第1章 计算机基础知识
本章知识点提要

1.数制

1)十进制（Decimal)

2)二进制（Binary）

3)八进制(Octal)

4)十六进制(Hexadecimal)

重点是二进制和十六进制

2.数制间转换

1）二、八、十六到十进制

2）十进制到二、八、十六进制

重点是十进制、二进制、十六进制互换（程序中最常用的是十进制和十六进制）

- 3.数的表示
- 1）原码
- 2）反码
- 3）补码
- 4.数的运算：补码的加减法
- 5.运算（加减法）结果溢出概念和判断
- 6.BCD码的加减法
- 7.浮点数的表示（一般规格化表示，float,doubl的表示）

第1章 计算机基础知识
1.1数制及其转换
1.1.1各种计数制

1. 十进制（Decimal）

用0, 1, 2, ……9表示；

逢十进一；

人们习惯用的计数方式；

2. 二进制（Binary）

用0，1两个数码表示；

逢二进一；

计算机物理好实现（触发器的二状态）；

也是计算机唯一能识别的码制；

3. 八进制(Octal)

用0，1，……7表示；

逢八进一；

3位二进制数表示1位8进制 ；

老的12位、24位或36位的机器上，在UNIX系统中有使用。目前基本不怎么使用。

4. 十六进制(Hexadecimal)

用0，1，2，……，9，A, B, C, D, E, F等16个数码表示；

逢十六进一；

二进制数的简化表示方法（1位十六进制表示4位二进制）；

重点：A-F 16进制(10进制）

A (10)	D(13)
B (11)	E(14)
C (12)	F(15)

十进制	十六进制	二进制（8421）
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
10	A	1010
11	B	1011
12	C	1100
13	D	1101
14	E	1110
15	F	1111

5.数制使用

1）表示

	书面表达	汇编程序中
十进制:	$(1011)_{10}$ 或 1011D (1011)	1011D(1011)
二进制:	$(1011)_{2}$ 或 1011B	1011B
八进制：	$(1011)_{8}$ 或 1011Q	1011Q
十六进制:	$(1011)_{16}$ 或 1011H	1011H

- 2）各数制表示的值

所有数制按数制的权展开得到人们熟悉的十进制。

二进制基为2，八进制基为8，十六进制基为16

1011(D) 表示$1*10^3+0*10^2+1*10^1+1*10^0$

1011B 表示 $1*2^3+0*2^2+1*2^1+1*2^0$

1011Q 表示 $1*8^3+0*8^2+1*8^1+1*8^0$

1011H 表示$1*16^3+0*16^2+1*16^1+1*16^0$

- 3）读法

1234H

6.预备知识

计算机中数一般是8位，16位，32位,64位的二进制组成，我们先讨论8位和16位数。

所谓8位数就是由8位二进制数组成

D_7 D_6 D_5 D_4 D_3 D_2 D_1 D_0

1 1 1 0 1 0 1 0 (B)

D_0为最低位，D_7为8位的最高位

16位数由16位二进制数组成

$D_{15}\ D_{14}\ D_{13}\ D_{12}\ D_{11}\ D_{10}\ D_9\ D_8\ D_7\ D_6\ D_5\ D_4\ D_3\ D_2\ D_1\ D_0$

$$\underbrace{1\ 1\ 1\ 0\ 1\ 0\ 1\ 0}_{\text{高8位}}\ \underbrace{1\ 0\ 1\ 0\ 1\ 0\ 1\ 0}_{\text{低8位}}\ (B)$$

D_0为16位二进制数的最低位，D_{15}为16位二进制数的最高位

在计算机中由8位（bit）二进制数组成的数称为一个字节（BYTE）数据

1 1 1 0 1 0 1 0 (B) = EAH

$$\underbrace{1\ 1\ 1\ 0\ 1\ 0\ 1\ 0}_{\substack{\text{高字节} \\ \text{高8位}}}\ \underbrace{1\ 0\ 1\ 0\ 1\ 0\ 1\ 0}_{\substack{\text{低字节} \\ \text{低8位}}}\ (\mathrm{B}) = \mathrm{EAAAH}$$

1.1.2数制之间的转化

1. 二进制, 八进制， 十六进制→十进制

方法都是根据对应的基按权展开

1) 二进制→十进制

$$11001100B=1\times2^7+1\times2^6+0\times2^5+0\times2^4+1\times2^3+1\times2^2+0\times2^1+0\times2^0=204$$

00000000B= 0
01100100B= 100
01111111B= 127
10000000B= 128
11111111B= 255

8位二进制数表示的范围0-255

0000000000000000B=0
1000000000000000B=32768
1111111111111111B=65535

16位范围0-65535

二进制小数转十进制

$101.101B=1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$

$=5.625$

2）八进制→十进制

$127.56Q=1\times8^2+2\times8^1+7\times8^0+5\times8^{-1}+6\times8^{-2}$

3）十六进制→十进制

$2CA.B6\ H=2\times16^2+12\times16^1+10\times16^0+11\times16^{-1}$

$+6\times16^{-2}$

2. 十进制→二进制, 八进制， 十六进制

1）十进制→二进制

（1）整数

325= 101000101B

十进制整数转换成二进制数，只需一次次地用2去除，得到一序列余数，将其逆序排列，就是二进制所表示的数。

例如：将十进制数325转换成二进制数。

2 | 325

2 | 162……余数为 1　　最低位

2 | 81……余数为 0

2 | 40……余数为 1

2 | 20……余数为 0

2 | 10……余数为 0

2 | 5……余数为 0

2 | 2……余数为 1

2 | 1……余数为 0

0……余数为 1　　最高位

所以，转换结果为：325=101000101B。

*总结：除2取余

187= 10111011　B　(自己练习)

（2）小数

十进制的小数部分转换成二进制，只要不断地用2去乘该小数，直到其小数部分为零为止。每次得到整数0或1，从第一次得到的整数读起，就是二进制所表示的小数。

例如：将十进制小数0.6875转换成二进制小数。

0.6875 × 2=1.3750……整数部分为 1　最高位

0.3750 × 2=0.7500……整数部分为 0

0.75 × 2=1.50………整数部分为 1

0.5 × 2=1.0………整数部分为 1　最低位

因此，转换结果为 0.6875=0.1011B。

*总结：乘2取整

0.6875=0.1011B

思考：

0. 3 能否用8位二进制精确表示 ？

答：采用×2取整的方法，得到：0100110011001。。。循环下去，采用8位只能有效表示，

0. 3≈01001100B。

结论：不是所有数在计算机里都能精确表示，对于某些数，存储数的位数越长则精度越高。

2）十进制→十六进制

方法1　十进制→二进制→十六进制

127=01111111B=7FH

方法2 十进制除16求余

127÷16=7	余	15（F）	最低位
7÷16=0	余	7	最高位

127=7FH

3)十进制→八进制

方法1　十进制→二进制→八进制

方法2　十进制除8求余

练习题

1.计算机能唯一识别的数制是（）

A.二进制

B.八进制

C.十进制

D.十六进制

2.102=____B=_____H =___Q

3.11000011.1+6EH+45Q= ____D

4.AB1FH＋0EFCH =____H

5.1E0AH-2345H =____H

1.2 计算机中数的表示
1.2.1 无符号数

无符号： 计算机中数的所有位都表示数值

8位无符号二进制

01111111B=	127	7FH
10000000B=	128	80H
11111111B=	255	FFH

16位无符号二进制

0000000000000000B=0	0000H
1111111111111111B=65535	FFFFH

8位二进制无符号数表示范围：0-255。

16位二进制无符号数表示范围：0-65535。

N 位二进制无符号数表示范围： 0 ～ 2^N-1。

1.2 计算机中数的表示
1.2.2 符号数的表示方法

1.符号数

符号数:最高位表示符号（ 1表示负数， 0表示正数），剩下的其他位表示数值大小。

符号数有三种形式 ：

原码

反码

补码

2.原码

定义：二进制数的最高位为符号位(0表示正数,1表示负数),其余的位表示数值。

例如：

$[+100]_{原} = \underline{0}\ 1100100B$

$[+127]_{原} = \underline{0}\ 1111111B$

$[-100]_{原} = \underline{1}\ 1100100B$

$[-127]_{原} = \underline{1}\ 1111111B$

符号位　数值部分

8位二进制原码所有数值：

$\underline{0}$ 0000000　$\underline{0}$ 1111111 ～ $\underline{1}$ 0000000 $\underline{1}$ 1111111

正数　　　　　　　　负数

小 $\longrightarrow$ 大 ～ 大 $\longrightarrow$ 小

N位二进制数的原码可表示的数值范围:

$-(2^{N-1}-1) \sim +(2^{N-1}-1)$，共$2^N-1$ 个。

8位二进制数：　-127 ～ +127

- 8位数0的原码：+0=0 0000000

  −0=1 0000000

即：数0的原码不唯一。

原码的特点

- 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：
  - 计算机中用原码实现加减运算比较困难
  - 0的表示不唯一

3.反码

定义：正数的反码与原码相同。负数的反码是将原码的数值位按位求反。

对一个数值X：

- 若$X>0$，则 $[X]_{反}=[X]_{原}$
- 若$X<0$， 则 $[X]_{反}$和$[X]_{原}$的符号位相同，数值部分按位求反

例如：

$[+\ \ 0]_{反} = \underline{0}\ \ 0000000$

$[+100]_{反} = \underline{0}\ \ 1100100$

$[+127]_{反} = \underline{0}\ \ 1111111$

$[-\ \ 0]_{反} = \underline{1}\ \ 1111111$

$[-100]_{反} = \underline{1}\ \ 0011011$

$[-127]_{反} = \underline{1}\ \ 0000000$

符号位　数值部分

例如：8位二进制反码的数值范围

正数　　　　　　　　　　负数

小 ⟶ 大 ～ 小 ⟶ 大

0 0000000　0 1111111 ～ 1 0000000　1 1111111

+0　　127　　～　　-127　　-0

表示范围：N位二进制数的反码可表示的数值范围:

$-(2^{N-1}-1) \sim +(2^{N-1}-1)$，共$2^N-1$ 个。

$[+0]_{反}=\underline{0}\ 0000000$

$[-0]_{反}=\underline{1}\ 1111111$

即：数0的反码也不是唯一的。

4.补码

定义：正数的补码与原码、反码相同。负数的补码是将原码的数值位按位求反加一。

<u>对一个数值X：</u>

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

正数： $[+\ \ 0]_{补} = \underline{0}\ \ 0000000$

$[+100]_{补} = \underline{0}\ \ 1100100$

$[+127]_{补} = \underline{0}\ \ 1111111$

符号位　数值部分

负数：

求 $[--100]_{补}$ = ？

$[--100]_{原}$ =$\underline{1}$ 1100100 第1步

$[--100]_{反}$ =$\underline{1}$ 0011011 第2步

$[--100]_{补}$ =$\underline{1}$ 0011100 第3步

符号位 数值部分

$[--127]_{补}$ =$\underline{1}$ 0000001

符号位 数值部分

补码的意义

- 钟表为例

  假设时钟停在8点，要把时针拨到3点
- 两种拨法

  逆时针拨：　8-5=3

  顺时针拨：　8+7=12+3=3 -----以12为模
- 对模12，有：
- 8-5=8+7　　-5和7互为补数

  $[-5]_{补}=12-5=7$

  8-5=8+(-5)=8+(12-5) =8+7=12+3

补码的作用：

- 加减法可以用同一加法电路实现
- 可将符号位参与到数字的运算中来

拓展：

通用计算机中的乘除法用移位和加法来实现，DSP中有专门的乘法器。

补码的定义 (2)

- 若X>0， 则 $[X]_{补}=[X]_{原}=X$
- 若X<0， 则 $[X]_{补}$=模-|X|， N位二进制模为2^N

求补码的方法

- 正数

  正数补码等于原码

- 负数：

  （1）原码取反加1

  （2）经验方法

  （3）$2^N - |X|$

利用经验方法求负数补码

写出负数对应的正数，从最低位往高位看，遇到第一个‘1’，包括第一个‘1’数字照写不变，第一个‘1’以后的数字取反。

X=-2　　X对应正数为$\underbrace{000000}_{\text{取反}}10$ B

[X]补=　　11111110 B

X=-100　X对应正数为$\underbrace{01100}_{\text{取反}}100$ B

[X]补=　　10011100 B

利用$2^N - |X|$ 求负数补码

8位数X=-10，　　$[X]_{补}$=

$=2^8-10$

=100H-0AH=F6H

8位数X=-101，$[X]_{补}$=

$=2^8-101$

=100H-65H=9BH

特殊数字的补码：

$X=-128 \quad [X]_{补}=10000000B=80H$

$[X]_{补}=2^8-128$

$=100H-80H=80H$

0的补码

$[+0]_{补} = [+0]_{原} = 00000000$

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

$= 1\ 00000000$

所以 $[\ 0\ ]_{补} = 00000000$

认识补码表示的数

- 8位补码

00000000B～ 11111111B 二进制

00H～ FFH 十六进制

正数：

00000000B～01111111B 二进制

00H～ 7FH 十六进制

0 ～ 127 十 进制

负数：

10000000B～11111111B 二进制

80H～ FFH 十六进制

-128 ～ -1 十进制

- 16位补码

0000H～FFFFH

0000H～ 7FFFH （正数）

0 ～ 32767

8000H～ FFFFH （负数）

-32768 ～ -1

1.2.3 符号数（补码）运算

1. 补码运算

规则：

$$[X+Y]_{补}=[X]_{补}+[Y]_{补}$$

$$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$$

$$[[Z_{补}]]_{补}=Z$$

特点：

采用同一种电路实现加减法；

符号位当成数的一部分参与运算；

2 .举例

例1　X=-100，Y=1，X+Y

$$
\begin{array}{lr}
[X]_{补} & 10011100 \\
[Y]_{补} & +\ 00000001 \\ \hline
[X+Y]_{补} & 10011101B
\end{array}
\xrightarrow{真值} -1100011B=-99
$$

结果正确

例2 X=64　Y=10　求X-Y

例2，4 要用到$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

例3　X=-78，Y=-15，X-Y

例4 X=-30　Y=-100　求X+Y

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

例2 X=64，Y=10，求X-Y

$$
\begin{array}{lr}
[X]_{补} & 01000000 \\
[-Y]_{补} & +\ 11110110 \\ \hline
[X-Y]_{补} & \underline{1}\ \ 00110110B \xrightarrow{\text{真值}} 54
\end{array}
$$

虽然有进位（模自然丢失），结果是正确的

（此题为书上的P9 例1-3，书上有错误）

例3　X=-78，Y=-15，求X-Y

$$
\begin{array}{lr}
[X]_{补} & 10110010 \\
[Y]_{补} & +\ 00001111 \\ \hline
[X-Y]_{补} & 11000001B
\end{array}
\xrightarrow{\text{真值}} -0111111 = -63
$$

结果是正确的

例4 X=-30，Y=-100，求X+Y

$$
\begin{array}{lr}
[X]_{补} & 11100010 \\
[Y]_{补} & \underline{+\ 10011100} \\
[X+Y]_{补} & \underline{\color{red}{1}}\ 01111110B \xrightarrow{\text{真值}} 126
\end{array}
$$

两个负数相加结果是正数，运算结果错误。

3. 溢出相关问题

1）**定义：** 计算机中数的运算结果超出了对应数的表示范围

X=-30，Y=-100，X+Y

$[X]_{补}$ 11100010

$[Y]_{补}$ + 10011100

$[X+Y]_{补}$ 1 01111110B $\xrightarrow{真值}$ 126

X+Y= -130

X+Y的运算结果超出对应8位补码所能表示的范围：-128----+127，所以结果错误

2） 溢出的处理方法

将参与运算的数的位数拓展到更多位，如8拓展为16位。

把一个8位补码数拓展到16位补码，要保证其数字的值不变。

拓展方法

正数： 高八位 补8个0

负数 ： 高八位 补8个1

8 位 ⟶ 16位

正数：0100000B　　0000000000100000B

（十进制 64 ）　　（ 十进制 64）

负数：11111111B　　1111111111111111B

(十进制 -1)　　(十进制 -1)

$[X]_{补}$=FF80H，X= ___ D

$[X]_{补}$=FF23H，X= ___ D

X=-30，Y=-100，X+Y，采用16位二进制计算，并判断结果是否溢出。

$[X]_{补}$	1111111111100010
$[Y]_{补}$	+ 1111111110011100
$[X+Y]_{补}$	1 11111111 01111110B
真值	-0000000 10000010B=-130

结果不溢出，拓展到16位数后结果正确，解决了原来8位数据运算结果溢出的问题。

3）溢出的判断方法

1）运算结果的范围（人）

X=-30，Y=-100，X+Y

X+Y=　　-130

最终结果超出了8位符号数（补码）表示的范围: -128---127　，所以溢出。

拓展:

在软件设计中，程序设计者要注意处理数据的范围，定义合适范围的变量，以免结果溢出，导致结果错误。

（2）参与运算的数本身的符号（人）

(a)异号数相加不溢出；

(b)同号数相加可能溢出，结果和参与运算数的符号一致则不溢出，不同则溢出

例1 X=-100，Y=1，X+Y

$[X]_{补}$ 10011100

$[Y]_{补}$ + 00000001

$[X+Y]_{补}$ 10011101B 真值 -1100011B=-99

异号数相加，不溢出， 结果正确

例2　X=64，Y=10，X-Y

$[X]_{补}$ 01100100

$[-Y]_{补}$ + 11110110

$[X-Y]_{补}$ 1 00110110B 真值 54

异号数相加（同号相减），不溢出，　结果正确

例3　X=-30，Y=-100，X+Y

$[X]_{补}$ 11100010

$[Y]_{补}$ + 10011100

$[X+Y]_{补}$ 1 01111110B 真值 126

负数加负数结果为正数，溢出，　结果不正确

例4 X=-78，Y=-15，X-Y

$$
\begin{array}{lr}
[X]_{补} & 10110010 \\
[Y]_{补} & +\ 00001111 \\
\hline
[X-Y]_{补} & 11000001B
\end{array}
\quad 真值 \quad -63
$$

异号数相加（同号相减），不溢出，结果正确

（3）通过最高和次高位的进位的异或关系来判断（计算机）

异或操作：

$$0 \oplus 0 = 0,\ 0 \oplus 1 = 1,\ 1 \oplus 0 = 1, 1 \oplus 1 = 0$$

8位由 $C_{D7} \oplus C_{D6} = ?$ 1溢出，0不溢出

16位由 $C_{D15} \oplus C_{D14} = ?$ 1溢出，0不溢出

a)X=-30，Y=-100，X+Y

C_{D7} C_{D6}

$[X]_{补}$	1 1 1 0 0 0 1 0
$[Y]_{补}$	+1 0 0 1 1 1 0 0
$[X+Y]_{补}$	<u>1</u> 0 1 1 1 1 1 1 0B

$C_{D6} = 0, C_{D7} = 1$

$C_{D7} \oplus C_{D6} = 1$ ，所有结果溢出，不正确

b)X=-30，Y=-100，X+Y

C_{D15} C_{D14}

$[X]_{补}$ 1111111111100010

$[Y]_{补}$ + 1111111110011100

$[X+Y]_{补}$ 1 11111111 01111110B

$$C_{D14}=1,\ C_{D15}=1$$

$C_{D15} \oplus C_{D14}=0$，所以结果不溢出，正确

溢出的判断方法总结

（1）通过最高和次高位进位的异或来判断（计算机）

异或为1　溢出

异或为0　不溢出

（2）参与运算的数本身的符号（人）

(a)异号数相加（同号数相减）肯定不溢出；

(b)同号数相加可能溢出，结果的符号和参与运算的数的符号一致则不溢出，不同则肯定溢出

（3）运算结果是否超过了表示范围（人）

1.3 十进制数（BCD码）和ASCII

1.3.1 BCD码

1、BCD (Binary Code Decimal)

定义：用二进制编码表示十进制数，每个十进制数用4位二进制数表示。

1234的BCD 书写记为 $(0001\ 0010\ 0011\ 0100)_{BCD}$

1234的BCD 程序写成 0001 0010 0011 0100 B

或 1234H （常用）

```
MOV   AX,1234H;
MOV   AX, 1234 (4D2H)
```

标准BCD码表示法

十进制数	标准BCD码	
0	0000	
1	0001	
2	0010	
3	0011	
4	0100	
5	0101	
6	0110	
7	0111	
8	1000	
9	1001	
10	0001 0000	
11	0001 0001	
63	0110 0011	

BCD码中大于9的1010----1111（A---F）是不可用

2.BCD码（加法）计算

BCD码计算9+6=？

	0000 1001	09H ；9的BCD码
+	0000 0110	06H ；6的BCD码
	0000 1111	0FH

F（1111）为非法BCD表码，正确BCD码结果应该为15，用15H表示。

错误原因：十进制加法‘逢十进一’，二进制加法按‘逢二进一’，十六进制（四位二进制）‘逢十六进一’，所有当数据位出现大于9时要进行加6调整。

BCD码计算9+6=？

	0000 1001	09H	
+	0000 0110	06H	；二进制加法
	0000 1111	0FH	；二进制数
+	0000 0110	06H	；调整
	0001 0101	15H	；15的BCD码

BCD码计算 27+82= ?

	0010 0111	27H；27的BCD
+	1000 0010	82H；82的BCD
	1010 1001	A9H ；A大于9需要调整
+	0110 0000	60H
(1)	0000 1001	(1)09H;109是结果

BCD码计算46+74= ?

	0100 0110	46H；46的BCD
+	0111 0100	74H；74的BCD
	1011 1010	BAH；高低都有非法码
+	0110 0110	66H；高低位都需要调整
(1)	0010 0000	(1)20H; 120是正确结果

BCD码计算88+89=？

1000 1000　　88H；88的BCD

+　1000 1001　　89H；89的BCD

(1) 0001 0001　(1)11H；高低四位都有进位

+　0110 0110　　66H；高低位都需要调整

(1) 0111 0111　(1)77H; 177是正确结果

BCD码 调整规则

- 1）当低四位出现A-F数时，需+06H调整
- 2）当低四位向高四位进位，需+06H调整
- 3）当高四位出现A-F数时，需+60H调整
- 4）当高四位有进位，需+60H调整
- 5）高低位同时出现上面情况，需+66H调整

拓展：

由于BCD码符合人们的使用数字的习惯，在实际系统的人机接口中经常使用。

虽然BCD运算在计算机上有专用的指令完成运算过程，包括调整，能得到需要的十进制结果。但是实际计算机系统是按二进制来进行运算的，每次BCD码计算都需要调整，程序设计繁琐且效率低，所以通常情况为了得到最终结果数据的BCD码，很少整个运算过程中使用BCD码运算。而是采用常规的二进制运算，只是将需要显示和交互的最终数据转化为BCD码。

1.3.2 ASCII码

ASCII码

美国信息交换标准代码（American Standard Code for Information Interchange ）

字符的ASCII码用一个字节（7位）来表示。

数字字符 ‘0’---‘9’的 ASCII码 为 30H---39H

小写字母 'a'---'z' 的 ASCII码 为 61H---7AH

（61H+19H=7A）

大写字母 'A'---'Z'的 ASCII码 为 41H---5AH

对应的大小写字母的ASCII码相差20H

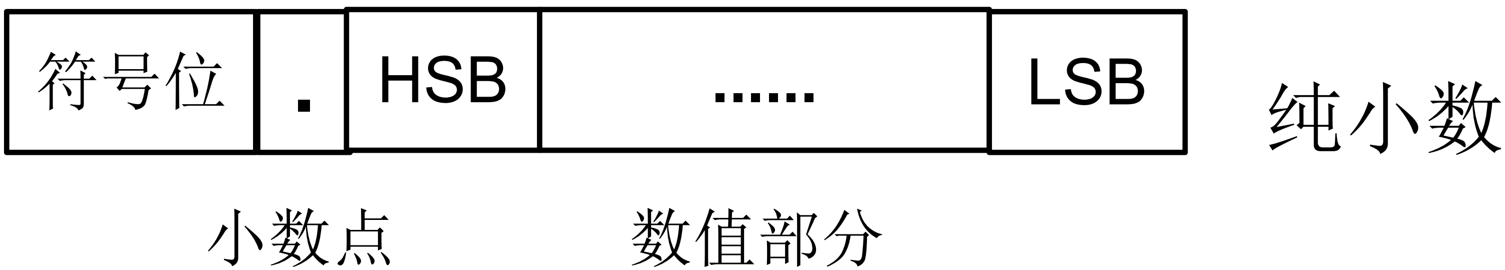
ASCII字符表(7位码)

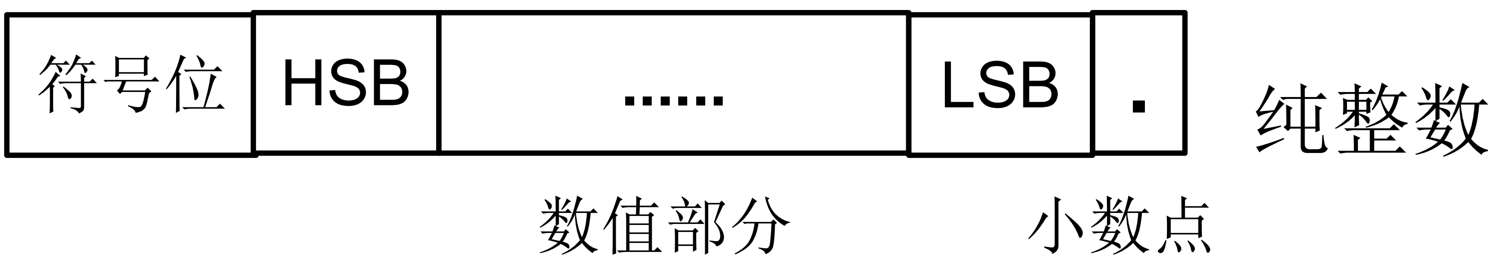
LSD		MSD 0	1	2	3	4	5	6	7
		000	001	010	011	100	101	110	111
0	0000	NUL	DEL	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	′	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
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	}
E	1110	SO	RS	.	>	N	{	n	~
F	1111	SI	US	/	?	O	←	o	DEL

1.4 定点数和浮点数

1.定点数的表示

数的小数点固定在一个位置不变。





1）带符号的定点小数（纯小数）

约定所有数的小数点位置固定在符号位之后

0	.	1 1 1 1 1 1 1
符号数	小数点	数值部分

最大正数

1	.	1 1 1 1 1 1 1
符号数	小数点	数值部分

最小负数

字长为8位的，纯小数可表示范围：

$-(1-2^{-7}) \sim 1-2^{-7}$

2）带符号的定点整数（纯整数）

约定所有数的小数点位置固定所有数值位之后

符号位	数值部分	小数点	
1	0 0 0 0 0 0 0	.	最大正数

符号位	数值部分	小数点	
1	1 1 1 1 1 1 1	.	最小负数

字长为N位的，纯小数可表示范围：

$$-（2^{N-1}-1) \sim 2^{N-1}-1$$

2.浮点数的表示

1101.101 可以写成：1.101101×2^3

0.1101101×2^4

0.01101101×2^5

二进制可表示：$N = M \times 2^E$

E_f	E	M_f	M
阶符	阶码位	尾符	尾数位
阶码		尾数	

阶符：表示阶码的正负，0 为正，1为负

尾符：表示尾数的正负，0 为正，1为负

- 规格化表示方法：小数的第一位为1(确保了唯一性）。

例1　阶码为7位　，尾数为9（二者均包括符号位），用标准格式化表示 1101.101B 。（共16位）

1101.101 规格化表示：0.1101101×2^4

E_f	……	M_f	……
阶符	阶码位	尾符	尾数位
0	***000100***	***0***	***11011010***

3.引入浮点数表示的意义

8位的原码二进制数

定点数 11111111~01111111

整数： -127 127 精度： 1

小数 $-(1-2^{-7})$ $(1-2^{-7})$ 精度：2^{-7}

浮点数： 5位阶码 + 3位尾码

01111111~01111011

E_f	E	M_f	M
阶符	阶码位	尾符	尾数位
阶码		尾数	

E_f	E	M_f	M

$\underbrace{\text{阶符} \qquad \text{阶码位}}_{\text{阶码}} \qquad \underbrace{\text{尾符} \qquad \text{尾数位}}_{\text{尾数}}$

浮点数： 5位阶码 ＋ 3位尾数

01111111~01111011

范围：$2^{15}\times(-0.75)\sim 2^{15}\times 0.75$

精度：$11111001 = 2^{-15}\times 0.25$

相同位数时，浮点数比定点数表示的范围更大，精度更高！

3. *IEEE754* 表示的 *Float* 型（和*Double*型）

1）Float 型　表示带小数的实数，用 4字节表示

D_{31}	D_{30}　　D_{23}	D_{22}　　D_0
1bit	8bit	23bit
符号位	阶码	尾数

1.符号位(Sign) : 0代表正，1代表为负 。

2.阶码（Exponent）:用于存储科学计数法中的指数数据，并且采用移码存储 。

3.尾数（Mantissa）：尾数部分 。

D_{31}	D_{30} — D_{23}	D_{22} — D_0
1bit	8bit	23bit
符号位	阶码	尾数

1.符号位：0代表正，1代表为负 。

2.阶码:阶码应该可正可负的（-127—+127），以127为0，没有符号位，正负数加上127再去表示。

3.尾数：将数化为 整数位为1的形式，即整数只有一位‘1’，小数点后为尾数。如：1.1001

-8.25= -1000.01B=-1.00001 $\times 2^3$***B***

D_{31}	D_{30} — D_{23}	D_{22} — D_0
1bit	8bit	23bit
符号位	阶码	尾数
1	*10000010*	*00001000000000000000000*
	(127+3)	

-8.25D对应的浮点数在内存中为： C1040000H

$$120.5 = 1111000.1B = 1.1110001 \times 2^{6}B$$

D_{31}	D_{30} … D_{23}	D_{22} … D_0
1bit	8bit	23bit
符号位	阶码	尾数
0	*10000101*	*11100010000000000000000*
	(127+6)	

120.5D对应的浮点数在内存中为：42F10000H

0. 3≈ 0.01001100110011001。。。

1. 00110011001100110011001×2^{-2}

D_{31}	D_{30} … D_{23}	D_{22} … D_0
1bit	8bit	23bit
符号位	阶码	尾数
0	01111101	00110011001100110011001
	（127-2）	

0.3的浮点数　3E999999H

DOUBLE