数字逻辑电路

数字逻辑基础

西安交通大学

电子物理与器件教育部重点实验室

等离子体与微波电子学研究所

张小宁

数字逻辑基础

第一节：数字逻辑概述

第二节：数制与编码

第三节：逻辑代数基础

第四节：数字逻辑门电路

数制与编码

1、数制

- 十进位计数制的特点、表示法，任意进制数
- 任意数制的特点、表示法、运算规则及相互关系

2、数制的相互转换

- 多项式替代法，基数乘除法，桥梁法
- 直接转换法，小数位置的确定

3、数的表示方法及其运算

- 小数点的表示，带符号数的表示及其运算
- 十进制数的代码表示及其运算

4、可靠性编码

1. 数制

数制：多位数码中的每一位数的构成及低位向高位进位的规则

- 日常生活中采用的是十进制计数制，计数规则“逢十进一”，

  例：0,1,2,3,4,5,6,7,8,9,10,11, 12,…,99,100, …,;

- 在计算机中多用的是二进制计数制，计数规则“逢二进一”

  例：0,1,10,11,100,101,110,111,…。

1.1 十进位计数制的特点、表示法:

数制：多位数码中的每一位数的构成及低位向高位进位的规则

(1) 10个有序的数字符号：0,1,2,3,4,5,6,7,8,9

(2) 符号：“.”，“+”，“-”

(3) “逢十进一”的计数规则，其中：“十” 为进位基数，简称基数。

1. 数制

十进制表示法：并列表示法；多项式表示

（1）并列表示法(位置计数法)：处在不同位置的数字具有不同的“(*Weight*)”。

万位	千位	百位	十位	个位		十分位	百分位	千分位	万分位	十万分位
10^4	10^3	10^2	10^1	10^0		10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}
1	2	3	4	5	.	6	7	8	0	9

一般表达式：$(K_{n-1}K_{n-2}\cdots K_1K_0.K_{-1}K_{-2}\cdots K_{-m})_{10}$　$(0 \leqslant K_i \leqslant 9)$

2）多项式表示法:

将并列式按“权”展开为按权展开式，称为多项式表示法。

$$12345.67809 = 1\times10^4+2\times10^3+3\times10^2+4\times10^1+5\times10^0$$
$$+6\times10^{-1}+7\times10^{-2}+8\times10^{-3}+0\times10^{-4}+9\times10^{-5}$$

一般表达式:

$$(K_{n-1}\times10^{n-1}+K_{n-2}\times10^{n-2}+\ldots+K_1\times10^1$$
$$+K_0\times10^0+K_{-1}\times10^{-1}+K_{-2}\times10^{-2}+\ldots$$
$$+K_{-m}\times10^{-m})_{10}$$
$$=(\Sigma K_i\times10^i)_{10}$$

1.2 任意R进位计数制的特点、表示法和运算规则

① R个有序的数字符号：0、1、…、R-1；符号：“.”，“+”，“-”

② “逢R进一”的计数规则，其中：R为进位基数或简称“基数”。

例： R=2，二进制，数字符号有0、1，逢二进一；R=10，十进制，数字符号有0,1,2,3,4,5,6,7,8,9，逢十进一；

表示方法：

①并列表示法：$(A_{n-1}A_{n-2}\ldots A_1A_0.A_{-1}A_{-2}\ldots A_{-m})_R$ $(0\le A_i\le R-1)$

②多项式表示法：$(A_{n-1}\times 10^{n-1}+A_{n-2}\times 10^{n-2}+\ldots+A_1\times 10^1+A_0\times 10^0$

$$+A_{-1}\times 10^{-1}+A_{-2}\times 10^{-2}+\ldots+A_{-m}\times 10^{-m})_R=(\sum A_i\times 10^i)_R$$

其中：n是整数位数， m是小数位数，$0\leqslant A_i\leqslant R-1$， R为进位基数，用十进制表示。当R=10时，则括号及括号外的基数R可以省略。

这两种表示法记作：N_R，读作：N的R进制表示。如：N=8，R=2时，8_2读作：8的二进制表示，即：$(1000)_2$

“10”的含义是什么？！！！（对应于相应数制的分位数），n的含义是什么？（也对应相应的分位数）

数制与编码

把下列数表达为多项式形式

$$(1234)_{10} = (1\times10^{3}+2\times10^{2}+3\times10^{1}+4\times10^{0})_{10}$$

$$(1CE8)_{16}=$$

$$(121.2)_{3}=$$

1. 数制

不同进位计数制的数值具有等值关系：

$N = N_{10} = (N)_{10}$　　$N_R \# (N)_R$，（N为十进制数）

例如：$13 = (13)_{10} = 13_8 = (15)_8 = (1101)_2$

运算规则：在R进制中，相应的运算（+，-，×，/）规则与十进制一样，逢R进1。

R=10	R=2	R=3	R=4	R=8	R=16
3	11	10	3	3	3
4	100	11	10	4	4
5	101	12	11	5	5
6	110	20	12	6	6
7	111	21	13	7	7
14	1110	112	32	16	E
15	1111	120	33	17	F
16	10000	121	100	20	10
17	10001	122	101	21	11
…	…	…	…	…	…

1. 数制

特例：二进制数为计算机运算的基础，特予以关注：

① 运算规则：＋、－、×、÷（补码原码反码）

（×、÷运算可以由+、－运算来实现）

加法规则：0＋0＝0　0＋1＝1＋0＝1　1＋1＝10

乘法规则：0×0＝0　0×1＝1×0＝0　1×1＝1，11×11=?

② 常用的二进制常。

i	R^i	i	R^i	i	R^i
-7	0.0078125	0	1	7	128
-6	0.015625	1	2	8	256
-5	0.03125	2	4	9	512
-4	0.0625	3	8	10	1024
-3	0.125	4	16	11	2048
-2	0.25	5	32	12	4096
-1	0.5	6	64	13	8192

数制与编码

1、数制

- 十进位计数制的特点、表示法，任意进制数
- 任意数制的特点、表示法、运算规则及相互关系

2、数制的相互转换

- 多项式替代法，基数乘除法，桥梁法
- 直接转换法，小数位置的确定

3、数的表示方法及其运算

- 小数点的表示，带符号数的表示及其运算
- 十进制数的代码表示及其运算

4、可靠性编码

2. 数制的相互转换

数值转换原则： $N_\alpha \rightarrow N_\beta$，转换是等值的： 即 $N_\alpha = N_\beta$

2.1 多项式替代法

要点：在β进制下完成 $N_\alpha \rightarrow N_\beta$ 的转换

例1：将 $(1CE8)_{16}$ 转换为十进制。 $(1CE8)_{16} = (?)_8$

$$(1CE8)_{16} = (1\times 10^3 + C\times 10^2 + E\times 10^1 + 8\times 10^0)_{16}$$ （10^3=1000指的是16进制的1000）

$$= (1\times 16^3 + 12\times 16^2 + 14\times 16^1 + 8\times 16^0)_{10}$$

$$= (4096 + 3072 + 224 + 8)_{10}$$

$$= (7400)_{10}$$ (计算是按十进制计算规则进行的)

$= 7400$ (十进制的R可以省略) $(16350)_8$ 这个数如何得到的？

2. 数制的相互转换

例2：将$(121.2)_3$转换为二进制。

$$(121.2)_3 = (1\times10^{2}+2\times10^{1}+1\times10^{0}+2\times10^{-1})_3$$
$$= (1\times11^{10}+10\times11^{1}+1\times11^{0}+10\times11^{-1})_2$$
$$= (1001+110+1+0.101010\cdots)_2$$
$$= (10000.101010\cdots)_2$$

(计算是按二进制计算规则进行的)

例3：将$(1234)_{10}$转换为十六进制。

$$(1234)_{10} = (1\times10^{3}+2\times10^{2}+3\times10^{1}+4\times10^{0})_{10}$$
$$= (1\times A^{3}+2\times A^{2}+3\times A^{1}+4\times A^{0})_{16}$$
$$= 3E8+C8+1E+4 = (4D2)_{16}$$ 列出计算过程

2. 数制的相互转换

多项式替代法的转换步骤，归纳如下：

$$N_\alpha = (A_{n-1}A_{n-2}\cdots A_1A_0.A_{-1}A_{-2}\cdots A_{-m})$$

$$= (A_{n-1}\times 10^{n-1} + A_{n-2}\times 10^{n-2} + \cdots + A_1\times 10^{1} + A_0$$

$$\times 10^{0} + A_{-1}\times 10^{-1} + A_{-2}\times 10^{-2} + \cdots + A_{-m}\times 10^{-m})$$

将α进制下的A_i、10 转换成β进制下的数

$$= (B_{n-1}\times \gamma^{n-1} + B_{n-2}\times \gamma^{n-2} + \cdots + B_1\times \gamma^{1}$$

$$+ B_0\times \gamma^{0} + B_{-1}\times \gamma^{-1} + B_{-2}\times \gamma^{-2} + \cdots$$

$$+ B_{-m}\times \gamma^{-m}) = N_\beta$$

其中：$(A_i)_\alpha = (B_i)_\beta$，$(10)_\alpha = (\gamma)_\beta$，则：$\gamma = \alpha_\beta$

α为任意进制而β为十进制时，用此方法进行转换。注意：多项式替代法是在β进制下完成N_α到N_β的转换的，因此，要求熟悉β进制的算术运算规则。

2.2 基数乘除法

要点：在α进制下完成 $N_\alpha \rightarrow N_\beta$ 的转换

(1) 整数部分转换用基数除法；　　(2) 小数部分转换用基数乘法

① 整数转换(基数除法)

例1 将十进制的179 转换成二进制数。　γ=2　（？）

$179 \div 2 = 89$　余1…… $c_0 \rightarrow b_0$

$89 \div 2 = 44$　余1…… $c_1 \rightarrow b_1$

$44 \div 2 = 22$　余0…… $c_2 \rightarrow b_2$

$22 \div 2 = 11$　余0…… $c_3 \rightarrow b_3$

$11 \div 2 = 5$　余1…… $c_4 \rightarrow b_4$

$5 \div 2 = 2$　余1…… $c_5 \rightarrow b_5$

$2 \div 2 = 1$　余0…… $c_6 \rightarrow b_6$

$1 \div 2 = 0$　余1…… $c_7 \rightarrow b_7$

即 $(179)_{10} = (1011\ \ 0011)_2$

2. 数制的相互转换

例2 将十进制的3417 转换成十六进制数。 γ =16

16 | 3417 余9 …… $c_0 \rightarrow b_0$

16 | 213 余5 …… $c_1 \rightarrow b_1$

16 | 13 余13 … $c_2 \rightarrow b_2$ …… D

即 $(3417)_{10} = (D59)_{16}$ 注意：次序

列竖式计算

例3 将二进制的1011 转换成三进制数。 γ=11

11 | 1011 余10 … $c_0 \rightarrow b_0$ …… 2

11 | 11 余0 …… $c_1 \rightarrow b_1$

11 | 1 余1 …… $c_2 \rightarrow b_2$

即 $(1011)_2 = (102)_3$

2. 数制的相互转换

整数转换(基数除法)小结

设: $N_\alpha = N_\beta = (b_{n-1}b_{n-2}\cdots b_1 b_0)_\beta \quad (0 \leqslant b_i \leqslant \beta - 1)$

$= (b_{n-1}\times 10^{n-1} + b_{n-2}\times 10^{n-2} + \cdots + b_1 \times 10^1 + b_0 \times 10^0)_\beta$

将b_i、10转换成α进制下的数，则：

$N_\alpha = (c_{n-1}\times \gamma^{n-1} + c_{n-2}\times \gamma^{n-2} + \cdots + c_1 \times \gamma^1 + c_0 \times \gamma^0), \quad (0 \leqslant c_i \leqslant \alpha - 1)$

$= (((\cdots((c_{n-1})\gamma + c_{n-2})\gamma + \cdots)\gamma + c_1)\gamma + c_0)_\alpha$

其中：$(b_i)_\beta = (c_i)_\alpha$，$(10)_\beta = (\gamma)_\alpha$，则：$\gamma = \beta_\alpha$

在α进制下，将该数除以γ，则余数分别为：$c_0, c_1, \cdots, c_{n-1}$，

再把$c_0, c_1, \cdots, c_{n-1}$转化为β进制下的：$b_0, b_1, \cdots, b_{n-1}$

②小数转换(基数乘法)

例1： 将$(0.4321)_{10}$转换成十六进制数。 $\gamma=16$，N = 0.4321

$0.4321\times16 = 6.9136$　得：　$D_{-1} = 6$ (6)

$0.9136\times16 = 14.6176$　　$D_{-2} = 14$(E)

$0.6176\times16 = 9.8816$　　$D_{-3} = 9$ (9)

$0.8816\times16 = 14.1056$　　$D_{-4} = 14$(E)

即　$(0.4321)_{10} \approx (0.6E9E)_{16}$

2. 数制的相互转换

例2： 将 $(0.375)_{10}$ 转换成二进制数。 γ=2

$$
\begin{array}{l}
0 \ . \ 3\ 7\ 5 \\
\times \qquad 2 \\
\hline
[0] . 7\ 5\ 0 \ldots\ldots \quad D_{-1} = 0 \text{ 转化为β进制下的 } C_{-1} \\
\times \qquad 2 \\
\hline
[1] . 5\ 0\ 0 \ldots\ldots \quad D_{-2} = 1 \text{转化为β进制下的 } C_{-2} \\
\times \qquad 2 \\
\hline
[1] . 0\ 0\ 0 \ldots\ldots \quad D_{-3} = 1 \text{转化为β进制下的 } C_{-3}
\end{array}
$$

即 $(0.375)_{10} = (0.011)_2$ 注意：次序

注意：

在转换中，若小数部分最终为零，则表明此数是精确转换，如例2；但若小数部分为循环小数或无限不循环小数，则表明此数不是精确转换，$(N)_\alpha$为$(N)_\beta$的近似值，即转换有误差。

2. 数制的相互转换

小数转换(基数乘法)小结

设：$N_\alpha = N_\beta$ $= (0.C_{-1}C_{-2}\ldots C_{-m})_\beta$ $= (C_{-1}\times 10^{-1} + C_{-2}\times 10^{-2} + \ldots + C_{-m}\times 10^{-m})_\beta$ $(0\le C_i \le \beta - 1)$	将 C_i、10转换成α进制下的数，则 $N_\alpha = (D_{-1}\times \gamma^{-1} + D_{-2}\times \gamma^{-2} + \ldots + D_{-m}\times \gamma^{-m})_\alpha$ 其中：$(C_i)_\beta = (D_i)_\alpha$，$(10)_\beta = (\gamma)_\alpha$，则：$\gamma = \beta_\alpha$ $(0\le D_i \le \alpha - 1)$

在α进制下，将该数乘以γ得：$D_{-1} + D_{-2}\times \gamma^{-1} + \cdots + D_{-m}\times \gamma^{-m+1}$

整数部分即 D_{-1}，再转化D_{-1}为β进制下的 C_{-1}

再乘以γ得：$D_{-2} + D_{-3}\times \gamma^{-1} + \cdots + D_{-m}\times \gamma^{-m+2}$

整数部分即 D_{-2}，再转化为β进制下的 C_{-2}

直至$D_{-m}\times \gamma^{-1}$时，再乘以γ得D_{-m}，得到第m位D_{-m}，再转化为β进制下的C_{-m}，则转换结束。

2.3 任意两种进制之间的转换：$N_\alpha \rightarrow N_\beta$

① 若熟悉β进制的运算规则，则采用多项式替代法完成转换。**(要求熟悉单个数字字符的转换)**

② 若熟悉α进制的运算规则，则采用基数乘除法完成转换；

③ 不熟悉α、β进制的运算规则，**可利用十进制作为转换桥梁。** $N_\alpha \rightarrow N_{10} \rightarrow N_\beta$

例 将$(1023.231)_4$转换成五进制数。

第一步：$(1023.231)_4$ 多项式替代法 $(\ ?\ )_{10}$

$$(1023.231)_4$$

$$= (1\times10^3+0\times10^2+2\times10^1+3\times10^0+2\times10^{-1}+3\times10^{-2}+1\times10^{-3})_4$$

$$= (1\times4^3+0\times4^2+2\times4^1+3\times4^0+2\times4^{-1}+3\times4^{-2}+1\times4^{-3})_{10}$$

$$= 64+0+8+3+0.5+0.1875+0.015625$$

$$= 75.703125$$ (如果直接转换为5进制，如何进行？试一试?)

2. 数制的相互转换

第二步： $(75.703125)_{10}$ —基数乘除法→ $(\ ?\ )_5$

整数部分与小数部分分别转换。

整数部分

5 | 7 5 …… $b_0=0$

5 | 1 5 …… $b_1=0$

5 | 3 …… $b_2=3$

0

小数部分

0 . 703125

× 5

[3] . 515626 …… $C_{-1}=3$

× 5

[2] . 578125 …… $C_{-2}=2$

× 5

[2] . 890625 …… $C_{-3}=2$

× 5

[4] . 453125 …… $C_{-4}=4$

即 $(75.703125)_{10}\approx(300.3224)_5$

∴ $(1023.231)_4\approx(300.3224)_5$

2.4 直接转换法

二进制数是计算机内部真正使用的数，但由于它的表示既**易出错**也**不易交流**，故常常用八进制或十六进制的形式表示。

二进制*Binary*，简称B，如 $(10)_2=(10)_B$；**八进制*Octal*，简称O**，如 $(10)_8=(10)_O$；

十六进制*Hexadecimal*，简称H，如 $(10)_{16}=(10)_H$。

转换： $N_\alpha \rightarrow N_\beta$

当基数α、β是2的幂次方时，可以进行直接转换。基数为 2^k 的进位制是将一个k位二进制字符串用一位数字字符表示，如$(5)_8=(101)_2$、$(5)_{16}=(0101)_2$

因此，用划分相应字符串的方法实现基数为 2^k 的进位制与二进制之间的直接转换。

注意：从小数点位置向左右划分，不足K位补0。

2. 数制的相互转换

基数为2与基数为2^k的关系：

R=2	R=4	R=8	R=16
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	10	4	4
101	11	5	5
110	12	6	6
111	13	7	7
1000	20	10	8
1001	21	11	9
1010	22	12	A
1011	23	13	B

R=2	R=4	R=8	R=16
1100	30	14	C
1101	31	15	D
1110	32	16	E
1111	33	17	F
10000	100	20	10
10001	101	21	11
10010	102	22	12
10011	103	23	13
10100	110	24	14
10101	111	25	15
10110	112	26	16
…	…	…	…

2. 数制的相互转换

例1 将$(1111\ 1010.\ 0111)_2$转换为八进制数。

二进制数 011 111 010 . 011 100

八进制数 3 7 2 . 3 4

即 $(1111\ 1010.0111)_2 = (372.34)_8$

例2 将$(AF.16C)_{16}$转换为八进制数。

十六进制数 A F . 1 6 C

二进制数 1010 1111 . 0001 0110 1100

八进制数 2 5 7 . 0 5 5 4

即 $(AF.16C)_{16} = (257.0554)_8$

2.5 数制转换时小数位数的确定

(1) 机器字长确定小数位数； (2) 给定小数位数； (4)转换前后数的精度相同。

例：将$(0.4321)_{10}$转换成十六进制时，小数位数应取几位?

设: α进制的小数有k位，转换成β进制后，至少具有相同精度的小数是 j 位，

则：$(0.1)_{\alpha}^{k} \geq (0.1)_{\beta}^{j}$

十进制中可表示为：$(\frac{1}{\alpha})^k \geq (\frac{1}{\beta})^j$，即：$\alpha^k \leq \beta^j$

两边取对数：$log_{\alpha}^{k} \leq log_{\beta}^{j}$，即 $k \lg\alpha \leq j \lg\beta$

$j \geq k \lg\alpha / \lg\beta = k \lg_{\alpha}^{\beta}$ （j 取最小的整数）

则 j 应满足不等式的整数：

$$k\frac{\log\alpha}{\log\beta} \leq j < k\frac{\log\alpha}{\log\beta} + 1$$

j 应满足下列不等式：

$$4\frac{\log 10}{\log 16} \leq j < 4\frac{\log 10}{\log 16} + 1$$

即：$3.320 \leq j < 4.320$

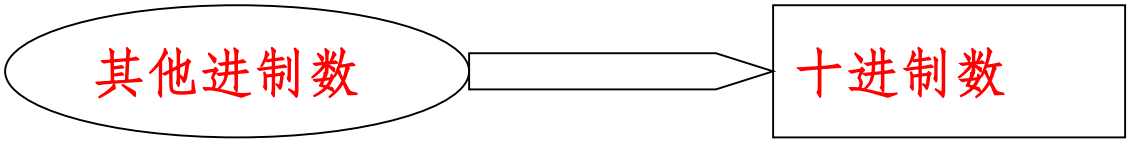
所以，小数位数应取4位。

即 $(0.4321)_{10} \approx (0.6E9E)_{16}$

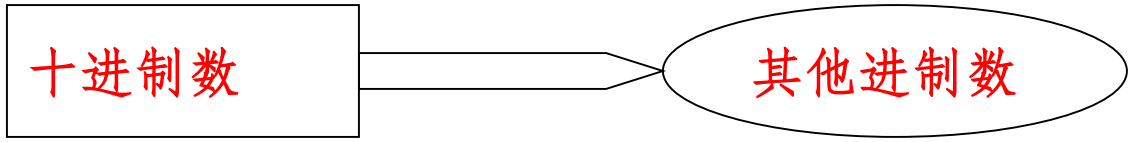
2. 数制的相互转换

总结

1. 多项式替代法

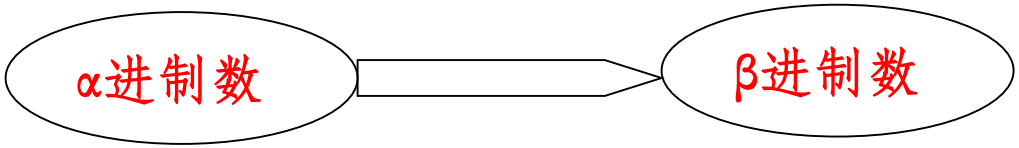


2. 基数乘除法（适用于十进制数转换成其他进制数）



3. 任意两种进制之间的转换

4. 直接转换法（α，β都是2^k）



数制与编码

1、数制

- 十进位计数制的特点、表示法，任意进制数
- 任意数制的特点、表示法、运算规则及相互关系

2、数制的相互转换

- 多项式替代法，基数乘除法，桥梁法
- 直接转换法，小数位置的确定

3、数的表示及其运算

- 小数点的表示，带符号数的表示及其运算
- 十进制数的代码表示及其运算

4、可靠性编码

3.1 数的小数点的表示

小数点的位置在计算机中是怎样表示的?

小数点在机器中并不存在，它只是人们约定的一个位置。 计算机中表示小数点位置的方法通常有两种： 一种是定点表示法；另一种是浮点表示法。

(1) 定点表示法

在计算机中数的小数点位置是固定的，一般固定在数的最高位之前或数的最低位之后。
常用方法：定点小数表示法；定点整数表示法。通常，小数点均约定在最高位之前。

但实际处理的数可能既有整数部分，又有小数部分。这就需要选取比例因子。

假设某计算机字长8位，规定最高位用来表示数的正、负号，其余7位表示数值。

3. 数的表示及其运算

现有一组数：**1101.01**，**-100.111**和**1110.1**，若用定点整数表示，可选取比例因子2^3，并用此比例因子乘上述各数，可得：

$1101.01 \times 2^3 = 110\ 1010$

$-100.111 \times 2^3 = -010\ 0111$

$1110.1 \times 2^3 = 111\ 0100$

它们在机器中的表示形式为：

0 1 1 0 1 0 1 0

1 0 1 0 0 1 1 1(原码)

0 1 1 1 0 1 0 0

若要用定点小数表示，则需提取比例因子2^{-4}，可得

$1101.01 \times 2^{-4} = 0.110101$

$-100.111 \times 2^{-4} = -0.0100111$

$1110.1 \times 2^{-4} = 0.11101$

它们在机器中的表示形式为：

0 1 1 0 1 0 1 0

1 0 1 0 0 1 1 1

0 1 1 1 0 1 0 0

（2）数的浮点表示法

计算机中数的小数点位置不是固定的，而是浮动的。因而，计算机必须表示出小数点位置的浮动情况。

在普通数学中，有“记阶表示法”(也称“科学表示法”)。一般说来，十进制数N可表示为: $N=10^J \times S$,其中，J是整数，称为阶码；S为小数，称为尾数。显然，只要有J和S，即可表示出N的值。

例：(十进制数):

2.537可表示为： $2.537 = 10^1 \times 0.253\ 7 = 10^2 \times 0.025\ 37$

0.002 537可表示为： $0.002\ 537=10^{-1} \times 0.025\ 37=10^{-2} \times 0.253\ 7$

同样，二进制数也可用这种方法表示，基数是十进制中的2，$(10)_2=(2)_{10}$。

例如**: 二进制数101.1和10.11可表示为**

$101.1=(10)^{11} \times 0.1011$ 和 $10.11=(10)^{10} \times 0.1011$

因此，这两个二进制数可用阶码和尾数表示为

101.1—>11（阶码），　0.1011

10.11—>10（阶码），　0.1011

不难看出，其尾数完全相同，仅阶码不同。**这种用阶码和尾数表示数的方法就是数的浮点表示法。**

表示浮点数时，需将一个字长划分为两部分，其中一部分表示阶码，另一部分表示尾数，其高位都定为符号位。

例如: 规定某一16位字长的前5位表示阶码的符号及数值，后11位表示尾数的符号及数值，如下所示：

数101.1 和10.11 的实际表示形式为

0　0011　0　1011000000

0　0010　0　1011000000

3.2 带符号数的代码表示及其运算

带符号数	真值	符号位	数值位
x	＋5	＋	5
y	－7	－	7

带符号（进位计数制）数的真值表示:

如：$(-111)_2$是带符号（二进制）数的真值表示

带符号(二进制)数的代码表示是指（在计算机中）：

带符号数的数值位和符号位都用 统一的代码形式，即仅取 0 和 1 两种数字符号表示，并且数值位为二进制数。 有三种代码表示：原码、反码和补码。

https://www.cnblogs.com/youxin/archive/2012/07/03/2575504.html

3. 数的表示及其运算

(1) 原码 (符号-数值表示法)

<0 时 $2^n + x$（x为绝对值，当 $2^n - x$ 时，x含符号）

原码的形成规则：

真值x	符号位S	数值位(二进制数)
原码$[x]_{原}$	+→0, -→1	不变,不变

例1 用8位二进制代码表示的原码

$x = +5$ $[x]_{原} = 0\ 000\ 0101$ $y = -7$ $[y]_{原} = 1\ 000\ 0111$

例2 由原码写出所对应的十进制数值：

$[x]_{原} = 01010101$ $x = +85$; $[x]_{原} = 11010101$ $x = -85$;

$[x]_{原} = 01111111$ $x = +127$; $[x]_{原} = 11111111$ $x = -127$;

$[x]_{原} = 00000000$ $x = +0$; $[x]_{原} = 10000000$ $x = -0$;

(2) 反码 Negative Number (对“1”) <0 时 $(2^{(n+1)}-1)-x$

反码的形成规则：

真值：X	符号位：S	数值位：(二进制数)
反码 $[x]_{反}$	＋→0 － →1	不变 按位取反

例 用8位二进制代码表示的原码、反码

$x = +5$ $[x]_{原} = 0\ 000\ 0101$ $[x]_{反} = 0\ 000\ 0101$

$y = -7$ $[y]_{原} = 1\ 000\ 0111$ $[y]_{反} = 1\ 111\ 1000$

$[z]_{反} = 0\ 000\ 0000$ $[z]_{反} = 1\ 111\ 1111$ $z = 0$

反码的运算规则： $[z]_{反} = [x]_{反} + [y]_{反}$

- 把符号位看作一位数值位，一律按加法规则来处理，所得结果的符号位也就是正确结果的符号。
- 当符号位产生进位时，将产生的进位要加到数值位的最低位。

(3) 补码 <0 时 $2^{(n+1)} - x$

补码的形成规则：

真值x	符号位S	数值位(二进制数)
补码 $[x]_{补}$	＋→0 － →1	不变 按位取反+1

例 用8位二进制代码表示的原码、补码

x = ＋5	$[x]_{原}$ = 0 000 0101	$[x]_{补}$ = 0 000 0101
y = －7	$[y]_{原}$ = 1 000 0111	$[y]_{补}$ = 1 111 1001
z = 0	$[y]_{原}$ = 0 000 0000	$[z]_{补}$ = 0 000 0000
Z=-128	$[z]_{补}$ = 1 000 0000	

补码的运算规则： [z]补= [x]补＋ [y]补

- 把符号位看作一位数值位，一律按加法规则来处理，所得结果的符号位也就是正确结果的符号。
- 当符号位产生进位时，将产生的进位丢掉。

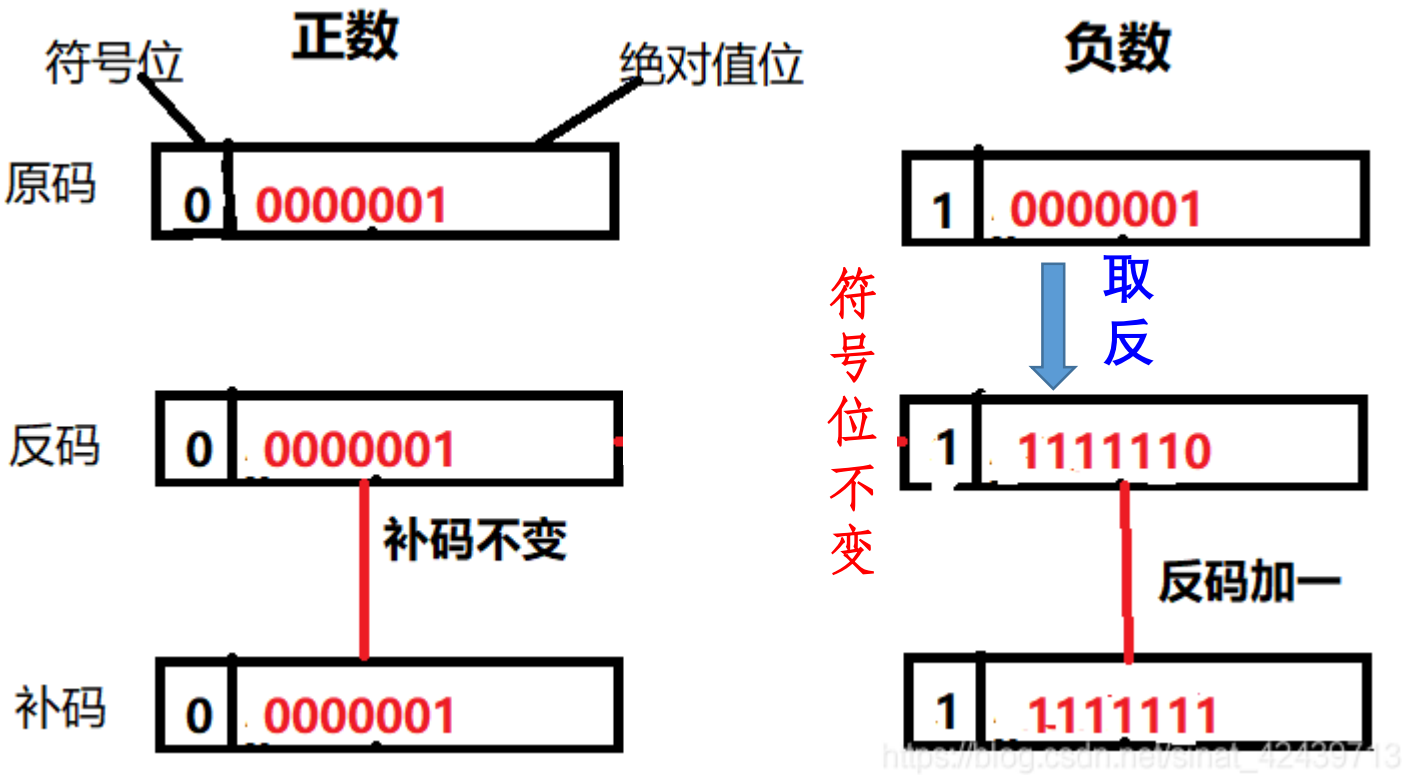
3. 数的表示及其运算

二进制代码	无符号数 x	$[x]_{原}$	$[x]_{反}$	$[x]_{补}$
0000	0	+0	+0	+0
0001	1	+1	+1	+1
0010	2	+2	+2	+2
0011	3	+3	+3	+3
0100	4	+4	+4	+4
0101	5	+5	+5	+5
0110	6	+6	+6	+6
0111	7	+7	+7	+7
1000	8	−0	−7	−8
1001	9	−1	−6	−7
1010	10	−2	−5	−6
1011	11	−3	−4	−5
1100	12	−4	−3	−4
1101	13	−5	−2	−3
1110	14	−6	−1	−2
1111	15	−7	−0	−1

试分析此表，你会有什么结论?

0111 →1000

原码、补码、反码的关系


正数
符号位
绝对值位
原码
0
0000001
反码
0
0000001
补码不变
补码
0
0000001
负数
1
0000001
取反
符号位不变
1
1111110
反码加一
1
1111111

数制与编码

1、 原码

原码是符号位加上真值的绝对值，即用第一位表示符号位，其余位表示值。

如果是8位二进制： [+1]原 = 0000 0001 [-1]原 = 0000 0001

第一位是符号位，所以8位二进制数的取值范围是：[1111 1111 , 0111 1111]即[-127 , +127]。 原码是人脑最容易理解和计算的表达方式。

计算十进制的表达式: 1-1=0

1 - 1 = 1 + (-1) = [00000001]原 + [10000001]原 =[10000010]原 = -2

如果用原码表示, 让符号位也参与计算, 显然对于减法来说, 结果是不正确的。这也就是为何计算机内部不使用原码表示一个数。

为了解决原码做减法的问题，出现了反码:

2、 反码

反码的计算方法：正数的反码是其本身，负数的反码是在其原码的基础上，符号位不变，其余各位取反。

[+1] = [0000 0001]原 = [0000 0001]反

[-1] = [1000 0001]原 = [1111 1110]反

如果一个反码表示的是负数，无法直观的看出来它的数值，通常要将其转换成原码再计算。

1 - 1 = 1 + (-1) = [0000 0001]原 + [1000 0001]原

= [00000001]反 + [1111 1110]反 = [1111 1111]反

= [1000 0000]原 = -0

发现用反码计算减法,结果的真值部分是正确的。 而唯一的问题其实就出现在“0”这个特殊的数值上。虽然人们理解上+0和-0是一样的,但是0带符号是没有任何意义的,而且会有[00000000]原和[1000 0000]原两个编码表示0。

3、 补码

补码的计算方法：正数的补码是其本身，负数的补码是在原码的基础上，符号位不变，其余各位取反，最后+1。（即在反码的基础上+1）

[+1] = [0000 0001]原 = [0000 0001]反 = [0000 0001]补

[-1] = [1000 0001]原 = [1111 1110]反 = [1111 1111]补

对于负数，补码表示方式也是无法直接看出其数值的。通常也需要转换成原码再计算其数值。

数制与编码

补码的出现,解决了0的符号以及两个编码的问题:

1-1 = 1 + (-1) = [0000 0001]原 + [1000 0001]原

= [0000 0001]补 + [1111 1111]补

= [0000 0000]补

= [0000 0000]原

这样0用[0000 0000]表示,反码中出现-0的问题不存在了。

且可以用[1000 0000]表示-128:

(-1) + (-127) = [1000 0001]原 + [1111 1111]原

= [11111111]补 + [1000 0001]补

= [1000 0000]补

-1+（-127）的结果应该是-128,在用补码运算的结果中,[1000 0000]补就是-128.

数制与编码

实际上是使用以前的-0的补码来表示-128, 所以-128并没有原码和反码表示.

(-128的补码表示[1000 0000]补，算出来的原码是[0000 0000]原, 这是不正确的)

使用补码, 不仅仅修复了0的符号以及存在两个编码的问题, 而且还能够多表示一个最低数.

这就是为什么8位二进制, 使用原码或反码表示的范围为[-127, +127], 而使用补码表示的范围为[-128, 127].

原码, 反码, 补码 再深入： 模和同余概念

https://blog.csdn.net/zl10086111/article/details/80907428?utm_medium=distribute.pc_relevant.none-task-blog-2~default~baidujs_title~default-9.no_search_link&spm=1001.2101.3001.4242

原码运算

1. 加法运算： $[z]_{原} = [x]_{原} + [y]_{原}$

① 当[x]原 、 [y]原 的符号 S 相同时，则[z]原的符号同[x]原的符号，[z]原的数值为两加数的和;

② 当[x]原 、 [y]原 的符号 S 相异时，先判断两数的绝对值的大小，[z]原的符号同大数的原码符号，[z]原的数值为较大数的绝对值与较小数的差的绝对值。

2. 减法运算: $[z]_{原} = [x]_{原} - [y]_{原}$

3. 数的表示及其运算

反码与补码的运算规则：

反码　　$[x+y]_{反}=[x]_{反}+[y]_{反}$；

　　$[x-y]_{反}=[x]_{反}+[-y]_{反}$；

补码　　$[x+y]_{补}=[x]_{补}+[y]_{补}$；

　　$[x-y]_{补}=[x]_{补}+[-y]_{补}$；

① **减法运算**：按加法运算完成；

② **符号位S的处理**：S被看成一位数码，并与数值位一样，按同样的加法规则进行处理，所得结果的符号位即是正确结果的符号位。

③ **进位的处理**：反码运算时符号位产生的进位要加到和数 的最低位去。补码运算时符号位产生的进位要丢掉。

(4) 带符号数的加、减运算

例1　求 z = x − y，　其中 x =+1010，y =+0011

(1) 原码运算：　$[x]_{原}$= 01010　　$[y]_{原}$= 00011

∵ x 绝对值 > y 绝对值 *(假如x=+0011，y=−1010)*

∴　$[z]_{原}$=[01010−00011]$_{原}$= 00111　　z = +0111

(2) 补码运算：　$[x]_{补}$= 01010

$[-y]_{补}$= [− 0011]$_{补}$= 11101

∵　01010

+11101

丢掉←1 00111

∴ $[z]_{补}$= 00111

z = + 0111

例1　求 z = x－y，　其中 x =＋1010，y =＋0011

(3) 反码运算：　$[x]_{反}$= 01010

$[-y]_{反}$= [－ 0011]$_{反}$= 11100

∵

010　10

＋111　00

1 001　10

＋　　1

00111

∴ $[z]_{反}$=　00111

z = ＋ 0111

又：　十进制-1-3=？用反码计算

3.3 十进制数的常用编码（代码）

十进制数的编码表示特点：

（1）既具有二进制数的形式，又具有十进制数的特点，从4 位二进制数16种代码中,选择10种来表示0~9个数码的方案有很多种。每种方案产生一种BCD码。

（2）可按位直接相互转换；

（3）可按位直接运算（要修正）。

例：

十进制整数	8421码	2421码	余3码
0	0000	0000	0011
1	0001	0001	0100
2	0010	0010	0101
3	0011	0011	0110
4	0100	0100	0111
5	0101	1011	1000
6	0110	1100	1001
7	0111	1101	1010
8	1000	1110	1011
9	1001	1111	1100

3. 数的表示及其运算

设：代码为 $A_3A_2A_1A_0$，代码运算若直接按二进制数运算，则运算结果要修正。

代码	对应的十进制数值	编码直接按位转换
8421码	有权码 (*Weighted code*) $8A_3+4A_2+2A_1+1A_0$	$(13)_{10}=(0001\ 0011)_{BCD}$ $(1\ 0111\ 0101\ 0000)_{BCD}=(1750)_{10}$
2421码	有权码、对9自补码 $2A_3+4A_2+2A_1+1A_0$	$(13)_{10}=(0001\ 0011)_{2421}$ $(1\ 1101\ 1011\ 0000)_{2421}=(1750)_{10}$
余3码	无权码、对9自补码 $8A_3+4A_2+2A_1+1A$ -0011 (?)	$(13)_{10}=(0100\ 0110)_{余3}$ $(100\ 1010\ 1000\ 0011)_{余3}=(1750)_{10}$

求BCD代码表示的十进制数

对于有权**BCD**码，可以根据位权展开求得所代表的十进制数。例如：

$$[0111]_{8421BCD}=0\times 8+1\times 4+1\times 2+1\times 1=(7)_D$$

对于一个多位的十进制数，需要有与十进制位数相同的几组**BCD**代码来表示。例如：

$$(463.5)_{10}=\left[\underset{4}{\underline{0100}}\ \underset{6}{\underline{0110}}\ \underset{3}{\underline{0011}}.\ \underset{5}{\underline{0101}}\right]_{8421BCD}$$

不能省略!

$$(863.2)_{10}=\left[\underset{8}{\underline{1110}}\ \underset{6}{\underline{1100}}\ \underset{3}{\underline{0011}}.\ \underset{2}{\underline{0010}}\right]_{2421BCD}$$

不能省略!

三种十进制数代码的分布图

四位二进制代码	8421码	2421码	余3码
0000	0000 0	0000 0	0000 非码区
0001	0001 1	0001 1	0001 非码区
0010	0010 2	0010 2	0010 非码区
0011	0011 3	0011 3	0011 0
0100	0100 4	0100 4	0100 1
0101	0101 5	0101 非码区	0101 2
0110	0110 6	0110 非码区	0110 3
0111	0111 7	0111 非码区	0111 4
1000	1000 8	1000 非码区	1000 5
1001	1001 9	1001 非码区	1001 6
1010	1010 非码区	1010 非码区	1010 7
1011	1011 非码区	1011 5	1011 8
1100	1100 非码区	1100 6	1100 9
1101	1101 非码区	1101 7	1101 非码区
1110	1110 非码区	1110 8	1110 非码区
1111	1111 非码区	1111 9	1111 非码区

3. 数的表示及其运算

ASCII 码（American Standard Code for Information Interchange，美国信息交换标准代码）

它共有128个代码，可以表示大、小写英文字母、十进制数、标点符号、运算符号、控制符号等，普遍用于计算机的键盘指令输入和数据等。

高四位 / 低四位		ASCII非打印控制字符										ASCII 打印字符												
		0000					0001					0010		0011		0100		0101		0110		0111		
		0					1					2		3		4		5		6		7		
		+进制	字符	ctrl	代码	字符解释	+进制	字符	ctrl	代码	字符解释	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	+进制	字符	ctrl
0000	0	0	BLANK NULL	^@	NUL	空	16	▶	^P	DLE	数据链路转意	32		48	0	64	@	80	P	96	`	112	p	
0001	1	1	☺	^A	SOH	头标开始	17	◀	^Q	DC1	设备控制 1	33	!	49	1	65	A	81	Q	97	a	113	q	
0010	2	2	☻	^B	STX	正文开始	18	↕	^R	DC2	设备控制 2	34	"	50	2	66	B	82	R	98	b	114	r	
0011	3	3	♥	^C	ETX	正文结束	19	‼	^S	DC3	设备控制 3	35	#	51	3	67	C	83	S	99	c	115	s	
0100	4	4	♦	^D	EOT	传输结束	20	¶	^T	DC4	设备控制 4	36	$	52	4	68	D	84	T	100	d	116	t	
0101	5	5	♣	^E	ENQ	查询	21	§	^U	NAK	反确认	37	%	53	5	69	E	85	U	101	e	117	u	
0110	6	6	♠	^F	ACK	确认	22	■	^V	SYN	同步空闲	38	&	54	6	70	F	86	V	102	f	118	v	
0111	7	7	•	^G	BEL	震铃	23	↨	^W	ETB	传输块结束	39	'	55	7	71	G	87	W	103	g	119	w	
1000	8	8	◘	^H	BS	退格	24	↑	^X	CAN	取消	40	(	56	8	72	H	88	X	104	h	120	x	
1001	9	9	○	^I	TAB	水平制表符	25	↓	^Y	EM	媒体结束	41	)	57	9	73	I	89	Y	105	i	121	y	
1010	A	10	◙	^J	LF	换行/新行	26	→	^Z	SUB	替换	42	*	58	:	74	J	90	Z	106	j	122	z	
1011	B	11	♂	^K	VT	竖直制表符	27	←	^[	ESC	转意	43	+	59	;	75	K	91	[	107	k	123	{	
1100	C	12	♀	^L	FF	换页/新页	28	∟	^\	FS	文件分隔符	44	,	60	<	76	L	92	\	108	l	124	|	
1101	D	13	♪	^M	CR	回车	29	↔	^]	GS	组分隔符	45	-	61	=	77	M	93	]	109	m	125	}	
1110	E	14	♫	^N	SO	移出	30	▲	^6	RS	记录分隔符	46	.	62	>	78	N	94	^	110	n	126	~	
1111	F	15	☼	^O	SI	移入	31	▼	^-	US	单元分隔符	47	/	63	?	79	O	95	_	111	o	127	Δ	^Back space

注：表中的ASCII字符可以用:ALT + "小键盘上的数字键" 输入

数制与编码

1、数制

- 十进位计数制的特点、表示法，任意进制数
- 任意数制的特点、表示法、运算规则及相互关系

2、数制的相互转换

- 多项式替代法，基数乘除法，桥梁法
- 直接转换法，小数位置的确定

3、数的表示方法及其运算

- 小数点的表示，带符号数的表示及其运算
- 十进制数的代码表示及其运算

4、可靠性编码

4. 可靠性编码

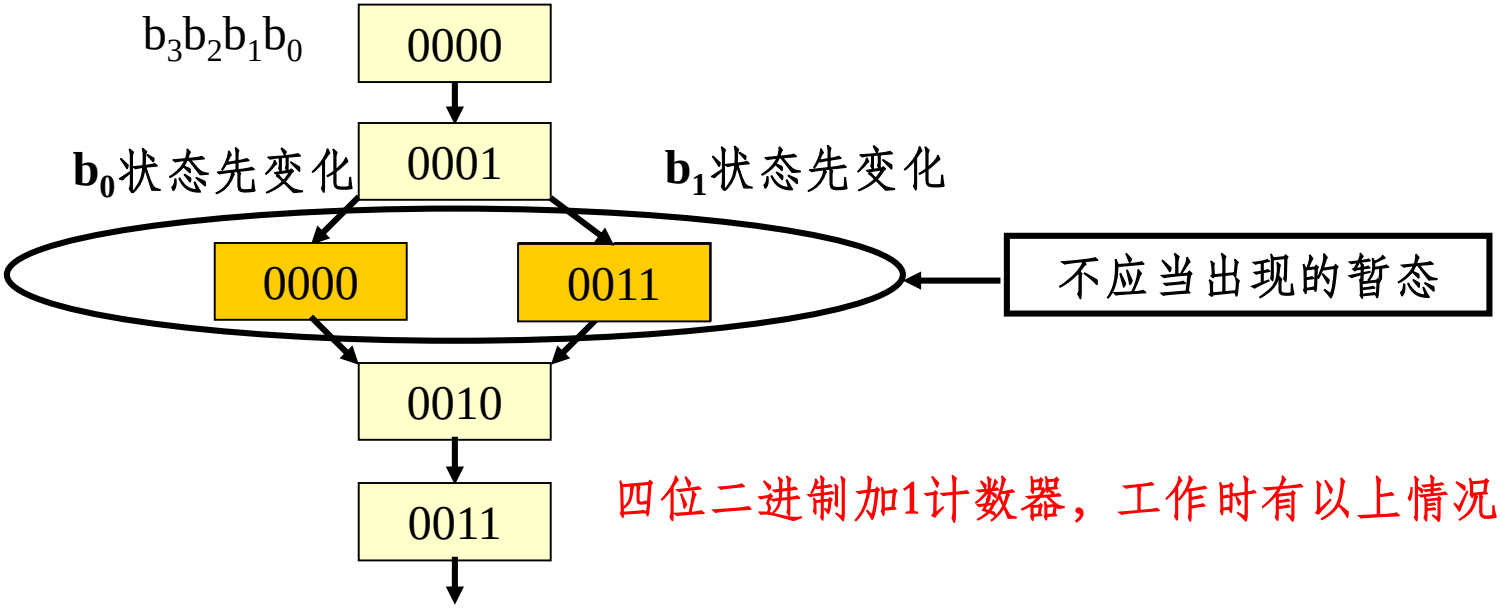
目的：解决代码在形成或传输过程中可能会发生的错误，提高系统的安全性

方法：使代码自身具有一种特征或能力

作用：不易出错；若出错时易发现错误； 出错时易查错且易纠错

4.1 格雷码（Gray)

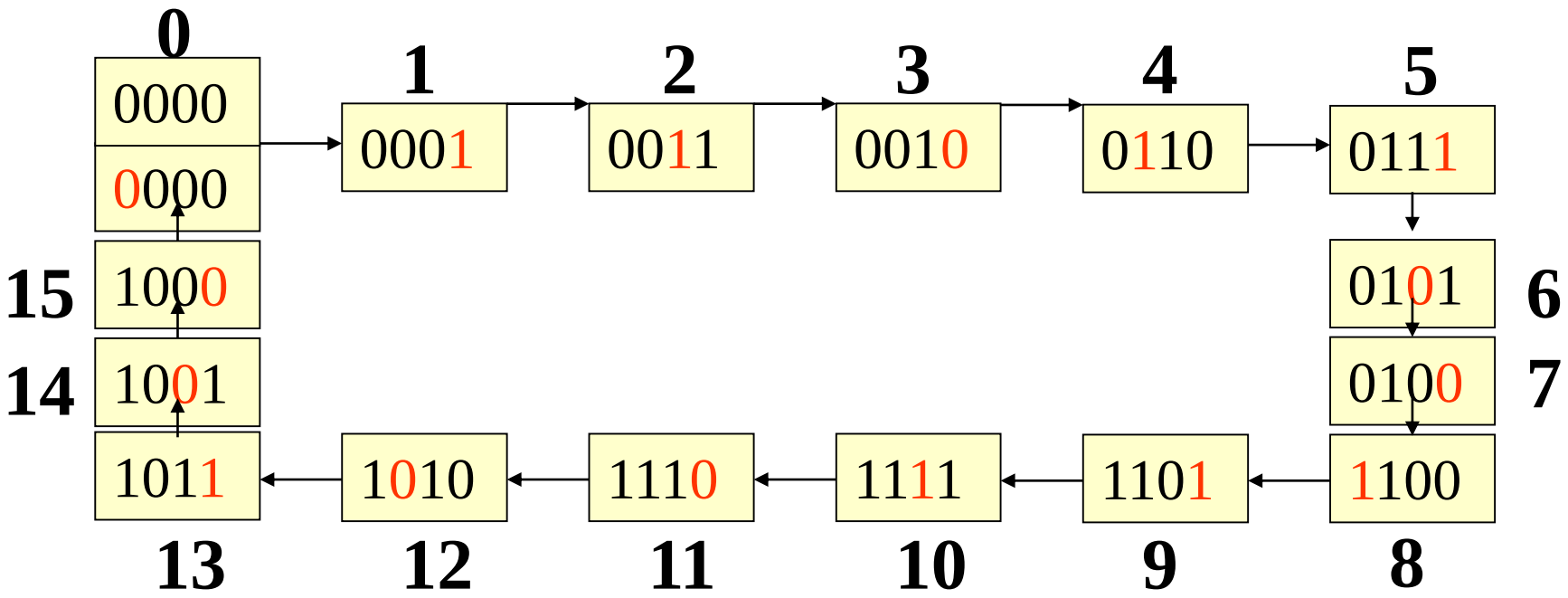
任意两个相邻数的代码只有一位二进制数不同，目的是解决代码生成时发生的错误。



四位二进制加1计数器，工作时有以上情况

4. 可靠性编码

当选用典型Gray码设计加1计数器时，就不会出现上述情况。如下：



4. 可靠性编码

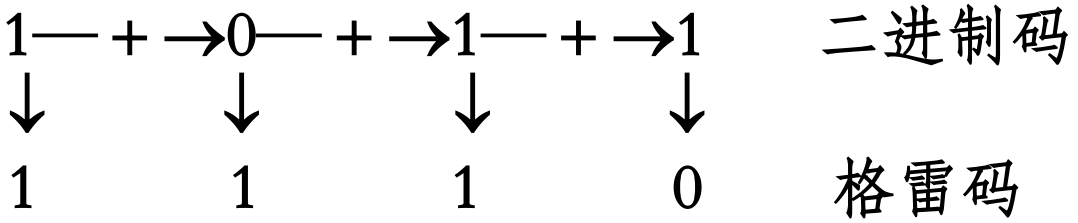
几种Gray码、步进码和二进制码对照表

十进制数	二进制数	典型Gray	十进制 Gray码(1)	十进制 Gray码(2)	步进码
0	0000	0000	0000	0000	00000
1	0001	0001	0001	0001	00001
2	0010	0011	0011	0011	00011
3	0011	0010	0010	0010	00111
4	0100	0110	0110	0110	01111
5	0101	0111	1110	0111	11111
6	0110	0101	1010	0101	11110
7	0111	0100	1011	0100	11100
8	1000	1100	1001	1100	11000
9	1001	1101	1000	1000	10000
10	1010	1111			
11	1011	1110			
12	1100	1010			
13	1101	1011			
14	1110	1001			
15	1111	1000			

4. 可靠性编码

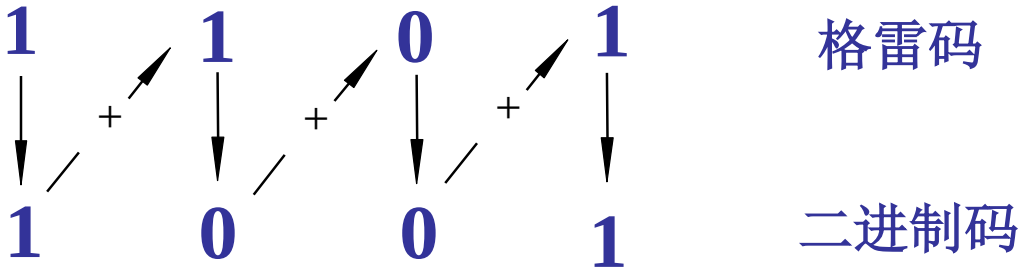
二进制码到格雷码的转换 (1011 B→1110 G)

(1) 格雷码的最高位（最左边）与二进制码的最高位相同。
(2) 从左到右，逐一将二进制码相邻的两位相加（舍去进位）,作为格雷码的下一位。



格雷码到二进制码的转换 (1101 G→1001 B)

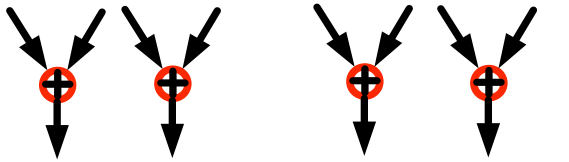
(1) 二进制码的最高位（最左边）与格雷码的最高位相同。
(2) 将产生的每一位二进制码，与下一位相邻的格雷码相加（舍去进位），作为二进制码的下一位。



4. 可靠性编码

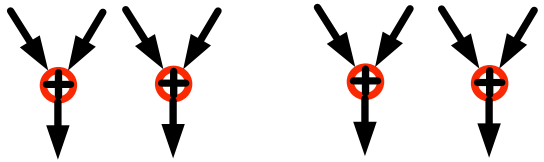
典型Gray码：由二进制码生成的，通过异或运算⊕完成： $G_i = B_{i1} \oplus B_i$ 。

设：二进制码 B　0　B_{n-1}　B_{n-2} … B_1　B_0



典型Gray码G_i:　G_{n-1}　G_{n-2} ……　G_1　G_2

例：二进制码 B　0　1　1 …　1　0



典型Gray码G_i:　1　0 ……　0　1

4. 可靠性编码

由典型Gray码也可以得到二进制码，如下：

设：典型Gray码　　$G_{n-1}\ G_{n-2}\ \cdots\ G_1\ G_0$

$\because G_{n-1}=B_{n-1}\oplus 0$　　$\therefore B_{n-1}=G_{n-1}$

$\because G_i=B_{i+1}\oplus B_i$，则等式两边同时$\oplus B_{i+1}$：

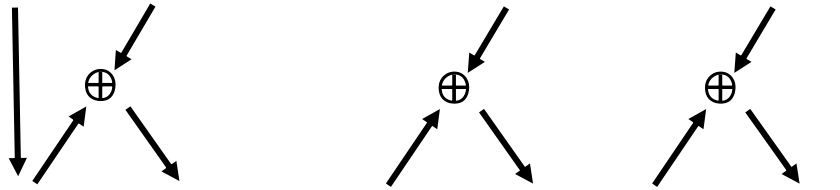
$$G_i\oplus B_{i+1}=B_{i+1}\oplus B_i\oplus B_{i+1}$$

$\therefore B_i=G_i\oplus B_{i+1}$

故　$B_{n-2}=G_{n-2}\oplus B_{n-1}\cdots\cdots,\ B_0=G_0\oplus B_1$

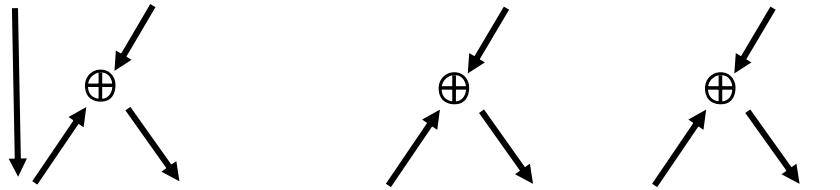
4. 可靠性编码

设：典型Gray码 G　G_{n-1}　G_{n-2} …　G_1　G_0



二进制码 B:　B_{n-1}　B_{n-2} ……　B_1　B_0

例：典型Gray码 G　1　1 …　1　0



二进制码 B:　1　0 ……　1　1

4. 可靠性编码

典型格雷码的特点：

所有对应于十进制数2^m-1 (m为正整数）的格雷码，都仅在m位上有 1，其他位都为0。

例：　　m = 1，$2^m-1=1$，　1的典型格雷码是0001；

　　m = 2，$2^m-1=3$，　3的典型格雷码是0010；

　　m = 3，$2^m-1=7$，　7的典型格雷码是0100；

　　m = 4，$2^m-1=15$，15的典型格雷码是1000。

这些数与 0 之间只有一位的差别，回到 0仍能保持一位差别的特点，所以称作循环码，特别适用做二进制码计数器。

十进制Gray码：代码特征应符合Gray码的要求，且十六选十。

使得9 回到 0 也只有一位差别，并保持循环码的特色，表中已列出了其中两种：

十进制Gray码(1)

十进制Gray码(2)

4.2 奇偶校验码 Parity Code

什么是奇偶校验：对数据传输正确性的一种校验方法。在数据传输前附加一位奇校验位，用来表示传输的数据中“1”的个数是奇数还是偶数，为奇数时，校验位置为“0”，否则置为“1”，用以保持数据的奇偶性不变。奇偶校验就是接收方用来验证发送方在传输过程中所传数据是否由于某些原因造成破坏。

奇校验：原始码流+校验位 总共有奇数个1： 1000 110（0）

偶校验：原始码流+校验位 总共有偶数个1： 1000 110（1）

例如，需要传输“1100 1110”，数据中含5个“1”，所以其奇校验位为“0”，同时把“1100 1110 0”传输给接收方。接收方收到数据后再一次计算奇偶性，“1100 1110 0”中仍然含有5个“1”，所以接收方计算出的奇校验位还是“0”，与发送方一致，表示在此次传输过程中未发生错误。

https://blog.csdn.net/greatwgb/article/details/8552652

https://zhuanlan.zhihu.com/p/26509678

4. 可靠性编码

原始码	奇校验 奇数个1	偶校验 偶数个1
1011000	10110000	10110001
1010000	10100001	10100000
0011010	00110100	00110101
0001000	00010000	00010001
0000000	00000001	00000000

4. 可靠性编码

奇偶校验码一个最为常见的应用场合就是ASCII码。

ASCII码占用一个字节，低7位是有效位，最高位用作奇偶校验。

举例：ASCII码 大写字母 A

十六进制 0x41

二进制 01000001 → 有效位为低7位

奇校验 11000001

偶校验 01000001

最高位是奇偶校验位

错误检测能力

例：ASCII码中的大写字母 A

奇校验 正确码流 1100 0001

错1位 1100 0011 变成了偶数个1，能检测出错误

错2位 1100 0010 变成了奇数个1，检测不出错误

错3位 1100 1010 变成了偶数个1，能检测出错误

偶校验 正确码流 01000001

错1位 0100 0011 变成了奇数个1，能检测出错误

错2位 0100 0010 变成了偶数个1，检测不出错误

错3位 0100 1010 变成了奇数个1，能检测出错误

奇偶校验码 Parity Code

校验码：信息位$B_{n-1\sim0}$　　校验位 P

偶校验：校验码P的取值使校验码中“1”的个数是偶数；

$$P_{偶} = B_{n-1} \oplus B_{n-2} \oplus \cdots \oplus B_1 \oplus B_0$$

奇校验：校验码P的取值使校验码中“1”的个数是奇数；

$$P_{奇} = B_{n-1} \oplus B_{n-2} \oplus \cdots \oplus B_1 \oplus B_0 \oplus 1$$

问题：① 奇偶校验码能发现单错，但不能对单错定位；

② 奇偶校验码不能发现双错。

对于问题①，或要求重新再发；或在信息传送中采用横向及纵向奇偶校验，即可对单错定位。

对于问题②，试想：为什么奇偶校验码不能发现双错？

4.3 海明校验码 Hamming codes

目的：不仅能检测出单错，还能校正单错

方法：增加校验位及相应的异或运算以四位信息位$B_4B_3B_2B_1$为例，在传输前生成它的海明校验码：

(1) 位序：

7	6	5	4	3	2	1
B_4	B_3	B_2	P_3	B_1	P_2	P_1

(2) 校验位的生成公式：

$$P_3 = B_4 \oplus B_3 \oplus B_2$$

$$P_2 = B_4 \oplus B_3 \oplus B_1$$

$$P_1 = B_4 \oplus B_2 \oplus B_1 \quad \text{偶校验}$$

https://blog.csdn.net/wlj323/article/details/48830819

4. 可靠性编码

“8421”海明码

位序	7	6	5	4	3	2	1
N.	B_4	B_3	B_2	P_3	B_1	P_2	P_1
0	0	0	0	0	0	0	0
1	0	0	0	0	1	1	1
2	0	0	1	1	0	0	1
3	0	0	1	1	1	1	0
4	0	1	0	1	0	1	0
5	0	1	0	1	1	0	1
6	0	1	1	0	0	1	1
7	0	1	1	0	1	0	0
8	1	0	0	1	0	1	1
9	1	0	0	1	1	0	0

4. 可靠性编码

对传输后的海明码进行检错和校错：

(3) 校验和：　　$S_3 = B_4 \oplus B_3 \oplus B_2 \oplus P_3$

$S_2 = B_4 \oplus B_3 \oplus B_1 \oplus P_2$

$S_1 = B_4 \oplus B_2 \oplus B_1 \oplus P_1$

① 当$S_3\,S_2\,S_1 = 0$时，接收到的信息是正确的；

② 当$1 \leqslant S_3\,S_2\,S_1 \leqslant 7$时，$S_3\,S_2\,S_1$所表示的二进制值便是出错的那一位的位序值。

例：接收到的海明码为：

7	6	5	4	3	2	1
B_4	B_3	B_2	P_3	B_1	P_2	P_1
0	0→1	0	1	0	1	0

则 $S_3\,S_2\,S_1$ =110，表示第6位(B_3)出错，改0为1。

4. 可靠性编码

出错表的确定

S_3 =	$B_4 \oplus$	$B_3 \oplus$	$B_2 \oplus$	P_3			
S_2 =	$B_4 \oplus$	B_3		$\oplus$	$B_1 \oplus$	P_2	
S_1 =	B_4	$\oplus$	B_2	$\oplus$	$B_1 \oplus$	P_1	
$S_3\,S_2\,S_1$	111	110	101	100	011	010	001	000
出错位序列	7	6	5	4	3	2	1	
出错位	B_4	B_3	B_2	P_3	B_1	P_2	P_1	

1. 每个校验位**P**必分布在$\mathbf{2^k}$位上，使其仅在一个校验和**S**中 出现；

2. 信息位B分布在非2^k位上，使其在一个以上的校验和S中 出现；

3. 若传送后海明码中的某一位出错，则将影响它所在的校验和$\mathbf{S_i}$，故能得到它的位序值，即可实现其单错的定位和校错。

4. 可靠性编码

设：信息位n位，校验位k位 则$(2^k-1)-k \geq n$ 或$(2^k-1) \geq n+k$ (?? n=5)

校验位数k	1	2	3	4	5	6	7	8
最大信息位数n	0	1	4	11	26	57	120	247
海明码位数 (2^k-1)	1	3	7	15	31	63	127	255

数制与编码

教材习题

- 1.2
- 1.4
- 1.7--1.8
- 1.12—1.14

下一节内容：

逻辑代数基础