第 1 章　绪　　论

1.1　计算机组成与工作原理

计算机是一种能自动、高速地进行大量信息处理的工具，一个完整的计算机系统由硬件和软件两部分组成。本节介绍的计算机组成主要指计算机硬件系统的基本组成。所谓硬件是指计算机系统中由电子线路和各种机电物理装置组成的实体，它是计算机实现其功能的物质基础。

1.1.1　指令系统

机器指令是要计算机执行某种操作的命令，且由计算机直接识别执行。一台计算机可以有许多指令，作用也各不相同，所有指令的集合称为计算机的指令系统。

指令系统是计算机基本功能具体而集中的体现。从计算机系统结构的角度看，指令系统是软件和硬件的界面，指令是对计算机进行程序控制的最小单位。

指令系统的内核是硬件。当一台机器的指令系统确定之后，硬件设计师根据指令系统的约束条件，构造硬件组织，由硬件支持指令系统功能得以实现；而软件设计师在指令系统的基础上建立程序系统，扩充和发挥机器的功能。

1. 指令的功能

指令按其功能可分为两种类型：一类是命令计算机的各个部件完成基本的算术逻辑运算、数据存取和数据传送等操作，属操作类指令；另一类则是用来控制程序本身的执行顺序，实现程序的分支、转移等，属控制转移类指令。

2. 指令的格式

指令格式是“指令字”用二进制代码表示的结构格式。一条指令通常由两部分组成：操作码和地址码。操作码指明计算机应该执行的某种操作的性质与功能；地址码则指出进行操作的数据（简称操作数）存放在何处，即指明操作数地址。一种例外情况是地址码位置存放的就是操作数本身。因此，一条指令的基本格式如下：

操作码 OP	地址码 AD

机器语言的语法规则就是机器指令的格式。

一条指令字中所包含的二进制码的总位数称为指令字长。

对不同种类的机器而言，指令系统的指令数目和指令字的具体组成呈现出很大的差异。

1.1.2 计算机组成——硬件系统

计算机的硬件系统由五个基本部分组成，即运算器、控制器、存储器、输入设备和输出设备，其中存储器又有内存储器和外存储器之分。

在计算机中，各部件之间来往的信息可分成三种类型：地址、数据(包括指令)和控制信号。图 1.1 是一般计算机的结构框图，它只画出了数据和部分地址信息。

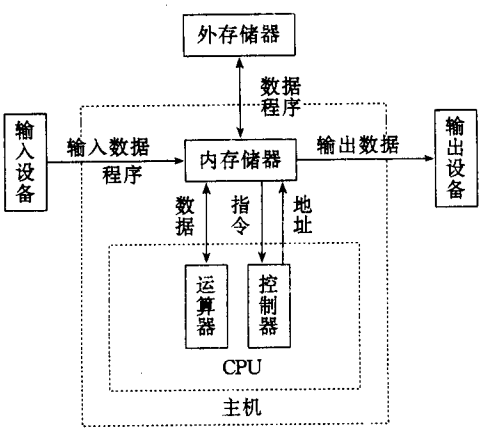


图 1.1 计算机结构框图

当前大部分计算机(特别是微机)各部件之间是用总线相连接，系统总线成为计算机内部传输各种信息的通道。图 1.2 是以总线连接的计算机框图。

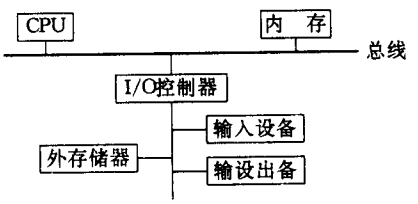


图 1.2 总线结构的计算机框图

运算器和控制器是计算机的核心，一般称为中央处理单元(Central Processing Unit)，简称 CPU。主机一般包括 CPU、内存，有时还包括外设控制器，它们通常放在主机柜中。当然这种划分主要是对大型机而言。

下面分别介绍各组成部分的基本功能、结构及工作原理。

1. 存储器(memory)

存储器的主要功能是存放程序和数据，程序是计算机操作的依据，数据是计算机操作的对象。为了实现自动计算，各种信息必须预先存放在计算机内的某个地方，这个地方就是存储器。

存储器有内存(主存)和外存(辅存)之分，外存是存放程序和数据的“仓库”，可以长

时间地保存大量信息，但程序必须调入内存方可执行，待处理的数据也只有进入内存后才能被程序加工。

存储器采取按地址存(写)取(读)的工作方式。一个内存体内包含许多存储单元，每个单元可以存放一个适当单位的信息。全部存储单元按一定顺序编号，这种编号就称为存储器的地址。

图 1.3 是主存储器结构框图。当要对存储器进行读写操作时，来自地址总线的存储器地址经地址译码器译码后，选中指定的存储单元，而读写控制电路根据读写命令实施对于存储器的存取操作，数据总线则用于传送写入内存或从内存取出的信息。

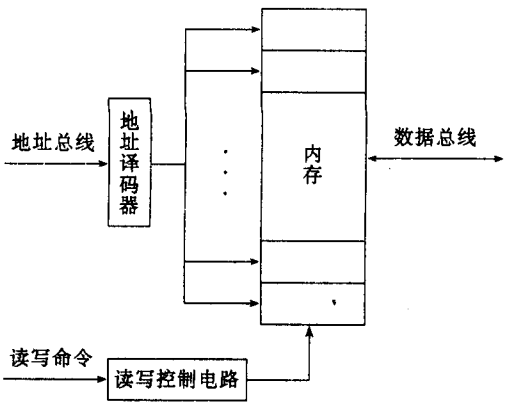


图 1.3 主存储器结构框图

现今绝大多数计算机的内存以半导体存储器为主，而外存储器通常是磁盘、磁带一类的磁表面存储器，近来出现的光盘则以其巨大的容量崭露头角。

由于外存设置在主机外部，通常归属外部设备。外存容量比内存大得多，但存取速度慢。

2. 中央处理机——CPU

CPU 是指令解释和执行的场所，是计算机的心脏，它主要由运算器、控制器和通用寄存器组成，见图 1.4。

(1) 运算器 (ALU)

其功能是完成算术运算和逻辑运算，它接受控制器发来的命令，然后执行具体操作。运算器执行何种操作将根据指令操作码而定。

(2) 数据寄存器 (DR)

它们可以是一组(若干)寄存器，用于暂存参加某种操作的数据、运算的(中间)结果，以及某种控制信息等。

从内存读出或将要写入内存的数据，也是放在某个数据寄存器中。

(3) 指令寄存器(IR)

用来保存当前正在执行的指令，从内存中取出的一条指令经数据寄存器送往 IR 中。IR 包括指令的操作码和地址码部分，操作码被送到指令译码器中译码，地址码送到地址

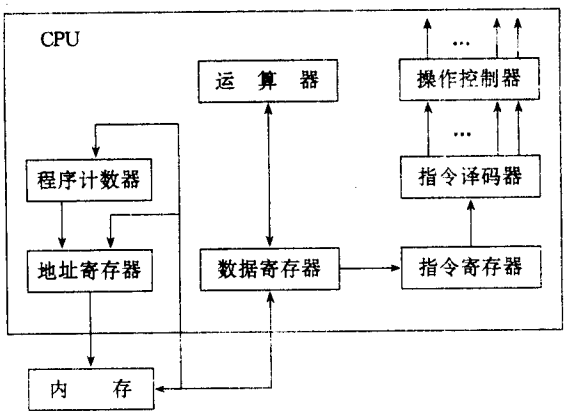


图 1.4 CPU 结构框图

形成部件。地址形成部件根据指令特征将地址码形成有效地址，送往主存的地址寄存器。对于转移指令，要将形成的有效转移地址送往程序寄存器中，实现程序的转移。

(4) 程序计数器 (PC)

又称指令地址寄存器，它的功能是存放当前要执行的指令地址。计算机通常按顺序逐条执行指令，这就是靠 PC 来实现的，每当执行完一条指令，PC 就自动加"1"即形成下一条指令地址。

当需要改变执行顺序、要求下一条指令从某个地址开始时，则必须在程序计数器中置入这个新的地址。

(5) 地址寄存器

它是主存地址总线上地址信息的发源地。无论是要执行的指令的地址，还是指令中所涉及到的操作数的地址，都必须先送到地址寄存器，然后发往主存。

(6) 操作控制器

操作控制器根据指令译码器对于指令操作码的译码，产生出实现指令功能所需要的全部动作的控制信号。这些控制信号按照一定的时间顺序发往各个部件，控制各部件的动作，所以控制器可以被看作是指挥硬件系统工作的首脑机构。

3. 输入/输出设备 (Input/Output)

简称 I/O 设备，它们实现了外部世界与主机之间的信息交换，提供了人机交互的硬件环境。

I/O 设备种类很多，例如终端就是最常用的 I/O 设备，图形终端不但能显示西文、汉字，还能显示彩色图形。打印机、绘图仪等都是常用的输出设备，I/O 设备通常设置在主机外部，也属外部设备。

1.1.3 程序的自动执行

所谓程序是为完成一项特定任务而用某种语言编写的一组指令序列。

计算机硬件系统最终只能执行由机器指令组成的程序。程序在执行前必须首先装入

内存，运行时逐条取出指令，并依次加以执行，正因为如此，人们称当前流行的计算机为存储程序式计算机。

一旦程序要执行的第一条指令地址被置入 PC，整个程序的执行将自动完成。图 1.5 描述了程序的执行过程。

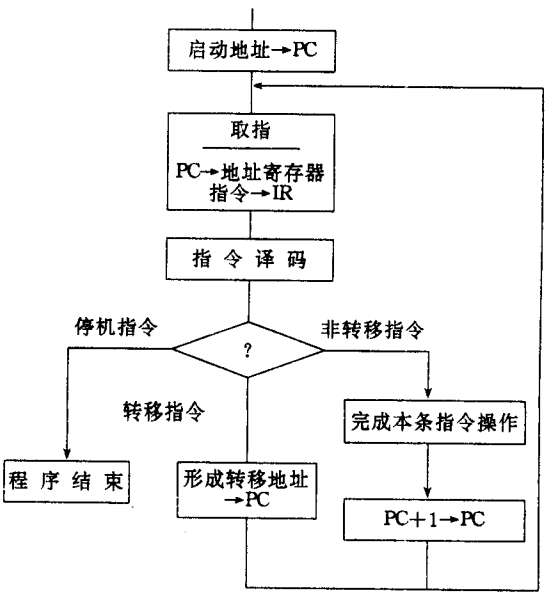


图 1.5　程序的执行过程

1.2　信息的表示与存储

计算机加工的对象是数据信息，而指挥计算机操作的是控制信息，因此计算机内部的信息可以分成两大类，如图 1.6 所示。

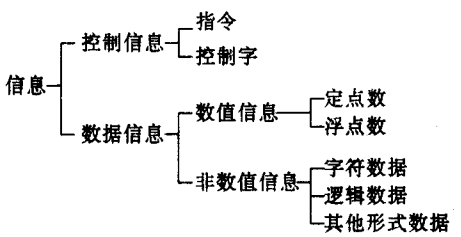


图 1.6　信息分类

本节主要介绍数据信息、控制信息的细节请参考有关硬件书籍。

1.2.1 计算机的数字系统

人们最熟悉十进制数系，但是，当我们讨论计算机时，二进制、八进制、十六进制系统表示法更为方便。无论哪种数系，其共同之处都是进位计数制。

一般说来，如果数制只采用 R 个基本符号，则称为基 R 数制，R 称为数制的“基数”，而数制中每一固定位置对应的单位值称为“权”。

进位计数制的编码符合“逢 R 进位”的规则，各位的权是以 R 为底的幂，一个数可按权展开成为多项式。例如，一个十进制数 256.47 可按权展开为：

$$256.47 = 2*10^{2} + 5*10^{1} + 6*10^{0} + 4*10^{-1} + 7*10^{-2}$$

对任一 R 进制的数 X，其值 V(X)可表示为

$$V(X) = \underbrace{\sum_{i=0}^{n-1} X_i R^i}_{\text{整数部分}} + \underbrace{\sum_{i=-1}^{-m} X_i R^i}_{\text{小数部分}}$$

这里 m、n 为正整数，R^i是第 i 位的权，在 X_0 与 X_{-1}之间用小数点隔开。通常，数字符号 X_i 应满足下列条件：

$$0 \leqslant X_i < R$$

换句话说，R 进制中的数使用 0～(R－1)个数字符号。

下面是我们需要熟悉的几种进位数制：

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

其中，十六进制的数符 A～F 分别对应十进制的 10～15。

对于二进制来说，基数为 2，每位的权是以 2 为底的幂，遵循逢二进一原则，基本符号只有两个：0 和 1。下面是二进制数的例子：

1011.01

目前，几乎所有的计算机都采用二进制的数系，原因何在呢？在机器内部，信息的表示依赖于机器硬件电路的状态，信息采用什么表示形式，直接影响到计算机的结构与性能。采用基 2 码表示信息，有如下几个优点：

1. 易于物理实现

因为具有两种稳定状态的物理器件是很多的，如门电路的导通与截止，电压的高与低，而它们恰好对应表示 1 和 0 两个符号。假如采用十进制，要制造具有十种稳定状态的物理电路，那是非常困难的。

2. 二进制数运算简单

数学推导证明，对 R 进制的算术求和，求积规则各有 R(R＋1)/2 种。如采用 十进制，就有 55 种求和与求积的运算规则；而二进制仅各有三种，因而简化了运算器等物理器件的设计。

3. 机器可靠性高

由于电压的高低、电流的有无等都是一种质的变化，两种状态分明，所以基 2 码的传递抗干扰能力强，鉴别信息的可靠性高。

4. 通用性强

基 2 码不仅成功地运用于数值信息编码(二进制)，而且适用于各种非数值信息的数字化编码，特别是仅有的两个符号 0 和 1 正好与逻辑命题的两个值“真”与“假”相对应，从而为计算机实现逻辑运算和逻辑判断提供了方便。

虽然计算机内部均用基 2 码(0 和 1)来表示各种信息，但计算机与外部交往仍采用人们熟悉和便于阅读的形式，如十进制数据、文字显示以及图形描述等，其间的转换则由计算机系统的硬件和软件来实现。

自然，基 2 码也有其不足之处，如它表示数的容量最小，表示同一个数，二进制较其他进制需要更多的位数。

下面讨论上述几种进位计数制之间的转换问题。

(1) R 进制转换为十进制

基数为 R 的数字，只要将各位数字与它的权相乘，其积相加，和数就是十进制数。

[**例 1.1**] $V(1101101.0101_2)$

$$=1*2^0+0*2^1+1*2^2+1*2^3+0*2^4+1*2^5+1*2^6+0*2^{-1}$$
$$+1*2^{-2}+0*2^{-3}+1*2^{-4}$$
$$=109.3125$$

[**例 1.2**] $V(3506.2_8)$

$$=6*8^0+0*8^1+5*8^2+3*8^3+2*8^{-1}$$
$$=1862.25$$

[**例 1.3**] $V(0.2A_{16})$

$$=2*16^{-1}+10*16^{-2}$$
$$=0.1640625$$

从上面几个例子可以看到：当从 R 进制转换到十进制时，可以把小数点作为起点，分别向左右两边进行，即对其整数部分和小数部分分别转换。对于二进制来说，只要把数位是 1 的那些位的权值相加，其和就是等效的十进制数。因此，二～十转换是最简便的，同时也是最常用的一种。

(2) 十进制转换为 R 进制

将十进制数转换为基数为 R 的等效表示时，可将此数分成整数与小数两部分分别转换，然后再拼接起来即可实现。

十进制整数转换成 R 进制的整数，可用十进制数连续地除以 R；其余数即为 R 系统的各位系数，此方法称之除 R 取余法。

我们知道，任何一个十进制整数 N，都可以用一个 R 进制数来表示：

$$N=X_0+X_1R^1+X_2R^2+\cdots+X_{n-1}R^{n-1}$$
$$=X_0+(X_1+X_2R^1+\cdots+X_{n-1}R^{n-2})R$$
$$=X_0+Q_1R$$

由此可知，若用 N 除以 R，则商为 Q_1，余数是 X_0。

同理 $Q_1=X_1+Q_2R$，Q_1 再除以 R，则商为 Q_2，余数是 X_1。依此类推：

$$Q_i = X_i + (X_{i+1} + X_{i+2}R^1 + \cdots + X_{n-1}R^{n-2-i})R = X_i + Q_{i+1}R$$

Q_i 除以 R，则商为 Q_{i+1}，余数是 X_i。

这样除下去，直到商为 0 止，每次除 R 的余数 $X_0, X_1, X_2, \cdots, X_{n-1}$，即构成 R 进制数。

[**例 1.4**] 将 57_{10}转换为二进制数，其过程如图 1.7 所示。

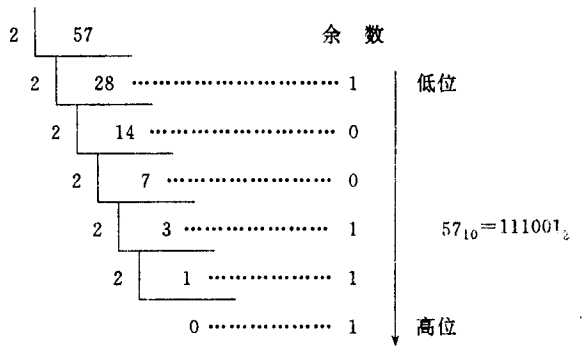


图 1.7 十进制数化为二进制数

[**例 1.5**] 将 168_{10}转换为八进制数，其过程如图 1.8 所示。

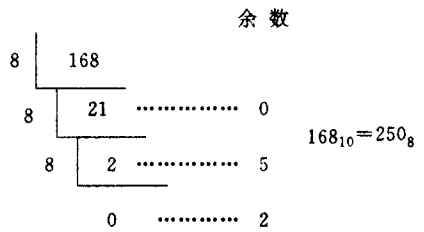


图 1.8 十进制数转换为八进制数

十进制小数转换成 R 进制数时，可连续地乘以 R，得到的整数即组成 R 进制的数，此法称为“乘 R 取整”。

我们可将某十进数小数，用 R 进制数表示：

$$V = \frac{X_{-1}}{R^1} + \frac{X_{-2}}{R^2} + \frac{X_{-3}}{R^3} + \cdots + \frac{X_{-m}}{R^m}$$

等式两边乘以 R 得到：

$$V \times R = X_{-1} + \left(\frac{X_{-2}}{R^1} + \frac{X_{-3}}{R^2} + \cdots + \frac{X_{-m}}{R^{m-1}}\right) = X_{-1} + F_1$$

式中，X_{-1}是整数部分，即 R 进制数小数点后第一位，F_1 是小数部分。

小数部分再乘以 R：

$$F_1 \times R = X_{-2} + \left(\frac{X_{-3}}{R^1} + \frac{X_{-4}}{R^2} + \cdots + \frac{X_{-m}}{R^{m-2}}\right) = X_{-2} + F_2$$

X_{-2}是整数部分，即 R 进制数小数点后第二位。

依此乘下去，直到小数部分为 0，或达到所要求的精度为止（小数部分可能永不为零）。

[**例 1.6**] 将 0.3125_{10}转换成二进制数，转换过程如图 1.9 所示。

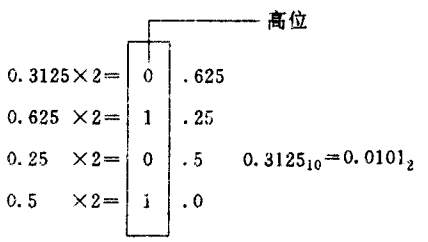


图 1.9 0.3125_{10}转换为二进制数

需要注意的是，十进制小数常常不能准确地换算为等值的二进制小数（或其他 R 进制数），有换算误差存在。

[**例 1.7**] 将 0.5627_{10}转换成二进制数，转换过程见图 1.10。

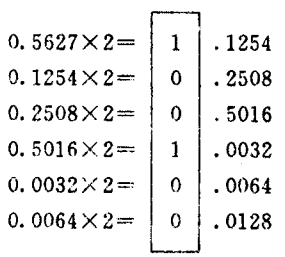


图 1.10 0.5627_{10}转换成二进制数

此过程会不断进行下去（小数位达不到 0），因此只能取到一定精度：

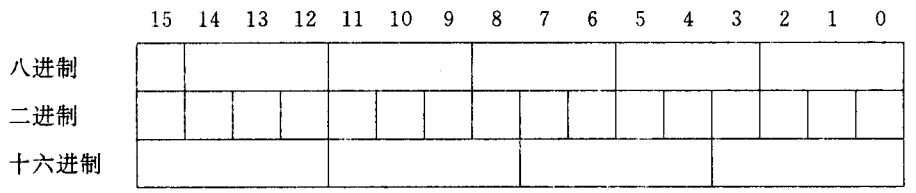
$0.5627_{10}=0.100100\cdots\cdots$

若将十进制数 57.3125 转换成二进制数，可分别进行整数部分和小数部分的转换，然后再拼在一起：

57.3125=111001.0101

(3) 二、八、十六进制的相互转换

二、八、十六进制的相互转换在应用中占有重要的地位，由于这三种进制的权之间有内在的联系，即：$2^3=8$ $2^4=16$，因而它们之间转换比较容易，即：每位八进制数相当于三位二进制数，每位十六进制数相当于四位二进制数。



（位组划分）

在转换时，位组划分是以小数点为中心向左右两边延伸，中间的 0 不能省略，两头不够时可以补 0。

例如：将 1011010.10_2 转换成八进制和十六进制数：

$\frac{001}{1}\ \frac{011}{3}\ \frac{010}{2} \cdot \frac{100}{4}$ 所以 $1011010.10_2 = 132.4_8$

$\frac{0101}{5}\ \frac{1010}{A} \cdot \frac{1000}{8}$ 所以 $1011010.10_2 = 5A.8_{16}$

将十六进制数 F7.28 变为二进制数：

十六	F				7				2				8			
二	1	1	1	1	0	1	1	1	0	0	1	0	1	0	0	0

所以 $F7.28_{16} = 11110111.00101$

将八进制数 25.63 转换为二进制数：

八	2			5			6			3		
二	0	1	0	1	0	1	1	1	0	0	1	1

$25.63_8 = 10101.110011$

1.2.2 信息存储单位

上一小节讨论到，在计算机内部各种信息都是以二进制编码形式存储，因此这里有必要介绍一下信息存储的单位。信息的单位常采用“位”、“字节”、“字”几种量纲。

1. 位(bit)：度量数据的最小单位，表示一位二进制信息。

2. 字节(byte)：一个字节由八位二进制数字组成(1 byte = 8 bit)，字节是信息存储中最常用的基本单位。计算机的存储器(包括内存与外存)通常也是以多少字节来表示它的容量。常用的单位有：

K 字节　　1 K = 1024 byte

M(兆)字节　　1 M = 1024 K

G 字节　　1 G = 1024 M

3. 字(word)：字是位的组合，并作为一个独立的信息单位处理。字又称为计算机

字，它的含意取决于机器的类型、字长以及使用者的要求。常用的固定字长有 8 位、16 位、32 位等。

信息单位用来描述机器内部数据格式，即数据（包括指令）在机器内的排列形式，如单字节数据，可变长数据(以字节为单位组成几种不同长度的数据格式)等。

4. 机器字长

在讨论信息单位时，还有一个与机器硬件指标有关的单位，这就是机器字长。

机器字长一般是指参加运算的寄存器所含有的二进制数的位数，它代表了机器的精度。机器的功能设计决定了机器的字长，一般大型机用于数值计算，为保证足够的精度，需要较长的字长，如 32 位，64 位等；而小型机，微机一般字长为 16 位、32 位等。

1.2.3 二进制数的编码表示

一个数在机内的表达形式称为“机器数”，而它代表的数值称为此机器数的“真值”。

前面已经提到，数值信息在计算机内是采用二进制编码表示。数有正、负之分，在计算机中如何表示符号呢？一般情况下，用“0”表示正号，“1”表示负号，符号位放在数的最高位。

例如：A＝(＋1011011)　　B＝(－1011011)　它们在机器中表示为：

A：	0	1	0	1	1	0	1	1
B：	1	1	0	1	1	0	1	1

其中最左边一位代表符号位，连同数字本身一起作为一个数。

数值信息在计算机内采用符号数字化处理后，计算机便可以识别和表示数符了。为了改进符号数的运算方法和简化运算器的硬件结构，人们研究了多种符号数的二进制编码方法，其实质是对负数表示的不同编码。

下面我们就来介绍几种常用的编码——原码、反码和补码。

1. 原码

将符号位数字化为 0 或 1，数的绝对值与符号一起编码，即所谓“符号——绝对值表示”的编码，称为原码。首先介绍如何用原码表示一个带符号的整数。

如果我们用一个字节存放一个整数，其原码表示如下：

$X=+0101011$　　$[X]_{原}=00101011$

$X=-0101011$　　$[X]_{原}=10101011$

这里，“$[X]_{原}$”就是机器数，X 称为机器数的真值。

而对于一个带符号的纯小数，它的原码表示就是把小数点左边一位用做符号位。

例如：$X=0.1011$　　$[X]_{原}=0.1011$

$X=-0.1011$　　$[X]_{原}=1.1011$

当采用原码表示法时，编码简单直观，与真值转换方便。但原码也存在一些问题：

(1) 零的表示不唯一，因为：

$[+0]_{原}=000\ldots0$　　$[-0]_{原}=100\ldots0$

零有二义性，给机器判零带来麻烦。

(2) 用原码进行四则运算时，符号位需单独处理，且运算规则复杂。例如加法运算，若两数同号，两数相加，结果取共同的符号；若两数异号，则要由大数减去小数，结果冠以大数的符号。还要指出，借位操作如果用计算机硬件来实现是很困难的。正是原码的不足之处，促使人们去寻找更好的编码方法。

2. 反码

反码很少使用，但作为一种编码方式和求补码的中间码，我们不妨先介绍一下。

正数的反码与原码表示相同。

负数的反码与原码有如下关系：符号位相同(仍用 1 表示)，其余各位取反(0 变 1，1 变 0)。

例如：$X=+1100110$　　$[X]_{原}=01100110$　　$[X]_{反}=01100110$

$X=-1100110$　　$[X]_{原}=11100110$　　$[X]_{反}=10011001$

$X=+0000000$　　$[X]_{原}=00000000$　　$[X]_{反}=00000000$

$X=-0000000$　　$[X]_{原}=10000000$　　$[X]_{反}=11111111$

和原码一样，反码中零的表示也不唯一。

当 X 为纯小数时，反码表示如下：

$X=0.1011$　　$[X]_{原}=0.1011$　　$[X]_{反}=0.1011$

$X=-0.1011$　　$[X]_{原}=1.1011$　　$[X]_{反}=1.0100$

3. 补码

(1) 模数的概念

模数从物理意义上讲，是某种计量器的容量，例如，我们日常生活中用的钟表，模数就是 12。钟表计时的方式是：达到 12 就从零开始(扔掉一个 12)，这在数学上是一种“取模运算(mod)”，mod 是求除法余数的算术运算符。例如：

$$14 \mod 12 = 2$$

如果现在的准确时间是 6 点整，而你的手表指向 8 点，怎样把表拨准呢？可以有二种方法：把表往后拨 2 小时，或把表往前拨 10 小时，效果是一样的，即：

$$8-2=6$$

$$(8+10) \mod 12 = 6$$

在模数系统中：

$$8-2=8+10 \quad (\mod 12)$$

上式之所以成立，是因为 2 与 10 对模数 12 是互为补数的 (2+10=12)。

因此，我们可以认可这样一个结论：在模数系统中，一个数减去另一个数，或者说一个数加上一个负数，等于第一个数加上第二个数的补数：

$$8+(-2)=8+10 \ (\mod 12)$$

我们称 10 为 −2 在模 12 下的“补码”。负数采用补码表示后，可以使加减法统一为加法运算。

在计算机中，机器表示数据的字长是固定的。对于 n 位数来说， 模数的大小是：n 位数全为 1，且最末位再加 1。实际上模数的值已经超过了机器所能表示的数的范围，因

此模数在机器中是表示不出来的。若运算结果大于模数，则模数自动丢掉，也就等于实现了取模运算。

如果有 n 位整数(包括一位符号位)，则它的模数为 2^n，如果有 n 位小数， 小数点前一位为符号位，则它的模数为 2。

(2) 补码表示法

由以上讨论得知，对一个二进制负数可用其模数与真值做加法（模减去该数的绝对值）求得其补码。

[**例 1.8**] $X = -0110$ $[X]_{补} = 2^4 + (-0110) = 1010$

$X = -0.1011$ $[X]_{补} = 2 + (-0.1011) = 1.0101$

由于机器中不存在数的真值形式，用上述公式求补码在机器中不易实现，但从上式可推导出一个简便方法。

对于一个负数，其补码由该数反码的最末位加 1 求得。

[**例 1.9**] 求 $X = -1010101$ 的补码。

$[X]_{原} = 11010101$

$[X]_{反} = 10101010$

$[X]_{补} = 10101011$

[**例 1.10**] 求 $X = -0.1011$ 的补码。

$[X]_{原} = 1.1011$ （求反码：保留符号位，其余各位求反）

$[X]_{反} = 1.0100$ （求补码：反码+0.0001）

$[X]_{补} = 1.0101$

对于正数来说，其原码、反码、补码形式相同。

补码的特点之一就是零的表示唯一：

$[+0]_{补} = \underbrace{0\,0\ldots0}_{n位}$ $[-0]_{补} = \underbrace{1\,1\ldots1}_{n位} + 1 = \boxed{1}\,\underbrace{0\,0\ldots0}_{n位}$

自动丢失

(3) 补码运算规则

当数值信息存储于计算机内时，可以采用原码方式，因原码与真值的转换十分方便，便于信息的输入、输出。

当数值信息参与算术运算时，采用补码方式是最方便的。首先符号位可作为数值参加运算，最后仍可得到正确的结果符号，符号无需单独处理；其次，采用补码进行运算时，减法运算可转换为加法运算，简化了硬件中的运算电路。

[**例 1.11**] 计算 $67-10=?$

让我们看一下计算机中的运算过程

$[+67]_{原} = 01000011$ $[+67]_{补} = [+67]_{原}$

$[-10]_{原} = 10001010$ $[-10]_{补} = 11110110$

$$
\begin{array}{rll}
 & 01000011 & [+67]_{补} \\
+ & 11110110 & [-10]_{补} \\
\hline
\boxed{1} & 00111001 = 57 &
\end{array}
$$

自然丢失

由于字长只有 8 位，因此加法最高位的进位自然丢失。

应当指出：补码运算的结果仍为补码。上例中，从结果符号位得知，结果为正，所以补码即为原码，转换成十进制数为 57。

如果结果为负，则是负数的补码形式，若要变成原码，需要对补码再求补，即可还原为原码。

[**例 1.12**] 10−67 = ?

$[+10]_{原} = 00001010 = [+10]_{补}$

$[-67]_{原} = 11000011 \quad [-67]_{补} = 10111101$

$$
\begin{array}{rl}
 & 00001010 \\
+ & 10111101 \\
\hline
 & 11000111
\end{array}
$$

$[结果]_{补} = 11000111 \quad [结果]_{原} = 10111001$

所以结果的真值为−0111001，十进制为−57。

用上面两个例子是否就可以证明，补码运算的结果总是正确的呢？下面再看一个例子：

[**例 1.13**] 85 + 44 = ?

$$
\begin{array}{rl}
 & 01010101 \\
+ & 00101100 \\
\hline
 & 10000001
\end{array}
$$

从结果的符号位可以看出，结果是一负数，但两个正数相加不可能是负数，问题出在什么地方呢？原来这是由于“溢出”造成的，即结果超出了一定位数的二进制数所能表示的数的范围。我们在后面一小节还要讨论这个问题。

1.2.4 定点数和浮点数

数值数据既有正、负之分，又有整数和小数之分。上一小节实际上是回答了如何处理正、负数的问题，本节要介绍小数点如何处理。

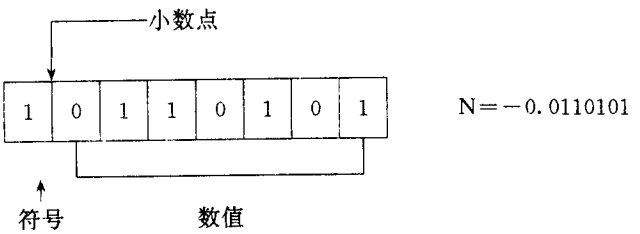
1. 定点数

定点数的小数点固定在某一位置上。用定点数表示一个数时，通常最高位（左边第一位）表示数的符号，即 0 表示正数，1 表示负数，其余位数表示定点数的二进制数值。小数点的位置，一种是放在符号位之后，数值的最高位之前，所表示的是定点小数；另

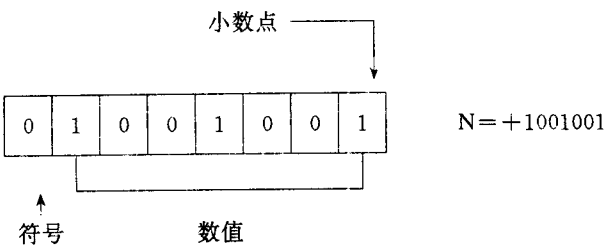
一种是将小数点放在右边最低位之后，所表示的是一个整数。

在实际机器中，没有用任何硬设备或代码表示小数点，它不过是人们的一种约定，但程序能判别它。一旦机器设计好，小数点位置就固定了，因此小数点在机器中是隐含的。至于小数点固定在哪个位置，要由机器说明书说明。

[例 1.14] 机器数的定点小数表示。



[例 1.15] 机器数的定点整数表示。



不管是定点小数还是定点整数，参与运算的数，中间结果及最后结果都必须在定点数所能表示的范围之内，否则就会产生“溢出”。因此，程序员在使用定点机进行计算时，要精心选择比例因子，以避免溢出的发生。

为了克服上述局限性，为了扩大机器的表数范围，在数值计算应用中通常都采用浮点方式，下面介绍数的浮点表示法。

2. 机器数的浮点表示

一个数 N 用浮点形式表示(即科学表示法)，可以写成：

$$N=M\times R^{E}$$

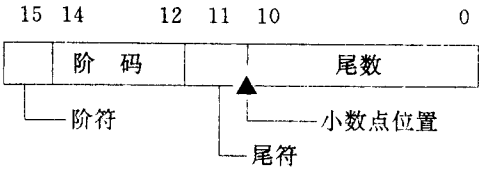
其中：R 表示基数，一旦机器定义好了基数值，就不能再改变了。因此基数在数据中不出现，是隐含的，一般取 2；

E 表示 2 的幂，称为数 N 的阶码，阶码确定了数 N 的小数点的位置，其位数反映了该浮点数所表示的数的范围；

M 表示数 N 的全部有效数字，称为数 N 的尾数，其位数反映了数据的精度。

阶码通常采用整数表示，尾数采用定点小数表示。阶码和尾数都是带符号的数，可以采用不同的码制表示法，例如尾数常用原码或补码表示，阶码多用补码表示。

浮点数的具体格式随不同机器而有所区别，例如一台 16 位机，其浮点数组成为阶码 4 位，尾数 12 位，浮点数格式如下：



下面是一个实际的例子，其中阶码，尾数分别用补码和原码表示。

0	0 1 0	1	110…… 0

表示 -0.11×2^{2}

1	1 0 1	0	110…… 0

表示 0.11×2^{-3}

在浮点运算过程中，为提高精度，要使尾数的有效数字尽可能占满尾数的位数，为此要对浮点数进行规格化操作，使尾数最高位具有非 0 的数字。如果最高位为 0，要对尾数进行左移的规格化处理，简称“左规”，尾数每左移一位，阶码减 1。

浮点数的运算较定点数要复杂一些，对于浮点数的加减法，首先要保证参加运算的数是规格化数，然后再“对阶”，即通过移动尾数使二个数的阶码相等，然后尾数相加(减)，运算结果再规格化。

1.2.5 机器数的表示范围，误差与溢出

机器中数的表示范围与数据位数及表示方法有关。一个 M 位整数(包括一位符号位)，如果采用原码或反码表示法，能表示的最大数为 $2^{m-1}-1$，最小数为$-(2^{m-1}-1)$。若用补码表示，能表示的最大数值为 $2^{m-1}-1$，最小数为-2^{m-1}。

这里要说明一点，由于补码中的“0”的表示是唯一的，故$[X]_{补}=100\cdots\cdots0$，对应的真值 $X=-2^{m-1}$，从而使补码的表示范围与原码有一点差异。(注意：补码 100……0 的形式是一个特殊的情况，权为 2^{m-1}位的 1 既代表符号又表示数值)。

例如，设 M＝8

原码表示范围为－127—＋127，即：

1	1	1	1	1	1	1	1	—	0	1	1	1	1	1	1	1

反码的表示范围也是－127—＋127，即：

1	0	0	0	0	0	0	0	—	0	1	1	1	1	1	1	1

补码的表示范围是－128—＋127，即：

1	0	0	0	0	0	0	0	—	0	1	1	1	1	1	1	1

一个 n 位小数（定点小数，小数点左边一位表示数的符号），采用原码或反码表示时，表数范围为$-(1-2^{-n})$—$(1-2^{-n})$。

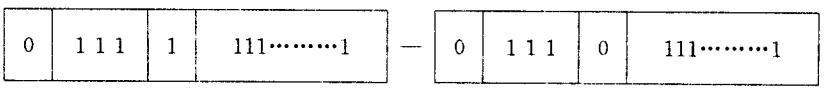
采用补码表示时，表数范围为 -1—$(1-2^{-n})$。

至于浮点数的表示范围，则由阶码位数和尾数位数决定。

若阶码用 r 位整数（补码）表示，尾数用 n 位定点小数（原码）表示，则浮点数范围是：

$$-(1-2^{-n})\times 2^{(2^{r-1}-1)} \text{ — } +(1-2^{-n})\times 2^{(2^{r-1}-1)}$$

如 r=4，n=11，尾符一位，即用 16 位字长表示一个浮点数，其数值范围：

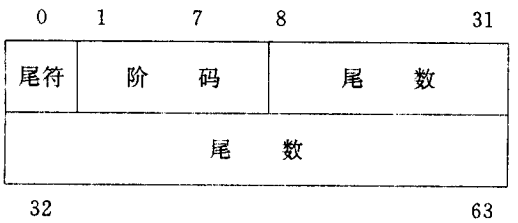


为了扩大数的表示范围，应该增加阶码的位数，每加一位，数的表示范围就扩大一倍。而要增加精度，就需要增加尾数的位数，在定长机器字中，阶码位数和尾数位数的比例要适当。但为了同时满足对数的范围和精度的要求，往往采用双倍字长甚至更多个字长来表示一个浮点数。

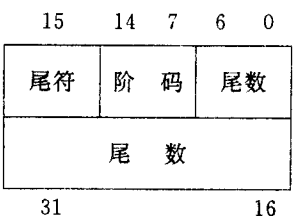
美国 IEEE 计算机协会制定的浮点数格式标准为 32 位和 64 位两种基本格式长度。32 位称为单精度，64 位称为双精度。

制定格式标准的主要目的是为了有尽可能高的精度，同时保持足够大的表数范围。

下面分别给出两种机器的浮点格式：



IBM 370 机型 长浮点数



注：在这两种格式中，阶码包括一位符号位！

PDP-11 机型浮点数

上面讨论了机器数的表示范围，我们从中可以看出，无论是定点数还是浮点数，由于受实际位数的限制，数的范围总是有一定的限制。当运算的结果超出了机器所能表示