# 计算机中的数及其运算

# 1. 计算机中的二进制数表示

## 数的表示方法

- 定点数
  - 定点整数
  - 定点小数
- 浮点数

① 编程时需要确定小数点位置；
② 难以表示两个大小相差较大的数
③ 存储空间利用率低

## 数的性质

- 无符号数
- 有符号数

数中所有的0和1都是数据本身

需用0或1表示数的性质（整

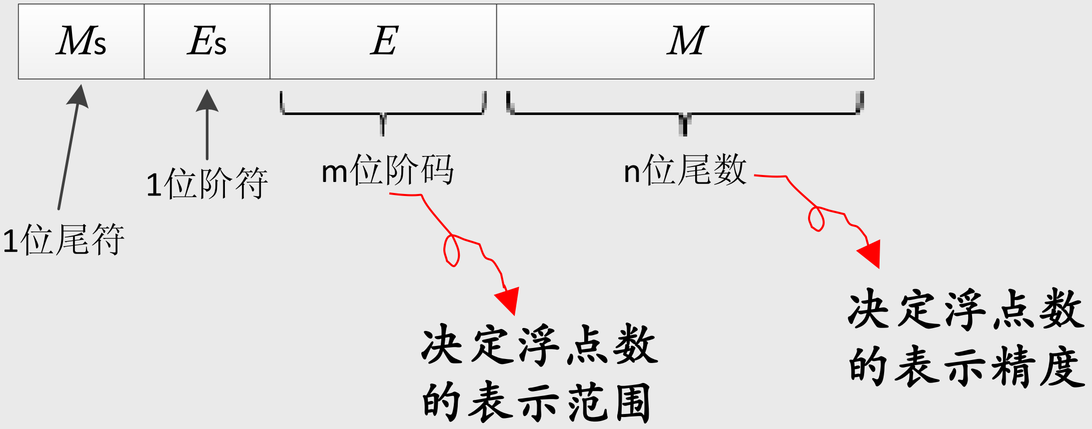
各种编码多视为无符号数

数值 符号数

机器数

# 浮点数

- **浮点数：** 小数点的位置可以左右移动的数
- 规格化浮点数
  - 尾数部分用纯小数表示，即小数点右边第1位不为0



# 2. 无符号数

- 无符号数的算术运算
  - 加法运算
    - 1+1=0（有进位）
  - 减法运算
    - 0-1=1（有借位）
  - 乘法运算
  - 除法运算

例：

- 00001011×0100 **=00101100B**
- 00001011÷0100 =00000010B

商=00000010B

余数=11B

每乘以2，相对于被乘数向左移动1位

每除以2，相对于被除数向右移动1位

# 3. 有符号数

- 有符号数：
  - 用最高位表示符号，其余是数值
- 符号数的表示方法：
  - 原码
  - 反码
  - 补码

0：表示正数

1：表示负数

**数的性质由设计者决定**

在低级语言程序设计中，根据数的性质由程序语言处理（按无符号数或有符号数处理）。

# 1）原码

- 最高位为符号位，其余为真值部分。
  - $[X]_{原}$=符号位+|绝对值|
- 优点：
  - 真值和其原码表示之间的对应关系简单，容易理解；
- 缺点：
  - 计算机中用原码进行加减运算比较困难
  - 0的表示不唯一。

# 数0的原码

- 8位数0的原码：
  - +0=0 0000000
  - −0=1 0000000

**数0的原码不唯一**

# 2）反码

对一个机器数X：

- 若X>0，则 $[X]_{反}=[X]_{原}$
- 若X<0，则 $[X]_{反}$= 对应原码的符号位不变，数值部分按位求反。

- 例：
  - X= -52 = -0110100

    $[X]_{原}$=<u>1</u> 0110100

    $[X]_{反}$=<u>1</u> 1001011

# 0的反码：

- $[+0]_{反} = [+0]_{原} = 00000000$
- $[-0]_{原} = 10000000$
- $[-0]_{反}$ = = $[-0]_{原}$数值部分分按位取反= 11111111

**即：数0的反码也不是唯一的。**

# 3）补码

定义：

- 若X>0， 则$[X]_{补}=[X]_{反}=[X]_{原}$
- 若X<0， 则$[X]_{补}=[X]_{反}+1$

# [例]

- $X = -52 = -0110100$

$[X]_{原} = 10110100$

$[X]_{反} = 11001011$

$[X]_{补} = [X]_{反} + 1 = 11001100$

# 0的补码：

- $[+0]_{补}=[+0]_{原}=00000000$
- $[-0]_{补}=[-0]_{反}+1=11111111+1$

$=\underline{1}\ 00000000$

对8位字长，进位被舍掉

# 补码的说明

- 钟表例：
  - 将指针从5点拨到1点
- 两钟拨法：
  - 逆时钟拨：5-4=1
  - 顺时钟拨：5+8=12+1 **=1**

12为模
自动丢失

**实现将减法运算转换为加法运算**

- 对模12，有：
  - 5-4=5+8
  - $[-4]_{补}$=12-4=8

8为-4的补数

**5-4=5+（-4）=5+（12-4）=5+8=12 +1**

# 补码的算术运算

- 通过引进补码，可将减法运算转换为加法运算。
- 即：
  - $[X+Y]_{补}=[X]_{补}+[Y]_{补}$
  - $[X-Y]_{补}=[X+(-Y)]_{补}=[X]_{补}+[-Y]_{补}$

# 例1：

- 66-51=66+（-51）=15
- 用二进制补码运算：
  - $[+66]_{补}= [+66]_{原}= 01000010$
  - $[-51]_{原}=10110011$
  - $[-51]_{补}=11001101$
  - $[+66]_{补}+ [-51]_{补}= 1\ 00001111$

    $=15$

# 例2：

- X=-52=-0110100，Y=116=+1110100，求X+Y=?
  - $[X]_{原}=10110100$
  - $[X]_{补}=[X]_{反}+1=11001100$
  - $[Y]_{补}=[Y]_{原}=01110100$
  - $[X+Y]_{补}=[X]_{补}+[Y]_{补}$

$=11001100+01110100$

$=01000000$

$X+Y=+1000000$

- **现代计算机系统中，程序设计时，负数可用“-”表示，由编译系统将其转换为补码。**
- **例：**
  - 若输入数= -3
  - 程序编译后的值=FDH

# 特殊数10000000

- 对无符号数:
  - $(10000000)_B=128$
- 在原码中定义为:
  - $(10000000)_B=-0$
- 在反码中定义为:
  - $(10000000)_B= -127$
- 在补码中定义为:
  - $(10000000)_B= -128$

# 4. 计算机能力的局限性

- 计算机的运算能力是有限的
  - 计算机无力解决无法设计出算法的问题
  - 无法处理无穷运算或连续变化的信息
- 计算机能够表示的数（表数）的范围是有限的
  - 计算机的表数范围受字长的限制
  - 例：对8位机：
    - 无符号数的最大值：1111 1111
    - 有符号正数的最大值：0111 1111

**当运算结果超出计算机表数范围时，将产生溢出**

# 1）无符号整数的表示范围：

- 当计算机中数的运行结果超出表数范围时，则产生溢出。
- 无符号整数的表数范围：
  - $0 \leqslant X \leqslant 2^{n}-1$

n表示字长

**<u>无符号数加减运算溢出的判断方法：</u>**

当最高位向更高位有进位（或借位）时则产生溢出

# [例]：

- 2个8位数的加法运算

$$
\begin{array}{r}
11111111 \\
+\ 00000001 \\
\hline
1\ \ 00000000
\end{array}
$$

溢出位超出8位有效字长

在有效字长范围内，结果为0

出错

最高位向前有进位，产生溢出

# 2）有符号整数的表示范围

- 原码和反码：
  - $-(2^{n-1}-1) \leqslant X \leqslant 2^{n-1}-1$
- 补码：
  - $-2^{n-1} \leqslant X \leqslant 2^{n-1}-1$
- 对8位二进制数：
  - 原码： -127 ～+127
  - 反码： -127 ～+127
  - 补码： -128 ～+127

# 符号数运算中的溢出判断

- 两个有符号二进制数**相加或相减**时，若运算结果超出可表达范围，则产生溢出
- 溢出的判断方法：
  - 最高位进位状态⊕次高位进位状态＝1，则结果溢出

**除法运算溢出时，产生"除数为0"中断**

**乘法运算无溢出问题**

# [例]：

- 若：X=01111000， Y=01101001

  则：X+Y=

$$
\begin{array}{r}
01111000 \\
+\ 01101001 \\
\hline
11100001
\end{array}
$$

次高位向最高位有进位，而最高位向前无进位，产生溢出。

（事实上，两正数相加得出负数，结果出错）

# 5. 符号二进制数与十进制的转换

- 转换方法:
  - 求出真值
  - 进行转换
- 计算机中的符号数默认以补码形式表示。

原码=符号位+绝对值

正数的补码=原码=符号位+绝对值

∵负数的补码≠原码　∴负数的补码≠符号位+绝对值

# 例：补码数转换为十进制数

- 设：
  - $[X]_{补}=\underline{0}\ 0101110B \longrightarrow$ 真值 $=+0101110B \longrightarrow X=+101110B=+46$

正数

- 若设：
  - $[X]_{补}=\underline{1}\ 1010010B \longrightarrow X\neq-1010010B \longrightarrow$ 欲求X真值，需对$[X]_{补}$再取补

正数

$$X=[[X]_{补}]_{补}=[11010010]_{补}=-0101110=-46$$

- 对正数：
  - 补码=反码=原码，且 原码=符号位+真值
  - 所以：正数补码的数制部分为真值
- 对负数：
  - 补码≠反码≠ 原码
  - 所以：负数补码的数制部分≠真值

因正数的反码、补码与其对应的原码相同，故其数值部分亦为真值。

只有原码的数值部分是真值

反码和补码的数值部分都不是真值

谢谢！