

东南大学计算机学院

计算机系统组成

主讲教师： 徐造林

第3章 数据的表示

- 本章讨论数据（数值数据和非数值数据）在计算机中的表示方法及运算。

3.1 数字化信息编码

3.1.1 数据、信息和媒体

- ◆ **数据**是对事实、概念或指令的一种特殊**表达形式**，可用人工的方式或者用自动化的装置进行**通信**、**翻译转换**或者进行**加工处理**。
- 在计算机系统中所指的数据都是以**二进制编码**形式出现的。

- 计算机内部由硬件实现的基本数据区分为数值型数据和非数值型数据。
- ◆ 信息是对人有用的数据，这些数据可能影响到人们的行为和决策。
- 计算机信息处理，实质上就是由计算机进行数据处理的过程。
- ◆ 媒体又称媒介、媒质，是指承载信息的载体。

◆ 与计算机信息处理有关的媒体有：

- 感觉媒体
- 表示媒体
- 存储媒体
- 表现媒体
- 传输媒体

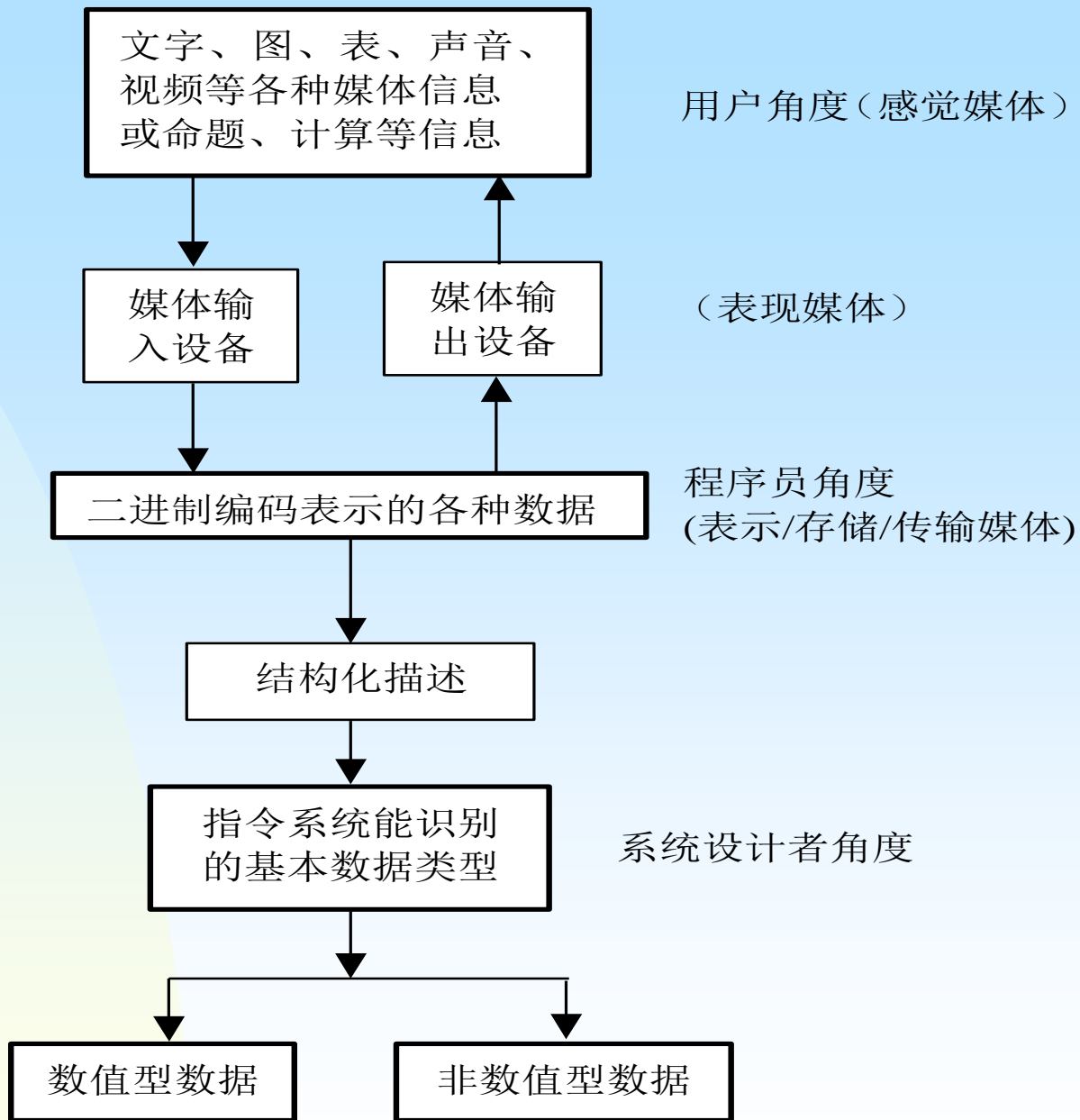


图 3.1 计算机外部信息与内部数据的转换

3.1.2 数字化信息编码

- **编码**是用少量简单的基本符号，对大量复杂多样的信息进行**一定规律的组合**。
- **数字化信息编码**就是用计算机**能够接受**的形式来表示各种数据。
- 计算机内部采用**二进制**表示的原因：
 - 1) 二进制只有**两种状态**，在数字电路中很容易实现。
 - 2) 二进制编码、**运算规则非常简单**。
 - 3) 二进制的“0”、“1”与二值逻辑一致，很容易**实现逻辑运算**。

- 编码时应遵循的原则：

- 1) 唯一性，每个代码只能唯一表示一个数据；
- 2) 合理性，代码结构要与分类体系相适应，尽量反映编码对象的特征。
- 3) 简单性，代码简单可以减少差错，并且易理解，易记忆；
- 4) 规范性，代码应与已有的国际或国家标准一致；
- 5) 可扩充性，应留有适当的后备容量，适应有可能出现的变化或增减。

3.2 数值数据的编码表示

- **数值数据**是表示数量多少和数值大小的数据。
- 在计算机内部，数值数据有两种表示方法：
 - 1) 直接用**二进制数**表示；
 - 2) 采用二进制编码的十进制数（**BCD**）表示。
- 表示一个**数值数据**要确定三个要素：
 - 1) **进位计数制**；
 - 2) **定 / 浮点表示**；
 - 3) **数的编码表示**。

3.2.1 进位计数制及其各进位制数之间的转换

- 在某个数字系统中，若采用 R 个基本符号（0, 1, 2, . . . , $R-1$ ）表示各位上的数字，则称其为基 R 数制，或称 **R 进制数字系统**， R 被称为该数字系统的基；
- 采用“**逢 R 进一**”的运算规则，对于每一个数位 i ，其位上的权为 R^i 。

◆ 在计算机系统中，常用的几种进位计数制：

- 二进制 $R=2$ ，基本符号为 0和1；
逢2进一。
- 八进制 $R=8$ ，基本符号为 0,1,2,3,4,5,6,7 ；
逢8进一。
- 十六进制 $R=16$ ，基本符号为 0,1,2,3,4,5,6,7,8,9,
A,B,C,D,E,F；
逢16进一。
- 十进制 $R=10$ ，基本符号为 0,1,2,3,4,5,6,7,8,9；
逢10进一。

例：十进制数2585.62代表的实际值是：

$$2 \times 10^3 + 5 \times 10^2 + 8 \times 10^1 + 5 \times 10^0 + 6 \times 10^{-1} + 2 \times 10^{-2}$$

例：二进制数 $(100101.01)_2$ 代表的实际值是：

$$(100101.01)_2 = 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2} = (37.25)_{10}$$

例：八进制数 $(14267.531)_8$ 代表的实际值是：

$$(14267.531)_8 = 1 \times 8^4 + 4 \times 8^3 + 2 \times 8^2 + 6 \times 8^1 + 7 \times 8^0 + 5 \times 8^{-1} + 3 \times 8^{-2} + 1 \times 8^{-3}$$

◆ 不同进位计数制之间的转换

1. R进制数转换成十进制数

- 任何一个R进制数转换成十进制数时，只要“按权展开”即可。

例1 二进制数转换成十进制数。

$$(10101.01)_2 = (1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 + 0 \times 2^{-1} + 1 \times 2^{-2})_{10} = (21.25)_{10}$$

例2 八进制数转换成十进制数。

$$(307.6)_8 = (3 \times 8^2 + 7 \times 8^0 + 6 \times 8^{-1})_{10} = (199.75)_{10}$$

例3 十六进制数转换成十进制数。

$$(3A.C)_{16} = (3 \times 16^1 + 10 \times 16^0 + 12 \times 16^{-1})_{10} = (58.75)_{10} \blacksquare$$


2. 十进制数转换成R进制数

- 任何一个十进制数转换成R进制数时，要将**整数**和**小数**部分**分别进行转换**。

(1) 整数部分的转换

- 整数部分的转换方法是“**除基取余，上右下左**”。

例1 将十进制整数835分别转换成二、八进制数。

	余数	低位
8 835	3	
8 104	0	
8 13	5	
8 1	1	
0		高位

$$(835)_{10} = (1503)_8$$

		余数	低位
2	835	1	↑
2	417	1	
2	208	0	
2	104	0	
2	52	0	
2	26	0	
2	13	1	
2	6	0	
2	3	1	
2	1	1	↑
	0		
			高位

$$(835)_{10} = (1101000011)_2$$

(2) 小数部分的转换

- 小数部分的转换方法是“**乘基取整，上左下右**”。

例2 将十进制小数0.6875分别转换成二、八进制数。

	整数部分	高位
$0.6875 \times 2 = \underline{1}.375$	1	
$0.375 \times 2 = \underline{0}.75$	0	
$0.75 \times 2 = \underline{1}.5$	1	
$0.5 \times 2 = \underline{1}.0$	1	
$(0.6875)_{10} = (0.1011)_2$		低位

	整数部分	高位
$0.6875 \times 8 = 5.5$	5	↓
$0.5 \times 8 = 4.0$	4	
$(0.6875)_{10} = (0.54)_8$		低位

例3 将十进制小数0.63转换成二进制数。

	整数部分	高位
$0.63 \times 2 = 1.26$	1	↓
$0.26 \times 2 = 0.52$	0	
$0.52 \times 2 = 1.04$	1	
$0.04 \times 2 = 0.08$	0	
$(0.63)_{10} = (0.1010)_2$ (近似值)		低位

(3) 含整数、小数部分的数的转换

- 将整数、小数部分分别进行转换，得到转换后的整数和小数部分，然后再这两部分组合起来得到一个完整的数。

例4 将十进制数835.6875转换成二、八进制数。

- 整数部分： $(835)_{10} = (1101000011)_2$
- 小数部分： $(0.6875)_{10} = (0.1011)_2$
- $(835.6875)_{10} = (1101000011.1011)_2$
 $= (1503.54)_8$

3. 二、八、十六进制数的相互转换

(1) 八进制数转换成二进制数

例1 将 $(13.724)_8$ 转换成二进制数

$$\begin{aligned} \bullet (13.724)_8 &= (\underline{001} \ \underline{011} . \underline{111} \ \underline{010} \ \underline{100})_2 \\ &= (1011.1110101)_2 \end{aligned}$$

(2) 十六进制数转换成二进制数

例2 将十六进制数 $2B.5E$ 转换成二进制数

$$\begin{aligned} \bullet (2B.5E)_{16} &= (\underline{0010} \ \underline{1011} . \underline{0101} \ \underline{1110})_2 \\ &= (101011.010111)_2 \end{aligned}$$

(3) 二进制数转换成八进制数

$$\bullet (10011.01)_2 = (\underline{010} \ \underline{011} . \underline{010})_2 = (23.2)_8$$

(4) 二进制数转换成十六进制数

$$\bullet (11001.11)_2 = (\underline{0001} \ \underline{1001} . \underline{1100})_2 \\ = (19.C)_{16}$$

◆ 各种进位制之间的对应关系

二进制数	八进制数	十进制数	十六进制数
0000	0	0	0
0001	1	1	1
0010	2	2	2
0011	3	3	3
0100	4	4	4
0101	5	5	5
0110	6	6	6
0111	7	7	7
1000	10	8	8
1001	11	9	9
1010	12	10	A
1011	13	11	B
1100	14	12	C
1101	15	13	D
1110	16	14	E
1111	17	15	F

3.2.2 定点与浮点表示

1. 定点表示

- 约定计算机中所有数据的小数点位置是**固定不变**的；该位置在计算机设计时已被**隐含地规定**。

±	1	0	1	0	1	1	0
---	---	---	---	---	---	---	---

定点小数

•



小数点位置 (隐含约定)

±	1	0	1	0	0	0	1
---	---	---	---	---	---	---	---

定点整数

•



小数点位置
(隐含约定)

- **n**位定点小数**X**（无符号），其表示范围为：

$$2^{-n} \leq |X| \leq 1 - 2^{-n}$$

- **n**位定点整数**X**（无符号），其表示范围为：

$$1 \leq |X| \leq 2^n - 1$$

- 采用定点数表示数的范围较小，运算很容易产生溢出，且选择适当的比例因子有时也很困难。

2. 浮点表示

- 在计算机中所表示的数，其小数点位置是可变的，这种数称为浮点数。
- 浮点数格式

数符 1	阶符 1	阶 e	尾数 m
------	------	-----	------

- 对于任意一个二进制数 X ，可表示为：

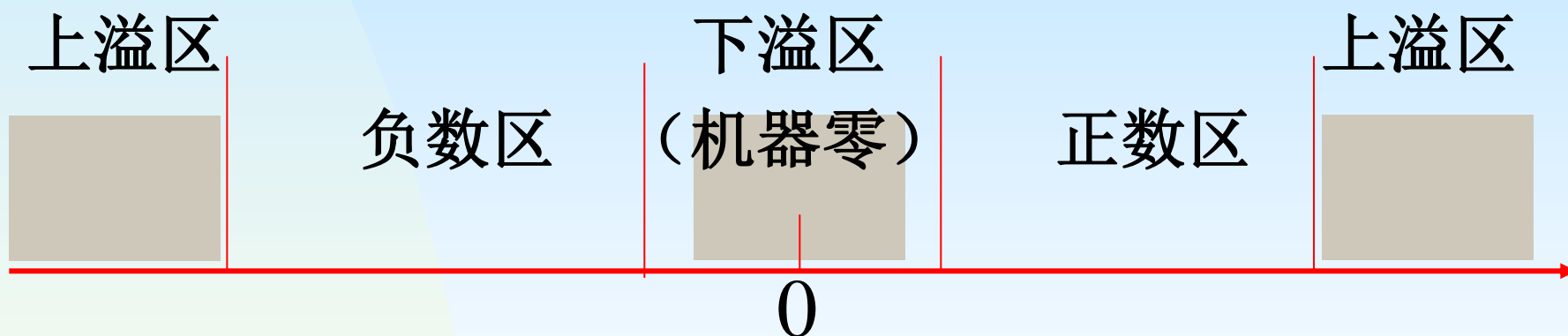
$$X = (-1)^S \times M \times R^E$$

- S 为0或1，决定数 X 的符号， M 为尾数，常用定点纯小数表示； E 为阶，一般用定点整数表示； R 为基数，隐含为2，也可以为 2^q ， q 可取2，3，4等正整数。

- 浮点数表示数的范围

$$|N_{\max}| = (1 - 2^{-m}) \times 2^{(2e-1)}$$

$$|N_{\min}| = 2^{-m} \times 2^{-(2e-1)}$$



- 浮点表示法的最大特点是它可以表示很大的数据范围以及较高的数据精度。

3.2.3 机器数（编码系统）

- **符号数字化**：0表示正号，1表示负号。
- **机器数**：数值数据在计算机**内部编码**表示的数。
- **真值**：机器数真正的值（用正、负号来表示正数和负数）。
- **机器数**具有的特点：
 - 1) 机器数具有一定的**范围**，超过这个范围便会发生溢出；
 - 2) 机器数将其真值的**符号数字化**；
 - 3) 机器数中依靠**格式上的约定**表示小数点的位置。

- 机器数X的真值为：

$$X_T = \pm X_{n-1}' X_{n-2}' \dots X_1' X_0' \quad (\text{定点整数})$$

$$X_T = \pm 0.X_{-1}' X_{-2}' \dots X_{-(n-1)}' X_{-n}' \quad (\text{定点小数})$$

- 数字化**编码**后的机器数（码）X表示为：

$$X = X_n X_{n-1} X_{n-2} \dots X_1 X_0$$

- 当 X_T 为正数时， $X_n=0$, $X_i=X_i'$ （定点整数）
或 $X_i=X_{(i-n)}'$ （定点小数）
- 当 X_T 为负数或0时， $X_n=1$ ，数值部分各位的取值就依赖于**相应的编码**表示方式。

◆ 机器数（编码）表示

1. 原码表示法

▲ 原码表示的机器数由符号位后直接跟上真值的数值构成。

• 定点小数：

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

• 如：
[+0]=0.00 . . . 0
[- 0]=1.00 . . . 0

- 定点整数:

$$[X]_{\text{原}} = \begin{cases} X & 0 \leq X < 2^n \\ 2^{n-1} - X & -2^{n-1} < X \leq 0, \text{ } n \text{ 为编码位数} \end{cases}$$

- 例: $[+13]_{\text{原}} = 01101$

$$[-13]_{\text{原}} = 11101$$

- 用原码实现乘除运算比较简便。

2. 补码表示法

(1) 模运算

- 对一个多于n位的数**丢弃高位**而保留低n位数；
这样一种操作运算就是“**模运算**”。

- 在模运算中，若A，B，M满足下列关系：

$$A=B+K\times M \quad (K\text{为整数})$$

则记为：

$$A\equiv B \pmod{M}$$

- 称B和A为**模M同余**。

- 假定现在钟表时针指向**10点**，要将它**拨向8点**，则有两种拨法：

① 倒拨2格： $10 - 2 = 8$

② 顺拨10格： $10 + 10 = 20 \equiv 8 \pmod{12}$

- 在模12系统中： $10 - 2 \equiv 10 + 10 \pmod{12}$

$- 2 \equiv 10 \pmod{12}$

- 称10是 $- 2$ 对模12的补码。

- 同样：

$- 3 \equiv 9 \pmod{12}$; $- 5 \equiv 7 \pmod{12}$ 等。

- 结论：“对于某一确定的模，某数减去小于模的另一数，总可以用该数加上模与另一数绝对值之差来代替”；补码可以用加法实现减法运算。
- 例1. “4位十进制数” 模运算系统（相当于只有四档的算盘）。

$$\begin{aligned} 9828 - 1928 &\equiv 9828 + (10^4 - 1928) \\ &\equiv 9828 + 8072 \\ &\equiv 7900 \pmod{10^4} \end{aligned}$$

(2) 补码的定义

- 一个负数的补码应等于模与该数绝对值之差。
即某负数 X 的补码为：

$$[X]_{\text{补}} = M + X \quad (\text{mod } M)$$

- 假定补码的位数为 n ，补码表示的定义为：

1) 定点整数：

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

2) 定点小数:

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

- 补码0的表示是唯一的:

$$[+0]_{\text{补}} = \textcolor{red}{0} 0 \dots 0$$

$$[-0]_{\text{补}} = 2^n - 0 = 1 \textcolor{red}{0} 0 \dots 0 = \textcolor{red}{0} 0 \dots 0 \pmod{2^n}$$

- 求-1的补码:

$$\text{对于整数补码: } [-1]_{\text{补}} = 2^n - 1 = 11 \dots 1 \quad (n \text{ 个 } 1)$$

$$\text{对于小数补码: } [-1]_{\text{补}} = 2 - 1 = \textcolor{blue}{1}.00 \dots 0 \quad (n-1 \text{ 个 } 0)$$

例. 设补码的位数为 n , 求负数 -2^{n-1} 的补码表示
(如 -2^7)。

• 由补码定义:

$$\begin{aligned} [-2^{n-1}]_{\text{补}} &= 2^n - 2^{n-1} = 2^{n-1} \\ &= \mathbf{1} 0 \dots 0 \text{ (} n-1 \text{ 个} 0 \text{)} \end{aligned}$$

例. 设补码的位数为6, 求负数 -0.10110 的补码表示。

$$\begin{aligned} [-0.10110]_{\text{补}} &= 2 - 0.10110 \\ &= 10.00000 - 0.10110 \\ &= \mathbf{1}.01010 \end{aligned}$$

▲ 补码能表示的数值范围

- 一个n位定点整数补码能表示的数值范围是：

$$-2^{n-1} \leq \mathbf{X} \leq 2^{n-1} - 1$$

- 一个n位定点小数补码能表示的数值范围是：

$$-1 \leq \mathbf{X} \leq 1 - 2^{-(n-1)}$$

- n=8，定点整数补码数值范围是：

$$-2^{8-1} \leq \mathbf{X} \leq 2^{8-1} - 1 \qquad -128 \sim +127$$

- ▲ 求**负数补码**的简单方法：“符号位固定为1，其余各位由真值中相应**各位取反后，末尾加1**所得”。
- 例。已知： $X = -0.1011010$ ，求 $[X]_{\text{补}}$ 。

$$\begin{aligned}[X]_{\text{补}} &= \mathbf{1.0100101} + \mathbf{0.0000001} \\ &= \mathbf{1.0100110}\end{aligned}$$

例。已知： $X = -1011010$ ，求 $[X]_{\text{补}}$ 。

$$\begin{aligned}[X]_{\text{补}} &= \mathbf{1\ 0100101} + \mathbf{0\ 0000001} \\ &= \mathbf{1\ 0100110}\end{aligned}$$

- ▲ 由补码求真值的简便方法：“若符号位为1，则真值的符号为负，其数值部分的各位由补码中相应各位取反后，末尾加1所得”。

例. 已知： $[X]_{\text{补}} = 1\ 011010$ ，求X。

$$X = -100101 + 1 = -100110$$

- ▲ 求一个补码其真值“取负”后的补码表示方法：“只要对该已知补码各位取反，末尾加1即可。”

例. 已知： $[X]_{\text{补}} = 1\ 011010$ ，求 $[-X]_{\text{补}}$

$$[-X]_{\text{补}} = 0\ 100101 + 1 = 0\ 100110$$

▲ 补码加减法

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

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

例：设 $[X]_{\text{补}} = 1.0101010$ ， $[Y]_{\text{补}} = 1.0111100$ ，请用补码求和方法计算 $(X - Y)$ 。

解：因为 $[-Y]_{\text{补}} = 0.1000100$

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

$$= 1.0101010 + 0.1000100$$

$$= 1.1101110$$

$$\text{所以 } (X - Y) = -0.0010010$$

▲ 由 $[X]_{\text{补}}$ 求 $[X/2]_{\text{补}}$ 算法（定点小数）：

设 $[X]_{\text{补}} = X_0. X_1 X_2 \dots X_{n-2} X_{n-1}$ ，由补码定义得：

$$[X]_{\text{补}} = X + 2X_0,$$

$$X = [X]_{\text{补}} - 2X_0 = X_0. X_1 X_2 \dots X_{n-2} X_{n-1} - 2X_0$$

$$= -X_0 + \sum_{i=1}^{n-1} X_i \cdot 2^{-i}$$

$$\frac{1}{2} X = -\frac{X_0}{2} + \frac{1}{2} \sum_{i=1}^{n-1} X_i \cdot 2^{-i} = -X_0 + \sum_{i=0}^{n-1} X_i \cdot 2^{-(i+1)}$$

$$[X/2]_{\text{补}} = X/2 + 2X_0 = X_0. X_0 X_1 X_2 \dots X_{n-2} X_{n-1}$$

- 由 $[X]_{\text{补}}$ 求 $[X/2]_{\text{补}}$ 的方法：将 $[X]_{\text{补}}$ 的符号位和数值位一起向右移动一位，符号位移走后还保持原来的值不变。

例4. 设 $[X]_{\text{补}} = 1.0101000$ ，则：

$$[X/2]_{\text{补}} = 1.1010100$$

$$[X/4]_{\text{补}} = 1.1101010$$

$$[X/8]_{\text{补}} = 1.1110101$$

(3) 变形补码

- 为了便于判断运算结果是否溢出，计算机中采用一种双符号位的补码表示方式，称为变形补码。

例：设 $x=0.11010$ ， $y=-0.11010$ ，则

$$[x]_{\text{补}} = 0.11010; \quad [y]_{\text{补}} = 1.00110$$

$$[x]_{\text{变补}} = 00.11010; \quad [y]_{\text{变补}} = 11.00110$$

- 当运算结果的变形补码最左边两位为“01”或“10”时，则表示发生溢出。“01”为正溢出，“10”为负溢出。

3. 反码表示法

- 负数反码的定义就是在相应的补码表示中再末尾减1（只求反而不加1）；
- 正数的原码、反码和补码相同；
- 负数的反码定义如下：

① 定点负整数：

$$[X]_{\text{反}} = (2^n - 1) + X$$

$$-2^{n-1} < X \leq 0$$

$$\text{mod } 2^n - 1 = 11...11$$

② 定点负小数:

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

$$\text{mod } 2 - 2^{-(n-1)} = 1.1\dots 11$$

- 反码表示的零有两种:

$$[+0]_{\text{反}} = 0.00\dots 0$$

$$[-0]_{\text{反}} = 1.11\dots 1$$

- 对反码而言，模数不是2而是 $(2 - 2^{-(n-1)})$ ，这意味着若运算中如果最高位有进位，就必须把它加到末位上去。这叫“**循环进位**”。

例. 设 $a=+0.1011$, $b=-0.1001$,
试用反码计算 $(a+b)$ 。

$$[a]_{\text{反}} = 0.1011, [b]_{\text{反}} = 1.0110$$

$$\begin{array}{r} 0.1011 \\ +) 1.0110 \\ \hline \boxed{1}0.0001 \\ \begin{array}{l} | \\ +) \quad \rightarrow 1 \dots \dots \text{循环进位} \end{array} \\ \hline 0.0010 \dots \dots \text{正确结果} \end{array}$$

4. 移码表示法

- 对于定点**小数X**，移码表示的定义为：

$$[X]_{\text{移}} = 1 + X \quad -1 \leq X < 1$$

- 对于n位二进制**整数X**，移码表示的定义为：

$$[X]_{\text{移}} = 2^{(n-1)} + X \quad -2^{(n-1)} \leq X < 2^{(n-1)}$$

- 移码表示的数值范围为：

小数： $-1 \leq X \leq 1 - 2^{-(n-1)}$

整数： $-2^{(n-1)} \leq X_T \leq 2^{(n-1)} - 1$ (编码范围?)

例. 设 $X=27$, $Y=-27$, $n=8$ 时, 求 $[X]_{\text{移}}$ 和 $[Y]_{\text{移}}$ 。

解: 本题 $n=8$, $2^{n-1}=128$, 即10000000, 所以

$$[X]_{\text{移}} = 1000000 + 00011011 = 10011011$$

$$[Y]_{\text{移}} = 1000000 - 00011011 = 01100101$$

- 移码与补码相比, 是符号位不同, 其余位相同。
- 在移码表示中, 真值“0”的表示是唯一的:

$$[0]_{\text{移}} = 100\dots00$$

• 表3.1 四种编码表示所代表的数值

$x_3x_2x_1x_0$	原码	反码	补码	移码
0000	0	0	0	-8
0001	+1	+1	+1	-7
0010	+2	+2	+2	-6
0011	+3	+3	+3	-5
0100	+4	+4	+4	-4
0101	+5	+5	+5	-3
0110	+6	+6	+6	-2
0111	+7	+7	+7	-1
1000	-0	-7	-8	0
1001	-1	-6	-7	1
1010	-2	-5	-6	2
1011	-3	-4	-5	3
1100	-4	-3	-4	4
1101	-5	-2	-3	5
1110	-6	-1	-2	6
1111	-7	-0	-1	7

5. 四种编码的比较

- (1) 原码、补码、反码都是为了解决负数在机器中的表示而提出的。
- (2) 常用移码表示浮点数的阶码。
- (3) 原码和反码都有+0和-0两种零的表示，而补码可唯一表示零。
- (4) 补码和反码的符号位可作为数值的一部分看待，可以和数值位一起参加运算。而原码的符号位必须和代表绝对值的数值位分开处理。

(5) 原码和反码能表示的正数和负数的范围相对零来说是对称的。补码的表示范围不对称，负数表示的范围较正数宽，能表示最小负数为 -2^{n-1} 或-1。

(6) 各种编码采用不同的方法进行移位处理。

- 对于带符号的定点数，进行算数移位；符号位不动，右移一位，意味着原数缩小一倍；左移一位，意味着原数扩大一倍。

- 各种编码的数值部分的**移位规则**如下：

① 原码

左移：高位移出，末位补0；移出非零时，发生溢出。

右移：高位补0，低位移出；移出时进行舍入操作。

② 补码

左移：高位移出，末位补0；移出的位**不同于符号位时**，发生溢出。

右移：高位补符，低位移出；移出时进行舍入操作。

③ 反码

左移：**高位移出，末位补符**；移出的位不同于符号位时，发生溢出。

右移：高位补符，低位移出；移出时进行舍入操作。

(7) 各种编码采用不同的方法进行填充处理（位扩充）：

① 原码

定点小数：在原数的末位后面补足0。

定点整数：符号位不变，在原数的符号位后补足0。

② 补码

定点小数：在原数的末位后面补足0。

定点整数：符号位不变，在原数的符号位后用数符补足所需的位数。

6. 无符号数的表示

- 当一个编码的**所有二进位**都用来表示数值时，该编码表示的是无符号数；
- 在全部是正数运算且**不出现负值**结果的情况下，使用无符号数表示，如地址编码；
- 无符号**整数**($n=8$)表示数值的范围；
- 无符号**小数**($n=8$)表示数值的范围。

3.2.4 浮点数的编码表示及IEEE 754浮点标准

1. 编码表示

- 用定点小数表示浮点数的尾数，用定点整数表示浮点数的阶。
- 浮点数的阶用移码表示；对每个阶都加上一个正的常数（称为偏置常数），使所有阶都转化为正整数。

数符	阶码	尾数
----	----	----

▲ 浮点数的规格化

- 若尾数用**基为2的补码表示**，则规格化数的标志为：
“尾数的符号位和数值部分最高位**具有不同的代码**”。
- 左规

若尾数用**变形补码**，则当结果的尾数出现**11.1xx...x**或**00.0xx...x**的形式时，需将尾数左移，阶减1，直到尾数为规格化数形式为止。

- 右规

当浮点运算结果的尾数出现**01.xx...x**或**10.xx...x**的形式时，**并不一定溢出**，应先将它右移1位，阶码加1，然后再**判断阶是否溢出**。

例1. 将十进制数65798转换为下述**IBM370**的短浮点数格式（32位）。

IBM370的短浮点数格式说明如下：



- **说明：** 0位：数符S； 1-7位：7位移码表示的阶码E（偏置常数=64）； 8-31位：6位**16**进制原码小数表示的尾数M（基**R=16**，所以阶码变化1等于尾数移动4位）。

解：

$$\because (65798)_{10} = (10106)_{16} = (0.101060)_{16} \times 16^5$$

$$\therefore \text{数符} S=0, \text{阶码} E=(\mathbf{64}+5)_{10}=(69)_{10}$$

$$=(\mathbf{1000101})_2$$

- 浮点数形式表示为：

0	1000101	0001	0000	0001	0000	0110	0000
---	---------	------	------	------	------	------	------

例2. 将十进制数65798 转换为下述典型的32位浮点数格式。



- 浮点数格式

0位：数的符号

1-8位：8位移码表示的阶码（偏置常数为128）

9-31位：24位二进制原码小数表示的尾数。规格化尾数的第一位总是1，故不保存。即虽只有23位，但可表示24位数据（基 $R=2$ ）。

$$\begin{aligned}
 \because (65798)_{10} &= (10106)_{16} \\
 &= (1\ 0000\ 0001\ 0000\ 0110)_2 \\
 &= (0.\textcolor{blue}{1}000\ 0000\ 1000\ 0011)_2 \times 2^{17}
 \end{aligned}$$

\therefore 数符S=0, 阶码E=(128+17)₁₀=(145)₁₀=(10010001)₂

• 故用该浮点数形式表示为:

0	1 0010001	000 0000 1000 0011 0000 0000
---	------------------	------------------------------

• **原码**规格化尾数的第一位总是1, 这里不保存

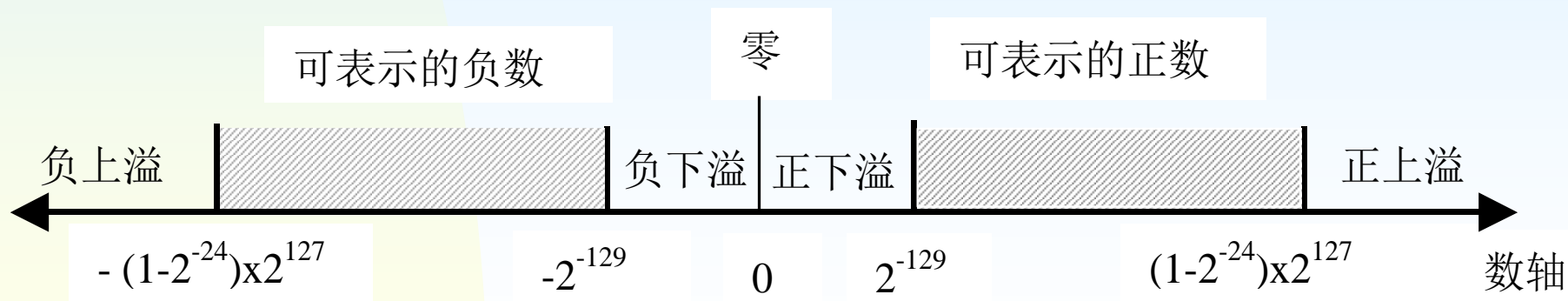
2. 浮点数编码表示的数值范围



- 上述格式(典型32位)的规格化浮点数的表示范围为:

正数: 最大值: $0.11...1 \times 2^{11...1} = (1-2^{-24}) \times 2^{127}$

最小值: $0.10...0 \times 2^{00...0} = (1/2) \times 2^{-128} = 2^{-129}$



- 定点数分布是等距且紧密的，而浮点数分布是不等距且稀疏的，越远离原点越稀疏。

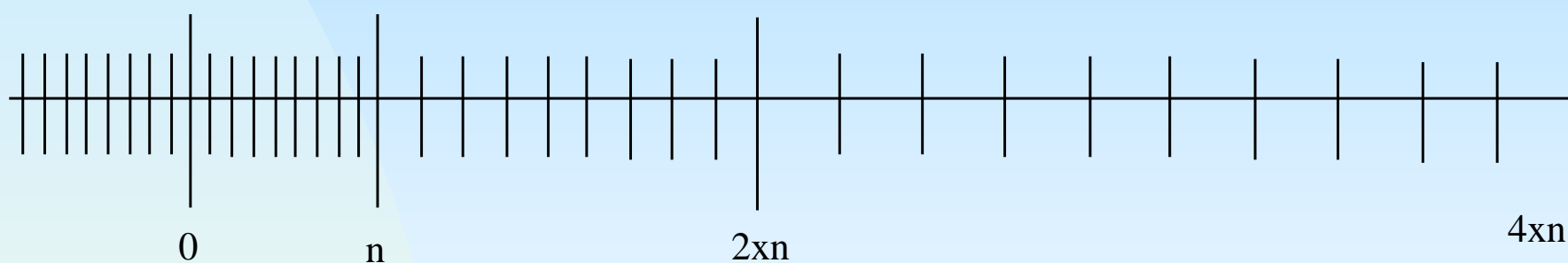


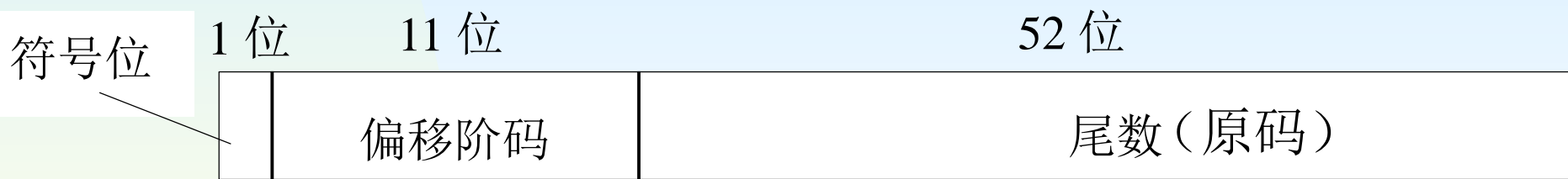
图3.2 浮点数的密度

- 在浮点数总位数不变的情况下，其阶码位数越多，表数范围越大，则精度越差。
- 提供单精度和双精度两种浮点数格式。

3. IEEE754 浮点数标准



(a) 单精度格式



(b) 双精度格式

图3.3 IEEE754 浮点数格式

表3.2 IEEE754 浮点数格式说明

参数	单精度	双精度
字宽（位数）	32	64
阶码宽度（位数）	8	11
阶码偏移量	127	1023
最大阶码	127	1023
最小阶码	-126	-1022
数的范围	$10^{-38} \sim 10^{+38}$	$10^{-308} \sim 10^{+308}$
尾数宽度	23	52
阶码个数	254	2046
尾数个数	2^{23}	2^{52}
值的个数	1.98×2^{31}	1.99×2^{63}

单精度(32位)

	符号	偏置指数	尾数	值
正零	0	0	0	0
负零	1	0	0	-0
正无穷	0	255(全1)	0	∞
负无穷	1	255(全1)	0	$-\infty$
无定义数	0或1	255(全1)	$\neq 0$	NaN
规格化非零正数	0	$0 < e < 255$	f	$2^{e-127}(1.f)$
规格化非零负数	1	$0 < e < 255$	f	$-2^{e-127}(1.f)$
非规格化正数	0	0	$f \neq 0$	$2^{e-126}(0.f)$
非规格化负数	1	0	$f \neq 0$	$-2^{e-126}(0.f)$

双精度(64 位)

	符号	偏置指数	尾数	值
正零	0	0	0	0
负零	1	0	0	-0
正无穷	0	2047(全1)	0	∞
负无穷	1	2047(全1)	0	$-\infty$
无定义数	0或1	2047(全1)	$\neq 0$	NaN
规格化非零正数	0	$0 < e < 2047$	f	$2^{e-1023}(1.f)$
规格化非零负数	1	$0 < e < 2047$	f	$-2^{e-1023}(1.f)$
非规格化正数	0	0	$f \neq 0$	$2^{e-1022}(0.f)$
非规格化负数	1	0	$f \neq 0$	$-2^{e-1022}(0.f)$

- IEEE754 格式浮点数真值的计算公式为：

规格化单精度浮点数格式： $(-1)^s \times 1.f \times 2^{e-127}$

规格化双精度浮点数格式： $(-1)^s \times 1.f \times 2^{e-1023}$

- 例3. 十进制数-0.75 转换为IEEE754的单精度浮点数格式。

解： $(-0.75)_{10} = (-0.11)_2 = (-1.1)_2 \times 2^{-1} = (-1)^s \times 1.f \times 2^{e-127}$

所以： $s = 1, f = 0.100\dots 0,$

$e = (127-1)_{10} = (126)_{10} = (01111110)_2$

规格化浮点数表示为：

1 **0111 1110** 1000 0000...0000 000

3.2.5 十进制数的二进制编码表示（BCD码）

1. 十进制有权码

- 十进制有权码是指表示每个十进制数位的四个二进制数位（称为基2码）都有一个确定的权。

2. 十进制无权码

- 十进制无权码是指表示每个十进制数位的四个基2码没有确定的权。

• 表3.3 四位有权码

十进制数	8421码	2421码	5211码	84-2-1码	4311码
0	0000	0000	0000	0000	0000
1	0001	0001	0001	0111	0001
2	0010	0010	0011	0110	0011
3	0011	0011	0101	0101	0100
4	0100	0100	0111	0100	1000
5	0101	1011	1000	1011	0111
6	0110	1100	1010	1010	1011
7	0111	1101	1100	1001	1100
8	1000	1110	1110	1000	1110
9	1001	1111	1111	1111	1111

• 表3.4 四位无权码

十进制数	余3码	格雷码(1)	格雷码(2)
0	0011	0000	0000
1	0100	0001	0100
2	0101	0011	0110
3	0110	0010	0010
4	0111	0110	1010
5	1000	1110	1011
6	1001	1010	0011
7	1010	1000	0001
8	1011	1100	1001
9	1100	0100	1000

3.3 非数值数据的编码表示

3.3.1 逻辑数据

- **逻辑数据**：将一个 n 位数据看成是由 n 个1位项组成，每项取值为0或1，表示逻辑值“假”和“真”。
- 逻辑数据只能参加逻辑运算，按位进行。
- 机器不能识别逻辑数据和数值数据，是靠指令的**操作码类型**来识别的。

3.3.2 字符

◆ 西文字符

- **西文字符**是由拉丁字母、数字、标点符号及一些特殊符号所组成的；所有字符的集合叫做“**字符集**”。
- 必须对字符进行**数字化编码**才在计算机内部进行处理，字符集中每一个字符都有一个代码，**代码具有唯一性**，互相区别。
- 构成了该字符集的代码表，简称码表。
- **ASCII码**，即美国标准信息交换码。

◆ 汉字字符

- 汉字系统必须处理以下几种汉字代码：
输入码 / 内码 / 字模点阵码。

1. 汉字的输入码

- 对每个汉字用相应的按键进行的编码表示就称为汉字的“**输入码**”，又称外码。

- 汉字输入编码方法大体分成四类：

- (1)数字编码：用一串数字来表示汉字的编码。

- (2)字音编码：基于汉语拼音的编码。

- (3)字形编码：将汉字的字形分解归类而给出的编码。

- (4)形音编码：将汉字读音和形状结合起来考虑的编码。

- 常见的有国标码、区位码、拼音码和五笔字型。

- 国标码（十六进制）= 区位码（十六进制）+ **2020H**

2. 字符集与汉字内码

- **汉字内码**：机器内部处理和存储汉字的一种代码。
- 汉字内码的选择，必须考虑以下几个因素：
 - (1) 不能有二义性，即不能和ASCII码有相同的编码。
 - (2) 要与汉字在字库中的位置有关系，以便于汉字的处理、查找。
 - (3) 编码应尽量短。
- 国标码：《信息交换用汉字编码字符集 基本集》(GB2312—80)。
- 机内码（十六进制）= 国标码（十六进制）+ **8080H**

3. 汉字的输出和汉字字库

- 一套汉字(GB2312)的所有字符的**形状描述信息**集合在一起称为**字形信息库**。
- 汉字的字形有两种描述方法：字模点阵描述和轮廓描述。
- 对于 16×16 点阵码，每个汉字用32个字节来表示；字库有**软字库**和**硬字库**之分。

3.3.3 多媒体信息

1. 图的编码表示

- 一种称为“图象”(image)方式，另一种称为“图”(graphics)方式。

2. 声音的编码表示

- 计算机处理的声音可分为三种：
 - 语音，即人的说话声；
 - 音乐，即各种乐器演奏出的声音；
 - 效果声，如掌声、打雷、爆炸等声音。
- 在计算机内部可用波形法和合成法两种方法来表示声音。

- 将声波波形转换成二进制表示形式的过程称为声音的“**数字化编码**”。
- 声音的数字化编码过程分为三步：
 - (1) 首先以固定的时间间隔对声音波形进行采样，使连续的声音波形变成**离散的采样信号**；每秒钟采样的次数被称为**采样频率**。
 - (2) 对得到的每个样本值进行**模数转换**，用一个二进制数来表示。这个过程即是所谓的**量化处理**。
 - (3) 最后对产生的二进制数据进行编码（有时还需进行数据压缩），按照规定的**统一格式**进行表示。

● 表3.5 几种不同参数的数字化声音信息

采样 频率 (kHz)	量化 精度 (比特)	每分钟的 数据量 (MB)	质量与应用
44.1	16	5.3	相当于激光唱片质量，用于最高质量要求的场合。
22.05	16 8	2.65 1.32	相当于调频广播质量，可用作伴音和声响效果。
11.025	16 8	1.32 0.66	相当于调幅广播质量，可用作伴音和解说词。

- 采用合成法的MIDI表示，其数据量要少得多（相差2-3个数量级），编辑、修改也较容易。主要适用于表现各种乐器所演奏的乐曲，不能用来表示语音等其它声音。
- 用声卡来完成对各种声音输入与数字化编码处理，并将处理后的数字波形声音还原为模拟信号声音，经功率放大后输出。

3. 视频信息的编码表示

- 通过**视频获取**设备（视频卡），将各类视频源（电视机、摄像机、VCD机或放像机）输入的视频信号转化为计算机内部可以表示的二进制数字信息。
- 对亮度(Y)、色差(U, V)三个分量分别进行采样和量化，得到一幅数字图象。

表3.6 几种常用数字视频的格式

名称	分辨率	量化精度	每秒钟的数据量
CCIR601	$720 \times 576 \times 25$	8+4+4	124Mb
CIF	$360 \times 288 \times 25$	8+4+4	26Mb
QCIF	$180 \times 144 \times 25$	8+4+4	6.5Mb

- 数字视频信息的数据量非常大，必须对数字视频信息再进行压缩编码。
- 在获取数字视频的同时就立即进行压缩编码，称为实时压缩。

3.4 数据校验码

- 为减少和避免计算机系统运行或传送过程中发生错误，在数据的编码上提供检错和纠错的支持。
- 能够发现某些错误或具有纠错能力的编码称为**数据校验码**或检错码。
- 常采取计算机硬件进行检错和纠错。
- “**冗余校验**”：除原数据信息外，还增加若干位编码，这些新增的代码被称为**校验位**。

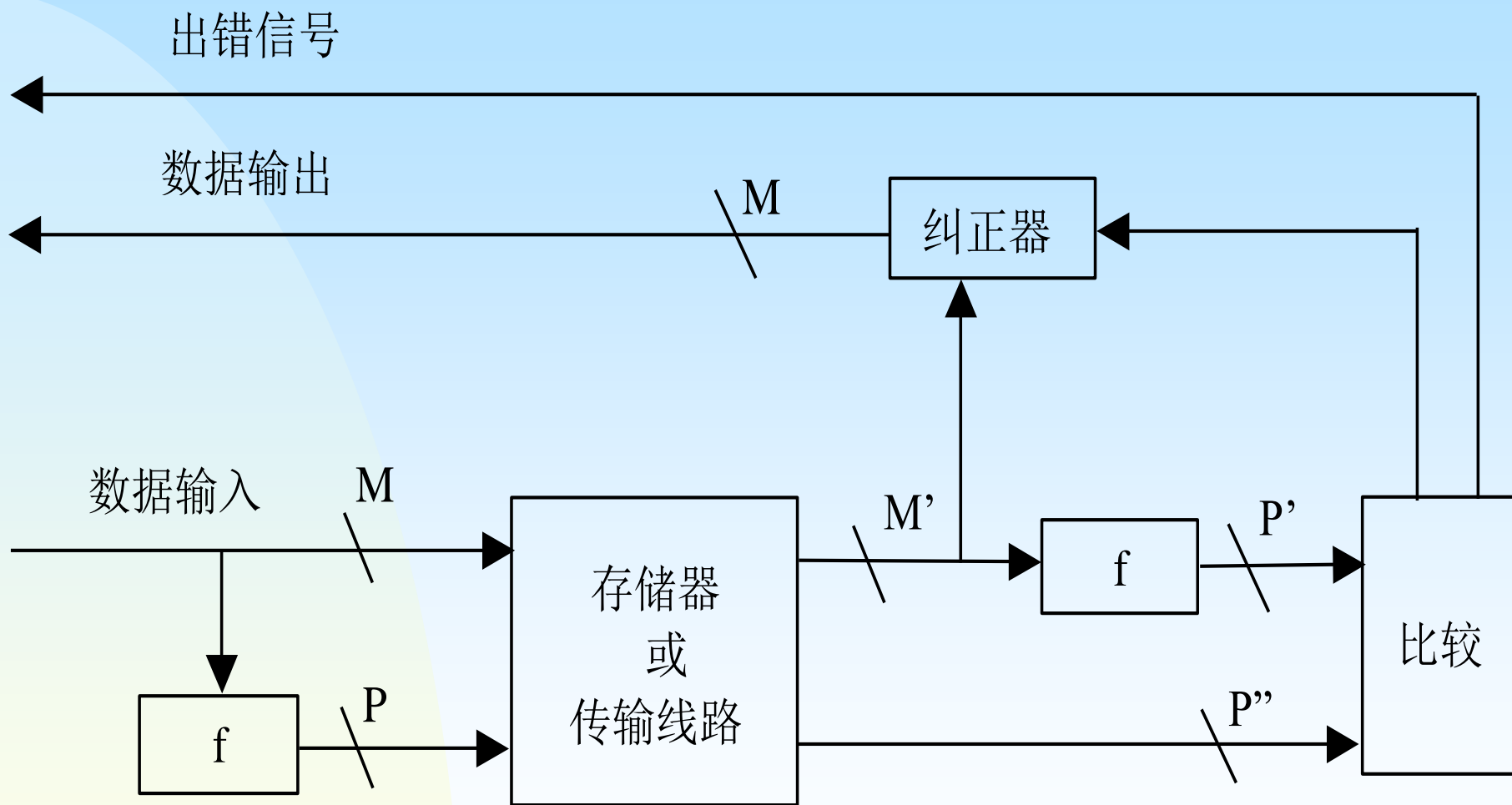


图3.4 数据校验过程示意图

“**码字**”：数据位和校验位按照某种规律排列得到的代码。

“**距离**”：将两个码字逐位比较，具有不同代码的位的个数；

“**码距**”：各码字间的最小距离。

- 例：“8421”编码中，2（0010）和3（0011）之间距离为1，所以“8421”码制的码距为1，记作 $d=1$ 。

- 码距与检错、纠错能力的关系为:

① 如果码距 d 为奇数, 则能发现 $d-1$ 位错,
或者能纠正 $(d-1)/2$ 位错。

② 如果码距 d 为偶数, 则能发现 $d/2$ 位错,
并能纠正 $(d/2-1)$ 位错。

3.4.1 奇偶校验码

◆ 奇偶校验法的基本思想

- 第一步：在源部件求出奇（偶）校验位P。

采用奇校验位， $P=b_{n-1} \oplus b_{n-2} \oplus \dots \oplus b_1 \oplus b_0 \oplus 1$ 。

采用偶校验位， $P=b_{n-1} \oplus b_{n-2} \oplus \dots \oplus b_1 \oplus b_0$ 。

- 第二步：在终部件求出奇（偶）校验位P'。

采用奇校验位， $P'=b_{n-1}' \oplus b_{n-2}' \oplus \dots \oplus b_1' \oplus b_0' \oplus 1$ 。

采用偶校验位， $P'=b_{n-1}' \oplus b_{n-2}' \oplus \dots \oplus b_1' \oplus b_0'$ 。

- 第三步：计算最终的校验位 P^* ，并根据其值判断有无奇偶错。
- 假定 P 在终部件接受到的值为 P'' ，则采用异或操作 $P^*=P' \oplus P''$ ，
 - ① 若 $P^*=1$ ，表示终部件接受的数据有奇数位错。
 - ② 若 $P^*=0$ ，表示终部件接受的数据正确或有偶数个错。

◆ 奇偶校验法的特点

- 任意两个码字之间至少有一位不同，码距 $d=2$ 。
- 只能发现奇数位出错，不能发现偶数位出错，也不具有纠错能力。
- 奇偶校验法的开销小，常被用于存储器读写检查或按字节传输过程中的数据校验。
- 奇偶校验码用于校验一字节长的代码还是有效的。

3.4.2 海明校验码

- ◆ 主要用于存储器中数据存取校验
 - 海明校验码实质上是一种**多重奇偶校验码**；能对错误位置进行定位，并将其纠正。
 - **故障字**（syndrome word）

1. 校验位的位数的确定

- 假定该数据的位数为n，**校验位为k**，组成**n+k**位码字；则故障字的位数也为k位。
- n和k必须满足下列关系：

$$2^K \geq 1+n+k, \quad \text{即: } 2^K - 1 \geq n+k$$

数据位	单纠错		单纠错/双检错	
	检查位	增加的百分比	检查位	增加的百分比
8	4	50	5	62.5
16	5	31.25	6	37.5
32	6	18.75	7	21.875
64	7	10.94	8	12.5
128	8	6.25	9	7.03
256	9	3.25	10	3.91

- 如故障字 $S_4 S_3 S_2 S_1$ 各位全部是0，表示没有发生错误。
- 如故障字仅有一位为1，表示校验位中有一位出错，不需要纠正。
- 如故障字中多位为1，表示有一个数据位出错，出错位置由故障字的数值来确定；纠正时只要将出错位取反即可。

3. 校验位的生成和检错、纠错

- 对于码字： $M_8M_7M_6M_5P_4M_4M_3M_2P_3M_1P_2P_1$
- 采用偶校验，源部件校验位为：

$$P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7$$

$$P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7$$

$$P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8$$

$$P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8$$

- 终部件码字为：

$$M_8' M_7' M_6' M_5' P_4'' M_4' M_3' M_2' P_3'' M_1' P_2'' P_1''$$

- 到达终部件后的校验位： $P'' = P_4'' P_3'' P_2'' P_1''$

- 重新校验后的校验位： $P' = P_4' P_3' P_2' P_1'$

$$P_1' = M_1' \oplus M_2' \oplus M_4' \oplus M_5' \oplus M_7'$$

$$P_2' = M_1' \oplus M_3' \oplus M_4' \oplus M_6' \oplus M_7'$$

$$P_3' = M_2' \oplus M_3' \oplus M_4' \oplus M_8'$$

$$P_4' = M_5' \oplus M_6' \oplus M_7' \oplus M_8'$$

- 故障字： $S = P'' \oplus P'$

- 例：假定一个8位数据M为：

$$\mathbf{M_8M_7M_6M_5M_4M_3M_2M_1=01101010。}$$

- 根据上述公式求出相应的校验位为：

$$\mathbf{P_1 = M_1 \oplus M_2 \oplus M_4 \oplus M_5 \oplus M_7 = 0 \oplus 1 \oplus 1 \oplus 0 \oplus 1 = 1}$$

$$\mathbf{P_2 = M_1 \oplus M_3 \oplus M_4 \oplus M_6 \oplus M_7 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1}$$

$$\mathbf{P_3 = M_2 \oplus M_3 \oplus M_4 \oplus M_8 = 1 \oplus 0 \oplus 1 \oplus 0 = 0}$$

$$\mathbf{P_4 = M_5 \oplus M_6 \oplus M_7 \oplus M_8 = 0 \oplus 1 \oplus 1 \oplus 0 = 0}$$

(1) 数据位 $M'=M=01101010$ ，校验位 $P''=P=0011$ ，所有位都无错。

- 这种情况下，因 $M'=M$ ，所以 $P'=P$ ，因此 $S = P'' \oplus P' = P \oplus P = 0000$ 。

(2) 数据位 $M'=01111010$ ，校验位 $P''=P=0011$ ，
即：数据位第5位(M_5)错。

- 这种情况下，对M'生成新的校验位P'为：

$$P_1' = M_1' \oplus M_2' \oplus M_4' \oplus M_5' \oplus M_7' = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$P_2' = M_1' \oplus M_3' \oplus M_4' \oplus M_6' \oplus M_7' = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$P_3' = M_2' \oplus M_3' \oplus M_4' \oplus M_8' = 1 \oplus 0 \oplus 1 \oplus 0 = 0$$

$$P_4' = M_5' \oplus M_6' \oplus M_7' \oplus M_8' = 1 \oplus 1 \oplus 1 \oplus 0 = 1$$

- 故障字 $S = P'' \oplus P' = 1001$

$$S_1 = P_1' \oplus P_1'' = 0 \oplus 1 = 1 \quad S_2 = P_2' \oplus P_2'' = 1 \oplus 1 = 0$$

$$S_3 = P_3' \oplus P_3'' = 0 \oplus 0 = 0 \quad S_4 = P_4' \oplus P_4'' = 1 \oplus 0 = 1$$

(3) 数据位 $M'=M=01101010$ ，校验位 $P''=1011$ ，
即：校验位第4位(P_4)错。

- 这种情况下，因 $M'=M$ ，所以 $P'=P$ ；
因此故障位 S 为：

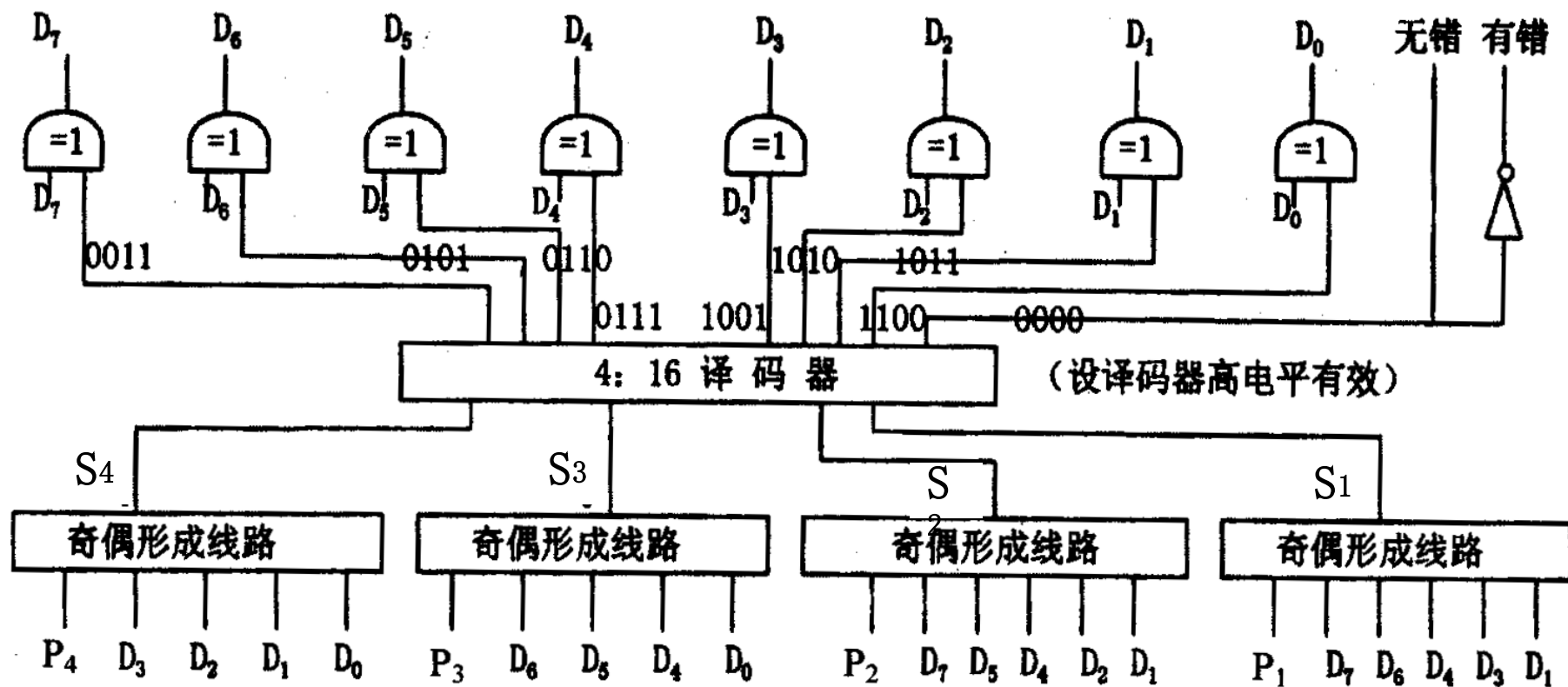
$$S_1 = P_1' \oplus P_1'' = 1 \oplus 1 = 0$$

$$S_2 = P_2' \oplus P_2'' = 1 \oplus 1 = 0$$

$$S_3 = P_3' \oplus P_3'' = 0 \oplus 0 = 0$$

$$S_4 = P_4' \oplus P_4'' = 0 \oplus 1 = 1$$

- ◆ 对单个位出错进行定位和纠错。这种码称为单纠错码 (SEC)。



3.4.3 循环冗余校验码(CRC码)

- 通过某种**数学运算**来建立数据和校验位之间的约定关系。

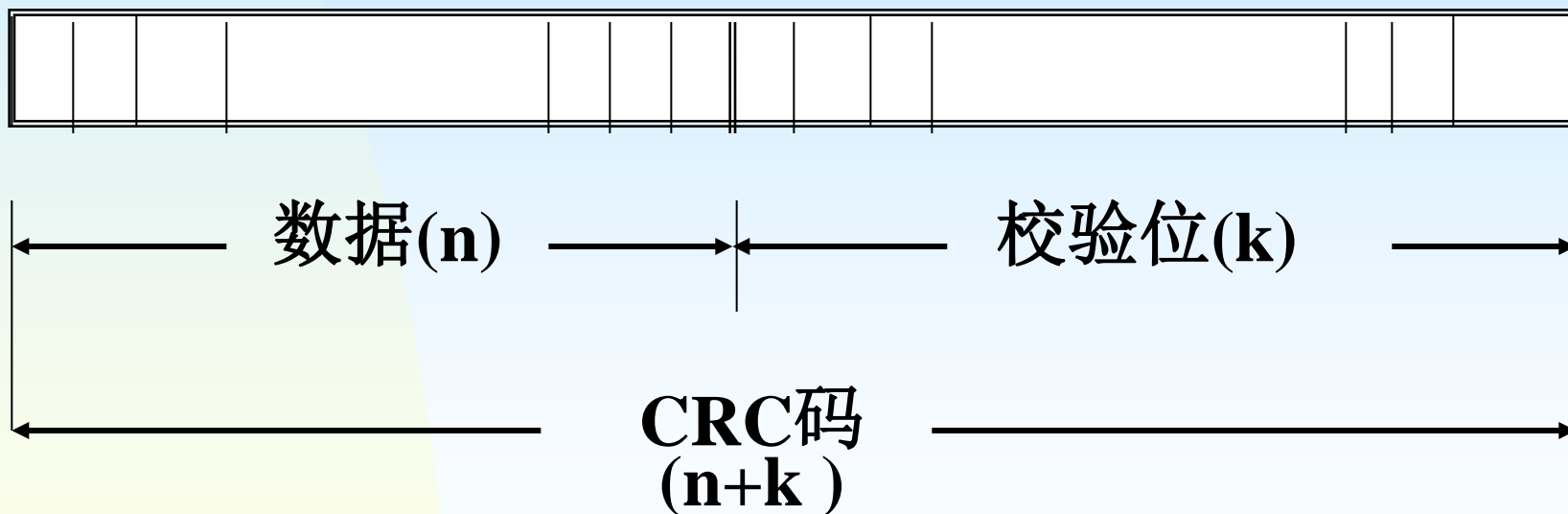
1. CRC码的检错方法

- $M(x)$ 为一个 **n** 位的二进制数， $G(x)$ 是一个 **$k+1$** 位的二进制数；

$$M(x) \times x^k / G(x) = Q(x) + \mathbf{R(x)} / G(x)$$

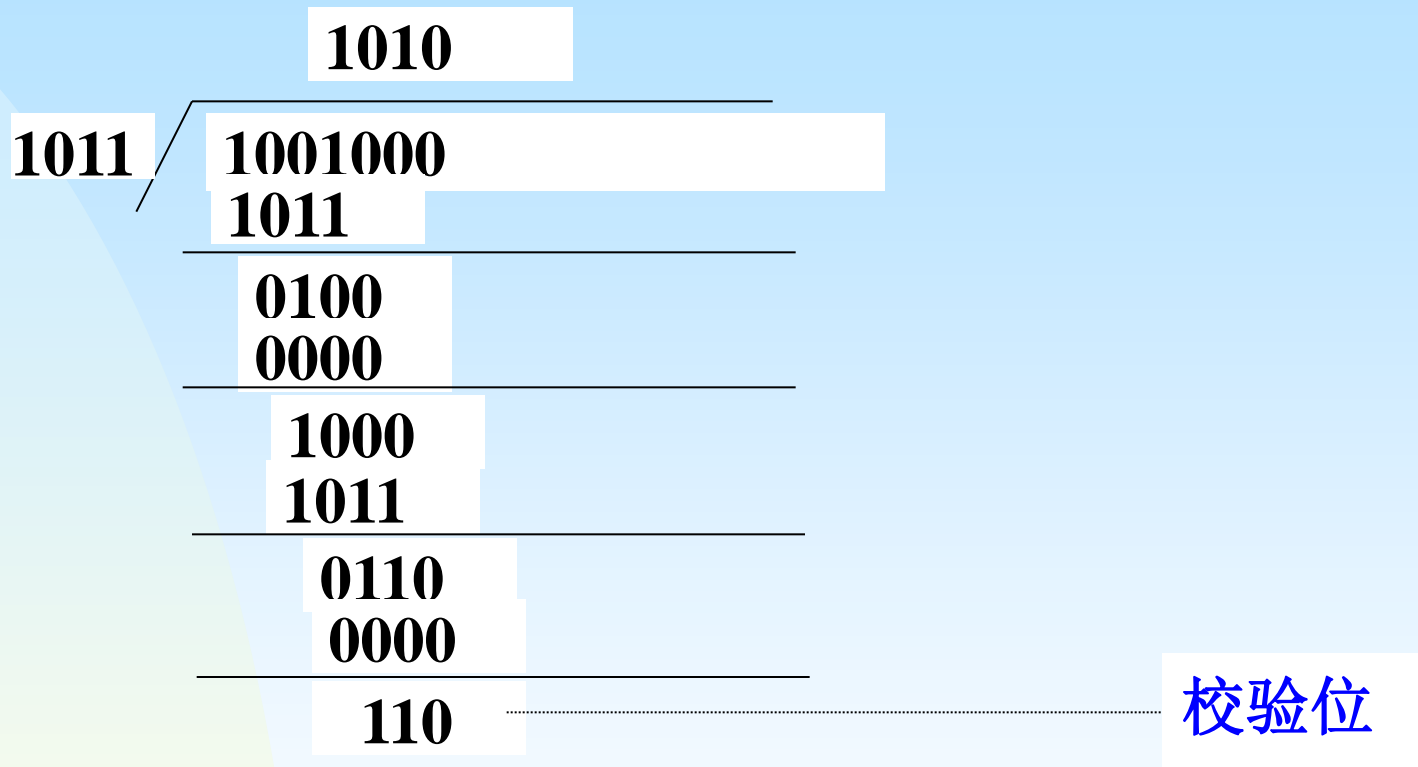
$R(x)$ 为 **k 位余数。**

- $M(x)$ 后接 $R(x)$ 形成一个 $n+k$ 位的代码，称为**循环冗余校验(CRC)码**，也称 $(n+k, n)$ 码。
- 一般要求： $2^k \geq 1+n+k$ 。



2. 校验位的生成

- 要传送的数据信息为：1001，多项式为： $M(x) = x^3 + 1$ ，生成多项式为： $G(x) = x^3 + x + 1$
- 用 $x^3 \times M(x)$ 去除以 $G(x)$ ，相除时采用“模2运算”的多项式除法。
- $x^3 \times M(x) \div G(x) = (x^6 + x^3) \div (x^3 + x + 1)$
余数多项式为 $x^2 + x$ 。



- 校验位为110，CRC码为**1001 110**。

- 若接收方的CRC码与发送方一致，即为1001 110时；

$$\begin{array}{r} 1010 \\ 1011 \overline{) 1001110} \\ \underline{1011} \\ 0101 \\ \underline{0000} \\ 1011 \\ \underline{1011} \\ 0000 \\ \underline{0000} \\ 000 \end{array} \quad \text{余数为0}$$

- 若接收方的CRC码有一位出错而变为10**11** 110时。

$$\begin{array}{r}
 \text{1011} \overline{) 1011110} \\
 \underline{1000} \\
 0011 \\
 \underline{0000} \\
 0011 \\
 \underline{0000} \\
 0110 \\
 \underline{0000} \\
 110
 \end{array}$$

余数不为0

- 任何确定余数编码与出错位的位置？

3. CRC码的纠错

	码字举例		余数	出错位
	$D_1D_2D_3D_4P_1P_2P_3$	$D_1D_2D_3D_4P_1P_2P_3$		
正确	1 0 1 0 0 1 1	1 0 1 1 0 1 0	0 0 0	无
错误	1 0 1 0 0 1 0	1 0 1 1 0 1 1	0 0 1	7
	1 0 1 0 0 0 1	1 0 1 1 0 0 0	0 1 0	6
	1 0 1 0 1 1 1	1 0 1 1 1 1 0	1 0 0	5
	1 0 1 1 0 1 1	1 0 1 0 0 1 0	0 1 1	4
	1 0 0 0 0 1 1	1 0 0 1 0 1 0	1 1 0	3
	1 1 1 0 0 1 1	1 1 1 1 0 1 0	1 1 1	2
	0 0 1 0 0 1 1	0 0 1 1 0 1 0	1 0 1	1

- 生成多项式为1011

4. 生成多项式的选取

▲ 生成多项式应满足的条件：

- 任何一位发生错误时，都应使余数不为0。
- 不同位发生错误时，余数应该不同。
- 对余数作模 2 除时，应使余数循环。

▲ 几种常用的生成多项式:

- **CRC-CCITT:** $G(x)=x^{16}+x^{12}+x^5+1$
- **CRC-12:** $G(x)=x^{12}+x^{11}+x^3+x^2+x+1$
- **CRC-16:** $G(x)=x^{16}+x^{15}+x^2+1$
- **CRC-32:** $G(x)=x^{32}+x^{26}+x^{23}+x^{16}+x^{12}+x^{11}+x^{10}$
 $+x^8+x^7+x^5+x^4+x^2+x+1$

作业一： P78~79—2, 3(1、 2、 4), 4,
—5, 6, 7(2、 4);

作业二： P79—10, 13, 14(-27/1024、 7.375)
—16(1、 2), 18;