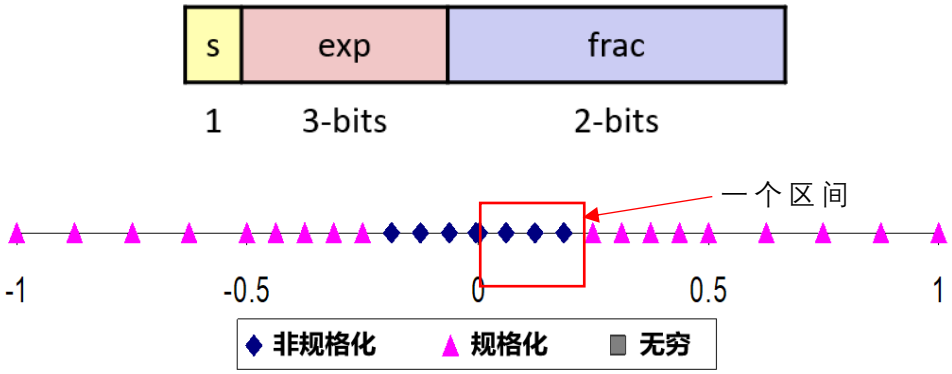


关于 c 中出现和 NaN 或 inf 比较的测试：

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(){
    if(NAN){
        printf("yes1\n");
    }
    if(NAN>1.0||NAN<=1.0){
        printf("yes2\n");
    }
    if(INT_MAX<INFINITY){
        printf("yes3\n");
    }
    if(INT_MIN>-INFINITY){
        printf("yes4");
    }
    return 0;
}
```

```
[Running]
yes1
yes3
yes4
[Done] exi
```

2.浮点数在数轴上的分布以及当阶码为零时，为什么规定 E=1-Bias



可以看到由于 frac 是 2 位，因此每个区间是被 4 个点等分的，且可以发现每个区间都是前面所有区间（到 0 为止）长度之和，因此越往后点之间的间隔是越大的。

同时可以发现非规格化数和第一个区间的规格化数里面的点是均匀分布的，这得益当阶码为零时，规定 E=1-Bias，你们可以试一试，当 E=0-Bias 时，点是否是均匀分布的。

3.一定要熟练地进行实数与浮点数之间的转换!

4.PPT43 页思考题答案（供参考）

（1）IEEE754 比整数部分 10 位+小数部分 20 位的表示方法有什么优点？缺点呢?

IEEE754 浮点数标准表示数的范围更大，但是精度没有整数部分 10 位+小数部分 20 位的表示方法高，而且表示的数的个数也没有另一种表示方法多（因为+-0 以及 NaN 的存在）。

（2）float 非无穷的最大值和最小值？最大值：$(2-2^{-23})x2^{127}$，最小值：$-(2-2^{-23})x2^{127}$

（3）float 数 1，65536，0.4，-1，0 的内存表示

1：0 01111111 00000000000000000000000 内存表示（小端）：00 00 80 3F

65536：0 10001111 00000000000000000000000 内存表示：00 00 80 47

0.4：0 10000001 10011001100110011001100 内存表示：CC CC CC 40

-1：1 01111111 00000000000000000000000 内存表示：00 00 80 BF

0：1 00000000 00000000000000000000000 内存表示：00 00 00 80

或 0 00000000 00000000000000000000000 内存表示：00 00 00 00

（4）一个数的 Float 形式是惟一的吗（除了 0）？是的

（5）每一个 IEEE754 编码对应的数是唯一的吗？不是的，还有 0 和 NAN

（6）简述 Float 数据的浮点数密度分布：上面第二点说过了

（7）c 语言中除以 0 一定报错溢出吗？整数会报错，浮点数会得无穷大

（8）int 与 float 都占 32 个二进制位，float 与 int 相比谁的数的个数多？各自是多少个？多多少？

肯定是 int 表示的数多，因为+-0 以及 NaN 的存在。Int 可以表示 2^{32} 个数，float 可以表示 $2^{32}-1-2^{24}$ 个数（减 1 是因为 0 有两种表示，需要减去一种，减 2^{24} 是因为 NaN 有这么多种，可以将 inf 看成一个数，因此可以不减）。Int 比 float 多 $1+2^{24}$ 个数。

（9）怎么判断和定义浮点数的无穷大以及 NaN?

NaN：阶码的每个二进制位全为 1，并且尾数不为 0；

无穷：阶码的每个二进制位全为 1，并且尾数为 0；符号位为 0，是正无穷，符号位为 1 是负无穷。

Eg: 在 c 中, 1.0/0 会得到正无穷, 对负数开根号或取对数或 inf-inf 会得到 NaN。

```c
#include <stdio.h>
#include <math.h>
#include <stdlib.h>
int main(){
    if(1.0/0==INFINITY){
        printf("yes\n");
    }
    if(isnan(sqrt(-1))){
        printf("yes1");
    }
    return 0;
}
```

```
[Running] cd "c:\Users\chenq\Desktop\pypc\" && gcc test.
test.c: In function 'main':
test.c:5:11: warning: division by zero [-Wdiv-by-zero]
    if(1.0/0==INFINITY){
          ^
yes
yes1
[Done] exited with code=0 in 0.457 seconds
```