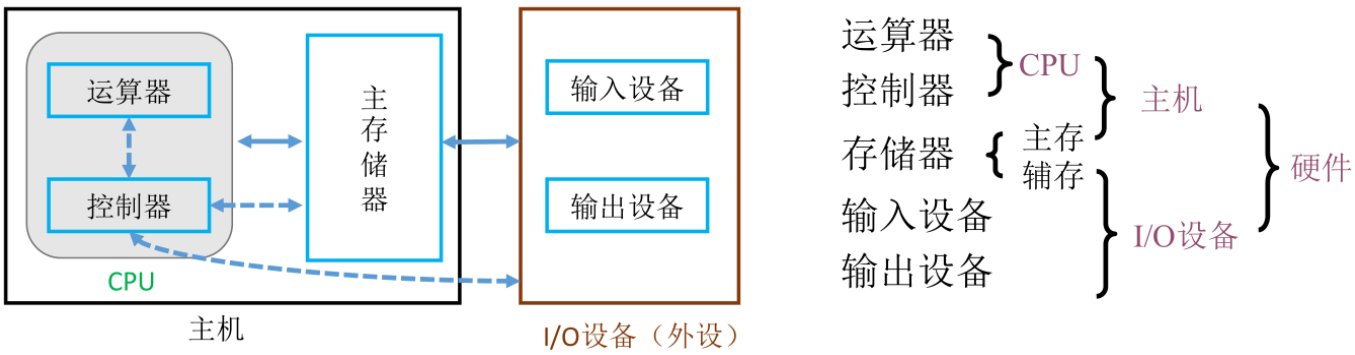
今天是第二章——数据的表示和运算。本章探讨数据如何在计算机中表示，运算器如何实现数据的算数、逻辑运算

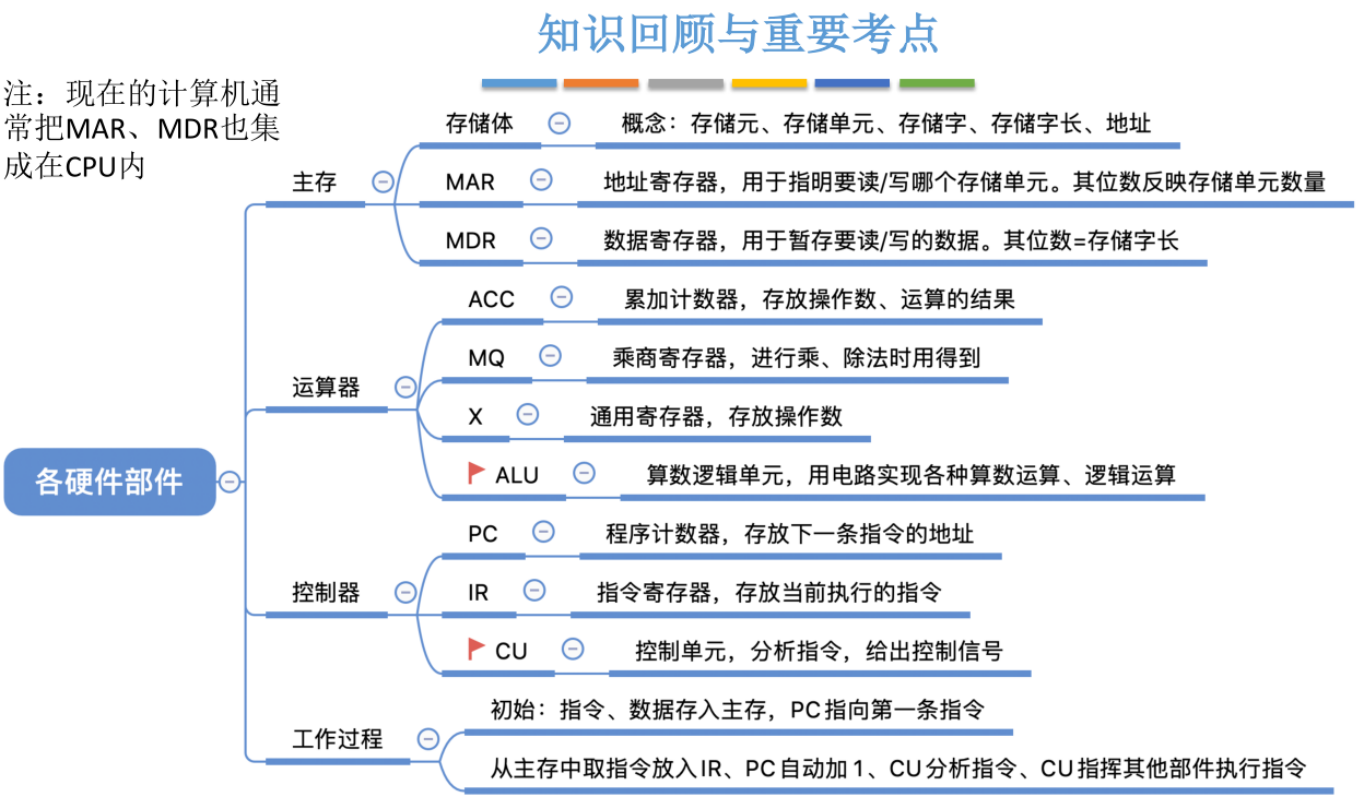
零、回顾

计算机硬件组成框架如下：



主机 = CPU + M.M， CPU = **运算器** + **控制器**

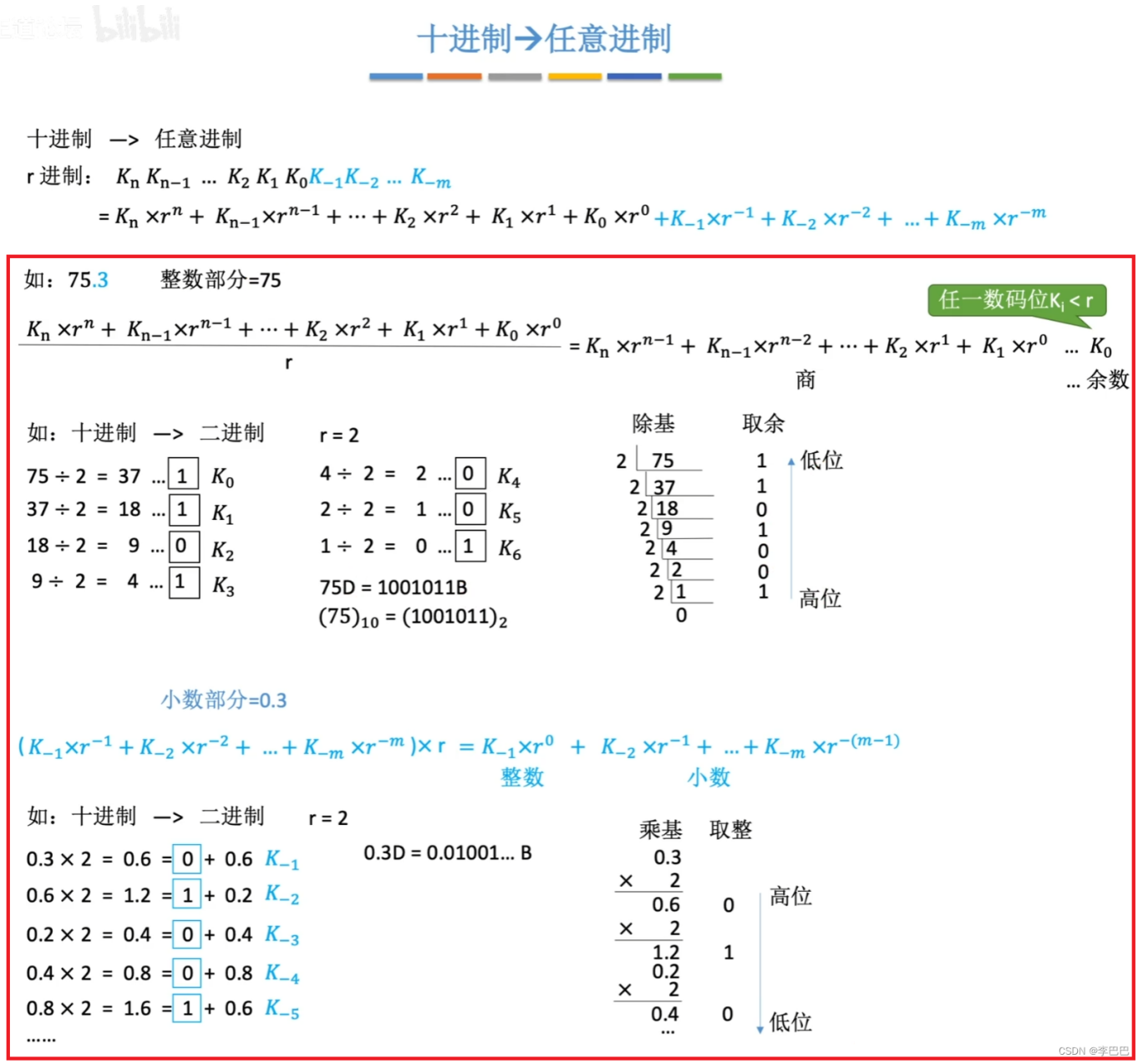
他们的详细组成和内容如下：



本章开始介绍运算器的相关原理

一、数制与编码

1.1 进位计数制

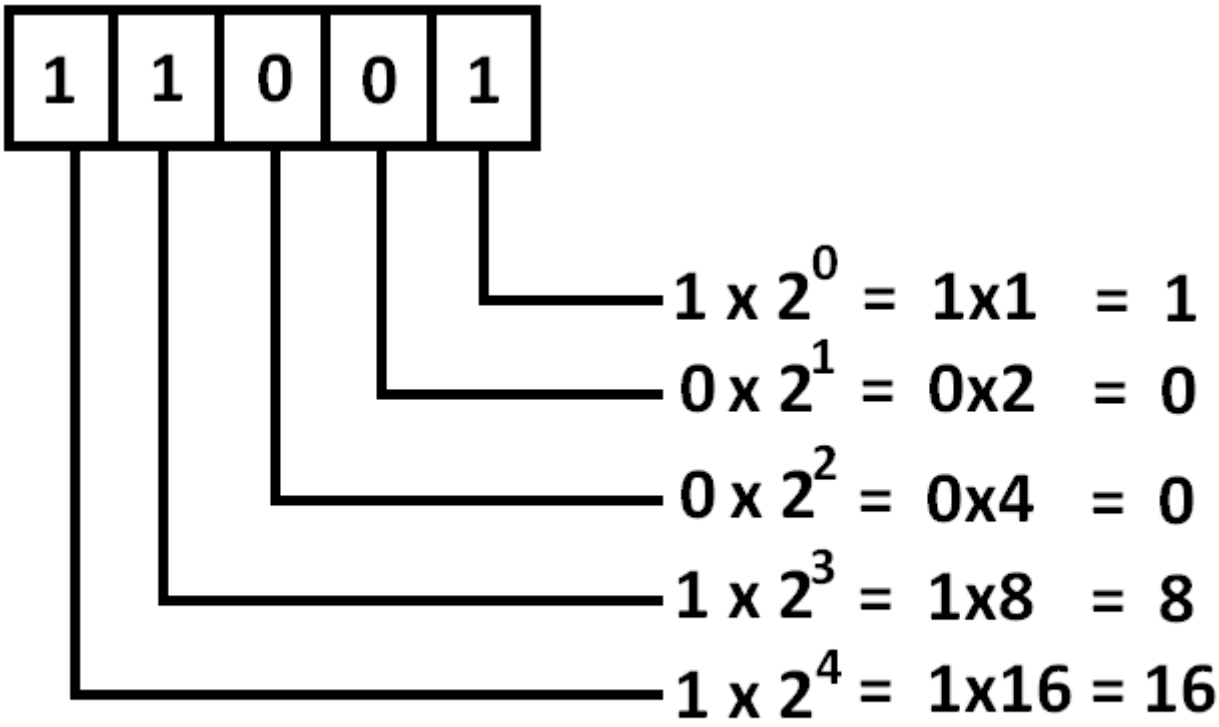


十进制转换为二进制需要分为两个部分：整数部分和小数部分

- 整数部分使用的是**除基取余**的方法
- 小数部分使用的是**乘基取整**的方法

这里值得注意的是：任意二进制小数都可以用十进制小数表示，但是并不是任意十进制小数都可以用二进制小数表示

二进制转换为十进制，如图所示，可以直接加上所有1对应位置上的权值；



r 进制： $K_n K_{n-1} \dots K_2 K_1 K_0 K_{-1} K_{-2} \dots K_{-m}$ 位权

$$= K_n \times r^n + K_{n-1} \times r^{n-1} + \cdots + K_2 \times r^2 + K_1 \times r^1 + K_0 \times r^0 + K_{-1} \times r^{-1} + K_{-2} \times r^{-2} + \dots + K_{-m} \times r^{-m}$$

基数：每个数码位所用到的不同符号的个数，r 进制的基数为 r

①可使用两个稳定状态的物理器件表示
②0，1 正好对应逻辑值 假、真。方便实现逻辑运算
③可很方便地使用逻辑门电路实现算术运算

二进制： 0,1

八进制： 0,1,2,3,4,5,6,7

十进制： 0,1,2,3,4,5,6,7,8,9

十六进制： 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

二进制： 101.1 —> $1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} = 5.5$

八进制： 5.4 —> $5 \times 8^0 + 4 \times 8^{-1} = 5.5$

十进制： 5.5 —> $5 \times 10^0 + 5 \times 10^{-1} = 5.5$

十六进制： 5.8 —> $5 \times 16^0 + 8 \times 16^{-1} = 5.5$

如：1111000010.01101

二进制 —> 八进制

3位一组，每组转换成对应的八进制符号

001 111 000 010 . 011 010

1 7 0 2 . 3 2 八进制

八进制—> 二进制

每位八进制对应的3位二进制

$(251.5)_8$ → $(010\ 101\ 001.\ 101)_2$

二进制 —> 十六进制

4位一组，每组转换成对应的十六进制符号

0011 1100 0010 . 0110 1000

3 C 2 . 6 8 十六进制

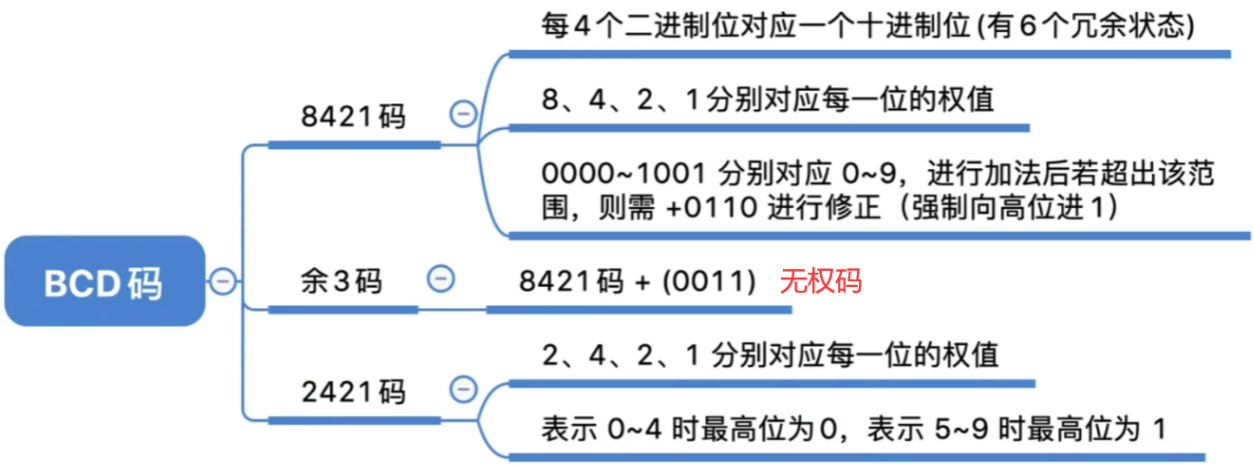
十六进制—> 二进制

每位十六进制对应的4位二进制

$(AE86.1)_{16}$ → $(1010\ 1110\ 0110.\ 0001)_2$

1.2 BCD码

BCD 码就是用 4 个 bit 位（二进制数）对应 1 个十进制位，是一种二进制的数字编码形式，用二进制编码来代替十进制代码。



CSDN @李巴巴

- BCD码的运算

8421码的映射关系：

0	1	2	3	4	5	6	7	8	9
0000	0001	0010	0011	0100	0101	0110	0111	1000	1001

十进制： 5 + 8 13 +6 1 3 1 3

8421码：0101 + 1000 1101 + 0110 1 0011 0001 0011

机算方法

不在映射表里

8421码中 1010～1111 没有定义

注：若相加结果在合法范围内，则无需修正。

CSDN @李巴巴

- 余 3 码（无权码）

余3码：8421码 + $(0011)_2$

0	1	2	3	4	5	6	7	8	9
0011	0100	0101	0110	0111	1000	1001	1010	1011	1100

- 2421 码

2421码：改变权值定义

5也可以表示为 0101，为了保持唯一性，0~4的第一位都为0
5~9的第一位都为1

0	1	2	3	4	5	6	7	8	9
0000	0001	0010	0011	0100	1011	1100	1101	1110	1111

1.3 ASCII 码

我们知道我们日常见到的数字（09）、符号比如：#、%、+、@等、英语字母如：(Az)、(a~z),它们在计算机中都是以二进制来表示存储的，咱们可以用不同的二进制数来表示，但是为了统一化、使大家的相互通信可以正常进行，那么就必须制定一套统一的标准来规范它，由此ASCII码随之诞生，大家都遵从这一统一的标准进行数据的通信和交流。

ASCII码

0	NUL	16	DLE	32	SPC	48	0	64	@	80	P	96	`	112	p
1	SOH	17	DC1	33	!	49	1	65	A	81	Q	97	a	113	q
2	STX	18	DC2	34	"	50	2	66	B	82	R	98	b	114	r
3	ETX	19	DC3	35	#	51	3	67	C	83	S	99	c	115	s
4	EOT	20	DC4	36	$	52	4	68	D	84	T	100	d	116	t
5	ENQ	21	NAK	37	%	53	5	69	E	85	U	101	e	117	u
6	ACK	22	SYN	38	&	54	6	70	F	86	V	102	f	118	v
7	BEL	23	ETB	39	'	55	7	71	G	87	W	103	g	119	w
8	BS	24	CAN	40	(	56	8	72	H	88	X	104	h	120	x
9	HT	25	EM	41	)	57	9	73	I	89	Y	105	i	121	y
10	LF	26	SUB	42	*	58	:	74	J	90	Z	106	j	122	z
11	VT	27	ESC	43	+	59	;	75	K	91	[	107	k	123	{
12	FF	28	FS	44	,	60	<	76	L	92	\	108	l	124	|
13	CR	29	GS	45	-	61	=	77	M	93	]	109	m	125	}
14	SO	30	RS	46	.	62	>	78	N	94	^	110	n	126	~
15	SI	31	US	47	/	63	?	79	O	95	_	111	o	127	DEL

可印刷字符：32～126，其余为控制、通信字符

数字：48(0011 0000)～57(0011 1001)

大写字母：65(0100 0001)～90(0101 1010)

小写字母：97(0110 0001)～122(0111 1010)

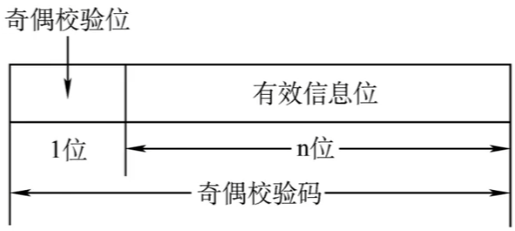
1.4 校验码

校验码中我们最常见的有以下三种：奇偶校验码、海明校验码和CRC循环冗余码；

- 奇偶校验码：这也是最常见的一种校验码，他又分为奇校验码和偶校验码；奇校验码指的是整个校验码（包含信息位和校验位）中1的个数是奇数个，而偶校验码指的是整个校验码中1的个数为偶数个；

奇偶校验码

奇校验码： 整个校验码（有效信息位和校验位）中“1”的个数为奇数。
偶校验码： 整个校验码（有效信息位和校验位）中“1”的个数为偶数。



【例2-3】给出两个编码1001101和1010111的奇校验码和偶校验码。

设最高位为校验位，余7位是信息位，则对应的奇偶校验码为：

奇校验：　1̲1001101　　0̲1010111

偶校验：　①0̲1001101　　②1̲1010111

偶校验的硬件实现：各信息进行异或（模2加）运算，得到的结果即为偶校验位

⊕：异或（模2加）

0⊕0 = 0
0⊕1 = 1
1⊕0 = 1
1⊕1 = 0

求偶校验位：
①1⊕0⊕0⊕1⊕1⊕0⊕1=0
②1⊕0⊕1⊕0⊕1⊕1⊕1=1

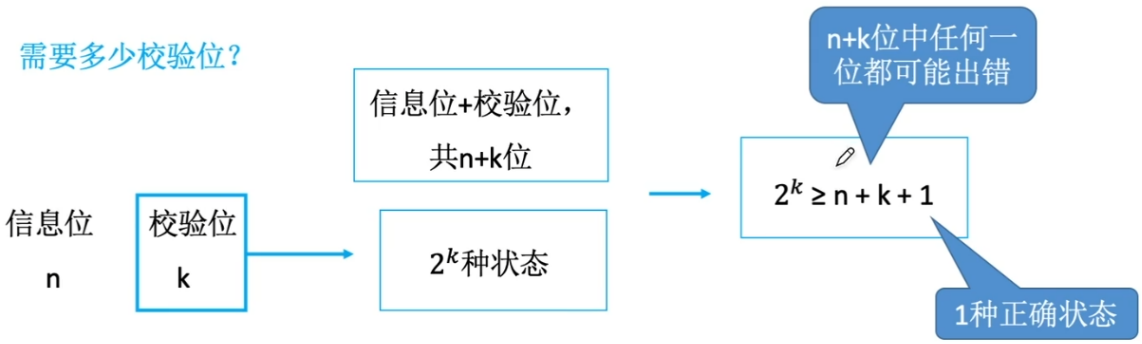
进行偶校验（所有位进行异或，若结果为1说明出错）：
0⊕1⊕0⊕0⊕1⊕1⊕0⊕1=0　未出错
1⊕1⊕0⊕1⊕0⊕1⊕1⊕0=1　出错
1⊕1⊕0⊕1⊕0⊕1⊕0⊕0=0　未检测到出错

非常简单明了，但是我们需要知道奇偶校验码是具有局限性的，只能发现奇数位的错误，不能发现偶数位的错误而且不能纠正错误。那为什么还会说它是最常见的呢？因为我们在通信最常发生的便是一位数据的错误了。**奇偶校验码的码距 d = 2，仅能检测出奇数位错误，无纠错能力。**

- 海明校验码：海明码的特点有四个字：纠一检二（纠正一位错误，检测两位错误）海明码往往会在首部加上全检验进行偶校验码；

海明校验码思路简介

海明码设计思路：将信息位分组进行偶校验 —> 多个校验位
—>多个校验位标注出错位置



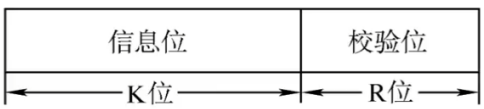
n	1	2-4	5-11	12-26	27-57	58-120
k	2	3	4	5	6	7

- CRC循环冗余码：循环冗余码应用的场景主要是有大量数据传输时，通过与生成多项式做模2运算得到一个新的二进制码；当接收端接收到数据时，再用相同的多项式进行模2运算，如果结果为0则表示数据没有错误；接收端检测时检测出一位数据错误后，纠正的方法包括：请求重发、删除数据以及通过余数值进行纠正。

循环冗余校验码的基本思想

```
  126          126          121
7|882        7|883        7|852
 7            7            7
 --           --           --
 18           18           15
 14           14           14
 --           --           --
  42           43           12
  42           42            7
 --           --           --
  0            1            5
```

数据出错导致余数改变——检测到错误



循环冗余校验码的思想：

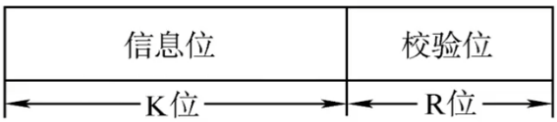
数据发送、接受方约定一个“除数”二进制

K个信息位+R个校验位作为“被除数”，添加校验位后需保证除法的余数为0

收到数据后，进行除法检查余数是否为0

若余数非0说明出错，则进行重传或纠错

循环冗余校验码



【例2-5】 设生成多项式为$G(x)=x^3+x^2+1$，信息码为101001，求对应的CRC码。

1. 确定K、R以及生成多项式对应的二进制码

K = 信息码的长度 = 6，R = 生成多项式最高次幂 = 3 → 校验码位数N = K + R = 9

生成多项式$G(x) = 1\cdot x^3 + 1\cdot x^2 + 0\cdot x^1 + 1\cdot x^0$，对应二进制码1101

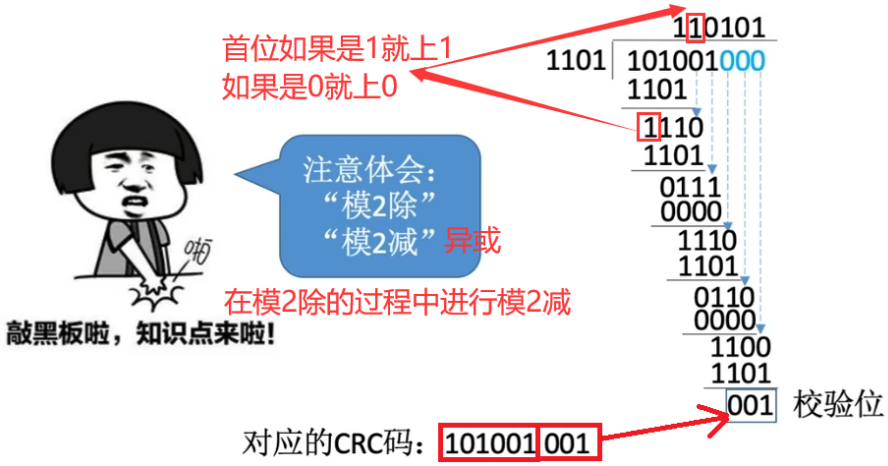
2. 移位

信息码左移R位，低位补0

1101 | 101001000

3. 相除

对移位后的信息码，用生成多项式进行模2除法，产生余数



4. 检错和纠错

发送：101001001　记为$C_9C_8C_7C_6C_5C_4C_3C_2C_1$

接收：101001001　用1101进行模2除 → 余数为000，代表没有出错

接收：101001011　用1101进行模2除 → 余数为010，~~代表C_2出错~~ 不严谨

理论上，循环冗余校验码的检错能力有以下特点：可检测出所有奇数个错误，可检测出所有双比特的错误，可检测所有小于等于检验位长度的连续错误。CRC **循环冗余码实际应用中一般只用来“检错”，不用来“纠错”**

二、定点数的表示和运算

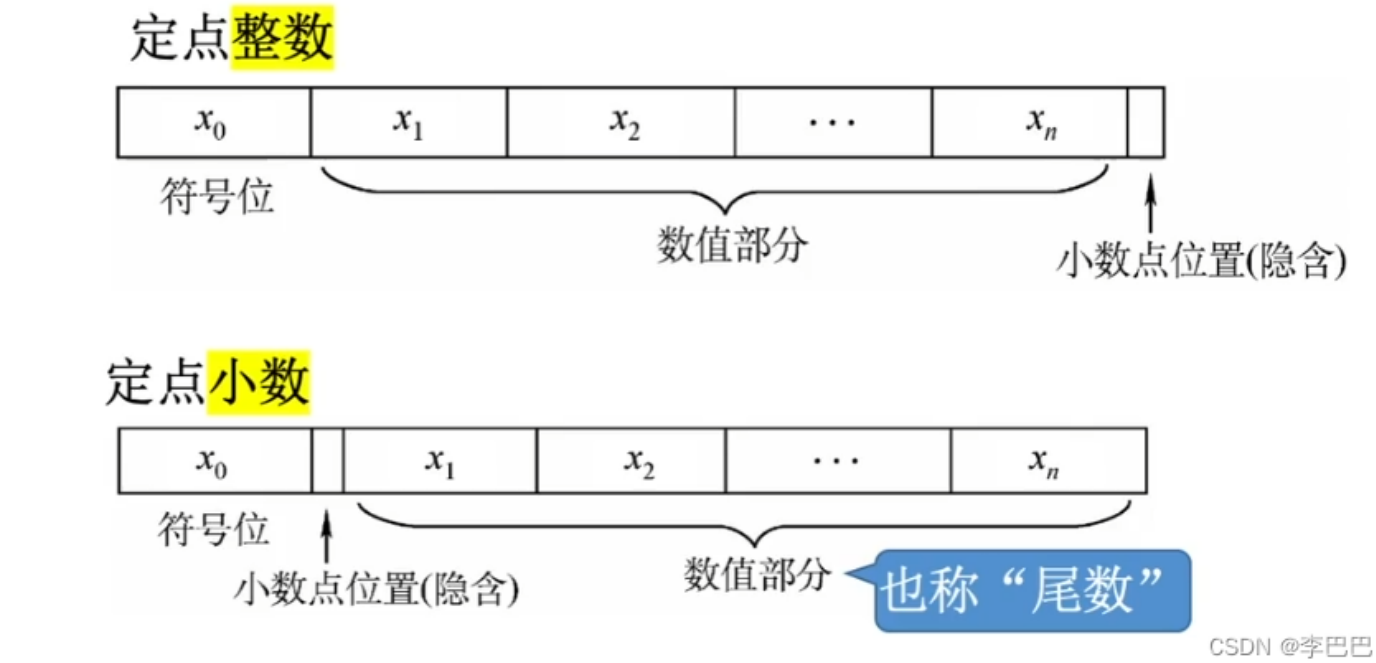
2.1 定点数的表示

（1）无符号

整个机器字长的全部二进制位均为数值位，**没有符号位**，相当于数的绝对值。若机器字长为 8 位，则数的表示范围 0~255。（通常只有无符号整数，没有无符号小数）

（2）有符号（原码、反码、补码、移码）

在机器中，数的正负我们无法识别，但是我们可以用二进制数来代替正负号。一般‘0’为正，‘1’为负，**符号位**一般在有效数的最前面。



在计算机中表示一个定点数包括原码、补码、反码和移码；

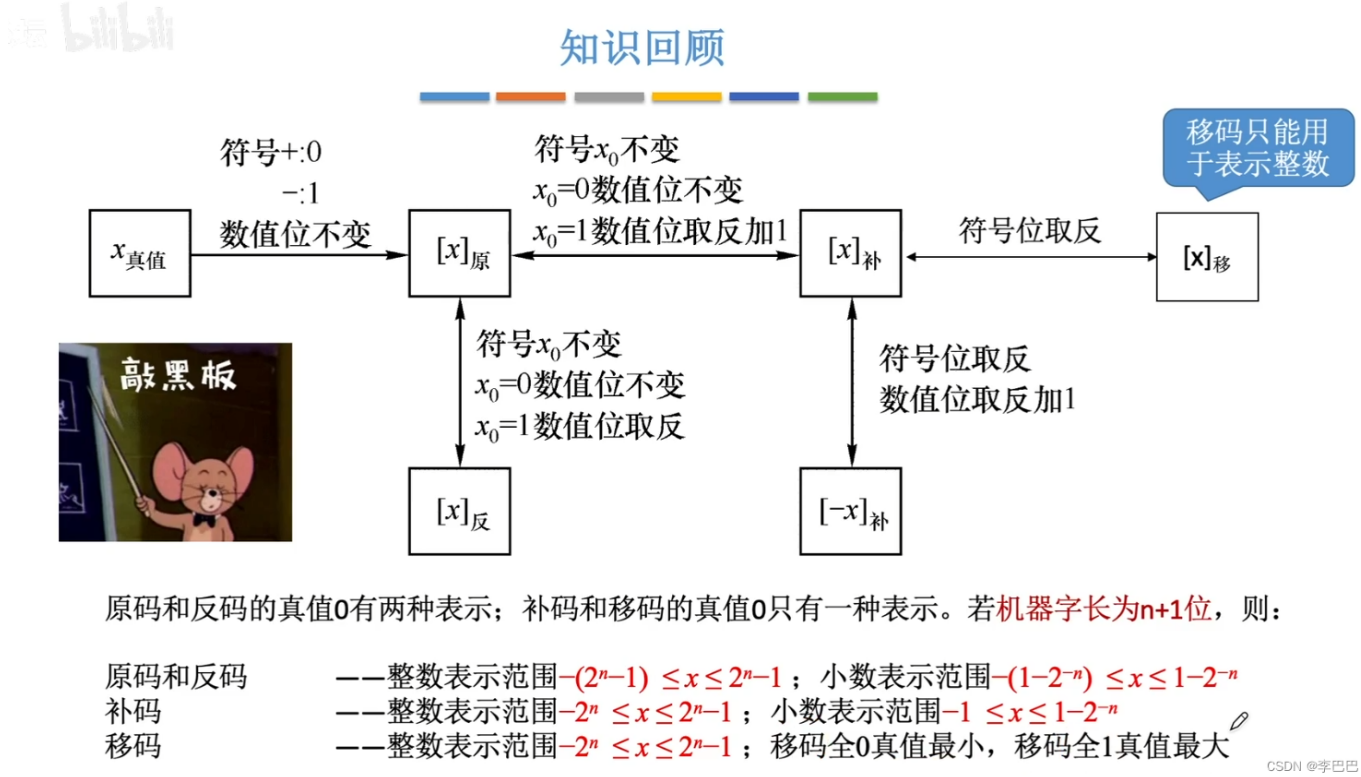
- 原码：针对正数的原码，我们会在最高位添加0，针对负数，我们会在最高位添1，其他位按照无符号数处理即可；原码是有正零和负零之分的，因此定点整数原码的表示范围只是：-(2n-1)~2n-1（关于原点对称），定点小数也是一样；
- 补码：补码可以说是这些码中最重要的，因为在计算机中存储的定点数便是补码形式，首先我们要知道为什么需要补码，因为在计算机的内部进行加减运算时，我们会发现两个正数相加还好，通过加法器就能实现，但是当两个正数相减时，这个减法器可就不好设计了，因此出现了补码。
    - 由于计算机硬件的位数是固定的，也就是寄存器中的位数是固定的，那么当两个数相加超过寄存器位数所能表示的最大整数时，其实就相当于进行了一次简单的模运算。打一个简单的比方：假设现在有三位寄存器，最大表示7（即111），我现在有两个数5（即101）和3（即011），两个数相加得到1000，但是因为寄存器只有三位，那么剩下的便只有000了，那如果变成了减法，其实5-3不就是5+(-3)，结果2也可以表示成10，那么-3

便也可以用5来替代；根据这一规律，我们发现负数是可以用与寄存器能够表示的最大数范围取模的正数表示的；

  - 具体实现其实很简单，**正数的补码和原码相同，负数的补码就是原码除了符号位取反加一。**补码的范围需要注意的是，补码的-0即1000……（n+1位）用来表示的是-2n，范围是：-2n到2n-1；小数将-0用来表示了-1，范围是：-1到1-2-n;

- 反码：反码是补码产生的一个衍生物，我们知道补码是原码取反加1，是的，你可能已经猜到了，原码不加1只取反便是反码了，当然这只是针对负数，正数就是用原码表示；范围同原码。

- 移码：移码是把补码的符号位取反；看似十分简单，但是移码非常有用的，因为移码保持了原有数据的大小顺序，所以，移码越大真值越大，当有找最大最小值或者大小比较问题时，我们首先需要想到移码；移码因为来自补码，所以范围同补码；



原码和反码的真值0有两种表示；补码和移码的真值0只有一种表示。若机器字长为n+1位，则：

原码和反码	——整数表示范围$-(2^n-1) \le x \le 2^n-1$；小数表示范围$-(1-2^{-n}) \le x \le 1-2^{-n}$
补码	——整数表示范围$-2^n \le x \le 2^n-1$；小数表示范围$-1 \le x \le 1-2^{-n}$
移码	——整数表示范围$-2^n \le x \le 2^n-1$；移码全0真值最小，移码全1真值最大

一些之前学习的Tips：

- 在补码表示法中，真值“0”的补码是唯一的，即0.00…0，而补码1.00…0则代表“1”(二进制小数为例)

  - 由于源码“0”的表示有“+0”和“-0”之分，因此范围是$[-(2^n-1), 2^n-1]$

  - 补码由于没有这个概念，源码“-0”对应的二进制表示在补码中表示绝对值最大的负数，因此范围是$[-2^n, 2^n-1]$

2.2 定点数的运算

(1) 移位运算

移位：通过改变各个数码位和小数点的相对位置，从而改变各数码位的位权。可用移位运算实现乘法、除法。

算术位移

- 原码的算术移位：符号位保持不变，仅对数值位进行移位。
  - 右移：高位补0，低位舍弃。若舍弃的位 = 0，则相当于 ÷ 2；若舍弃的位 ≠ 0，则会**丢失精度**
  - 左移：低位补0，高位舍弃。若舍弃的位 = 0，则相当于 × 2；若舍弃的位 ≠ 0，则会**出现严重错误**
- 反码的算术位移

- 补码的算术位移

补码的算数移位——正数的补码与原码相同，因此对正数补码的移位运算也和原码相同。
右移：高位补0，低位舍弃。
左移：低位补0，高位舍弃。

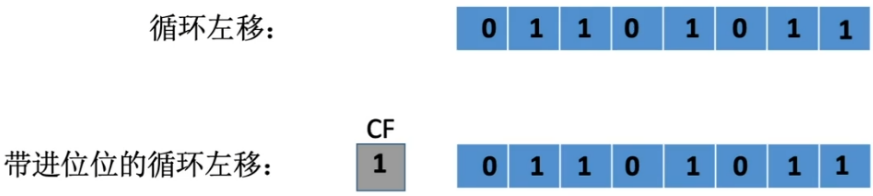
补码的算数移位——负数补码=反码末位+1
导致反码最右边几个连续的1都因进位而变为0，直到进位碰到第一个0为止。
规律——负数补码中，最右边的1及其右边同原码。最右边的1的左边同反码
负数补码的算数移位规则如下：
右移（同反码）：高位补1，低位舍弃。
左移（同原码）：低位补0，高位舍弃。

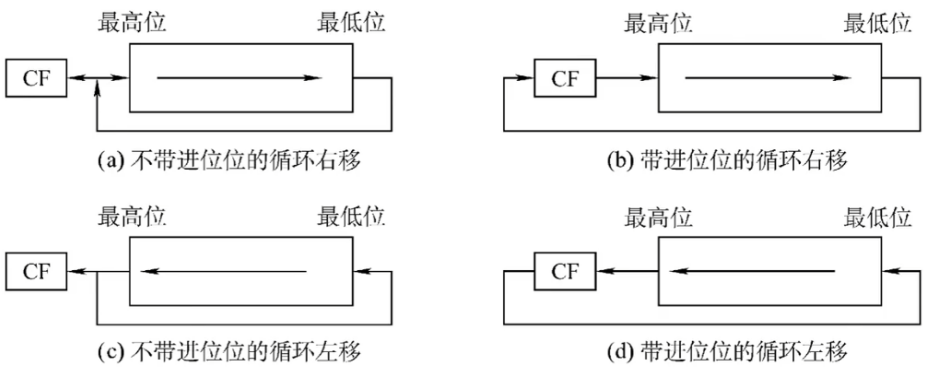
逻辑位移

- 逻辑右移：高位补 0，低位舍弃
- 逻辑左移：低位补 0，高位舍弃

逻辑位移可以看作对 “无符号数” 的算术移位。

循环位移





(2) 加减运算

原码的加减运算

原码的加法运算：

正+正 →绝对值做加法，结果为正
负+负 →绝对值做加法，结果为负

} 可能会溢出

正+负 →绝对值大的减绝对值小的，符号同绝对值大的数
负+正 →绝对值大的减绝对值小的，符号同绝对值大的数

原码的减法运算，“减数”符号取反，转变为加法：

正-负→正+正
负-正→负+负
正-正→正+负
负+正→负-负

补码的加减运算：对于补码来说，无论加法还是减法，**最后都会转变成加法**，由加法器实现运算，符号位也参与运算。

设机器字长为8位（含1位符号位），$A=15$，$B=-24$，求$[A+B]_{补}$和$[A-B]_{补}$

	原码	补码
$A=+1111$	→ 0,0001111	→ 0,0001111
$B=-11000$	→ 1,0011000	→ 1,1101000

负数补→原：①数值位取反+1；②负数补码中，最右边的1及其右边同原码。最右边的1的左边同反码

$[A+B]_{补}=[A]_{补}+[B]_{补}=0,0001111+1,1101000=1,1110111$

原码：1,0001001　真值-9

$[A-B]_{补}=[A]_{补}+[-B]_{补}=0,0001111+0,0011000=0,0100111$　真值+39

$[-B]_{补}$：　$[B]_{补}$连同符号位一起取反加1

溢出判断

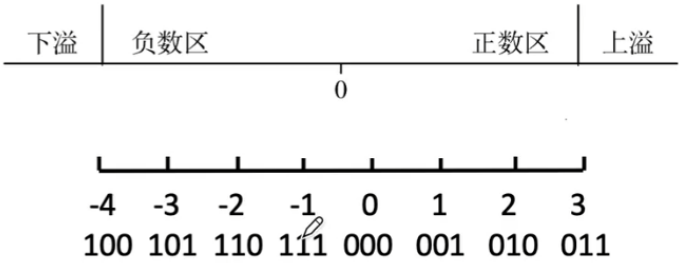
溢出判断

设机器字长为8位（含1位符号位），$A=15$，$B=-24$，求$[A+B]_{补}$和$[A-B]_{补}$

$C=124$，求$[A+C]_{补}$和$[B-C]_{补}$

$[A+C]_{补}=0.0001111+0.1111100=1.0001011$　　真值-117

$[B-C]_{补}=1.1101000+1.0000100=0.1101100$　　真值+108



只有“正数+正数”才会上溢 —— 正+正=负

只有“负数+负数”才会下溢 —— 负+负=正

(3) 补码一位乘

补码一位乘法（手算模拟）

设机器字长为5位（含1位符号位，$n=4$），$x=-0.1101$，$y=+0.1011$，采用Booth算法求$x\cdot y$

$[x]_{补}=11.0011$，$[-x]_{补}=00.1101$，$[y]_{补}=0.1011$

	(高位部分积)	(低位部分积/乘数)	说明
	00.0000	0.1011 0 丢失位	起始情况
+$[-x]_{补}$	00.1101	辅助位	$Y_4Y_5=10$，$Y_5-Y_4=-1$，则+$[-x]_{补}$
	00.1101		
右移	00.0110 -----	10.1011 0	右移部分积和乘数
+0	00.0000		$Y_4Y_5=11$，$Y_5-Y_4=0$，则+0
	00.0110		
右移	00.0011 -----	010.1011 0	右移部分积和乘数
+$[x]_{补}$	11.0011		$Y_4Y_5=01$，$Y_5-Y_4=1$，则+$[x]_{补}$
	11.0110		
右移	11.1011 -----	0010.1101 0	右移部分积和乘数
+$[-x]_{补}$	00.1101	原符号位	$Y_4Y_5=10$，$Y_5-Y_4=-1$，则+$[-x]_{补}$
	00.1000		
右移	00.0100 -----	00010.1011 0	右移部分积和乘数
+$[x]_{补}$	11.0011		$Y_4Y_5=01$，$Y_5-Y_4=1$，则+$[x]_{补}$
	11.0111	构成$[x\cdot y]_{补}$	

正数算数右移

负数算数右移

最后多一次加法

n轮加法、算数右移，加法规则如下：

辅助位 - MQ中最低位 = 1时，(ACC)+$[x]_{补}$

辅助位 - MQ中最低位 = 0时，(ACC)+0

辅助位 - MQ中最低位 = -1时，(ACC)+$[-x]_{补}$

补码的算数右移：

符号位不动，数值位右移，正数右移补0，负数右移补1（符号位是啥就补啥）

$[x\cdot y]_{补}=11.01110001$

即$x\cdot y=-0.10001111$