西安电子科技大学
XIDIAN UNIVERSITY

第二章 计算机系统中的数据表示

西安电子科技大学

人工智能学院

目录

基本要求

- （1）熟练掌握数的编码表示(原码、补码，变形补码、反码、移码)。
- （2）掌握定点数与浮点数、规格化浮点数的概念
- （3）了解非数值信息的编码表示
- （4）掌握奇偶校验码、海明码、循环码编码方法和用途

目录

◻引言： 逻辑运算

1. 基本逻辑运算

逻辑与（**AND**），也称为逻辑乘，符号：∧或·

逻辑或（**OR**），也称为逻辑加，符号：∨或+

逻辑异或（**XOR**），也称为按位加，模2加，符号⊕

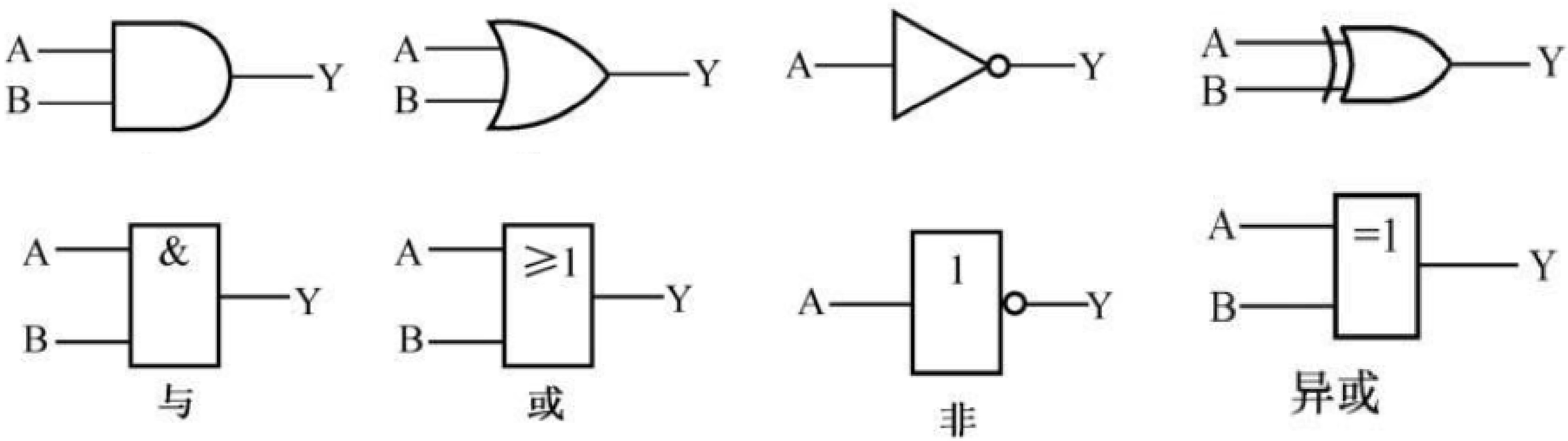
逻辑非（**NOT**），也称为求反，符号：$\bar{X}$

X_i	Y_i	$X_i \cdot Y_i$	X_i+Y_i	$X_i \oplus Y_i$	$\bar{X}_i$
0	0	0	0	0	1
0	1	0`	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

逻辑运算

2. 逻辑运算部件

与门、或门、反相器、异或门



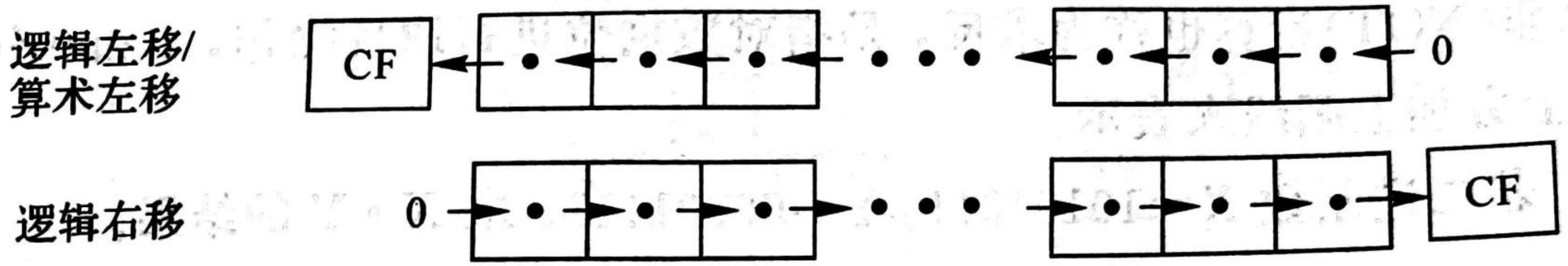
机器数的移位运算

- 计算机的移位运算分为逻辑移位、算术移位、循环移位三种。主要区别在于符号位和移出的数据位的处理方法不同。
- 计算机的移位寄存器的字长是固定的，当进行左移和右移时，寄存器的最低位和最高位会出现空余位，相应地最高位和最低位也会被移出，那么，对空余位补充“0”还是“1”呢？移出的数据位如何处理呢？这与移位的种类和机器数的编码方法有关。

机器数的移位运算

1. 逻辑移位

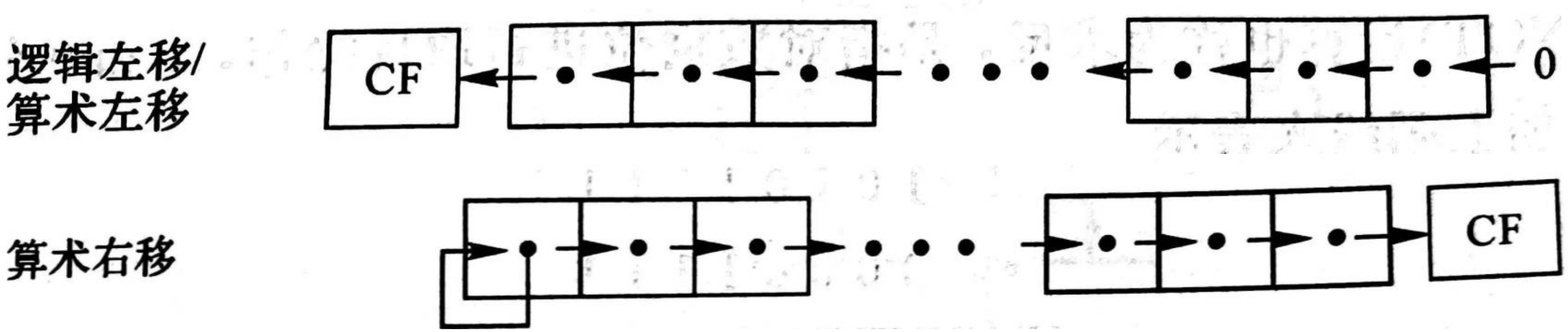
- 将数据视为无符号数据，移位的结果只是数据各位在位置上发生了变化，无符号数据的数值（无正负）放大或缩小。
  - 逻辑左移时，高位移出，低位补“ 0”。
  - 逻辑右移时，低位移出，高位补“ 0”。
  - 移出的数据位一般置入标志位 CF（进位/借位标志）。



逻辑与移位运算

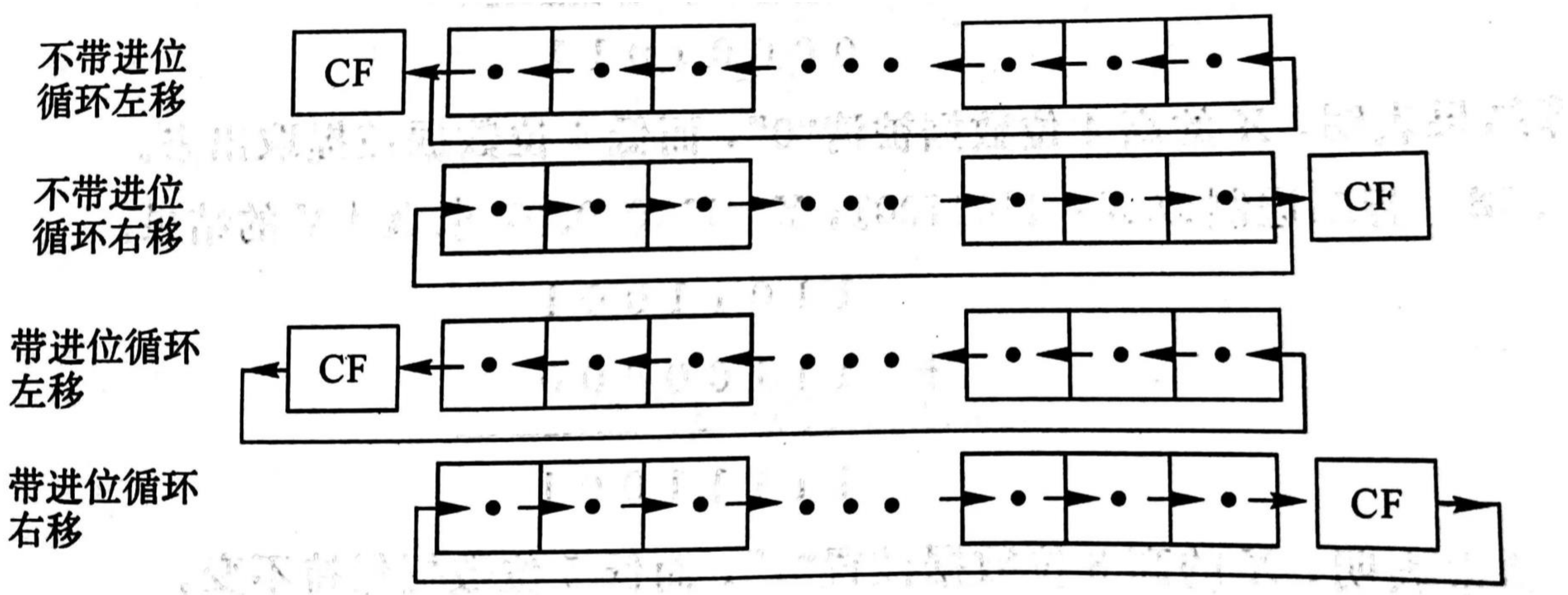
2. 算术移位

- 对象是带符号数据，即各种编码表示的机器数，结果是在数值的绝对值上进行放大或缩小，同时符号位必须要保持不变。
- 算术左移（SAL）：与逻辑左移操作方法相同，高位移入CF标志位，低位补“ 0”。在不超出编码表示范围的前提下，算术左移1位等于对操作数做乘2运算。
- 算术右移（SAR）：最低位移入CF标志位，高位位用符号位填入。对补码而言，算术右移1位等于对操作数做除2运算。



3. 循环移位

- 不带进位的循环左移（ROL）
- 不带进位的循环右移（ROR）
- 带进位的循环左移（RCL）
- 带进位的循环右移（RCR）



数据表示

- 数据：是对事实、概念或指令的一种特殊表达形式，可以用人工方式或自动化装置进行通信、翻译转换或加工处理。
  - 数值型数据：具有特定值的一类数据，可用来表示数量的多少，可比较其大小。
  - 非数值型数据：包括字符数据、逻辑数据、图画、声音和活动图像数据等。
  - 数据表示研究的是计算机硬件能够直接识别、可以被指令系统直接调用的那些基本数据类型。

数据编码

- 在计算机内部，各种数据（也称信息）都必须经过数字化编码后才能被传送、存储和处理。要使计算机能处理各种各样的信息，则必须对这些信息进行编码。
- 编码就是采用少量的基本符号，选用一定的组合原则，以表示大量复杂多样的信息。
- 计算机处理的信息可分为两大类：

  数值信息；

  非数值信息；

数的进制及转换

- 数值数据是表示数量多少和数值大小的数据。即在数轴上能找到其对应的点。
- 机器数：各种数值数据在计算机中的表示形式。
- 真值：机器数对应的实际数值称为数的真值。

计算机常用各种进制数的表示

- 基数：计数制中用到的数码的个数，用**R**表示。
- 位权：以基数为底的指数，指数的幂是数位的序号。
- 对一个数**S**，其基数为**R**，则：

$$
\begin{aligned}
(S)_R &= \pm(\mathrm{K}_{n-1}\mathrm{K}_{n-2}\ldots\mathrm{K}_2\mathrm{K}_1\mathrm{K}_0.\mathrm{K}_{-1}\mathrm{K}_{-2}\ldots\mathrm{K}_{-m}) \\
&= \pm(\mathrm{K}_{n-1}\mathrm{R}^{n-1}+\mathrm{K}_{n-2}\mathrm{R}^{n-2}+\ldots+\mathrm{K}_1\mathrm{R}^1+\mathrm{K}_0\mathrm{R}^0+\mathrm{K}_{-1}\mathrm{R}^{-1}+\ldots+\mathrm{K}_{-m}\mathrm{R}^{-m}) \\
&= \pm\sum_{i=-m}^{n-1}\mathrm{K}_i\mathrm{R}^i
\end{aligned}
$$

1.进位计数制和数制之间的转换

计算机常用各种进制数的表示

进位制	二进制	八进制	十进制	十六进制
规则	逢二进一	逢八进一	逢十进一	逢十六进一
基数	R=2	R=8	R=10	R=16
基本符号	0,1	0,1,2,…,7	0,1,2,…,9	0,1,..,9,A,..,F
权	2^i	8^i	10^i	16^i
形式表示	B	O	D	H

1.进位计数制和数制之间的转换

【例】将二进制数$(11001.01)_2$、八进制数$(216.3)_8$、十六进制数$(7A.C)_{16}$转换成十进制数。

【解】 “按权展开”

$(11001.01)_2$

$=(1\times2^4+1\times2^3+0\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2})_{10}$

$=(25.25)_{10}$

$(216.3)_8$

$=(2\times8^2+1\times8^1+6\times8^0+3\times8^{-1})_{10}$

$=(142.375)_{10}$

$(7A.C)_{16}$

$=(7\times16^1+10\times16^0+12\times16^{-1})_{10}$

$=(122.75)_{10}$

十进制转换为二进制数

- 任一十进制数N，$N=N_{整}+N_{小}$。将这两部分分开转换

①整数部分的转换：采用“除2求余法”，转换方法为：连续用2除，求得余数（1或0）分别为K_0、K_1、K_2、…，直到商为0，所有余数排列$K_{n-1}K_{n-2}\ldots K_2K_1K_0$ 即为所转换的二进制整数部分。

②小数部分的转换：采用“乘2取整法”。转换方法为：连续用2乘，依次求得各整数位（0或1）K_{-1}、K_{-2}、…、K_{-m}，直到乘积的小数部分为0。在小数转换过程中，出现F_i恒不为0时，可按精度要求确定二进制小数的位数。

1.进位计数制和数制之间的转换

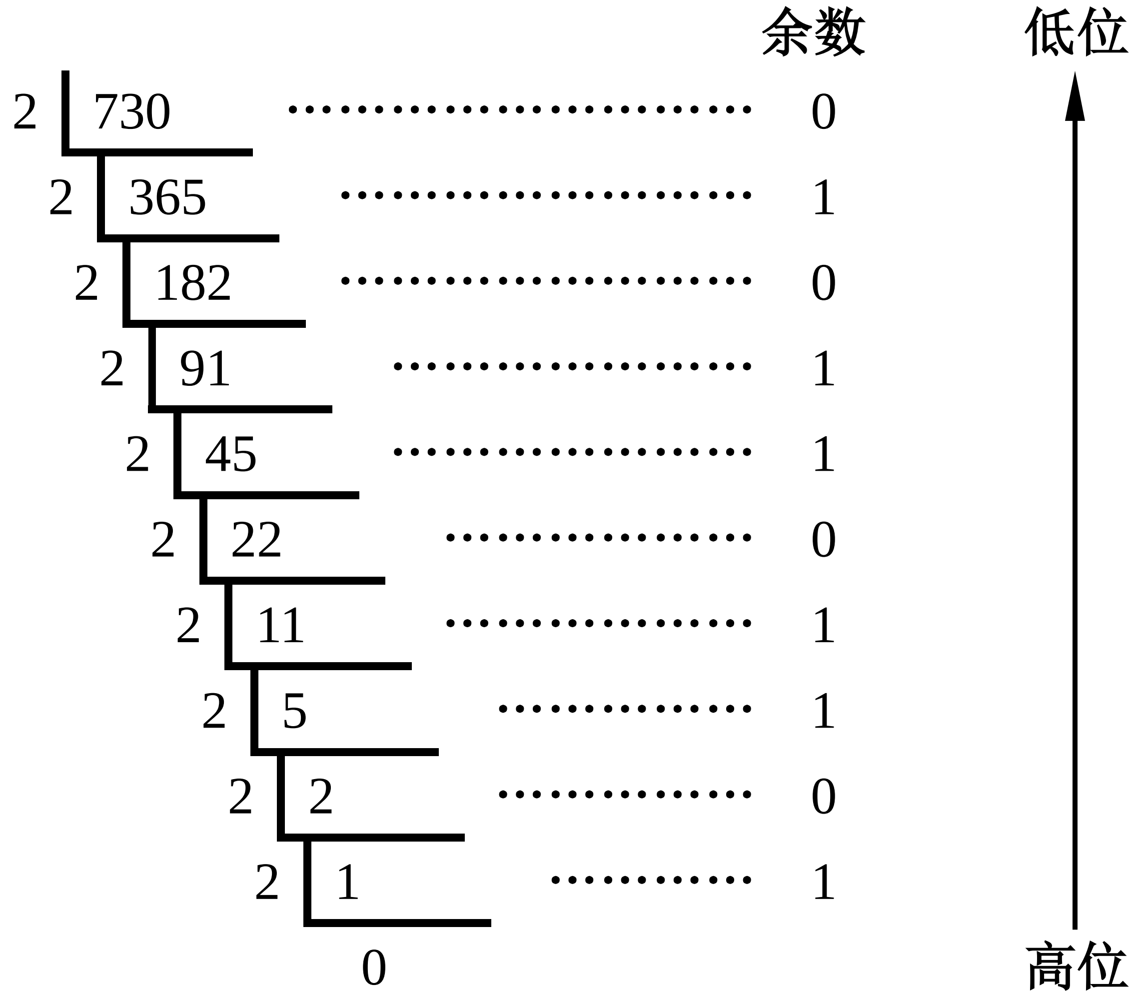
【例】将十进制数730.8125转换成二进制数、八进制数。

【解】①整数部分的转换：“除基取余，先低后高”

除数	被除数	余数	
8	730	2	低位
8	91	3	
8	11	3	
8	1	1	
	0	0	高位

因此，$(730)_{10}=(1332)_8$

1.进位计数制和数制之间的转换



因此，$(730)_{10}=(1011011010)_2$

1.进位计数制和数制之间的转换

【例】将十进制数730.8125转换成二进制数、八进制数。

【解】② 小数部分的转换：“乘基取整，先高后低”

将十进制数$(0.8125)_{10}$转换为八进制数：

$0.8125\times 8=6.5$　　整数部分＝6 （高位）

$0.5\times 8=4.0$　　整数部分＝4 （低位）

因此，$(0.8125)_{10}=(0.64)_8$

$\therefore (730.8125)_{10}=(1332.64)_8$

【例】将十进制数730.8125转换成二进制数、八进制数。

【解】 ② 小数部分的转换："乘基取整，先高后低"

将十进制数$(0.8125)_{10}$转换为二进制数：

$0.8125 \times 2 = 1.625$	整数部分＝1 （高位）
$0.625 \times 2 = 1.25$	整数部分＝1
$0.25 \times 2 = 0.5$	整数部分＝0
$0.5 \times 2 = 1.0$	整数部分＝1 （低位）

因此，$(0.8125)_{10} = (0.1101)_2$

$\therefore (730.8125)_{10} = (1011011010.1101)_2$

十进制数转换为八进制数、十六进制数

- 将十进制数转换为八进制数、十六进制数时，使用的方法与十进制数转换成二进制数的方法基本相同，只是求整数部分时是用商除以8或16，取其余数；小数部分改用乘以8或16，取其整数即可。

1.进位计数制和数制之间的转换

二进制数与八进制、十六进制数间的转换

- 二进制转化成八(十六)进制
  - 整数部分：从右向左按三(四)位分组，不足补零
  - 小数部分：从左向右按三(四)位分组，不足补零

【例】

$(\underline{001}\ \underline{011}\ \underline{010}\ \underline{110}.\underline{101}\ \underline{011}\ \underline{100})_2 = (1326.534.)_8$

1 3 2 6 5 3 4

$(\underline{0101}\ \underline{1101}.\underline{0101}\ \underline{1010})_2 = (5D.5A)_{16}$

5 D 5 A

八进制、十六进制数与二进制数间的转换

- 八(十六)进制转化成二进制
  - 一位八进制数对应三位二进制数
  - 一位十六进制数对应四位二进制数
- 【例】

$(247.63)_8 = (\underline{010}\ \underline{100}\ \underline{111}.\underline{110}\ \underline{011})_2$

$(F5A.6B)_{16} = (\underline{1111}\ \underline{0101}\ \underline{1010}\ \underline{0110}.\underline{0110}\ \underline{1011})_2$

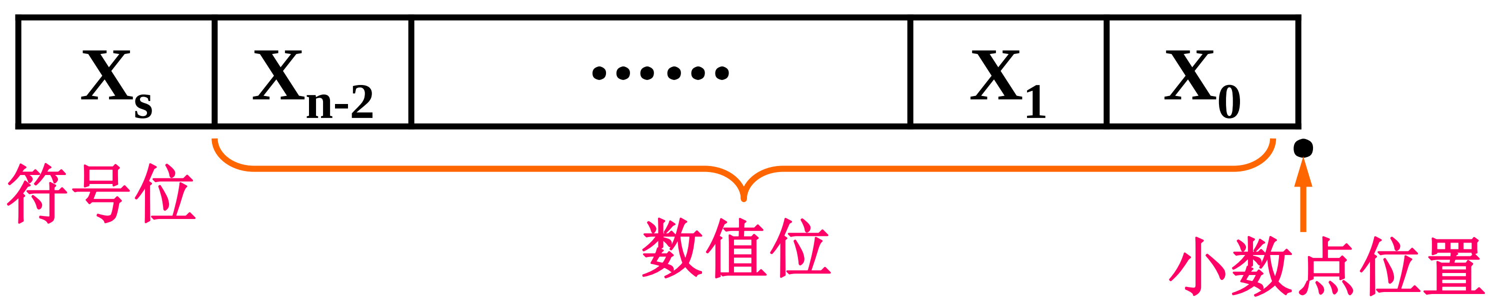
2.定点数

- 无符号数与有符号数
  - 无符号数表示正数，在机器数中没有符号位；
  - 有符号数用数字化的“0”表示“正”，“1”表示“负”，符号放在有效数字的最前面。
- 定点数与浮点数
  - 约定小数点位置固定不变的数称为定点数，计算机中定点数只定义为纯整数或纯小数形式；
  - 浮点数采用类似科学计数法的数值表示方法。

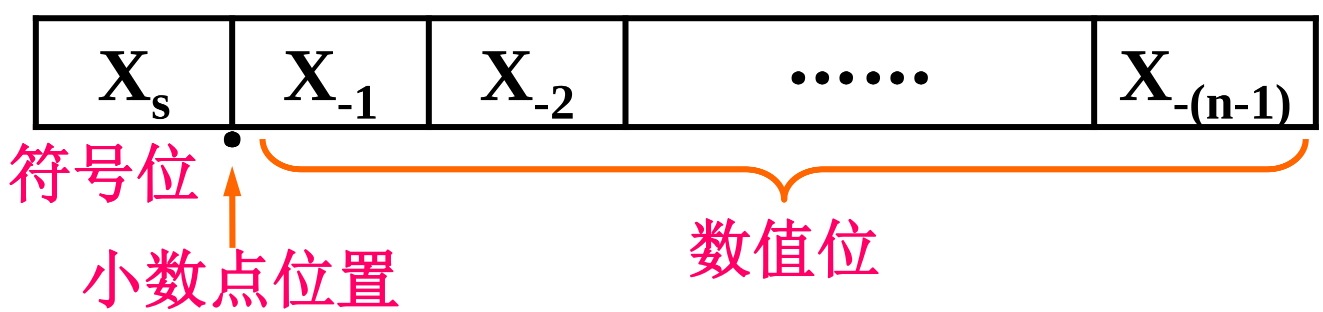
2. 定点数

- 定点数是指小数点位置固定不变的数。小数点的位置通常只有两种约定，相应地有两种类型的定点数，即定点整数和定点小数。

带符号定点整数格式：



带符号定点小数格式：



2. 定点数

【例】

X=+1010110. → 纯整数：X = 01010110.

正数，符号位取0

Y= - 1101001. → 纯整数：Y = 11101001. （原码）

负数，符号位取1

X=+0.11011 → 纯小数：X = 0.11011

符号位取0

Y=-0.10101 → 纯小数：X = 1.10101 （原码）

符号位取1

2. 定点数

- 浮点数 N 由三部分来决定其数值大小：阶码 E、尾数 M 和阶码的底 R：

$$N=R^{E}\cdot M$$

在计算机中，阶码的底 R 隐含表示，一般约定为 2、 8 或 16。浮点数实际上由定点数组成：它的阶码是定点整数，它的尾数是定点小数。 E和M可正可负。

阶码	尾数
$E_S E_1 E_2 \cdots\cdots E_m$	$M_S M_1 M_2 \cdots\cdots M_n$

（c）浮点数格式

原码表示

- **原码（True form）也称“符号—数值”表示法、带符号的绝对值表示。最高位是符号位，0表示正数，1表示负数，数值位即真值的绝对值。**
- 定点整数的原码定义如下：

$$[X]_{原} = \begin{cases} X & 0 \le X \le 2^{n-1}-1 \\ 2^{n-1}-X = 2^{n-1}+|X| & -(2^{n-1}-1) \le X \le 0 \end{cases}$$

- 定点小数的原码定义如下：

$$[X]_{原} = \begin{cases} X & 0 \le X \le 1-2^{-(n-1)} \\ 1-X = 1+|X| & -(1-2^{-(n-1)}) \le X \le 0 \end{cases}$$

原码表示

由定义可以求出原码的表示形式，

- 对于定点整数：

若 $X= + X_1X_2 \ldots X_{n-1}$， 则$[X]_{原}= 0\ X_1X_2 \ldots X_{n-1}$；

若 $X= - X_1X_2 \ldots X_{n-1}$， 则$[X]_{原}= 1\ X_1X_2 \ldots X_{n-1}$。

- 对于定点小数：

若 $X= + 0 . X_1X_2 \ldots X_{n-1}$， 则$[X]_{原}= 0 . X_1X_2 \ldots X_{n-1}$；

若 $X= - 0 . X_1X_2 \ldots X_{n-1}$， 则$[X]_{原}= 1 . X_1X_2 \ldots X_{n-1}$。

2. 定点数

【例】若机器字长n＝8，则

$[+35]_{原} = (00100011)_2$

$[-35]_{原} = 2^7-(-35)$

$=(10000000)_2+(00100011)_2$

$=(10100011)_2$

$[+0.8125]_{原} = (0.1101000)_2$

$[-0.8125]_{原} = 1-(-0.8125)$

$=(1.0000000)_2+(0.1101000)_2$

$=(1.1101000)_2$

2. 定点数

- 原码的性质：
  - “0”不唯一。 [+0]原＝0 0000000，[-0]原＝1 0000000 (n=8)
  - 表示范围（机器字长为n）：
    - 定点小数：$-(1-2^{-(n-1)}) \sim +(1-2^{-(n-1)})$
    - 定点整数：$-(2^{n-1}-1) \sim +(2^{n-1}-1)$
  - 负数的原码大于正数的原码。
- 真值与原码之间的转换：
  - 真值转原码，符号位+/-转0/1，数值位照写；
  - 原码转真值，符号位0/1转+/-，数值位照写。

原码的优缺点：

- 优点：
  - 简单、直观，机器数和真值间的相互转换很容易；
  - 实现乘、除运算的规则简单。
- 缺点：
  - 实现加、减运算的规则较复杂；
  - “0”的表示不唯一。**[+0]原＝0 0000000，[-0]原＝1 0000000 (n=8)**

多选题 1分

已知8位编码$[X]_{原}=10110101$，其可能的正确二进制真值是

A. -110101 √

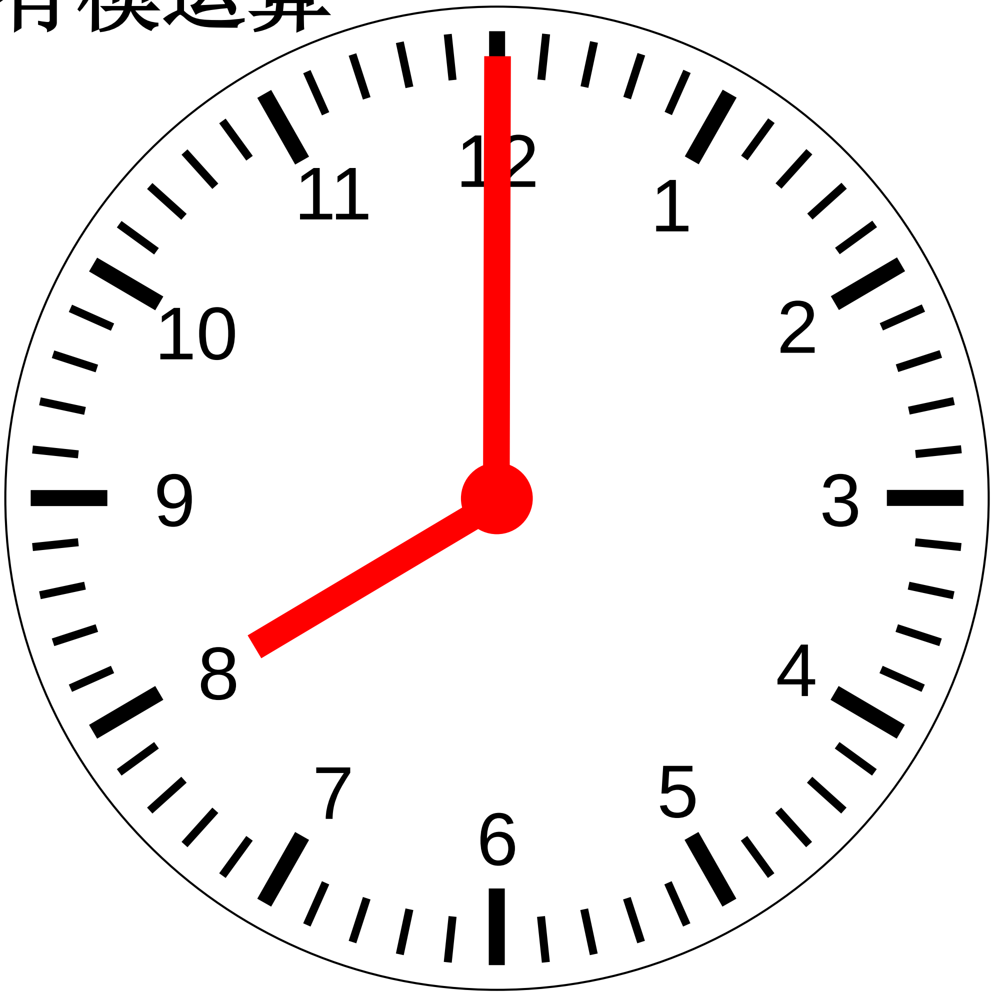
B. -0110101 √

C. -0.110101

D. -0.0110101 √

2. 定点数

- 无模运算
- 有模运算



8-3=5

8+9=17 mod 12 =5

补码表示

- 有模运算：在一定数值范围内进行的运算。在一个有模运算系统中，一个数与它除以模后得到的余数是等价的。
  - 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
  - 假定M为模，若数a、b满足$a+b=M$，则称a、b互为补数。
  - 在有模运算中，减去一个数等于加上这个数对模的补数。

2. 定点数

- 定点**整数**的补码定义：

$$[X]_{补}=\begin{cases} X=[X]_{原} & 0\le X\le 2^{n-1}\text{-}1 \\ 2^n+X=2^n-|X| & -2^{n-1}\le X<0 \end{cases} \qquad \text{MOD } 2^n$$

$$\mathbf{[X]_{补}=2^n+X\ (mod\ 2^n)}$$

- 定点**小数**的补码定义：

$$[X]_{补}=\begin{cases} X=[X]_{原} & 0\le X\le 1\text{-}2^{-(n-1)} \\ 2+X=2-|X| & -1\le X<0 \end{cases} \qquad \text{MOD } 2$$

$$\mathbf{[X]_{补}=2+X\ (mod\ 2)}$$

2. 定点数

由定义可以求出补码的表示形式，

- 对于定点整数：

若 $X=+X_1X_2 \ldots X_{n-1}$， 则$[X]_{补}=0, X_1X_2 \ldots X_{n-1}$；

若 $X=-X_1X_2 \ldots X_{n-1}$， 则$[X]_{补}=1\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}+1$

- 对于定点小数：

若 $X=+0.X_1X_2 \ldots X_{n-1}$， 则$[X]_{补}=0, X_1X_2 \ldots X_{n-1}$；

若 $X=-0.X_1X_2 \ldots X_{n-1}$， 则$[X]_{补}=1\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}+0.00\ldots1$

2. 定点数

"按位取反，末位加一"：

数值部分自低位向高位搜索，第一个1及其以右的各位0保持不变，以左的各高位按位取反。

$$
\begin{array}{rll}
X\cdots XX\ \ 10\cdots00 & —— & \text{原始数值} \\
\downarrow\text{按位取反} & & \\
\overline{X}\cdots\overline{X}\ \overline{X}\ \ 01\cdots11 & —— & \text{原始数值按位取反} \\
+\quad 1 & —— & \text{末位加一} \\
\hline
\boxed{\overline{X}\cdots\overline{X}\ \overline{X}}\ \boxed{10\cdots00} & —— & \text{原始数值按位取反、末位加一之后的结果} \\
\text{按位取反}\uparrow \qquad \uparrow\text{不变} & & \\
\boxed{X\cdots XX}\ \boxed{10\cdots00} & —— & \text{原始数值}
\end{array}
$$

2. 定点数

【例】若机器字长$n=8$，

求定点整数 0、－1、－128 的补码表示。

【解】 $[0]_{补}=[0]_{原}=0=(0000\ 0000)_2$

$[-1]_{补}=2^8+(-1)$

$=(1\ 0000\ 0000)_2-(0000\ 0001)_2$

$=(1111\ 1111)_2$

$[-128]_{补}=2^8+(-128)$

$=(1\ 0000\ 0000)_2-(1000\ 0000)_2$

$=(1000\ 0000)_2$

2. 定点数

【例】若机器字长n＝8，则

$[+35]_{补}$＝（0010 0011）$_2$

$[-35]_{补}$＝（1101 1101）$_2$

$[+0.8125]_{补}$＝（0.1101000）$_2$

$[-0.8125]_{补}$＝（1.0011000）$_2$

2^{0}	1
2^{1}	2
2^{2}	4
2^{3}	8
2^{4}	16
2^{5}	32
2^{6}	64
2^{7}	128
2^{8}	256
2^{9}	512
2^{10}	1024

2^{-1}	0.5
2^{-2}	0.25
2^{-3}	0.125
2^{-4}	0.0625
2^{-5}	0.03125

补码的性质（以定点小数为例）

1）补码的符号位

用补码表示的数，若其最高位为“**0**”，则此数为正；若其最高位为“**1**”，则此数为负。

2）补码中0的表示：

由补码定义，$[0]_{补}=0$，

∴ **0**的补码是唯一的。

3）补码的表示范围：

假设机器字长为n，用补码表示的定点小数，其表示范围为：

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

用补码表示的定点整数，其表示范围为：

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

➢ 比原码多表示一个值：-2^{n-1} 和-1，原因是真值“0”只占用一个编码 0 0….0，另一个在原码中表示“-0”的编码 1 0….0 则用来表示值 -2^{n-1}和-1。

4）负数的补码值大于正数的补码值

5）补码与真值、原码之间的相互转换

当X≥0时，$X=[X]_{原}=[X]_{补}$

$$\text{真值 } X \xleftrightarrow[\text{数值位不变}]{+\longleftrightarrow 0} [X]_{补}=[X]_{原}$$

当X＜0时，假设机器字长为n，由定义得：

$$[X]_{补}=2+X$$

$$=\overbrace{1.11\cdots\cdots1}^{n个1}+X+\overbrace{0.00\cdots\cdots0}^{n-1个0}\ 1$$

$$=\underbrace{\overbrace{1.11\cdots\cdots1}^{n个1}-|X|}_{|X|按位取反}+\underbrace{\overbrace{0.00\cdots\cdots0}^{n-1个0}\ 1}_{末位加一}$$

2. 定点数

5）补码与真值、原码之间的相互转换

当X＜0时，假设机器字长为n，

由定义 $[X]_{补}=2+X$ 得：$-X=2-[X]_{补}$，

又因为：$-X=|X|$，因此

$$|X|=-X=2-[X]_{补}$$

$$=\underbrace{\overbrace{1.11\cdots\cdots1}^{n个1}-[X]_{补}}_{[X]_{补}按位取反}+\underbrace{\overbrace{0.00\cdots\cdots0}^{n-1个0}\ 1}_{末位加一}$$

5）补码与真值、原码之间的相互转换

结论：当真值X＜0时，

$$\text{真值 } X \xleftrightarrow[\text{数值按位取反，末位加1}]{- \longleftrightarrow 1} [X]_{\text{补}}$$

$$[X]_{\text{原}} \xleftrightarrow[\text{数值按位取反，末位加1}]{\text{符号位1不变}} [X]_{\text{补}}$$

2. 定点数

【例】假设机器字长n＝8，已知

X_1＝＋0.1011001，

X_2＝0，

X_3＝－0.1101100，

求其原码及补码。

解： $[X_1]_{补}= [X_1]_{原}=X_1=(0.1011001)_2$

$[X_2]_{补}=(0.000\ 0000)_2$

$[X_2]_{原}=(0.000\ 0000)_2=(1.000\ 0000)_2$

$[X_3]_{原}=(1.1101100)_2$

$[X_3]_{补}=(1.0010100)_2$

2. 定点数

【例】已知

$[X_1]_{补}=0.101\ 0110$，

$[X_2]_{补}=1.110\ 0101$，

$[X_3]_{补}=1.000\ 0000$，求其真值及原码。

解：$X_1= 0.101\ 0110$

$[X_1]_{原}= 0.101\ 0110$

$X_2= -0.001\ 1011$

$[X_2]_{原}= 1.001\ 1011$

$X_3= -1$

$[X_3]_{原}$不存在

2. 定点数

6）补码的扩展(只用结论，推导自学)

实际应用中，有时需要扩充补码的位数，

- 定点小数：在其低位填充适当位数的“0”
- 定点整数：符号位扩展

【例】已知定点小数X_1、X_2用8位表示的补码如下：

$[X_1]_补=0.1010110$，$[X_2]_补=1.1100101$。

现将$[X_1]_补$、$[X_2]_补$扩展为16位表示。

【解】 $[X_1]_补=0.1010110\ 00000000$

$[X_2]_补=1.1100101\ 00000000$

结论1：要将n位纯小数补码变为$2n$位，只需在末尾添加n个“0”即可。

6）补码的扩展

要将n位定点整数补码用$2n$位表示，如何处理？

即：如何将MOD 2^n的补码变成MOD 2^{2n}的补码。

推导过程：

用MOD $2^n[X]_补$表示X以2^n为模的补码，

用MOD $2^{2n}[X]_补$表示X以2^{2n}为模的补码。

$$\text{MOD}\ \ 2^n[X]_补=\begin{cases} X & 0\le X<2^{n-1} \\ 2^n+X & -2^{n-1}\le X<0 \end{cases} \qquad \text{MOD}\ \ 2^n$$

$$\text{MOD}\ \ 2^{2n}[X]_补=\begin{cases} X & 0\le X<2^{2n-1} \\ 2^{2n}+X & -2^{2n-1}\le X<0 \end{cases} \qquad \text{MOD}\ \ 2^{2n}$$

2. 定点数

6）补码的扩展

要将*n*位定点整数补码用2*n*位表示，如何处理？

当X≥0时，

$$
\begin{aligned}
\text{MOD}\ \ 2^{2n}[X]_{补} &= X = \text{MOD}\ \ 2^{n}[X]_{补} \\
&= \underbrace{00\cdots\cdots0}_{2n个0} + \text{MOD}\ \ 2^{n}[X]_{补} \\
&= \underbrace{00\cdots\cdots0}_{n个0}\ \ \text{MOD}\ \ 2^{n}[X]_{补}
\end{aligned}
$$

2. 定点数

6）补码的扩展

要将n位定点整数补码用$2n$位表示，如何处理？

当X＜0时，

$$
\begin{aligned}
\text{MOD}\ \ 2^{2n}[X]_{补} &= 2^{2n} + X = 2^{2n} + (-2^{n} + \text{MOD}\ \ 2^{n}[X]_{补}) \\
&= \underbrace{100\cdots\cdots0}_{2n个0} - \underbrace{100\cdots\cdots0}_{n个0} + \text{MOD}\ \ 2^{n}[X]_{补} \\
&= \underbrace{111\cdots\cdots1}_{n个1}\ \underbrace{00\cdots\cdots0}_{n个0} + \text{MOD}\ \ 2^{n}[X]_{补} \\
&= \underbrace{111\cdots\cdots1}_{n个1}\ \text{MOD}\ \ 2^{n}[X]_{补}
\end{aligned}
$$

6）补码的扩展

要将*n*位定点整数补码用2*n*位表示，如何处理？

综合X≥0、X＜0时的情况，

$$\mathrm{MOD}\ \ 2^{2n}[X]_{补} = \underbrace{X_s X_s \cdots\cdots X_s}_{n个}\mathrm{MOD}\ \ 2^{n}[X]_{补}$$

其中，X_S为 MOD $2^n[X]_{补}$的符号位。

结论2：将整数补码的模扩大2^n倍，只需将$[X]_{补}$的符号位向左复制*n*位即可。

6）补码的扩展

要将*n*位定点整数补码用2*n*位表示，如何处理？

【例】已知定点整数X_1、X_2用8位表示的补码如下：

$[X_1]_补$＝01010110，$[X_2]_补$＝11100101。

将$[X_1]_补$、$[X_2]_补$扩展为16位表示。

【解】16位表示的$[X_1]_补$、$[X_2]_补$如下：

$[X_1]_补$＝00000000 01010110

$[X_2]_补$＝11111111 11100101

7）补码的算数右移（除2运算）

某一个数的补码经算数右移1位后，其最低有效位被移出，最高位（符号位）如何处理？

以定点小数为例

已知 $[X]_补$，求$\left[\frac{1}{2}X\right]_补$。

当$X\geq0$时， $[X]_补=X$，

$$[\frac{1}{2}X]_补=\frac{1}{2}X=\frac{1}{2}[X]_补=0\cdot2^0+\frac{1}{2}[X]_补$$

当$X<0$时， $[X]_补=2+X$， 则 $X=-2+[X]_补$

$$[\frac{1}{2}X]_补=2+\frac{1}{2}X=2+\frac{1}{2}(-2+[X]_补)=1+\frac{1}{2}[X]_补=1\cdot2^0+\frac{1}{2}[X]_补$$

7）补码的算数右移（除2运算）

某一个数的补码经算数右移1位后，其最低有效位被移出，最高位（符号位）如何处理？

已知 $[X]_补$，求$\left[\frac{1}{2}X\right]_补$。

两式合并： $$[\frac{1}{2}X]_补 = X_S \cdot 2^0 + \frac{1}{2}[X]_补$$

$$[X]_补 \xrightarrow[\text{按位右移一位}]{\text{符号位不变}} \left[\frac{1}{2}X\right]_补$$

7）补码的算数右移（除2运算）

【例】已知

$$[X_1]_补=0.110\ 1010,$$

$$[X_2]_补=1.010\ 0110,$$

$$[X_3]_补=1.000\ 0000,$$

求：

$$\left[\frac{1}{2}X_1\right]_补 = 0.011\ 0101$$

$$\left[\frac{1}{2}X_2\right]_补 = 1.101\ 0011$$

$$\left[\frac{1}{2}X_3\right]_补 = 1.100\ 0000$$

8）补码的算数左移（乘2运算）

算术左移可能产生溢出。因此，需增加一位二进制位，或者说将模扩大一倍。

由$[X]_补$求$[2X]_补$:

$$[X]_补 = 2 + X \quad \mathrm{MOD}\ 2$$

$$[2X]_补 = 4 + 2X \qquad\qquad \mathrm{MOD}\ 4$$

$$[2X]_补 = 4 + 2X = 4 + 2(-2 + [X]_补) = 2[X]_补$$

结论：已知$[X]_补$求$[2X]_补$只需将$[X]_补$的各位左移一位，末位补0。

8）补码的算数左移（乘2运算）

【例】假设机器字长$n=8$，已知$[X_1]_补=0.0110100$，$[X_2]_补=1.0010110$，求$[2X_1]_补$、$[2X_2]_补$。

【解】

$[2X_1]_补=00.1101000=0.1101000$　　未溢出

$[2X_2]_补=10.0101100=0.0101100$　　溢出

8）补码的算数左移（乘2运算）

- 变形补码：采用双符号位。左符是真正的符号位，右符用来判别“溢出”。
- 当使用变形补码（双符号位）进行运算时，
  - 若运算结果的两个符号位相同，则不发生溢出；
  - 若运算结果的两个符号位相异，则结果溢出。此时，最高位为符号；次高位为溢出的数值而非符号。

反码表示

反码常用来作为由原码求补码或者由补码求原码的中间过渡。

反码又称为**1的补码**（**one’s complement**），下面我们给出定点小数和定点整数的反码定义。

2. 定点数

- 设机器字长为**n**位，定点整数的反码定义：

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

同补码类似，反码也有模除概念

$[X]_{补}$ =2ⁿ-1+X (mod 2ⁿ-1)

- 设机器字长为**n**位，定点小数的反码定义：

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

n位定点小数的反码机器数的模为 1，因此，反码又称为“ 1 的补码”

$[X]_{补}=2\text{-}2^{-(n-1)}+X \quad (\mathrm{mod}\ 2\text{-}2^{-(n-1)})$

2. 定点数

由定义可以求出反码的表示形式，

- 对于定点整数：

若 $X=+X_1X_2\ldots X_{n-1}$，则$[X]_反=0,X_1X_2\ldots X_{n-1}$；

若 $X=-X_1X_2\ldots X_{n-1}$，则$[X]_反=1\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}$

- 对于定点小数：

若 $X=+0.X_1X_2\ldots X_{n-1}$，则$[X]_反=0,X_1X_2\ldots X_{n-1}$；

若 $X=-0.X_1X_2\ldots X_{n-1}$，则$[X]_反=1\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}$

-

2. 定点数

【例】若机器字长n＝8，求反码

$$35 = +00100011$$
$$-35 = -00100011$$

$[+35]_{反} = (00100011)_2$

$[-35]_{反} = (2^8-1)+(-35)$

$= (11111111)_2-(00100011)_2$

$= (11011100)_2$

$$0.8125 = +0.1101000$$
$$-0.8125 = -0.1101000$$

$[+0.8125]_{反} = (0.1101000)_2$

$[-0.8125]_{反} = (2-2^{-7})+(-0.8125)$

$= (1.1111111)_2-(0.1101000)_2$

$= (1.0010111)_2$

2. 定点数

反码的性质：

- 最高位为符号位，0表示正，1表示负。
- 两种0的表示，使得反码与真值不能一一对应：

  $[+0]_{反}=00\ldots0$，$[-0]_{反}=11\ldots1$
- 表示范围（假设机器字长为n）：
  - 定点小数：$-(1-2^{-(n-1)})\sim+(1-2^{-(n-1)})$
  - 定点整数：$-(2^{n-1}-1)\sim+(2^{n-1}-1)$
- 负数的反码大于正数的反码。

2. 定点数

反码的性质：

- 反码与原码及真值之间的转换：
  - 当$X \geq 0$时，由定义：$[X]_{反} = [X]_{原} = X$
  - 当$X < 0$时，

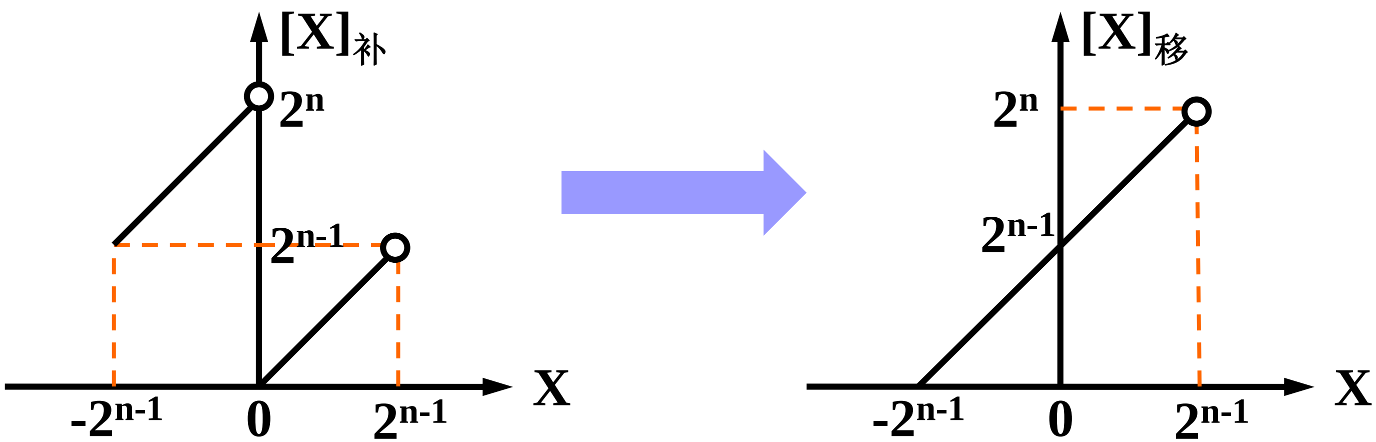
$$\text{真值}X \xleftrightarrow[\text{数值位不变}]{- \longleftrightarrow 1} [X]_{原} \xleftrightarrow[\text{数值位按位取反}]{\text{符号位不变}} [X]_{反}$$

- 负数的反码与补码的关系（以小数为例）：
  - $[X]_{反} = 2 - 2^{-(n-1)} + X$
  - $[X]_{补} = 2 + X$

  可见，负数的反码末位加1即为其补码（对整数同样成立）。

移码表示

- 以整数为例，在数值上，负数的补码比正数的大，而给每个数都加一个偏移量2^{n-1}后，补码与真值的关系发生了如下变化：



- 即把负端的数据全部移到了正端，因此被称为“移码”，又叫“增码”。

2. 定点数

- 计算机中常用移码来表示浮点数的阶码 →即指数。阶码为整数，故我们只讨论定点整数的移码表示。
- 机器字长为n位，则移码定义：

$$[X]_{移}=2^{n-1}+X,\quad -2^{n-1}\leq X\leq 2^{n-1}-1$$

- 由于$[X]_{补}=2^n+X \pmod{2^n}$

$$[X]_{移}=2^{n-1}+[X]_{补} \pmod{2^n},\quad -2^{n-1}\leq X\leq 2^{n-1}-1$$

- 因此，可以得出结论：求移码的表示形式只需将补码的符号位取反即可。

若 $X=+X_1X_2\ldots X_{n-1}$，则$[X]_{移}=1,X_1X_2\ldots X_{n-1}$；

若 $X=-X_1X_2\ldots X_{n-1}$，则$[X]_{移}=0,\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}+1$

2. 定点数

【例】假设机器字长n＝8，已知

X_1＝＋101011，X_2＝－110010，

求其原码、补码和移码。

【解】

$[X_1]_{原}$＝ 00101011 $[X_2]_{原}$＝ 10110010

$[X_1]_{补}$＝ 00101011 $[X_2]_{补}$＝ 11001110

$[X_1]_{移}$＝ 10101011 $[X_2]_{移}$＝ 01001110

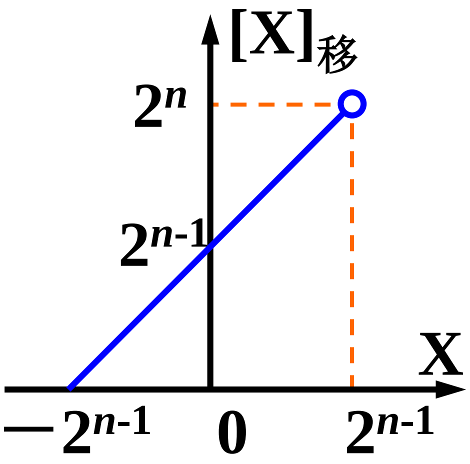
2. 定点数

移码的性质：

- 符号位："0"表示负，"1"表示正。
- "0"的移码表示是唯一的。

$$[+0]_{移}=[-0]_{移}=2^{n-1}+0=1\underbrace{00\cdots\cdots0}_{n-1个0}$$

- 表示范围：n位机器字长，$-2^{n-1}\le X\le 2^{n-1}-1$
- -2^{n-1} 的移码表示为00…0
- $[X]_{移}$与X呈线性正比关系。
  - 当且仅当X＞Y，$[X]_{移}$＞$[Y]_{移}$
  - 移码被广泛用来表示浮点数的阶



2. 定点数

定点整数X与$[X]_{原}$、$[X]_{补}$、$[X]_{移}$的对应关系（机器字长$n=8$）

真值X		X的原码表示$[X]_{原}$	X的补码表示$[X]_{补}$	X的移码表示$[X]_{移}$	
十进制表示	二进制表示			二进制表示	十进制表示
+127	+01111111	01111111	01111111	11111111	255
+126	+01111110	01111110	01111110	11111110	254
……	……	……	……	……	……
+2	+00000010	00000010	00000010	10000010	130
+1	+00000001	00000001	00000001	10000001	129
0	00000000	00000000 10000000	00000000	10000000	128
−1	−00000001	10000001	11111111	01111111	127
−2	−00000010	10000010	11111110	01111110	126
……	……	……	……	……	……
−126	−01111110	11111110	10000010	00000010	2
−127	−01111111	11111111	10000001	00000001	1
−128	−10000000	无法表示	10000000	00000000	0

定点机器数转换

（1）机器数转换为真值

- 四种定点机器数转换为真值的方法要点是：
  - 首先根据机器数的符号位确定真值的正负；
  - 然后对照机器数的定义和表示，反方向求出真值的绝对值。

（2）机器数之间的相互转换

- 原码、补码、反码和移码之间的相互转换，最简单的方法是
  - 先求出它们的真值
  - 然后再转换为另一种表示方法。

（1）原码：假设$[X]_{原} = X_0, X_1X_2\ldots X_{n-1}$

若 $X_0=0$，则 $X=+X_1X_2\ldots X_{n-1}$

若 $X_0=1$，则 $X=-X_1X_2\ldots X_{n-1}$

（2）补码：假设$[X]_{补} = X_0, X_1X_2\ldots X_{n-1}$

若 $X_0=0$，则 $X=+X_1X_2\ldots X_{n-1}$

若 $X_0=1$，则 $X=-(\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}+1)$

（3）反码：假设$[X]_{反} == X_0, X_1X_2\ldots X_{n-1}$

若 $X_0=0$，则 $X=+X_1X_2\ldots X_{n-1}$

若 $X_0=1$，则 $X=-\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}$

（4）移码：假设$[X]_{移} = X_0, X_1X_2\ldots X_{n-1}$

若 $X_0=1$，则 $X=+X_1X_2\ldots X_{n-1}$

若 $X_0=0$，则 $X=-(\overline{X_1}\,\overline{X_2}\ldots\overline{X_{n-1}}+1)$

1. 浮点数表示

- 在计算机中引入类似于十进制科学计数法的方法来表示实数，称为浮点数表示法，因其小数点位置不固定而得名。
- 浮点数的代码由两部分组成：阶码*E*（Exponent）和尾数*M*（Mantissa）。浮点数表示的数值为：

$$N = (-1)^{Ms} \times M \times B^{E}$$

其中*B*是阶码的底(Base)，即尾数*M*的基值(Radix)，一般为2(也有规定为4、8或16的)。它是隐含约定，不需要存储，因为它对所有数都是相同的。

3. 浮点数

- *E*是阶码，即指数，带符号定点整数，常用移码表示。
- *M*是尾数，带符号定点纯小数，常用补码或原码表示。M_S是尾数的符号位(即数符)，安排在最高位，表示该浮点数的正负。
- 尾数给出有效数字的位数，决定了浮点数的精度。阶码指明小数点在数据中的位置，决定了浮点数的范围。

M_S	E_S	$E_1E_2\cdots\cdots E_m$.	. $M_1M_2\cdots\cdots M_n$
数符	阶符	阶码数值	尾数数值

多选题

设浮点数字长16位，基底为2。其中阶码6位(含1位阶符)，用移码表示；尾数10位(含1位数符)，用补码表示。格式为

数符	阶码	尾数

试给出十进制数X=－0.019287的浮点数编码。

提示： X≈－0.000001001111_2（有限精度表示）

A. 1 000011 001001111

B. 1 011101 110110001

C. 1 011100 101100010

D. 1 011011 011000100

答案

设浮点数字长16位，基底为2。其中阶码6位(含1位阶符)，用移码表示；尾数10位(含1位数符)，用补码表示。格式为

数符	阶码	尾数

试给出十进制数X=-0.019287的浮点数编码。

A. 1 000011 001001111

B. 1 011101 110110001 √

C. 1 011100 101100010 √

D. 1 011011 011000100 √

答案解析

X= －0.019287

X≈ －0.000001001111_2 （有限精度表示）

因为尾数数值9位，

若X≈－0.000 001001111_2=－$0.001001111_2 \times 2^{-00011}$

则编码为 1 011101 110110001 B

若X≈-0.000001001111_2=$-0.010011110_2 \times 2^{-00100}$

则编码为 1 011100 101100010 C

若X≈-0.000001001111_2=$-0.100111100_2 \times 2^{-00101}$

则编码为 1 011011 011000100 D

2.规格化浮点数

- 为简化浮点数操作、充分利用尾数的二进制位数来表示更多的有效数字，通常采用浮点数规格化形式，即将尾数的绝对值限定在某个范围之内。
- 规格化浮点数是M有效数的最高有效位为非零的数，即$M_{n-1}=1$。
- 如果阶码的底为2，则规格化浮点数的尾数应满足条件：

$$1/2 \le | M | \le 1$$

2.规格化浮点数

- 计算机件对尾数的机器数形式的规格化判定方法：
  - M为原码表示时，当最高有效位（M_1）为1时， 浮点数为规格化，即尾数为x.1xxx…x形式
  - M位补码表示时，当符号位（Ms）与最高有效位（M_1）相异时，浮点数为规格化，即尾数为 0.1xxx…x形式或者为1.0xxx…x形式
  - 对于原码和补码表示的尾数，规格化浮点数要求的尾数表示范围是不同的，例如，尾数-0.5 用原码表示是规格化的，而对于补码表示是非规格化的。

3. 浮点数

- 对于非规格化浮点数，可以通过修改阶码和左右移尾数的方法来使其变为规格化浮点数，这个过程叫做规格化；
- 若尾数进行右移实现的规格化，则称为右规；
- 若尾数进行左移实现的规格化，则称为左规。

3. 浮点数

- 左规：
  - 对于原码表示的尾数，当最高有效位为 0 时，必须进行左规，尾数每左移 1 位，阶码减 1，直至尾数变为x.1xxx…x形式；
  - 对于补码表示的尾数，当符号位与最高有效位相同时，也必须左规，即尾数每左移 1 位，阶码减 1，直至把尾数中第一个不同于符号位的“ 0”或“ 1”移至最高有效位，变为为 0.1xxx…x形式或者为 1.0xxx…x形式

3. 浮点数

- 右规：
  - 只有当在浮点数的运算过程中，尾数发生了溢出，即超出了尾数所能表示的数据范围时，才需要进行右规处理，即尾数右移 1 位，阶码加 1，由于参加运算的浮点数必须是规格化的，因此，一般只需右规一次即可。
  - 在此必须明确指出的是，浮点数的溢出是由阶码是否溢出决定的，而非尾数，尾数发生溢出可以通过右规处理，只有阶码发生溢出，浮点数才被判定为溢出。

3. 浮点数

（接续上一个多选题）

$X=-0.019287$

$X\approx-0.\ 0000\ 0100\ 1111_2$ （有限精度表示）

因为尾数数值9位，

若$X\approx-0.\ 0000\ 0100\ 1111_2=-1.001111_2\times2^{-6}$ 非规格化

若$X\approx-0.0000\ 0100\ 1111_2=-0.010011110_2\times2^{-4}$ 非规格化

则编码为 1 011100 101100010

右规

左规

规格化浮点数

若$X\approx-0.000001001111_2=-0.100111100_2\times2^{-5}$

则编码为 1 011011 011000100

3. 浮点数

【例】将十进制数x=+13/128 写成二进制定点数和浮点数（尾数数值部分取7位，阶码数值部分取7位，阶码和数码各取1位，阶码采用移码，尾数用补码表示），并分别写出定点数和浮点数的编码。

解： $x=+13/128_{10} = 1101_2 \div 2^7=0.0001101_2=0.1101\times 2^{-11}$

定点数编码为：[x]原= [x]补=[x]反=0.0001101_2

$-3_{10}=-00000011_2$

$[-3]_{补}=11111101_2$

$[-3]_{移}=01111101_2$

规格化浮点数编码为

数符	阶符	阶码	尾数
0	0	1111101	1101000

3. 浮点数

规格化浮点数的特点：

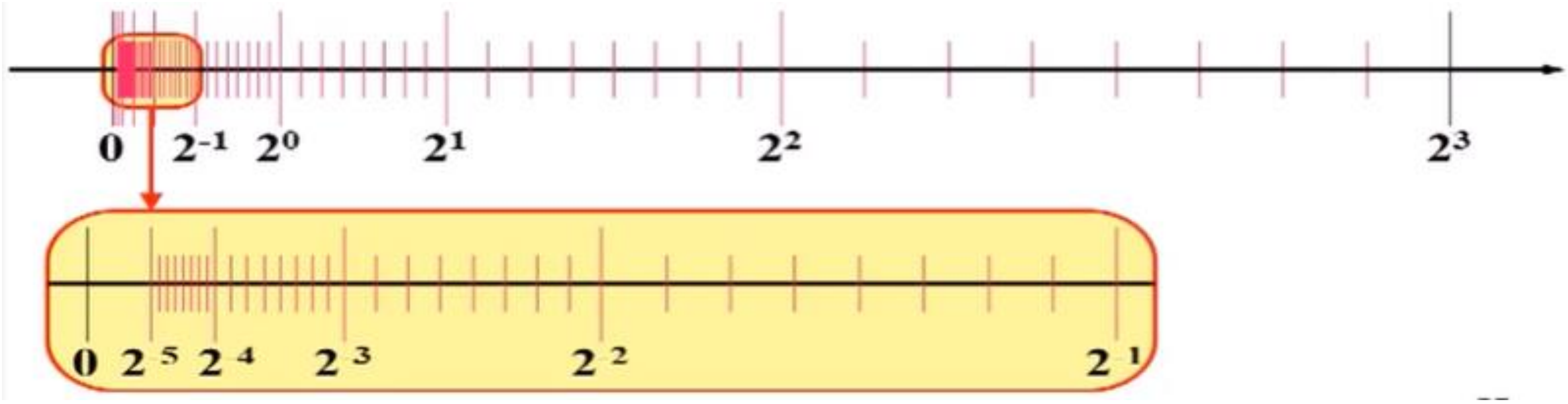
（1）机器零：

在浮点数的表示范围中，有两种情况被称为机器零：

- （1） 若浮点数的尾数为零，无论阶码为何值；
- （2） 或者当阶码的值遇到比它能表示的最小值还要小时（阶码负溢出），无论其尾数为何值。

（2）浮点数在数轴上的分布

- 假设浮点数的规格化尾数只有4位精度用原码表示，那么在任意两个2的整数次幂之间，有2^3=8个可以表示浮点数（小数点后第1位总是1）。正尾数的形式为0.1xxx
- 假设阶码用3位移码表示，则可能的阶码为-4，-3，-2，-1，0，1，2，3



3. 浮点数

- 用n位二进制数表示的浮点数，能表示不同值的最大数目仍是2^n，只是把这些数沿数轴正负两个方向在更大范围内散布开。
- 浮点表示的数不再像定点数那样沿数轴等距分布，而是越靠近原点，数越密集；越远离原点，数越稀疏。
- 浮点数溢出：数的大小超出的浮点数表示范围。原因：指数部分太大，以至于无法用有限的指数字段表示出来。
  - 数太大：$E>E_{max}$ →阶码上溢→浮点数溢出
  - 数太小：$E<E_{min}$ →阶码下溢→ N=0

（3）规格化浮点数表示的范围

- 浮点数的表示范围通常由四个点界定：最小（负）数、最大负数、最小正数、最大（正）数。其中，最大负数又称为负精度－δ，最小正数称为正精度＋δ，它们是浮点数所能表示的最精确的值，相当于浮点数的分辨率。
- 位于最大负数和最小正数之间的数据（除 0 外），浮点数无法表示，称为下溢。对于下溢的处理，计算机直接将其视为机器零。与定点机器数相同，当一个数据大于最大（正）数，或者小于最小（负）数时， 机器也无法表示， 称为上溢，上溢又称溢出。

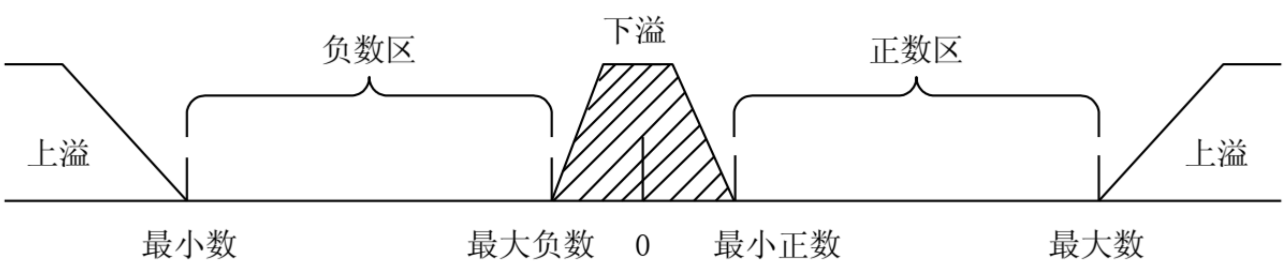


图3.3　浮点数的表示范围

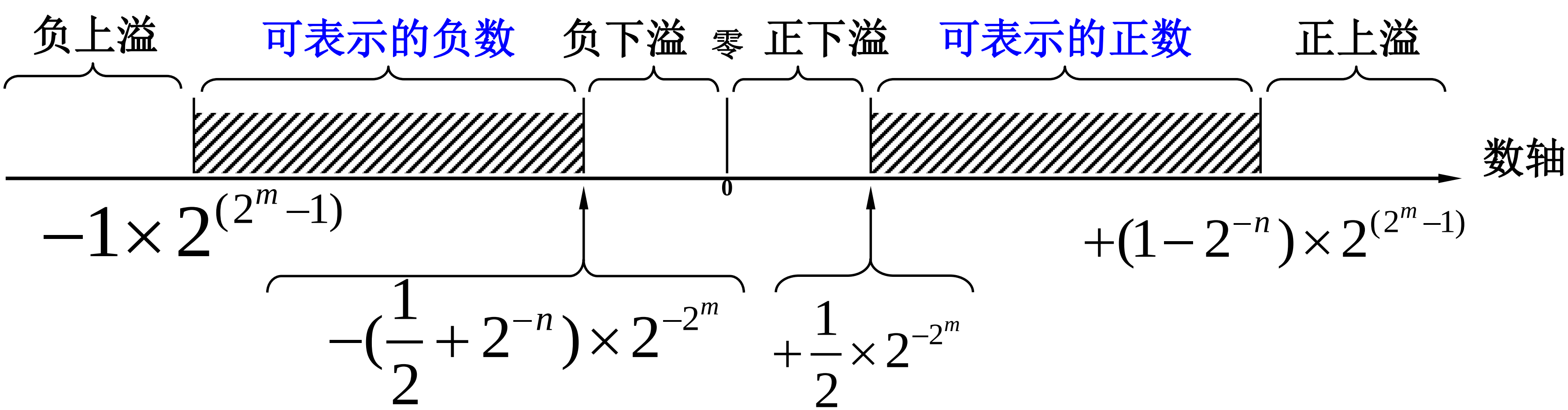
（3）浮点数表示的范围

- 设浮点数字长16位，基底为2。其中阶码6位(含1位阶符)，用移码表示；尾数10位(含1位数符)，用补码表示。
- （1）规格化表示范围

	真值	浮点数表示
最小数	$-1\times 2^{31} = -2^{31}$	1，11111　1.000000000
最大负数	$-(2^{-1}+2^{-9})\times 2^{-32}$	0，00000　1.011111111
最小正数	$2^{-1}\times 2^{-32} = 2^{-33}$	0，00000　0.100000000
最大数	$(1-2^{-9})\times 2^{31}$	1，11111　0.111111111

3. 浮点数

阶码和规格化尾数（补码编码）的数值范围			
阶码最小值	-2^m	阶码最大值	2^m-1
尾数最小负值	-1	尾数最大负值	$-(1/2+2^{-n})$
尾数最小正值	$+1/2$	尾数最大正值	$+(1-2^{-n})$



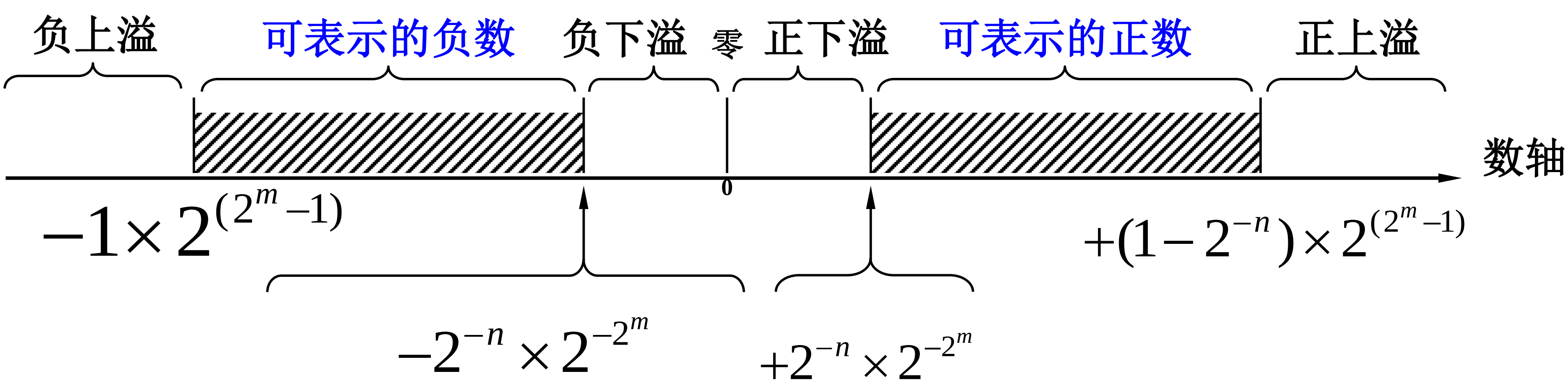
（3）浮点数表示的范围

- 设浮点数字长16位，基底为2。其中阶码6位(含1位阶符)，用移码表示；尾数10位(含1位数符)，用补码表示。
- （2）非规格化表示范围

	真值	浮点数表示
最小数	$-1\times 2^{31} = -2^{31}$	1，11111　1.000000000
最大负数	$-2^{-9}\times 2^{-32} = -2^{-41}$	0，00000　1.111111111
最小正数	$2^{-9}\times 2^{-32} = 2^{-41}$	0，00000　0.000000001
最大数	$(1-2^{-9})\times 2^{31}$	1，11111　0.111111111

3. 浮点数

阶码和非规格化尾数（补码编码）的数值范围			
阶码最小值	-2^m	阶码最大值	2^m-1
尾数最小负值	-1	尾数最大负值	-2^{-n}
尾数最小正值	$+2^{-n}$	尾数最大正值	$+(1-2^{-n})$



3. IEEE 754标准

- 浮点数的表示标准：IEEE 754，1985年由IEEE制定。
- 目的：
  - 便于程序从一类处理器移植到另一类处理器上。
  - 研制更为复杂的数值运算程序。
- 规定
  - 尾数用原码表示，小数点前隐含一个“1”。 S1.xxx
  - 基值隐含为2。
  - 阶码用移码表示，移码的偏移量有专门约定：n位移码的偏移值为$2^{n-1}-1$。正向比负向的多表示一个数。
  - 指数（阶码）的最大值、最小值作为特殊标记预留，用来标记某些异常事件或机器零。
  - 单精度(float)、双精度(double)：单精度扩展、双精度扩展。

IEEE 754标准

参数	格式			
	单精度	单精度扩展	双精度	双精度扩展
字宽	32	≥43	64	≥79
尾数有效位数p（包含隐含位）	24	≥32	53	≥64
指数位数	8	≥11	11	≥15
指数偏移值	+127	未指定	+1023	未指定
最大指数E_{max}	+127	≥+1023	+1023	≥+16383
最小指数E_{min}	−126	≤−1022	−1022	≤−16382

IEEE 754 格式参数

IEEE 754标准

- 真值表示：$(-1)^{s} \times (1.f) \times 2^{e-127/1023}$
- 单精度格式：由三个字段组成：23位的尾数*f*，8位移码表示的指数*e*（偏移值为127），以及1位符号*s*。

s	*e*[30:23]	*f*[22:0]
31	30 … 23	22 … 0

单精度格式位模式	IEEE浮点数的值
$0<e<255$	$(-1)^{s} \times 2^{e-127} \times 1.f$（规格化数）
$e=0$；$f \neq 0$（*f*中至少有一位不为零）	$(-1)^{s} \times 2^{-126} \times 0.f$（非规格化数）
$e=0$；$f=0$（*f*中所有位均为零）	$(-1)^{s} \times 0.0$（有符号的零）
$s=0$；$e=255$；$f=0$（*f*中所有位均为零）	+INF（正无穷大）
$s=1$；$e=255$；$f=0$（*f*中所有位均为零）	−INF（负无穷大）
$s=u$；$e=255$；$f \neq 0$（*f*中至少有一位不为零）	NaN（非数）

IEEE 754单精度格式位模式表示的值

IEEE 754标准

- IEEE 754单精度格式二进制位与其对应的浮点数真值举例

通用名称	浮点数的IEEE754编码（二进制）	十进制真值
＋0	0 00000000 000...00	＋0.0
－0	1 00000000 000...00	－0.0
1	0 01111111 000...00	1.0
2	0 10000000 000...00	2.0
最大规格化数	0 11111110 111...11	$3.40282347\times10^{+38}$
最小正规格化数	0 00000001 000...00	1.17549435×10^{-38}
最大非规格化数	0 00000000 111...11	1.17549421×10^{-38}
最小正非规格化数	0 00000000 000...01	1.40129846×10^{-45}
＋∞	0 11111111 000...00	正无穷
－∞	1 11111111 000...00	负无穷
非数	0 11111111 100...00	NaN（不惟一）

IEEE 754标准

- 真值表示：$(-1)^s \times (1.f) \times 2^{e-127/1023}$

- 双精度格式：由三个字段组成：52位尾数f，11位移码表示的指数e（偏移值为1023），以及1位符号s。

s	*e*[52:62]	*f*[51:32]	*f*[31:0]
63	62 52	51 32	31 0

IEEE 754标准

双精度格式位模式表示的数

双精度格式位模式	IEEE浮点数的值
$0 < e < 2047$	$(-1)^s \times 2^{e-1023} \times 1.f$ （规格化数）
$e=0$；$f \neq 0$ （f中至少有一位不为零）	$(-1)^s \times 2^{-1022} \times 0.f$ （非规格化数）
$e=0$；$f=0$ （f中所有位均为零）	$(-1)^s \times 0.0$ （有符号的零）
$s=0$；$e=2047$；$f=0$ （f中所有位均为零）	$+$INF （正无穷大）
$s=1$；$e=2047$；$f=0$ （f中所有位均为零）	$-$INF （负无穷大）
$s=u$；$e=2047$；$f \neq 0$ （f中至少有一位不为零）	NaN （非数）

IEEE 754标准

IEEE 754 双精度格式二进制位与其对应的浮点数真值举例

通用名称	浮点数的IEEE 754编码（十六进制）	十进制真值
＋0	00000000 00000000	＋0.0
－0	80000000 00000000	－0.0
1	3ff00000 00000000	1.0
2	40000000 00000000	2.0
最大规格化数	7fefffff ffffffff	$1.797693134862315 7\times 10^{+308}$
最小正规格化数	00100000 00000000	$2.2250738585072014\times 10^{-308}$
最大非规格化数	000fffff ffffffff	$2.2250738585072009\times 10^{-308}$
最小正非规格化数	00000000 00000001	$4.9406564584124654\times 10^{-324}$
＋∞	7ff00000 00000000	正无穷
－∞	fff00000 00000000	负无穷
非数	7ff80000 00000000	NaN（不惟一）

IEEE 754标准

符号	阶码	尾数	真值
0	**0**	**0**	**0**
1	**0**	**0**	**- 0**
0	**255/2047（全1）**	**0**	∝
1	**255/2047（全1）**	**0**	**-** ∝
0或1	**255/2047（全1）**	**≠0**	**NaN**
0或1	**255/2047（全1）**	**≠0**	**NaN**
0	**0<e<255/2047**	**f**	$(1.f) \times 2^{e-127/1023}$
1	**0<e<255/2047**	**f**	$-(1.f) \times 2^{e-127/1023}$
0	**0**	**f≠0**	$(0.f) \times 2^{e-126/1022}$
1	**0**	**f≠0**	$-(0.f) \times 2^{e-126/1022}$

主观题

（接续上一个多选题）

X=-0.019287

$X \approx -0.000001001111_2$ （有限精度表示）

请给出X的IEEE 754单精度编码。

答案解析

X=-0.019287

$X \approx -0.000001001111_2$ （有限精度表示）

首先将**X**表示为

$X \approx -0.000001001111_2 = -1.001111_2 \times 2^{-6} = -1.f \times 2^{e-127}$

IEEE 754单精度浮点数

s	e	f

$s=1$，$e=01111001$，$f=0011110\underbrace{0......0}_{17个}$

则编码为 **$1\ 01111001\ 0011110\underbrace{0......0}_{17个}{}_2 = BC9E0000H$**

IEEE 754标准

- 下溢的处理

【例】某正浮点数，用6位二进制数表示（省略符号位），尾数f用3位二进制数表示。加上小数点前的隐含位，尾数有4位有效数字。其他编码规则按照IEEE754标准执行。阶码e用3位二进制移码表示。

移码的偏移量为011，$e_{min}=-2$，$e_{max}=+3$，

$e_{min}-1=-3$，$e_{max}+1=+4$保留，用来表示0、非规格化数、±∞和NaN。

3. 浮点数

e_2	e_1	e_0	f_{-1}	f_{-2}	f_{-3}

规格化浮点数的值：$1.f_{-1}f_{-2}f_{-3} \times 2^{e_2e_1e_0-011}$

$x=1.011\times 2^{-1}$

$y=1.001\times 2^{-1}$

$x\neq y$；但是，

$x-y= 0.010\times 2^{-1}=1.000\times 2^{-3}=0$（下溢）

保证 $x=y \Leftrightarrow x-y=0$ 成立很重要

如 if (x≠y) then z=1/(x-y)

解决办法：渐进下溢。

3. 浮点数

- 渐进下溢：使用非规格化数，当指数是e_{min}时，尾数不必进行规格化。

  这样上例中1.000×2^{-2}不再是最小的浮点数，最小浮点数为0.001×2^{-2}，因此：$x - y= 0.010\times2^{-1} = 0.100\times2^{-2}$。运算结果可以用非规格化数表示而不必清零。

- 使用渐进下溢时，$x=y \Leftrightarrow x-y=0$ 始终成立。
- 非规格化数无需加前导1，为了区别于规格化数，IEEE754标准规定，当阶码$e=e_{min}-1$、尾数$f\neq0$（f中至少有一位不为零）时，浮点数为非规格化数，真值为$(-1)s\times2^{emin}\times0.f$

- 下溢的处理
  - 渐进下溢
  - 清零

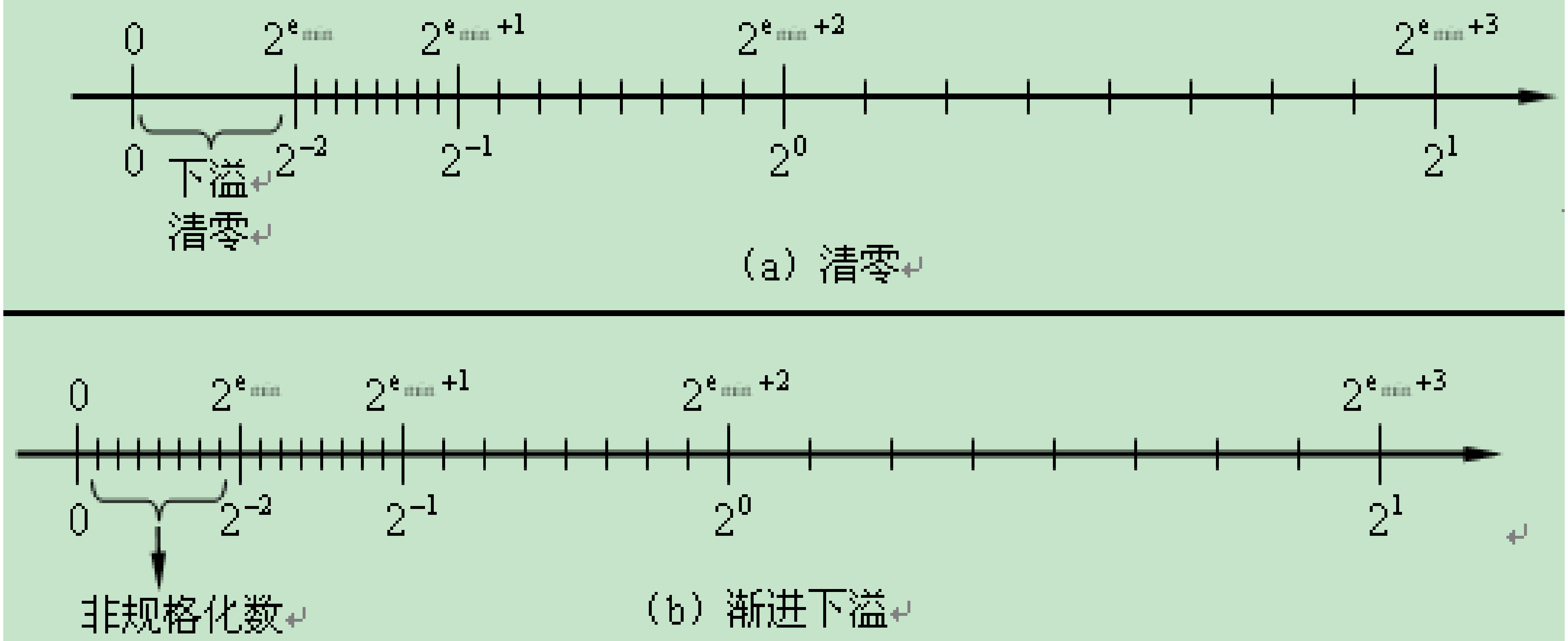


图 2-18 清零与渐进下溢的比较

IEEE 754标准

- 舍入模式 ——就近舍入
- 舍入到最接近的可表示值；
- 当有两个最接近的可表示值时，首选“偶数”值。

	就近舍入模式	四舍五入模式
Round(0.5)	0	1
Round(1.5)	2	2
Round(2.5)	2	3
Round(3.5)	4	4

IEEE 754标准

- 舍入模式 ——就近舍入
- 舍入到最接近的可表示值；
- 当有两个最接近的可表示值时，首选“偶数”值。

原始二进制尾数	就近舍入之后
1.0100110 10010	1.0100111
1.0100110 01111	1.0100110
1.0100110 10000	1.0100110
1.0100111 10000	1.0101000

IEEE 754标准

- 舍入模式 —— 朝0舍入
  - 朝数轴原点方向舍入，就是简单的截尾。
  - 使取值的绝对值比原值的绝对值小
  - 容易导致误差积累

例如 int(1.234)=1

int(-1.234)=-1

IEEE 754标准

- 舍入模式——朝+∞舍入
  - 对正数来说，只要多余位不全为0，则向最低有效位进1；
  - 对负数来说，则是简单的截尾。

例如 ceil(1.234)=2

ceil(-1.234)=-1;

IEEE 754标准

- 舍入模式——朝－∞舍入
  - 对正数来说，则是简单截尾；
  - 对负数来说，只要多余位不全为0，则向最低有效位进1。

例 floor(1.234)=1

floor(-1.234)=-2

指数移码的偏移

- 指数移码偏移值的规定
  - 8位移码，偏移值为128
  - 11位移码，偏移值为1024
- IEEE 754标准
  - 单精度格式，8位移码的偏移值取127，因此e_{max}=127，e_{min}=－126
  - 双精度格式，11位移码的偏移值取1023，因此有e_{max}=1023，e_{min}=－1022
  - 移码与补码不具有符号取反的特性。

3. 浮点数

- **IEEE 754标准**
  - 单精度格式，**8**位移码的偏移值取**127**，因此e_{max}=**127**，e_{min}=－**126**
  - 双精度格式，**11**位移码的偏移值取**1023**，因此有e_{max}=**1023**，e_{min}=－**1022**
  - 如此设计的原因：
    - 为了使|e_{min}|<e_{max}条件成立，从而保证绝对值最小的浮点数的倒数（$1/2^{emin}$）不上溢。
    - 虽然偏移值如此取值后，绝对值最大的浮点数的倒数（$1/2^{emax}$）会下溢，但是下溢通常比上溢更容易处理，比如运算结果仍可以用**IEEE754**规定的非规格化数来表示。

与IEEE 754标准相关的标准

IEEE 754-2008

- 原IEEE浮点数标准只定义了32位（binary32）和64（binary64）两种浮点数，即C程序员常说的单精度浮点数和双精度浮点数。
- 新的2008版本，还定义了16位（binary16），128（binary128）和k位（binary{*k*}，*k*≥128）的标准。
- 加入了十进制浮点数格式：decimal32、decimal64、decimal128、decimal{*k*}（*k*≥32）

3. 浮点数

【例】若浮点数x的IEEE754单精度存储格式为$(42150000)_{16}$，求浮点数x的十进制真值。

解：将16进制数展开后，可得二进制数格式如下：

0100 0010 0001 0101 0000 0000 0000 0000

S 阶码(8位) 尾数(23位)

指数=阶码e-$(127)_{10}$

$=(10000100)_2-(01111111)_2=(00000101)_2=(5)_{10}$

包含隐含位1的尾数=1.*f*

$=(1.001\ 0101\ 0000\ 0000\ 0000\ 0000)_2=(1.001\ 0101)_2$

因此，$x=(-1)^s\times(1.f)\times2^{e-127}$

$=+(1.001010)_2\times2^5=(100101.01)_2=(37.25)_{10}$

3. 浮点数

【例】将数$(22.78125)_{10}$转换成IEEE 754单精度浮点数的二进制存储格式。

解： $(22.78125)_{10}=(10110.11001)_2=(1.011011001)_2\times 2^4$

符号位s = 0

阶码e = 指数+ $(127)_{10}=(4+127)_{10}=(10000011)_2$

尾数f = $(011011001)_2$

∴32位单精度浮点数的二进制存储格式为：

$(0100\ 0001\ 1011\ 0110\ 0100\ 0000\ 0000\ \ 0000)_2=(41B64000)_{16}$

3. 浮点数

【例】将数$(22.78125)_{10}$转换成IEEE 754单精度浮点数的二进制存储格式。

解： $(22.78125)_{10}=(10110.11001)_2=(1.011011001)_2\times 2^4$

符号位s = 0

阶码e = 指数+ $(127)_{10}=(4+127)_{10}=(10000011)_2$

尾数f = $(011011001)_2$

∴32位单精度浮点数的二进制存储格式为：

$(0100\ 0001\ 1011\ 0110\ 0100\ 0000\ 0000\ \ 0000)_2=(41B64000)_{16}$

3. 浮点数

【总结】IEEE754标准的移码与真值之间的转换：

IEEE754标准的移码 $\xrightarrow[\text{② 加 1}]{\text{① 符号位取反}}$ 真值的补码

真值的补码 $\xrightarrow[\text{② 减 1}]{\text{① 符号位取反}}$ IEEE754标准的移码

- **ASCII字符编码**

美国国家信息交换标准代码（American Standard Code for Information Interchange，简称ASCII码）是广泛使用的一种字符编码方案，1967年，成为官方标准，即ASCII码。

ASCII码选用了128个常用字符及控制码，用7位二进制位编码，通常用一个字节来表示。其中，最高位预留作为奇偶校验位，用于数据传输过程中的错误检测，在机器中通常使其为0。

5.非数值数据

ASCII编码表		0	1	2	3	4	5	6	7
	$b_6 b_5 b_4$ / $b_3 b_2 b_1 b_0$	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	Sp	0	@	P	`	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	-	=	M	]	m	}
E	1110	SO	RS	.	>	N	^	n	~
F	1111	SI	US	/	?	O	_	o	DEL

33个控制符

表 2－9 ASCII 编码中的控制码

NUL	Null 空	DLE	Data link escape 数据链路转义
SOH	Start of heading 标题开始	DC1	Device control 1 设备控制 1
STX	Start of text 文本开始	DC2	Device control 2 设备控制 2
ETX	End of text 文本结束	DC3	Device control 3 设备控制 3
EOT	End of transmission 传输结束	DC4	Device control 4 设备控制 4
ENQ	Enquiry 查询	NAK	Negative acknowledge 否定应答
ACK	Acknowledge 确认	SYN	Synchronous idle 同步
BEL	Bell (beep) 提示音（蜂鸣）	ETB	End of transmission block 传输块结束
BS	Backspace 退格	CAN	Cancel 取消
HT	Horizontal tab 水平制表	EM	End of medium 载体结束
LF	Line feed, new line 换行，新行	SUB	Substitute 替换
VT	Vertical tab 垂直制表	ESC	Escape 转义
FF	Form feed, new page 换页，新页	FS	File separator 文件分隔符
CR	Carriage return 回车	GS	Group separator 组分隔符
SO	Shift out 移出	RS	Record separator 记录分隔符
SI	Shift in 移入	US	Unit separator 单元分隔符
		DEL	Delete/Idle 删除/空位

汉字编码

- 汉字输入编码
  - 拼音
  - 字形
  - 数字
- 汉字内码
  - **GB2312-1980** **6763个汉字+682个其他符号**
  - **GBK** **21003个汉字+883个其他符号**
  - **GB18030-2000** **27484个汉字+少数民族文字**
  - **ISO/IEC10646-1、Unicode 2.0**

汉字编码

- 区位码：1981年颁布
  - 两个字节表示一个汉字，每个字节7位编码。
  - 将汉字和图形符号排列在一个94行、94列的二位代码表中，“区位码”（十进制编码）
  - 国标码=(区位码)$_{16}$+2020H
  - 汉字内码=(国标码)$_{16}$+8080H

【例】“和”

区位：	26	45
16进制：	1A	2D
国标码：	3A	4D
汉字内码：	BA	CD

汉字编码

GB18030-2000 码位范围分配图

<table>
<tr><th>字节数</th><th colspan="4">码位空间</th><th>码位数目</th></tr>
<tr><td>单字节</td><td colspan="4">00H～80H</td><td>129</td></tr>
<tr><td rowspan="2">双字节</td><td colspan="2">第一字节</td><td colspan="2">第二字节</td><td rowspan="2">23940</td></tr>
<tr><td colspan="2">81H～FEH</td><td colspan="2">40H～7EH
80H～FEH</td></tr>
<tr><td rowspan="2">四字节</td><td>第1字节</td><td>第2字节</td><td>第3字节</td><td>第4字节</td><td rowspan="2">1587600</td></tr>
<tr><td>81H～FEH</td><td>30H～39H</td><td>81H～FEH</td><td>30H～39H</td></tr>
</table>

汉字的输出

- 汉字字模码：为了将汉字的字形显示输出，汉字信息处理系统还需要配有汉字字模库，也称字形库，它集中了全部汉字的字形信息。

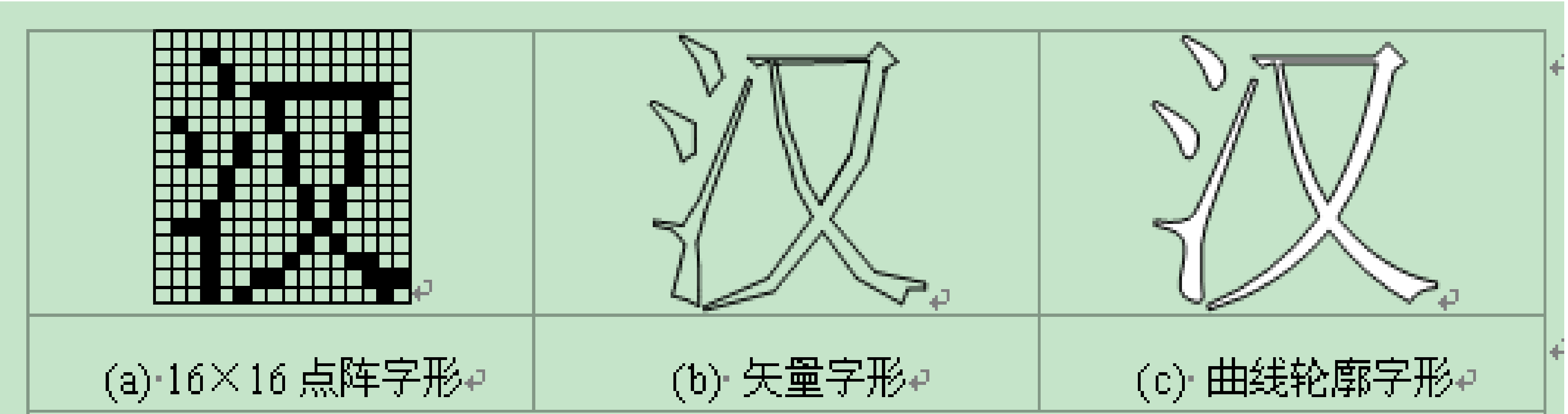


图 2-8 字形信息的表示方法

Unicode与UTF-8

- 为了统一表示世界各国的文字，1993年ISO公布了国际标准ISO/IEC 10646，简称UCS。为各种文字规定了统一的编码方案，又称为Unicode。
- Unicode的基本思想：给每一个字符和符号分配一个永久的、唯一的编码方案。
- 17个平面，每个平面有65536个码点，可表示的字符超过了110万。全世界的语言大概有20万个符号。

Unicode与UTF-8

- Unicode 只是一个符号集，它只规定了符号的二进制代码，却没有规定这个二进制代码应该如何存储。
- 出现了 Unicode 的多种存储方式：UTF-8，UCS-2，UTF-16，UCS-4，UTF-32。
- UTF-8 就是在互联网上使用最广的一种 Unicode 的实现方式。

Unicode与UTF-8

- **UTF-8 是一种变长的编码方式。它可以使用1~4个字节表示一个符号，根据不同的符号而变化字节长度。**

表 2.8 UTF-8 编码方案

字符码点范围	位数	字节 1	字节 2	字节 3	字节 4
U+0000~U+007F	7	0×××××××			
U+0080~U+07FF	11	110×××××	10××××××		
U+0800~U+FFFF	16	1110××××	10××××××	10××××××	
U+10000~U+10FFFF	21	11110×××	10××××××	10××××××	10××××××

5.非数值数据

UTF-8 的编码规则：

- **1）对于单字节的符号，字节的第一位设为0，后面7位为这个符号的 Unicode 码。因此对于英语字母，UTF-8 编码和 ASCII 码是相同的。**
- **2）对于n字节的符号（n > 1），第一个字节的前n位都设为1，第n + 1位设为0，后面字节的前两位一律设为10。剩下的没有提及的二进制位，全部为这个符号的 Unicode 码。**
- **解读 UTF-8 编码非常简单。如果一个字节的第一位是0，则这个字节单独就是一个字符；如果第一位是1，则连续有多少个1，就表示当前字符占用多少个字节。**

6. 检错与纠错码

数据出错的原因

- 元器件故障
- 存储介质
- 通信过程中的噪声干扰

如何减少或避免数据错误

- 提高计算机系硬件本身的可靠性
- 采取数据检错和校正措施：在每个字中添加一些校验位，用来确定字中出现错误的位置。

经济成本、设计目标

最小海明距离

- 两个编码字之间对应位置数值不同的位置数目为两个编码字的海明距离。
- 例：假定某编码方案有下列8个合法编码：000000、001011、010101、011110、100110、101101、110011、111000。找出任意编码字之间的海明距离，可以发现最小海明距离为$D_{min}=3$。
- 假设读到的编码字为001000，与上述8中合法编码字都不相同，因此至少存在一位错误。
- 与所有合法编码比较，得到差额向量：(1, 2, 4, 3, 4, 3, 5, 2)，取差额向量中的最小值，得到的校正结果：000000。

校验码基本概念

根据纠错理论，最小码距与检错、纠错能力的关系

检错：$D_{min} \geq D+1$

纠错：$D_{min} \geq 2C+1$

同时检错纠错：$D_{min} \geq D+C+1$，$D > C$

- 最小海明距离D_{min}越大，则检测错误位数D越大，纠正错误位数C也越大，且纠错能力恒小于或等于检错能力。
- D_{min}可以保证在发生D或C位变化时，所有合法码字间都有足够的距离加以区别，而非法码字只会和惟一的一个合法码字比较接近（海明距离最近）。
- 错误校正的方法：将非法编码字用位数相差最少（海明距离最近）的合法编码字来代替。

6. 检错与纠错码

- 设信息码为k位，欲检测1位错误，增添r位校验位，组成k+r位的校验码。为准确对错误定位或指出码字无错，校验位数r应满足：

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

r确定了2^r种状态，全0表示无错，余下的组合指示错误位的位置。

- 信息码长度k与校验位位数r的对应

k	r(最小)
1	2
2~4	3
5~11	4
12~26	5
27~57	6
58~120	7

1). 奇偶校验码

最简单且应用广泛的检错码，采用一位校验位：奇校验或偶校验。

- 设$x=x_{n-1}x_{n-2}\ldots x_0$是一个n位字，则奇校验位$\overline{C}$定义为

$$\overline{C}=x_{n-1}\oplus x_{n-2}\oplus\ldots\oplus x_0$$

式中⊕代表按位加，表明只有当x中包含有奇数个1时，才使$\overline{C}=1$，即$C=0$。

- 同理，偶校验位C定义为

$$C=x_{n-1}\oplus x_{n-2}\oplus\ldots\oplus x_0$$

即x中包含偶数个1时，才使$C=0$。

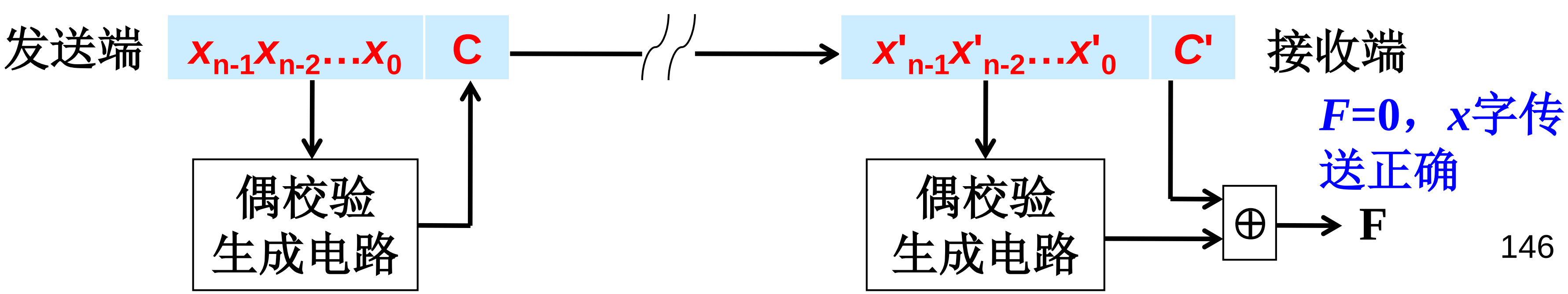
1). 奇偶校验码

- 假设信息码x要从部件 A 传送到部件 B。在源点 A处，算出校验位C，并与信息码x合并为 ($x_{n-1}x_{n-2}\ldots x_0$C)，送到B。假设在B点接收到的是$x'=(x'_{n-1}x'_{n-2}\ldots x'_0C')$，然后计算

$$F=x'_{n-1}\oplus x'_{n-2}\oplus\ldots\oplus x'_0\oplus C'$$

- 奇校验：F=0，收到信息有错；F=1，x字传送正确

  偶校验：F=1，收到信息有错；F=0，x字传送正确



6. 检错与纠错码

- 缺点：
  - 只能检测每个字中所产生的奇数个错误
  - 不具备纠错能力
- 优点
  - 开销小
  - 常用于校验1字节长的数据

  通常1字节长的数据编码发生错误时，1位出错的概率较大，两位以上同时错处的概率极小。
- 适用场合：
  - 存储器读写校验
    - 内存的存储过程，发生一位错的概率最大
    - 电路简单、速度高、易于实现
  - 按字节传输中的数据校验：串行传输

6. 检错与纠错码

字符‘A’（ASCII码为41H）的奇校验编码可能是

A. 01000001

B. 11000001 √

C. 10000010

D. 10000011 √

6. 检错与纠错码

行列校验码（水平垂直一致校验码、二维奇偶校验码、矩阵码）

– 垂直冗余检查VRC（Vertical Redundancy Check）
– 纵向冗余检查LRC（Longitudinal Redundancy Check ）

例：给定一个字符串“We are friends.”,采用二维奇偶校验检查（偶校验）

1. 给出发送端的二维数据组织
2. 人为引入错误，演示VRC和LRC的检错过程

6. 检错与纠错码

列号

行号	8 (VRC)	7	6	5	4	3	2	1	字符	7位ASCII	偶校验ASCII
1	1	1	0	1	0	1	1	1	W	57	D7
2	0	1	1	0	0	1	0	1	e	65	65
3	1	0	1	0	0	0	0	0		20	A0
4	1	1	1	0	0	0	0	1	a	61	E1
5	0	1	1	1	0	0	1	0	r	72	72
6	0	1	1	0	0	1	0	1	e	65	65
7	1	0	1	0	0	0	0	0		20	A0
8	0	1	1	0	0	1	1	0	f	66	66
9	0	1	1	1	0	0	1	0	r	72	72
10	0	1	1	0	1	0	0	1	i	69	69
11	0	1	1	0	0	1	0	1	e	65	65
12	1	1	1	0	1	1	1	0	n	6E	EE
13	1	1	1	0	0	1	0	0	d	64	E4
14	0	0	1	0	1	1	1	0	.	2E	2E
15	0	1	1	1	1	0	0	0	LRC		

6. 检错与纠错码

行号	8 (VRC)	7	6	5	4	3	2	1	字符	7位ASCII	偶校验ASCII
	列号										
1	1	1	0	1	0	1	1	1	W	57	D7
2	0	1	1	0	0	1	0	1	e	65	65
3	1	0	1	0	0	0	0	0		20	A0
4	1	1	0	0	0	0	0	1	a	61	E1
5	0	1	1	1	0	0	1	0	r	72	72
6	0	1	1	0	0	1	0	1	e	65	65
7	1	0	1	0	0	0	0	0		20	A0
8	0	1	1	0	0	1	1	0	f	66	66
9	0	1	1	1	0	0	1	0	r	72	72
10	0	1	1	0	1	0	0	1	i	69	69
11	0	1	1	0	0	1	0	1	e	65	65
12	1	1	1	0	1	1	1	0	n	6E	EE
13	1	1	1	0	0	1	0	0	d	64	E4
14	0	0	1	0	1	1	1	0	.	2E	2E
15 (LRC)	0	1	1	1	1	0	0	0			

6. 检错与纠错码

行号	8 (VRC)	7	6	5	4	3	2	1	字符	7位ASCII	偶校验ASCII
1	1	1	1	1	0	1	0	1	W	57	D7
2	0	1	1	0	0	1	0	1	e	65	65
3	1	0	1	0	0	0	0	0		20	A0
4	1	1	1	0	0	0	0	1	a	61	E1
5	0	1	1	1	0	0	1	0	r	72	72
6	0	1	1	0	0	1	0	1	e	65	65
7	1	0	1	0	0	0	0	0		20	A0
8	0	1	1	0	0	1	1	0	f	66	66
9	0	1	1	1	0	0	1	0	r	72	72
10	0	1	1	0	1	0	0	1	i	69	69
11	0	1	1	0	0	1	0	1	e	65	65
12	1	1	1	0	1	1	1	0	n	6E	EE
13	1	1	1	0	0	1	0	0	d	64	E4
14	0	0	1	0	1	1	1	0	.	2E	2E
15 (LRC)	0	1	1	1	1	0	0	0			

6. 检错与纠错码

行号	8	7	6	5	4	3	2	1	字符	7位ASCII	偶校验ASCII
1	1	1	0	1	0	1	1	1	W	57	D7
2	0	1	1	0	0	1	0	1	e	65	65
3	1	0	1	0	0	0	0	0		20	A0
4	1	1	0	0	0	0	0	1	a	61	E1
5	0	1	1	1	0	0	1	0			
6	0	1	1	0	0	1	0	1			
7	1	0	1	0	0	0	0	0			
8	0	1	1	0	0	1	1	0			
9	0	1	1	1	0	0	1	0			
10	0	1	1	0	1	0	0	1			
11	0	1	1	0	0	1	0	1			
12	1	1	1	0	1	1	1	0			
13	1	1	1	0	0	1	0	0	d	64	E4
14	0	0	1	0	1	1	1	0	.	2E	2E
15	0	1	1	1	1	0	0	0	LRC		

- 可发现并纠正数据块中的1位错误
- 可发现两位同时出错，但无法纠正错误

6. 检错与纠错码

2). 循环冗余校验码——CRC (Cyclic Redundancy Check)

- 在m位信息码（目标发送数据）后再拼接k位校验码，使整个编码长度为n位，因此这种编码也叫（n,m）码。
- 一种常用的纠错码，能检出2位错，纠正1位错，用于数据包传输。CRC码在数据通信中被广泛采用，也常用于外存储器的数据校验。
- 编码与校验电路简单。
- 编码原理：利用某种数学运算建立有效信息位与校验位之间的约定关系。循环码是一种基于模2运算建立编码规律的校验码。

CRC校验的具体做法

（1）选定一个标准除数（k位二进制数据串）

（2）在要发送的数据（m位）后面加上K-1位0，然后将这个新数（m+k-1位）以模2除法的方式除以上面这个标准除数，所得到的余数也就是该数据的CRC校验码（注：余数必须比除数少且只少一位，不够就补0）

（3）将这个校验码附在原n位数据后面，构成新的n=m+k-1位数据，发送给接收端。

（4）接收端将接收到的数据除以标准除数，如果余数为0则认为数据正确。

6. 检错与纠错码

- 采用模2算术运算
  - 通过模2减法实现模2除法；
  - 以模2加法将所得余数拼接在被校验数据的后面，形成能除尽的被校验数据。
- 生成多项式应满足的要求：
  - 任何一位发生错误都应使余数不为0；
  - 不同位发生错误应当使余数不同；
  - 应满足余数循环规律
- 生成多项式的表示
  - 如，生成多项式G=1011，表示生成多项式为$G(x)=x^3+x+1$

6. 检错与纠错码

- 常用的生成多项式

N	K	码距d	G(x)多项式	G(x)
7	4	3	x^3+x+1	1011
7	4	3	x^3+x^2+1	1101
7	3	4	$x^4+x^3+x^2+1$	11101
7	3	4	x^4+x^2+x+1	10111
15	11	3	x^4+x+1	10011
15	7	5	$x^8+x^7+x^6+x^4+1$	111010001
31	26	3	x^5+x^2+1	100101
31	21	5	$x^{10}+x^9+x^8+x^6+x^5+x^3+1$	11101101001
63	57	3	x^6+x+1	1000011
63	51	5	$x^{12}+x^{10}+x^5+x^4+x^2+1$	1010000110101

西安电子科技大学 XIDIAN UNIVERSITY

6. 检错与纠错码

【例】

① 假设信息字节为 $F=1001010_2$；

② 选取 $G=1011_2$；

③ 将F左移 $l-1$ 位，形成 $F'=1001010000_2$；

④ 用 F′ 做被除数、G 做除数，进行模2除法。忽略商，余数为 $R=111_2$。

⑤ 把余数加到 F' 中，组成要发送的信息M：

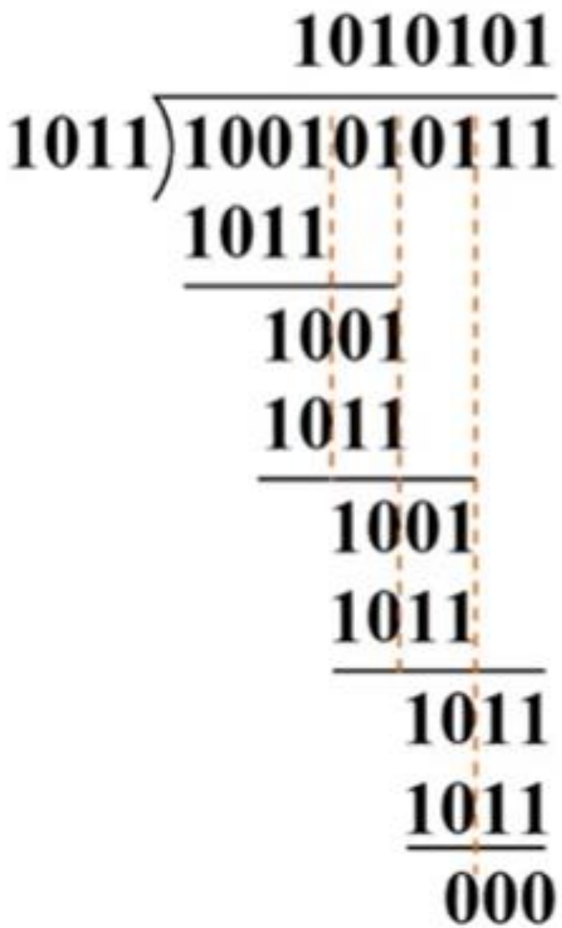
$1001010000_2+111_2=1001010111_2$。

```
           1010101
     ______________
1011 ) 1001010000
       1011
       ----
        1001
        1011
        ----
          1000
          1011
          ----
           1100
           1011
           ----
            111
```

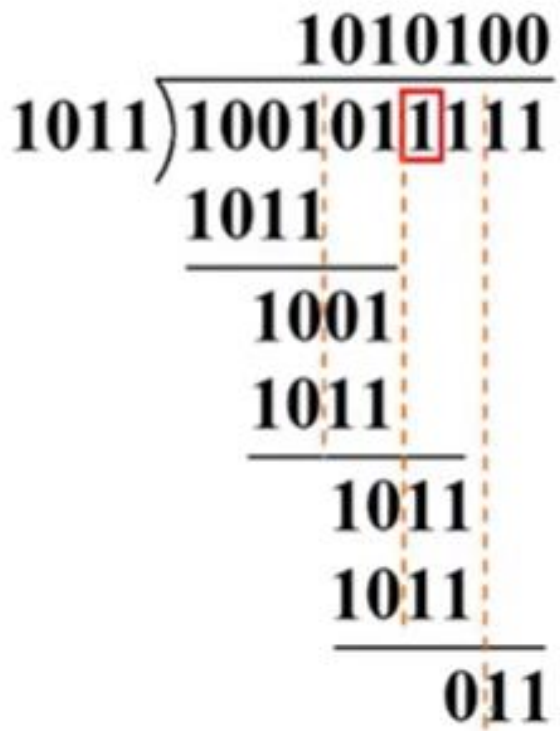
6. 检错与纠错码

【例】

接收器：解码校验（正确的情况）



接收器：解码校验（1位出错的情况）



XIDIAN UNIVERSITY

6. 检错与纠错码

$G(X)=1011_2$	编码举例1		编码举例2		余数	出错位置
	数据位 6 5 4 3	校验位 2 1 0	数据位 6 5 4 3	校验位 2 1 0		
正确	1 0 0 1	1 1 0	1 1 0 0	0 1 0	0 0 0	无
错误	1 0 0 1	1 1 1	1 1 0 0	0 1 1	0 0 1	0
	1 0 0 1	1 0 0	1 1 0 0	0 0 0	0 1 0	1
	1 0 0 1	0 1 0	1 1 0 0	1 1 0	1 0 0	2
	1 0 0 0	1 1 0	1 1 0 1	0 1 0	0 1 1	3
	1 0 1 1	1 1 0	1 1 1 0	0 1 0	1 1 0	4
	1 1 0 1	1 1 0	1 0 0 0	0 1 0	1 1 1	5
	0 0 0 1	1 1 0	0 1 0 0	0 1 0	1 0 1	6

(7，4)循环码编码、余数与出错位置的关系

6. 检错与纠错码

- CRC的生成多项式的阶数越高，误判的概率就越小。
- 常用CRC标准生成多项式
  - CRC-12

$$X^{12}+X^{11}+X^{3}+X^{2}+X+1$$

  - CRC-16

$$X^{16}+X^{15}+X^{2}+1$$

  - CRC-CCITT（国际电联）

$$X^{16}+X^{12}+X^{5}+1$$

  - CRC-32

$$X^{32}+X^{26}+X^{23}+X^{22}+X^{16}+X^{12}+X^{11}+X^{10}+X^{8}+X^{7}+X^{5}+X^{4}+X^{2}+X+1$$

3）.海明码

- 某些系统需要具备纠正合理数量错误的能力。
- 海明码（**hamming code**）由**Richard Hamming**于**1950**年提出。
  - 主要用于存储器数据的校验与纠正。
  - 采用奇偶校验的原理，错误检测和校正能力随着信息字中加入奇偶校验位的数目线性增加。
  - 适用于最有可能发生随机错误的系统
    - 每一位的出错概论相同
    - 每一位与其他位是否出错没有任何关联

海明编码步骤

1. 计算校验位的位数

假设信息位为 k 位， 增加 r 位校验位，构成 n=k+r 位海明码字。若要求海明码能纠正一位错误，用 r 位校验位产生的 r 位指误字来区分无错状态及码字中 n 个不同位置的一位错误状态，则要求 r 满足：$2^r \geq r+k+1$。

表3.6 有效信息位数 k 与校验位位数 r 的对应关系

k 值	r 最小值
1～4	3
5～11	4
12～26	5
27～57	6
58～120	7

2. 确认有效信息和校验位的位置

假设 k 位有效信息从高到低为 $D_k\ D_{k-1}\ \ldots D_2\ D_1$，添加的 r 位校验位为 $P_r\ P_{r-1}\ \ldots P_2\ P_1$，则它们构成 n=k+r 位的海明码排列设为 $H_nH_{n-1}\ \ldots H_2\ H_1$， H 的下标被称为海明位号，则第 i位的校验位 P_i 必须位于位号为 2^{i-1} 的位置，即 $P_i = H_j$， $j = 2^{i-1}$，其中， i=1， 2，…r；有效信息则在其余的海明码位置上顺序排列。

例如，校验位 P_1 位于第 1 位（ H_1），校验位 P_2 位于第 2 位（H_2），校验位 P_3 位于第 4位（ H_4），以此类推。 k=8， r=4 的海明码的排列如下

H_{12}	H_{11}	H_{10}	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1

3. 分组

分组的原则是：校验位只参加一组奇偶校验，有效信息则参加至少两组的奇偶校验，若 $D_i = H_j$，则 D_i 参加那些位号之和等于 j 的校验位的分组校验。例如，D_1（H_3）必须参加 P_1P_2 组的校验，因为 3=2+1，D_7（H_{11}）必须参加 $P_4P_1P_2$ 组的校验，因为 11=8+2+1。k=8，r=4 的海明码的分组如表 3.7。

表3.7　k=8，r=4 的海明码分组

序号 \ 分组	H_{12}	H_{11}	H_{10}	H_9	H_8	H_7	H_6	H_5	H_4	H_3	H_2	H_1
	D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
P_4	√	√	√	√	√							
P_3	√					√	√	√	√			
P_2		√	√			√	√			√	√	
P_1		√		√		√		√		√		√

4. 进行奇偶校验，合成海明码

首先，按照分组和奇偶校验的规律将每个校验位的生成表达式写出，然后，再带入有效信息的值，依次得出校验位的取值，最后将校验位按各自的位置插入，与有效信息一起合成海明码。

k=8， r=4 的海明码分组偶校验表达式如下：

$$P_4 = D_8 \oplus D_7 \oplus D_6 \oplus D_5$$

$$P_3 = D_8 \oplus D_4 \oplus D_3 \oplus D_2$$

$$P_2 = D_7 \oplus D_6 \oplus D_4 \oplus D_3 \oplus D_1$$

$$P_1 = D_7 \oplus D_5 \oplus D_4 \oplus D_2 \oplus D_1$$

4. 进行奇偶校验，合成海明码

$$P_4 = D_8 \oplus D_7 \oplus D_6 \oplus D_5$$

$$P_3 = D_8 \oplus D_4 \oplus D_3 \oplus D_2$$

$$P_2 = D_7 \oplus D_6 \oplus D_4 \oplus D_3 \oplus D_1$$

$$P_1 = D_7 \oplus D_5 \oplus D_4 \oplus D_2 \oplus D_1$$

如有效信息为 11101001，则可以纠错一位的海明码为 1 1 1 0 1 1 0 0 0 1 0 1，带下划线的是校验位 $P_4\,P_3\,P_2\,P_1$，分别取值 1001。

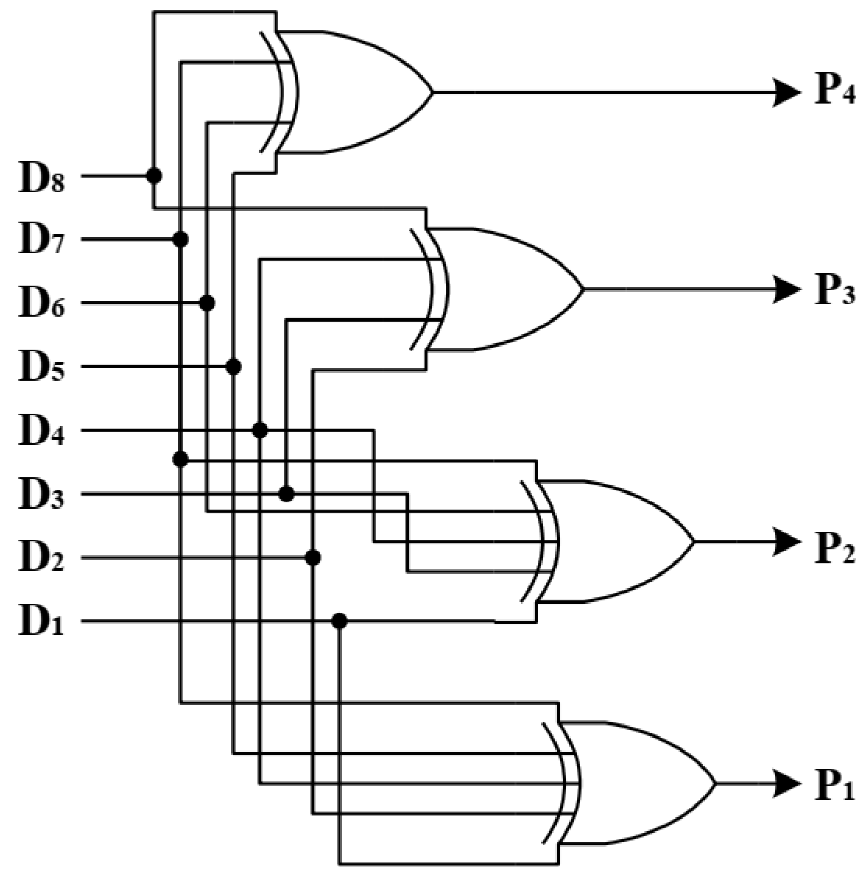


图3.7 k=8，r=4 的海明校验的编码电路

译码：

按照监督关系式算出指误字 $S_r\ S_{r-1}\ldots S_2\ S_1$， 若为全零，则说明各组奇偶性全部无变化，信息正确，将相应的有效信息位析取出来使用； 否则，指误字的十进制值，就是出错位的海明位号。

k=8， r=4 的海明码分组偶校验的监督表达式:

$$S_4 = P_4 \oplus D_8 \oplus D_7 \oplus D_6 \oplus D_5$$

$$S_3 = P_3 \oplus D_8 \oplus D_4 \oplus D_3 \oplus D_2$$

$$S_2 = P_2 \oplus D_7 \oplus D_6 \oplus D_4 \oplus D_3 \oplus D_1$$

$$S_1 = P_1 \oplus D_7 \oplus D_5 \oplus D_4 \oplus D_2 \oplus D_1$$

6. 检错与纠错码

译码：

$$S_4 = P_4 \oplus D_8 \oplus D_7 \oplus D_6 \oplus D_5$$

$$S_3 = P_3 \oplus D_8 \oplus D_4 \oplus D_3 \oplus D_2$$

$$S_2 = P_2 \oplus D_7 \oplus D_6 \oplus D_4 \oplus D_3 \oplus D_1$$

$$S_1 = P_1 \oplus D_7 \oplus D_5 \oplus D_4 \oplus D_2 \oplus D_1$$

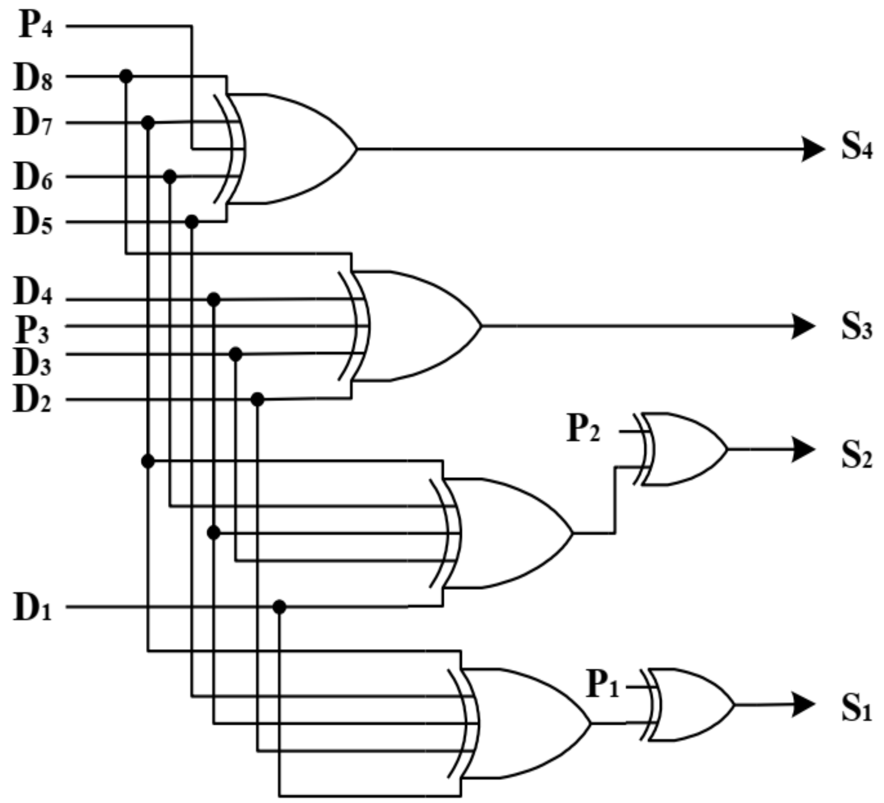


图3.8　k=8，r=4 的海明校验的译码电路

收到的海明码为 1 1 0 0 1 1 0 0 0 1 0 1，计算得到 $S_4\,S_3\,S_2\,S_1$=1010，则表明是 H_{10}（D_6）出错，将 H_{10} 取反即可，正确海明码为 1 1 1 0 1 1 0 0 0 1 0 1。

6. 检错与纠错码

P3～P0编码的十进制值就是出错位的序号，利用常见的4-16译码器输出，与相应的数据位相异或，达到纠错的目的。

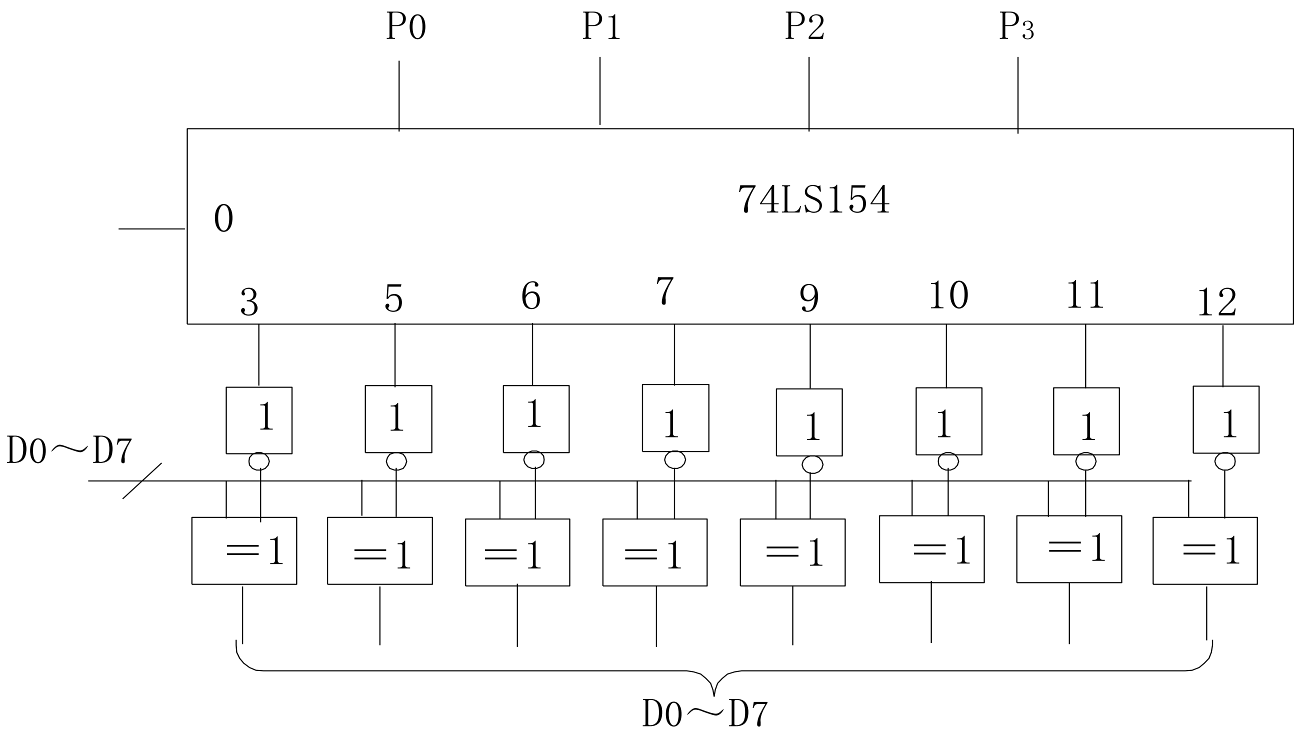


图2.13　纠错电路原理图

THE END！

THANKS