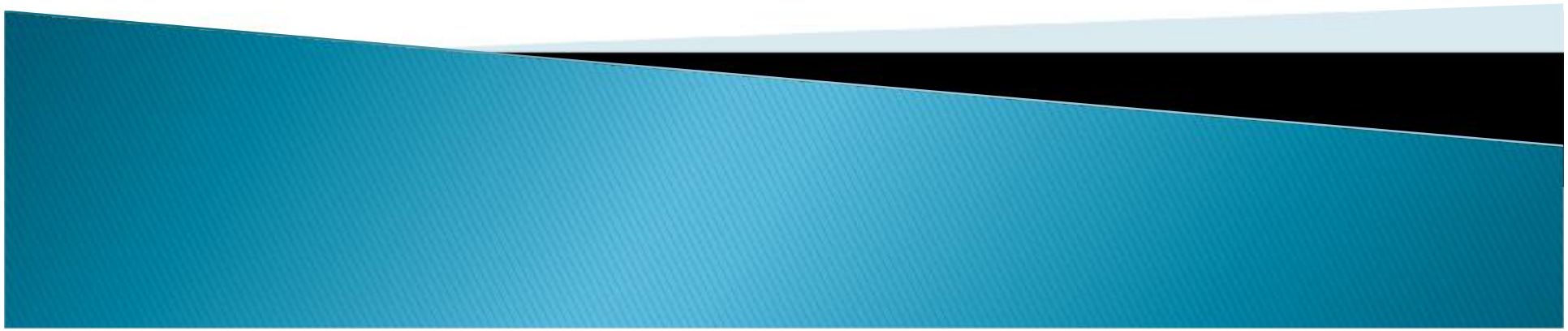
计算机系统 I

第二章

Bits, Data Types,and Operations

数据表示



数据表示

- 计算机的基本功能是进行数值运算以及对信息进行加工处理。在计算机内部，各种数值和信息都采用了数字化编码，即用最简单的二进制数码来表示。
- 本讲主要介绍常用的进位计数制、二进制运算、无符号数和带符号数的表示方法、数的定点与浮点表示方法、常见信息的编码方法等。
- 熟悉和掌握本章的内容是进行运算器设计的基础和学习计算机系统原理的最基本要求。

数据在计算机内部是怎么表示的?

- 从最低层看, 计算机实质是一个电子设备
  - 通过控制电子的流动进行工作

- 电路上很容易区分的两种情况
  1. 高电平(电压存在)　　–state “1”
  2. 低电平（电压不存在）–state “0”

- 能不能考虑根据电压值来区分更多状态?
  - 需要复杂的控制和检测电路。
  - 打开开关 vs 调整和测量电压

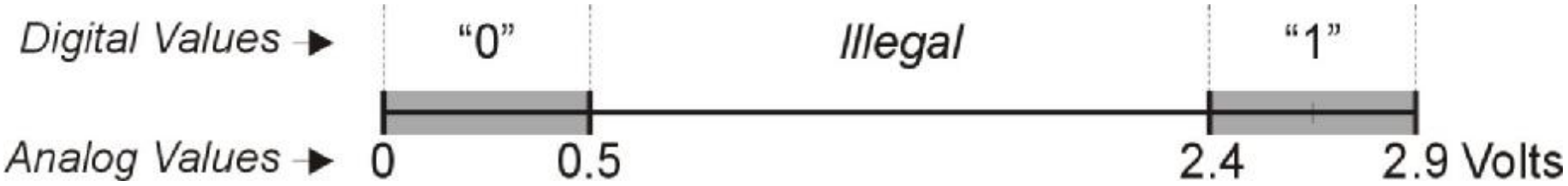
计算是采用二进制的数字系统

数字系统

- 变量具有有限个状态，
- 用有限个符号表示

二进制数字系统:

- 最简单的数字系统
- has two states: 0 and 1



- 二进制系统最基本的信息单元：位 ***bit***.
- 用多个位来表示超过两个状态的数值。
  - 两位（ two bits）二进制可以表示4个状态的数值:
    00, 01, 10, 11
  - 三位（ three bits）二进制可以表示8个状态的数值:
    000, 001, 010, 011, 100, 101, 110, 111
  - ***n bits 可以表示2^n 个状态的数值.***

计算机中的信息分类

- 数据信息
  - 数值型数据：整数，正数，负数，实数，有理数，无理数…
  - 非数值型数据：文本，声音，图像，逻辑值……
- 控制信息：指令编码

先考虑数值数据：

最常用的实数：-8.25 如何在计算机中表示？

1、**采用什么进位计数制（需要转换为二进制，数制的转换）**

（-1000.01）$_2$

2、**如何使符号（-，+号）数字化（机器数的编码方法）**

3、**如何处理和表示小数点?**

4 、**如何方便硬件实现运算。**

1 符号的处理

1）无符号数

只能表示正整数.($0<=N<=2^n-1$) n:数的位数

2） 符号数:

它的最高位被用来表示该数的符号位, 不表示数值位。

在计算机中一个数的数值部分和符号都要用0、1编码。通常, 用数的最高位（MSB—Most Significant Bit）表示数的正负

MSB = 0, 表示正数, 如＋1011表示为01011;

MSB = 1, 表示负数, 如－1011表示为11011;

2 小数点：定点与浮点表示法

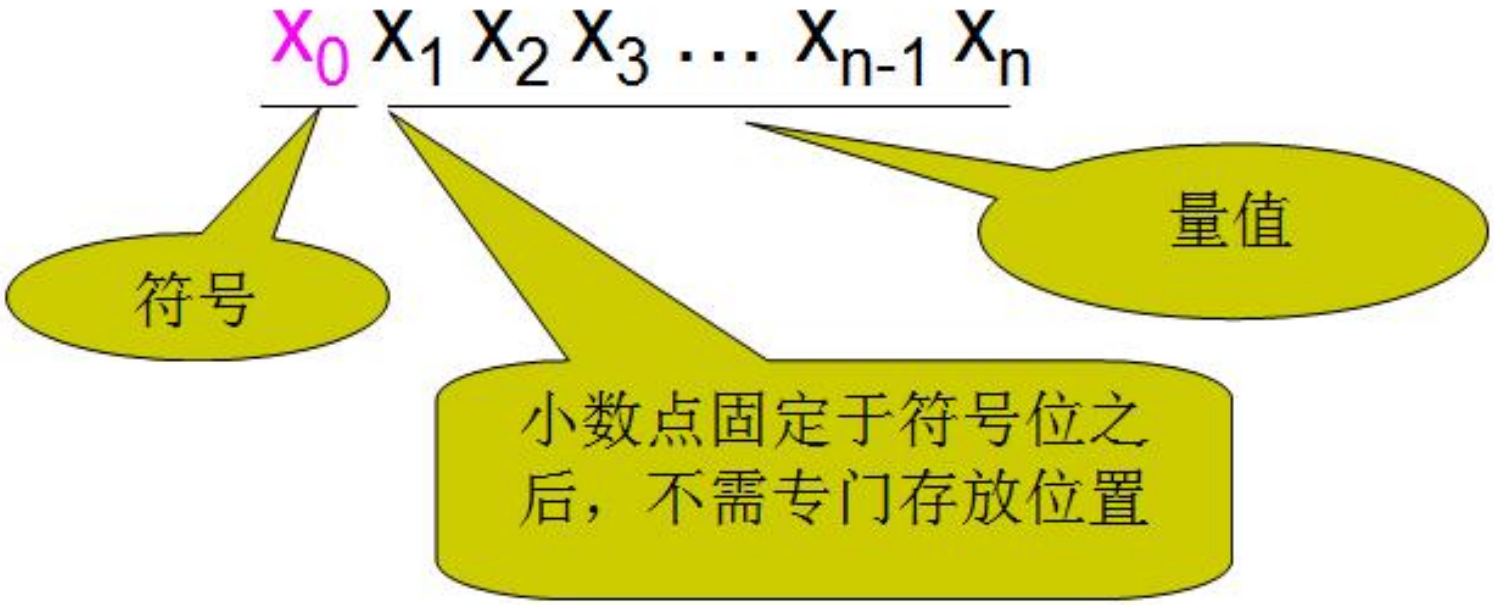
定点数表示法

- 在计算机中，图示的小数点“.”实际上是不表示出来的，是事先约定好固定在那里的。对一台计算机来说，一旦确定了一种小数点的位置，整个系统就不再改变。
- 计算机中采用的定点数表示法通常把小数点固定在数值部分的最高位之前，或把小数点固定在数值部分的最后。前者用来表示纯小数，后者用于表示整数。
- 只能处理定点数的计算机称为定点计算机。在这种计算机中机器指令访问的所有操作数都是定点数。

定点数表示法

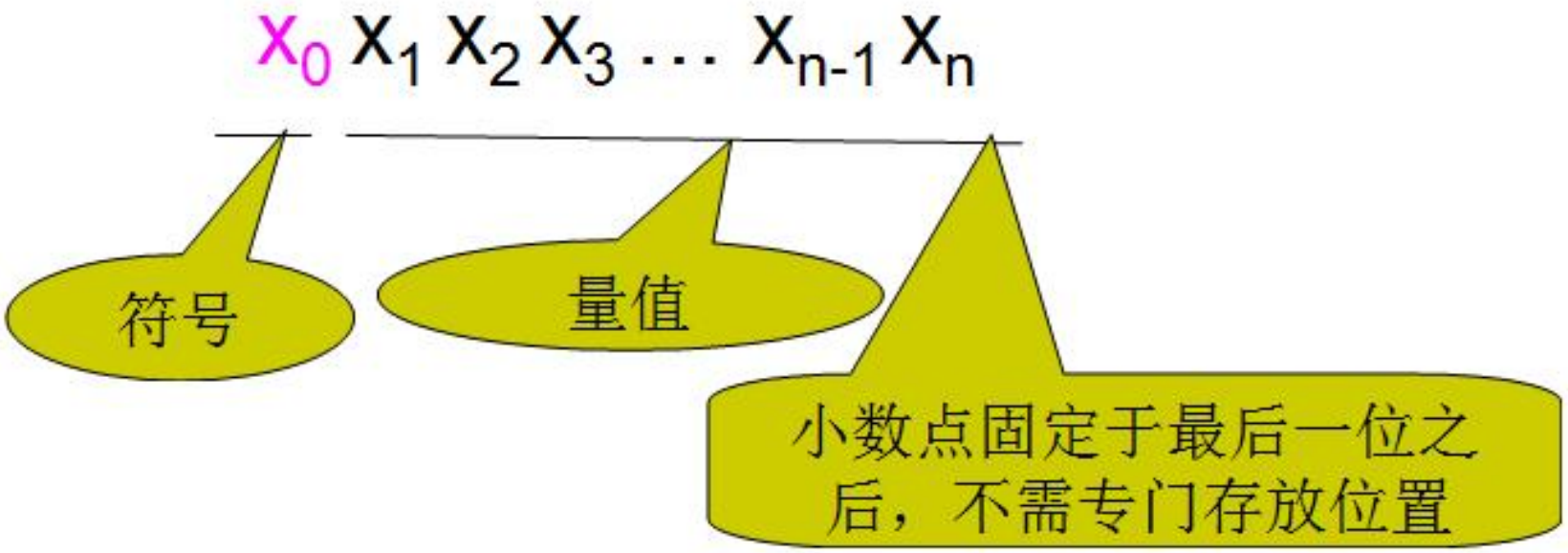
定点数要选择合适的比例因子，确保初始数据、中间结果和最后结果都在定点数的表示范围之内，否则就会产生“溢出”。

2.1 定点纯小数



表示数的范围是 $-1 < X < +1$

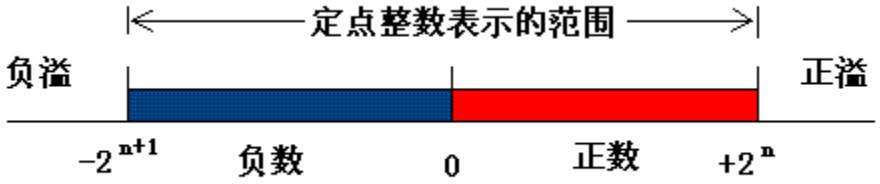
2.2定点纯整数



表示数的范围是 $-(2^n-1) \leqslant X \leqslant +(2^n-1)$

定点表示方法特点

- 定点数表示数的范围受字长限制，表示数的范围有限;
- 定点表示的精度有限
- 机器中，常用定点纯整数表示



2.3 浮点数表示法

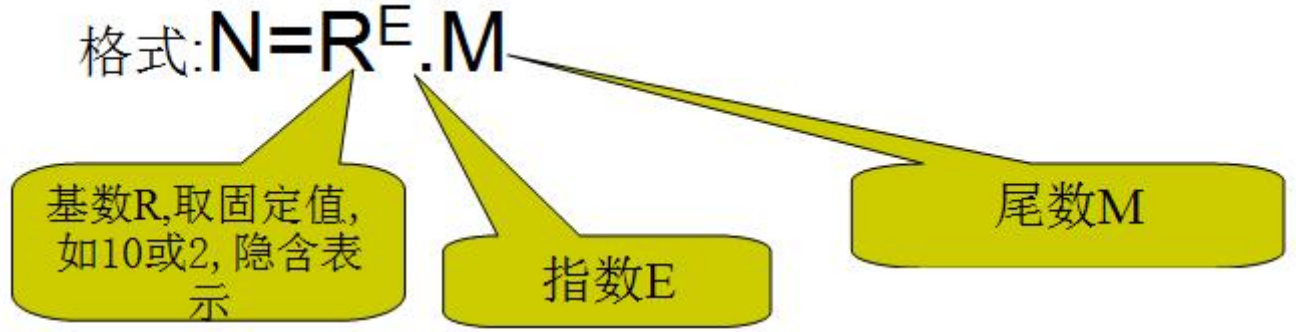
- **浮点表示法**

  (来源于科学计数法)

  电子质量(克)： $9\times10^{-28} = 0.9\times10^{-27}$

  太阳质量(克)： $2\times10^{33} = 0.2\times10^{34}$

- 小数点的位置可按需浮动，这就是浮点数。



机器中表示

浮点数的规格化表示

$(1.75)_{10} = 1.11 \times 2^0$

$= 0.111 \times 2^1$ (规格化表示)

$= 0.0111 \times 2^2$

规格化表示：阶码为纯小数，且小数点后必须为一位有效数字，对二进制而言，必须为1

目的：提高表示精度

3 常用定点数表示方法

- 原码表示法
- 补码表示法
- 反码表示法
- 移码表示法

3.1 原码表示法

原码表示法是一种最简单的机器数表示法，其最高位为符号位，符号位为0时表示该数为正，符号位为1时表示该数为负，数值部分与真值相同。

若真值为纯小数，其原码形式为$X_S.X_1X_2\cdots X_n$，其中X_S表示符号位。

原码的定义为：$[X]_{原}=\begin{cases} X & 0\le X<1 \\ 1-X=1+|X| & -1<X\le 0 \end{cases}$

例：

$X=0.0110$，$[X]_{原}=X=0.0110$

$X=-0.0110$，$[X]_{原}=1-X=1-(-0.0110)=1+0.0110=1.0110$

3.1 原码表示法

若真值为纯整数，其原码形式为$X_nX_{n-1}\cdots X_1X_0$，其中X_n为符号位。

原码的定义为：$[X]_{原}=\begin{cases} X & ,\ 0\leqslant X<2^n \\ 2^n-X=2^n+|X|, & -2^n<X\leqslant 0 \end{cases}$

例：$X=1101$　$[X]_{原}=X=01101$

$X=-1101$, $[X]_{原}=2^n-X=2^4-(-1101)=10000+1101=11101$

原码表示中，真值0有两种不同的表示形式：

$[+0]_{原}=00000$,　$[-0]_{原}=10000$

原码的优点是直观易懂，机器数和真值间的转换很容易，

3.1 原码表示法

用原码实现乘、除运算的规则简单（符号位可参与运算：xor）。

缺点是加、减运算规则较复杂。（符号位不可直接参与运算） 计算：1+(-1)

00001+10001=10010=-2

符号位如何参与运算

原则：

(1)使符号位能与有效值部分一起参加运算,从而简化运算规则.

(2)符号位参与运算后（减法可转换为加负数）使减法运算转换为加法运算,进一步简化计算机中运算器的线路设计

为解决机器内负数的符号位能与数值位一起参加运算的问题，引入了补码的概念。

3.2 补码

- 计算机运算有模运算: 参与运算的数据和结果被限定在一定范围内,该范围称 为模。

- 受字长限制,属于有模运算.
  定点小数x0.x1x2…xn ， 以2为模
  定点整数x0x1x2…xn ， 以2^{n+1}为模

- 有模运算性质
  生活例子：现为北京时间下午4点，但钟表显示为7点。有两种办法校对：
  (1) 做减法 7-3 = 4　(逆时针退3格)
  (2) 做加法 7+9 = 16 (顺时针进9格)
  16 (mod 12) = 16-12 = 4 (以12为模，变成4).
  定义钟表的模为12：钟表所能表达的数的个数
  定义-3和9互为补数 即 7-3 ó （7+9） mod 12

(1) 补码表示

- 有模运算：X-Y= [X+(-Y)补] mod 补数
  负数X的补数：模+X=模-|X|
- 确定了“模”，就可找到一个与负数等价的正数(该正数是负数的补数)来代替此负数，而这个正数可用模加上负数本身求得，这样就可把减法运算用加法实现了。
-
- 补码表示法利用模和互补的概念，可使减法运算转化成加法，从而简化计算机的运算器电路。
- 例子：1位数符,4位数值位. 计算10-3
  二进制运算通过丢弃高位自动进行模运算。

(2) 补码表示

若真值为纯小数，其原码形式为$X_nX_{n-1}\cdots X_1X_0$，其中X_n为符号位。

- 补码的定义为：$[X]_{补}=\begin{cases} X & ,0\leqslant X<1 \\ 2+X=2-|X| & ,-1<X\leqslant 0 \end{cases}$

- 例3.16：$X=0.0110$　$[X]_{补}=X=0.0110$

$X=-0.0110$　$[X]_{补}=2+X=2+(-0.0110)=10$

-0.0110

$=1.1010$

(2) 补码表示

若真值为纯整数，其原码形式为$X_nX_{n-1}\cdots X_1X_0$，其中X_n为符号位。

补码的定义为：$[X]_{补}=\begin{cases} X & 0\leqslant X<2^n \\ 2^{n+1}+X=2^{n+1}-|X| & -2^n<X\leqslant 0 \end{cases}$

例3.17：$X=1101$　$[X]_{补}=X=01101$

$X=-1101$　$[X]_{补}=2^{n+1}+X=2^5+(-1101)=$

$100000-1101$

$=10011$

在补码表示中，真值0的表示形式是唯一的：

$[+0]_{补}=[-0]_{补}=00000$。10000定义为最小的数-16

3.3 补码运算规则

补码加减法:

- 补码的一个重要特点是它可以直接进行加减法运算，并且计算简单，因此计算机中基本采用补码加减法。
- 运算规则：

（1）参与运算的操作数用补码表示，符号位作为数的一部分直接参与运算，所得即为补码表示的运算结果。

（2）若操作码为加，则两数直接相加；

（3）若操作码为减，则将减数变补后再与被减数相加。

定点加减运算

- 计算公式

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

- 例1

已知X=9，Y=3，求$[X+Y]_{补}$，$[X-Y]_{补}$，$[Y-X]_{补}$。

解： $[X+Y]_{补}=[X]_{补}+[Y]_{补}=01001+00011=01100$

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}=01001+11101=100110$

$[Y-X]_{补}=[Y]_{补}+[-X]_{补}=00011+10111=11010$

定点加减运算

- 例2： 计算 x+y

  (1)x=-1011 y=(-1000)

  (2)x=+1010 y=+1000

解：

$$\begin{array}{rr} & 10101 \\ + & 11000 \\ \hline & 101101 \end{array}$$

溢出判断

一.溢出

机器数字长8位，其中含1位数符，补码运算，定点整数表示范围为：

-128 ~ +127 （10000000~ 01111111）

所谓溢出就是指运算结果大于机器所能表示的最大正数或小于机器所能表示的最小负数。

溢出判断

- 正溢：两个正数相加后结果超出允许的表示范围。

  两个正数相加得到负数

- 负溢：两个负数相加后结果超出允许的表示范围。

  两个负数相加得正数

溢出判断

- 溢出判别

定义：补码运算时符号位直接参与运算，产生的符号位进位为C_f。最高有效数位产生的进位为C。

溢出判断

判别方法：

溢出= $C_f \oplus C$

（**Cf**为符号位运算产生的进位，**C**为最高有效数位产生的进位。）

- **Cf** 和**C**不同时表明溢出

- 例3

  X补=10101　　Y补=11000　　求X+Y

3.3 反码

（1）定义

为方便计算机实现对负数求补。可通过反码求补码。将求补操作改变为简单的求反。

- 对正数来说，其反码和原码的形式是相同的，而负数的反码是符号位为1，数值部分等于其各位的绝对值求反。如:X　　$[X]_{原}$　　$[X]_{反}$

  ＋1101　　01101　　01101

  －1101　　11101　　10010

- 在反码表示中，真值0也有两种不同的表示形式:

$[+0]_{反}=00000$

$[-0]_{反}=11111$

（2）负数的补码=反码+1

证明：负数的补码=反码+1

假设二进制补码形式为$X_nX_{n-1}\cdots X_1X_0$，其中X_n为符号位，共有n+1位。定点整数表示。

证明: 负数补码的定义： $[X]_{补} = 2^{n+1} - |X|$

其中 |X|为X符号位由1改为0，

可知一个负数|X|加上它的反码$x_{反}$，则此全部二进制位是满的，也就是全部是1，其值为$2^n + 2^{n-1} + \ldots + 2^2 + 2^1 + 2^0 = 2^{n+1} - 1$

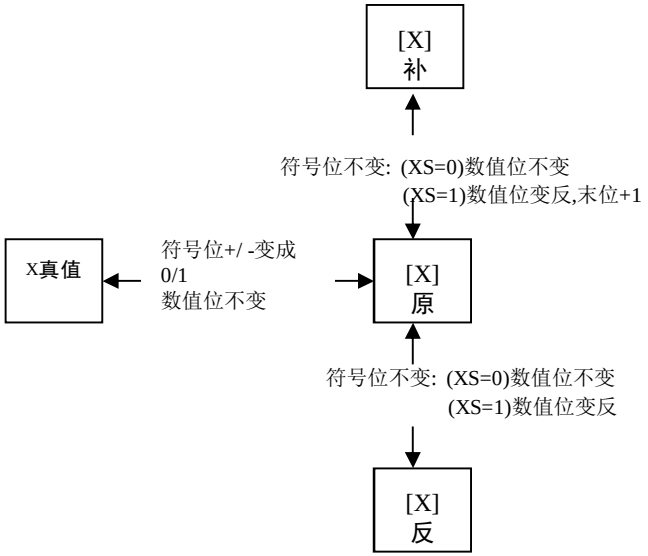
我们得出一个结论

$x_{反} + |x| = 2^{n+1} - 1$， $x_{反} = 2^{n+1} - 1 - |x|$

根据负数补码的定义： $[X]_{补} = 2^{n+1} - |X| = (2^{n+1} - X - 1) + 1 = x_{反} + 1$

通过反码，计算机内部可以通过求反操作和加法得到负数的补码。

定点原码、补码、反码的比较与转换



三种机器数及真值间的转换关系

(1)比较

● 对正数而言,上述三种码都等于真值本身。

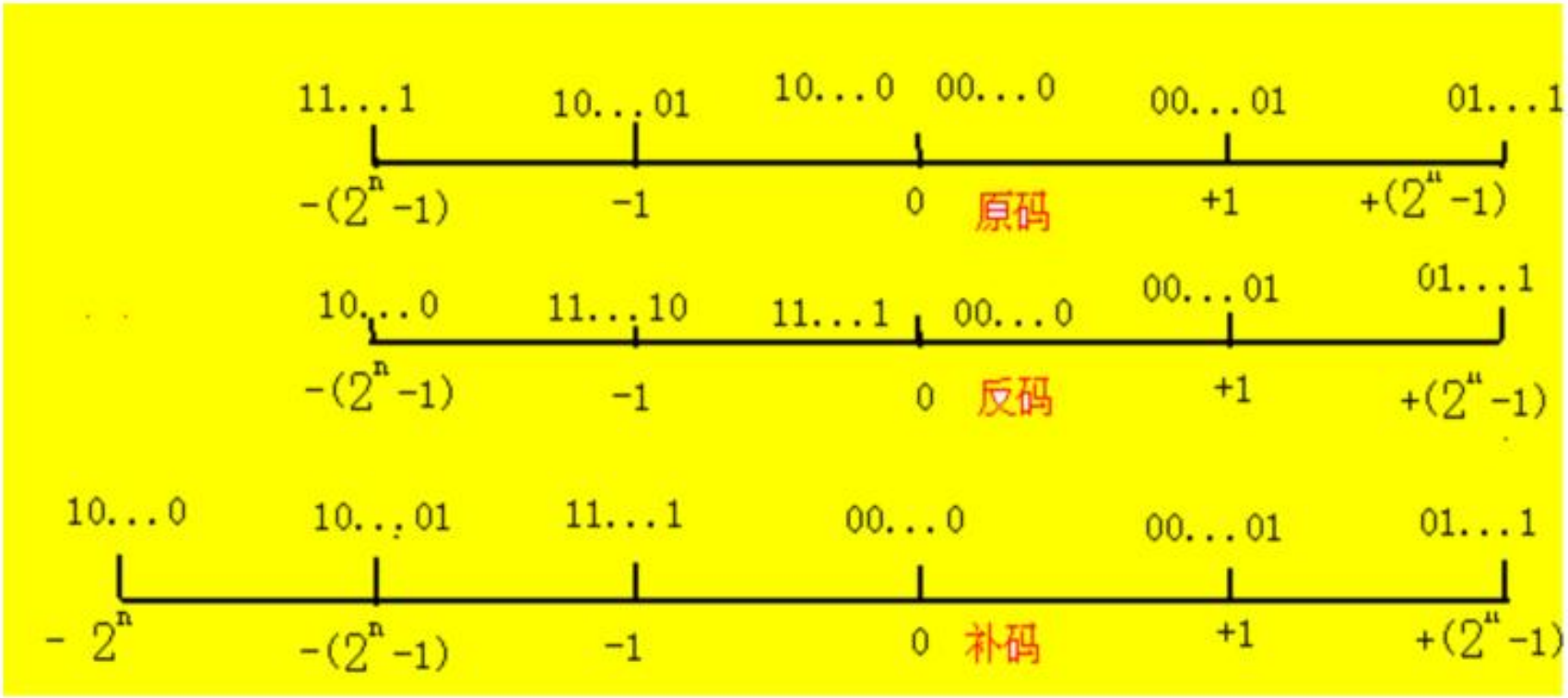
● 最高位都表示符号位,补码的符号位可与数值位一样看待,和数值位一起参加运算; 但原码的符号位必须与数值位分开处理。

● 原码和反码的真值0各有两种不同的表示方式,而补码的真值0表示是唯一的。

(2)转换

三种机器数及真值的转换关系如上图所示。从图中可见,真值X与补码或反码间的转换是通过原码实现的,当然,对于已熟练掌握转换方法也可直接完成真值X与补码或反码间的转换。

[例] 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。（n:数值位数，符号位1位）



符号扩展

- 8位数加+4位数

- 在左侧添’0’

4-bit		8-bit	
0100	(4)	00000100	(still 4)
1100	(-4)	00001100	(12, not -4)

- 在左侧添加符号位

4-bit		8-bit	
0100	(4)	00000100	(still 4)
1100	(-4)	11111100	(still -4)

3.4 移码

- 移码也叫增码或偏码，常用于表示浮点数中的阶码。

  规格化浮点数大小比较：阶码比较(浮点数加减运算需要对阶比较)

  思路：统一加一个偏移量，将负数空间移动到正整数空间方便比较

- 移码就是在真值X基础上加一常数，此常数被称为偏置值，相当于X在数轴上向正向偏移了若干单位，这就是“移码”一词的由来。即：

- 对字长为N的计算机，若最高位为符号位，数值位n=N-1位，当偏移量取2^n时，其真值X对应的移码的表示公式为：

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

  $[X]_{移} \quad (0 \leqslant X < 2^{n+1})$

  例如：X=＋1011　$[X]_{移}$=11011　符号位“1”表示正号

  X=－1011　$[X]_{移}$=00101　符号位“0”表示负号

- 移码与补码的关系：[X]移与[X]补的关系是符号位互为反码，

  例如：X=＋1011　$[X]_{移}$=11011　$[X]_{补}$=01011

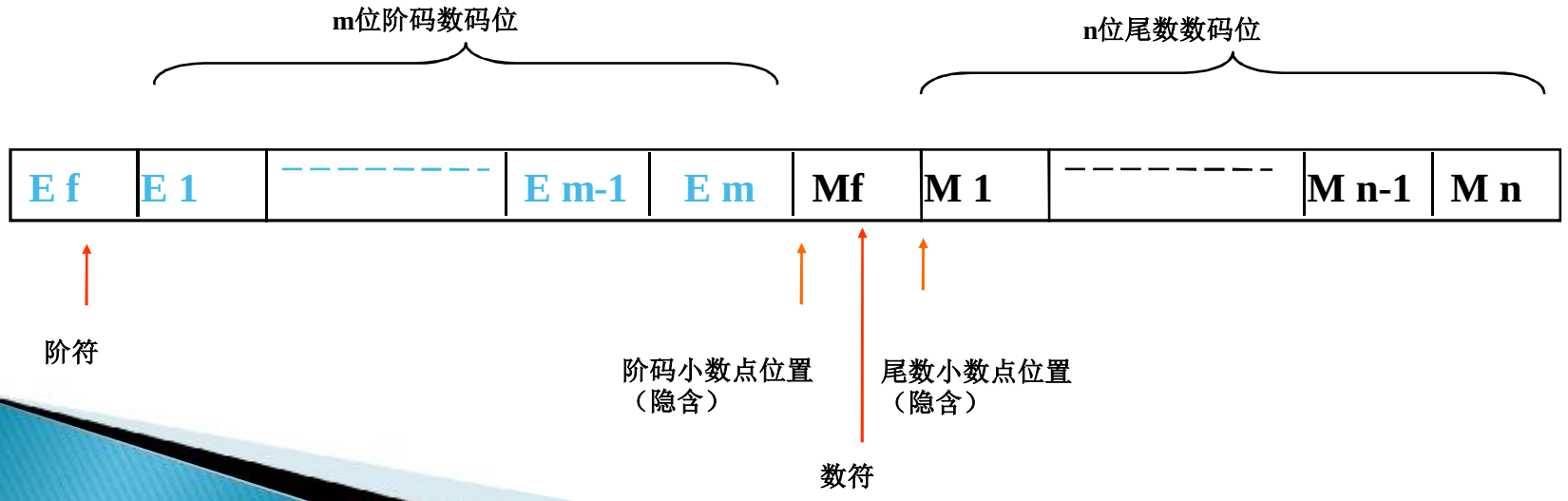
  X=－1011　$[X]_{移}$=00101　$[X]_{补}$=10101

移码 = 补码符号位取反

4 浮点数表示

- **浮点数的表示：** $N=2^{E}\cdot M$

其中E是一个二进制整数，称为数N的阶码，2为阶码的基数，M是二进制小数，称为数N的尾数。E和M可正可负。尾数M表示数N的全部有效数据，阶码E指明该数的小数点位置，表示数据的大小范围。



浮点数的表示-IEEE 754

- IEEE 754标准：

  1、 阶码采用移码表示，基为2 ：移码方法对两个指数大小的比较和对阶操作都比较方便，因为阶码域值大者其指数值也大。

  采用这种方式时，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如：127(01111111)，即 E＝e＋127 。

  说明：IEEE 754中定义的短浮点数（32位）的移码的偏置值是127(3FH)；长浮点数（64位）的偏置值是1023（3FFH）。

  2、尾数符号位占用一位。1：负数 0：正数

  3、尾数用原码，采用隐含位表示：对规格化的二进制浮点数，约定最高位总是“1”,为使尾数能多表示一位有效值，可将这个“1”隐含.

- 浮点数的阶码的位数决定数的表示范围，尾数的位数决定数的精度，精度不够造成误差

区别：规格化浮点数与IEEE754 规格化（隐藏位技术）

- 规格化浮点数（最高有效数值位为1）
  规格化浮点数的尾数M应该在下列范围内（纯小数最高有效数位为1）
  0.5<=|M|<=1
  因为$0.5=2^{-1}$
  因此：对于正数规格化浮点数的表示方法为：
  0.1xx...xx

对于负数原码：1.1xx...xx
对于负数补码：1.0xxxxx (与符号位不同)

- IEEE754 规格化-隐藏位技术
  尾数采用原码表示，符号位单独表示，最高位为1
  .1(xx...xxx) 左移一位1.xx...xxx，阶码-1 保存.xx...x(1默认不保存)
  .1(xx..xxxx)左移一位1.xx...xxx，阶码-1 保存.xx...x(1默认不保存)

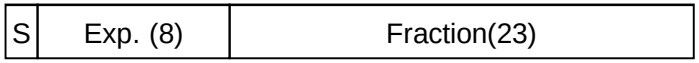
实际计算时需要恢复原值

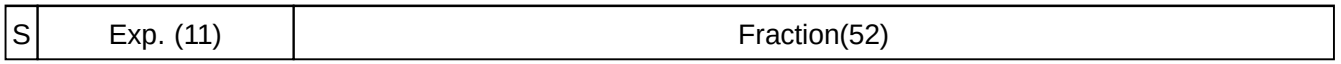
注意:隐含的“1”是一位整数(即位权为2^0),在浮点格式中表示出来的尾数是纯小数,用原码表示.

例：$(12)_{10}=(1100)_2$,将它规格化为1.1×2^3,其中整数部分的1将不存储在23位尾数内.故尾数数值实际上是24位,即1位隐含位加23位小数位

浮点数的表示-IEEE 754

类型	数符	阶码	尾数位	总位数	偏置值
短浮点数	1	8	23	32	7FH(127)
长浮点数	1	11	52	64	3FFH
临时浮点数	1	15	64	80	3FFFH





IEEE 754标准中有三种形式的浮点数, 格式见表**1**。
短浮点数即单精度浮点数，长浮点数即双精度浮点数,
都采用隐含尾数最高数位的方法, 故增加了一位尾数。
临时浮点数又称扩展精度浮点数，无隐含位。
临时浮点数仅用于计算过程，它不属于 **IEEE** 标准，
在某些计算机的系统中，用于减少计算误差。

短浮点数:最高位为数符位 ;其后是**8**位阶码,
以**2**为底, 用移码表示, 阶码的偏移值为**127**
(叫移**127**码**)**; 其余**23**位是尾数的数值位。
长数值位为**24**位

浮点数的表示（2）

- IEEE 754 浮点格式参数

参数	单精度	双精度
字长	32	64
阶码位数	8	11
阶码偏移量	127	1023
最大阶码	127	1023
最小阶码	-126	-1022
表示范围	$10^{-38}, 10^{+38}$	$10^{-308}, 10^{+308}$
尾数位数	23	52
阶码个数	254	2046
尾数个数	2^{23}	2^{52}
值的个数	$1.98*2^{31}$	$1.99*2^{63}$

浮点数的表示（3）

- **IEEE 754 浮点格式**

	单精度				双精度			
	符号位	阶码	尾数	值	符号位	阶码	尾数	值
正无限	0	255	0	¥	0	2047	0	¥
负无限	1	255	0	-¥	1	2047	0	-¥
Quiet NaN	0 或 1	255	高位=0	NaN	0 或 1	2047	高位=0	NaN
Signaling NaN	0 或 1	255	高位=1	NaN	0 或 1	2047	高位=1	NaN
正规格化非 0	0	0<e<255	f	$2^{e-127}(1.f)$	0	0<e<2047	f	$2^{e-1023}(1.f)$
负规格化非 0	1	0<e<255	f	$-2^{e-127}(1.f)$	1	0<e<2047	f	$-2^{e-1023}(1.f)$
正非规格化非 0	0	0	f!=0	$2^{e-126}(0.f)$	0	0	f!=0	$2^{e-1022}(0.f)$
负非规格化非 0	1	0	f!=0	$-2^{e-126}(0.f)$	1	0	f!=0	$-2^{e-1022}(0.f)$
正 0	0	0	0	0	0	0	0	0
负 0	1	0	0	-0	1	0	0	-0

- Ø 把 Singaling NaN 当作操作数引起例外，把 Quiet NaN 当作操作数结果也为 Quiet NaN，可以用尾数部分区分它们。
- Ø 非规格化数用于阶码下溢的情况，填补最小数和 0 之间的“空隙”。

IEEE 浮点数标准754 (短浮点数)

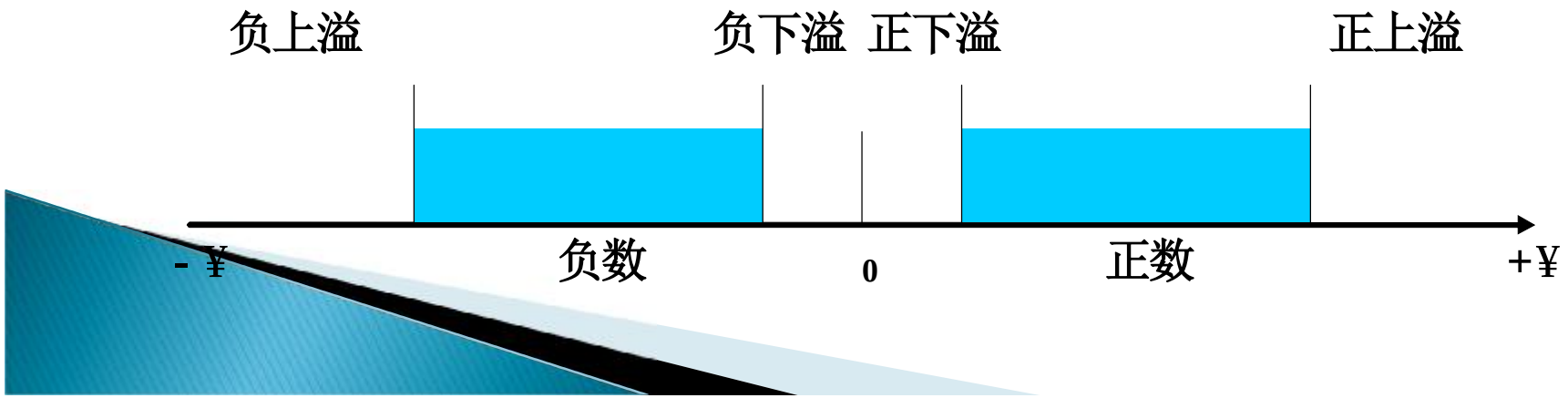
S(1位)	E(8位)	M(23位)	X(共32位)
符号位	0	0	0
符号位	0	不等于0	$(-1)^{S}\cdot 2^{-126}\cdot(0.M)$ 非规格化
符号位	1到254之间	不等于0	$(-1)^{S}\cdot 2^{E-127}\cdot(1.M)$ 规格化
符号位	255	0	无穷大
符号位	255	不等于0	NaN(非数值)

鉴于IEEE754标准对计算机界的重要贡献,发挥关键作用的数学家Kahan于1989年被授予图灵奖。

浮点数的表示范围

- 浮点数的溢出表现为阶码的溢出

- **浮点数的上溢** (overflow)
  数据太大，以至于大于阶码所能表示的数值
- **浮点数的下溢** (underflow)
  数据太小，以至于小于阶码所能表示的数值



浮点数举例

例1：将$(100.25)_{10}$转换成短浮点数格式。

解：(1)把十进制数转换成二进制数

$(100.25)_{10}=(1100100.01)_2$

(2)规格化二进制数

$1100100.01=1.10010001\times 2^{6}$

(3)计算出阶码的移码（偏置值＋阶码真值）

01111111(127H)＋110 ＝10000101

注意：短浮点数的阶码偏置值是1111111(127H)。

(4)以短浮点数格式存储该数

该数的符号位＝0 ，阶码＝10000101

尾数＝10010001000000000000000 23位（原码）

所以$(100.25)_{10}$的短浮点数代码为

0； 10000101 ； 10010001000000000000000

十六进制值是42C88000H。

？$(-100.25)_{10}$

浮点数举例

例2：将短浮点数C1C90000H转换成十进制数。

解：（1）把十六进制数转换成二进制形式，并分离出符号位、阶码和尾数

因为，C1C90000H＝11000001100100100000000000000000B

所以，　　符号位＝1

阶码＝10000011（用黑体字表示）→　　8位

尾数＝10010010000000000000000　　→　　23位

（2）计算出阶码的真值（即移码－偏置值）

10000011 -1111111＝100　真值为4

（3）以规格化二进制数形式写出此数

1.1001001×2^{4}

（4）写成非规格化二进制数形式

11000.001

(5)转换成十进制数,并加上符号位

$(11000.001)_2=(25.125)_{10}$

5 二进制逻辑运算

一个二进制数位可以用来表示一个二值逻辑型的数据，但逻辑型数据并不存在进位关系。

这里的与、或、非逻辑可以用与门、或门、非门电路实现 。

X	Y	X与Y	X或Y	X非	X异或Y
0	0	0	0	1	0
0	1	0	1	1	1
1	0	0	1	0	1
1	1	1	1	0	0

6 移位操作

- 移位

  1、逻辑移位：左移时低位补0，右移时高位补0

  2、循环移位：首尾相接，左右循环移动

  3、算术移位：移位后没有溢出时，左移一位相当于乘2（*2），右移一位相当于除2（x1/2）

  (1) 原码移位：保持符号位不变，各位依次左/右移，缺位补0。

  (2) 补码左移：各位依次左移，最高有效位移到符号位，末位补0

  (3) 补码右移：连同符号位一起，各位依次右移，符号位的值移到最高有效位，而符号位本身保持不变

- A为二进制数：

  可用1/2*A 代表将A右移一位

  2A或2*A代表将A左移动一位

移位操作

1. 移位类型

逻辑移位　　：数码位置变化

算术移位　　：数码位置变化，数值变化，
符号位不变。

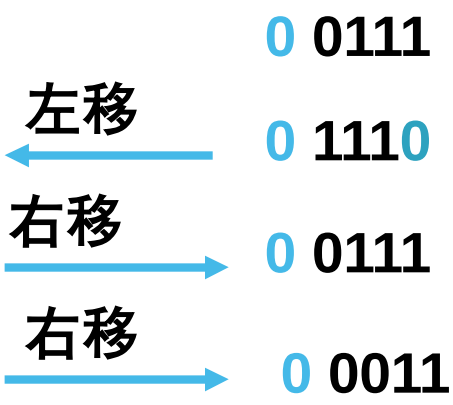
算术左移：

1 0 0 1 1 1 1　　(-15)

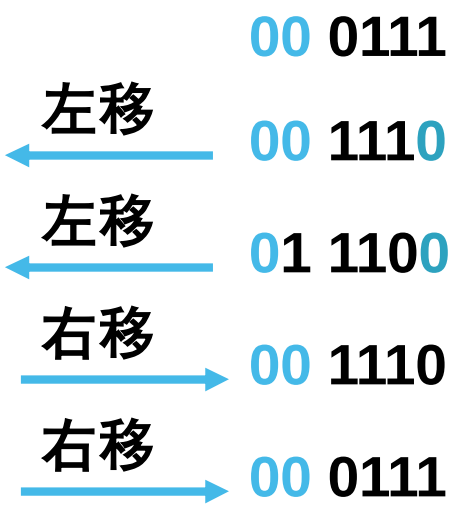
1 0 1 1 1 1 0　　(-30)

2.正数补码移位规则

（1）单符号位：



（2）双符号位：



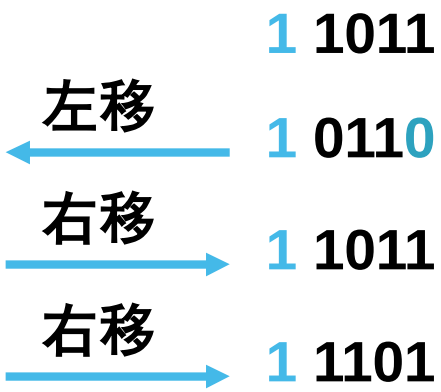
（3）移位规则

数符不变　　（单：符号位不变；双：第一符号位不变）。

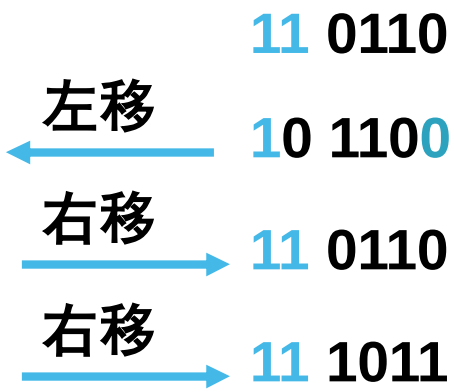
空位补0　　（右移时第二符号位移至尾数最高位）。

3.负数补码移位规则

（1）单符号位：



（2）双符号位：



（3）移位规则

数符不变　　（单：符号位不变；双：第一符号位不变）。

左移空位补0

右移空位补1　　（第二符号位移至尾数最高位）。

例题：

- 已知x=0.1011,y=-0.0101

  求$1/2x_{补}$，$1/4x_{补}$，$-x_{补}$，

  $1/2y_{补}$，$1/4y_{补}$，$-y_{补}$

解：$[x]_{补}=0.1011$, $[y]_{补}=1.1011$

$[1/2X]_{补}=0.010111$ $[1/2y]_{补}=1.110111$

$[1/4X]_{补}=0.001011$ $[1/2y]_{补}=1.111011$

$[-X]_{补}=1.0101$ $[-y]_{补}=0.0101$

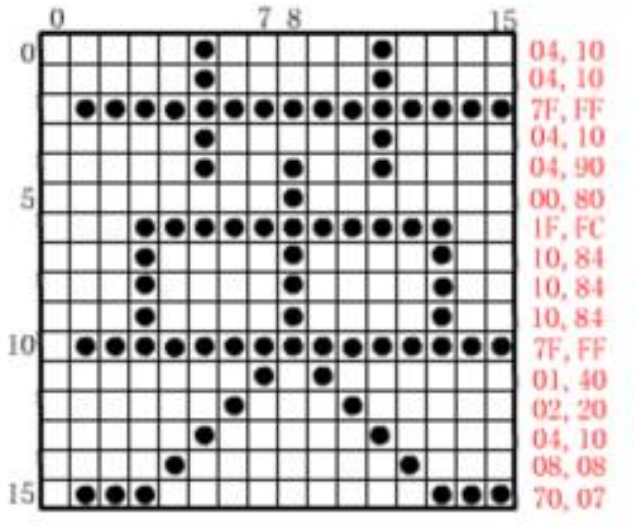
7 非数值数据编码

ASCII字符编码

- 符号数据：字符信息用数据表示，如ASCII等；
- 字符表示方法ASCII:用一个字节来表示,低7位用来编码(128),最高位为校验位
- 字符串的存放方法：C语言

中文编码–汉字的存放

- 点阵



- 汉字库

输入码-》汉字内码(地址)-》汉字库-》点阵图形

中文的编码表示

- 汉字的表示方法 （一级汉字3755个，二级汉字3008个）
  - 输入码
    - 国标码
    - 一级（16~55）*94
    - 二级（56~87）*94
    - 图形符号（682个）（01~09）*94
  - 拼音、五笔
- 汉字内码：汉字信息的存储，交换和检索的机内代码，两个字节组成，每个字节高位都为1（区别于英文字符）

其他数据类型的表示

- 文本，字符串
  - sequence of characters, terminated with NULL (0)
  - typically, no hardware support
- 图像
  - array of pixels
    - monochrome: one bit (1/0 = black/white)
    - color: red, green, blue (RGB) components (e.g., 8 bits each)
    - other properties: transparency
  - hardware support:
    - typically none, in general-purpose processors
    - MMX -- multiple 8-bit operations on 32-bit word
- 声音
  - sequence of fixed-point numbers

LC-3 使用的Data Types

- Some data types are supported directly by the instruction set architecture.
- For LC-3, there is only one hardware-supported data type:
  - 16-bit 2's complement signed integer
  - Operations: ADD, AND, NOT
- Other data types are supported by interpreting 16-bit values as logical, text, fixed-point, etc., in the software that we write.

作业

- Blackboard 发布
- 要求独立完成，坚决杜绝不诚信现象。
- 注意提交时间，不接收迟交作业。