计算机组成原理

第二章 运算方法和运算器

中国地质大学计算机学院
刘 超

主要内容

- 2.1 数据与文字的表示方法
- 2.2 定点加法、减法运算
- 2.3 定点乘法运算
- 2.4 定点除法运算
- 2.5 定点运算器的组成
- 2.6 浮点运算和浮点运算器

2.1数据与文字的表示方法

- 1 数值数据的表示方法：
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- 2 文字编码

1、进位记数法:

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

- 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
- Σ 表示m+k+1位的值求累加和

例子

- $(104.56)10 = 1\times10^{2} + 0\times10^{1} + 4\times10^{0} + 5\times10^{-1} + 6\times10^{-2}$
- $(0xF96)_{16} = F\times16^{2} + 9\times16^{1} + 6\times10^{0}$
- $(10010001)_{2} = 1\times2^{7} + 0\times2^{6} + 0\times2^{5} + 1\times2^{4} + 0\times2^{3} + 0\times2^{2} + 0\times2^{1} + 1\times2^{0}$

计算机数据编码需要考虑的因素：

- 数的类型(小数、整数、实数和复数）
- 数值范围
- 数值精确度
- 数值存储和处理所需的硬件代价

计算机数据编码特点

- 少量简单的基本符号表示大量复杂的信息
- 状态简单
- 电路实现简单
- 运算方便
- 硬件成本

Human VS 计算机

- 人们日常生活采用10进制
  - 天生10个手指
- 计算机采用二进制
  - 计算机采用电子开关
  - 开关仅仅包括两个状态 ON /OFF

编码特点比较

- 0123456789共10种状态，状态过多
- 运算组合状态过多
- 加法组合数= $C_{10}^{2}+10$ $=10*9/2!+10$ $=55$

八进制： $C_{8}^{2}+8=8*7/2!+8=36$

四进制： $C_{4}^{2}+4=4*3/2!+4=10$

二进制： $C_{2}^{2}+2=2*1/2!+2=3$

结论：二进制的组合状态最少

二进制编码特点

- 1）容易实现
  - 符号个数最少，“0、1”
  - 用数字电路的两个状态表示(如电压高低）
- 2）运算简单
  - 二进制数据运算规则仅三种(10进制有55种)

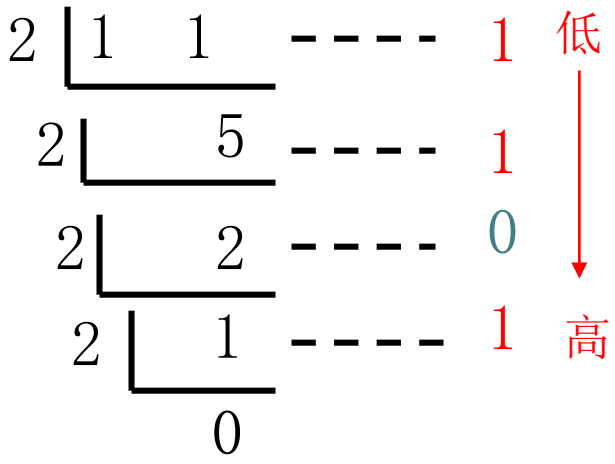
    0+1=1+0=1，1+1=0, 0+0=0
  - 一个异或门即可完成该运算
- 3）工作可靠
- 4）逻辑判断简单
  - 与二值逻辑的 真　假 两个值对应

进制转换

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

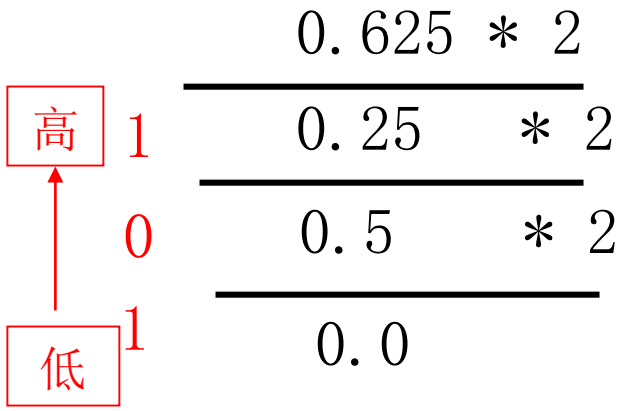
十进制转二进制

整数部分除2取余



除尽为止 1011

小数部分乘2取整



求得位数满足要求为止

二进制转十进制

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

从二进制数求其十进制的值，逐位码权累加求和

- $10010001=1\times2^7+0\times2^6+0\times2^5+1\times2^4+0\times2^3+0\times2^2+0\times2^1+1\times2^0$

二到八或十六进制转换

二到八　　从小数点向左右三位一分组

$(10\ 011\ 100.01)_2 = (\ 234.2\ )_8$

010

二到十六　从小数点向左右四位一分组

$(1001\ 1100\ .\ 01)_2 = (\ 9C\ .\ 4\ )_{16}$

0100

说明：整数部分不足位数对转换无影响，小数部分不足位数要补零凑足，否则出错。

二进制数据计量单位

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

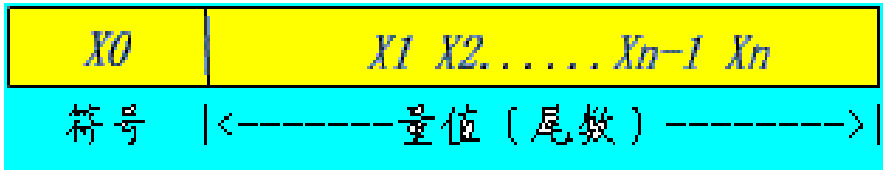
- 30GB=???Byte　1Mbits=???
- 30 GB drive = 30×10^{9} = 28×2^{30} bytes　1 Mbit/s = 10^{6} bps

2.1数据与文字的表示方法

- **1 数值数据的表示方法：**
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- **2 文字编码**

定点格式和浮点格式

- **定点格式**容许的数值范围有限，但要求的处理硬件比较简单。
- **浮点格式**容许的数值范围很大，但要求的处理硬件比较复杂。
- **定点表示**：约定机器中所有数据的小数点位置是固定不变的。由于约定在固定的位置，小数点就不再使用记号“.”来表示。通常将数据表示成**纯小数(定点小数)**或**纯整数(定点整数)**。
- 定点数 $x = x_0 x_1 x_2 \cdots x_n$ 在定点机中表示如下(x_0:符号位，0代表正号，1代表负号):

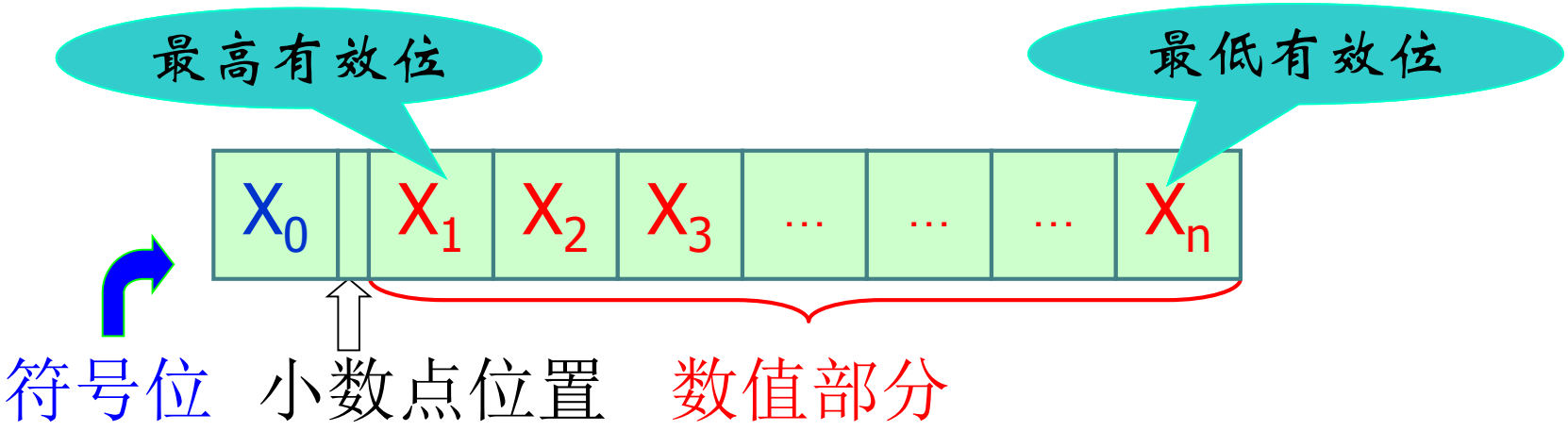


纯小数(定点小数)

- 数值表示：$x = x_0.x_1x_2\ldots x_n \quad x_i=\{0,1\},\ 1\le i\le n$

$$x_1 2^{-1} + \ldots + x_{n-1}2^{-n+1} + x_n 2^{-n}$$

- 表示范围：$0 \le |x| \le 1-2^{-n}$ **(2.1)**



- 例如：x=0.10101　　其数值=$2^{-1}+2^{-3}+2^{-5}$=21/32

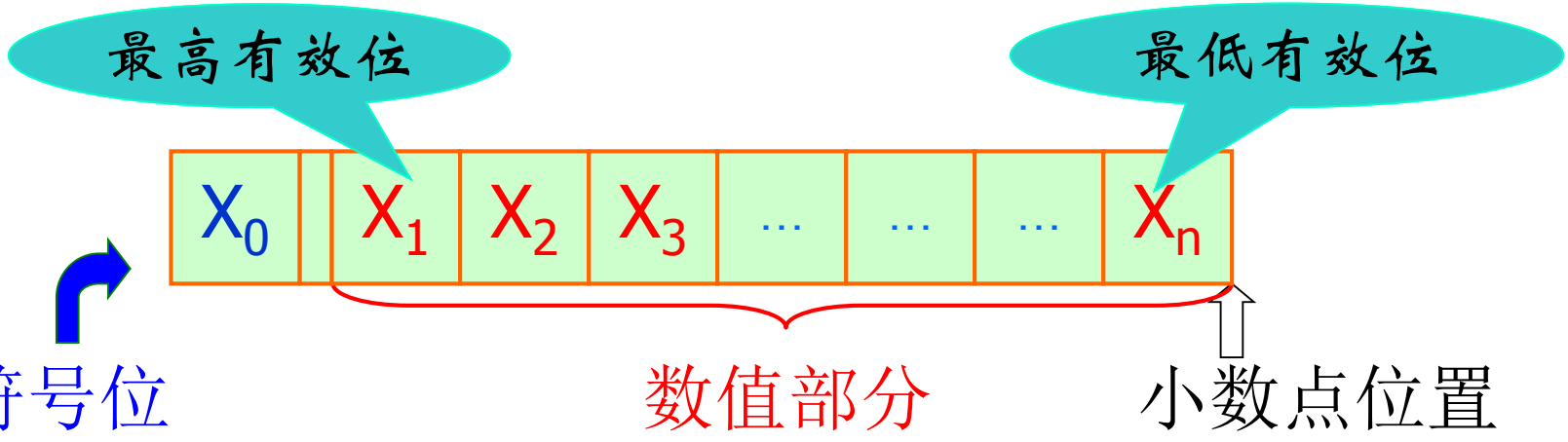
纯整数(定点整数):

- 数值表示：$x = x_0x_1x_2\ldots x_n \quad x_i=\{0,1\},\ 1\le i\le n$

$$x_0 2^{n-1} + x_1 2^{n-2} + \ldots + x_{n-1} 2^1 + x_n$$

表示范围:

$$0 \le |x| \le 2^n - 1 \qquad \textbf{(2.2)}$$



例如: x=010101　　其数值=$2^4+2^2+2^0$=21

浮点数如何表示?

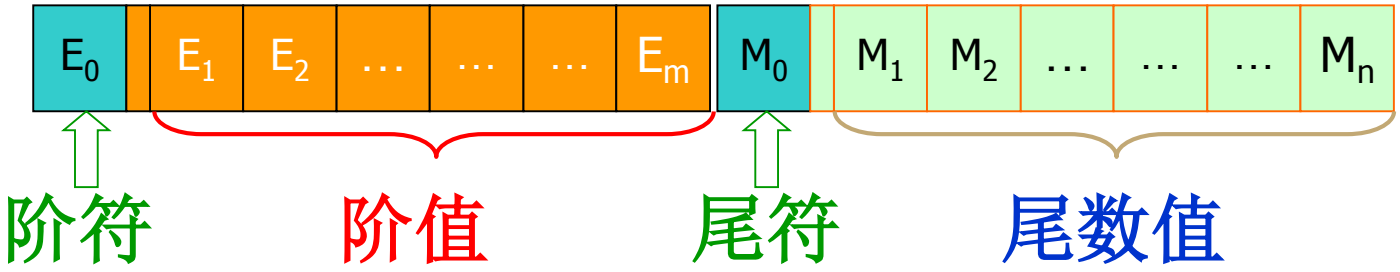
- 参与运算的数据通常既包括整数也包括小数部分。
- 如何表示？如何运算？？
  - 将数据按照一定比例因子缩小成定点小数或扩大成定点整数进行表示和运算
  - 运算完毕后再根据比例因子还原成实际数值
- 计算机中有专门的**浮点运算器件**

浮点数如何表示

- 电子的质量 $9\times10^{-28}g$
- 太阳的质量$2\times10^{33}g=0.2\times10^{34}$
- 科学记数法表示十进制数：$N=10^{E}\times M$ **(2.3)**
- 任意进制数：$N=R^{e}\times m$ **(2.4)**
  - m称为尾数，是一个纯小数，e是比例因子的阶数，称为浮点数的指数，是一个整数，R为基数
- 把一个数的数的范围和有效数字在计算机的一个存储单元中分别表示，这种把**数的范围**和**精度**分别表示的方法，数的小数点位置随比例因子的不同而在一定范围内自由浮动。
- 对于二进制机器，规定公式2.4中R为2、8，16

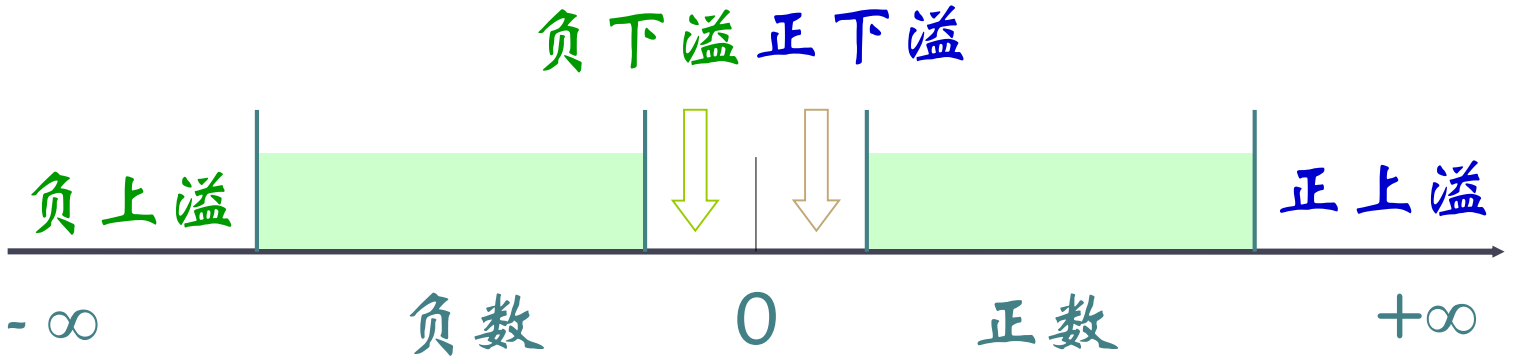
浮点数的表示

- 一个机器浮点数由**阶码**和**尾数**及其**符号位**组成。
  - **尾数**：用定点小数表示，给出有效数字的位数**决定了浮点数的表示精度**；
  - **阶码**：用整数形式表示，指明小数点在数据中的位置，**决定了浮点数的表示范围**。
- $N=R^{e}\times m=2^{E}\times M=2^{\pm e}\times(\pm m)$



- 机器字长一定时，阶码越长，表示范围越大，精度越低
- 浮点数表示范围比定点数大，精度高

浮点数的表示范围

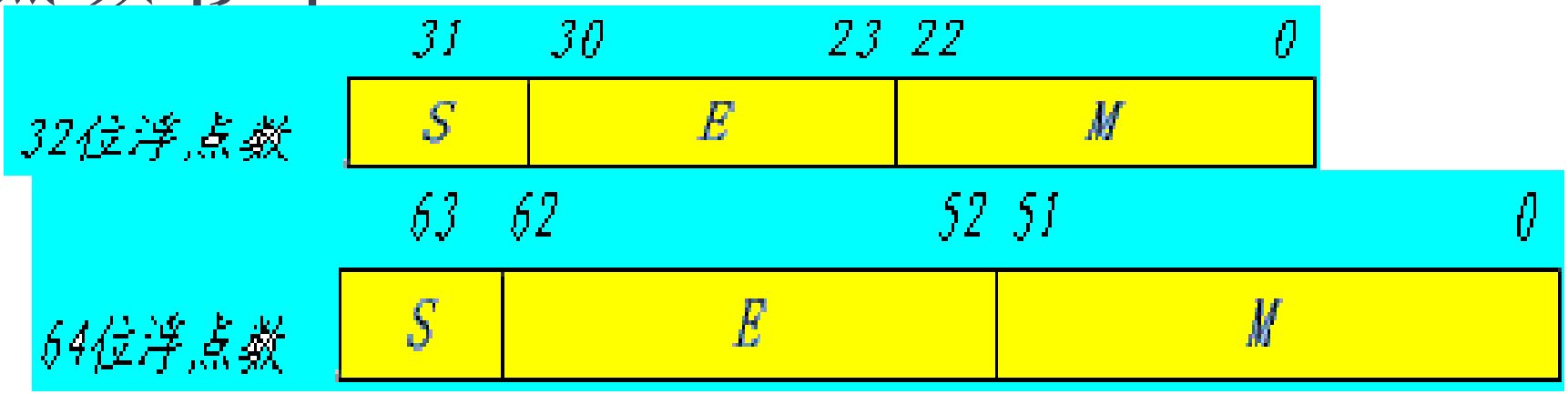


- $N=2^{E}\times M$
- $|N|\to\infty$ 产生正上溢或者负上溢
- $|N|\to 0$ 产生正下溢或者负下溢

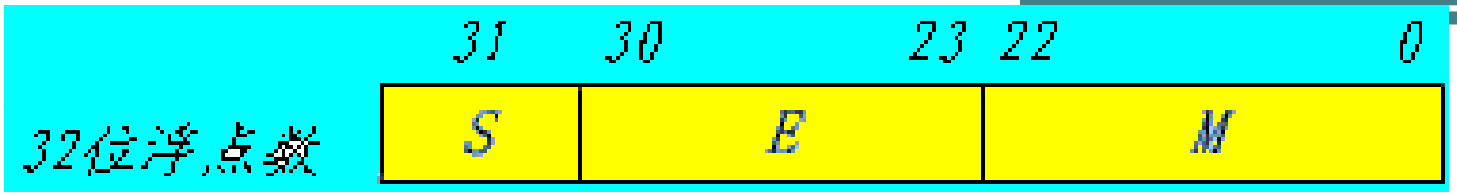
浮点数的规格化问题(重难点)

- 为提高数据的表示精度，当尾数的值不为0时，其绝对值应≥0.5，即尾数域的最高有效位应为1，否则以修改阶码同时左右移小数点的办法，使其变成这一表示形式，这称为浮点数的**规格化表示**。
  - $0.05*10^{1}$, $50*10^{-2}$, $5*10^{-1}$
  - $0.01*2^{1}$, $1*2^{-2}$, $1*2^{-1}$
- 尾数最高有效位为1的数称为**规格化数**。
  - 在尾数中表示最多的有效数据位
  - 数据表示的唯一性
- 两种规格化数 1.XXXXX , 0.1XXXXX
- 机器零：全部为0，特殊的数据编码

浮点数标准 IEEE754



- 构成：阶码E，尾数M，符号位S
- $N = (-1)^S * M * 2^E$
- 对于32位的浮点数：
  - *S*：**浮点数的符号位**，1 位，0表示正数，1表示负数。
  - *M*：尾数，23位，用小数表示，小数点放在尾数域的最前面。
  - *E*：**阶码，8 位阶符采用隐含方式，即采用移码方式来表示正负指数。将浮点数的指数真值*e* 变成阶码 *E* 时，应将指数 *e* 加上一个固定的偏移值**127(01111111)



- 一个规格化的32位浮点数 x 的真值可表示为

$$x=(-1)^s\times(1.M)\times 2^{E-127} \quad e=E-127 \qquad (2.5)$$

一个规格化的64位浮点数 x 的真值为

$$x=(-1)^s\times(1.M)\times 2^{E-1023} \quad e=E-1023 \qquad (2.6)$$

- 当浮点数的尾数为0，不论其阶码为何值，或者当阶码的值遇到比它能表示的最小值还小时，不管其尾数为何值，计算机都把该浮点数看成零值，称为**机器零**。（**除去E取值全零和全1**）

E=全0,M=全0,S=0 || 1 ,正零，负零。

E=全1（255），M=全0，S=0 || 1，+∞，-∞

E=全1，M不等于0，NaN。

e=[-126，+127]

32位浮点数绝对值取值范围10^{-38}--10^{38}

IEEE754 规格化浮点数表示范围

格式	最小值	最大值
单精度	$E_{min}=1$, M=0, $1.0\times2^{1-127}=2^{-126}$	$E_{max}=254$, f=1.1111..., 1.111...1$\times2^{254-127}$ $=2^{127}\times(2-2^{-23})$
双精度	$E_{min}=1$, M=0, $1.0\times2^{1-1023}=2^{-1022}$	$E_{max}=2046$, f=1.1111...,1.111...1$\times2^{2046-1023}$ $=2^{1023}\times(2-2^{-52})$

一个奇怪的程序

```
main()
{
  double a,b,c;   int d;
  b=3.3;   c=1.1;
  a=b/c;
  d=b/c;
  printf("%f,%d",a,d);
}
```

3.000000,2

??????????

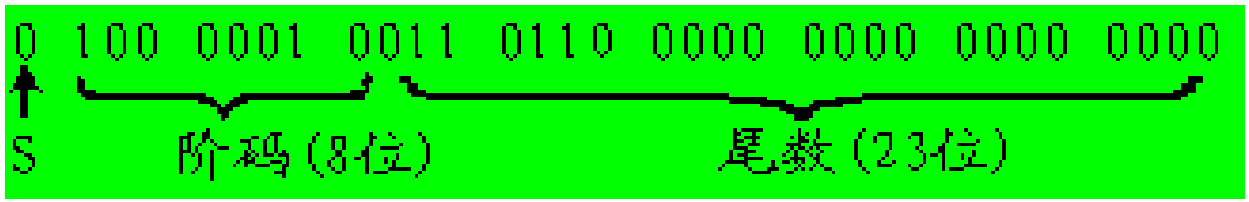
二进制存储

浮点数不是精确数

例 1

若浮点数 x 的二进制存储格式为$(41360000)_{16}$，求其32位浮点数的十进制值。(IEEE754标准)

[解:] 将十六进制数展开后，可得二进制数格式为



指数e＝阶码－127＝10000010－01111111

＝00000011=$(3)_{10}$

1.M＝1.011 0110 0000 0000 0000 0000

＝1.011011

于是有 $x=(-1)^s\times 1.M\times 2^e$

$=+(1.011011)\times 2^3=+1011.011=(11.375)_{10}$

例2

$(-1)^s \times 1.M \times 2^{E-127}$

将十进制数20.59375转换成32位浮点数的二进制格式来存储。(IEEE754标准)

[解:] 首先分别将整数和分数部分转换成二进制数：

$$20.59375 = 10100.10011$$

然后移动小数点，使其在第1，2位之间

$10100.10011 = 1.010010011 \times 2^4$ $e = 4$

于是得到：$S = 0$， $E = 4 + 127 = 131$，

$M = 010010011$

最后得到32位浮点数的二进制存储格式为：

0100 0001 1010 0100 1100 0000 0000 0000 =

$(41A4C000)_{16}$

2.1数据与文字的表示方法

- **1 数值数据的表示方法：**
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- **2 文字编码**

十进制数串的表示方法

目前,大多数通用性较强的计算机都能直接处理十进制形式表示的数据。十进制数串在计算机内主要有两种表示形式:

a.字符串形式

字符串形式：一个字节存放一个十进制的数位或符号位。为了指明这样一个数，需要给出该数在主存中的起始地址和位数(串的长度)。

b.压缩的十进制数串形式

压缩的十进制数串形式：一个字节存放两个十进制的数位。它比前一种形式节省存储空间，又便于直接完成十进制数的算术运算，是广泛采用的较为理想的方法。

用压缩的十进制数串表示一个数,要占用主存连续的多个字节。每个数位占用半个字节(即4个二进制位),其值可用二—十编码(BCD码)或数字符的ASCII码的低4位表示。**符号位**也占半个字节并放在最低数字位之后,其值选用四位编码中的六种冗余状态中的有关值,如用**12(c)表示正号**，用**13(d)表示负号**。在这种表示中,规定数位加符号位之和必须为偶数,当和不为偶数时,应在最高数字位之前补一个0。此时,表示一个数要占用该偶数值的一半那么多个字节,例如 +123 和−12 分别被表示成:

1 2 3 $C_{(+123)}$　　0 1 2 $D_{(-12)}$

在上述表示中,每一个小框内给出一个数值位或符号位的编码值(用十六进制形式给出),符号位在数字位之后。 前两个小框占一个字节，后两个小框占一个字节。

与第一种表示形式类似,要指明一个压缩的十进制数串,也得给出它在主存中的首地址和数字位个数(不含符号位),又称位长,位长为0的数其值为0。十进制数串表示法的优点是位长可变,许多机器中规定该长度从0到31,有的甚至更长。

2.1数据与文字的表示方法

- **1 数值数据的表示方法：**
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- **2 文字编码**

自定义数据表示

在传统的计算机体系结构中，用指令本身来说明操作数据的类型。如**定点加法**表示操作数是纯小数或纯整数；**浮点加法**表示操作数是浮点数；**十进制加法**表示操作数是BCD数。由于操作数据类型不同，要设三种不同的指令(操作码)来加以区分。

自定义数据表示则用数据本身来说明数据类型。表示形式有两种，即**标志符数据表示**(B5000)和**描述符数据表示**(B-67001)。

标志符数据表示

- 标志符数据表示要求对每一个数据都附加标志符，其格式如下：

  标识符　　数据

  - 其中标志符指明后面的数据所具有的类型，如整数、浮点数、BCD数、字符串等。
- 标志符数据表示的优点：
  - 1）简化指令系统和程序设计；2）简化编译程序，3）由硬件自动变换数据类型；4）便于程序调试和查错。
- 缺点：
  - 1）使数据区域占用的存储空间增加；2）微观上使指令执行的速度减慢。

描述符数据表示

- 描述符数据表示主要用来描述多维结构的数据类型，如向量、矩阵、记录等。其格式为：
  - 描述符标志位　特征标记　数据块长度　数据块起始地址
- 描述符标志位部分指明这是一个数据描述符；特征标记部分指明数据的各种特征；长度部分指明数组中元素个数；起始地址部分指明数据块的首地址。
- 标志符与描述符表示的区别是：

(1)标志符与每个数据相连，二者合起来存放在一个存储单元，而描述符要和数据分开存放。

(2)描述符表示中，先访问描述符，后访问数据，至少增加一次访存。

(3)描述符是程序的一部分，而不是数据的一部分

作业

- 3、 规格化数据表示
- 4（1）、IEEE754标准

2.1数据与文字的表示方法

- **1 数值数据的表示方法：**
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- **2 文字编码**

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符号数的**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

- **定点小数**原码表示是

$$[x]_{原}=\begin{cases} x & 1>x\geq 0 \\ 1-x=1+|x| & 0\geq x>-1 \end{cases} \qquad (2.7)$$

式中$[x]_{原}$是机器数，x是真值。

- 例如，$x=+0.1001$,则$[x]_{原}=0.1001$

$x=-0.1001$,则$[x]_{原}=1.1001$

- 对于0，原码有“+0”、“-0”之分，有两种形式：

$[+0]_{原}=0.000\ldots0$

$[-0]_{原}=1.000\ldots0$

原码表示法

- **定点整数**的原码表示是

$$[x]_{原}=\begin{cases} x & 2^n > x \geq 0 \\ 2^n - x = 2^n + |x| & 0 \geq x > -2^n \end{cases} \quad (2.8)$$

- 问题：01011001+11001101=???
- 原码运算 符号相异时必须作减法。

If signs are different, sign of result will be sign of larger operand.

原码运算的缺点

- 1）Arithmetic circuit complicated.
- 2） Also, two zeros. What would two 0s mean for programming?
- 总结：加法运算复杂。当两数相加时，如果是同号则数值相加；如果是异号，则要进行减法。而在进行减法时还要比较绝对值的大小，然后大数减去小数，最后还要给结果选择符号。
- 为了解决这些矛盾，人们找到了补码表示法。

补码表示法

- 负数用补码表示时，可以把减法转化为加法。在计算机中实现起来比较方便。
- **定点小数补码表示**的定义是

$$[x]_{补}=\begin{cases} x & 1>x\geq 0 \\ 2+x=2-|x| & 0\geq x\geq -1 \end{cases} \pmod 2 \quad (2.9)$$

- 例如：

$x=+0.1011$,则$[x]_{补}=0.1011$

$x=-0.1011$, 则$[x]_{补}=10+x=10.0000-0.1011=1.0101$

- 0的补码表示只有一种形式：$[+0]_{补}=[-0]_{补}=0.0000 \pmod 2$

补码表示法

- 定点整数补码表示的定义是，

$$[x]_{补}=\begin{cases} x & 2^n > x \geq 0 \\ 2^{n+1}- x = 2^{n+1}-|x| & 0 \geq x \geq -2^n \end{cases} \pmod{2^{n+1}} \quad (2.10)$$

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LSB加1。符号位为1

- $[-1010\,1010]_{补}$
  $=1\ 0101\,0101+"1"$
  $=1\ 0101\,0110$
- $[-0.010101]_{补}=1.101011$

补码优缺点

- 1）采用补码表示法进行减法运算比原码方便。
  因为不论数是正还是负，机器总是做加法，减法运算可变成加法运算。
- 2）但是根据补码定义，求负数的补码要从2减去｜x|。为了用加法代替减法，结果还得在求补码时作一次减法，这显然是不方便的。
- 如何解决？
  反码表示法可以解决负数的求补问题。

反码表示法

- 所谓反码，就是二进制的各位数码取反。
  - 若$x_i=1$，则$\overline{x}_i=0$；若$x_i=0$，则$\overline{x}_i=1$。
  - 数值上面的一横表示反码的意思。在计算机中用触发器寄存数码，若触发器Q端输出表示原码，则其Q端输出就是反码。因此反码是容易得到的。
- 符号位表示方法与原码相同。

反码表示法

- 定点小数反码表示的定义为

$$[x]_{反} = \begin{cases} x & 1 > x \geq 0 \\ (2-2^{-n})+x & 0 \geq x > -1 \end{cases} \quad (2.11)$$

其中n代表数的位数。

- 反码0的表示：

  $[+0]_{反}=0.000\ldots0$

  $[-0]_{反}=1.111\ldots1$

- 以2为基数的反码又称为1的补码。

定点小数反码公式证明

- 定点小数
- -1<x<=0时
  - 假设 $x=-0.\overline{x}_1\overline{x}_2\ldots\overline{x}_n$
  - 假$[x]_{反}=1.x_1x_2\ldots x_n$
  - $[x]_{反}+|x|=1.11\ldots1$
  - $=1.11\ldots1+0.00\ldots1-0.00\ldots1$
  - $=10.00\ldots0-0.00\ldots1$
  - $=2-2^{-n}$
- $[x]_{反}=2-2^{-n}-|x|=2-2^{-n}+x$

反码表示法

- 定点整数,反码表示的定义为

$$[x]_{反}=\begin{cases} x & 2^n > x \geq 0 \\ (2^{n+1}-1)+x & 0 \geq x > -2^n \end{cases} \qquad (2.13)$$

定点整数反码公式证明

- 定点整数
- $-2^n < x <= 0$时
  - 假设　$x = -\overline{x}_1\overline{x}_2\ldots\overline{x}_n$
  - 假设 $[x]_{反} = 1x_1x_2\ldots x_n$
  - $[x]_{反} + |x| = 111\ldots1$

    $= 111\ldots1 + 000\ldots1 - 000\ldots1$

    $= 1000\ldots0 - 000\ldots1$

    $= 2^{n+1} - 1$
- $[x]_{反} = 2^{n+1} - 1 - |x| = 2^{n+1} - 1 + x$

反码VS补码

- 比较反码与补码的公式，对于定点小数，

$$[x]_{反}=(2-2^{-n})+x$$

$$[x]_{补}=2+x$$

可得到，$[x]_{补}=[x]_{反}+2^{-n}$ (2.12)

- 对于定点整数，

$$[x]_{反}=2^{n+1}-1+x$$

$$[x]_{补}=2^{n+1}+x$$

可得到，$[x]_{补}=(2^{n+1}-1+x)+1=[x]_{反}+1$（2.13）

- 这就是通过反码求补码的重要公式。由公式可知，

若要一个负数变补码,其方法是符号位置1,其余各位0变1,1变0,然后在最末位(2^{-n})上加1。

移码表示法

- 对定点整数,移码的定义是

$$[x]_{移}=2^n+x \qquad 2^n>x\geq -2^n \qquad (2.14)$$

- 保持数据原有大小顺序，便于进行比较操作。
- 通常仅用于表示整数，**表示浮点数的阶码**。
- 例子：

  当正数 $x=+10101$ 时,$[x]_{移}=110101$ ；

  当负数 $x=-10101$ 时,$[x]_{移}=2^5+x=2^5-10101=001011$。

- 移码与补码的关系？
  - 与补码的数据位相同。
  - 移码中符号位 x_0表示的规律与原码/补码/反码相反。

4种编码对照

Binary	原码	反码	补码	移码
000	+0	+0	0	-4
001	+1	+1	+1	-3
010	+2	+2	+2	-2
011	+3	+3	+3	-1
100	-0	-3	-4	0
101	-1	-2	-3	+1
110	-2	-1	-2	+2
111	-3	-0	-1	+3

Number Stored

Number Represented

定点小数机器码表示范围

- $X_0.X_1X_2X_3...X_{n-1}X_n$
- n+1位定点数，数据位n位
- 原码，反码表示区间一致
  - $[2^{-n}-1, 1-2^{-n}]$
  - $(-1,1)$
- 补码
  - $[-1, 1-2^{-n}]$
  - $[-1,1)$ $2+x$

定点整数数机器码表示范围

- $X_0X_1X_2X_3...X_{n-1}X_n$
- n+1位定点数，数据位n位
- 原码，反码表示区间一致
  - $[1-2^n, 2^n-1]$
  - $(-2^n, 2^n)$
- 补码，移码
  - $[-2^n, 2^n-1]$
  - $[-2^n, 2^n)$ 2^n+x

几种编码的应用

- 移码主要用于表示浮点数的阶码
- 补码加减法运算方便，得到了广泛的应用。
- 目前计算机中广泛采用补码表示方法。
- 少数机器采用原码进行存储和传送，运算的时候改用补码。
- 还有些机器在做加减法时用补码运算，在做乘除法时用原码运算。

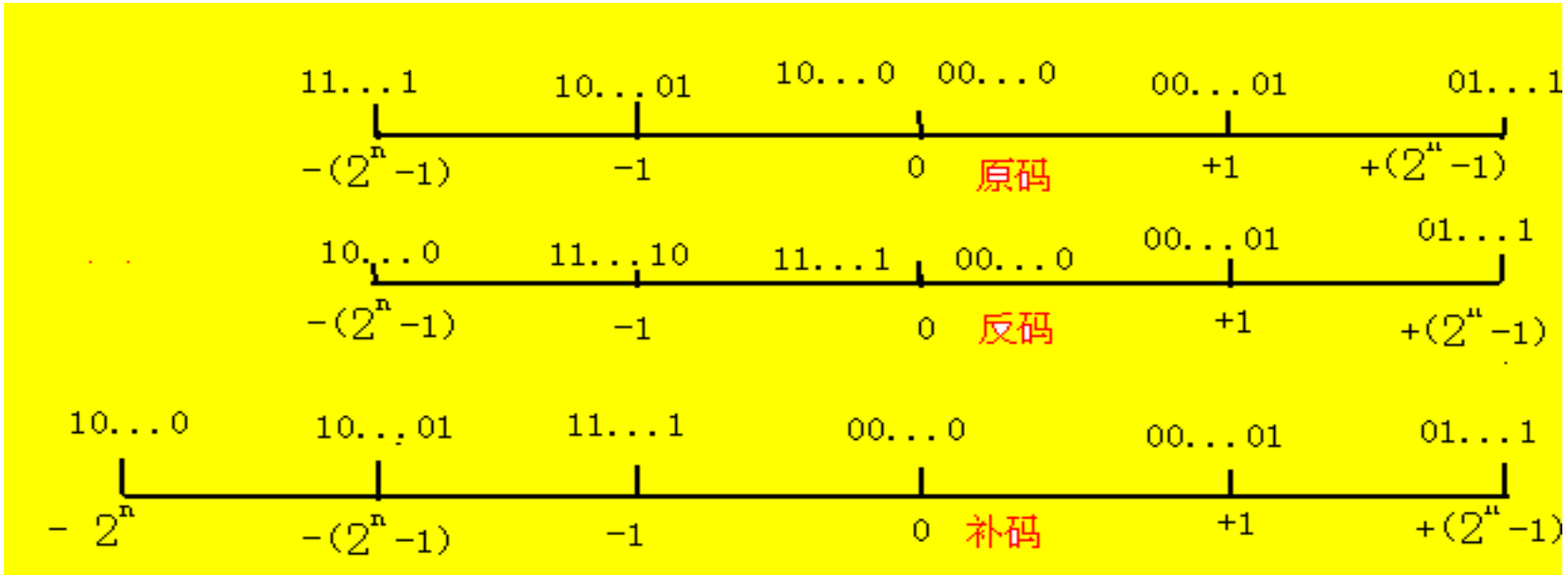
几种机器编码简便方法对比

机器码	真值为正数	真值为负数
原码	符号位为0,等于真值本身	符号位为1,数值位为真值本身 简便编码方法：加符号位
补码	符号位为0,等于真值本身	符号位为1,逐位取反,末位加1
反码	符号位为0,等于真值本身	符号位为1,逐位取反
移码	符号为1,数值位为真值本身	符号位为0,数值位逐位取反,末位加1

例3

问：以定点整数为例,用数轴形式说明原码、反码、补码表示范围和可能的数码组合情况。

[解:] 原码、反码、补码表示分别示于下图。与原码、反码不同,在补码表示中“0”只有一种形式,且用补码表示负数时范围可到-2^n。



例4

- 将十进制真值(－127,－1,0,＋1,＋127)列表表示成二进制数及原码、反码、补码、移码值。

[解:] 二进制真值 x 及其诸码值列于下表,其中0在$[x]_{原}$$[x]_{反}$中有两 种表示。由表中数据可知,补码值与移码值差别仅在于符号位不同。

真值x（十进制）	真值x（二进制）	[x]原	[x]反	[x]补	[x]移
-127	-01111111	11111111	10000000	10000001	00000001
-1	-00000001	10000001	11111110	11111111	01111111
		00000000	00000000		
0	00000000			00000000	10000000
		10000000	11111111		
+1	+00000001	00000001	00000001	00000001	10000001
+127	+01111111	01111111	01111111	01111111	11111111

例 5

- 设机器字长16位,定点表示,尾数15位,数符1位,问:

(1)定点原码整数表示时,最大正数是多少?最小负数是多少?

(2)定点原码小数表示时,最大正数是多少?最小负数是多少?

[解:]

(1)定点原码整数表示

$$最大正数值=(2^{15}-1)_{10}=(+32767)_{10}$$

0 111 111 111 111 111

$$最小负数值=-(2^{15}-1)_{10}=(-32767)_{10}$$

1 111 111 111 111 111

(2)定点原码小数表示

$$最大正数值=(1-2^{-15})_{10}=(+0.111...11)_2$$

$$最小负数值=-(1-2^{-15})_{10}=(-0.111..11)_2$$

例6

- 假设由S,E,M三个域组成的一个32位二进制字所表示的非零规格化浮点数 x，真值表示为：

  $$x=(-1)^S\times(1.M)\times 2^{E-128}$$

  问：它所表示的规格化的最大正数、最小正数、最大负数、最小负数是多少？

[解:]

(1)最大正数

0 11 111 111 111 111 111 111 111 111 111 11

$$x=[1+(1-2^{-23})]\times 2^{127}$$

(2)最小正数

0 00 000 000 000 000 000 000 000 000 000 00

$$x=1.0\times 2^{-128}$$

(3)最小负数(负数绝对值最大）

1 11 111 111 111 111 111 111 111 111 111 11

$$x=-[1+(1-2^{-23})]\times 2^{127}$$

(4)最大负数(负数绝对值最小）

1 00 000 000 000 000 000 000 000 000 000 00

$$x=-1.0\times 2^{-128}$$

2.1数据与文字的表示方法

- 1 数值数据的表示方法：
  - 进位计数法
  - 数值数据表示的特点
  - 进位计算制
  - 定点数表示
  - 浮点数表示
  - 十进制数串表示
  - 自定义数据表示
  - 机器码表示（原码、反码、补码、移码）
- 2 文字编码

字符与字符串的表示方法

- 1）字符表示法 **characters representation**
  - 如何使用数值表示字符数据？
    - *ASCII-American Standard Code for Information Interchange (ANSI 7bits)*
      - **7个bit的ASCII码**(美国国家信息交换标准字符码),它包括10个十进制数码,52个英文字母和34个专用符号,如$,%,＋,＝等，32个控制字符，如 <CR> <ESC>等，共**128**个元素。 **ASCII**码规定8个二进制位的最高位为0,余下7位可以给出128个编码。
    - *Unicode（补充讲）*

表2.1 ASCII字符编码表

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	、	p
0001	SOH	DC1	!	1	A	Q	a	q
0010	STX	DC2	“	2	B	R	b	r
0011	ETX	DC3	#	3	C	S	c	s
0100	EOT	DC4	$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	‘	7	G	W	g	w
1000	BS	CAN	(	8	H	X	h	x
1001	HT	EM	)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VT	ESC	+	;	K	[	k	{
1100	FF	FS	,	<	L	\	l	|
1101	CR	GS	-	=	M	]	m	}
1110	SO	RS	.	>	N		n	~
1111	SI	US	/	?	O	___	o	DEL

字符串数据表示

- **字符串**是指连续的一串字符,通常方式下,它们占用主存中连续的多个字节,每个字节存一个字符。

例子：将字符串：

IF␣A>B␣THEN␣READ(C)

从高位字节到低位字节依次存在主存中。

[解:]

设主存单元长度由4个字节组成。每个字节中存放相应字符的ASCII值，文字表达式中的空格“␣”在主存中也占一个字节的位置。因而每个字节分别存放十进制的73、70、32、65、62、66、32、84、72、69、78、32、82、69、65、68、40、67、41、32。

主　存

I	F	空	A
>	B	空	T
H	E	N	空
R	E	A	D
(	C	)	空

2）汉字表示法

- 汉字表示法 Chinese characters representation
  - 不是所有语言都能用128个字符表示。
  - 8 bit数据仅能表示256个字符，常用汉字6000多个，故其无法表示汉字。
  - 如何表示汉字呢？
    - **GB2312国家标准**采用16位表示
    - 与ASCII字符的区别，最高有效位MSB=1

GB2312-80国家标准

- 1981年，GB2312-80国家标准，包括6763个汉字/682个非汉字字符，称为国标码或国际交换码
- GB2312字符集的构成：
  - 一级常用汉字3755个，按汉语拼音排列
  - 二级常用汉字3008个，按偏旁部首排列
  - 非汉字字符682个

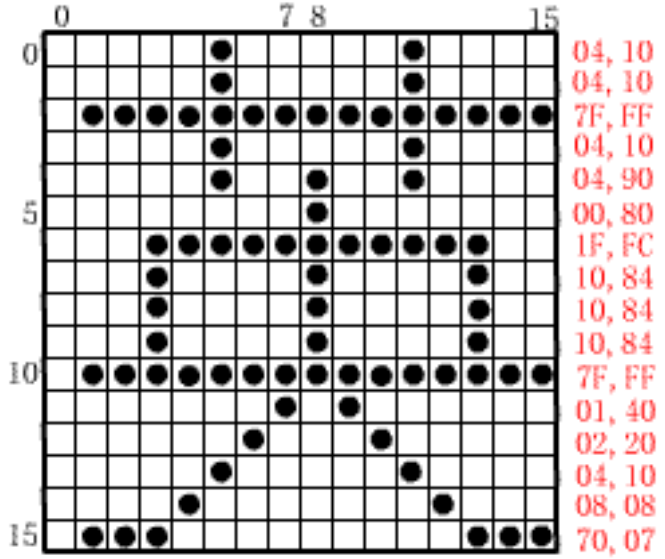
汉字编码标准

- GB2312-1980(GB0)(简体)
  - 6763个汉字
- GB13000-1993
  - 20902个汉字 (Unicode 1.1版本)
- 汉字扩展规范GBK1.0 标准1995（非国家标准）
  - 21003个字符（兼容GB2312）
- GB18030-2000(1/2/4字节编码)
  - 27484汉字 （向下兼容GB2312 GBK，GB13000）

- 汉字内码也称为机内代码。
- 外码，即输入码，也可以采用数字编码，但是难于记忆。
- 人们发明了**拼音码** 和**字形编码**
  - **拼音码**是以汉字拼音为基础的输入方法。使用简单方便，但汉字同音字太多，输入重码率很高，同音字选择影响了输入速度。
  - **字形编码**是用汉字的形状来进行的编码。把汉字的笔划部件用字母或数字进行编码，按笔划的顺序依次输入，就能表示一个汉字。

汉字字模码(显示输出用)

- **字模码**是用点阵表示的汉字字形代码,它是汉字的输出形式。字模点阵只能用来构成**汉字库**,而不能用于机内存储。字库中存储了每个汉字的点阵代码。当显示输出或打印输出时才检索字库,输出字模点阵,得到字形。



字符编码应用

- <META content="text/html; charset=gb2312"
  ... http-equiv=Content-Type>
- charset=gb2312 简体中文
  charset=big5 繁体中文
  charset=utf-8 unicode多语言

补充： Unicode

- www.unicode.org
- 用于克服字符数字的限制
- 为所有语言中的字符分配唯一的代码
- 16 bit 字符集，65536 Unicode 字符
- 提供唯一的代码
  - 不论任何平台
  - 不论任何程序
  - 不论任何语言

补充： Unicode

- Universal Character Set ISO
- UCS
  - ISO 10646
  - UCS-2 UCS-4
- UTF (Unicode Transform format)
  - UTF-7
  - UTF-8
  - UTF-16

数据校验码

- 解决编码传输问题。
- 在编码中引入一定冗余，增加代码的最小码距，使编码出现一个错误时就成为非法代码。
  - 奇偶信息的校验
  - 海明校验
  - CRC 循环冗余校验

- 一位校验位的**奇/偶校验**：

设 $x=(x_0 x_1 \ldots x_{n-1})$ **奇校验位** C 定义为

$$\overline{C}=x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1} \quad (2.15)$$

即，$C=\overline{x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}}$，使得(校验位+数据位)中1的个数为奇数。

  - ⊕代表按位加,只有当 x 中有**奇数个1**时,才使C＝0。

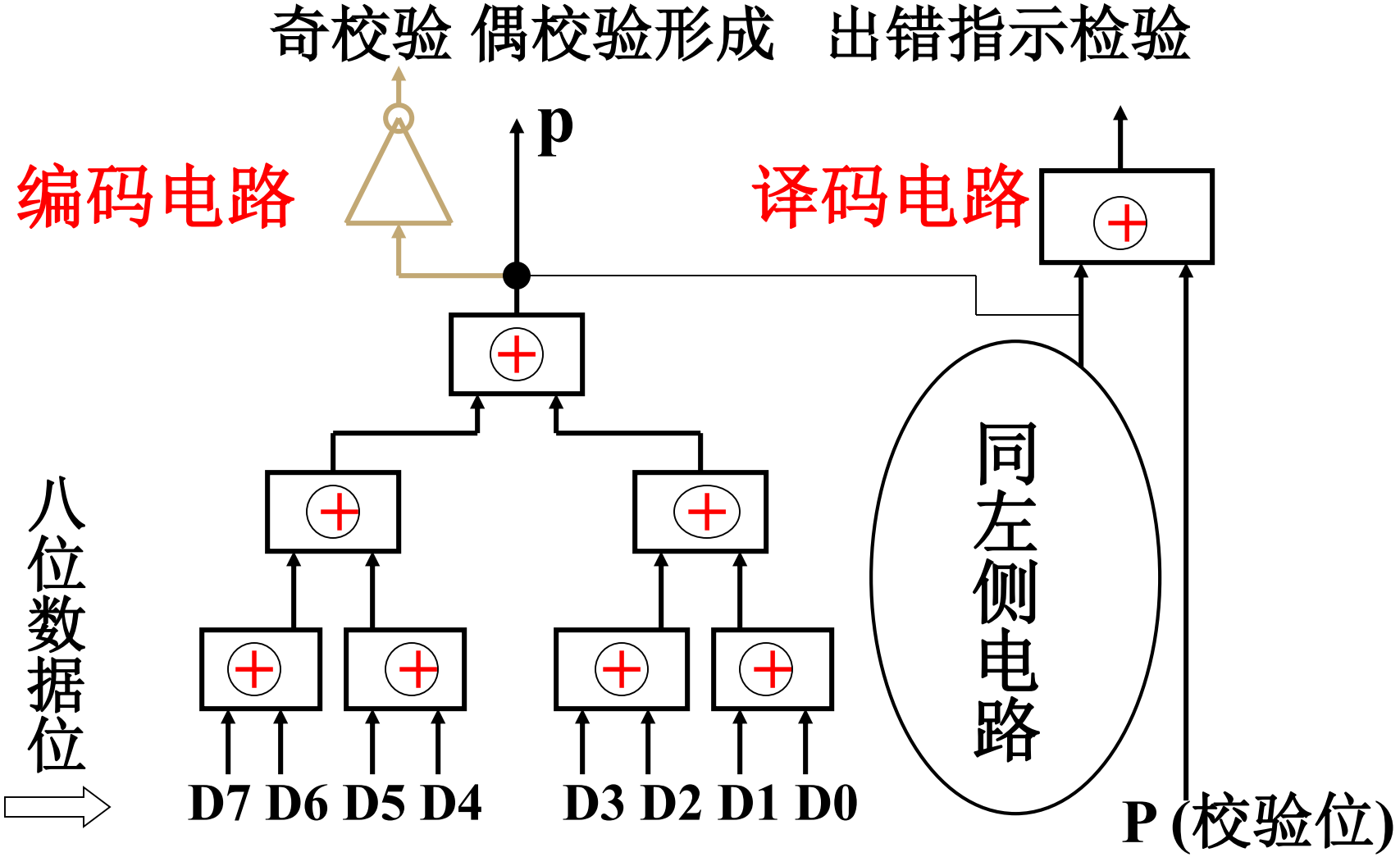
- 同理,**偶校验位** C 定义为

$$C=x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1} \quad (2.16)$$

即 x 中包含**偶数个1**时,才使C＝0。

- **检错码**：$F=C \oplus x_0 \oplus x_1 \oplus \ldots \oplus x_{n-1}$
  - F=0表示数据正常，否则表示出错。

奇偶校验码的实现电路


奇校验 偶校验形成
出错指示检验
编码电路
p
译码电路
同左侧电路
八位数据位
D7 D6 D5 D4
D3 D2 D1 D0
P (校验位)

例7

- 已知下表中左面一栏有5个字节的数据。请分别用奇校验和偶校验进行编码,填在中间一栏和右面一栏。

数　据	偶校验编码 C	奇校验编码 C
1 0 1 0 1 0 1 0	1 0 1 0 1 0 1 0 **0**	1 0 1 0 1 0 1 0 **1**
0 1 0 1 0 1 0 0	0 1 0 1 0 1 0 0 **1**	0 1 0 1 0 1 0 0 **0**
0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 **0**	0 0 0 0 0 0 0 0 **1**
0 1 1 1 1 1 1 1	0 1 1 1 1 1 1 1 **1**	0 1 1 1 1 1 1 1 **0**
1 1 1 1 1 1 1 1	1 1 1 1 1 1 1 1 **0**	1 1 1 1 1 1 1 1 **1**

海明校验Hamming Codes

- **奇偶校验**
  - 一个校验位，可提供一位错或奇数个错，但不能确定是哪一位错，也不能发现偶数个位错，更无法识别错误信息。的位置。
- **海明码**
  - 多个奇偶校验组
  - 既能检错，也能纠错
- **CRC循环冗余校验码**
  - 检错，纠错码
  - 可检测出所有的双错、奇数位错
  - 可检测所有小于、等于校验位长度的突发错