第二讲 计算机的数据

华中科技大学
电子信息与通信学院
School of Electronic Information and Communications

学习目标

- 不同数制的转换、数据在计算机系统的表示方式、数据的存储方式，包括大字节序和小字节序的区别
- 计算机数据运算的基础知识，包括符号数、无符号数、定点数、浮点数的加减运算规则

1.5 计算机中的信息表示

- 数制互换

2进制	10进制	16进制	2进制	10进制	16进制
0000B	0	0x0	1000B	8	0x 8
0001B	1	0x 1	1001B	9	0x 9
0010B	2	0x 2	1010B	10	0x a
0011B	3	0x 3	1011B	11	0x b
0100B	4	0x 4	1100B	12	0x c
0101B	5	0x 5	1101B	13	0x d
0110B	6	0x 6	1110B	14	0x e
0111B	7	0x 7	1111B	15	0x f

$$n = \sum_i A_i B^i$$

十进制->其他进制

- 整数部分：采用除B取余法
  - 即将10进制整数除以B，得到一个商数和余数，再将商数除以B又得到一个商数和余数，如此继续除下去直至商为0为止。以最后所得的余数作为最高位，将各次所得的余数按“后得先排”的顺序写下来即得B进制转换结果。
- 小数部分：采用乘B取整法，
  - 即将10进制纯小数乘以B，摘取乘积中的整数后保留小数部分再乘B，如此继续下去直至乘积小数部分为零或者得到要求的精度为止。将各次摘取的整数依先后顺序写出来即为转换的B进制纯小数结果。

例1. 1将10.5转换为2进制数

- 首先取整数部分10

2	10	0
2	5	1
2	2	0
2	1	1
	0	

整数部分转换得到1010B

- 取小数部分0.5

0.5	
* 2	
1.0	1

小数部分转换得到0.1B

把整数部分和小数部分合并就得到了整个结果，即10.5=1010.1B

整数在计算机中的表示

- 计算机中常用的数据为8、16、32或64位（b）。
  - 8位二进制数称为字节（B），
  - 16位二进制数称为半字（h），
  - 32位二进制数称为字（w），
  - 64位二进制数称为双字（dw）
- 计算机中的数区分为符号数和无符号数
  - 无符号数所有二进制位都表示数值 $$数值=\sum_{i=0}^{n-1} b_i 2^i$$
  - 符号数采用数的符号和数值部分一起编码的方法来表示，最高位表示符号，正号用“0”表示，负号用“1”表示

- 符号数编码方法有原码、反码和补码
  - 正数的原码、反码和补码表示法都是最高位表示符号位，其余位表示数值位，即原码的表示方式。
  - 负数的反码在原码的基础上将所有数值位取反，符号位不变；负数的补码将反码的最低位加1获得。
- 符号数在计算机中采用补码表示
  - 补码表示的n位数的数值

$$\text{数值} = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i$$

- 例1.3 -105在计算机中采用8位、16位、32位不同类型的补码数表示结果如下:
  - -105=1001 0111B=0x97（8位），
  - -105=1111 1111 1001 0111B=0xff97(16位),
  - -105=1111 1111 1111 1111 1111 1111 1001 0111B=0xffffff97（32位）

小数在计算机中的表示

- 定点数（fixed point）
  - 约定小数点隐含在某一个固定位置上，

十进制整数195的定点表示

十进制纯小数-0.6876的定点表示

- 浮点数（floating point）
  - 小数点位置可以浮动

IEEE754标准定义的浮点数格式

- 规范的（Normalized）浮点数表达方式

$$\pm 1.\mathrm{dd...d} \times \beta^{\pm e}$$

IEEE 单精度浮点数

符号 Sign	指数 Exponent	尾数 Mantissa
1 bit	8 bits	23 bits

IEEE 双精度浮点数

符号 Sign	指数 Exponent	尾数 Mantissa
1 bit	11 bits	52 bits

单精度数的偏差值为 127，而双精度数的偏差值为 1023

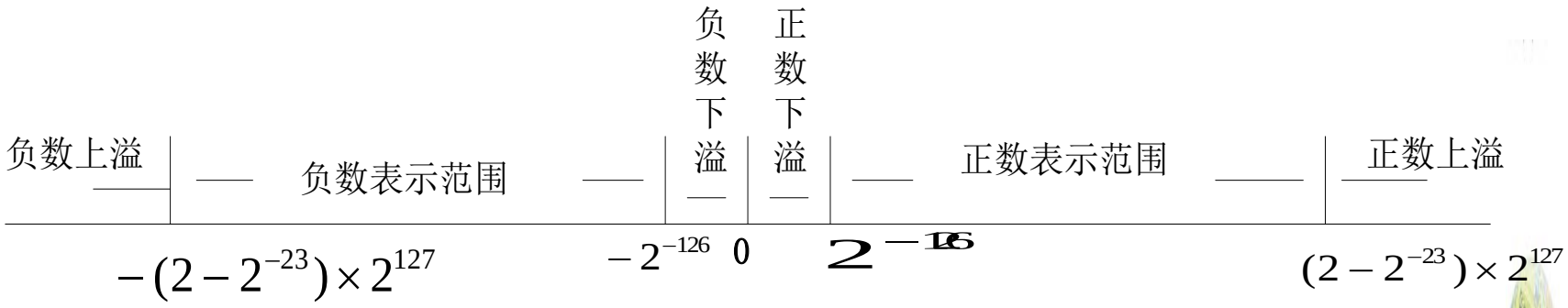
规范浮点数意味着尾数的小数点左侧为 1，在保存尾数的时候，省略小数点前面这个 1

- 例1. 4将实数 -9.625 表达为单精度的浮点数格式。
  - 9.625完整的二进制形式 1001.101
  - 规范浮点数表达为 1.001101×2^3
  - 负数，所以符号域为 1
  - 指数为 3，所以指数域为 3 + 127 = 130，即二进制的 10000010
  - 尾数省略掉小数点左侧的 1 之后为 001101，右侧用零补齐
  - 最终结果为：
    - 1 10000010 00110100000000000000000
    - 0xC11A0000

1100 0001 0001 1010 0000 0000 0000 0000
C 1 1 A 0 0 0 0

- 注意
  - 不断将产生的小数部分乘以 2 的过程掩盖了一个事实：很多小数不能经过有限次这样的过程而得到精确的结果（比如最简单的 0.1）
  - 浮点数尾数域的位数是有限的，为此，浮点数的处理办法是持续该过程直到由此得到的尾数足以填满尾数域，之后对多余的位进行舍入。换句话说，十进制到二进制的变换并不能保证总是精确的，而只能是近似值。
  - 只有很少一部分十进制小数具有精确的二进制浮点数表达

单精度数表示范围

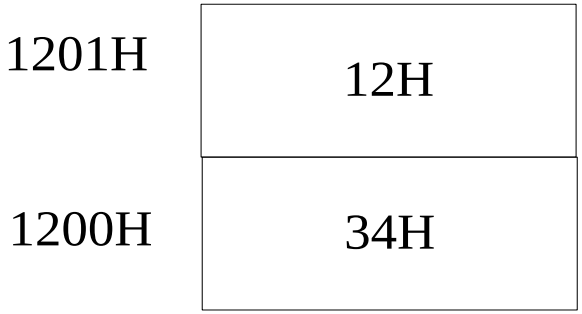


- 上溢表示超出计算机所能表示的数的绝对值的最大值，
- 下溢表示超出计算机所能表示的数的绝对值的最小值

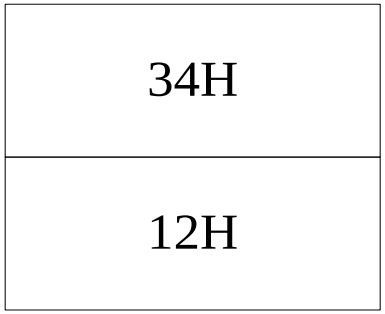
数的存储

- 存储器以字节为单位存储数据，
- 多个字节的数据采用连续的多个存储单元存放
- 在计算机系统中存在两种不同的数据存储方式：
  - 大字节序（big endian）：
    - 最高字节存储在最低地址的存储单元，而最低字节存放在最高地址的存储单元上。
    - PowerPC 、SPARC、Mortolora系列和绝大多数的RISC处理器。
  - 小字节序（little endian）：
    - 最高字节存储在最高地址的存储单元，而最低字节存放在最低地址的存储单元上。
    - 采用这种机制的处理器有intel架构的CPU (Intel、AMD)。

- 一个16位的数据0x1234在两种不同存储机制下存放到地址为0x1200的存储器中的存储映像。



小字节序存储映像



大字节序存储映像

字符在计算机中的表示

- 不同类型的文字符号、图片、视频、音频等，都需要经过某种编码转化成为计算机能够识别的二进制信息。
- ASCII(American Standard Code for Information Interchange)码，即美国标准信息交换码，这是一种7位的2进制编码，可表示128个字符，包括英文大小写字母与数字0～9，表1-2列出了全部ASCII码及其表示方法。

1.6 计算机运算基础

- 无符号数运算
  - 两个无符号数相加，由于两个加数均为正数，因此其和也是正数。当和超过其位数所允许的范围时，就向更高位进位。

```
  0111 1111
+ 1010 0000
-----------
1 0001 1111 = 287
↑
└ 进位
```

  - 两个无符号数相减，被减数大于或等于减数，无借位，结果为正；被减数小于减数，有借位，结果为负

```
  1100 0000
- 0000 1010
-----------
  1011 0110 = 182
```

```
  0000 1010
- 1100 0000
-----------
1 0100 1010 = -10110110B = -182
↑
└ 借位
```

符号位（借位）扩展为16位补码
1111 1111 0100 1010

符号数运算

- n位二进制补码数，除去一位符号位，还有n-1位表示数值，所能表示的补码的范围为：-2^{n-1}～$(2^{n-1}-1)$。如果运算结果超过此范围就会产生溢出。

```
  0110 1001
+ 0011 0010
-------------
  1001 1011= 155 或  -101
```

```
  1001 0111
+ 1100 1110
-------------
1 0110 0101
↑
└ 进位
```

- 溢出的判断

设符号位的进位为CY，数值部分向符号位的进位为CS，则溢出的逻辑表达式：OF=CY⊕CS

浮点数运算

- 操作过程
  - 0 操作数检查
  - 比较指数大小，并完成指数对齐
    - 向指数大的对齐，尾数右移（小数点左侧的1补上）
  - 尾数求和运算（小数点左侧的1补上）
  - 结果规范化
  - 舍入处理
  - 溢出处理

例1. 5 试采用单精度浮点数计算(1.0 + 123456.789e30) + (-123456.789e30)和
1.0 + (123456.789e30 + (- 123456.789e30))的结果

- 1.0表示为单精度浮点数为：0x3f80 0000
  - 0 011 1111 1 000 0000 0000 0000 0000 0000
  - 阶码域0x7f 尾数域0符号0
- 123456.789e30表示为单精度浮点数为：0x79be371e
  - 0 111 1001 1 011 1110 0011 0111 0001 1110
  - 阶码域0xf3 尾数域0x3e371e符号0
- -123456.789e30表示为单精度浮点数为：0xf9be371e
  - 1 111 1001 1 011 1110 0011 0111 0001 1110
  - 阶码域0xf3 尾数域0x3e371e符号1

- (1.0 + 123456.789e30) + (-123456.789e30)
  - 1.0 + 123456.789e30
    - 1.0阶码域0x7f, 123456.789e30阶码域0xf3
    - 1.0尾数需右移0xf3-0x7f=0x74=116位才能实现阶码对齐，此时小数点左侧的1根据舍入原则已经丢弃，因此尾数1.0在移位之后变为0
    - 加法运算之后和的尾数即123456.789e30尾数域0x3e371e
    - 得到的规范化结果为123456.789e30：0x79be371e
  - 123456.789e30+ (-123456.789e30)
    - 两数据仅符号位不同，尾数运算时直接抵消
    - 规范化之后的结果为0
- 注意：此时计算机得到的结果为错误的结果

- 1.0 + (123456.789e30 + (- 123456.789e30))
  - (123456.789e30 + (- 123456.789e30))
    - 两数据仅符号位不同，尾数运算时直接抵消
    - 规范化之后的结果为0
  - 1.0+0
    - 0为特殊数值，不需要进行对阶，直接得到结果1.0
- 此结果为正确结果
- 当参与运算的两数阶码相差较大，尾数移位可能带来较大误差

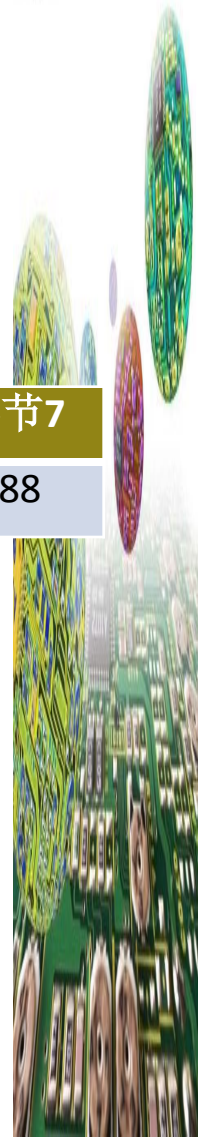
1.7 C语言数据类型的含义

- 典型的数据类型对应32位机的数据位宽

数据类型	字节数
char	1
short int	2
int	4
long	4
char *	4
float	4
double	8

- 例1.6某大字节序的计算机系统内存中存放有如表1-6所示数据，试指出地址0x1234 5678表示的char，short int，int，float，double型数据符号数和无符号数的十进制值？

地址	字节0	字节1	字节2	字节3	字节4	字节5	字节6	字节7
0x1234 5678	0x81	0x82	0x83	0x84	0x85	0x86	0x87	0x88



- 整形数据转换为十进制结果

数据类型	16进制值	10进制值
Char	0x81	-127
Unsigned char	0x81	129
Short int	0x8182	-32382
Unsigned short	0x8182	33154
int	0x81828384	-2122153084
unsigned	0x81828384	2172814212

- Float 型的数据16进制形式为0x81828384，二进制为：
  - 1 00000011 00000101000001110000100
    - 符号域为1，表示负数
    - 指数域为0000 0011，指数为3-127=-124
    - 尾数域为0000 0101 0000 0111 0000 100 ，实质尾数为1.00000101000001110000100，
  - 因此其十进制数为
    - -（1.00000101000001110000100B）×2^{-124}=-4.7943174E-38

- Double型的数据16进制形式为0x8182838485868788，二进制形式为：
  - 1 00000011000 0010100000111000010010000101100001101000011110001000
    - 符号域为1，表示负数
    - 指数域为00000011000，指数为24-1023=-999
    - 尾数域为0010100000111000010010000101100001101000011110001000，实质尾数为
    - 1. 0010100000111000010010000101100001101000011110001000
  - 因此其十进制数为：
    - -1. 0010100000111000010010000101100001101000011110001000B×2^{-999}= -2.1597750994171683E-301

小字节序内存数据类型实验

- PC VC++

```
char *c1;
unsigned char *c0;
int *i1,*i2;
unsigned int *i0;
short int *s1;
unsigned short *s0;
float *f1;
double *d1;
d1=(double *)malloc(8);
c1=(char *)d1;
c0=(unsigned char *)d1;
i1=(int *)d1;
i0=(unsigned int *)d1;
s1=(short int *)d1;
s0=(unsigned short *)d1;
f1=(float *)d1;
i2=(int *)(i1+1);
```

f1	0x003b1000
	-3.09178e-036
hAccelTable	0xcccccccc
msg	{msg=0xcccccccc wp=0xcc lp=0xcccccccc}
i2	0x003b1004
	-2004384123
i1	0x003b1000
	-2071756159
i0	0x003b1000
	2223211137
d1	0x003b1000
	-1.4249914579614907e-267
s1	0x003b1000
	-32127
c1	0x003b1000 "当儀庠噲 瓠韩弧弧弧
	-127 '?
s0	0x003b1000
	33409
c0	0x003b1000 "当儀庠噲 瓠韩弧弧弧
	129 '?

Address: 0x003b1000

```
003B1000  81 82 83 84 85 86  当儀庠
003B1006  87 88 FD FD FD FD  噲
```

f1	0x003b1000
	-3.09178e-036
hAccelTable	0xcccccccc
msg	{msg=0xcccccccc wp=0xcccc lp=0xcccccccc}
i2	0x003b1004
	0x88878685
i1	0x003b1000
	0x84838281
i0	0x003b1000
	0x84838281
d1	0x003b1000
	-1.4249914579614907e-267
s1	0x003b1000
	0x8281
c1	0x003b1000 "当儀庠噲 瓠韩弧弧弧
	0x81 '?
s0	0x003b1000
	0x8281
c0	0x003b1000 "当儀庠噲 瓠韩弧弧弧
	0x81 '?

运算

```
unsigned int x,y,z,u;
x=1;
y=4294967295;
z=x+y;
u=x-y;
int a,b,c,d;
a=2147483647;
b=-2147483648;
c=a+1;
d=b-1;
```

Name	Value
a	0x7fffffff
b	0x80000000
c	0x80000000
d	0x7fffffff
x	0x00000001
y	0xffffffff
z	0x00000000
u	0x00000002

Watch1 | Watch2 | Watch3 | Watch4

Name	Value
a	2147483647
b	-2147483648
c	-2147483648
d	2147483647
x	1
y	4294967295
z	0
u	2

Watch1 | Watch2 | Watch3 | Watch4

作业

- 10,11,12