

第2章 数制与编码

王冬青

济事楼410

同济大学软件学院

- 2. 1 进位记数制与数制转换
- 2. 2 带符号数的表示方法
- 2. 3 数的定点表示与浮点表示
- 2. 4 常用的其他编码

2.1 进位记数制与数制转换

2.1.1 进位记数制及其表示

1. **进位记数制**：按进位方式实现记数的一种规则，简称进位制。
 - 数码 d_i ：每位的数码值。
 - 基数 R ：表示某种进位制所具有的数字符号的个数。
 - 位权 R^i ：表示某种进位制的数中不同位置上数字的单位数值。

例如：十进制数：235.67

二进制数：110101.001

2. 两种表示方法

对于任意一个R进制数N:

➤ 并列表示法

$$(N)_R = (r_{n-1}r_{n-2}\dots r_1r_0 \cdot r_{-1}r_{-2}\dots r_{-m})_R$$

➤ 按权展开式（多项式表示法）

$$\begin{aligned}(N)_R &= r_{n-1}R^{n-1} + r_{n-2}R^{n-2} + \dots + r_1R^1 + r_0R^0 + r_{-1}R^{-1} + r_{-2}R^{-2} \\ &\quad + \dots + r_{-m}R^{-m} \\ &= \sum_{i=-m}^{n-1} r_i R^i\end{aligned}$$

n: 整数的位数 **m**: 小数的位数 **R**: 基数

r_i: R进制中各个数字符号, 即有 $r_i \in \{0, 1, 2, \dots, R-1\}$

3. 十进制(D)

- 基数：10
- 10个数字符号：0、1、2、3、4、5、6、7、8、9
- 运算规律：逢十进一，即： $9+1=10$ 。

4. 二进制(B)

- 基数：2
- 两个数字符号：0、1
- 运算规律：逢二进一，即： $1+1=(10)_2$ 。

4. 二进制（续）

➤ 特点

- 只有0和1两个数码
 - 任何具有两个稳定状态的元件都可以用来表示一位二进制数。
- 运算规则简单
 - 加法规则： $0+0=0$, $0+1=1+0=1$, $1+1=10$
 - 减法规则： $0-0=0$, $0-1=1$ （向高位借位）
 $1-0=1$, $1-1=0$
 - 乘法规则： $0\times 0=0$, $0\times 1=1\times 0=0$, $1\times 1=1$
 - 除法规则： $0\div 1=0$, $1\div 1=1$

5. 八进制(O)

- 基数：8
- 八个数字符号：0、1、2、3、4、5、6、7
- 运算规律：逢八进一，即： $7+1=(10)_8$ 。

6. 十六进制(H)

- 基数：16
- 16个数字符号：0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
- 运算规律：逢十六进一，即： $15+1=(10)_{16}$ 。

编程语言中常见表示方式：

1. 十进制整数：直接写出数字；
2. 十六进制整数：
以0x或0X开头，包含数字0-9以及字母A-F或a-f。
3. 二进制整数：以0b或0B开头，只包含0和1。
4. 八进制整数：以0开头，只包含0-7。

不同基数的进位制数

$R=10$	$R=2$	$R=8$	$R=16$
0	0	0	0
1	1	1	1
2	10	2	2
3	11	3	3
4	100	4	4
5	101	5	5
6	110	6	6
7	111	7	7
8	1000	10	8
9	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F
16	10000	20	10
17	10001	21	11
18	10010	22	12
19	10011	23	13
20	10100	24	14

2.1.2 数制转换

1. 直接转换法

➤ 二进制数与八进制数之间的转换

八进制	0	1	2	3	4	5	6	7
二进制	000	001	010	011	100	101	110	111

二进制数 \longrightarrow 八进制数

- 整数部分与小数部分同时进行转换
- 以小数点为中心，分别向左、向右，每3位分一组（不足3位补0） $2^3 = 8$
- 把每一组二进制数转换成八进制数

例1 将二进制数11010.1101转换为八进制数。

$$\begin{array}{ccccccc} 011 & 010 & . & 110 & 100 \\ 3 & 2 & . & 6 & 4 \end{array}$$

所以 $(11010.1101)_2 = (32.64)_8$

八进制数 \longrightarrow 二进制数

- 整数部分与小数部分同时进行转换
- 把八进制数中的每一位数分别转换成3位的二进制数

例2 将八进制数357.6转换为二进制数。

$$\begin{array}{ccccccc} 3 & 5 & 7 & . & 6 \\ 011 & 101 & 111 & . & 110 \end{array}$$

所以 $(357.6)_8 = (11101111.11)_2$

➤ 二进制数与十六进制数之间的转换

- 转换方法类似
- $2^4=16$ ，4位二进制数与一位十六进制数

十六进制	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
二进制	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1101	1110	1111

例3 将二进制数1010110110.110111转换为十六进制数。

0010 1011 0110 . 1101 1100
2 B 6 . D C

所以 $(1010110110.110111)_2 = (2B6.DC)_{16}$

例4 将十六进制数5D.6E转换为二进制数。

$$\begin{array}{ccccccc} 5 & & \text{D} & . & 6 & & \text{E} \\ 0101 & 1101 & . & 0110 & 1110 & & \end{array}$$

$$\text{所以 } (5\text{D}.6\text{E})_{16} = (1011101.0110111)_2$$

2. 多项式替代法

- 任意两个 α 、 β 进制数之间的转换
- **方法**：先将 α 进制的数在 α 进制中按权展开，然后替代成相应 β 进制中的数，最后在 β 进制中计算即可得 β 进制的数。

例5 将二进制数1101.101转换成十进制数。

先把二进制数的并列表示法展开成多项式表示法，则有

$$(1101.101)_2 = [1 \times (10)^{11} + 1 \times (10)^{10} + 0 \times (10)^1 + 1 \times (10)^0 + 1 \times (10)^{-1} + 0 \times (10)^{-10} + 1 \times (10)^{-11}]_2$$

再把等式右边的二进制数替代成十进制数，则得

$$(1101.101)_2 = [1 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 1 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}]_{10}$$

在十进制中计算等式右边之值，得

$$(1101.101)_2 = (8 + 4 + 1 + 0.5 + 0.125)_{10} = (13.625)_{10}$$

例6 $(123.4)_8 = (?)_{10}$

$$\begin{aligned}(123.4)_8 &= [1 \times (10)^2 + 2 \times (10)^1 + 3 \times (10)^0 + 4 \times (10)^{-1}]_8 \quad (\text{展开}) \\ &= (1 \times 8^2 + 2 \times 8^1 + 3 \times 8^0 + 4 \times 8^{-1})_{10} \quad (\text{替代}) \\ &= (64 + 16 + 3 + 0.5)_{10} \quad (\text{在十进制中计算}) \\ &= (83.5)_{10}\end{aligned}$$

这种方法用于 α 进制 \rightarrow 十进制较方便。

3. 基数乘 / 除法(重点)

➤ 基数乘 / 除法分为基数乘法和基数除法两种。

- 对于整数的转换，采用基数除法；
- 对于小数的转换，采用基数乘法。

➤ 基数除法（除2取余法）

- 先将 α 进制的整数在 α 进制中连续除以 β ，求得各次余数 $(k_i)_{\alpha}$ ；
- 然后将各余数替代成 β 进制中相应的数字符号 $(k_i)_{\beta}$ 。
- 最后按照并列表示法列出即得 β 进制的整数。

例7 将十进制整数25转换为二进制数，即 $(25)_{10} = (?)_2$

2		25	余数	低位
2		12	$1=k_0$	
2		6	$0=k_1$	
2		3	$0=k_2$	
2		1	$1=k_3$	
		0	$1=k_4$	高位

所以，转换结果为 $(25)_{10} = (11001)_2$

例8 $(785)_{10} = (?)_8$

	余数	
8 785		低位
8 98	1	↑
8 12	2	
8 1	4	
0	1	高位

所以 $(785)_{10} = (1421)_8$

例9 $(687)_{10} = (?)_{16}$

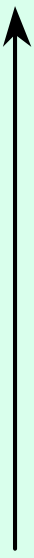
	余数	
16 687		低位
16 42	15	↑
16 2	10	
0	2	高位

由于 $(15)_{10} = (F)_{16}$, $(10)_{10} = (A)_{16}$, $(2)_{10} = (2)_{16}$

所以 $(687)_{10} = (2AF)_{16}$


计算机组成原理

(44)₁₀转化为二进制数

2	44	余数	低位
2	22 0=K ₀	
2	11 0=K ₁	
2	5 1=K ₂	
2	2 1=K ₃	
2	1 0=K ₄	
	0 1=K ₅	
			高位

所以: (44)₁₀ = (101100)₂

(58)₁₀转化为二进制数

2	58	余数	低位
2	29 0=K ₀	
2	14 1=K ₁	
2	7 0=K ₂	
2	3 1=K ₃	
2	1 1=K ₄	
	0 1=K ₅	
			高位

所以: (58)₁₀ = (111010)₂

➤ 基数乘法（乘2取整法）

- 先将 α 进制的小数在 α 进制中连续乘以 β 求得各次乘积的整数部分 $(k_i)_{\alpha}$ ；
- 然后将各整数替代成 β 进制中相应的数字符号 $(k_i)_{\beta}$ ；
- 最后按照并列表示法列出即得 β 进制的小数。

例10 将十进制小数0.6875转换为二进制数。

$$\text{即 } (0.6875)_{10} = (?)_2$$

将上述过程写成简单算式如下：

所以，转换结果为

$$(0.6875)_{10} = (0.1011)_2$$

$$(0.32)_{10} = (?)_2$$

注意：进制转换不会改变有理数特性

0.6875			
$\times \quad 2$	整数		
<hr/>			
1.3750 1=k ₋₁	高位	
0.3750			
$\times \quad 2$			
<hr/>			
0.7500 0=k ₋₂		
0.7500			
$\times \quad 2$			
<hr/>			
1.5000 1=k ₋₃		
0.5000			
$\times \quad 2$			
<hr/>			
1.0000 1=k ₋₄	低位	

计算机组成原理

$(0.375)_{10}$ 转化为二进制数

0.375			
$\times 2$	整数	高位	
<hr/>			
0.750	$0=K_{-1}$		
0.750			
$\times 2$			
<hr/>			
1.500	$1=K_{-2}$		
0.500			
$\times 2$			
<hr/>			
1.000	$1=K_{-3}$		低位

所以: $(0.375)_{10} = (0.011)_2$

- 利用基数乘 / 除法可以将一个十进制混合小数很方便地转换为任何 β 进制的数。只要将整数部分和纯小数部分按上述规则分别进行转换，然后将所得的数组组合起来即可。

例11 $(78.12)_{10} = (?)_5$

整数部分

5	78	余数	低位
5	15 3	↑
5	3 0	
	0 3	低位

小数部分

0.12		
× 5		
0.60	整数	
0.60 0	高位
× 5		↓
3.00 3	低位

所以 $(78.12)_{10} = (303.03)_5$

- 2. 1 进位记数制与数制转换
- 2. 2 带符号数的表示方法
- 2. 3 数的定点表示与浮点表示
- 2. 4 常用的其他编码

2.2 带符号数的表示方法

1. 机器数与真值

- **机器数**：用二进制编码表示的数据。
- **真值**：与机器数对应的实际数据。

2. 机器数包括两种

- **无符号数**：没有符号的整数，如地址、编号等。
 - 例如 $(10010110)_2$ 表示：96H（十进制数150）
 - 字长为 n 位的无符号数的表示范围： $0 \sim 2^n - 1$
- **带符号的数**

- 符号“+”和“-”的表示
 - “0”：正号“+”
 - “1”：负号“-”
 - 用机器数的最高位表示符号位
- 带符号的机器数表示方法：原码、补码和反码

2.2.1 原码表示法

1. 原码的定义

设 X ：二进制数，数值部分的位数为 n 。

➤ 当 $X = \pm 0.X_1X_2\dots X_n$ （纯小数）

$$[X]_{\text{原}} = \begin{cases} X & 0 \leq X < 1 \\ 1 - X = 1 + |X| & -1 < X \leq 0 \end{cases}$$

➤ 当 $X = \pm X_1X_2\dots X_n$ （纯整数）

$$[X]_{\text{原}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^n - X = 2^n + |X| & -2^n < X \leq 0 \end{cases}$$

例12 已知 $n=4$ ，求 X 的原码 $[X]_{\text{原}}$ 。

- 解：
- ① $X = +0.1101$ $[X]_{\text{原}} = 0.1101$
 - ② $X = -0.1101$ $[X]_{\text{原}} = 1 - X = 1.1101$
 - ③ $X = +1101$ $[X]_{\text{原}} = 01101$
 - ④ $X = -1101$ $[X]_{\text{原}} = 2^4 - X = 10000 + 1101 = 11101$

2. 原码的特点

- 原码表示直观、易懂，与真值的转换容易。
- 真值0有两种不同的表示形式

$$[+0]_{\text{原}} = 000\dots0 \quad [-0]_{\text{原}} = 100\dots0$$
- 用原码实现乘、除运算的规则很简单，但实现加减运算比较复杂。

2.2.2 补码表示法

1. 模&同余

- **模**：指一个计量器的容量，可用M表示。

例如：大家所熟悉的钟表，是以12为计数循环的，

模M=12

一个4位的二进制计数器，计数范围为0~15，

模M= 16

- **同余**：指两整数A和B除以同一正整数M，所得余数相同。这时称A和B对M同余，即A和B在以M为模时是相等的，可写成：

$$A=B \pmod{M} \text{ 或 } A=B+kM \quad (k\text{为整数})$$

1. 例如钟表：模 $M=12$ ，故3点和15点、5点和17点是同余的，它们可以写作：

$$3 \equiv 15 \pmod{12}, \quad 5 \equiv 17 \pmod{12}$$

$$15 = 3 + 12, \quad 5 = 17 - 12$$

➤ 将减法运算转化成加法运算

假设当前时针停在7点，现在要将时针调到5点，可以有两种方法实现：

(1) 将时针倒拨2格（2小时）： $7 - 2 = 5$ 做减法

(2) 将时针正拨10格（10小时）： $7 + 10 = 17 \equiv 5 \pmod{12}$ 做加法

从上可得： $7 - 2 \equiv 7 + 10 \pmod{12}$

-2与10对模12互补，也可以说-2的补码是10（以12为模）。

1. 例: $9-5=9+(-5) \equiv 9+(12-5)=9+7 \equiv 4 \pmod{12}$

7为-5的补码

例: $65-25=65+(-25) \equiv 65+(100-25)$

$=65+75 \equiv 40 \pmod{100}$

75为-25的补码

2. 补码的定义

设 X : 二进制数, 数值部分的位数为 n 。

➤ 当 $X = \pm 0.X_1X_2\dots X_n$ (纯小数)

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 1 \\ 2+X=2-|X| & -1 \leq X \leq 0 \end{cases} \pmod{2}$$

➤ 当 $X = \pm X_1 X_2 \dots X_n$ (纯整数)

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^{n+1} + X = 2^{n+1} - |X| & -2^n \leq X \leq 0 \end{cases} \quad (\text{mod } 2^{n+1})$$

例13 已知 $n=4$, 求 X 的补码 $[X]_{\text{补}}$ 。

- 解:
- | | |
|-----------------|--|
| ① $X = +0.1101$ | $[X]_{\text{补}} = 0.1101$ |
| ② $X = -0.1101$ | $[X]_{\text{补}} = 2 + X = 1.0011$ |
| ③ $X = +1101$ | $[X]_{\text{补}} = 01101$ |
| ④ $X = -1101$ | $[X]_{\text{补}} = 2^5 + X = 100000 - 1101 = 10011$ |

3. 补码的特点

➤ 由真值求补码

- 符号位：“0”表示正号“+”，“1”表示负号“-”。
- 数值部分
 - 正数：数值部分与真值形式相同
 - 负数：将真值的数值部分按位将“1”变为“0”，“0”变为“1”（按位变反），且在最低位加1。

例如：

$$\begin{array}{rcl}
 & X = -0.1101 & \\
 \text{X 的真值} & \begin{array}{c} -0.1101 \\ \downarrow \downarrow \downarrow \downarrow \end{array} & \text{按位变反} \\
 & \begin{array}{c} 1.0010 \\ \text{最低位}+1 \end{array} & \\
 \text{X 的补码} & \begin{array}{c} \hline 1.0011 \end{array} &
 \end{array}$$

$$\begin{array}{rcl}
 & X = -1101 & \\
 \text{X 的真值} & \begin{array}{c} -1101 \\ \downarrow \downarrow \downarrow \downarrow \end{array} & \text{按位变反} \\
 & \begin{array}{c} 10010 \\ \text{最低位}+1 \end{array} & \\
 \text{X 的补码} & \begin{array}{c} \hline 10011 \end{array} &
 \end{array}$$

- 真值0的表示形式是**唯一的**

$$[+0]_{\text{补}} = [-0]_{\text{补}} = 000\dots 00$$

- 补码的加减运算规则简单

- 符号位与数值位部分一样参加运算。
- 运算后如有进位产生，则把这个进位舍去不要，相当于舍去一个模。

$$[X+Y]_{\text{补}} = [X]_{\text{补}} + [Y]_{\text{补}}$$

$$[X-Y]_{\text{补}} = [X]_{\text{补}} + [-Y]_{\text{补}}$$

- 已知 $[Y]_{\text{补}}$ ，求 $[-Y]_{\text{补}}$ 。

将 $[Y]_{\text{补}}$ 按位变反，且在最低位加1。

➤ 数据表示范围

以1B（8位）的数据为例：

二进制	十进制（原码）	无符号数十进制	有符号十进制（补码）
0000 0000B:	+0	0	+0
0000 0001B:	+1	1	+1
0000 0010B:	+2	2	+2
.....			
0111 1111B:	+127	127	+127
1000 0000B:	-0	128	-128
1000 0001B:	-1	129	-127
1000 0010B:	-2	130	-126
.....			
1111 1111B:	-127	255	-1

n位表示无符号数范围： $0 \sim 2^n - 1$ ，表示有符号数范围： $-2^{n-1} \sim 2^{n-1} - 1$

➤ 原码、反码、补码的关系

正数的补码 = 原码 = 反码

负数的补码 = 绝对值反码 + 1 (2^n -原码)

例15 已知 ① $Y = +0.1101$; ② $Y = -0.1101$; ③ $Y = +1101$;
④ $Y = -1101$ 。 $n=4$;
求 $[Y]_{\text{补}}$ 、 $[-Y]_{\text{补}}$ 。

解： ① $[Y]_{\text{补}} = 0.1101$ $[-Y]_{\text{补}} = 1.0011$
 ② $[Y]_{\text{补}} = 1.0011$ $[-Y]_{\text{补}} = 0.1101$
 ③ $[Y]_{\text{补}} = 01101$ $[-Y]_{\text{补}} = 10011$
 ④ $[Y]_{\text{补}} = 10011$ $[-Y]_{\text{补}} = 01101$

例16 假设数据位数是8位，求+124和-124的补码表示。

解： $124 = 127 - 3 = (111\ 1111 - 11)_2 = (111\ 1100)_2$

+124的补码为0111 1100 (7CH)；

-124的绝对值(+124)的反码为1000 0011，+1后为-124的补码：
1000 0100 (84H)

例17 假设数据位数是8位，求补码表示为1111 0110 (F6H) 的数值。

解：对补码求反码：0000 1001
+1后：0000 1010 → 十进制10
所求数值是 -10。

2.2.3 反码表示法

1. 反码的定义

设 X ：二进制数，数值部分的位数为 n 。

➤ 当 $X = \pm 0.X_1X_2\dots X_n$ （纯小数）

$$[X]_{\text{反}} = \begin{cases} X & 0 \leq X < 1 \\ 2 - 2^{-n} + X & -1 < X \leq 0 \end{cases} \quad (\text{mod } (2 - 2^{-n}))$$

➤ 当 $X = \pm X_1X_2\dots X_n$ （纯整数）

$$[X]_{\text{反}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^{n+1} - 1 + X & -2^n < X \leq 0 \end{cases} \quad (\text{mod } (2^{n+1} - 1))$$

例16 已知 ① $X = +0.1101$; ② $X = -0.1101$;

③ $X = +1101$; ④ $X = -1101$ 。 $n=4$;

求 X 的反码 $[X]_{\text{反}}$ 。

解： ① $[X]_{\text{反}} = 0.1101$

② $[X]_{\text{反}} = 2 - 2^{-4} + X = 1.0010$

③ $[X]_{\text{反}} = 01101$

④ $[X]_{\text{反}} = 2^5 - 1 + X = 100000 - 1 - 1101 = 10010$

2. 反码的特点

➤ 由真值求反码

- 符号位：“0”表示正号“+”，“1”表示负号“—”。
- 数值部分
 - 正数：数值部分与真值形式相同
 - 负数：将真值的数值部分按位变反

➤ 真值0有两种不同的表示形式

$$[+0]_{\text{反}} = 000\dots 0 \quad [-0]_{\text{反}} = 111\dots 1$$

➤ 反码的加减运算比补码的复杂

3. 8位二进制整数的无符号数、原码、补码、反码表示的真值

二进制表示	无符号数	原码	补码	反码
0000 0000	0	+0	+0	+0
0000 0001	1	+1	+1	+1
...
0111 1111	127	+127	+127	+127
1000 0000	128	-0	-128	-127
1000 0001	129	-1	-127	-126
...
1111 1110	254	-126	-2	-1
1111 1111	255	-127	-1	-0

2.2.4 移码表示法

1. 移码的定义（只讨论纯整数移码）

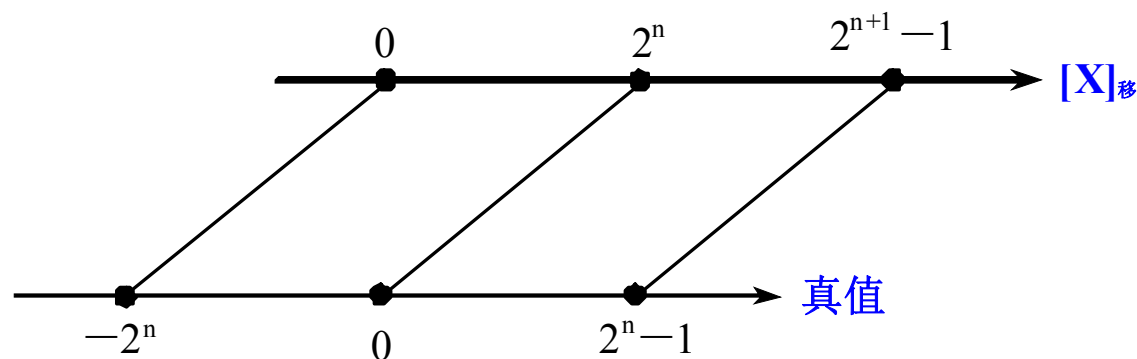
设 X : n 位的二进制数

真值为 $+X_1X_2\dots X_n$ 或 $-X_1X_2\dots X_n$

纯整数移码的定义: $[X]_{\text{移}} = 2^n + X \quad -2^n \leq X < 2^n$

2. 移码就是在真值的基础上加一常数 (2^n)

- 这个常数称为偏移值
- 相当于 X 在数轴上向正方向偏移了若干单位



移码和真值的映射

例17 已知 ① $X = +1101$; ② $X = -1101$; $n=4$;

求 X 的移码 $[X]_{\text{移}}$ 。

解：① $[X]_{\text{移}} = 2^4 + X = 10000 + 1101 = 11101$

② $[X]_{\text{移}} = 2^4 + X = 10000 - 1101 = 00011$

3. 移码的特点

➤ 移码与补码的关系

移码与补码数值部分相同，符号位相反。

即只需将 $[X]_{\text{补}}$ 的符号位变反，就得到 $[X]_{\text{移}}$ 。

当 $0 \leq X < 2^n$ 时， $[X]_{\text{移}} = [X]_{\text{补}} + 2^n$

当 $-2^n \leq X < 0$ 时， $[X]_{\text{移}} = [X]_{\text{补}} - 2^n$

例：① $X = +1101$ ； $[X]_{\text{补}} = 01101$ ； $[X]_{\text{移}} = 11101$

② $X = -1101$ ； $[X]_{\text{补}} = 10011$ ； $[X]_{\text{移}} = 00011$

➤ 移码表示法中，数的最高位（符号位）：

- 如果为“0”，表示该数为负数；
- 如果为“1”，表示该数为正数。

➤ 采用移码的目的：为了能够从机器数的形式上直接判断两数真值的大小，便于浮点数运算中的对阶操作。

例如：两个浮点数相加

$$1.101 \times 2^{-2} + 1.001 \times 2^2$$

指数用补码：1110 < 0010 ?

用移码：1.101x2⁻²⁺⁸ + 1.001x2²⁺⁸

指数用移码：0110 < 1010 **easy**

注意：移码和补码仅第一位不同

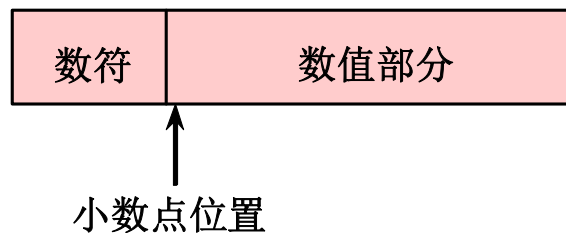
- 2.1 进位记数制与数制转换
- 2.2 带符号数的表示方法
- 2.3 数的定点表示与浮点表示
- 2.4 常用的其他编码

2.3 数的定点表示与浮点表示

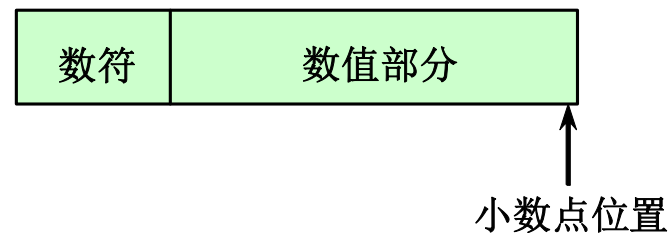
2.3.1 数的定点表示

1. 小数点的位置是固定不变的

- **定点小数**（纯小数）：把小数点固定在数值部分的最高位之前。
- **定点整数**（纯整数）：把小数点固定在数值部分的最低位之后。



(a) 定点小数



(b) 定点整数



2. 数值部分为 n 位的机器数的表示范围（不包含符号位）

- 定点小数原码： $-(1-2^{-n}) \sim (1-2^{-n})$
- 定点小数补码： $-1 \sim (1-2^{-n})$
- 定点整数原码： $-(2^n-1) \sim (2^n-1)$
- 定点整数补码： $-2^n \sim (2^n-1)$

3. 定点表示缺陷：数据表示范围受限，只能靠增加位数来提高表示范围（昂贵的开销）。

2.3.2 数的浮点表示（二进制的科学计数法）

- 小数点的位置不固定，视需要而浮动。
- 浮点数的一般表示形式

$$X = M \times 2^E$$

将数据表示范围和精度分开表示

其中：

E：阶码，用定点整数表示。阶码的值决定了数中小数点的实际位置。阶码的位数决定了数据表示的范围。

M：尾数或有效值，用定点小数表示。尾数的位数决定了数据表示的精度。（可以是规格化，也可以是非规格化）

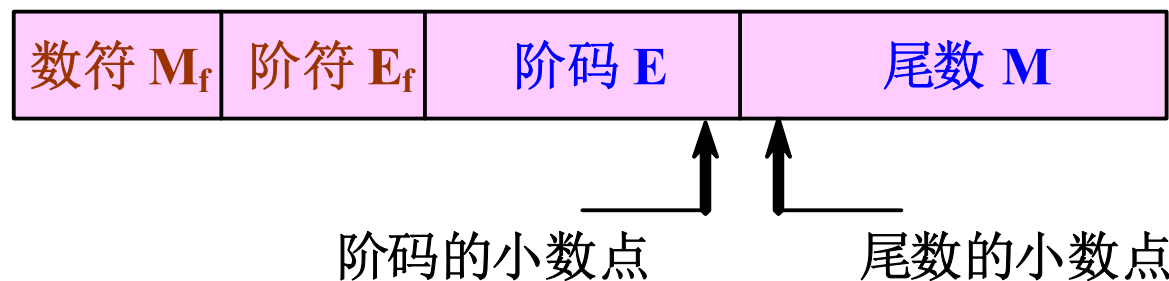
阶码和尾数可以采用原码、补码、反码中任何一种编码方法来表示，但阶码通常采用移码。

例如：

$X = +0.01100101 \times 2^{-101}$ 阶码：-101；尾数+0.01100101

$Y = -0.11010011 \times 2^{+110}$ 阶码：+110；尾数-0.11010011

➤ 用4部分来表示一个浮点数



1. 规格化浮点数

- 用浮点表示法表示一个数时，表示形式不唯一。

例如 $X = +0.01100101 \times 2^{-101}$

$X = +0.11001010 \times 2^{-110}$ （尾数左移（小数点右移），阶码减1）

$X = +0.001100101 \times 2^{-100}$ （尾数右移（小数点左移），阶码加1）

- 规格化浮点数

- 当浮点数的基数为2时，如果其尾数M满足：

$$\frac{1}{2} \leq |M| < 1$$

则该浮点数为规格化浮点数。

- 否则称其为非规格化浮点数。

例18 分别将十进制数 -54 、 $+\frac{13}{128}$ 转换成规格化浮点数表示。
阶码用移码，尾数用补码。其浮点数格式如下所示：

1 位 M_f	1 位 E_f	4 位阶码	10 位尾数
-----------	-----------	-------	--------

其中 M_f 为数符， E_f 为阶符。

解： $(-54)_{10} = (-110110)_2 = (-0.1101100000 \times 2^{110})_2$

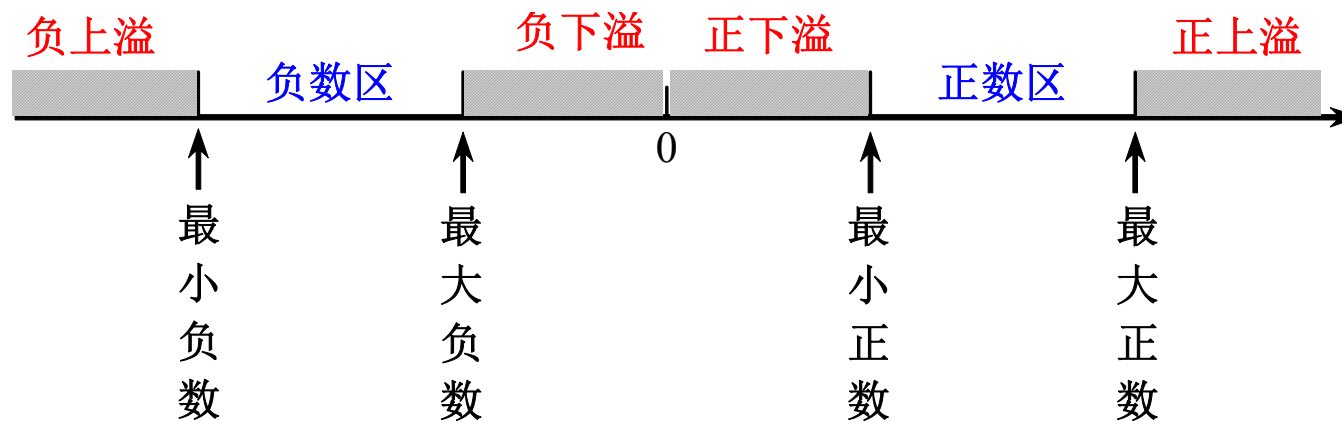
1	1	0110	0010100000
---	---	------	------------

$(+\frac{13}{128})_{10} = (+0.0001101000)_2 = (+0.1101000000 \times 2^{-11})_2$

0	0	1101	1101000000
---	---	------	------------

2. 浮点数的表示范围

设 浮点数的阶码为 m 位，尾数为 n 位，数符和阶符各一位，
则 浮点数的表示范围：



浮点数所能表示数的范围处于最大正数到最小正数、最大负数到最小负数之间。

最大正数 = 最大正尾数 $\times 2^{\text{最大阶码}}$

最小正数 = 最小正尾数 $\times 2^{\text{最小阶码}}$

最大负数 = 最大负尾数 $\times 2^{\text{最小阶码}}$

最小负数 = 最小负尾数 $\times 2^{\text{最大阶码}}$

IEEE754的单精度表示范围:

负数: $-3.402823\text{E}+38 \sim -1.401298\text{E}-45$

正数: $+1.401298\text{E}-45 \sim +3.402823\text{E}+38$

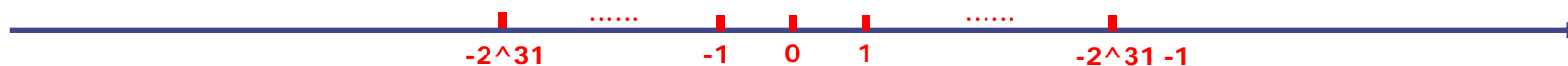
阶码采用移码，尾数采用补码

浮点表示所对应的最大正数、最小正数、最大负数、最小负数

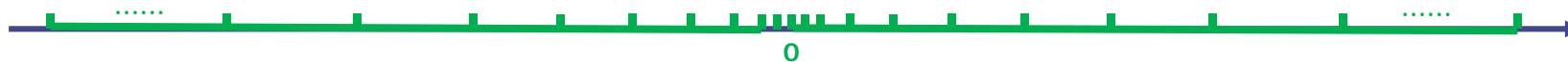
典型数据	浮点形式 数符 阶符 阶码(m) 尾数(n)				真值
非规格化 最小正数	0	0	00...0	00...01	$+ 2^{-n} \times 2^{-2^m}$
规格化 最小正数	0	0	00...0	10...00	$+ 2^{-1} \times 2^{-2^m}$
最大正数	0	1	11...1	11...11	$+ (1 - 2^{-n}) \times 2^{+(2^m - 1)}$
规格化 最大负数	1	0	00...0	11...11	$- 2^{-n} \times 2^{-2^m}$
非规格化 最大负数	1	0	00...0	01...11	$- (2^{-1} + 2^{-n}) \times 2^{-2^m}$
最小负数	1	0	11...1	00...00	$- 1 \times 2^{+(2^m - 1)}$

- 如果一个数超出了数的表示范围，则称为**溢出**。
 - 若该数处于最小正数和最大负数之间，称为**下溢**。
 - 若该数大于最大正数或小于最小负数，称为**上溢**。
- 尾数的位数决定了数据**表示的精度**，增加其位数可以增加有效数字的位数；而阶码的位数决定了数据**表示的范围**。

32位整型数据分布情况：



32位浮点数据分布情况：



浮点数精度带来的“困扰”

```
(gdb) print (1e20-1e20)+3.14
$6 = 3.1400000000000000001
(gdb) print 1e20+(-1e20)+3.14
$7 = 3.1400000000000000001
(gdb) print 1e20+(-1e20+3.14)
$8 = 0
```

第3次运算3.14相对于-1e20已经超出了它的精度表示范围（被忽略了） -- 大数吃小数

计算机组成原理



Prof. Kahan
Prof. Emeritus
UC Berkeley

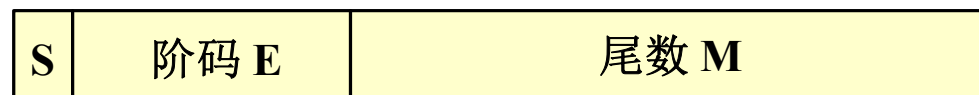
1985年Intel邀请加州大学伯克利分校的 William Kahan教授来为8087 FPU设计浮点数格式;
Intel的浮点数格式设计完成地如此出色以致于IEEE决定采用非常接近的方案作为IEEE的标准

1989年Kahan教授获得图灵奖

3. IEEE 754标准

➤ 每个浮点数由3部分组成

数符S 阶码E 尾数M



<https://zhuanlan.zhihu.com/p/657886517> -- FP16、BF16

➤ 两种基本浮点格式

单精度浮点格式

双精度浮点格式

➤ 两种扩展浮点格式

扩展单精度浮点格式

扩展双精度浮点格式

IEEE 754 标准浮点数基本格式

基本格式	数符位数	阶码位数 (含1位符号位)	尾数位数	总位数
单精度浮点数	1	8	23	32
双精度浮点数	1	11	52	64
扩展单精度浮点数	1	≥ 11	31	≥ 43
扩展双精度浮点数	1	≥ 15	≥ 63	≥ 79

➤ 以单精度浮点数格式为例

- 数符S: 0表示正数, 1表示负数。
- 阶码E: 由1位符号位和7位数值组成。

采用偏移值为127的移码, 即: 阶码 = 127 + 数值

数值 = 阶码 - 127

- 规定阶码的取值范围为: 1~254
- 阶码值0和255用于表示特殊数值
- 尾数M: 23位, 采用原码, 规格化表示。

由于对于规格化数原码来说, 其尾数的最左边一位必定为1(特殊值和非规格化数除外), 所以可以把这个1丢掉, 而把其后的23位放入尾数字段中。

IEEE754中的23位尾数实际上是表示了24位的有效数字。

➤ IEEE 754单精度浮点数的特征参数

特征参数	特征值	特征参数	特征值
符号位数	1	尾数位数	23
阶码位数	8	尾数个数	2^{23}
阶码偏移值	127	最大规格化数	2^{128}
阶码取值范围 (移码)	$1 \sim 254$	最小规格化数	2^{-126}
阶码取值范围 (真值)	$-126 \sim 127$	可表示十进制 数范围	$10^{-38} \sim 10^{38}$
阶码个数	254	最小非规格化 数	$2^{-149} \approx 10^{-45}$

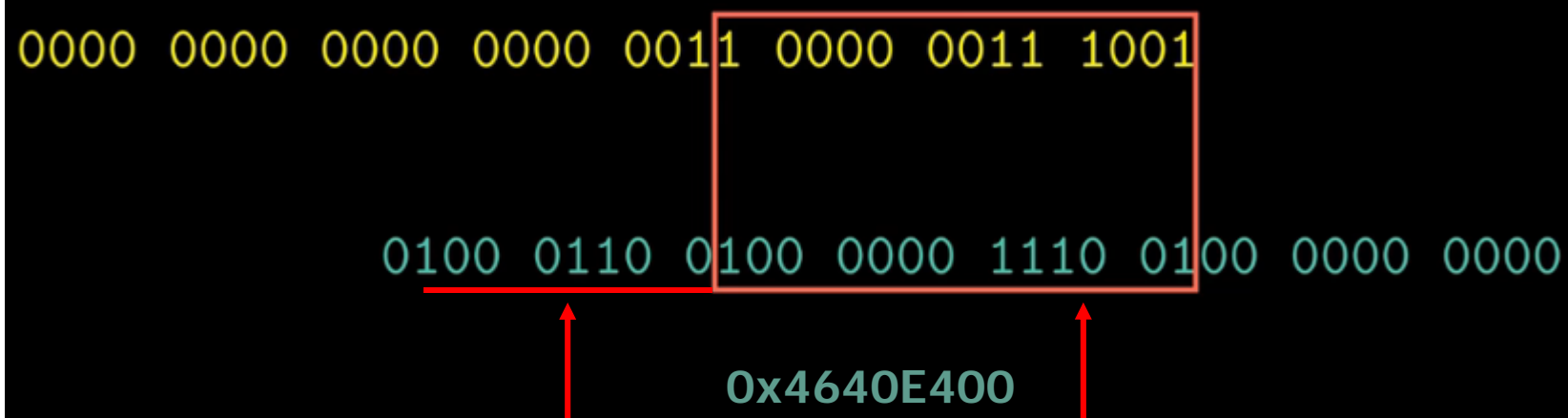
- +19.5的单精度浮点数格式
(二进制表示为10011.1或 1.00111×2^4)
- 0、 $\pm\infty$ 和NaN（非数）几个特别值的表示

数值	符号(1位)	尾数(23位)	阶码(8位)
+19.5	0	001 1100 0000 0000 0000 0000	1000 0011
0	0	000 0000 0000 0000 0000 0000	0000 0000
$\pm\infty$	0或1	000 0000 0000 0000 0000 0000	1111 1111
NaN	0或1	非0的任意值	1111 1111
非规格化数	0或1	非0的任意值	0000 0000

IEEE 754单精度浮点数示例：

int: 12345

0x00003039



float/double: 12345.0 = 1.1000000111001B * 2¹³

阶码: (10001100)₂ = 140

指数: 140 - 127 = 13

尾数

double: 0x40c81c8000000000

IEEE 754单/双精度浮点数示例在x86平台验证：

名称	值	类型
f	12345.000	float
a	0x00003039	int
p	0x0018ff1c	float *
	12345.000	float

地址: 0x0018FF1C
0x0018FF1C 00 e4 40 46

12345.0的单精度浮点
0x4640e400

名称	值	类型
f	12345.00000000000000	double
a	0x00003039	int
p	0x0018ff18	double *
	12345.00000000000000	double

地址: 0x0018FF18
0x0018FF18 00 00 00 00 80 1c c8 40

双精度
0x40c81c8000000000

注意是小端字节顺序

<https://www.wolframalpha.com> -- 进行数学计算

➤ C语言面试题

如何判断一个浮点数是不是等于0?

```
double x;  
if (0.0 == x) ?
```

```
double x;  
if (fabs (x) <= 1e-15) ?
```

浮点数有时候存在误差修订问题。

- 2. 1 进位记数制与数制转换
- 2. 2 带符号数的表示方法
- 2. 3 数的定点表示与浮点表示
- 2. 4 常用的其他编码

2.4 常用的其他编码

2.4.1 十进制数的二进制编码

二—十进制编码：用若干位二进制代码来表示一位十进制数字符号的方法。

1. 8421 (BCD) 码

- 最常用的一种二-十进制编码，它常简称为BCD码。
- 将十进制的每个数字符号用4位二进制数表示，各位的权从左到右分别为8、4、2、1。
- 这样用二进制数的0000~1001来分别表示十进制的0~9。

➤ 主要特点

- 它是一种**有权码**。各位的权从左到右依次为**8、4、2、1**
设8421(BCD)码的各位为 $a_3a_2a_1a_0$ ，则它所代表的值为
$$N=8a_3+4a_2+2a_1+1a_0$$
- 编码简单直观，它与十进制数之间的转换只要直接按位进行就可。

例如

$$\begin{aligned}(91.76)_{10} &= (\underline{1001} \ \underline{0001} . \underline{0111} \ \underline{0110})_{\text{BCD}} \\ (\underline{0110} \ \underline{0000} \ \underline{0001} . \underline{0010})_{\text{BCD}} &= (601.2)_{10}\end{aligned}$$

2. 余3码

- 对应于同样的十进制数字，余3码比相应的BCD码多出0011，所以叫余3码。
- 一个十进制数用余3码表示时，只要按位表示成余3码即可。

例如

$$(90.61)_{10} = (1100\ 0011 . 1001\ 0100)_{\text{余3}}$$

➤ 特点

- 它是一种对9的自补码。
 - 每一个余3码只要自身按位取反，便可得到其对9之补码。

例如：

十进制数字5的余3码为1000，5对9之补是 $9-5=4$ ，而4的余3码是0111，它正好是5的余3码1000按位取反而得。

- 两个余3码相加，所产生的进位相应于十进制数的进位，但所产生的和要进行修正后才是正确的余3码。
 - 如果没有进位，则和需要减3；如果发生了进位，则和需加3。

例如

$\begin{array}{r} 2 \\ +) 3 \\ \hline 5 \end{array}$	$\begin{array}{r} 0\ 1\ 0\ 1 \\ +) 0\ 1\ 1\ 0 \\ \hline 1\ 0\ 1\ 1 \\ -) \quad 1\ 1 \\ \hline 1\ 0\ 0\ 0 \end{array}$	$\begin{array}{r} 9 \\ +) 1 \\ \hline 10 \end{array}$	$\begin{array}{r} 1\ 1\ 0\ 0 \\ +) 0\ 1\ 0\ 0 \\ \hline 1\ 0\ 0\ 0\ 0 \\ +) \quad 1\ 1 \\ \hline 0\ 0\ 1\ 1 \end{array}$
--	---	---	--

3. 2421码

➤ 一种有权码

➤ 2421码的权从左到右分别为2、4、2、1

设2421码中的各位为 $a_3a_2a_1a_0$ ，则它所代表的值为

$$N = 2a_3 + 4a_2 + 2a_1 + 1a_0$$

十进制数	8421码	余3码	2421码	5421码
0	0000	0011	0000	0000
1	0001	0100	0001	0001
2	0010	0101	0010(1000)	0010
3	0011	0110	0011(1001)	0011
4	0100	0111	0100(1010)	0100
5	0101	1000	1011(0101)	0101(1000)
6	0110	1001	1100(0110)	0110(1001)
7	0111	1010	1101(0111)	0111(1010)
8	1000	1011	1110	1011
9	1001	1100	1111	1100
权	8421		2421	5421

2.4.2 字符代码

1. 对各种数字、字母和符号这些字符进行的编码叫做**字符代码**。
2. 在我国获得广泛使用的字符代码有两种
 - 五单位：用五位二进制数来表示不同的字符，它是由电报用的电传打字机的电传码稍加修改而成的。
 - 七单位：用七位二进制数来表示不同的字符。
 - 这是目前应用最广泛的一种字符代码。
3. 国际标准码ASCII码（俗称阿斯克码）

- 是一种七单位代码
- 美国用于信息交换的标准代码（American Standard Code for Information Interchange）
- 128个字符
 - ❑ 26个大写的英文字母和26个小写的英文字母
 - ❑ 10个数字符号
 - ❑ 34个专用符号
 - ❑ 32个控制字符

表 1.5 部分 ASCII 码表

符 号	7 位 ASCII	八 进 制	十六进制	符 号	7 位 ASCII	八进制	十六进制
A	100 0001	101	41	Y	101 1001	131	59
B	100 0010	102	42	Z	101 1010	132	5A
C	100 0011	103	43	0	011 0000	060	30
D	100 0100	104	44	1	011 0001	061	31
E	100 0101	105	45	2	011 0010	062	32
F	100 0110	106	46	3	011 0011	063	33
G	100 0111	107	47	4	011 0100	064	34
H	100 1000	110	48	5	011 0101	065	35
I	100 1001	111	49	6	011 0110	066	36
J	100 1010	112	4A	7	011 0111	067	37
K	100 1011	113	4B	8	011 1000	070	38
L	100 1100	114	4C	9	011 1001	071	39
M	100 1101	115	4D	空格	010 0000	040	20
N	100 1110	116	4E	.	010 1110	056	2E
O	100 1111	117	4F	(010 1000	050	28
P	101 0000	120	50	+	010 1011	053	2B
Q	101 0001	121	51	\$	010 0100	044	24
R	101 0010	122	52	*	010 1010	052	2A
S	101 0011	123	53)	010 1001	051	29
T	101 0100	124	54	-	010 1101	055	2D
U	101 0101	125	55	/	010 1111	057	2F
V	101 0110	126	56	,	010 1100	054	2C
W	101 0111	127	57	=	011 1101	075	3D
X	101 1000	130	58	<RETURN>	000 1101	015	0D

4. 国家标准码为GB1988-80

低4位代码 ($b_4b_3b_2b_1$)	高3位代码($b_7b_6b_5$)							
	000	001	010	011	100	101	110	111
0000	NUL	TC ₇	SP	0	@	P	、	p
0001	TC ₁	DC ₁	!	1	A	Q	a	q
0010	TC ₂	DC ₂	“	2	B	R	b	r
0011	TC ₃	DC ₃	#	3	C	S	c	s
0100	TC ₄	DC ₄	¥	4	D	T	d	t
0101	TC ₅	TC ₈	%	5	E	U	e	u
0110	TC ₆	TC ₉	&	6	F	V	f	v
0111	BEL	TC ₁₀	‘	7	G	W	g	w
1000	FE ₀	CAN	(8	H	X	h	x
1001	FE ₁	EM)	9	I	Y	i	y
1010	FE ₂	SUB	*	:	J	Z	j	z
1011	FE ₃	ESC	+	;	K	[k	{
1100	FE ₄	IS ₄	,	<	L	\	l	
1101	FE ₅	IS ₃	-	=	M]	m	}
1110	SO	IS ₂	.	>	N	^	n	~
1111	SI	IS ₁	/	?	O	_	o	

2.4.3 可靠性编码

1. 格雷 (Gray) 码

- **特点**：从一个代码变为相邻的另一代码时，只有一位发生变化。
- 一种典型的Gray码
- 一种**无权码**
- Gray码与二进制码之间有简单的转换关系

设二进制码为： $B = B_n B_{n-1} \dots B_1 B_0$

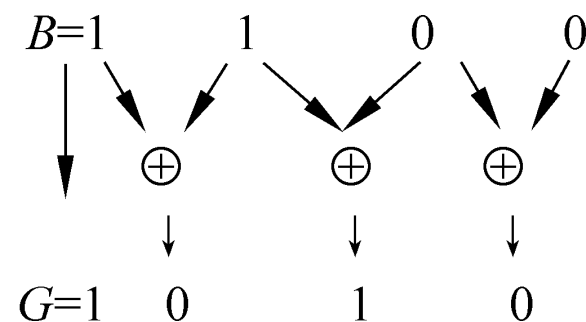
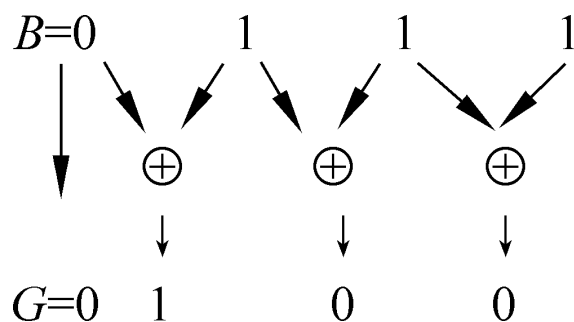
其对应的Gray码为： $G = G_n G_{n-1} \dots G_1 G_0$

则有 $G_n = B_n$ $G_i = B_{i+1} \oplus B_i$
 $i = 0, 1, \dots, n-1$

一种典型的Gray码

十进制数	二进制码	典型的Gray码(16进制循环)
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

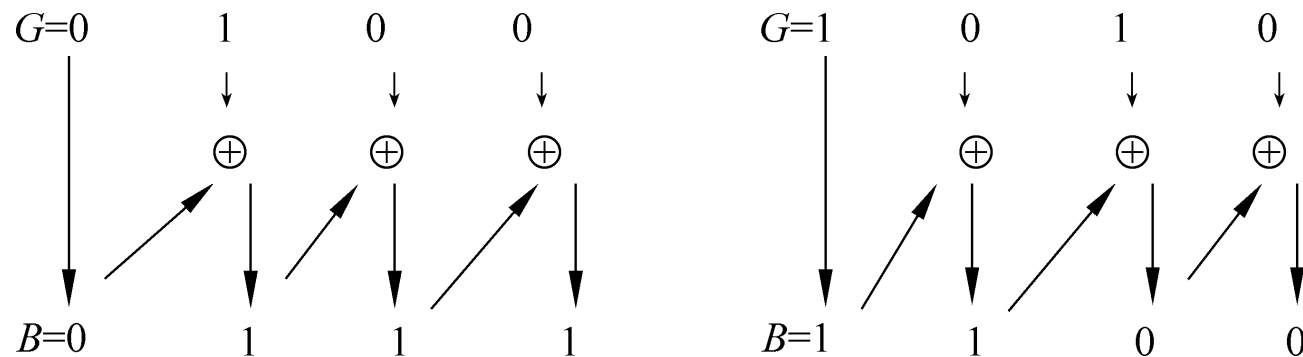
例如，把二进制码0111和1100转换成Gray码：



如果已知Gray码，也可以用类似方法求出对应的二进制码，其方法如下：

$$B_n = G_n \quad B_i = B_{i+1} \oplus G_i \quad (i < n)$$

例如，把Gray码0100和1010转换成二进制码：



➤ Gray码可被用作二-十进制编码。

- ❑ 十进制数的两种Gray码
- ❑ 修改的Gray码又叫余3Gray码，它具有循环性，即十进制数的头尾两个数（0与9）的Gray码也只有一位不同，构成一个“循环”。
- ❑ 所以Gray码有时也称循环码。

十进制数	典型Gray码	修改Gray码(10进制循环)
0	0000	0010
1	0001	0110
2	0011	0111
3	0010	0101
4	0110	0100
5	0111	1100
6	0101	1101
7	0100	1111
8	1100	1110
9	1101	1010

2. 奇偶校验码

➤ 编码方法

- 在 n 位有效信息位的最前面或最后面增加一位二进制校验位 P ，形成 $n+1$ 位的奇偶校验码。
- 如果 $n+1$ 位的奇偶校验码中“1”的个数为奇数，则称为奇校验。
- 如果“1”的个数为偶数，则称为偶校验。

例如：① 8位二进制信息00101011

奇校验码：001010111

偶校验码：001010110

② 8位二进制信息00101010

奇校验码：001010100

偶校验码：001010101

设n位有效信息位： $X_n X_{n-1} \dots X_2 X_1$

其后增加一位二进制校验位： P

那么它们之间的关系为：

奇校验： $P = X_n \oplus X_{n-1} \oplus \dots \oplus X_2 \oplus X_1 \oplus 1$

偶校验： $P = X_n \oplus X_{n-1} \oplus \dots \oplus X_2 \oplus X_1$

➤ 奇偶校验码的校验方法

- 如果奇校验码中“1”的个数为偶数，或者偶校验码中“1”的个数为奇数，则编码出错了。

- 校验方程

奇校验:
$$E = \overline{X_n \oplus X_{n-1} \oplus \cdots \oplus X_2 \oplus X_1 \oplus P}$$

偶校验:
$$E = X_n \oplus X_{n-1} \oplus \cdots \oplus X_2 \oplus X_1 \oplus P$$

如果 $E=0$ ，则编码正确；如果 $E=1$ ，则编码出错。

- 奇偶校验码只能发现一位或奇数个位出错，不能发现偶数个位同时出错。**不能纠正错误。**

3. 海明校验码

➤ 实现原理

- ❑ 在 n 位有效信息位中增加 k 位校验位，形成一个 $n+k$ 位的编码；
 - ❑ 把编码中的每一位分配到 k 个奇偶校验组中；
 - ❑ 每一组只包含一位检验位，组内按照奇校验或偶校验的规则求出该组中的校验位。
- 在海明校验码中，有效信息位的位数 n 与校验位数 k 满足如下关系：

$$2^k - 1 \geq n + k$$

➤ 海明校验码中有效信息位数与校验位数的关系

n	最小k值	n	最小k值
1~4	3	27~57	6
5~11	4	58~119	7
12~26	5		

(1) 校验码的编码方法

海明校验码的编码过程可分3个步骤进行。

第一步：确定有效信息位与校验位在编码中的位置；

- 最终形成的 $n+k$ 位海明校验码

$$H_{n+k}H_{n+k-1}\cdots H_2H_1$$

- 各位的位号按从右到左的顺序

$$1, 2, \dots, n+k$$

每个校验位 P_i 所在的位号： 2^{i-1} ($i=1, 2, \dots, k$)

- 有效信息位按原排列顺序依次安排在其它位置上

例如：设7位有效信息位： $X_7X_6X_5X_4X_3X_2X_1$

$n=7$ 校验位位数 $k=4$

构成的海明校验码： 11 位

4个校验位 $P_4P_3P_2P_1$ 应分别位于位号为 2^{i-1} 的位置上， $i=1, 2, 3, 4$ ，即位号为 2^0 、 2^1 、 2^2 、 2^3 。

11位海明校验码的编码排列为：

位号：	11	10	9	8	7	6	5	4	3	2	1
编码：	H_{11}	H_{10}	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	X_7	X_6	X_5	P_4	X_4	X_3	X_2	P_3	X_1	P_2	P_1

第二步：将 $n+k$ 位海明校验码中的每一位分到 k 个奇偶校验组中。

分组的方法如下：

- 将校验码中的每一位的位号 M 写成 k 位二进制数的形式： $M_{k-1}M_{k-2}\dots M_1M_0$
- 对于编码中的任何一位 H_M ，依次按从右至左（即从低位到高位）的顺序查看其 $M_{k-1}M_{k-2}\dots M_1M_0$ 的每一位 $M_j(j=0, 1, \dots, k-1)$ ，若该位为“1”，则将 H_M 分到第 j 组。

上面的例子共分为4组。

例：11位（7+4）海明校验码的分组结果

第一步	位号	11	10	9	8	7	6	5	4	3	2	1
	位号对应的二进制数	1011	1010	1001	1000	0111	0110	0101	0100	0011	0010	0001
	编码	X_7	X_6	X_5	P_4	X_4	X_3	X_2	P_3	X_1	P_2	P_1
第二步	第0组	X_7		X_5		X_4		X_2		X_1	0	P_1 1
	第1组	X_7	X_6			X_4	X_3			X_1	P_2 1	0
	第2组					X_4	X_3	X_2	P_3		0	0
	第3组	X_7	X_6	X_5	P_4						0	0

第三步：根据分组结果，每一组按奇校验或偶校验求出校验位，形成海明校验码。

- 若采用奇校验，则每一组中“1”的个数为奇数；
- 若采用偶校验，则每一组中“1”的个数为偶数。

在上面的例子中：

采用奇校验

$$\begin{aligned}P_1 &= \overline{X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1} \\P_2 &= \overline{X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1} \\P_3 &= \overline{X_4 \oplus X_3 \oplus X_2} \\P_4 &= \overline{X_7 \oplus X_6 \oplus X_5}\end{aligned}$$

采用偶校验

$$\begin{aligned}P_1 &= X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1 \\P_2 &= X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1 \\P_3 &= X_4 \oplus X_3 \oplus X_2 \\P_4 &= X_7 \oplus X_6 \oplus X_5\end{aligned}$$

例19 在上面的例子中，若7位有效信息位为

$$X_7X_6X_5X_4X_3X_2X_1=1001101$$

求其海明校验码。

解：若采用奇校验，则：

$$P_1 = \overline{X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1} = \overline{1 \oplus 0 \oplus 1 \oplus 0 \oplus 1} = 0$$

$$P_2 = \overline{X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1} = \overline{1 \oplus 0 \oplus 1 \oplus 1 \oplus 1} = 1$$

$$P_3 = \overline{X_4 \oplus X_3 \oplus X_2} = \overline{1 \oplus 1 \oplus 0} = 1$$

$$P_4 = \overline{X_7 \oplus X_6 \oplus X_5} = \overline{1 \oplus 0 \oplus 0} = 0$$

将这些校验位与有效信息位一起排列，可得11位海明校验码为：

$$10001101110$$

若采用偶校验，则：

$$P_1 = X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 = 1$$

$$P_2 = X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$P_3 = X_4 \oplus X_3 \oplus X_2 = 1 \oplus 1 \oplus 0 = 0$$

$$P_4 = X_7 \oplus X_6 \oplus X_5 = 1 \oplus 0 \oplus 0 = 1$$

将这些校验位与有效信息位一起排列，可得11位海明校验码为：

10011100101

(2) 检验码的校验方法

➤ 校验的方法

- 将 $n+k$ 位海明校验码按编码时采用的方法，重新再分成 k 个组。
 - 奇校验：每一组中“1”的个数应该为奇数
 - 偶校验：每一组中“1”的个数应该为偶数
- 如果不满足，则表示该校验码出错了。

➤ 具体实现

- 在分成的 k 个组中，将每一组中所有的信息位异或起来，得到 k 位校验结果 $E_{k-1}E_{k-2}\dots E_1E_0$ 。
(指误字)

- 若奇校验中

$$\overline{E_{k-1}} \overline{E_{k-2}} \dots \overline{E_1} \overline{E_0} = 00\dots 00$$

或偶校验中

$$E_{k-1} E_{k-2} \dots E_1 E_0 = 00\dots 00$$

则该校验码正确，没有出错。

- 否则结果有错，这时候 $E_{k-1} E_{k-2} \dots E_1 E_0$ 代码所对应的十进制数就是校验码中出错信息位的位号。

➤ 纠正方法

将出错信息位变反即可。

上面例子中的11位校验码进行校验：

$$E_0 = X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1 \oplus P_1$$

$$E_1 = X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1 \oplus P_2$$

$$E_2 = X_4 \oplus X_3 \oplus X_2 \oplus P_3$$

$$E_3 = X_7 \oplus X_6 \oplus X_5 \oplus P_4$$

例20 在例11中，采用奇校验的11位海明校验码应为：
10001101110。若接收到的代码为**10001101110**和**10001111110**，分别
检验它们有无错误？若有，判别出错位置。

解：① 若接收到的代码为**10001101110**，则得到检验结果为：

$$E_0 = X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1 \oplus P_1 = 1 \oplus 0 \oplus 1 \oplus 0 \oplus 1 \oplus 0 = 1$$

$$E_1 = X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1 \oplus P_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$E_2 = X_4 \oplus X_3 \oplus X_2 \oplus P_3 = 1 \oplus 1 \oplus 0 \oplus 1 = 1$$

$$E_3 = X_7 \oplus X_6 \oplus X_5 \oplus P_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

因为 $\overline{E_3} \overline{E_2} \overline{E_1} \overline{E_0} = \mathbf{0000}$ ，所以收到的海明校验码没有错。

② 若接收到的代码为**1000111110**，则得到检验结果为：

$$E_0 = X_7 \oplus X_5 \oplus X_4 \oplus X_2 \oplus X_1 \oplus P_1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$E_1 = X_7 \oplus X_6 \oplus X_4 \oplus X_3 \oplus X_1 \oplus P_2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$E_2 = X_4 \oplus X_3 \oplus X_2 \oplus P_3 = 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$E_3 = X_7 \oplus X_6 \oplus X_5 \oplus P_4 = 1 \oplus 0 \oplus 0 \oplus 0 = 1$$

因为 $\overline{E_3} \overline{E_2} \overline{E_1} \overline{E_0} = \mathbf{0101}$ ，不为全“0”，表示收到的校验码有错。

因为 $\overline{E_3} \overline{E_2} \overline{E_1} \overline{E_0}$ 为**0101**，指出是**第5位**信息出错，将第5位信息（即 **X_2** ）变反，就得到正确的代码。

(3) 改进

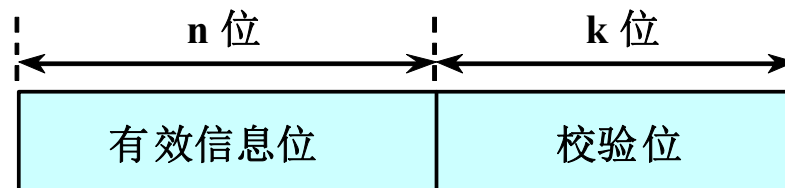
- 上述海明校验码**只能**发现一位错误，并指出是哪一位错了。
- 在前面讲的海明校验码的基础上，再增加一位校验位，使得有效信息位的位数 n 与校验位数 k 满足如下关系：

$$2^{k-1} \geq n + k$$

- 称为**扩展的海明校验码**
- 能发现并纠正一位出错，还能发现两位出错，但不能指出是哪两位。（ $n=7$ 时， k 最小取5）

4. 循环冗余校验码

- 简称CRC码 (Cyclic Redundancy Check)
- 在 n 位有效信息位后拼接 k 位校验位构成，通过数学运算建立有效信息位与校验位之间的关系，形成一个 $n+k$ 位的代码。
- 常被称为 $(n+k, n)$ 码



循环冗余校验码的格式

(1) 模2运算

以按位模2相加的四则运算，运算时不考虑进位和借位。

➤ **模2加减**：就是按位作异或运算，模2加与模2减的结果一样。运算规则为：

$$0 \pm 0 = 0$$

$$0 \pm 1 = 1 \pm 0 = 1$$

$$1 \pm 1 = 0$$

例21 按模2加减规则，求 $1100 + 1010$ ； $1010 - 0111$ ； $1010 + 1010$ 。

解： $1100 + 1010 = 0110$

$$1010 - 0111 = 1101$$

$$1010 + 1010 = 0000$$

- **模2乘：**与一般二进制乘法唯一不同的就是最后按模2加求部分积之和。

例22 按模2乘规则，求 1010×101 。

解： $1010 \times 101 = 100010$

$$\begin{array}{r}
 1010 \\
 \times 101 \\
 \hline
 1010 \\
 0000 \\
 1010 \\
 \hline
 100010
 \end{array}$$

(a) 模2乘

$$\begin{array}{r}
 101 \\
 \overline{)10010} \\
 \underline{101} \\
 011 \\
 \underline{000} \\
 110 \\
 \underline{101} \\
 11
 \end{array}$$

(b) 模2除

➤ 模2除

- 每一次都是按模2减求余数。
- 若余数（初始为被除数）最高位为1，则商“1”；
- 若余数最高位为0，则商“0”。
- 当余数的位数小于除数位数时，除法结束。

例23 按模2除规则，求 $10010 \div 101$ 。

解： $10010 \div 101 = 101$ 余数为11

(2) CRC的编码方法

设待编码的 n 位有效信息位： $C_n C_{n-1} \dots C_2 C_1$

循环冗余校验码的编码步骤如下：

第一步：将n位有效信息位表示为多项式M (x) 的形式：

$$M(x) = C_n x^{n-1} + C_{n-1} x^{n-2} + \dots + C_2 x + C_1$$

例如：8位有效信息位11010011可以表示为：

$$M(x) = x^7 + x^6 + x^4 + x + 1$$

第二步：选择一个k+1位的生成多项式G (x) ，然后按模2除，用M (x) ·x^k除以G (x) ，得到k位余数R (x) 和商Q (x) 。

$$\frac{M(x) \bullet x^k}{G(x)} = Q(x) + \frac{R(x)}{G(x)}$$

第三步：得到的k位余数就是所求的校验位，将它拼接在n位有效信息位的后面，即得到n+k位的CRC码。

例24 选择生成多项式 $G(x)=x^3+x+1$ ，即为1011。将4位有效信息1100编码成7位CRC码。

解： $M(x) = x^3 + x^2 = 1100$

$M(x) \cdot x^3 = x^6 + x^5 = 1100000$ （即1100左移3位）

模2除： $M(x) \cdot x^3 / G(x) = 1100000 / 1011 = 1110 + 010 / 1011$

即 $R(x) = 010$

得到7位的CRC码为：1100010，这种CRC码称为（7，4）码。

(3) CRC的校验方法

将收到的CRC码用原来的生成多项式 $G(x)$ 去除。

- 若得到的余数为0，则接收到的代码没有错；
- 若余数不为0，则表示接收到的代码中的某一位出错了。
 - 不同的位出错，所对应的余数不同；
 - 根据余数判定是哪一位错了；
 - 将相应位变反就得正确的代码。
- 对应于例2.12中的生成多项式 $G(x)=x^3+x+1$ ，下表列出了(7, 4)码的出错模式。

(7, 4) 码的出错模式 (生成多项式 $G(x)=1011$)

	D_7	D_6	D_5	D_4	D_3	D_2	D_1	余数	出错位
正确码	1	1	0	0	0	1	0	000	无
错误码	1	1	0	0	0	1	<u>1</u>	001	1
	1	1	0	0	0	<u>0</u>	0	010	2
	1	1	0	0	<u>1</u>	1	0	100	3
	1	1	0	<u>1</u>	0	1	0	011	4
	1	1	<u>1</u>	0	0	1	0	110	5
	1	<u>0</u>	0	0	0	1	0	111	6
	<u>0</u>	1	0	0	0	1	0	101	7

从表中可以看出：如果得到的余数为100，则表示第3位 D_3 错了；
如果得到的余数为111，则表示第6位 D_6 错了。

(4) CRC码的生成多项式

- 生成多项式应满足下列要求
 - 任何一位出错都应使余数不为0;
 - 不同位出错应使余数不同;
 - 对余数继续作模2除法, 应使余数循环。
- 有3种多项式成为标准而被广泛运用。

$$\text{CRC}_{12} = x^{12} + x^{11} + x^3 + x^2 + x + 1$$

$$\text{CRC}_{16} = x^{16} + x^{15} + x^2 + 1$$

$$\text{CRC}_{\text{CCITT}} = x^{16} + x^{12} + x^5 + 1$$

作业:

2.1-2.4 2.6 2.7

2.9-2.12

2.14 2.16