南京大学
NANJING UNIVERSITY

数据的机器级表示

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

数据的机器级表示

- **数制和编码**
- 整数的表示
- 实数的表示
- 非数值数据的编码表示
- 数据的宽度和存储

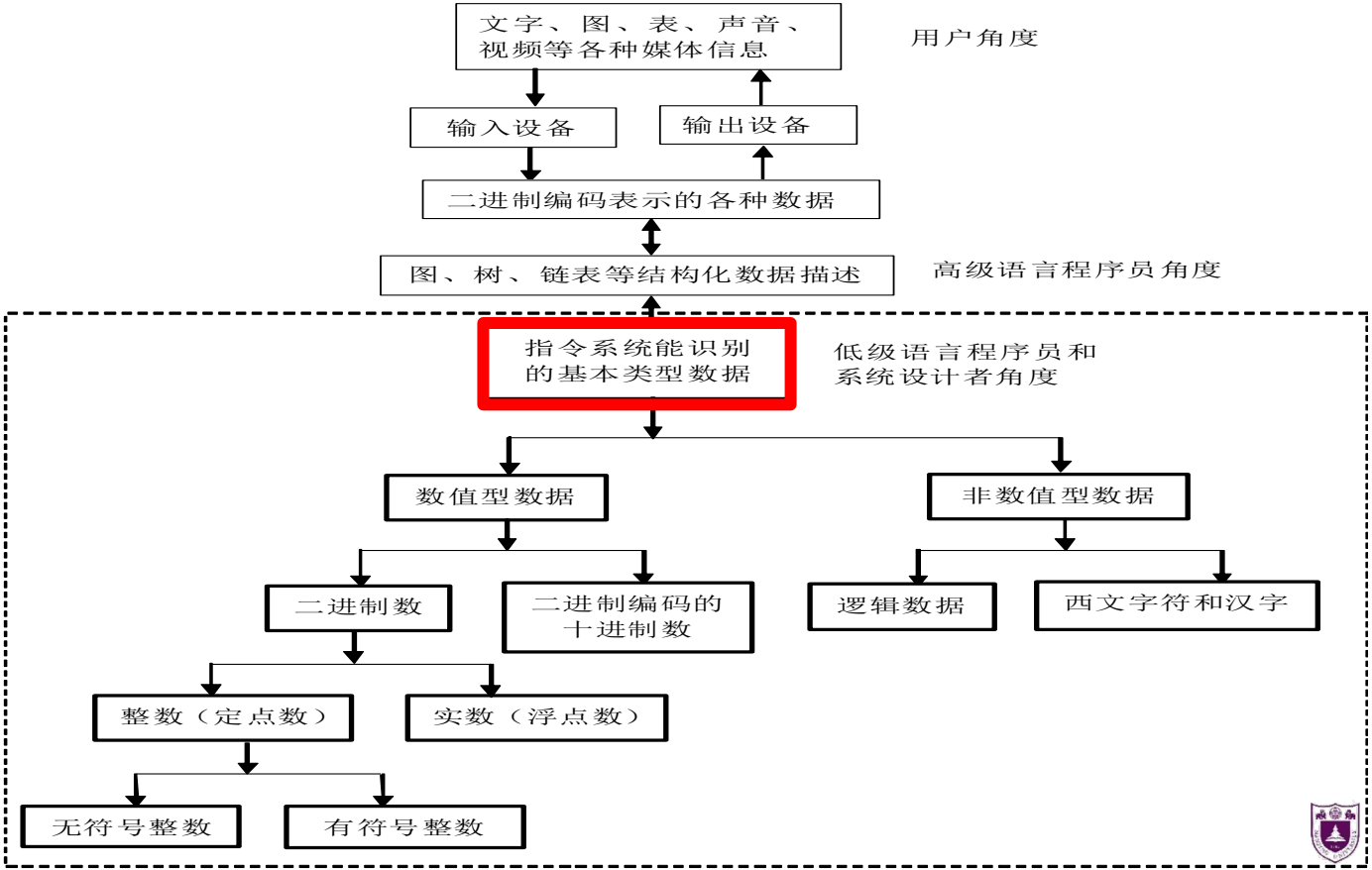
数制和编码

- **信息的二进制编码**
  - 计算机的**外部信息与内部机器级数据**
    - 外部：从不同处理角度，数据有不同形态
    - **内容**：所有信息都用**二进制（即：0和1）进行编码**
  - 用**二进制编码**的原因：
    - 制造二个稳定态的物理器件容易
    - 二进制编码、计数、运算规则简单
    - 正好与逻辑命题对应，便于逻辑运算，并可方便地用逻辑电路实现算术运算

数制和编码

- **信息的二进制编码**
  - 机器级数据分两大类：
    - **数值数据**
      - **整数**：无符号整数、带符号整数；定点数表示
      - **实数**：浮点数表示
    - **非数值数据**：逻辑数（包括位串）、西文字符和汉字
  - 对于给定的0/1序列，在未确定采用的**进位计数制**、**定/浮点表示**、**编码规则**之前，它的值是无法确定的。

数制和编码



数制和编码

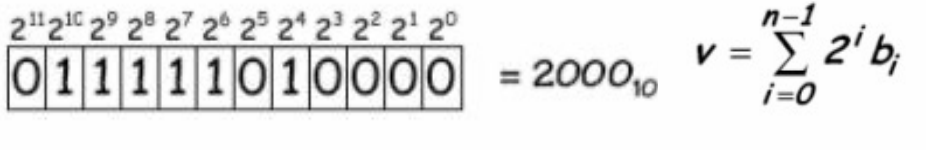
- **进位计数制**
  - **十进制数**：$D=d_n d_{n-1}\cdots d_1 d_0 . d_{-1} d_{-2}\cdots d_{-m}$ （m，n 为正整数）

    其数值为：$V(D)=d_n\times 10^n+d_{n-1}\times 10^{n-1}+\cdots+d_1\times 10^1+d_0\times 10^0+d_{-1}\times 10^{-1}+d_{-2}\times 10^{-2}+\cdots+d_{-m}\times 10^{-m}$

    $5836.47=5\times 10^3+8\times 10^2+3\times 10^1+6\times 10^0+4\times 10^{-1}+7\times 10^{-2}$
  - **二进制数**：$(100101.01)_2=1\times 2^5+0\times 2^4+0\times 2^3+1\times 2^2+0\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}=(37.25)_{10}$
  - **R进制数**：采用R个基本符号(0, 1, 2, …, R-1)，逢R进一，第i位的权是R^i，R成为该数字系统的基数。

数制和编码

- **进位计数制**
  - **八进制、十六进制**
    - **计算机用二进制表示所有信息！**
      为什么要引入 8 / 16进制？
      - 8 / 16进制是二进制的简便表示。便于阅读和书写！
      - 它们之间对应简单，转换容易。
      - 在**机器内部用二进制**，**在屏幕或其他外部设备上表示时，转换为10进制或8/16进制数**，可缩短长度
    - 早期有用8进制数简便表示2进制数
    - 现在基本上都用**16进制数表示机器数**



$2^3=8$ $2^4=16$

Octal - base 8	Hexadecimal - base 16	
000 - 0	0000 - 0	1000 - 8
001 - 1	0001 - 1	1001 - 9
010 - 2	0010 - 2	1010 - a
011 - 3	0011 - 3	1011 - b
100 - 4	0100 - 4	1100 - c
101 - 5	0101 - 5	1101 - d
110 - 6	0110 - 6	1110 - e
111 - 7	0111 - 7	1111 - f

一个8进制数字用3位二进制数字表示

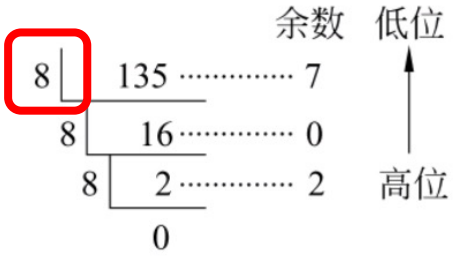
一个16进制数字用4位二进制数字表示

数制和编码

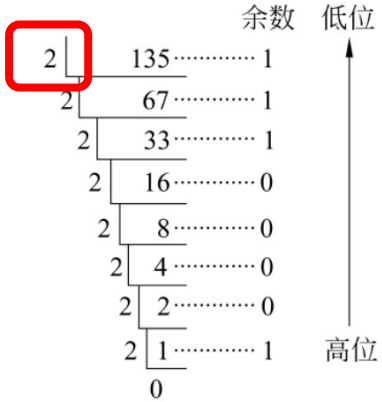
- **进位计数制**
  - 计算机系统中使用的几种进位记数制：
    - **二进制**R=2, 基本符号0, 1.
    - **八进制**R=8, 基本符号0, 1, 2, 3, 4, 5, 6, 7.
    - **十进制**R=10，基本符号0, 1, 2, 3, 4, 5, 6, 7, 8, 9.
    - **十六进制**R=16，基本符号0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.
  - 二进制数，用**B**表示，如10011B
  - 八进制数，用**O**表示，如1246O
  - 十进制数，用**D**表示（或省略），如56D或56
  - 十六进制，用**H**表示，如308FH，3C.5H
  - 或者用一个**下标表示数的基**：$11001_2 = 25_{10}$，11001B= 25

数制和编码

- **R进制数转换成十进制数：按权展开**
  - **二进制数：** $10101.01B=1\times2^4+0\times2^3+1\times2^2+0\times2^1+1\times2^0+0\times2^{-1}+1\times2^{-2}=21.25$。
  - **十六进制数：** $3A.CH=3\times16^1+10\times16^0+12\times16^{-1}=58.75$。
- **十进制数转换成R进制数：整数和小数部分分别转换**
  - **整数部分转换：除基取余，上低下高**



135=207O



135=1000 0111B

数制和编码

- **十进制数转换成R进制数**：整数和小数部分分别转换
  - 小数部分转换：乘基取整，上高下低

0.6875×2=1.375	整数部分=1	（高位）	0.6875=0.1011B
0.375×2=0.75	整数部分=0	↓	
0.75×2=1.5	整数部分=1		
0.5×2=1.0	整数部分=1	（低位）	
0.6875×8=5.5	整数部分=5	（高位）	0.6875=0.54O
0.5×8=4.0	整数部分=4	（低位）	
0.63×2=1.26	整数部分=1	（高位）	0.63=0.1010...B (近似值)
0.26×2=0.52	整数部分=0	↓	
0.52×2=1.04	整数部分=1		
0.04×2=0.08	整数部分=0	（低位）	

数制和编码

- **十进制数转换成R进制数：整数和小数部分分别转换**
  - 整数部分转换：除基取余，上低下高
  - 小数部分转换：乘基取整，上高下低

实际按简便方法先转换为二进制数，再按需转换为8/16进制数

整数：2、4、8、16、…、512、1024、2048、4096、…、65536

小数：0.5、0.25、0.125、0.0625、0.03125、……

例：4123.25=4096+16+8+2+1+0.25=1 0000 0001 1011.01B = $(101B.4)_{16}$

4023=(4096-1)-64-8=1111 1111 1111B-100 0000B-1000B

=1111 1011 0111B = FB7H = $(FB7)_{16}$

数制和编码

- **二、八、十六进制的相互转换：将每一个十六进制、八进制数字改写成等值的4位或3为二进制数，保持高低位次序不变。**

  - **十六进制转二进制：** 2B.5EH＝0010 1011.0101 1110B＝101011.0101111B。

  - **二进制转十六进制：** 11001.11B＝0001 1001.1100B＝19.CH；

    整数部分高位补0　　小数部分低位补0

  - **二进制转八进制：** 11001.11B＝011001.110B＝31.6O。

数制和编码

- **十进制整数转换为二进制整数：先确定最接近十进制数x的权2^n**
  - **X大于或等于2^n：**
    - **求x和最接近权的差；**
    - **确定小于该差值并最接近该差值的权；**
    - 重复上一步，直到差为0为止；
    - 将这些权对应位置1，其他位为0，得到二进制数。

例 2.8 将十进制数 8261 转换成二进制数。

解：最靠近 8261 的权是 8192，8261－8192＝69；69－64＝5；5－4＝1；1－1＝0。因为 8192＝2^{13}，64＝2^6，4＝2^2，1＝2^0，故第 0，2，6，13 位为 1，其余位为 0，即结果为 10 0000 0100 0101B。

数制和编码

- **十进制整数转换为二进制整数：先确定最接近十进制数x的权2^n**
  - **X小于2^n：**
    - **求2^n-1和x的差d；**
    - **确定小于该差值并最接近该差值的权；**
    - 重复上一步，直到差为0为止；
    - **将2^n-1减去d**，得到二进制数。

解：最靠近 8161 的权是 8192，d＝8192－1－8161＝30；30－16＝14；14－8＝6；6－4＝2；2－2＝0。d 对应的二进制数为 1 1110，故结果为 1 1111 1111 1111－1 1110＝1 1111 1110 0001B。

数制和编码

- **定点和浮点表示**
  - **定点数**：小数点位置约定在固定位置
  - **浮点数**：小数点位置约定为可浮动
    - 用一个**定点小数和一个定点整数**来表示

- **机器数**：计算机内部编码表示后的数 $X = X_{n-1}\ X_{n-2}\cdots X_1\ X_0$

  符号部分 **数值部分**

- **真值**：机器数真正的值（现实世界中带有正负号的数）

$$X_T = \pm X'_{n-2}\cdots X'_1 X'_0 \quad （当\ X\ 为定点整数时）$$

$$X_T = \pm 0.X'_{n-2}\cdots X'_1 X'_0 \quad （当\ X\ 为定点小数时）$$

数制和编码

- **原码表示法**

$$(1)\ 当\ X_T\ 为正数时,X_{n-1}=0,X_i=X_i'(0\leqslant i\leqslant n-2);$$
$$(2)\ 当\ X_T\ 为负数时,X_{n-1}=1,X_i=X_i'(0\leqslant i\leqslant n-2)。$$

数值10(1010B)，X_T=+000 1010, X=0000 1010B　数值-0.625 (-0.101B), X=1101 0000B

- **原码0有两种表示**

$$[+0]_{原}=0\,00\cdots0$$
$$[-0]_{原}=1\,00\cdots0$$

- **优点**：直观方便；
- **缺点**：
  - 0 的表示不唯一，不利于程序员编程；
  - 加、减运算方式不统一，特别当 a<b时，实现 a-b比较困难。
  - 需额外对符号位进行处理，不利于硬件设计。

从50年代开始，整数都采用补码来表示；但浮点数的尾数用原码定点小数表示。

数制和编码

- **补码表示法：模运算**

在一个模运算系统中，一个数与它除以“模”后的余数等价。

例1：时钟是一种模12系统 （13 mod 12 等于1，即13点钟等于1点钟）

→ 假定钟表时针指向10点，要将它拨向6点，

有两种拨法：

① 倒拨4格：10- 4 = 6

② 顺拨8格：10+8 = 18 ≡ 6 (mod 12)

模12系统中：10- 4 ≡ 10+8 (mod 12)

- 4 ≡ 8 (mod 12)

- 4的模12补码等于8；
同样有 -3 ≡ 9 (mod 12);
-5 ≡ 7 (mod 12) 等

结论1：**一个负数的补码等于模减该负数的绝对值。**

结论2：对于某一确定的模，**数x减去小于模的数y，总可以用数x加上-y的补码来代替。**

→ 补码（模运算）：实现 + 和 – 的统一

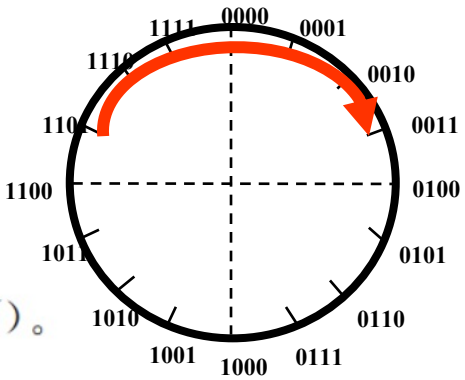
数制和编码

- **补码的定义**
  - **正数**：符号为0，数值部分是它本身
  - **负数**：模与该负数绝对值之差

（1）当 X_T 为正数时，$[X_T]_{补}=X_T=M+X_T(\mathrm{mod}\ M)$；

（2）当 X_T 为负数时，$[X_T]_{补}=M-|X_T|=M+X_T(\mathrm{mod}\ M)$。

$$[X_T]_{补}=M+X_T(\mathrm{mod}\ M)$$



当n=4，模是2^4

  - **假设补码有n位：**
    - **定点整数：** $[X_T]_{补}=2^n+X_T(-2^{n-1}\leqslant X_T<2^{n-1},\mathrm{mod}\ 2^n)$
    - **定点小数：** $[X_T]_{补}=2+X_T\quad(-1\leq X_T<1,\mathrm{mod}\ 2)$

注：实际上在计算机中并不使用补码定点小数！无需掌握该知识点

数制和编码

- **补码表示法：计算机中的运算器是模运算系统**

例：8位二进制加法器模运算系统

计算0111 1111 - 0100 0000 = ?

0111 1111 - 0100 0000 = 0111 1111 + (2^8 - 0100 0000)

=0111 1111 + 1100 0000 = 1 0011 1111 (mod 2^8)

= 0011 1111

只留余数，1被丢弃

舍弃高位而保留低n位：

- 低n位不能表示正确结果，发生“**溢出**”；
- 低n位可以表示正确结果

结论：一个负数的补码等于对应正数补码的“各位取反、末位加1”

数制和编码

- **特殊数据的补码表示**

假定机器数有n位

① $[-2^{n-1}]_{补} = 2^n - 2^{n-1} = 10\ldots0$（n-1个0） （mod 2^n）

② $[-1]_{补} = 2^n - 0\ldots01 = 11\ldots1$（n个1） （mod 2^n）

③ $[-1.0]_{补} = 2 - 1.0 = 1.00\ldots0$（n-1个0） （mod 2）

④ $[+0]_{补} = [-0]_{补} = 2^n \pm 0 = 1\ 00..0\ (\text{mod } 2n) = 00\ldots0$（n个0）

注：计算机中并不会出现-1.0的补码（**定点小数用原码表示**），这里只是想说明**同一个真值在机器中可能有不同的机器数**！

数制和编码

- **计算真值的补码**

例: 设**机器数有8位**，求**123和-123的补码**表示。

如何快速得到123的二进制表示？

解: 123 = 127- 4 = 01111111B - 100B = 01111011B

-123= - 01111011B

$[\mathbf{01111011}]_{补}= 2^8 + 01111011 = 100000000 + 01111011$

$= 01111011 \pmod{2^8}$，即 7BH。

$[\mathbf{-01111011}]_{补}= 2^8 - 01111011 = 10000\ 0000 - 01111011$

$= 1111\ 1111 - 0111\ 1011 + 1$

$= 1000\ 0100 + 1$ ← **数值部分各位取反，末位加1**

$= 1000\ 0101$，即 85H。

南京大学 NANJING UNIVERSITY

数制和编码

- **计算补码的真值**

真值→补码：数值部分各位取反，末位加1

补码→真值：数值部分末位减1，各位取反

例如：假设补码位数为8，则X=-110 0011的补码为1001 1101.

- 求真值(1)：数值部分末位减1，则得到1001 1100；
- 求真值(2)：数值部分各位取反，110 0011 （等价于111 1111-001 1100）

  → 计算可转化为（111 1111 -001 1101）+1，也即“各位取反，末位加1”

符号为0，则为正数，**数值部分相同**

符号为1，则为负数，**数值各位取反，末位加1**

数制和编码

- **变形补码**

为了判断结果是否溢出，采用双符号位的补码表示，也称模4-补码；左边是真正的符号位，右边用于判断溢出。

十进制	补码	变形补码	十进制	二进制取反	补码	变形补码
0	0000	00000	-0	1111	0000	00000
1	0001	00001	-1	1110	1111	11111
2	0010	00010	-2	1101	1110	11110
3	0011	00011	-3	1100	1101	11101
4	0100	00100	-4	1011	1100	11100
5	0101	00101	-5	1010	1011	11011
6	0110	00110	-6	1001	1010	11010
7	0111	00111	-7	1000	1001	11001
8	1000	01000	-8	0111	1000	11000

+0和-0表示唯一

值太大，用4位补码无法表示，故“溢出”！但用变形补码可保留符号位和最高数值位。

- **变形补码**

假定变形补码的位数为 $n+1$(其中符号占 2 位,数值部分占 $n-1$ 位),则变形补码可如下表示:

$$[X_T]_{变补} = 2^{n+1} + X_T \quad (-2^{n-1} \leqslant X_T < 2^{n-1}, \bmod 2^{n+1})$$

例 2.20 已知 $X_T = -1011$,分别求出变形补码取 6 位和 8 位时 $[X_T]_{变补}$。

解: $[X_T]_{变补} = 2^6 - 1011 = 100\ 0000 - 00\ 1011 = 11\ 0101$

$[X_T]_{变补} = 2^8 - 1011 = 100\ 000000 - 00\ 001011 = 11\ 110101$

数制和编码

- **反码表示法**

  与补码相比的差异在于，负数的各位取反但末位不加1. (在计算机中**很少被使用**)

- **移码表示法**

  - 将每一个数值加上一个偏置常数
  - 一般来说，当编码位数为n时，偏置取 2^{n-1}或$2^{n-1}-1$

    n=4: $E_{移}= E+ 2^3$ (2^3 =1000B)

    -8 (+8) ~ 0000B

    -7 (+8) ~ 0001B

    …

    0 (+8) ~ 1000B

    …

    +7 (+8) ~ 1111B
  - 为什么要用移码来表示指数（阶码）？

    便于浮点数加减运算时的对阶操作（比较大小）

0的移码表示唯一

移码和补码仅第一位不同

移码主要用来表示浮点数阶码！

数据的机器级表示

- 数制和编码
- **整数的表示**
- 实数的表示
- 非数值数据的编码表示
- 数据的宽度和存储

整数的表示

- **无符号整数的表示**
  - 一个编码的**所有二进位均表示数值而没有符号位**时，该编码表示无符号整数，也称无符号数。
  - 一般在**全部是正数运算且不出现负值**结果的场合下，使用无符号数表示。例如，地址运算，指针表示等。
  - **位排列顺序有2种**
    - ✓ 高到低位从左到右：0000 0000 0000 0000 0000 0000 0000 1011 ← LSB
    - ✓ 高到低位从右到左：1101 0000 0000 0000 0000 0000 0000 0000 ← MSB
    - ✓ Leftmost和rightmost这两个词有歧义，故用LSB(Least Significant Bit)来表示最低有效位，用MSB来表示最高有效位
    - ✓ 高位到低位多采用从左往右排列
  - 能表示的**最大值大于位数相同的带符号整数的最大值**：例如，8位无符号整数最大是255（1111 1111），而8位带符号整数最大为127（0111 1111）

整数的表示

- **带符号整数的表示**
  - 又称有符号整数，**必须用一个二进位来表示符号**
  - 现代计算机中带符号整数**都用补码表示**
  - **为什么用补码表示带符号整数**？
    - ✓ 数0的表示唯一，方便使用；
    - ✓ 与原码和移码相比，补码运算系统是模运算系统，加、减运算统一；
    - ✓ 比原码和反码多表示一个最小负数
    - ✓ 与反码相比，不需要通过循环进位来调整结果。

整数的表示

- **带符号整数与无符号数的比较**
  - **扩充操作有差别**
    - 例如，MIPS提供了两种加载指令（load byte unsigned / load byte）
      - 无符号数：lbu $t0, 0($s0) ；$t0高24位补0 （称为0扩展）
      - 带符号整数： lb $t0, 0($s0) ；$t0高24位补符 （称为符号扩展）
  - **数的比较有差异**
    - 无符号数：MSB为1的数比MSB为0的数大
    - 带符号整数： MSB为1的数比MSB为0的数小
    - 例如，MIPS中提供了不同的比较指令，如：
      - 无符号数：sltu $t0, $s0, $s1 （set less than unsigned）
      - 带符号整数： slt $t1, $s0, $s1 （set less than）
  - **溢出判断有差异**（无符号数根据最高位是否有进位判断溢出，通常不判）
    - MIPS规定：无符号数运算溢出时，不产生“溢出异常”

假定：$s0=1111 1111 1111 1111 1111 1111 1111 1111
$s1=0000 0000 0000 0000 0000 0000 0000 0001
则：$t0和$t1分别为多少？ 答案：$t0和$t1分别为0和1。

整数的表示

- **带符号整数与无符号数的比较**

例1（扩展操作）：在32位机器上输出si, usi, i, ui的十进制（真值）和十六进制值（机器数）是什么？

```
short  si = -32768;
unsigned short  usi = si;
int  i = si;
unsigned int ui = usi ;
```

提示：
$32768=2^{15}$
=1000 0000 0000 0000B

	真值	机器数
si	= -32768	80 00
usi	= 32768	80 00
i	= -32768	FF FF 80 00
ui	= 32768	00 00 80 00

带符号整数：符号扩展
无符号数：0扩展

整数的表示

- **C语言中的整数类型**
  - **无符号数**：unsigned int (short / long)；**带符号整数**： int (short / long)
  - 常在一个数的后面加一个“u”或“U”表示无符号数
  - 若**同时有无符号和带符号整数**，则C编译器**将带符号整数强制转换为无符号数**
  - 假定以下关系表达式在**32位用补码表示的机器上**执行，结果是什么？

关系表达式	类型	结果	说明
0 = = 0U	无	1	00…0B = 00…0B
-1 < 0	带	1	11…1B (-1) < 00…0B (0)
-1 < 0U	无	0*	11…1B (2^{32}-1) > 00…0B(0)
2147483647 > -2147483647 - 1	带	1	011…1B (2^{31}-1) > 100…0B (-2^{31})
2147483647U > -2147483647 - 1	无	0*	011…1B (2^{31}-1) < 100…0B(2^{31})
2147483647 > (int) 2147483648U	带	1*	011…1B (2^{31}-1) > 100…0B (-2^{31})
-1 > -2	带	1	11…1B (-1) > 11…10B (-2)
(unsigned) -1 > -2	无	1	11…1B (2^{32}-1) > 11…10B (2^{32}-2)

数据的机器级表示

- 数制和编码
- 整数的表示
- **实数的表示**
- 非数值数据的编码表示
- 数据的宽度和存储

实数的表示

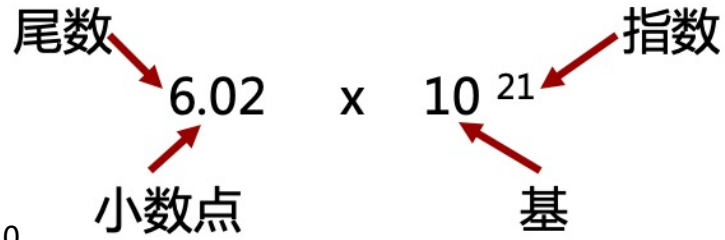
- **科学计数法与浮点数**

十进制数:

° **规格化形式**: 小数点前只有一位非0数

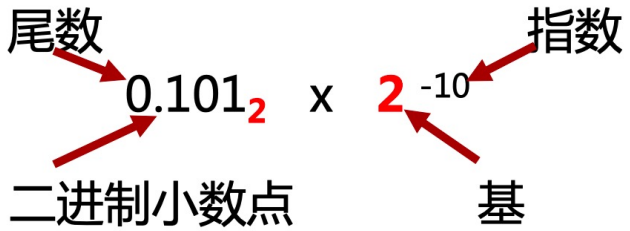
° 对于数 1/1,000,000,000

- **唯一的规格化形式: 1.0 x 10^{-9}**
- 非规格化形式不唯一: 0.1 x 10^{-8}, 10.0 x 10^{-10}



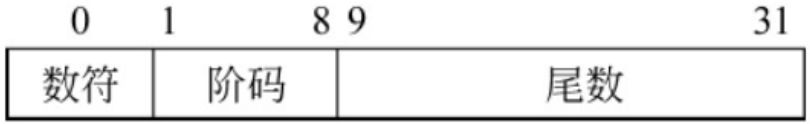
二进制数

- 只要**对尾数和指数分别编码**，就可表示一个**浮点数**（即：实数）



实数的表示

- **浮点数的表示**
  - 例：画出下述32位浮点数格式的**规格化**数的表示范围。

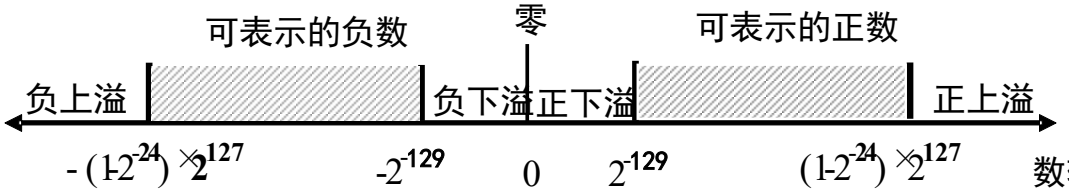


$+/-0.1xxxxx \times 2^{E}$

第0位数符S；第1～8位为8位移码表示阶码E（偏置常数为128）；**第9～31位为24位二进制原码小数表示的尾数M。规格化尾数的小数点后第一位总是1**，故规定第一位默认的“1”不明显表示出来。这样可用23个数位表示24位尾数。

最大正数：$0.11\ldots1 \times 2^{11\ldots1} = (1-2^{-24}) \times 2^{127}$　最小正数：$0.10\ldots0 \times 2^{00\ldots0} = (1/2) \times 2^{-128}$

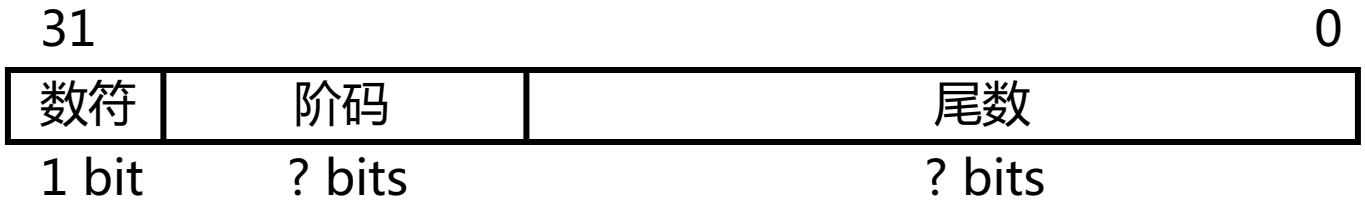
- 因为原码是对称的，所以其表示范围关于原点对称。



- ✓ **机器0：尾数为0或落在下溢区中的数**
- ✓ **浮点数范围比定点数大，但数的个数没变多，故数之间更稀疏，且不均匀**

实数的表示

- **浮点数表示**
  - 32-bit 规格化浮点数：



(基可以是 2/ 4 / 8 / 16，约定信息，无需显式表示）

  - 早期的计算机，**各自定义自己的浮点数格式**

问题：浮点数表示不统一会带来什么问题？

实数的表示

- **IEEE 754 浮点数标准**
  - 直到80年代初，各个机器内部的浮点数表示格式还没有统一；因而相互不兼容，机器之间传送数据时，带来麻烦
  - 1970年代后期, IEEE成立委员会着手制定浮点数标准
  - 1985年完成浮点数标准IEEE 754的制定
  - **现在所有计算机都采用IEEE 754来表示浮点数**

This standard was primarily the work of one person, UC Berkeley math professor William Kahan.

1989
ACM Turing
Award Winner!

www.cs.berkeley.edu/~wkahan/ieee754status/754story.html

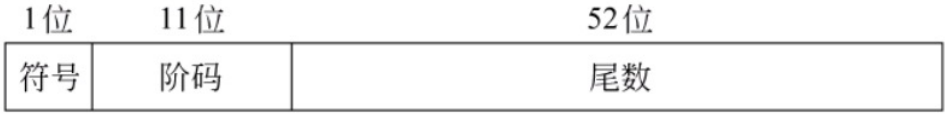
Prof. William Kahan ("Father" of the IEEE 754 standard)

实数的表示

- **IEEE 754 浮点数标准**

(a) 32位单精度格式

- 符号s: 1位
- 阶码e: 8位，移码，偏置$2^{n-1}-1=127$
- 尾数f：23位，原码，**第一位总为1**，缺省/**隐藏**



(b) 64位双精度格式

- 符号s: 1位
- 阶码e: 11位，移码，偏置1023
- 尾数f：52位，原码，**隐藏第1位的1**

图 2.2　IEEE 754 浮点数格式

规格化数： $+/-1.xxxxxxxxxx_2 \times 2^E$

单精度： $(-1)^s \times (1 + f) \times 2^{(e-127)}$　阶码e范围为0000 0001 (-126) ~ 1111 1110 (127)

（全0和全1用来表示特殊值）

双精度： $(-1)^s \times (1 + f) \times 2^{(e-1023)}$

实数的表示

- **(IEEE 754)二进制浮点数转换成十进制数**

十六进制数BEE00000H，其对应的二进制浮点数，

1	0111 1101	110 0000 0000 0000 0000 0000

$$(-1)^S \times (1 + f) \times 2^{(e-127)}$$

- 数符s: 1 → 负数
- 阶码e: $0111\ 1101_{two} = 125_{ten}$，125 - 127 = -2
- 尾数f: $1 + 1x2^{-1} + 1x2^{-2} + 0x2^{-3} + 0x2^{-4} + 0x2^{-5} + \ldots$
  $=1+2^{-1}+2^{-2} = 1+0.5+0.25 = 1.75$
- 十进制数: $-1.75_{ten}x2^{-2} = -0.4375$

实数的表示

- **十进制数转换成(IEEE 754)二进制浮点数**

1. 十进制数: -12.75
2. 转换整数部分: 12 = 8 + 4 = 1100_2
3. 转换小数部分: .75 = .5 + .25 = $.11_2$
4. **合并整数部分和小数部分，并进行规格化**: 1100.11 = 1.10011 x 2^3
5. **计算阶码:** 127 + 3 = 128 + 2 = $1000\ 0010_2$

1	1000 0010	100 1100 0000 0000 0000 0000

十六进制表示为C14C0000H

实数的表示

- **特殊位序列的解释**
  - **全0阶码全0尾数：+0/-0**

    阶码: 全0

    尾数: 全0

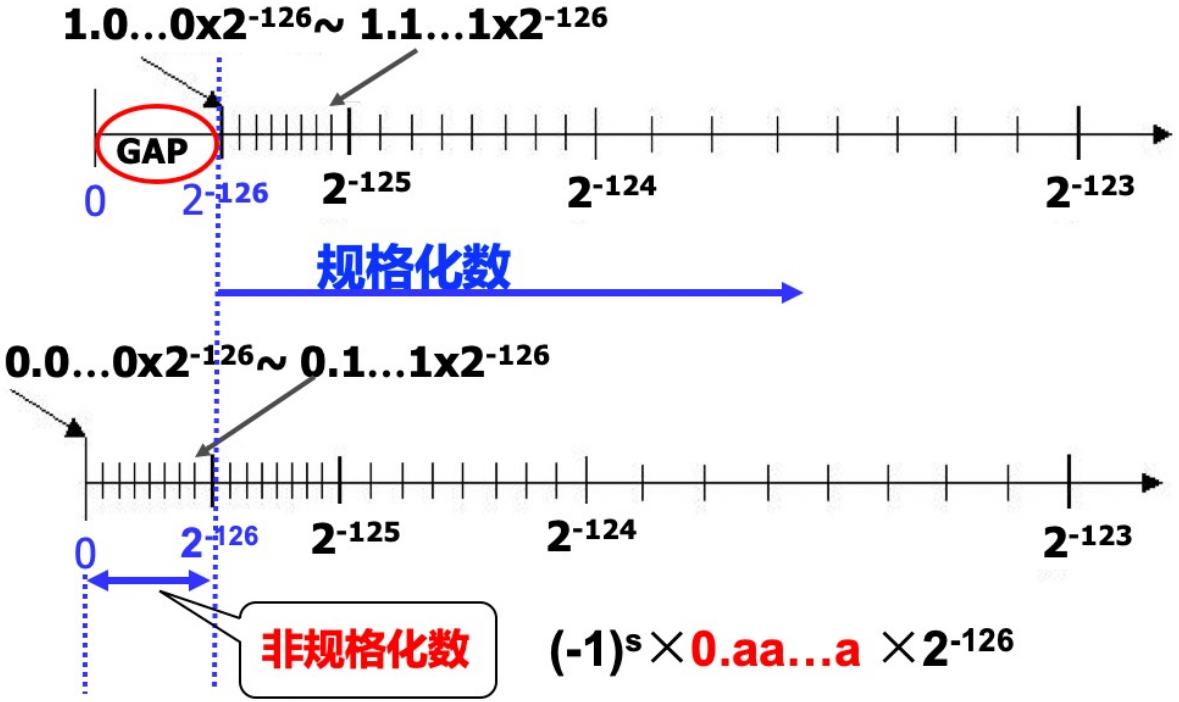
    符号：+ 或者 0.

    +0: 0 00000000 00000000000000000000000

    -0: 1 00000000 00000000000000000000000

实数的表示

- **全0阶码非0尾数：非规格化（尾数高位有一个或多个连续0，隐藏位为0）**



用于**处理阶码下溢**，当出现比最小规格化数还小的数时程序能继续执行。

实数的表示

- **全1阶码全0尾数： +∞/-∞**

在浮点数中, **除数为0的结果是 +/- ∞**, 不是溢出异常.（整数除0则为异常）

使计算机在出现异常的情况下能继续执行，并为程序提供错误检测功能。

如何表示 +∞/-∞?

- +∞ : 0 11111111 00000000000000000000000
- -∞ : 1 11111111 00000000000000000000000

操作数为无穷大，产生明确结果

5.0 / 0 = +∞, -5.0 / 0 = -∞

5+(+∞) = +∞, (+∞)+(+∞) = +∞

5 - (+∞) = -∞, (-∞) - (+∞) = -∞ etc

操作数为无穷大，产生不发信号的非数NaN

(+∞)+(-∞) , (+∞)-(+∞) , ∞/∞ etc

实数的表示

- **全1阶码非0尾数： NaN (not a number)**

NaN表示一个没有定义的数，称为非数。

- 尾数最高有效位是1：不发NaN（不发“异常”通知）
- 尾数最高有效位是0：发NaN（发“异常”通知）

表 2.3　产生不发信号 NaN 的操作

运 算 类 型	产生不发信号 NaN 的计算操作
所有	对通知 NaN 的任何计算操作
加减	无穷大相减：(+∞)+(−∞)、(+∞)−(+∞)等
乘	0×∞
除	0/0 或∞/∞
求余	X MOD 0 或 ∞ MOD y
平方根	$\sqrt{x}$ 且 $x<0$

实数的表示

- **阶码非全0且非全1：规格化非0数**
  - **阶码范围**：单精度（1~254），双精度（1~2046）
  - **指数范围**：单精度（-126~+127），双精度（-1022~1023）

$$(-1)^s \times 1.f \times 2^{e-127} \quad \text{和} \quad (-1)^s \times 1.f \times 2^{e-1023}$$

  - **最大取值**：单精度（+1.11...1X 2^{127}，约 +3.4 X 10^{38}），
    双精度（约 +1.8 X 10^{308}）

实数的表示

- **C语言中的浮点数类型**
  - **int (32位)、float (32位)、double (64位)类型转换**：
    - ✓ int → float：不会溢出，但可能数据被舍入
    - ✓ int/float → double：能保留精确值
    - ✓ double → float：可能溢出，可能舍入
    - ✓ float/double → int：数据可能向0方向被截断。
  - **例子：**假定i, f, d分别为int, float, double，下述是否永真。

A. i==(int) (float) i **不是**
B. f==(float) (int) f **不是**
C. i==(int) (double) i **是**
D. f==(float) (double) f **是**
E. d==(float) d **不是**
F. f==-(-f) **是**
G. (d+f)-d==f **不是**

数据的机器级表示

- 数制和编码
- 整数的表示
- 实数的表示
- **非数值数据的编码表示**
- 数据的宽度和存储

非数值数据的编码表示

- **逻辑值**
  - **表示**
    - 用一位表示。例如，真：1 / 假：0
    - N位二进制数可表示N个逻辑数据，或一个位串
  - **运算**
    - 按位进行
    - 如: 按位与 / 按位或 / 逻辑左移 / 逻辑右移 等
  - **识别**
    - 逻辑数据和数值数据在形式上并无差别，也是一串0/1序列，机器靠指令来识别。
  - **位串**
    - 用来表示若干个状态位或控制位（OS中使用较多）

非数值数据的编码表示

- **西文字符**
  - **特点**
    - 是一种拼音文字，用有限几个字母可拼写出所有单词
    - 只对有限个字母和数学符号、标点符号等辅助字符编码
    - 所有字符总数不超过256个，使用7或8个二进位可表示
  - **表示**（常用**编码为7位ASCII码**）
    - 十进制数字：0/1/2…/9
    - 英文字母：A/B/…/Z/a/b/…/z
    - 专用符号：+/-/%/*/&/……
    - 控制字符（不可打印或显示）
  - **操作**
    - 字符串操作，如: 传送/比较等

非数值数据的编码表示

- **汉字字符**
  - **特点**
    - 汉字是表意文字，一个字就是一个方块图形。
    - 汉字数量巨大，总数超过6万字，给汉字在计算机内部的表示、汉字的传输与交换、汉字的输入和输出等带来了一系列问题。
  - **编码形式**
    - 有以下几种汉字代码：
      - **输入码**：对汉字用相应按键进行编码表示，用于输入
      - **内码**：用于在系统中进行存储、查找、传送等处理
      - **字模点阵或轮廓描述**: 描述汉字字模点阵或轮廓，用于显示/打印

问题：西文字符有没有输入码？有没有内码？有没有字模点阵或轮廓描述？

非数值数据的编码表示

- **汉字字符**
  - **向计算机输入汉字的方式**：
    ① 手写汉字联机识别输入，或者是印刷汉字扫描输入后自动识别，这两种方法现均已达到实用水平。
    ② 用语音输入汉字，虽然简单易操作，但离实用阶段还相差很远。
    ③ 利用英文键盘输入汉字：每个汉字用一个或几个键表示，这种对每个汉字用相应按键进行的编码称为汉字“输入码”，又称外码。输入码的码元为按键。是最简便、最广泛的汉字输入方法。
  - **常用的方法有**：搜狗拼音、五笔字型、智能ABC、微软拼音等
  - **使用汉字输入码的原因**：
    ① 键盘面向西文设计，一个或两个西文字符对应一个按键，非常方便。
    ② 汉字是大字符集，专门的汉字输入键盘由于键多、查找不便、成本高等原因而几乎无法采用。

非数值数据的编码表示

- **汉字字符**
  - 问题：西文字符常用的内码是什么？（ASCII码）
  - **对于汉字内码的选择**，必须考虑以下几个因素：
    - ① 不能有二义性，即不能和ASCII码有相同的编码。
    - ② 尽量与汉字在字库中的位置有关，便于汉字查找和处理。
    - ③ 编码应尽量短。
  - **国标码**（国标交换码）：1981年我国颁布了《信息交换用汉字编码字符集·基本集》(GB2312—80)。该标准选出6763个常用汉字，为每个汉字规定了标准代码，以供汉字信息在不同计算机系统间交换使用
  - 汉字的国标码是唯一的，但汉字内码可能随系统的不同而有差别。

非数值数据的编码表示

- **汉字字符：GB2312-80字符集**
  - **由三部分组成：**

    ① 字母、数字和各种符号，包括英文、俄文、日文平假名与片假名、罗马字母、汉语拼音等共687个

    ② 一级常用汉字，共3755个，按汉语拼音排列

    ③ 二级常用汉字，共3008个 ，不太常用，按偏旁部首排列
  - **汉字的区位码**
    - 码表由94行、94列组成，行号为区号，列号为位号，各占7位
    - 指出汉字在码表中的位置，共14位，区号在左、位号在右
  - **汉字的国标码**
    - 每个汉字的区号和位号各自加上32（20H），得到其“国标码”
    - 国标码中区号和位号各占7位。在计算机内部，为方便处理与存储，前面添一个0，构成一个字节

非数值数据的编码表示

- **汉字字符：汉字内码**
  - 至少**需2个字节**才能表示一个汉字内码。为什么？

    –由汉字的总数决定！
  - 可在**GB2312国标码**的基础上产生汉字**内码**

    –为与ASCII码区别，将国标码的两个字节的第一位置“1”后得到一种汉字内码

    例如，汉字“大”在**码表**中位于第20行、第83列。因此**区位码**为0010100 1010011，国标码为**0**0110100 **0**1**1**10011，即3473H。前面的34H和字符“4”的ACSII码相同，后面的73H和字符“s”的ACSII码相同，将每个字节的最高位各设为“1”后，就得到其内码：B4F3H (**1**011 0100 **1**111 0011B)，因而不会和ASCII码混淆。

区位码→(区号、位号各加上32) →**国标码**→(每个字节最高位设1) →**内码**

非数值数据的编码表示

- **国际字符集**
- **国际字符集的必要性**
  - 不同地区使用不同字符集内码，如**中文GB2312 / Big5、日文Shift-JIS / EUC-JP**等。在安装中文系统的计算机中打开日文文件，会出现**乱码**。
  - 为使所有国际字符都能互换，必须创建一种涵盖全部字符的多字符集。
- **国际多字符集**
  - 通过对各种地区性字符集规定使用范围来唯一定义各字符的编码。
  - **国际标准ISO/IEC 10646**提出了一种包括全世界现代书面语言文字所使用的所有字符的标准编码，有4个字节编码(**UCS-4**)和2字节编码(**UCS-2**) 。
  - 我国（包括香港、台湾地区）与日本、韩国联合制订了一个统一的汉字字符集（CJK编码），共收集了上述不同国家和地区共约2万多汉字及符号，采用2字节编码 (即：UCS-2) ，已被批准为国家标准 (**GB13000**)。
  - Windows操作系统(中文版)已采用中西文统一编码，收集了中、日、韩三国常用的约2万汉字，称为“**Unicode**” ，采用2字节编码，与UCS-2一致 。

非数值数据的编码表示

- **汉字的字模点阵码和轮廓描述**
  - 为便于打印、显示汉字，**汉字字形必须预先存在机内**
    - 字库 (font)：所有汉字形状的描述信息集合
    - 不同字体 (如宋体、仿宋、楷体、黑体等) 对应不同字库
    - 从字库中找到字形描述信息，然后送设备输出

      问题：如何知道到哪里找相应的字形信息？ **区位码←国标码←内码**
  - **字形主要有两种描述方法**：
    - 字模点阵描述（图像方式）
    - 轮廓描述（图形方式）
      - 直线向量轮廓
      - 曲线轮廓

数据的机器级表示

- 数制和编码
- 整数的表示
- 实数的表示
- 非数值数据的编码表示
- **数据的宽度和存储**

数据的宽度和存储

- **数据的宽度和单位**
  - **比特（bit）**是计算机中处理、存储、传输信息的最小单位
  - 二进制信息的计量单位是“**字节”(Byte)**，也称“位组”，1字节为8比特
    - 现代计算机中，存储器**按字节编址**
    - 字节是最小可寻址单位
    - 如果以**字节**为一个排列**单位**，**则LSB表示最低有效字节，MSB表示最高有效字节**
  - 除比特和字节外，还经常使用**“字”(word)**作为单位：字可能由2个、4个、8个甚至16个字节组成。

数据的宽度和存储

- **数据的宽度和单位**

- “字”和“字长”的概念不同
  - **“字长”指定点运算数据通路的宽度**：

    数据通路指CPU内部数据流经的路径以及路径上的部件，主要是CPU内部进行数据运算、存储和传送的部件，这些部件的宽度基本上要一致，才能相互匹配。因此，”字长”等于**CPU内部总线的宽度、运算器的位数、通用寄存器的宽度**等。
  - **“字”表示被处理信息的单位，用来度量数据类型的宽度**
  - **字和字长的宽度可以一样，也可不同。**

    例如，x86体系结构定义“字”的宽度为16位，但从386开始字长就是32位了。

- IA-32中的“字”有多少位？(16位)　　字长多少位呢？(32位)

数据的宽度和存储

- **主存、主频、带宽**
  - 存储二进制信息时的度量单位要比字节或字大得多
  - **容量**经常使用的单位有（**2 的幂次方**）：
    - “千**字节**”（**KB**），1KB=2^{10}字节=1024B
    - “兆字节”（MB），1MB=2^{20}字节=1024KB
    - “千兆字节”（GB），1GB=2^{30}字节=1024MB
    - “兆兆字节”（TB），1TB=2^{40}字节=1024GB
  - 通信中的**带宽**使用的单位有（**10的幂次方**）：
    - “千**比特**/秒”（**kb/s**），1kbps=10^3 b/s=1000 bps
    - “兆比特/秒”（Mb/s），1Mbps=10^6 b/s =1000 kbps
    - “千兆比特/秒”（Gb/s），1Gbps=10^9 b/s =1000 Mbps
    - “兆兆比特/秒”（Tb/s），1Tbps=10^{12} b/s =1000 Gbps

如果把b换成B，则表示字节而不是比特（位）：例如，10MBps表示 10兆字节/秒

数据的宽度和存储

- **C语言中数值数据类型的宽度**
- **高级语言支持多种类型、多种长度的数据**
  - 例如，C语言中char类型的宽度为1个字节，可表示一个字符（非数值数据），也可表示一个8位的整数（数值数据）
  - 不同机器上表示的同一种类型的数据可能宽度不同
- 程序中的数据有相应的机器级表示方式和相应的处理指令

  (在第四章指令系统介绍具体指令)

从表中看出：**同类型数据并不是所有机器都采用相同的宽度**，分配的字节数随机器字长和编译器的不同而不同。

C语言中数值数据类型的宽度 (单位：字节)

C声明	典型32位机器	Compaq Alpha机器
char	1	1
short int	2	2
int	4	4
long int	4	8
char*	4	8
float	4	4
double	8	8

Compaq Alpha是一个针对高端应用的64位机器，即字长为64位

数据的宽度和存储

- **数据的存储和排列顺序**

- 80年代开始，几乎所有通用机器都用**字节编址**
- ISA设计时要考虑的两个问题：
  - 如何**根据一个地址取到一个32位的字**？- 字的存放问题
  - 一个字能否存放在任何地址边界？- 字的边界对齐问题

$65535=2^{16}-1$

$[-65535]_{补}=FFFF0001H$

- 例如，若 int i = -65535，存放在内存100号单元（即占100#～103#），则**用“取数”指令访问100号单元取出 i 时**，必须清楚 i 的4个字节是如何存放的。

小端方式

FF	FF	00	01
103	102	101	100
MSB			LSB

大端方式

100	101	102	103

- **大端方式（Big Endian）**：MSB所在的地址是数的地址，e.g. IBM 360/370, Motorola 68k, MIPS, Sparc, HP PA
- **小端方式（Little Endian）**：LSB所在的地址是数的地址，e.g. Intel 80x86, DEC VAX

有些机器两种方式都支持，可通过特定控制位来设定采用哪种方式。

数据的宽度和存储

- **数据的存储和排列顺序：大端方式和小端方式**

例1: 十六进制数ABCDH存放在1000号单元

大端方式: ⟶

CD	1001
AB	**1000**

小端方式: ⟶

AB	1001
CD	**1000**

例2: 十六进制数00ABCDEFH存放在1000号单元

大端方式: ⟶

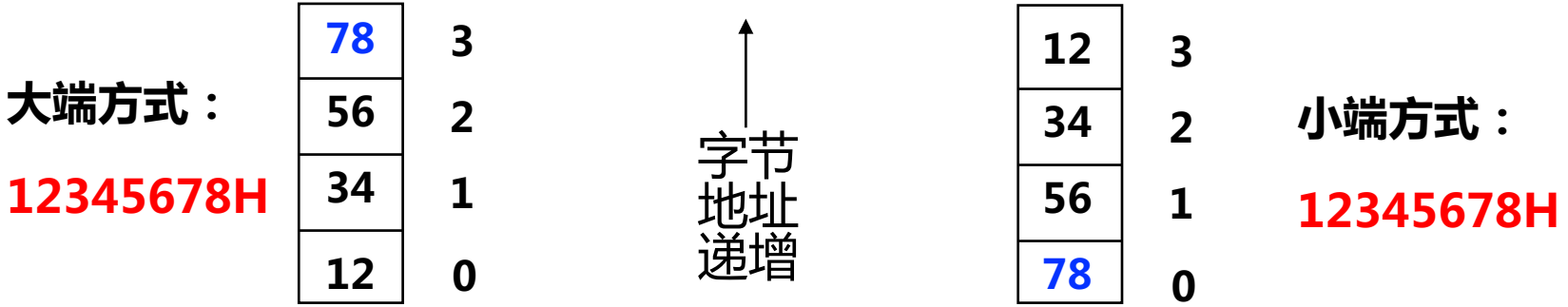
00	**1000**
AB	1001
CD	1002
EF	1003

小端方式: ⟶

00	1003
AB	1002
CD	1001
EF	**1000**

数据的宽度和存储

- **数据的存储和排列顺序：字节交换问题**



- **存放方式不同**的机器间程序移植或数据通信时，会发生什么问题？
  - ✓ 每个系统内部是一致的，但在系统间通信时可能会发生问题！
  - ✓ **因为顺序不同，需要进行顺序转换**
- 音、视频和图像等文件格式或处理程序都涉及到**字节顺序问题**
  - ✓ 小端方式: GIF, PC Paintbrush, Microsoft RTF,etc
  - ✓ 大端方式: Adobe Photoshop, JPEG, MacPaint, etc

数据的宽度和存储

- **数据的存储和排列顺序：对齐——要求数据的地址是相应的边界地址**
- 目前机器**字长一般为32位或64位**，而存储器地址按字节编址
- 指令系统支持对**字节、半字、字及双字**的运算，也有位处理指令
- 各种不同长度的数据存放时，有两种处理方式:
  - **按边界对齐**（假定**存储字**的宽度为32位，按字节编址）
    - 字地址：4的倍数
    - 半字地址：2的倍数
    - 字节地址：任意

每4个字节可同时读写

  - **不按边界对齐**

    坏处：可能会增加访存次数！（学了第七章存储器组织后会更明白！）

数据的宽度和存储

- **数据的存储和排列顺序**：对齐——要求数据的地址是相应的边界地址

  如：int i, short k, double x, char c, short j,……

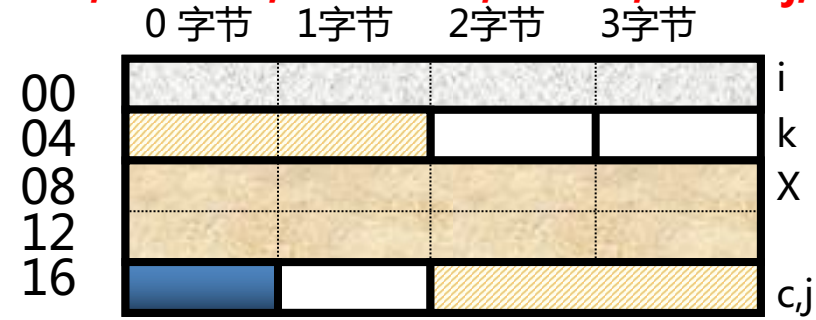
- **存储器按字节编址**：

  每次只能读写某个字地址开始的4个单元中连续的1个、2个、3个或4个字节

按边界对齐

x：2个周期

j：1个周期



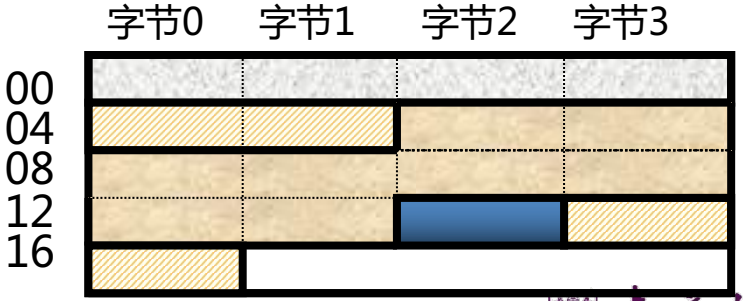
则：&i=0; &k=4; &x=8; &c=16; &j=18;……

- **不按字节编制**：虽节省了空间，但**增加了访存次数**！需要权衡，目前来看，浪费一点存储空间没有关系！

边界不对齐

x：3个周期

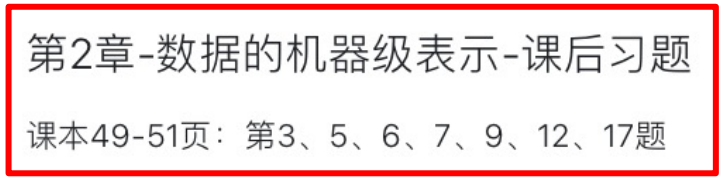
j：2个周期



则： &i=0; &k=4; &x=6; &c=14; &j=15;……

课程习题（作业）——截止日期：9月23日晚23:59

- **课本49-51页**：第3、5、6、7、9、12、17题
- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）



- 命名：学号+姓名+第*章。
- 若提交遇到问题请及时发邮件或在下一次上课时反馈。

课程习题（作业）——截止日期：9月23日晚23:59

3. 实现下列各数的转换。

(1) $(25.8125)_{10}=(?)_2=(?)_8=(?)_{16}$

(2) $(101101.011)_2=(?)_{10}=(?)_8=(?)_{16}$

(3) $(4E.C)_{16}=(?)_{10}=(?)_2$

5. 假定机器数为 8 位(1 位符号,7 位数值),写出下列各二进制整数的补码和移码(偏置常数为 128)表示。

+1001,−1001,+1,−1,+10100,−10100,+0,−0

6. 已知 $[x]_{补}$,求 x。

(1) $[x]_{补}=1.1100111$　　(2) $[x]_{补}=10000000$

(3) $[x]_{补}=0.1010010$　　(4) $[x]_{补}=11010011$

课程习题（作业）——截止日期：9月23日晚23:59

7. 假定一台 32 位字长的机器中带符号整数用补码表示，浮点数用 IEEE 754 标准表示，寄存器 R1 和 R2 的内容分别为 R1：0000 108BH，R2：8080 108BH。不同指令对寄存器内容进行不同的操作，因而，不同指令执行时寄存器内容对应的真值不同。假定执行下列运算指令时，操作数为寄存器 R1 和 R2 的内容，则 R1 和 R2 中操作数的真值分别为多少？

（1）无符号整数加法指令。

（2）带符号整数乘法指令。

（3）单精度浮点数减法指令。

9. 以下是一个 C 语言程序，用来计算一个数组 a 中每个元素的和。当参数 len 为 0 时，返回值应该是 0，但是在机器上执行时，却发生了存储器访问异常。请问这是什么原因造成的，并说明程序应该如何修改。

```
1  float sum_elements(float a[],unsigned len)
2  {
3      int i;
4      float result=0;
5
6      for (i=0; i<=len-1; i++)
7          result+=a[i];
8
9      return result;
10 }
```

课程习题（作业）——截止日期：9月23日晚23:59

12. 以 IEEE 754 单精度浮点数格式表示下列十进制数。

+1.75，+19，−1/8，258

17. 假定在一个程序中定义了变量 x、y 和 i，其中，x 和 y 是 float 型变量（用 IEEE 754 单精度浮点数表示），i 是 16 位 short 型变量（用补码表示）。程序执行到某一时刻，$x=-0.125$、$y=7.5$、$i=100$，它们都被写到了主存（按字节编址），其地址分别是 100、108 和 112。请分别画出在大端机器和小端机器上变量 x、y 和 i 在内存的存放位置。

课程实验——截止日期：10月14日晚23:59

- 提交方式：https://selearning.nju.edu.cn/（教学支持系统）

- 命名：学号+姓名+实验*。
- **提交：文件打包，提交ZIP压缩文件。**

课程实验——截止日期：10月14日晚23:59

- **实验一 无符号数和带符号整数的表示**

- **实验目的:** 通过无符号数和带符号整数之间的相互转换来理解无符号数和补码整数的表示。
- **实验要求:** 编程验证**第2章课后习题8**的关系表达式，并给出源程序和执行结果。
- **实验报告:**
  - ✓ 1. 给出源程序(文本文件)和执行结果。
  - ✓ 2. 根据实验结果，回答下列问题。
    (1)你的机器字长多少位? int 类型的位数、最小值和最大值各是多少?
    (2)在你的机器上，-1 用 int 类型和 unsigned int 类型表示的结果分别是多少?

课程实验——截止日期：10月14日晚23:59

8. 假定机器 M 的字长为 32 位，用补码表示带符号整数。下表第一列给出了在机器 M 上执行的 C 语言程序中的关系表达式，请参照已有的表栏内容完成表中后三栏内容的填写。

关系表达式	运算类型	结果	说　明
0==0U			
−1<0			
−1<0U	无符号整数	0	11…1B ($2^{32}-1$)>00…0B(0)
2147483647>−2147483647−1	带符号整数	1	011…1B ($2^{31}-1$)>100…0B (-2^{31})
2147483647U>−2147483647−1			
2147483647>(int)2147483648U			
−1>−2			
(unsigned)−1>−2			

注：表中第 4 和第 5 行的−2147483647−1 没有写成−2147483648，因为编译器处理一个形如−x 的表达式时，通常会先读取表达式 x，然后对 x 取负。当 x=2147483648 时，因为用 32 位补码无法表示 x，所以，写成−2147483648 时可能会发生意想不到的结果。

提问

Q & A

殷亚凤
智能软件与工程学院
苏州校区南雍楼东区225
yafeng@nju.edu.cn，https://yafengnju.github.io/

南京大学 NANJING UNIVERSITY