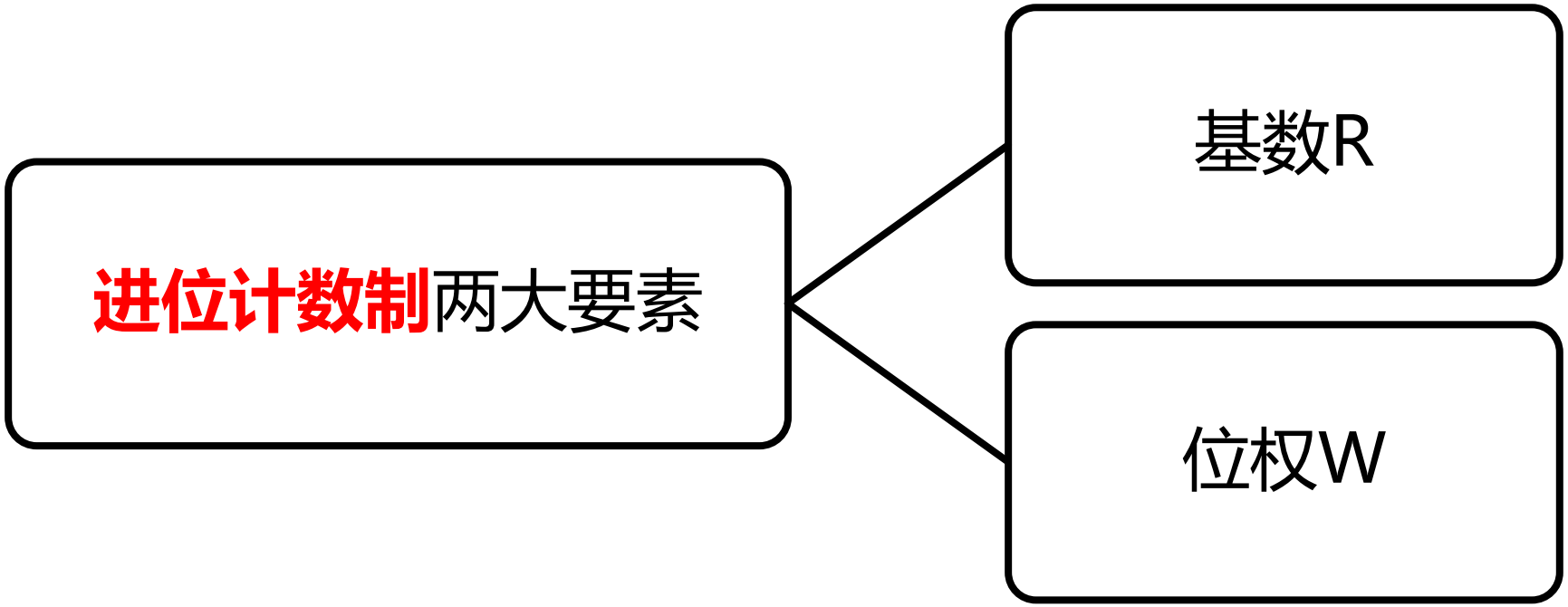
计算机中的数值表示



基数R

- R代表基本数码的**个数,**二进制就是0和1这2个数
- R进制的主要特点是**逢R进1**
- 基数R的数制称为**R进制数**

R	进制	数码符号	进制规则
R=2	二进制数(2)	0、1	逢2进1
R=10	十进制数(10)	0 ~ 9	逢10进1

位权Wi

珠峰架构

- 位权Wi 是指第i位上的数码的**权重值**,位权与数码所处的位置**i**有关
- 例如 R=10(十进制数),各个数码的权为10^i，i表示数码所处的位置
- 个位i=0,位权是$10^0=1$
- 十位i=1,位权是$10^1=10$
- $(22.22)_{10}=2x10^1+2x10^0+2x10^{-1}+2x10^{-2}$

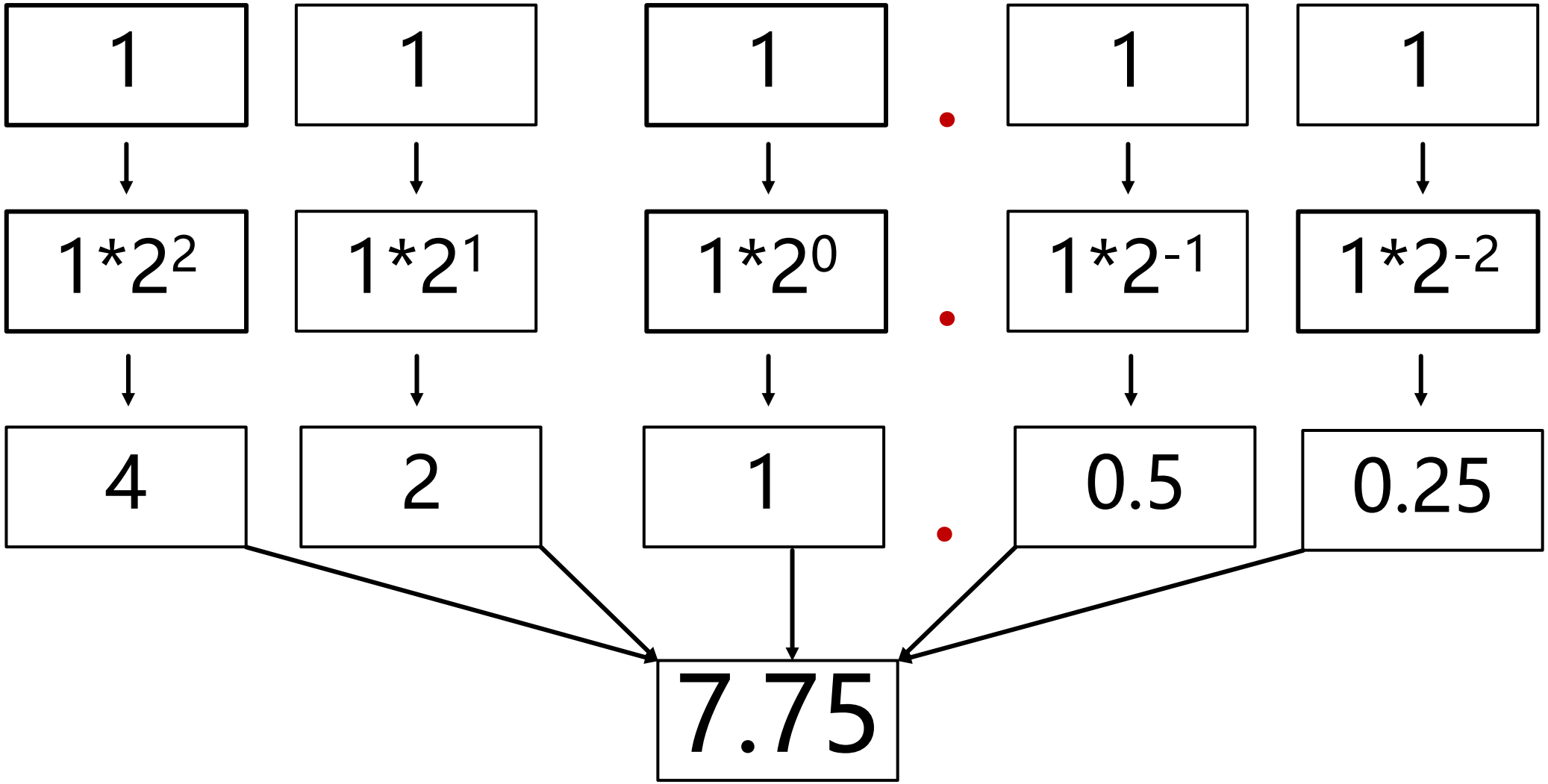
R	进制	数码符号	进制规则	位权 Wi	例子
R=2	二进制数	0、1	逢2进1	2^i	$(11.11)_2$
R=10	十进制数	0 ~ 9	逢10进1	10^i	$(99.99)_{10}$

常用数制对应关系

十进制	二进制
1	0001
2	0010
3	0011
4	0100
5	0110
6	0111
7	1000
8	1000
9	1001
10	1110
11	1011
12	1100
13	1101
14	1110
15	1111

二进制转十进制

- 方法: 按权**展**开,加权求**和,**以$(111.11)_2$为例



十进制转二进制

- **整数部分:** 除2取余,直到商为0,最先得到的余数是最低位,最后得到的余数是最高位.
- **小数部分:** 乘2取整,直到积为0或者达到精度要求为止,最先得到的整数是高位
- 例如 $(7.75)_{10}=(111.11)_2$

余数

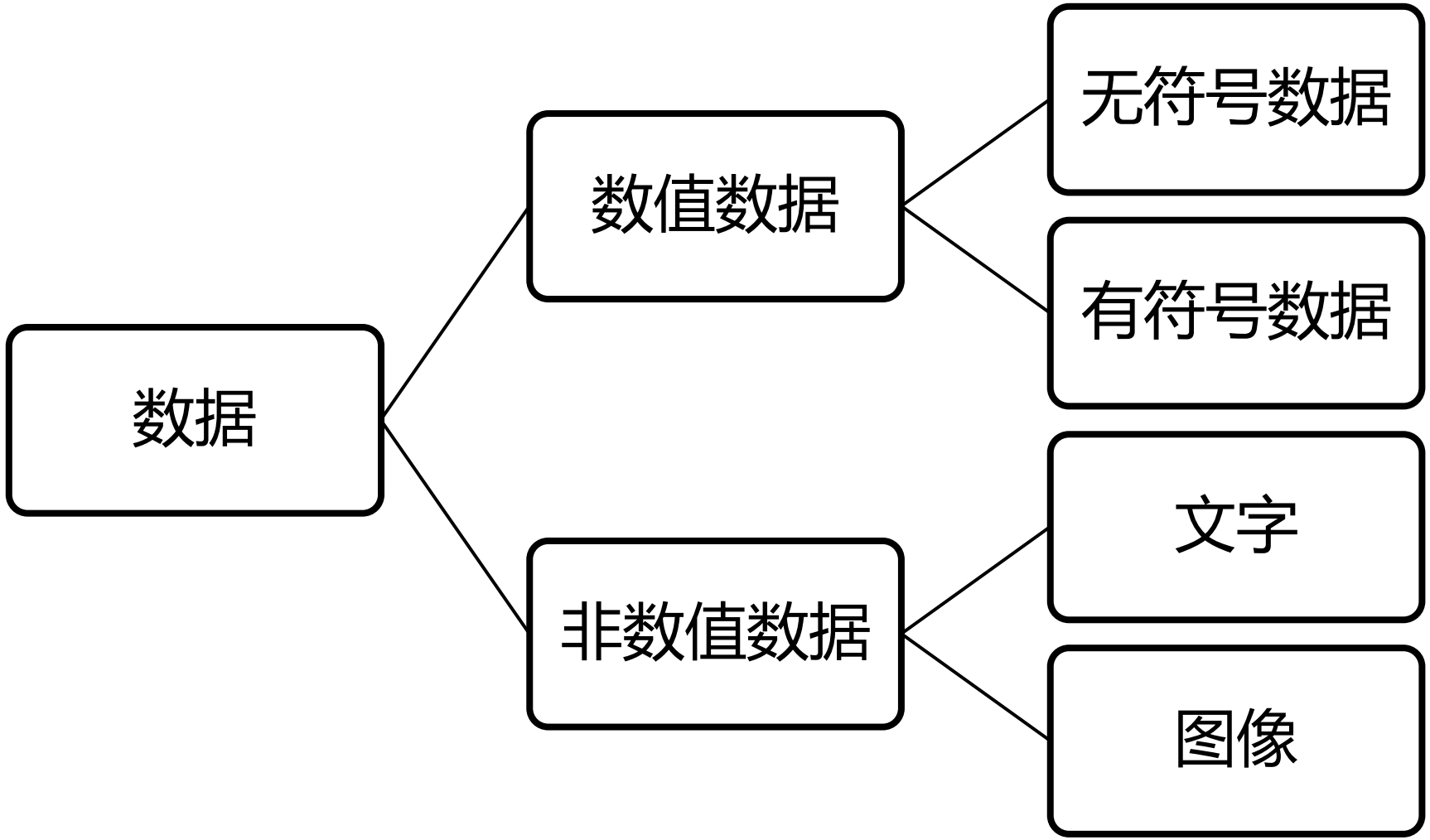
2 | 7	1	(最低位)
2 | 3	1	
2 | 1	1	(最高位)
0		

111

0.75	取整	
x 2		
0.5	1	(最高位)
x 2		
0	1	(最低位)

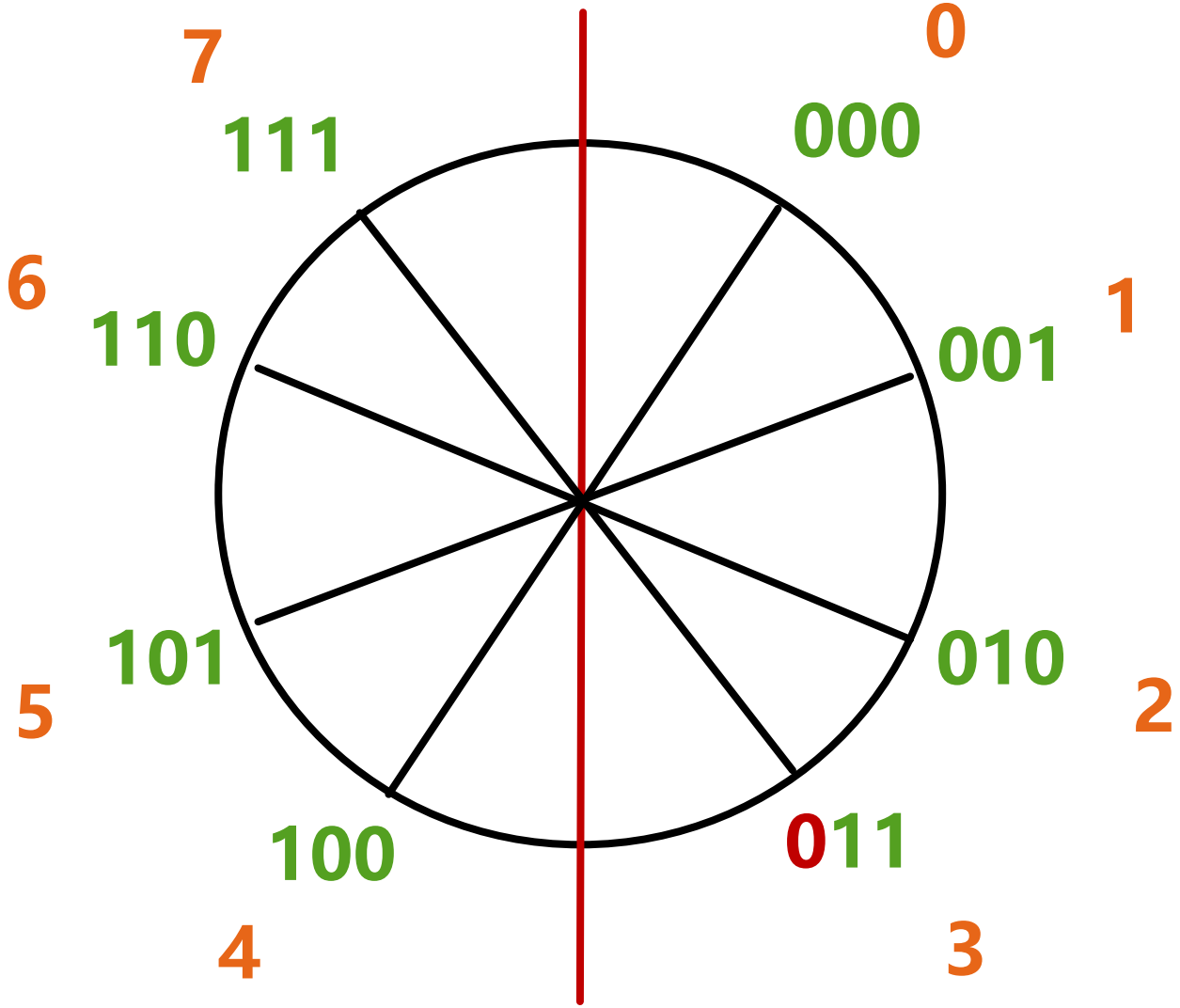
11

计算机中的数据



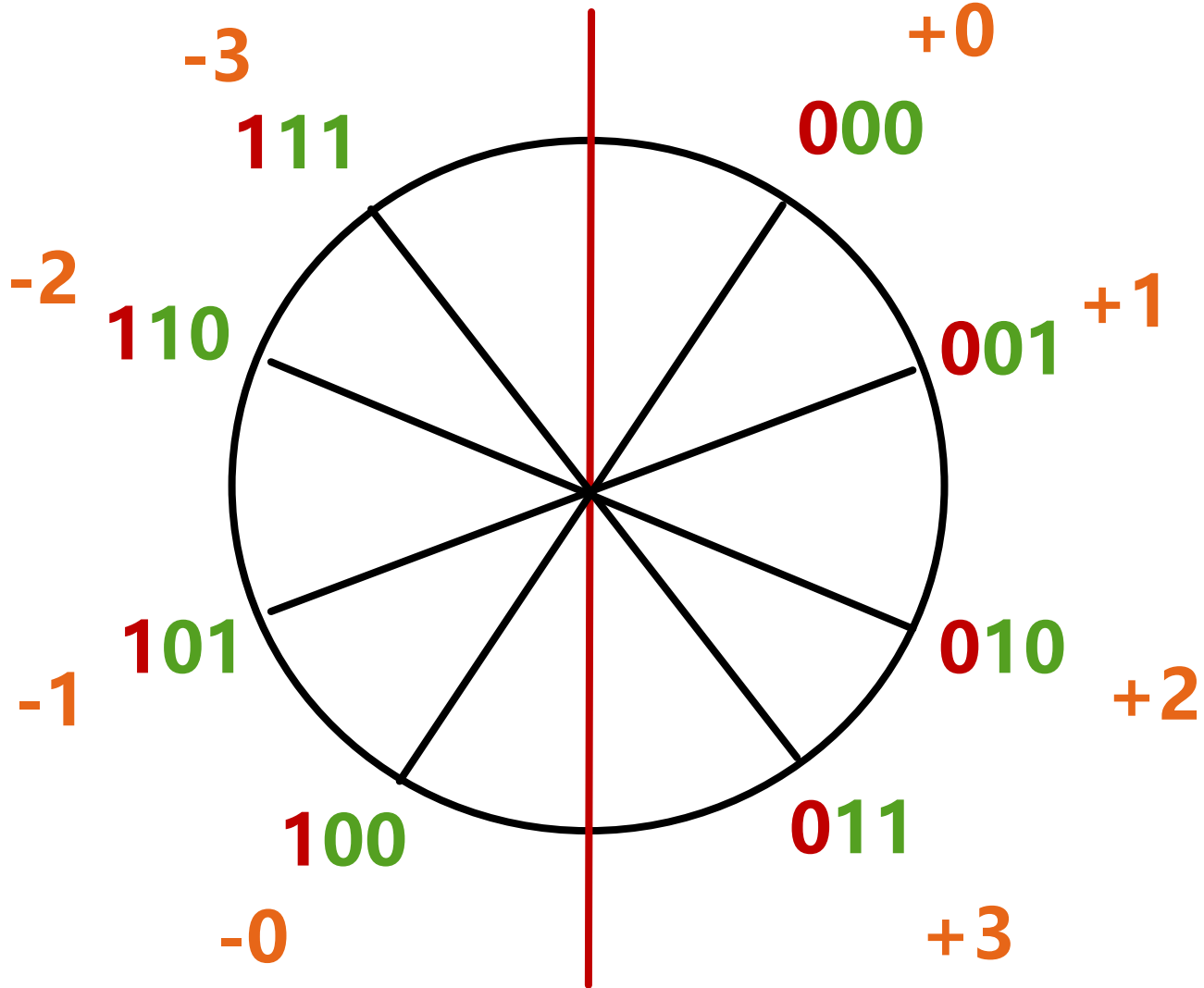
无符号数据的表示

原码: 3个bit能表示8个数 0 1 2 3 4 5 6 7



有符号数据的表示

- 符号: 用0、1表示正负号,放在数值的最高位
- **原码:** 3个bit能表示8个数 +0 +1 +2 +3 -0 -1 -2 -3,4正4负



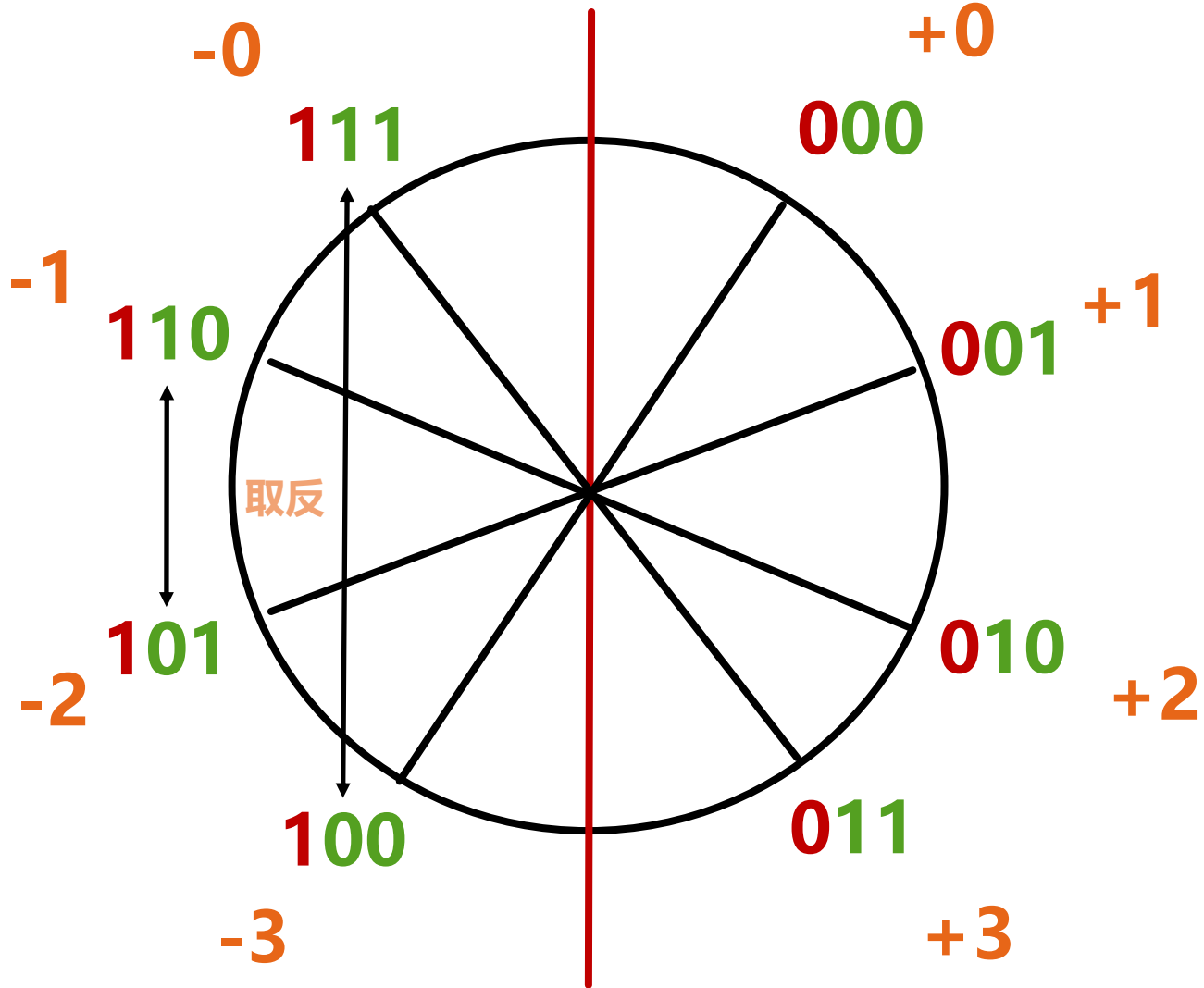
010+101=111

(+2)+(-1)=-3?

是否能用加法表示减法

反码

- **反码:正数不变,负数的除符号位外取反**



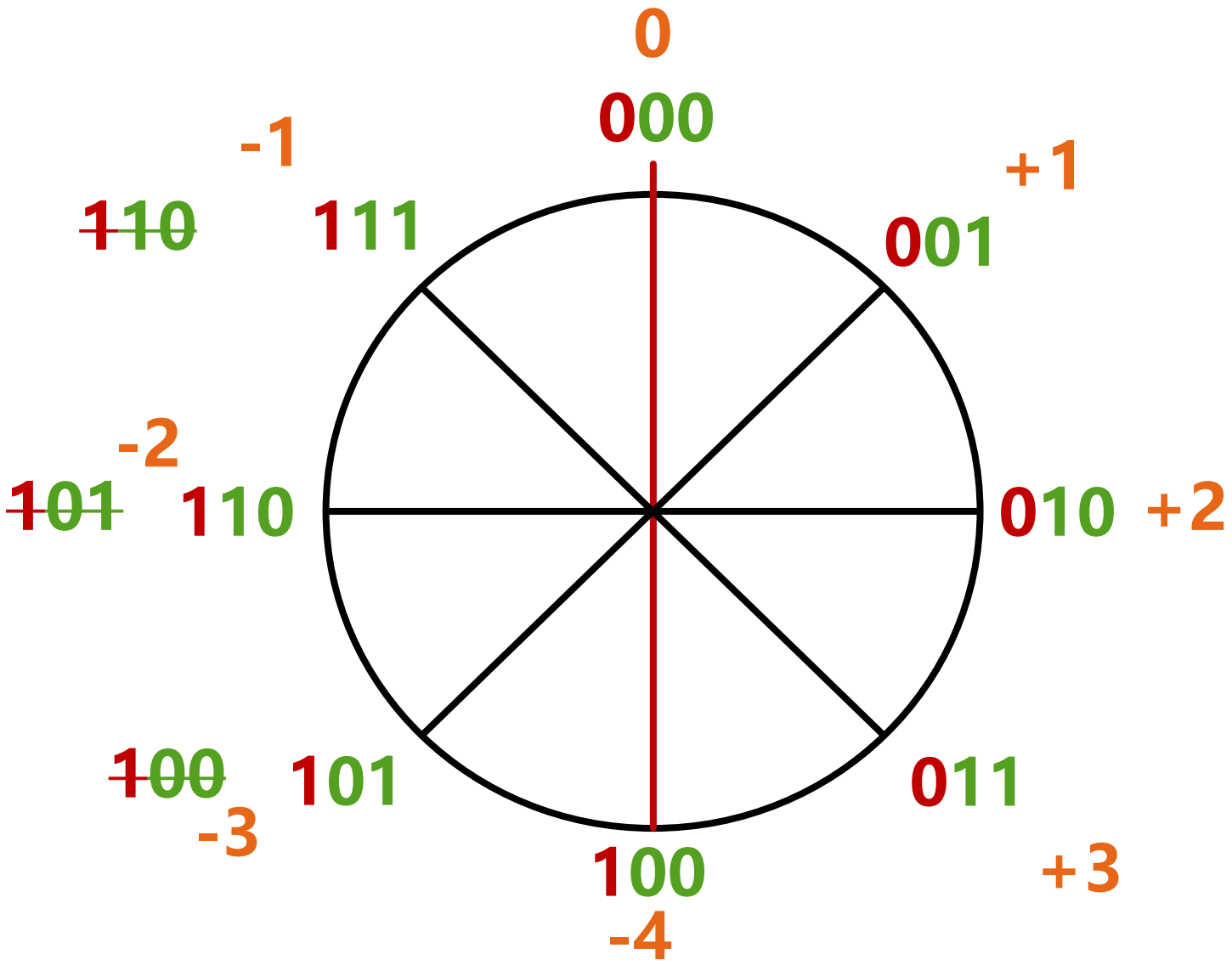
001+110=111

(+1)+(-1)=-0

0有2个表示方式+0和-0?

补码

- **补码:正数不变,负数在反码的基础上加1**



最高位溢出舍弃

001+111=~~1~~000

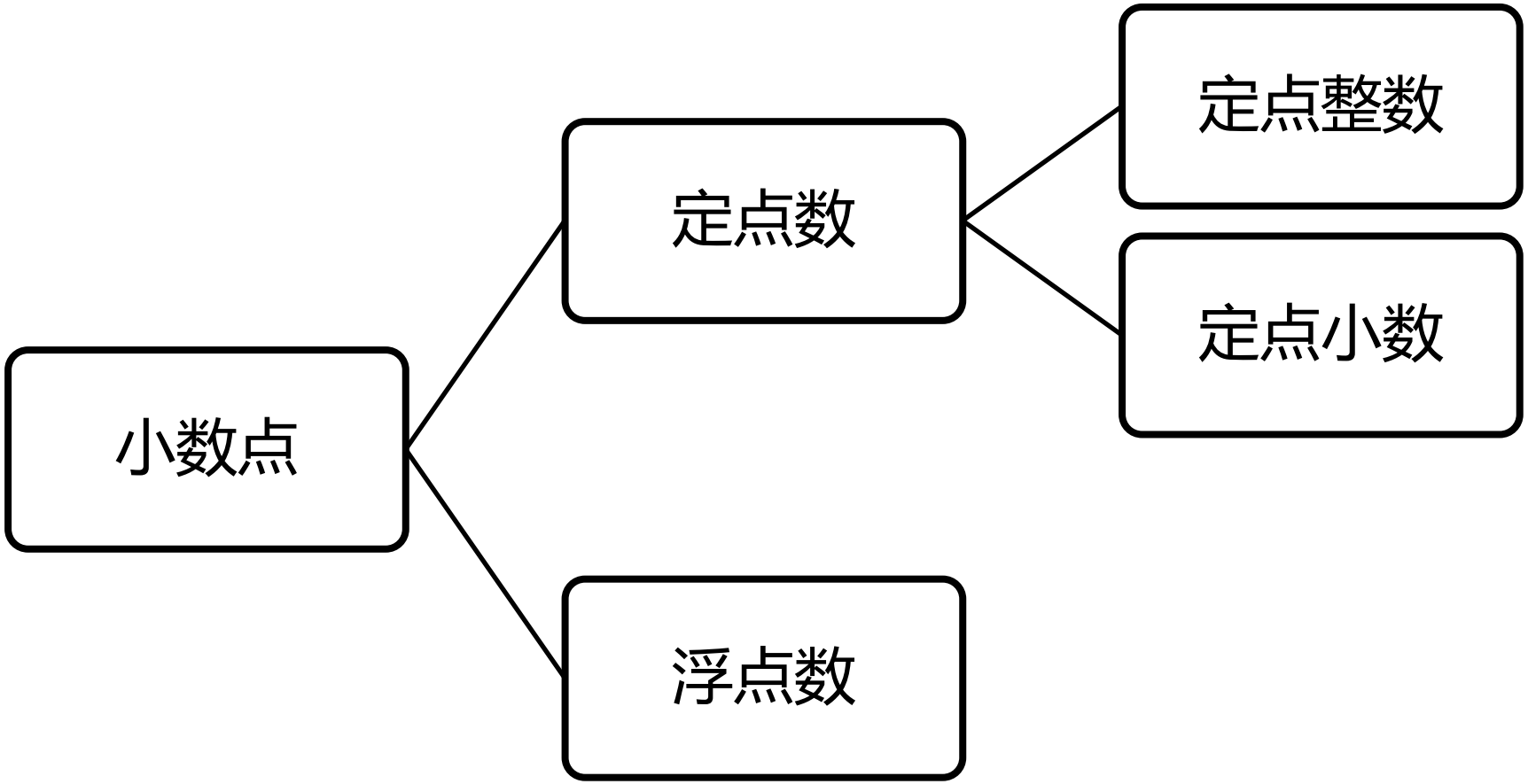
(+1)+(-1)=0

010+110=~~1~~000

(+2)+(-2)=0

小数点表示

- 在计算机中,小数点及其位置是**隐含**规定的,小数点并**不占用**存储空间
- **定点数:** 小数点的位置是固定不变的,分为定点整数和定点小数
- **浮点数:**小数点的位置是会变化的



定点小数

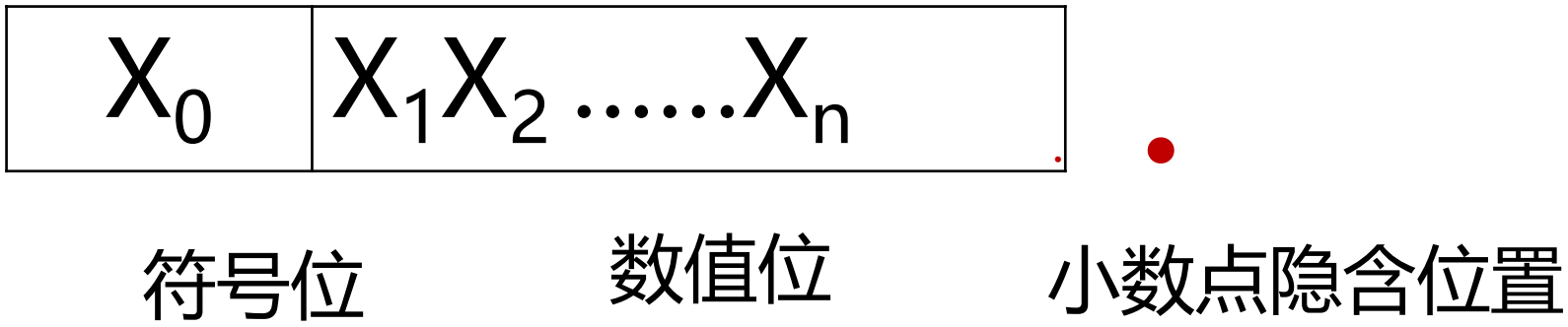
- 定点小数: 小数点隐含固定在**最高数据位的左边**,整数位则用于表示符号位,用于表示**纯小数.**

X_0	.	X_1X_2 ……X_n
符号位	小数点隐含位置	数值位

- 例如: $(0.110011)_2=0.5+0.25+0+0+0.03125+0.015625=0.796875$

定点整数

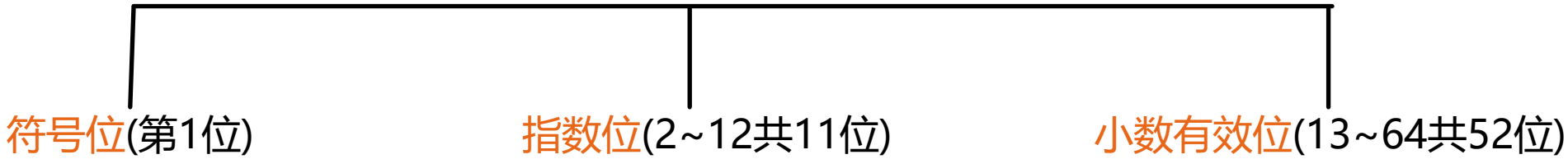
- 定点整数: 小数点位置隐含固定在**最低位之后**,最高位为符号位,用于表示**纯整数.**



- 例如: $(0110011.)_2=1+2+16+32=51$

IEEE754标准

- **JavaScript采用的是双精度(64位)**
- **符号位决定了一个数的正负，指数部分决定了数值的大小，小数有效位部分决定了数值的精度**



- 例如: $(3.5)_{10}=(11.1)_2=1.11*2^1$

0	00000000001	110...00
符号位	指数位(11位)	有效位(52位)

- 一个数在 JavaScript 内部实际的表示形式 $(-1)^{符号位}*1.有效位*2^{指数位}$
- 精度最多53个二进制位, $-(2^{53}-1)$到$2^{53}-1$
- 指数部分最大值是 2017($2^{11}-1$),分一半表示负数,JavaScript能够表示的数值范围是2^{1024}~2^{-1023}

0.1+0.2!=0.3

珠峰架构

十进制(0.1) ⇨ 二进制 (0.0001100110)

0.1		取整	
x 2			
0.2	0		(最高位)
x 2			
0.4	0		
x 2			
0.8	0		
x 2			
0.6	1		循环(0110)
x 2			
0.2	1		
x 2			
0.4	0		(最低位)

1/3+1/3+1/3=3/3=1

0.333+ 0.333+ 0.333=0.999

十进制(0.2) ⇨ 二进制 (0.001100110)

0.2		取整	
x 2			
0.4	0		(最高位)
x 2			
0.8	0		
x 2			
0.6	1		循环(0110)
x 2			
1.2	1		
x 2			
0.4	0		(最低位)

0.1+0.2求和

```
  0.0001100110
+ 0.0011001100
--------------
  0.0100110110
```

0.3

```
  0.0100110011
```

十进制(0.3) ⇨ 二进制 (0.0100110011)

0.3		取整	
x 2			
0.6	0		(最高位)
x 2			
0.2	1		
x 2			
0.4	0		
x 2			
0.8	0		
x 2			
0.6	1		循环(0110)
x 2			
0.2	1		
x 2			
0.4	0		(最低位)

JS大数相加

- 列竖式方法 从低位向最高位计算的
- 1.把原始数字进行倒序
- 2.从个位起开始依次相加

```
    1 2 3 4 5
+   5 4 3 2 1
-----------------
    6 6 6 6 6
```

0	1	2	3	4
5	4	3	2	1

+

0	1	2	3	4
1	7	3	4	5

=

0	1	2	3	4	
6	1	7	6	6	

```
    1 9
+   1 2
-----------
    3 1
```

0	1
9	1

+

0	1
2	1

=

0	1
1	3

JS大数相加

珠峰架构

```
let numA = "1234567890",numB = "123456789";
let numAArray = numA.split("").map((item) => parseInt(item)).reverse();
let numBArray = numB.split("").map((item) => parseInt(item)).reverse();
let sum = [].fill(0,0,(numA.length >= numB.length ? numA.length : numB.length) + 1);
for (let i = 0; i < numAArray.length; i++) {
  sum[i] = numAArray[i];
}
let up = 0;
for (let i = 0; i < numBArray.length; i++) {
  sum[i] = sum[i] + numBArray[i] + up;
  if (sum[i] > 9) {
    sum[i] = sum[i] % 10;
    up = 1;
  } else {
    up = 0;
  }
}
if (sum[sum.length - 1] == 0) {sum.pop()}
let result = sum.reverse().join("");
console.log(Number(numA) + Number(numB));
console.log(result);
```