



计算机组成原理



二、计算机数据表示



本章主要内容

- 2.1 数据表示的作用
- 2.2 数值数据表示
- 2.3 非数值数据表示
- 2.4 数据信息的校验



2.1 数据表示的作用

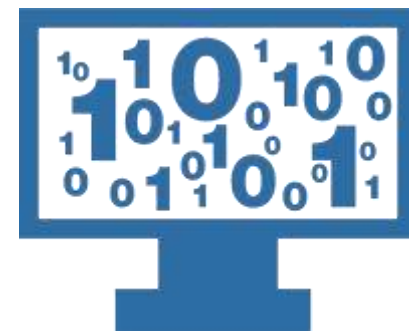
■ 将数据按照某种方式组织，以便机器硬件能直接识别和使用

■ 数据表示考虑因素

- **数据的类型：** 数值/非数值、小数、整数、英文字符、汉字
- **表示的范围和精度：** 满足日常计算需要
- **存储和处理的代价：** 硬件开销，处理性能
- **软件的可移植性：** 方便在不同机器之间移植

■ 现代计算机采用**二进制**进行数据表示

- **可以表示任何数据信息**
- **状态数最少，易与简单的物理状态对应，运算电路易实现**



不同进制编码特点

■ 十进制

- 0123456789共10种状态, 状态过多

- 运算组合状态过多

 - ◆ 10进制加法组合数 = $C_{10}^2 + 10 = 55$ 8进制 36 种 2进制 3 种

■ 二进制

- 符号数最少, “0、1” 物理上容易实现,

- 可以表示任何对象(字符, 数值, 逻辑值)

- 运算规则简单

 - ◆ $0 + 1 = 1 + 0 = 1$ $1 + 1 = 0$ $0 + 0 = 0$

 - ◆ 仅仅三种运算规则, 一个异或门即可完成该运算

进制表示

$$N = \sum_{i=-k}^m D_i * r^i$$

N 代表一个数值

r 是这个数制的基(Radix)

i 表示这些符号排列的位号

D_i 是位号为i的位上的一个符号

r^i 是位号为i的位上的 1 代表的值

$D_i * r^i$ 是第i位的所代表的实际值，表示m+k+1位的值求累加和

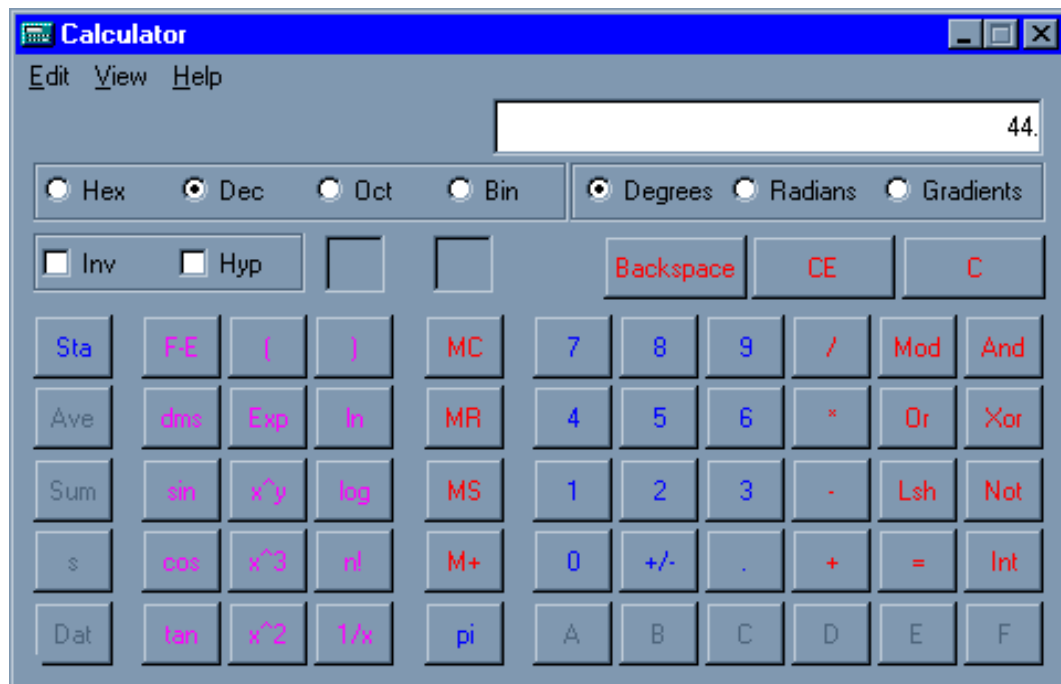
■ $(10456)_{10} = 1 \times 10^4 + 0 \times 10^3 + 4 \times 10^2 + 5 \times 10^1 + 6 \times 10^0$

■ $(0xF96)_{16} = F \times 16^2 + 9 \times 16^1 + 6 \times 16^0$

■ $(10010001)_2 = 1 \times 2^7 + 0 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 0 \times 2^1 + 1 \times 2^0$

进制转换

- 二进制数转八进制
- 二进制数转十六进制
- 二进制数转十进制
- 十进制数转二进制



二到八或十六进制转换

■ 二进制转到八进制 从小数点向左右三位一分组

$$(10\ 011\ 100\ .\ 01)_2$$

$$(10\ 011\ 100\ .\ 010)_2 = (234\ .\ 2)_8$$

■ 二进制转十六进制 从小数点向左右四位一分组

$$(1001\ 1100\ .\ 01\)_2$$


$$(1001\ 1100\ .\ 0100)_2 = (9C\ .\ 4)_{16}$$

小数部分不足位数要补零凑足，否则出错

十进制转二进制

11.6 转二进制


整数部分除2取余

$11 \div 2 = 5 \dots 1$	 高位
$5 \div 2 = 2 \dots 1$	
$2 \div 2 = 1 \dots 0$	
$1 \div 2 = 0 \dots 1$	

除尽为止

1011

小数部分乘2取整

$0.6 * 2 = 1.2$	 高位
$0.2 * 2 = 0.4$	
$0.4 * 2 = 0.8$	
$0.8 * 2 = 1.6$	

余数为零或满足位数为止

0.1001 1001...

进制转换的简单运算方法

■ 17/128的二进制表示方法???

■ 大数的转换方法，记住几个常用的2的幂

$$2^5 = 32 \quad 2^6 = 64 \quad 2^7 = 128 \quad 2^8 = 256 \quad 2^9 = 512$$

$$2^{10} = 1024 (1\text{Kilo}) \quad 2^{11} = 2048 \quad 2^{12} = 4096$$

$$2^{13} = 8192 \quad 2^{14} = 16384 \quad 2^{15} = 32768$$

$$2^{16} = 65536 \quad 2^{20} = 1048576 \quad (1\text{Mega})$$

$$2^{30} = 1073741824 \quad (1\text{Giga}) \quad 2^{40} = 1\text{Tera}$$

更大的单位?

$$2^{50} = 1 \text{ Peta} \quad 2^{60} = 1 \text{ Exa} \quad 2^{70} = 1 \text{ Zetta} \quad 2^{80} = 1 \text{ Yotta}$$



几个简化运算的例子

■ $130 = 128 + 2 = 10000010$

■ $65539 = 65536 + 3 = 100000000000000011$

■ $2010 = 2047 - 37 = 11111111 - 32 - 4 - 1$

■ $11111110111 = 2^{12} - 1 - 8$

■ $17/128 = 10001/10000000 = 0.0010001$

Kilo, Mega, Giga, Tera, Peta, Exa, Zetta, Yotta physics.nist.gov/cuu/Units/binary.html

■ 1Mbits/s = ???

□ 1 Mbit/s = 10^6 bps

■ SI (International System of Units) 仅指10进制

■ 硬盘厂商及通讯行业是计算机行业唯一使用SI因子的

Name	Abbr	Factor	SI size
Kilo	K	$2^{10} = 1,024$	$10^3 = 1,000$
Mega	M	$2^{20} = 1,048,576$	$10^6 = 1,000,000$
Giga	G	$2^{30} = 1,073,741,824$	$10^9 = 1,000,000,000$
Tera	T	$2^{40} = 1,099,511,627,776$	$10^{12} = 1,000,000,000,000$
Peta	P	$2^{50} = 1,125,899,906,842,624$	$10^{15} = 1,000,000,000,000,000$
Exa	E	$2^{60} = 1,152,921,504,606,846,976$	$10^{18} = 1,000,000,000,000,000,000$
Zetta	Z	$2^{70} = 1,180,591,620,717,411,303,424$	$10^{21} = 1,000,000,000,000,000,000,000$
Yotta	Y	$2^{80} = 1,208,925,819,614,629,174,706,176$	$10^{24} = 1,000,000,000,000,000,000,000,000$

本章主要内容

- 2.1 数据表示的作用
- 2.2 数值数据表示
- 2.3 非数值数据表示
- 2.4 数据信息的校验



2.2 数值数据表示方法

■ 2.2.1 数的机器码表示

■ 2.2.2 定点数表示

■ 2.2.3 浮点数表示

■ 2.2.4 十进制编码

■ 2.2.5 计算机中的数据类型

2.2.1 数的机器码表示

■ 真值 (书写用)

□ 将用 “+” 、 “-” 表示正负的二进制数称为真值

■ 机器不能识别书写格式，计算机如何表示负数？

■ 机器码 (机器内部使用)

□ 将符号和数值一起编码表示的二进制数称为机器码

□ **原码** Signed magnitude **反码** One' s complement

□ **补码** Two' s complement **移码** Biased notation

原码表示法 (Signed magnitude)

- 增加符号位 Add a sign bit
- 最高位为符号位, 0: 正, 1: 负, 数值位不变
 - 符号位的权值是多少?

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

符号位权值是 2^n

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

符号位权值是1

原码表示示例

■ $[+0]_{\text{原}} = 0.000\dots0$

■ $[-0]_{\text{原}} = 1.000\dots0$ 两个机器零

■ $[-0.1111]_{\text{原}} = 1.1111$

■ $[+0.1111]_{\text{原}} = 0.1111$

■ $[+111011]_{\text{原}} = 0111011$

■ $[-111011]_{\text{原}} = 1111011$

原码表示区间

■ 定点小数 $x_0.x_1x_2\cdots x_n$

□ $-0.111\dots1 \sim +0.111\dots1$

□ $[-(1-2^{-n}), 1-2^{-n}]$ 或 $(-1, 1)$

■ 定点整数 $x_0x_1x_2\cdots x_n$

□ $-0111\dots1 \sim +0111\dots1$

□ $[-(2^n-1), 2^n-1]$ 或 $(-2^n, 2^n)$

■ 对称区间

原码特性

■ 直观易懂

- 第一位为符号位、其他为数值位

■ 两个机器零

■ 加、减运算方式不统一

- 符号相异加法不能直接运算
- 特别当 $a < b$ 时，实现 $a - b$ 比较困难

■ 从50年代开始，整数都采用补码来表示

■ 但浮点数的尾数用原码定点小数表示

$$01011001_2 = 89_{10}$$

$$+ \underline{11001101}_2 = \underline{-77}_{10}$$

$$00100110_2 = 38_{10}$$

反码表示法 One's Complement

■ 所谓反码，就是二进制的各位数码取反，符号位与原码相同

■ Example: $7_{10} = 00111_2$

$$-7_{10} = 11000_2$$

■ $[+0]_{\text{反}} = 0.000\dots0$

$[-0]_{\text{反}} = 1.111\dots1$

两个机器零

■ $[0.1111]_{\text{反}} = 0.1111$

$[-0.1111]_{\text{反}} = 1.0000$

■ $[111011]_{\text{反}} = 01110$

$[-111011]_{\text{反}} = 1000100$

反码表示法...

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

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

反码公式证明

■ 定点小数 $-1 < x \leq 0$ 时

□ 假设 $x = -0.x_1x_2\cdots x_n$

□ 假 $[x]_{\text{反}} = 1.x_1x_2\cdots x_n$

□ $[x]_{\text{反}} + |x| = 1.11\cdots 1$

$$= 1.11\cdots 1 + 0.00\cdots 1 - 0.00\cdots 1$$

$$= 10.00\cdots 0 - 0.00\cdots 1$$

$$= 2 - 2^{-n}$$

■ $[x]_{\text{反}} = 2 - 2^{-n} - |x| = 2 - 2^{-n} + x$

■ 定点整数证明方法相同

反码表示区间

■ 定点小数 $x_0.x_1x_2\dots x_n$

□ $-0.111\dots 1 \sim +0.111\dots 1$

□ $[-(1-2^{-n}), 1-2^{-n}]$ 或 $(-1, 1)$

■ 定点整数 $x_0x_1x_2\dots x_n$

□ $-0111\dots 1 \sim +0111\dots 1$

□ $[-(2^n-1), 2^n-1]$ 或 $(-2^n, 2^n)$

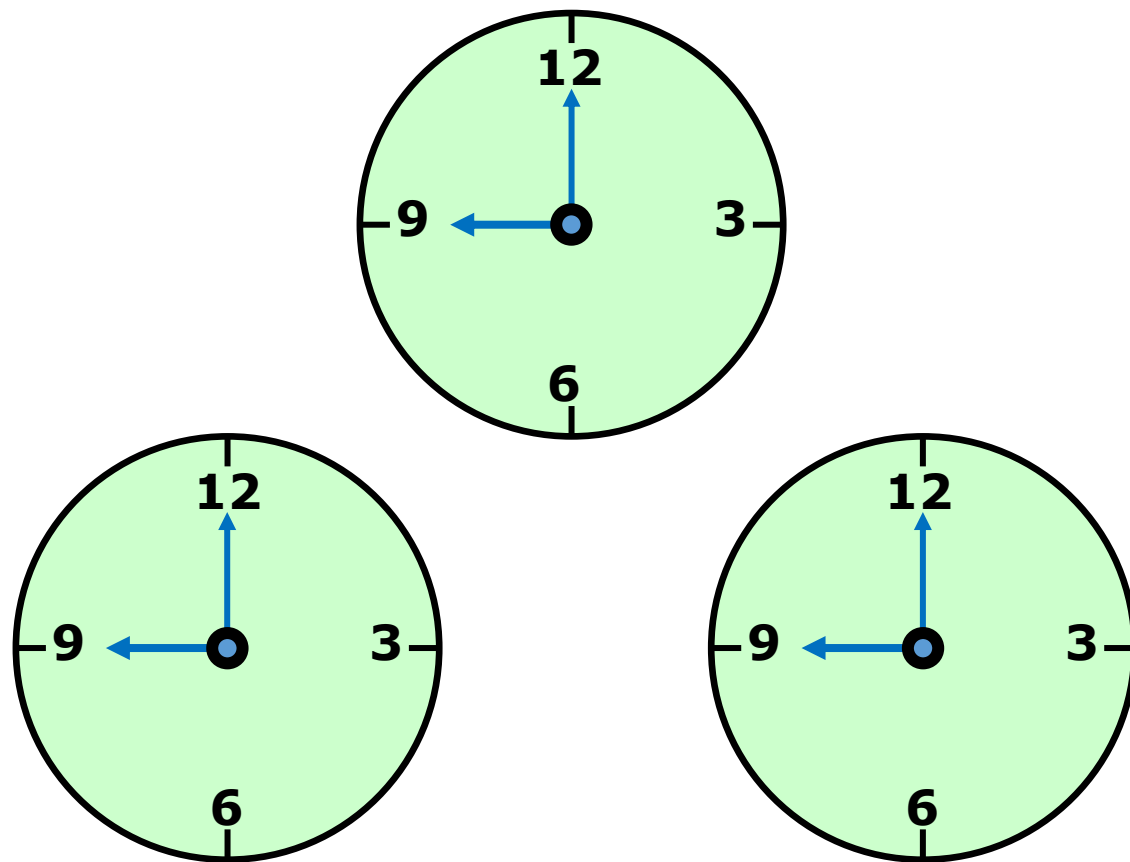
■ 对称区间

反码特性

- 两个机器零
- 求反用逻辑门容易实现
- 运算仍然很复杂
 - 相加时需要将符号位的进位位增加到最低有效位上

有趣的时钟

■ 9与-3、21等效



同余的概念

- 假定有两个数a和b，若用某一个整数m去除，所得的余数相同，就称a,b两个数对m同余,记作：

$$a \equiv b \pmod{m} \quad \text{模为} m$$

- 假设X,Y,Z三个数，满足下列关系： $Z=nX+Y$ (n为整数),则称Z和Y对模X是同余的，记作：

$$Z \equiv Y \pmod{X} \quad Y \equiv Z \pmod{X}$$

- 以12为模 $9 \equiv 12 + 9 \equiv 24 + 9 \equiv 36 + 9$

- $9 \equiv 21 \equiv 33 \equiv 45 \equiv -3 \equiv 12 - 3 \equiv 9$

例子

■ $7 + (-3)$

$$= 7 + (12 - 3)$$

$$= 7 + 9$$

$$= 16$$

$$= 4$$

■ 表示负数的时候如利用模的性质转换成正数,

■ 即可将原码运算中的减法变成加法运算

补码公式

- 模：符号位进位位的权值
- 真值为正数，补码等于原数据
- 真值为负数，增加一个模

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

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

补码与反码的关系

定点小数

$$[x]_{\text{反}} = 2 - 2^{-n} + x$$

$$[x]_{\text{补}} = 2 + x$$

$$= (2 - 2^{-n} + x) + 2^{-n}$$

$$= [x]_{\text{反}} + 2^{-n}$$

整数

$$[x]_{\text{反}} = 2^{n+1} - 1 + x$$

$$[x]_{\text{补}} = 2^{n+1} + x$$

$$= (2^{n+1} - 1 + x) + 1$$

$$= [x]_{\text{反}} + 1$$

当X为负数时，补码等于反码末位加1

补码编码的简便方法

- 正值直接取其原来的二进制码，符号位为0

- 负值则逐位取反，末位加1，符号位为1

- $[-10101010]_{\text{补}}$

$$= 1\ 01010101 + 1$$

$$= 1\ 01010110$$

- $[-0.010101]_{\text{补}} = 1.101011$

- 扫描法

- 从最右侧开扫描找到第一个1，该数位左侧所有数据位取反，其他数据位不变

例子

■ $X = -0.11111111$

$$[X]_{\text{补}} = ???$$

$$\begin{aligned}[X]_{\text{补}} &= 1.00000000 \\ &+ 0.00000001 \\ &= 1.00000001\end{aligned}$$

■ $X = -0.00000001$

$$[X]_{\text{补}} = 1.11111111$$

■ $X = -000000001$

$$[X]_{\text{补}} = 111111111$$

■ $X = -0.00000000$

$$[X]_{\text{补}} = ???$$

$$\begin{aligned}[X]_{\text{补}} &= 1.11111111 \\ &+ 0.00000001 \\ &= \underline{10}.00000000 \\ &= 0.00000000\end{aligned}$$

补码特性

■ 零有唯一的表示方式

$$[+0.0000]_{\text{补}} = [-0.0000]_{\text{补}} = 0.0000$$

■ -1.0的补码

$$[-1.0000]_{\text{补}} = 0.1111 + 0.0001 = 1.0000$$

$$2 + X = 2 - 1 = 1$$

补码表示区间

■ 定点小数 $x_0.x_1x_2\dots x_n$

□ $-1.000\dots 0 \sim +0.111\dots 1$

□ $[-1, 1-2^{-n}]$ 或 $[-1, 1)$

■ 定点整数 $x_0x_1x_2\dots x_n$

□ $-1000\dots 0 \sim +0111\dots 1$

□ $[-2^n, 2^n-1]$ 或 $[-2^n, 2^n)$

■ 非对称区间，左侧多一个数

双符号位补码（变形补码） 模=?

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

例: 0001010110
 1101010001

$$[X]_{\text{补}} = \begin{cases} X & 0 \leq X < 1 \\ 4 + X & -1 \leq X < 0 \end{cases}$$

例: 00.01010110
 11.01010001

符号位01表示正溢出，10表示负溢出，最高位表示正确符号位

补码加减法的实现

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

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

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

$$[[Y]_{\text{补}}]_{\text{补}} = \text{对 } [Y]_{\text{补}} \text{ 逐位取反, 再在最低位加 } 1$$

补码加减法运算实例

$x=0.1011$ $y=-0.0101$ 用模4补码 求 $x+y$ $x-y$

$$[x]_{\text{补}} = 00\ 1011, \quad [y]_{\text{补}} = 11\ 1011$$

$$[-y]_{\text{补}} = 00\ 0101$$

$$\begin{array}{r} 00\ 1011 \\ +\ 11\ 1011 \\ \hline 100\ 0110 \end{array}$$

$x+y$

$$\begin{array}{r} 00\ 1011 \\ +\ 00\ 0101 \\ \hline 01\ 0000 \end{array}$$

$x-y$

补码表示中的符号位扩展

由 $[X]_{\text{补}}$ 求 $[X/2]_{\text{补}}$

原符号位不变，符号位与数值位均右移一位，

$[X]_{\text{补}} = 10010$ 则 $[X/2]_{\text{补}} = 11001$ 符号向右扩展

$[2x]_{\text{补}} = ?$ 左移，末位补零，符号变化溢出

不同位数的整数补码相加减时，如何运算

$$\begin{array}{r} 0101010111000011 \\ + \quad \quad \quad 10011100 \\ \hline ?????????????????? \end{array}$$

$$\begin{array}{r} 0101010111000011 \\ + \quad \quad \quad 00011100 \\ \hline ?????????????????? \end{array}$$

补码表示中的符号位扩展...

■ 不同位数的整数补码相加减时

□ 位数少的补码符号位向左扩展

□ 一直扩展到符号位对齐

$$\begin{array}{r} 0101010111000011 \\ + 10011100 \\ \hline ? \end{array}$$

$$\begin{array}{r} 0101010111000011 \\ + 1111111110011100 \\ \hline 0101010101011111 \end{array}$$

补码特性

- 唯一的机器零
- 符号位可以直接参与运算
- 减法可以变成加法，运算电路统一
- 负数比整数多一个

移码表示法 Biased/Excess Notation

定义

$$[x]_{\text{移}} = 2^n + x \quad -2^n \leq x < 2^n$$

保持数据原有大小顺序，便于直接进行比较，偏移量也可是非幂次方常量

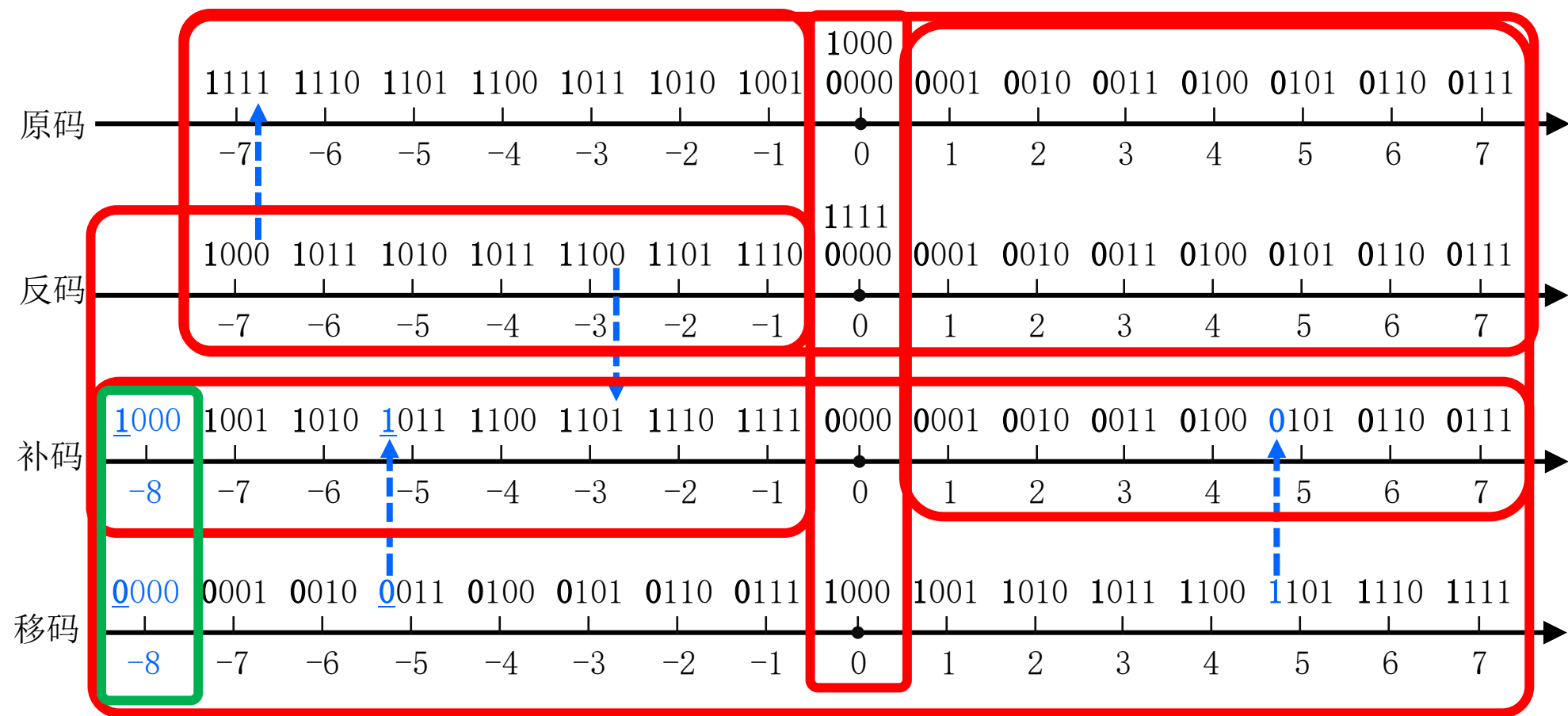
□ 与补码的符号位相异，数据位相同

□ $X = +10101$ $[X]_{\text{移}} = 2^5 + 10101 = 110101$

□ $X = -10101$ $[X]_{\text{移}} = 2^5 - 10101 = 001011$

□ 仅用于表示整数，通常表示浮点数的阶码

不同机器码对比



负数的补码等于反码末位加1

不同机器码公式对比

	$-2^n \leq x \leq 0$	$-1 \leq x \leq 0$	$x \geq 0$
原码	$2^n - x$	$1 - x$	x
反码	$2^{n+1} - 1 + x$	$2 - 2^{-n} + x$	x
补码	$2^{n+1} + x$	$2 + x$	x
移码	$2^n + x$	无	$2^n + x$

定点数机器码表示范围

■ $n+1$ 位定点数，数据位 n 位

	定点整数		定点小数	
原码反码	$[1-2^n, 2^n-1]$	$(-2^n, 2^n)$	$[-2^{-n}-1, 1-2^{-n}]$	$(-1, 1)$
补码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	$[-1, 1-2^{-n}]$	$[-1, 1)$
移码	$[-2^n, 2^n-1]$	$[-2^n, 2^n)$	小数无移码	

机器码小结

■ 4类机器码应用

- 原码 用来表示浮点（实）数的尾数
- 反码 已不用于表示数值数据
- 补码 50年代开始成为整数标准
- 移码 用于浮点数阶码

■ 补码优势

- 模运算，加、减运算统一
- 唯一0，方便使用



课前习题

1. 将二进制补码 11111011 转换为十进制数
2. -25转换为8位的原码、反码和补码
3. 使用8位补码计算 $45 - 67$

课前习题答案

1. 将二进制补码 11111011 转换为十进制数

补码: 11111011 \rightarrow 反码: 11111010 (补码减1) \rightarrow 原码: 10000101 (反码取反) \rightarrow 十进制数: -5

2. -25转换为8位的原码、反码和补码

数值部分为25的二进制 (0011001) , 原码: 10011001, 反码: 11100110, 补码: 11100111

3. 使用8位补码计算45 - 67

45的补码: 00101101

-67的补码: 原码11000011 \rightarrow 反码10111100 \rightarrow 补码10111101

相加: 00101101 + 10111101 = 11101010

结果补码11101010转换为原码: 减1得11101001 \rightarrow 取反得10010110 对应十进制: $-(16 + 4 + 2) = -22$

结果补码: 11101010, 十进制结果: -22

2.2 数值数据表示方法

■ 2.2.1 数的机器码表示

■ 2.2.2 定点数表示

■ 2.2.3 浮点数表示

■ 2.2.4 十进制编码

■ 2.2.5 计算机中的数据类型

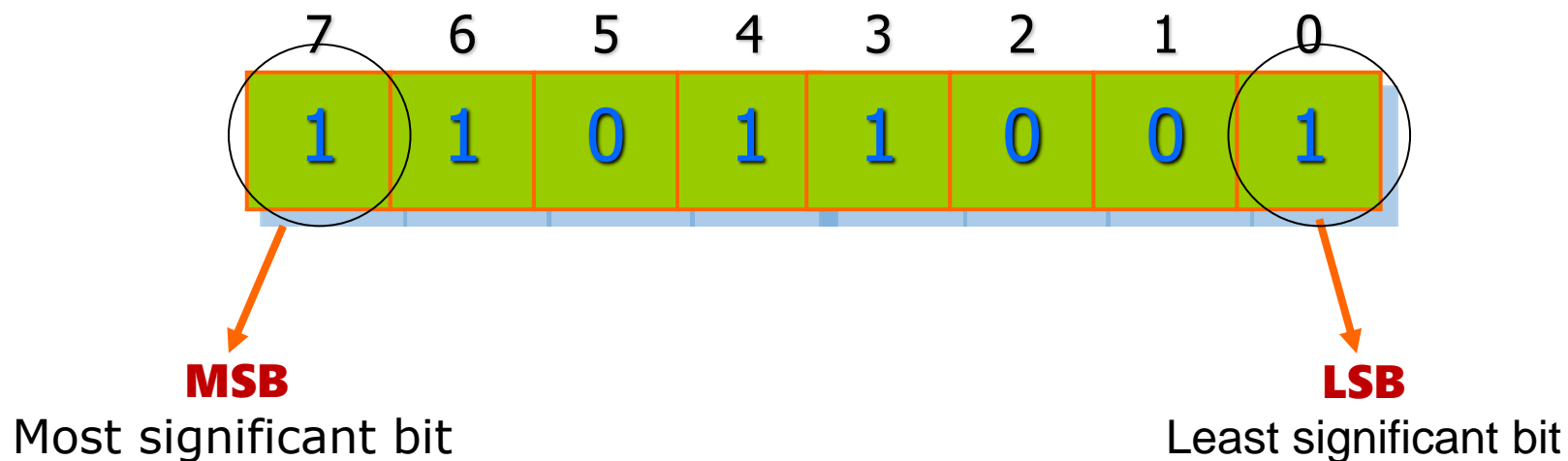
2.2.2 定点数表示

■ 定点表示 (小数点位置**固定**的数)

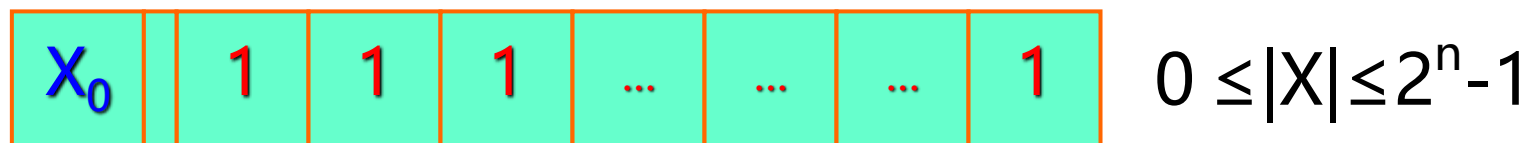
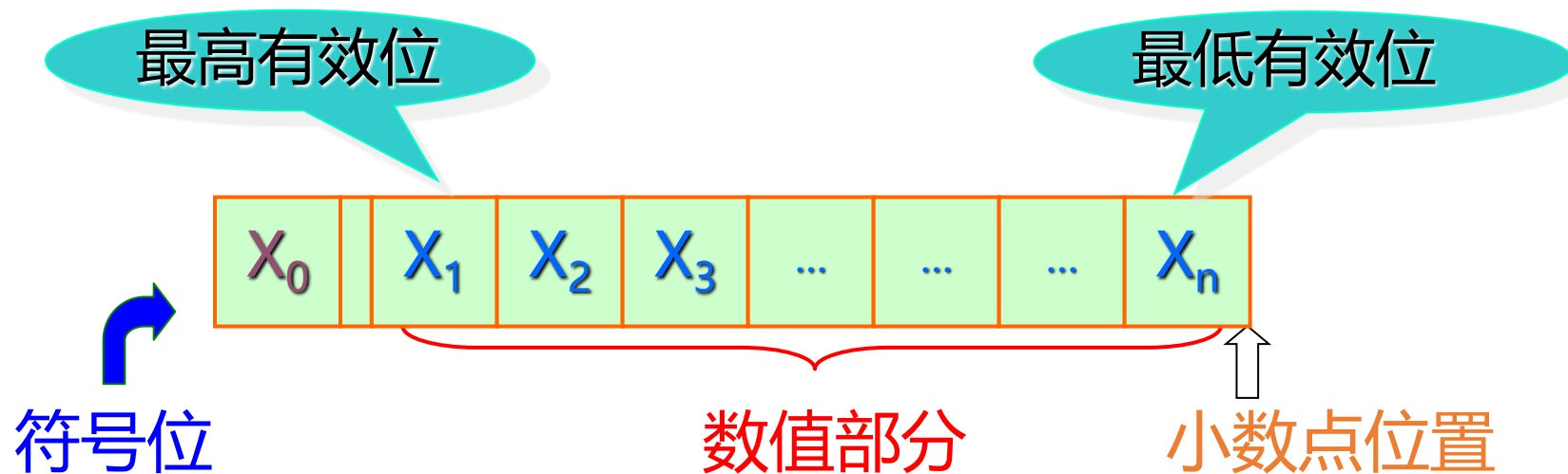
- 定点整数
- 定点小数
- 仅能表示纯小数及纯整数

基本术语Terminology

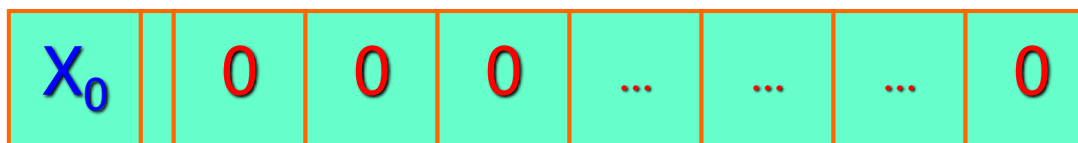
- 计算机利用寄存器存储数据
- 寄存器中每个位称bit (Binary Digit)
- 最高有效位 (**MSB**) 最低有效位 (**LSB**)



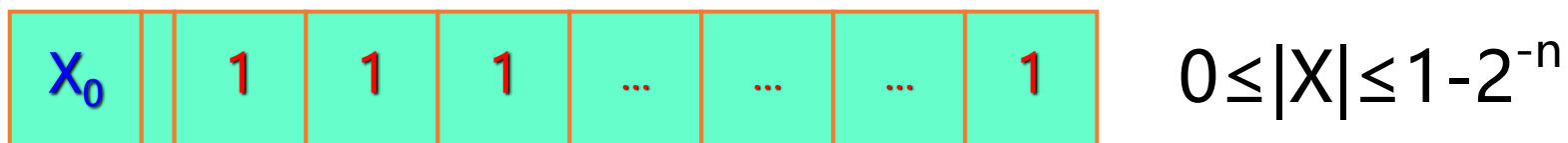
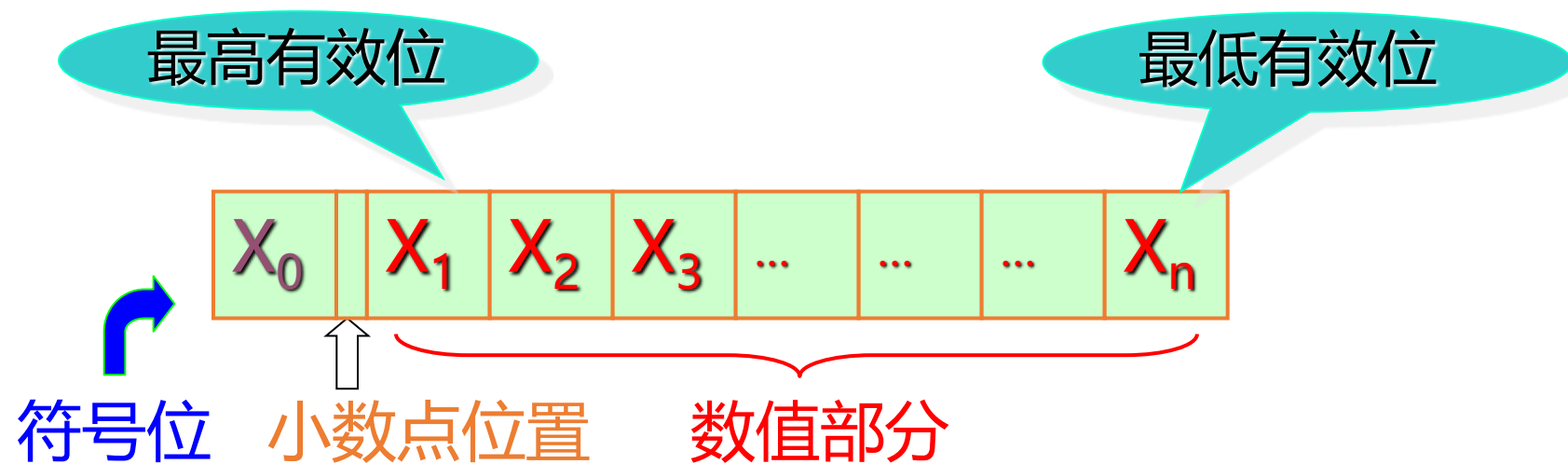
定点整数



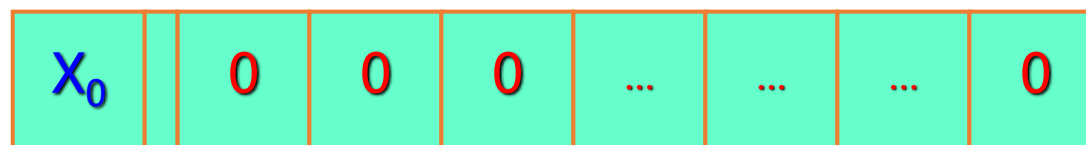
上溢



定点小数



下溢/上溢



2.2 数值数据表示方法

- 2.2.1 数的机器码表示

- 2.2.2 定点数表示

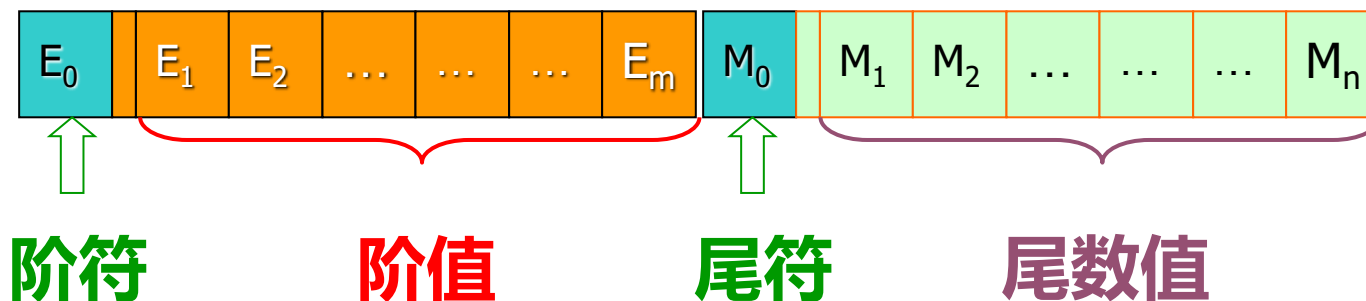
- 2.2.3 浮点数表示

- 2.2.4 十进制编码

- 2.2.5 计算机中的数据类型

浮点数的表示

■ $N = 2^E \times M = 2^{\pm e} \times (\pm 0.m)$

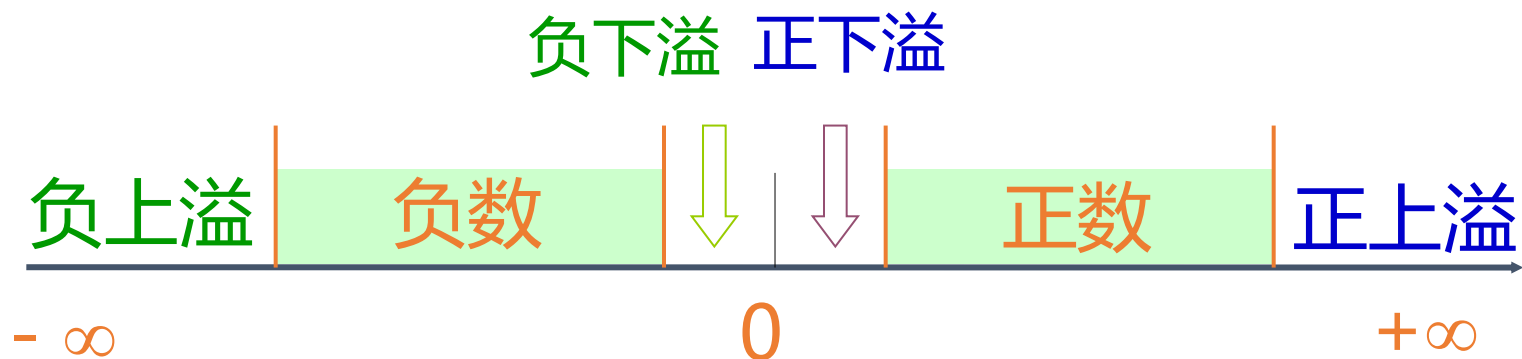


- 二进制浮点数可表示成**阶码**E(Exponent)和**尾数**M(Mantissa)两部分，其中**阶码E是定点整数**，而**尾数M是定点小数**。
- **阶码的位数决定数据表示的范围**，阶码的位数越多，能表示的数据范围就越大，而阶码的值决定了小数点的位置；**尾数的位数决定数据表示的精度**。
- 阶码长度相同时，分配给尾数的数位越多，数据表示的精度就越高。

浮点数的表示范围与精度

- 阶码越长，表示范围越大，精度越高
- 阶码相同，尾数越长，数据精度越高
- 浮点数扩大了数值表示的范围，未增加表示数值的个数
- 绝对值越大，浮点数分布越稀疏，浮点数是离散空间

浮点数的表示范围



- 浮点数有效扩大了数据表示范围，但受计算机字长限制，浮点数仍然存在溢出现象。图中“**负数区域**”“**正数区域**”及“0”是可表示的数据区域。
- 当浮点数绝对值超过正数最大值时发生**上溢**，左、右两侧分别对应正上溢和负上溢，分别表示正无穷和负无穷；当非0浮点数绝对值小于正数最小值时，发生**下溢**。
- 若运算结果发生上溢，浮点运算器件会显示溢出标志；若运算结果发生下溢，虽然此时数据不能被精确表示，但由于发生下溢时数的绝对值很小，可作为机器0处理。同样，当一个浮点数在正、负数区域中但并不在某个数轴刻度上时，也会出现精度溢出的问题，此时只能用近似数表示。

规格化

- 同一浮点数如果采用浮点数格式，可能存在多种表示形式，如 0.01111×2^{101} 还可以表示成 0.11110×2^{100} 。尾数小数点的位置不同，就会有不同的尾数和阶码组合，这将给浮点数的表示带来麻烦。
- 为了使浮点数的表示形式唯一并进一步提高数据的表示精度，通常要求浮点数在数据表示时对尾数进行规格化处理。所谓**规格化处理就是使得尾数真值最高有效位为1**

$$N = 2^E \times M = 2^{\pm e} \times (\pm 1.m)$$

- 在实际表示时可以采用与定点数小数点一样的表示策略，无须单独表示最高有效位上的1，需要进行数据运算时再恢复最高有效位的表示，被隐藏的这一位又称为**隐藏位**。

例子 (规格化)

- 设阶码2位, 尾数4位 (隐藏位, 规格化)

- 可表示 $2^{-11} * 1.0000$ --- $2^{11} * 1.1111$

- 0.001 --- 1111.1

- 设阶码3位, 尾数3位 (隐藏位, 规格化)

- 可表示 $2^{-111} * 1.000$ --- $2^{111} * 1.111$

- 0.0000001 --- 11110000

“Father” of the IEEE 754 standard

- 80年代初，各机器内部浮点数表示还没有统一
- 相互不兼容，机器之间传送数据时很麻烦
- 1970年代后期, IEEE着手制定浮点数标准
- 1985年完成浮点数标准IEEE 754的制定，沿用至今

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.



www.cs.berkeley.edu/~wkahan/ieee754status/754story.html



Prof. William Kahan

浮点数标准 IEEE754

32浮点数 *float*

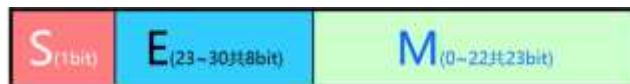


64浮点数 *double*



- 构成：阶码E，尾数M，符号位S
- 阶码和尾数的位数的分配实际是表示范围和精度的折衷
- $N = (-1)^S \times M \times 2^E$
- 阶码移码表示，偏移量127/1023，尾数原码表示，符号位放在第一位
 - 32浮点数偏移量是127而不是标准移码的128，采用127 偏移量的最主要的原因是使得任何一个规格化数的倒数能用另外一个浮点数表示，而采用128 偏移量表示的最小规格化数的倒数会发生溢出。

浮点数标准 IEEE754



■ 规格化数(Normal)

$$1 \leq e \leq 254/2046 \quad (-1)^S \times \mathbf{1}.M \times 2^{E-127} \quad (-1)^S \times \mathbf{1}.M \times 2^{E-1023}$$

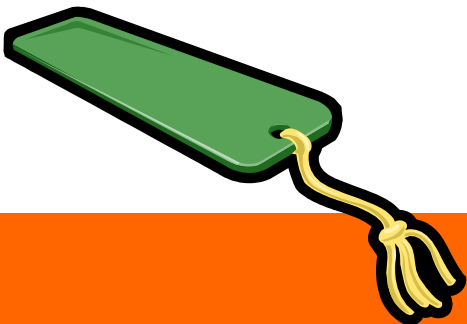
■ 非规格化数(Subnormal)

$$e=0 \quad (-1)^S \times 0.M \times 2^{-126} \quad (-1)^S \times 0.M \times 2^{-1022}$$

■ 尾数原码表示，表示区间对称

■ 规格化数最高数字位恒为1，该位为**隐藏位**，可节约尾数数据位

32位单精度浮点数编码格式

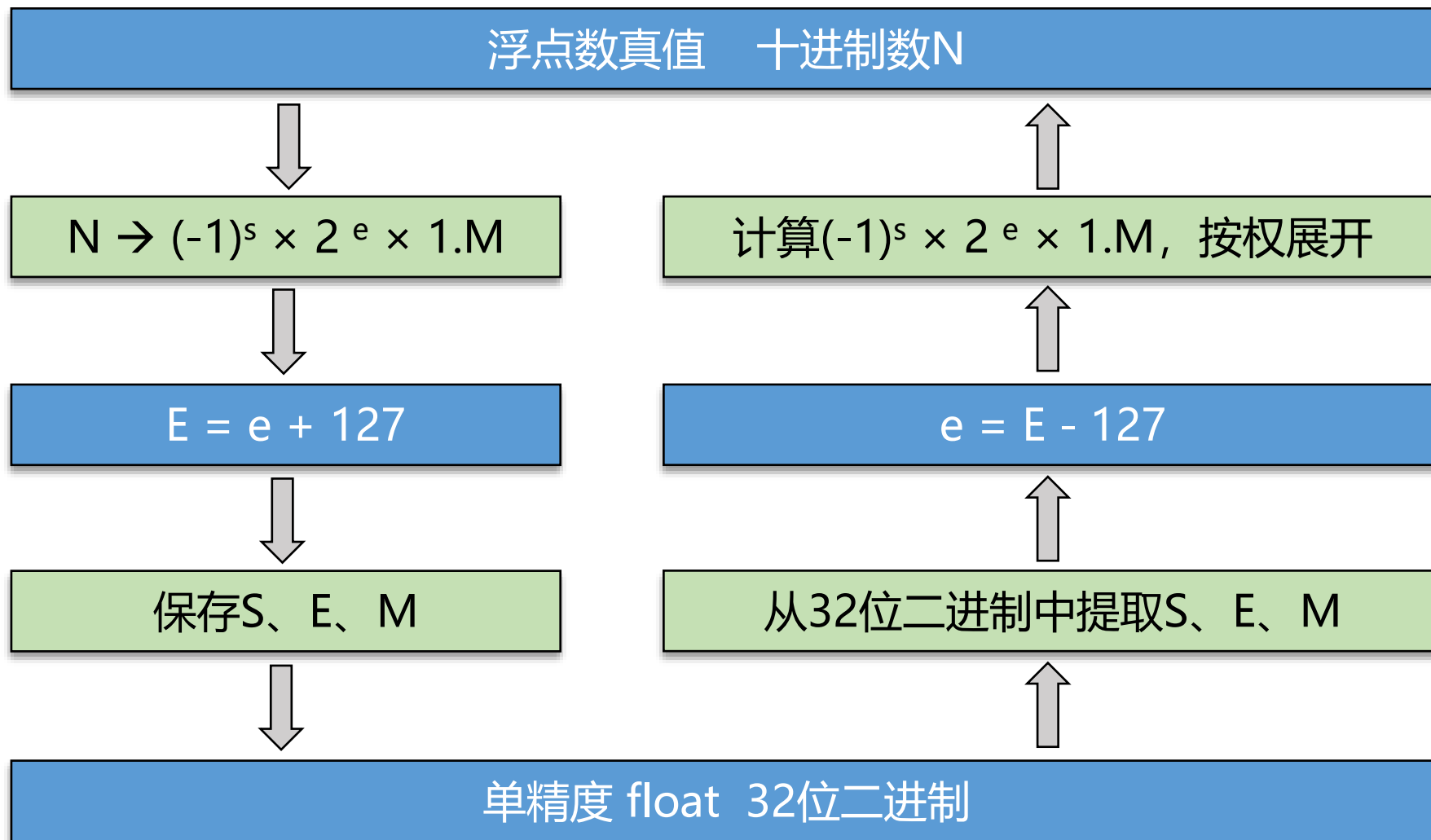


符号位	阶码	尾数	表示
0/1	255	1xxxx	NaN Not a Number
0/1	255	非零 0xxxx	sNaN Signaling NaN
0	255	0	$+\infty$
1	255	0	$-\infty$
0/1	1~254	M	$(-1)^S \times (1.M) \times 2^{(E-127)}$
0/1	0	M (非零)	$(-1)^S \times (0.M) \times 2^{(-126)}$
0/1	0	0	+0/-0

IEEE754 规格化浮点数表示范围

格式	最小值	最大值
单精度规格化 $(-1)^s \times 1.m \times 2^{E-127}$	$E_{\min}=1, M=0,$ $1.0 \times 2^{1-127} = 2^{-126}$	$E_{\max}=254, M=1.1111...1 \times 2^{254-127}$ $= 2^{127} \times (2-2^{-23})$ 约 $+3.4 \times 10^{38}$
单精度非规格化 $(-1)^s \times 0.m \times 2^{-126}$	$E=0, M=2^{-23},$ $2^{-23} \times 2^{-126} = 2^{-149}$	$E=0, M=0.1111...1 \times 2^{-126}$ $= 2^{-126} \times (1-2^{-23})$
双精度规格化 $(-1)^s \times 1.m \times 2^{E-1023}$	$E_{\min}=1, M=0,$ $1.0 \times 2^{1-1023} = 2^{-1022}$	$E_{\max}=2046,$ $M=1.1111...1 \times 2^{2046-1023}$ $= 2^{1023} \times (2-2^{-52})$
双精度非规格化 $(-1)^s \times 0.m \times 2^{-1022}$	$E=0, M=2^{-52},$ $2^{-52} \times 2^{-1022} = 2^{-1079}$	$E=0, M=0.11111...1 \times 2^{-1022}$ $= 2^{-1022} \times (1-2^{-52})$

float型与真值之间的变换流程



浮点数转换实例

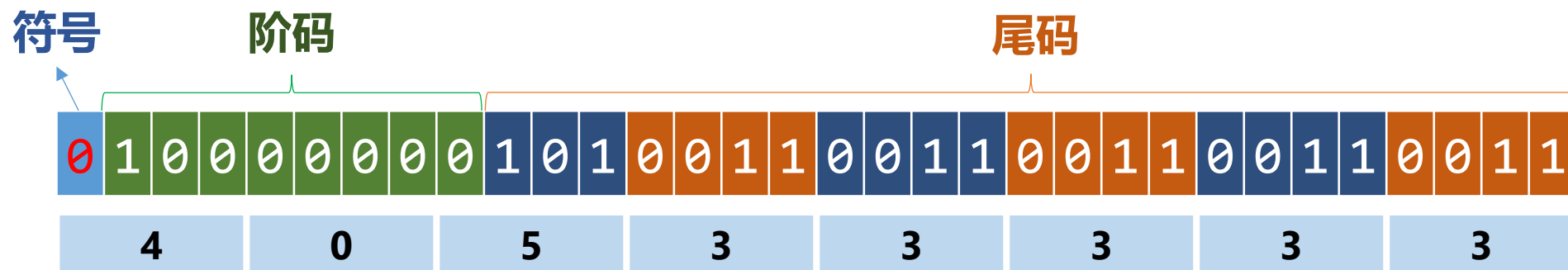
■ $3.3_{10} \rightarrow \text{IEEE 754 float}$

十进制转换二进制 $3.3_{10} = 11.01001100110011001100110011\ldots_2$

规格化 $= 1.1010011001100110011001100110011 \times 2^1$

■ 取23位尾数，舍入后会变小

■ 8位阶码 $= 1 + 127 = 10000000_2$



浮点数转换实例

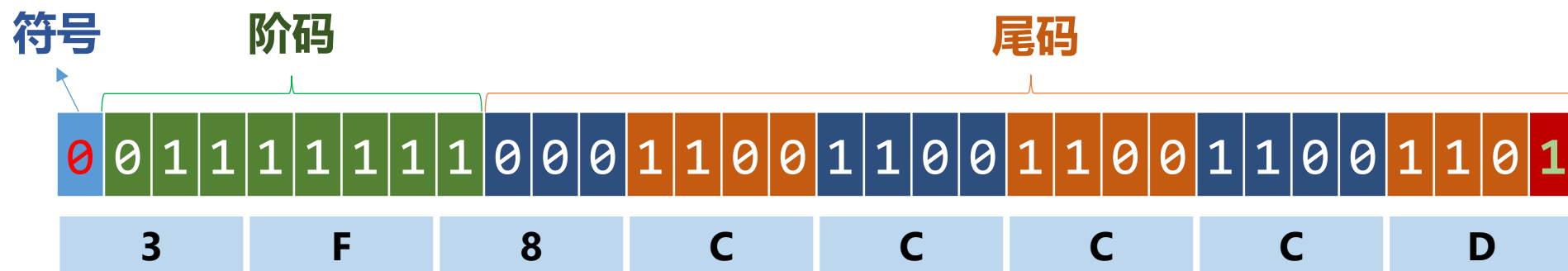
■ $1.1_{10} \rightarrow$ IEEE 754 float

十进制转换二进制 $1.1_{10} = 1.00011001100110011001100110011\dots_2$

规格化 $= 1.00011001100110011001100110011 \times 2^0$

■ 取23位尾数，舍入后会变大

■ 8位阶码 $= 0 + 127 = 01111111_2$



习题： P55页, 2.9题中 (1) 和 (3) , 2.10, 2.13

2.9 (1)

首先转换成二进制数： (-110.101)

规格化的形式： $-110.101 = -1.10101 \times 2^2$ 。

得到：

$S=1$, $e=2$, $E=2+127=10000001$, $M=101\ 01000000\ 0000\ 0000\ 0000$

最后得到32位浮点数的二进制存储格式为：

$11000000110101000000000000000000 = (C0D40000)_{16}$

2.9 (3)

将64000转换成二进制数(111110100000 0000)

变成1.M的形式:1111101000000000=1.1111010000000000 $\times 2^{15}$ 。

于是得到:

$S=0$, $e=15$, $E=15+127=10001110$, $M=11110100000\ 0000\ 00000000$

最后得到32位浮点数的二进制存储格式为:

01000111011110100000000000000000=(477A0000)₁₆

2.10

$$43940000H = (01000011100101000000000000000000)_2$$

$$S=0, \quad E=(10000111)_2 - 127 = 8, \quad M=1.00101$$

所以表示数为 1.00101×2^8 , 对应的十进制数为 296。

2.13

1. 最大正数, 阶码: 011, 尾码: 0.111111, 真值: $2^3 \times (1 - 2^{-6})$
2. 最小正数, 阶码: 100, 尾码: 0.000001, 真值: $2^{-4} \times 2^{-6}$
3. 最大负数, 阶码: 100, 尾码: 1.111111, 真值: $-2^{-4} \times 2^{-6}$
4. 最小负数, 阶码: 011, 尾码: 1.000000, 真值: -2^3

2.2 数值数据表示方法

■ 2.2.1 数的机器码表示

■ 2.2.2 定点数表示

■ 2.2.3 浮点数表示

■ 2.2.4 十进制编码

■ 2.2.5 计算机中的数据类型

2.2.4 十进制编码

■ Binary coded decimal 二进制编码的十进制

■ 几种BCD码

- 8421码 $(8 \times X_3 + 4 \times X_2 + 2 \times X_1 + 1 \times X_0)$ 有权码
- 2421码 $(2 \times X_3 + 4 \times X_2 + 2 \times X_1 + 1 \times X_0)$ 有权码
- 余三码 $(8 \times X_3 + 4 \times X_2 + 2 \times X_1 + 1 \times X_0) + 0011$

表 2.8 十进制数的 8421 码、2421 码和余 3 码

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

- 这3种编码在进行算术运算时都需要进行结果的校正，对8421码实现算术运算时，如果两数之和大于10，则运算结果需要加6修正，并向高位进位;
- 2421码运算对结果的修正问题比8421码更为复杂;
- 余3码进行加法时如不产生进位，则运算结果需要减3; 若产生进位，则将进位送入高位，当前位加3校正。

2.2 数值数据表示方法

- 2.2.1 数的机器码表示
- 2.2.2 定点数表示
- 2.2.3 浮点数表示
- 2.2.4 十进制编码
- 2.2.5 计算机中的数据类型

2.2.5 计算机中的数据类型

■ 汇编语言中的数据类型

- 寄存器、内存数据没有数据类型
- 具体类型取决于指令操作符

■ 高级语言中的数据类型

- 有具体的数据类型
- C语言 char、int、float 等

汇编语言中不同指令集的数据运算类型

ISA	无符号运算	有符号运算	浮点运算
X86	ADD/SUB 加减		FADD/FSUB 加减
	MUL/DIV 乘除	IMUL/IDIV 乘除	FMUL/FDIV 乘除
MIPS32	ADDU/SUBU 加减	ADD/SUB 加减	ADD.S ADD.D /SUB.S SUB.D 加减
	MULTU/DIVU 乘除	MULT/DIV 乘除	MUL.S MUL.D / DIV.S DIV.D 乘除
RISC-V32	ADD/SUB 加减		FADD.S FADD.D / FSUB.S FSUB.D 加减
	MULHU/DIVU 乘除	MULH/DIV 乘除	FMUL.S FMUL.D / FDIV.S FDIV.D 乘除

高级语言中的数据类型

■ 非数值数据 Qualitative

- C语言 char型

■ 数值数据 Quantitative

- 整数 Integers

- ◆ Signed / Unsigned

- ◆ C语言 short int long 型

- ◆ int8_t uint8_t int16_t uint16_t int32_t uint32_t int64_t uint64_t

- 非整数 Non-integers (Real)

- ◆ Signed / Unsigned

- ◆ C语言 float double 型

程序中数据类型的宽度

■ 高级语言支持多类型、多长度的数据

- C语言中char型的宽度为1字节
- 可表示一个字符（非数值数据）
- 也可表示一个8位的整数（数值数据）

■ 同类型数据在不同机器上位宽有差异

- 分配字节数与机器字长和编译器相关

C变量	典型32位机	Compaq Alpha
char	1	1
short	2	2
int	4	4
long int	4	8
char *	4	8
float	4	4
double	8	8

Compaq Alpha是64位机器

整形变量表示范围

■ 有符号数采用补码表示

值		<i>char</i> (8 位)	<i>short</i> (16 位)	<i>int</i> (32 位)	<i>long</i> (64 位)
无符号 最大值	机器码	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
	真值	255	65,535	4,294,967,295	18,446,744,073,709,551,615
有符号 最小值	机器码	0x80	0x8000	0x80000000	0x8000000000000000
	真值	-128	-32,768	-2,147,483,648	-9,223,372,036,854,775,808
有符号 最大值	机器码	0x7F	0x7FFF	0x7FFFFFFF	0x7FFFFFFFFFFFFFFF
	真值	127	32,767	2,147,483,647	9,223,372,036,854,775,807
-1	机器码	0xFF	0xFFFF	0xFFFFFFFF	0xFFFFFFFFFFFFFFFF
0	机器码	0x00	0x0000	0x00000000	0x0000000000000000

本章主要内容

- 2.1 数据表示的作用
- 2.2 数值数据表示法
- 2.3 非数值数据表示法
- 2.4 数据信息的校验



2.3 非数值数据表示法

- 2.3.1 字符表示法 characters
- 2.3.2 汉字表示法 Chinese characters

字符表示法

- 如何使用数值表示字符数据
- ASCII码（American Standard Code for Information Interchange），中文翻译为：美国信息交换标准代码。被国际标准化组织（International Organization for Standardization, ISO）定为国际标准，称为ISO 646标准，适用于所有拉丁文字字母。ASCII码由美国国家标准学会（American National Standard Institute , ANSI）于1967年第一次规范发布，1986年为最近一次更新。

128 Standard ASCII codes

ASCII 码使用一个字节进行编码，分为标准ASCII 码与扩展ASCII 码。标准ASCII 码也叫基础ASCII码，规定最高位为0，其他7位表示数值，其范围为0~127，包括编码32个控制符、10个数字、52个大小写字母及其他符号。

表 9.1 ASCII 字符编码表

b ₆ b ₅ b ₄				000	001	010	011	100	101	110	111
b ₃	b ₂	b ₁	b ₀								
0	0	0	0	NUL	DLE	SP	0	@	P	'	p
0	0	0	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	STX	DC2	"	2	B	R	b	r
0	0	1	1	ETX	DC3	#	3	C	S	c	s
0	1	0	0	EOT	DC4	\$	4	D	T	d	t
0	1	0	1	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	ACK	SYN	&	6	F	V	f	v
0	1	1	1	BEL	ETB	'	7	G	W	g	w
1	0	0	0	BS	CAN	(8	H	X	h	x
1	0	0	1	HT	EM)	9	I	Y	i	y
1	0	1	0	LF	SUB	*	:	J	Z	j	z
1	0	1	1	VT	ESC	+	;	K	[k	{
1	1	0	0	FF	FS	,	<	L	\	l	
1	1	0	1	CR	GS	—	=	M]	m	}
1	1	1	0	SO	RS	•	>	N	^	n	~
1	1	1	1	SI	US	/	?	O	-	o	DEL

ASCII

表 1-5 标准 ASCII 概括总结

分类	十六进制值	二进制值	十进制值	符号
32 个控制符	0x00	0000 0000	0	NUL (null) 空字符
	
	0x1F	0001 1111	31	US (unit separator) 单元分隔符
空格及 15 个标点符号	0x20	0010 0000	32	(space) 空格
	0x21~0x2F	0010 0001~0010 1111	33~47	! " # \$ % & (右单引号) () * + , - . /
10 个数字	0x30~0x39	0011 0000 ~ 0011 1001	48~57	0~9
6 个符号	0x3A~0x40	0011 1010~0100 0000	58~64	: ; < = ? @
26 个大写字母	0x41~0x5A	0100 0001~0101 1010	65~90	大写字母: A~Z
5 个符号	0x5B~0x60	0101 1010~0110 0000	91~96	\ (反斜杠)] (右中括号) ^ (脱字符) _ (下划线) ` (左单引号)
26 个小写字母	0x61~0x7A	0110 0001~0111 1010	97~122	小写字母: a~z
4 个符号	0x7B ~ 0x7E	0111 1011 ~ 0111 1110	123 ~ 126	{ } ~
删除符号	0x7F	0111 1111	127	DEL (delete)

汉字表示法

随着计算机的发展，一些非英语国家也开始使用计算机，此时128个字符就不够用了，

- 法国就将ASCII编码扩展为8位用于表示法语，称为扩展ASCII编码。
- 汉字数量众多，为此汉字编码采用了双字节编码,为与ASCII编码兼容并区分,汉字编码双字节的最高位MSB都为1,也就是实际使用了14位来表示汉字。这就是1980年颁布的国家标准**GB2312**，也称**国标码**。
- 中文编码《信息交换用汉字编码字符集》是由中国国家标准总局1980年发布，标准号是GB 2312-1980。GB2312标准共收录6763个汉字，为了表示更多的汉字，1995年又颁布了《汉字编码扩展规范》（GBK）GBK与GB2312标准兼容，同时支持ISO/IEC10646-1和GB 13000-1的全部中、日、韩（CJK）汉字，共计20902字。GB 18030-2005《信息技术-中文编码字符集》收录了70244个汉字。
- GB2312编码理论上能表示 $2^{14}=16384$ 个编码，而实际上仅包含了7445个字符，其中6763个常用汉字、682个全角非汉字字符。

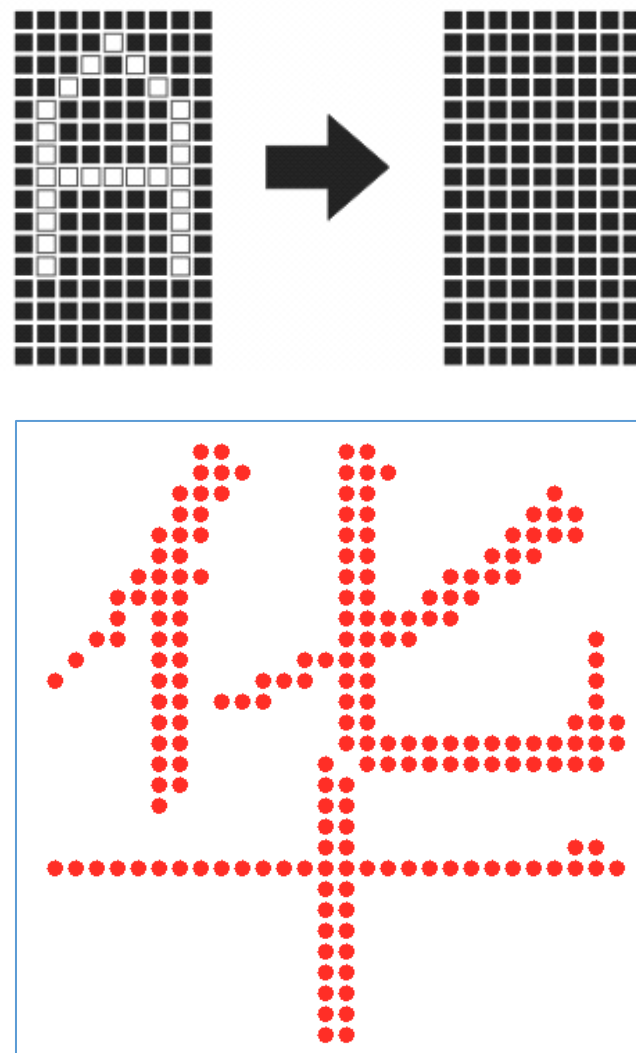
汉字表示法

- 为了检索方便，采用94*94矩阵对字符中所有汉字进行编码
 - 矩阵的每一行称为“区”，每一列称为“位”，区号和行号都从1开始编码，采用十进制表示，所有字符都在矩阵中有唯一的位置，这个位置可以用区号和位号组合表示，称为汉字的区位码，区位码“1818”是“膊”字。
 - 区位码和GB2312编码之间可以互相转换: 区位码 + A0A0H = GB2312编码。但区位码比GB2312编码更为直观简单，而且在存储汉字字形码字库时空间浪费最小，检索更方便。

区位码	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
16区	啊	阿	埃	挨	哎	唉	哀	皑	癌	蔼	矮	艾	碍	爱	隘	鞍	氨	安	俺	按
17区	薄	雹	保	堡	饱	宝	抱	报	暴	豹	鲍	爆	杯	碑	悲	卑	北	辈	背	贝
18区	病	并	玻	菠	播	拨	钵	波	博	勃	搏	铂	箔	伯	帛	舶	脖	膊	渤	泊
19区	场	尝	常	长	偿	肠	厂	敞	畅	唱	倡	超	抄	钞	朝	嘲	潮	巢	吵	炒
20区	础	储	矗	搐	触	处	揣	川	穿	椽	传	船	喘	串	疮	窗	幢	床	闯	创

字形码

- 字形码是汉字的输出码，也称字型码。最初计算机输出汉字时都采用**图形点阵**的方式。
- 在计算机中可用一组二进制数表示点阵，用0表示白点，用1表示黑点。常见汉字字形点阵有16x16、24x24、32x32、48x48，点阵越大，汉字显示和输出质量越高。
- 一个32x32点阵的汉字字形码需要使用1024位=128个字节表示，这128个字节中的信息是汉字的数字化信息，即**汉字字模**，相比机内码，其占用较大的存储空间。右图所示为32x32点阵的“华”字的字模。



本章主要内容

- 2.1 数据表示的作用
- 2.2 数值数据表示法
- 2.3 非数值数据表示法
- 2.4 数据信息的校验



2.3 数据信息的校验

- 2.4.1 码距和校验
- 2.4.2 奇偶校验
- 2.4.3 海明校验
- 2.4.4 CRC循环冗余校验

数据校验基本原理

- 受元器件质量、电路故障、噪声干扰等因素的影响，计算机在对数据进行处理、传输和存储过程中难免出现错误
- 校验码：编码中引入一定**冗余**，增加**最小码距**，使编码符合某种**规则**，当编码出现一个或多个错误时变成**非法代码**（不符合规则）

- 奇偶信息的校验

编码中1的个数的奇偶性

- 海明校验

多校验组的奇偶性，检错码为出错位

- CRC 循环冗余校验

编码能被生成多项式整除，余数循环



数据校验的主要流程

- 发送方对原始数据按照预定编码规则进行编码，生成包含冗余信息的校验码，
- 校验码经过不可靠的传输或存储后，由接收方利用解码模块解析并判断校验码是否符合预定的编码规则
- 如不符合编码规则，在传输或存储的过程中发生了错误，需要纠错或者重传。



身份证的秘密

■ 身份证编码格式

#	行政区划编码						出生年月日								顺序码			?
A	2	1	0	3	0	2	1	9	7	2	0	3	0	4	2	7	2	x
W	7	9	10	5	8	4	2	1	6	3	7	9	10	5	8	4	2	1

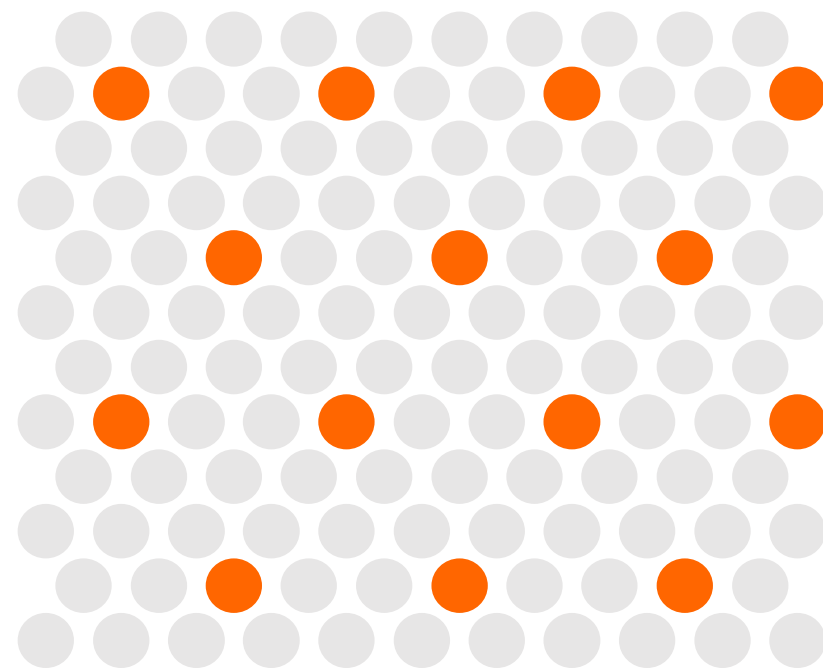
■ 各位权值： $W_i=2^{(i-1)}(\text{mod } 11)$

■ 校验位 = $\sum(A_i \times W_i) \text{ mod } 11$

余数	0	1	2	3	4	5	6	7	8	9	10
P	1	0	x	9	8	7	6	5	4	3	2

码距概念

- **码距**：任意两个合法编码间不同的二进制位数
- **最小码距**：编码集合中，任意两个码字的最小码距称为该编码集的码距。
- 校验码的目的是扩大码距，通过编码规则识别错误代码
- 码距越大，抗干扰能力、纠错能力越强，数据冗余越大，编码效率越低
- 选择码距应考虑信息出错概率和系统容错率
 - 奇偶校验 最小码距为2
 - 海明码 最小码距为3



码距与纠错性能

最小码距	检错	纠错
1	0	0
2	1	0
3	2	或 1
4	2	且 1
5	2	且 2
6	3	且 2
7	3	且 3

2.3 数据信息的校验

- 2.4.1 码距和校验
- 2.4.2 奇偶校验
- 2.4.3 海明校验
- 2.4.4 CRC循环冗余校验

奇偶校验

海明编码

CRC校验

2.4.2 奇偶校验

- 通过检测二进制代码中1的个数的奇偶性(分别对应奇校验和偶校验)进行数据校验。
- 奇校验
 - **冗余位:** 1位 校验位 P
 - **编码规则:** 是指校验码 (数据 + 校验位) 中1的个数为奇数 **最小码距: 2**
 - 0000 → 00001 (奇校验) 0001 → 00011 (偶校验)

2.4.2 奇偶校验

■ 逻辑电路产生奇偶校验位

□ 被校验信息D1D2...D8

偶校验: $P = D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8$

奇校验: $P = \overline{D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8}$

□ 接收方收到对方发送的校验码之后，生成检错码

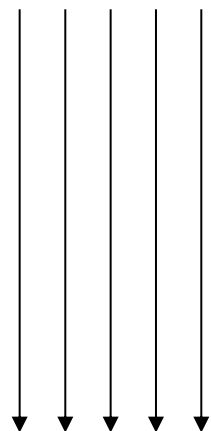
偶校验

检错码: $G = P_1 \oplus D_1 \oplus D_2 \oplus D_3 \oplus D_4 \oplus D_5 \oplus D_6 \oplus D_7 \oplus D_8$

G=1: 数据出错

奇偶校验性能

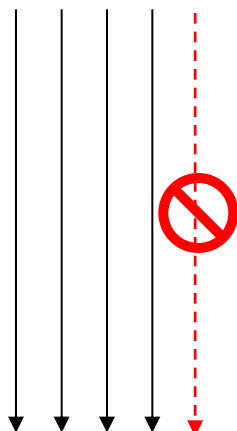
00011



00011

正确传输

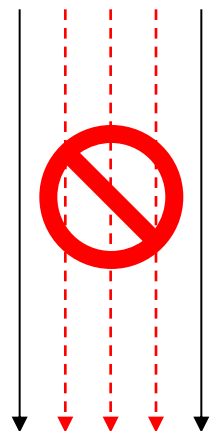
00011



00010

正常检错

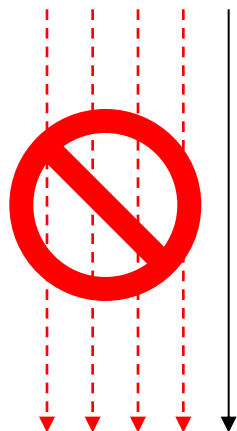
00011



01101

正确检错

00011



11101

不能检错

识别奇数位错，不能识别偶数位错；不能纠错，不保证正确，实现简单，编码效率高

二维奇偶校验

- 将原始数据信息按某种规律分成若干个校验组，每个数据位至少位于两个以上的校验组，当校验码中的某一位发生错误时，能在多个检错位中被指出，使得偶数位错误也可以被检查出，甚至还可以指出最大可能是哪位出错，从而将其纠正，这就是多重奇偶校验的原理

二维奇偶校验

■ 多重奇偶校验最典型的例子是**交叉奇偶校验**，其基本原理是将待编码的原始数据信息构造**成行列矩阵式结构**，同时进行行和列两个方向的奇偶校验。

表 2.20 交叉偶校验

	C ₆	C ₅	C ₄	C ₃	C ₂	C ₁	C ₀	P _r
R ₃	1	0	1	0	1	1	0	0
R ₂	1	1	1	0	1	1	0	1
R ₁	0	0	1	0	0	0	1	0
R ₀	1	1	0	0	1	0	0	1
P _c	1	0	1	0	1	0	1	0

5个行检错码，8个列检错码

校验性能?

- All 1-bit errors

0110100	1
1011010	0
00 0 0110	1
1110101	1
1001011	0
1000110	1

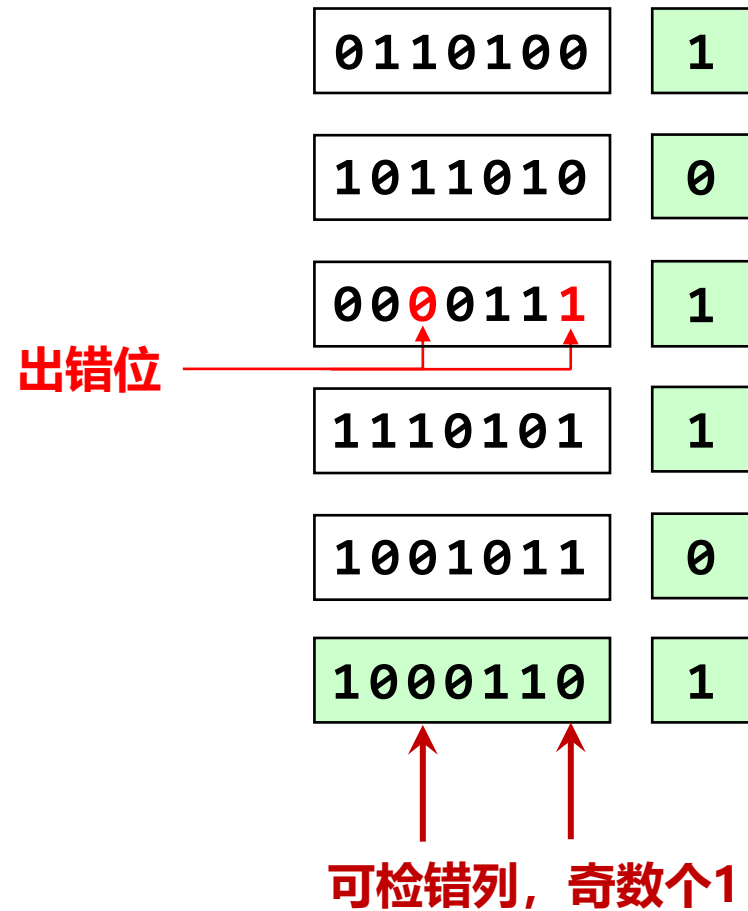
出错位

可检错行，奇数个1

可检错列，奇数个1

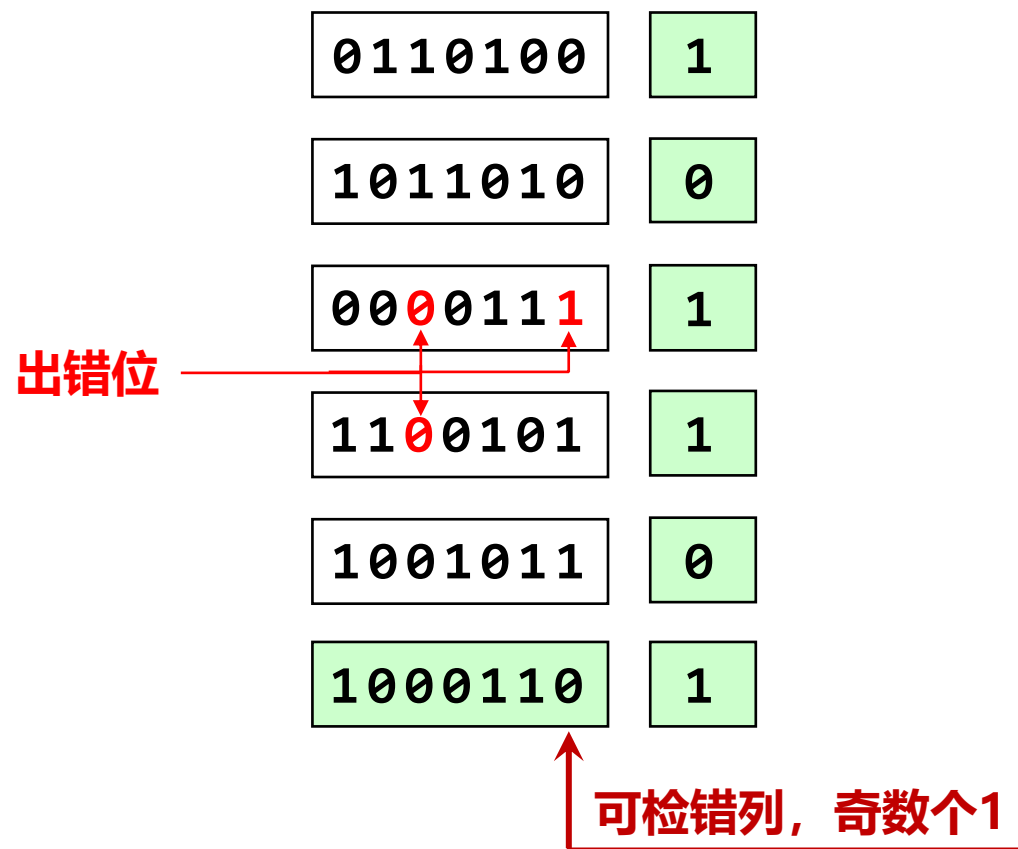
校验性能?

- All 2-bit errors



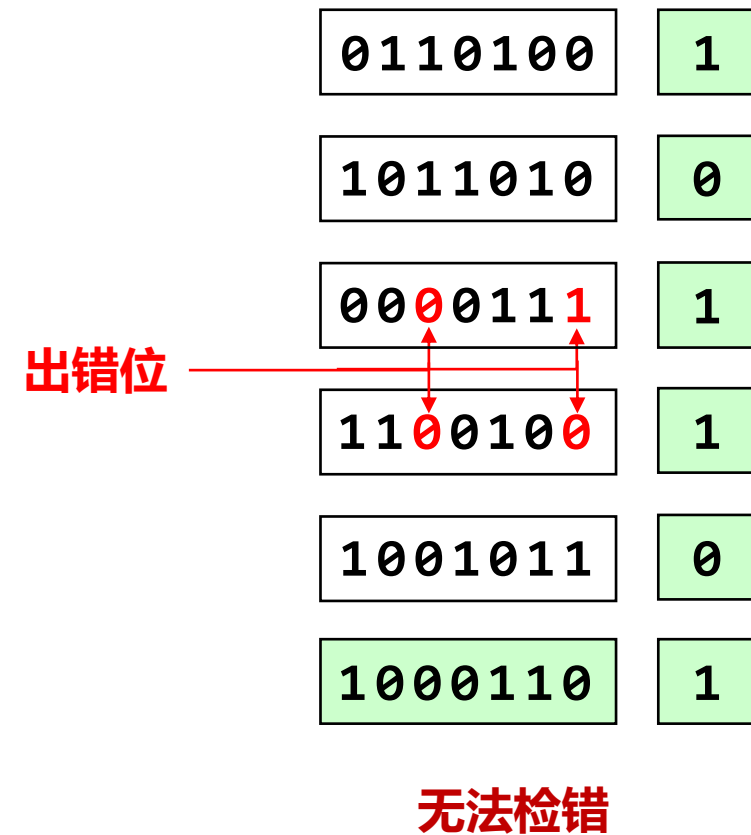
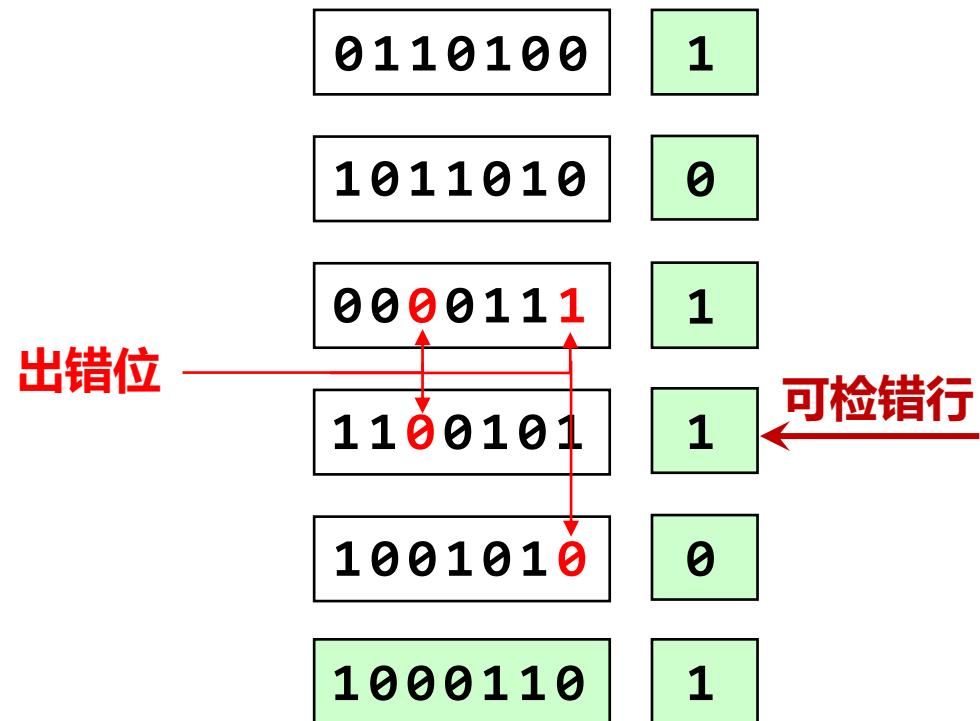
校验性能?

- All 3-bit errors



校验性能?

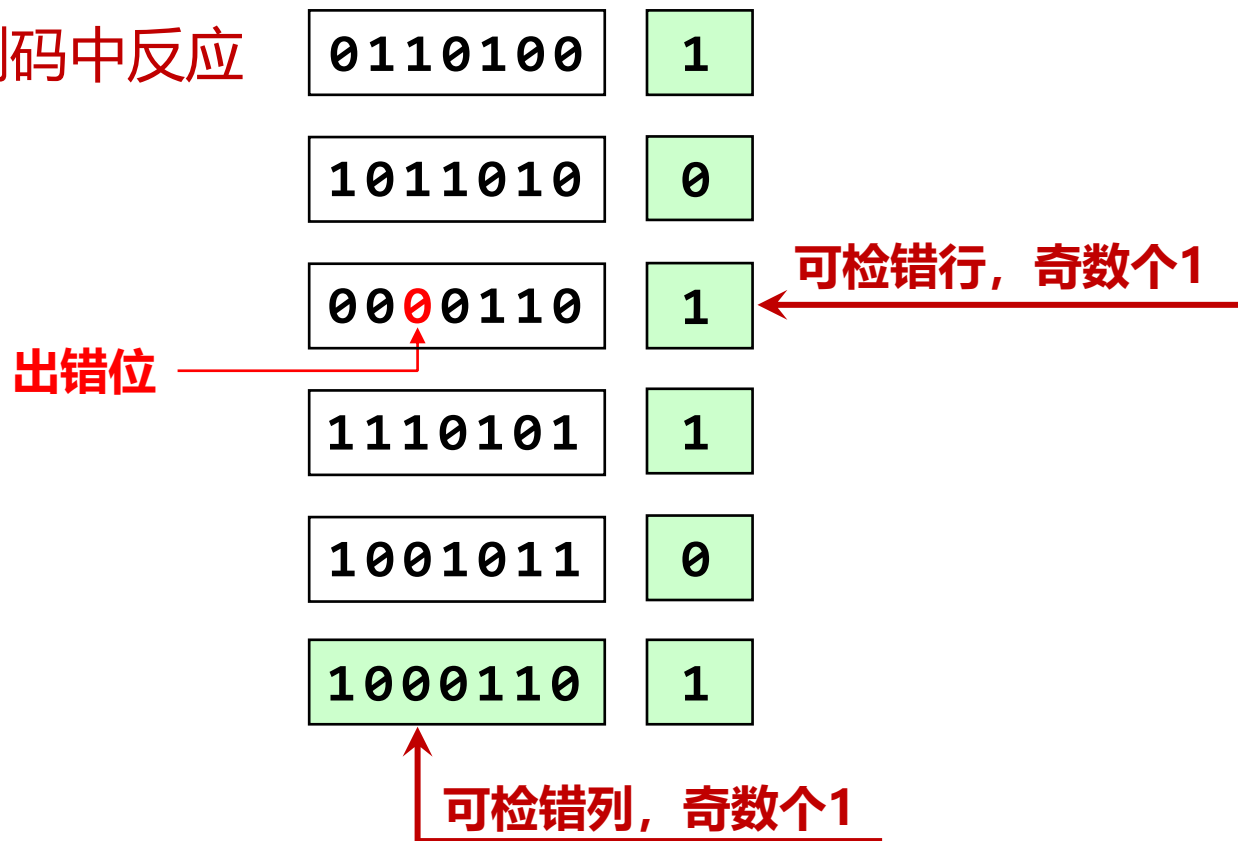
■ Most 4-bit errors



二维奇偶校验的启示

■ 多个奇偶校验组

- 一个数据位参加多个校验组
- 一个数据位发生错误可在多个检测码中反应
- 可有效提高检错能力



习题

有效信息为01011011，写出其奇校验与偶校验码。

习题答案

- 奇校验码为 010110110;
- 偶校验码为 010110111;

2.3 数据信息的校验

- 2.4.1 码距和校验
- 2.4.2 奇偶校验
- 2.4.3 海明校验
- 2.4.4 CRC循环冗余校验

奇偶校验

海明编码

CRC校验

2.3.2 海明校验 Hamming Codes

■ 奇偶校验

- 一个校验位
- 只能检错，无法纠错

■ 海明码

- 多个奇偶校验组
- 既能检错，也能纠错
- 最小码距为3



Richard Wesley Hamming
1950



- 海明校验码不仅能发现出错，而且还能指出哪一位出错。它的原理是在数据中加入几个校验位，并把数据的每一个二进制位分配在几个奇偶校验组中。
- 设有 r 位校验位，则共有0到 2^r-1 个共 2^r 个组合。若用0表示无差错，则剩余 2^r-1 个值表示有差错，并指出错在第几位。由于差错可能发生在 k 个数据位中或 r 个校验位中，因此有：

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

海明校验码的校验位数

- 根据数据位 k 与校验位 r 的关系： $2^r \geq k+r+1$ ，可以得到下表：

数据位 k	校验位 r	总位数 n
1	2	3
2~4	3	5~7
5~11	4	9~15
12~26	5	17~31
27~57	6	33~63
58~120	7	65~127

海明校验码的校验位置

- 校验位和数据位是如何排列的?

校验位排列在 2^{i-1} ($i = 1, 2, \dots$) 的位置上

例：有一个BCD码 为 $D_4D_3D_2D_1$,由此生成一个海明码
($k=4, r=3$)

7	6	5	4	3	2	1
D_4	D_3	D_2	P_3	D_1	P_2	P_1
			2^2		2^1	2^0

有一字节的信息需生成海明码 ($k=8, r=4$)

D_8	D_7	D_6	D_5	P_4	D_4	D_3	D_2	P_3	D_1	P_2	P_1
				8				4		2	1

(4,3)码分组依据

$G_3G_2G_1$	出错位	备注 (假设只有1位错)	
000	数据正常		
001	H_1 出错	$G_3G_2=0$ 表示仅仅 P_1 位出错	P_1 存放在 H_1 位置
010	H_2 出错	$G_3G_1=0$ 表示仅仅 P_2 位出错	P_2 存放在 H_2 位置
100	H_4 出错	$G_2G_1=0$ 表示仅仅 P_3 位出错	P_3 存放在 H_4 位置
011	H_3 出错	H_3 参与 $G_2 G_1$ 两校验组	D_1 存放 H_3
101	H_5 出错	H_5 参与 $G_3 G_1$ 两校验组	D_2 存放 H_5
110	H_6 出错	H_6 参与 $G_3 G_2$ 两校验组	D_3 存放 H_6
111	H_7 出错	H_7 参与 $G_3 G_2 G_1$ 三校验组	D_4 存放 H_7

(4,3)码数据位映射

$G_3G_2G_1$	出错位	备注
011	H_3 出错	H_3 参与 $G_2\ G_1$ 校验组
101	H_5 出错	H_5 参与 $G_3\ G_1$ 校验组
110	H_6 出错	H_6 参与 $G_3\ G_2$ 校验组
111	H_7 出错	H_7 参与 $G_3\ G_2\ G_1$ 校验组

H_7	H_6	H_5	H_4	H_3	H_2	H_1
D_4	D_3	D_2	P_3	D_1	P_2	P_1

■ $G_1(P_1, H_3, H_5, H_7)$

■ $G_2(P_2, H_3, H_6, H_7)$

■ $G_3(P_3, H_5, H_6, H_7)$

■ $G_1 = P_1 \oplus D_1 \oplus D_2 \oplus D_4$

■ $G_2 = P_2 \oplus D_1 \oplus D_3 \oplus D_4$

■ $G_3 = P_3 \oplus D_2 \oplus D_3 \oplus D_4$

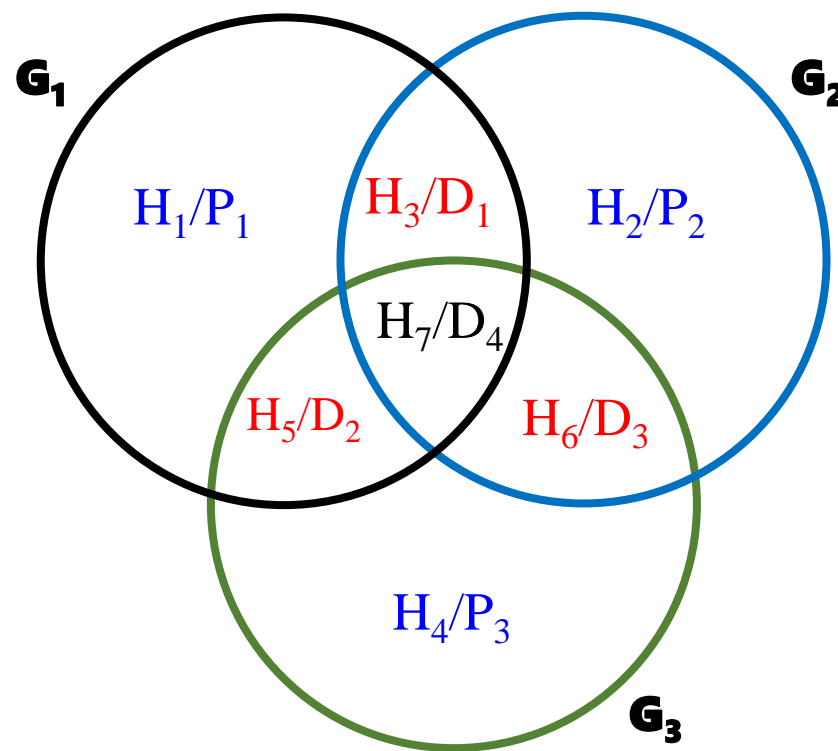
(4,3)码编码表

原始数据	校验码	十进制值	原始数据	校验码	十进制值
0000	0000000	0	1000	1001011	75
0001	0000111	7	1001	1001100	76
0010	0011001	25	1010	1010010	82
0011	0011110	30	1011	1010101	85
0100	0101010	42	1100	1100001	97
0101	0101101	45	1101	1100110	102
0110	0110011	51	1110	1111000	120
0111	0110100	52	1111	1111111	127

最小码距为3，判两位错或纠1位错

(4,3)码检错、纠错性能 最小码距3

- 当 $G_3G_2G_1 \neq 000$ ，表示海明码发生错误，在假设一位错的前提下，可以利用检错码的值找到编码中的出错位置并取反纠正错误。
- 当出现两位错时，假设图所示的H3、H5同时发生错误，由于H3参与了G1、G2组的检验，H5参与了G1、G3组的检验，G1组发生了两位错，检错位为0，而G2、G3组均发生了一位错，故对应的检错码应该是110，和H6出错的检错码重叠，此时无法区分是一位错还是两位错。造成这种问题的根本原因是码距有限，检错码的状态数不足以区分两种错误模式，因此海明码的纠错是有假设前提的。



(k, r) 码校验分组方法

■ (k, r) 海明码的校验如何分组，设检错码为 $G_r \dots G_4 G_3 G_2 G_1$

编码布局	P ₁	P ₂	D ₁	P ₃	D ₂	D ₃	D ₄	P ₄	D ₅	D ₆	D ₇	D ₈	D ₉	D ₁₀	D ₁₁	P ₅	D ₁₂	D ₁₃	D ₁₄	D ₁₅	D ₁₆
------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	----------------	-----------------	-----------------	----------------	-----------------	-----------------	-----------------	-----------------	-----------------

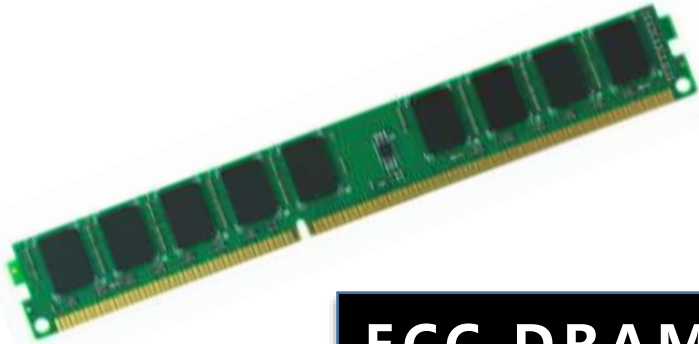
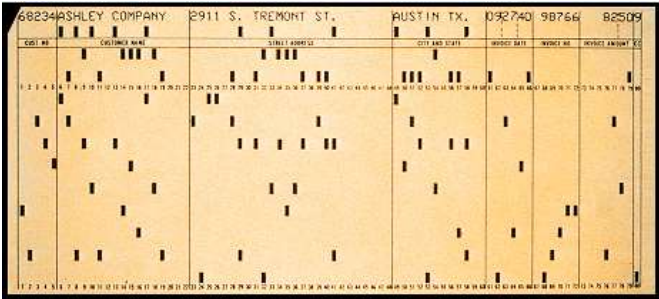
■ 根据编码规则填写海明校验组分布表

	H ₁	H ₂	H ₃	H ₄	H ₅	H ₆	H ₇	H ₈	H ₉	H ₁₀	H ₁₁	H ₁₂	H ₁₃	H ₁₄	H ₁₅	H ₁₆	H ₁₇	H ₁₈	H ₁₉	H ₂₀	H ₂₁
检错码	00001	00010	00011	00100	00101	00110	00111	01000	01001	01010	01011	01100	01101	01110	01111	10000	10001	10010	10011	10100	10101
对应位	P ₁	P ₂	D ₁	P ₃	D ₂	D ₃	D ₄	P ₄	D ₅	D ₆	D ₇					P ₅					
G ₁ 组	√		√		√		√		√		√										
G ₂ 组	P ₁ = D ₁ ⊕ D ₂ ⊕ D ₄ ⊕ D ₅ ⊕ D ₇ ⊕ ...																				
G ₃ 组				√	√	√	√														
G ₄ 组								√	√	√	√										
G ₅ 组																√					

海明码特点

编码布局	P ₁	P ₂	D ₁	P ₃	D ₂	D ₃	D ₄	P ₄	D ₅	D ₆	D ₇	D ₈	D ₉	D ₁₀	D ₁₁	P ₅	D ₁₂	D ₁₃	D ₁₄	D ₁₅	D ₁₆
k	1	2 ~ 4		5 ~ 11		12 ~ 26		27 ~ 57		58 ~ 120		...									
r	2	3		4		5		6		7		...									

- 编码效率高：数据增加一倍，校验位只增加一位
- 可纠正一位错
- 50年代发明时用于自动处理穿孔卡片的故障
- 现在普遍用于ECC DRAM芯片
- RAID2，卫星通讯



ECC DRAM

ECC (Error Checking and Correcting)

■ P48, 例题2.11

习题

设 8 位有效信息为 01101110，试写出它的海明校验码。给出过程，说明分组检测方式，并给出指错字及其逻辑表达式。如果接收方收到的有效信息变成 01101111，说明如何定位错误并纠正错误。

习题答案

1. 设 8 位有效信息为 01101110，试写出它的海明校验码。给出过程，说明分组检测方式，并给出指错字及其逻辑表达式。如果接收方收到的有效信息变成 01101111，说明如何定位错误并纠正错误。

被校验位有 8 位，设校验位有 r 位，因为 $8 + r \leq 2^r - 1$ ，所以 $r = 4$ 。

校验位分别位于第 2^0 、 2^1 、 2^2 、 2^3 位，即海明码为

D8D7D6D5P4D4D3D2P3D1P2P1。

根据表 2.22 海明码分组规则，校验位的值：

$$P1 = D7 \oplus D5 \oplus D4 \oplus D2 \oplus D1 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 1$$

$$P2 = D7 \oplus D6 \oplus D4 \oplus D3 \oplus D1 = 1 \oplus 1 \oplus 1 \oplus 1 \oplus 0 = 0$$

$$P3 = D8 \oplus D4 \oplus D3 \oplus D2 = 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$P4 = D8 \oplus D7 \oplus D6 \oplus D5 = 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

海明码 D8D7D6D5P4D4D3D2P3D1P2P1 = 011001111001

习题答案

接收方接收到的有效位只有 D1 位出差，则接收到的海明编码为 011001111101

$$G1 = P1 \oplus D7 \oplus D5 \oplus D4 \oplus D2 \oplus D1 = 1 \oplus 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$G2 = P2 \oplus D7 \oplus D6 \oplus D4 \oplus D3 \oplus D1 = 0 \oplus 1 \oplus 1 \oplus 1 \oplus 1 \oplus 1 = 1$$

$$G3 = P3 \oplus D8 \oplus D4 \oplus D3 \oplus D2 = 1 \oplus 0 \oplus 1 \oplus 1 \oplus 1 = 0$$

$$G4 = P4 \oplus D8 \oplus D7 \oplus D6 \oplus D5 = 0 \oplus 0 \oplus 1 \oplus 1 \oplus 0 = 0$$

检错码 $G4G3G2G1 = 0011 = 3$ ，如果假设只有 1 位错，则是海明码 H3 出错，也就是 D1 出错，将对应位取反即可。

2.3 数据信息的校验

- 2.4.1 码距和校验
- 2.4.2 奇偶校验
- 2.4.3 海明校验
- 2.4.4 CRC循环冗余校验

奇偶校验

海明编码

CRC校验

CRC循环冗余校验码 Cyclic redundancy check

- 编码规则：编码可被生成多项式整除

- 模2除法，余数为0（高概率正确），否则出错

- 设CRC码 N 位，其中数据位 k 位，校验位 r 位（冗余位）

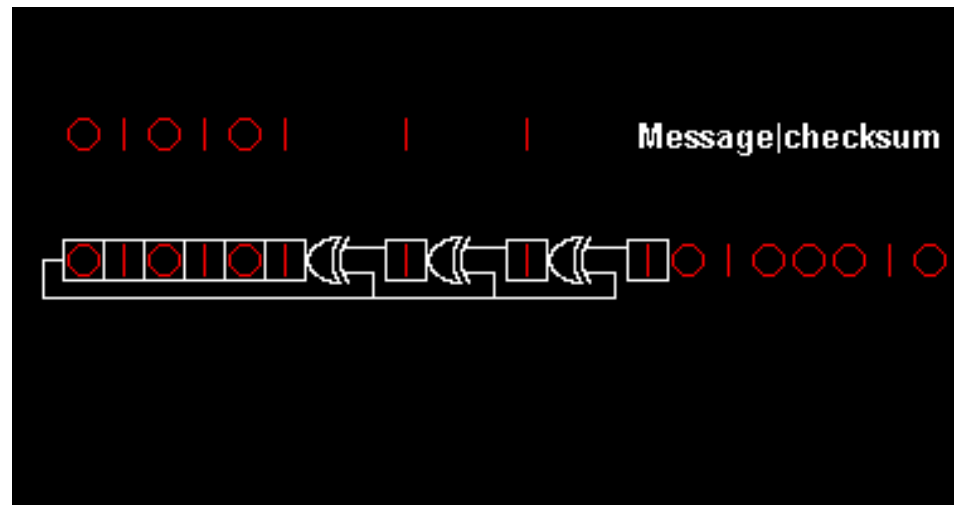
$$N = k + r \leq 2^r - 1$$

(2^r 个余数，0表示正确，N位中的每一位出错余数皆不同)

- 可检错，可纠错



W. Wesley Peterson
1961



模2运算规则

- 加法：按位加不考虑进位
- 减法：按位减不考虑借位
 - 异或运算，不考虑进位
- 乘法：部分积之和按模2加法计算
- 除法：余数首位为1，商上1，否则为0
 - $1100000 \div 1011$
 - $1100000 = 1011 * 1110 + 010$

The diagram illustrates the mod-2 division process for $1100000 \div 1011$. The divisor 1011 is shown in pink on the left. The dividend 1100000 is shown in black and red. The quotient 1110 is shown in blue at the top right, with an arrow pointing to it labeled "商". The remainder 010 is shown in yellow at the bottom right, with an arrow pointing to it labeled "余数". The steps of the division are shown as follows:

- Step 1: 1011 is subtracted from 1100 (the first four bits of the dividend), resulting in 11 .
- Step 2: The next bit 0 is brought down, making the new dividend 110 .
- Step 3: 1011 is subtracted from 110 (the first three bits of the new dividend), resulting in 1 .
- Step 4: The next bit 0 is brought down, making the new dividend 101 .
- Step 5: 1011 is subtracted from 101 (the first three bits of the new dividend), resulting in 0 .
- Step 6: The next bit 0 is brought down, making the new dividend 001 .
- Step 7: 1011 is subtracted from 001 (the first three bits of the new dividend), resulting in 0 .
- Step 8: The next bit 0 is brought down, making the new dividend 000 .
- Step 9: 1011 is subtracted from 000 (the first three bits of the new dividend), resulting in 0 .

The final result is a quotient of 1110 and a remainder of 010 .

模2除法

CRC编码规则

- 将待编码的k位有效信息位组表达为多项式M(x)

$$M(x) = b_{k-1}x^{k-1} + b_{k-2}x^{k-2} + \cdots + b_1x^1 + b_0 \quad x=2$$

- 将**数据左移 r 位**，空出r位校验位 **(冗余位)**，变成 $M(x) \cdot x^r$
- 将 $M(x) \cdot x^r$ 除以生成多项式 $G(x)$ ，商为 $Q(x)$ ，余数 $R(x)$

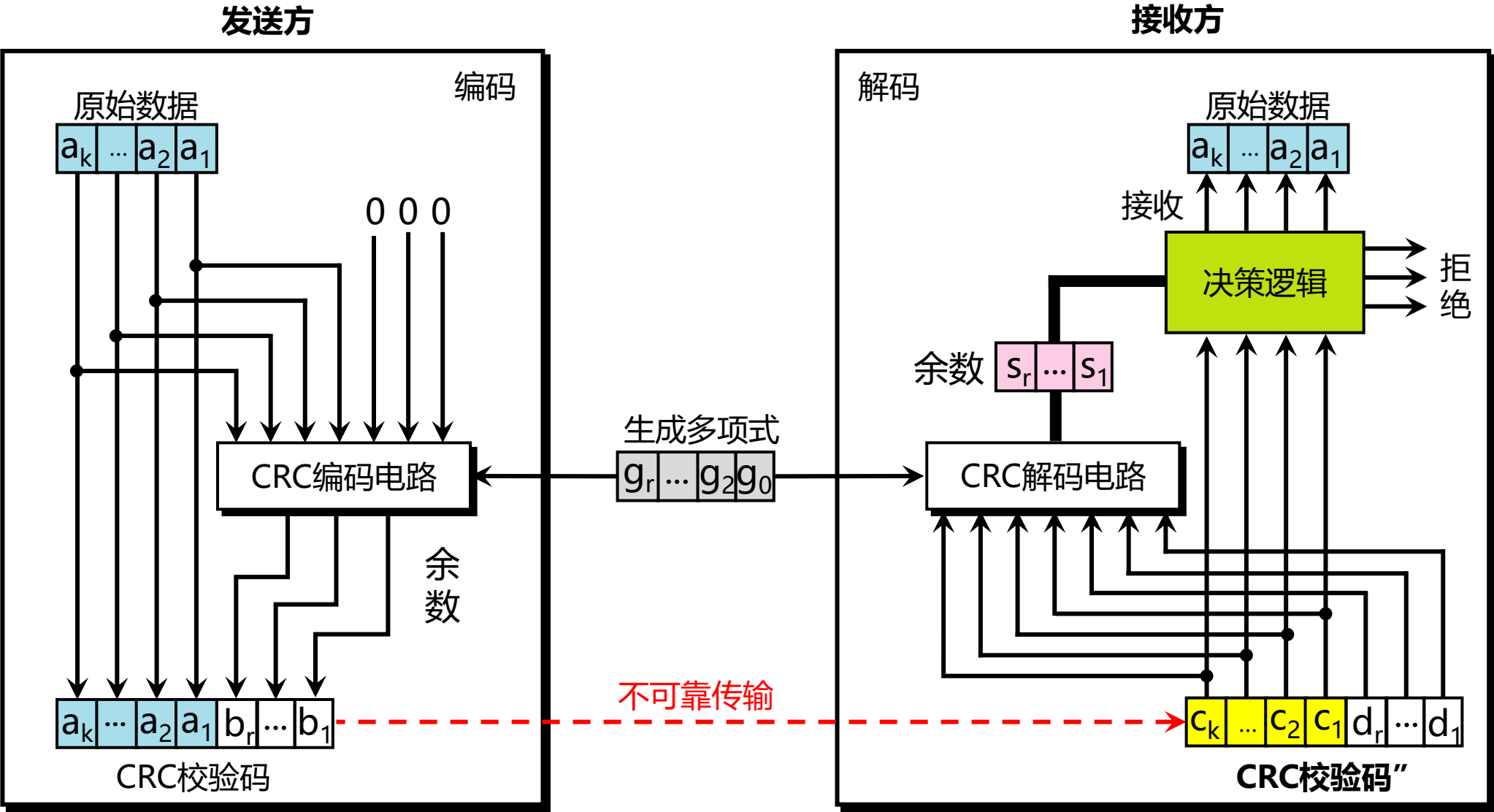
$$M(x) \cdot x^r = Q(x) \cdot G(x) + R(x)$$

- 将**余数填充在校验位**

$$M(x) \cdot x^r + R(x) = Q(x) \cdot G(x) + R(x) + R(x) = Q(x) \cdot G(x)$$

编码规则： CRC编码可被G(x)表示的编码整除

CRC 编解码过程



生成多项式 Generator polynomial

■ 生成多项式特征

- 最高位与最低为必须是1
- 任意位发生错误都应使余数不为0
- 不同位发生错误余数不同
- 余数左移一位继续作模2除，应使余数循环，**循环周期** $N=k+r$ ？

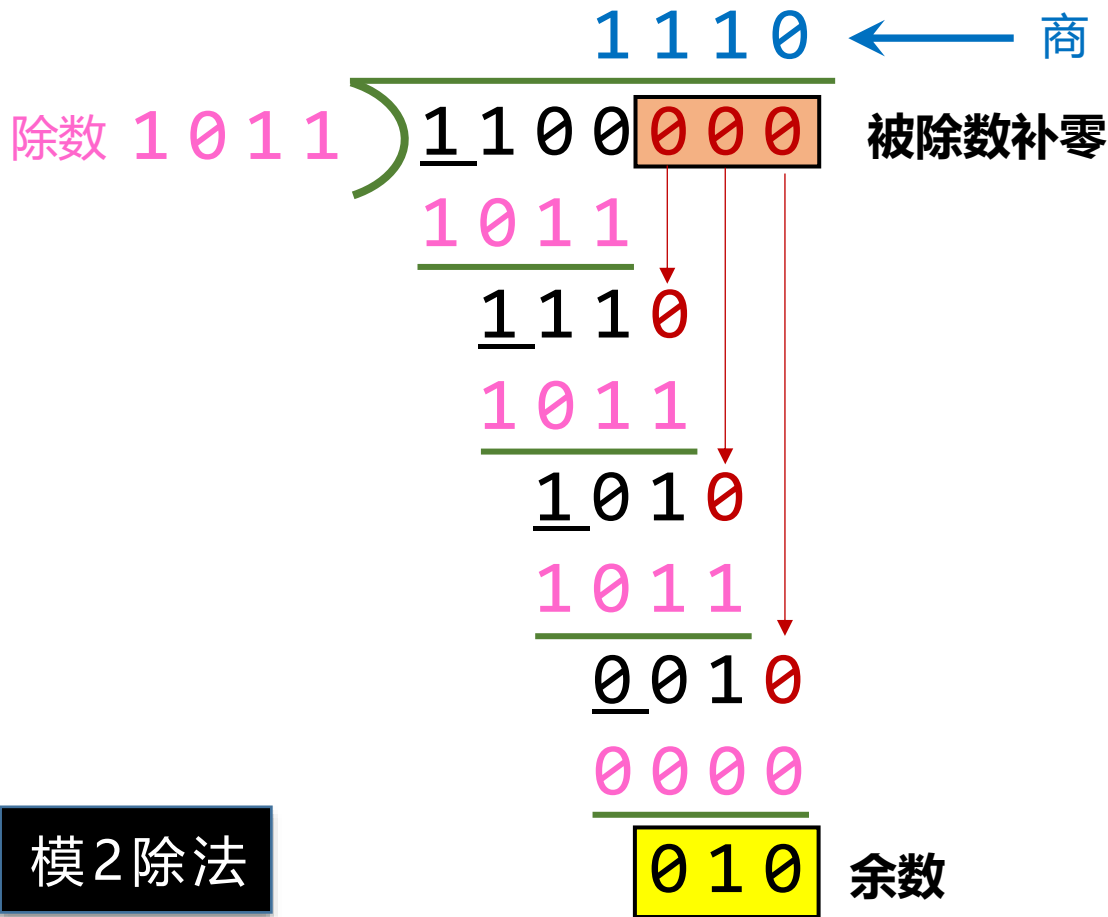
常见生成多项式与应用

CRC编码	用途	多项表达式
CRC-1	奇偶校验	$x+1$
CRC-3	GSM移动网络	$x^3+ x+ 1$
CRC-4	ITU-T G.704	$x^4+ x+ 1$
CRC-5-ITU	ITU-T G.704	$x^5+ x^4 + x^2+1$
CRC-5-EPC	二代RFID	$x^5+ x^3+1$
CRC-5-USB	USB令牌包	$x^5+ x^2+1$
CRC-6-GSM	GSM移动网络	$x^6+ x^5+ x^3+x^2+x+1$
CRC-7	MMC/SD卡	x^7+x^3+1
CRC-16-CCITT	USB、 bluetooth	$x^{16}+ x^{15}+x^2+1$
CRC-32	Ehernet, SATA MPEG-2,PKZIP,Gzip...	$x^{32}+x^{26}++x^{23}+x^{22}+x^{16}+x^{12}$ $+x^{11}+x^{10}+x^8+x^7+x^5+x^4+x^2+x+1$

编码过程 (7,4)循环码G(x)=1011

1 1 0 0

原始数据



原始数据 余数

1 1 0 0 0 1 0

校验码

(7,4)循环码 余数循环模式 G(x)=1011

全零永远
是合法编码

A ₁ ~A ₇	余数	出错位
0000000	000	无
0000001	001	7
0000010	010	6
0000100	100	5
0001000	011	4
0010000	110	3
0100000	111	2
1000000	101	1

根据不同的余数来
纠正不同的出错位

检错过程 1位错 (0001000) 2位错 (1010000)

除数 1 0 1 1

0 0 0 1 0 0 0

0 0 0 0

0 0 1 0

0 0 0 0

0 1 0 0

0 0 0 0

1 0 0 0

1 0 1 1

0 1 1

0 0 0 1 ← 商

模2除法

余数

除数 1 0 1 1

1 0 1 0 0 0 0

1 0 1 1

0 0 1 0

0 0 0 0

0 1 0 0

0 0 0 0

1 0 0 0

1 0 1 1

0 1 1

1 0 0 1 ← 商

模2除法

余数

一位错、两位错余数均不为零，但余数有重叠，判两位错，或纠1位错

CRC (N,k)码检错性能

$$r=N-k$$

- 所有小于等于r长度的**突发错**

- 通讯中常见，各出错位之间有因果关系
- 突发错长度：第一和最后一个错位之间的距离

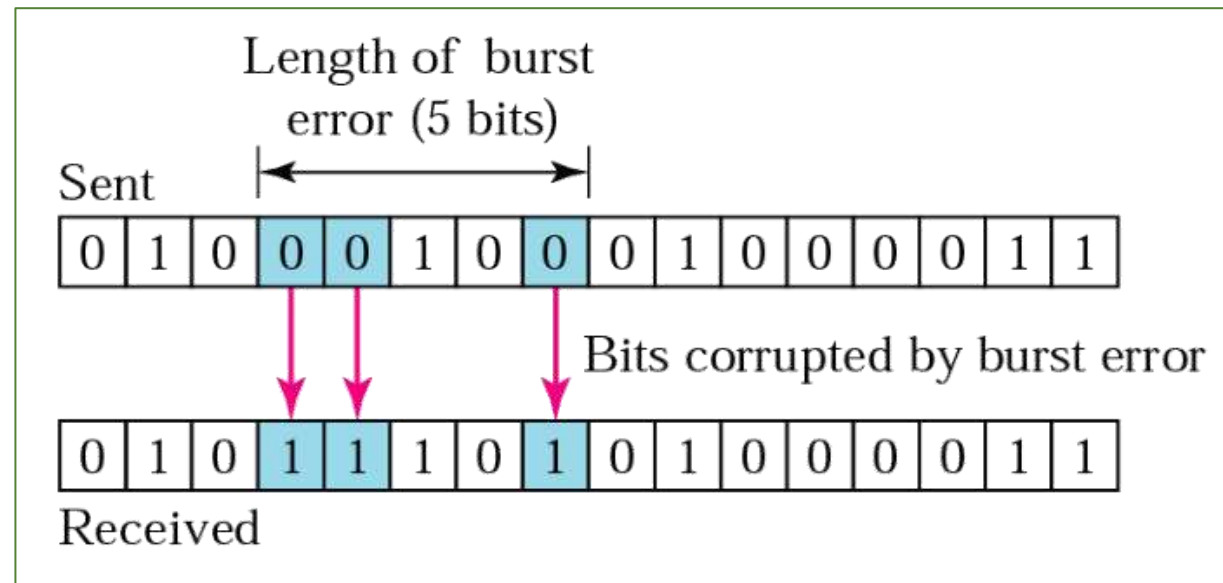
- $(1-2^{-r+1})$ 比例的r+1长度的突发错

- $(1-2^{-r})$ 比例的大于r+1长度的突发错

- 所有小于最小码距的任意位数的错误

- 如果生成多项式中1的数目为偶数，可检测所有奇数错

- 广泛运用于**通信传输**领域，**磁存储**领域



校验码总结

	偶校验	可检一位错海明码	CRC循环冗余校验码
冗余信息	$r=1$	$r>1$	$r>1$
编码规则	数位中1的个数为偶数	r个校验组中1的个数为偶数	校验码被生成多项式整除
校验码公式	$P = D_1 \oplus D_2 \dots \oplus D_k$	$P_i = H_1 \oplus H_2 \dots H_r$	$P = (M(x) * 2^r) \% G(x)$
最小码距	2	3	≥ 3
检错依据	$G = P \oplus D_1 \oplus D_2 \dots \oplus D_k = 0$	$G_r \dots G_2 G_1 = 0$	$R(x) = 0$
编码特性	实现简单，编码效率高	可纠一位错 数位较多时编码效率高	检错能力更强 数位较多时编码效率高

习题

设要采用CRC码传送数据信息 $x=1001$ ，当生成多项式为 $G(x)=1101$ 时，请写出它的循环冗余校验码。

习题

设要采用CRC码传送数据信息 $x=1001$ ，当生成多项式为 $G(x)=1101$ 时，请写出它的循环冗余校验码。

做模2除法

$$100100/1101=1111+011/1101$$

CRC校验码：1001011