数据的机器级表示

一、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

机器级数据分两大类：

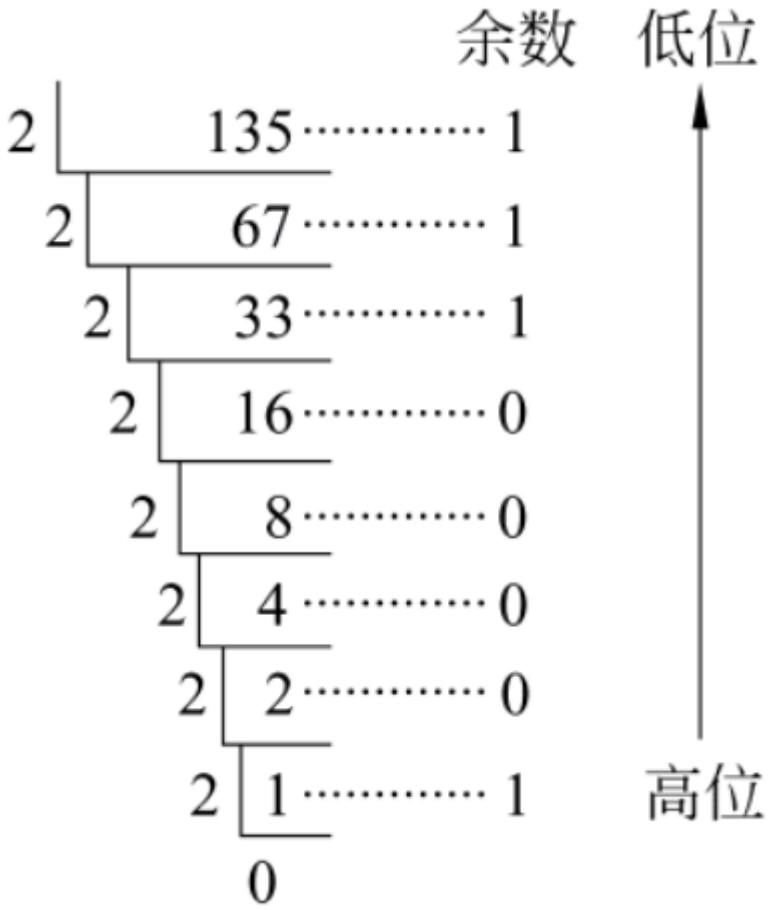
- 数值数据
  - 整数：无符号整数、带符号整数；定点数表示
  - 实数：浮点数表示
- 非数值数据：逻辑数（包括位串）、西文字符和汉字

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

进位计数制

R进制：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。

- R进制数转换成十进制数：按权展开
- 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

$0.6875\times2=1.375$ 整数部分$=1$ （高位）

$0.375\times2=0.75$ 整数部分$=0$

$0.75\times2=1.5$ 整数部分$=1$

$0.5\times2=1.0$ 整数部分$=1$ （低位）

- 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：
  - X小于2^n：求2^{n-1}和x的差d

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）

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

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

1. 原码表示法

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

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

加、减运算方式不统一，特别当a<b时，实现a-b比较困难

1. 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

- 一个负数的补码等于模减该负数的绝对值
- 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

一个负数的补码等于对应正数补码的“**各位取反、末位加1**”

- 计算补码的真值
  - 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
  - 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
- 变形补码

1. 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)

1. 移码表示法

二、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

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

带符号整数的表示

现代计算机中带符号整数都用补码表示

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)

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

三、实数的表示

科学计数法与浮点数

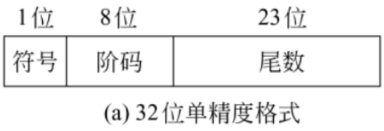
1. 十进制数:

- 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: $1.0x10^{-9}$
- 非规格化形式不唯一: $0.1x10^{-8}, 10.0x10^{-10}$

1. 二进制数

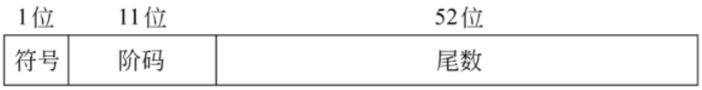
- 只要对尾数和指数分别编码，就可表示一个浮点数（即：实数）

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$_2$ x 2^E

单精度：$(-1)^s$ x (1 + f) x $2^{(e-127)}$ 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：$(-1)^s$ x (1 + f) x $2^{(e-1023)}$

- (IEEE 754)二进制浮点数转换成十进制数
- 十进制数转换成(IEEE 754)二进制浮点数
- 特殊尾序列的解释

1. 全0阶码全0尾数：+0/-0

1. 阶码: 全0

2. 尾数: 全0
3. 符号：+ 或者0.

2. 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。
3. 全1阶码全0尾数： +∞/-∞
   1. 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）
   2. 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。
   3. +∞ : 0 11111111 00000000000000000000000
      -∞ : 1 11111111 00000000000000000000000
   4. 操作数为无穷大，产生不发信号的非数NaN：(+∞)+(-∞)，(+∞)-(+∞)， ∞/∞ etc
4. 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 $x<0$

5. 阶码非全0且非全1：规格化非0数

C语言中的浮点数类型

1. 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。

四、非数值数据的编码表示

逻辑值

1. 表示
   1. 用一位表示。例如，真：1 / 假：0
   2. N位二进制数可表示N个逻辑数据，或一个位串
2. 运算
   1. 按位进行
   2. 如: 按位与/ 按位或/ 逻辑左移/ 逻辑右移
3. 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。
4. 位串

- 用来表示若干个状态位或控制位（OS中使用较多）

西文字符

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

汉字字符

1. 特点
    1. 汉字是表意文字，一个字就是一个方块图形。
    2. 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。
2. 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

五、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
- “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。
- 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

- 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

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

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。

数据的存储和排列顺序

1. 大端方式和小端方式

大端方式（Big Endian）：MSB所在的地址是数的地址

小端方式（ Little Endian）：LSB所在的地址是数的地址

1. 字节交换问题
2. 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：
    1. 按边界对齐（假定存储字的宽度为32位，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
    2. 不按边界对齐：虽节省了空间，但增加了访存次数