第 1 章　计算机程序设计引论

1.1　计算机组成

计算机系统由硬件和软件两部分组成。硬件是构成计算机系统的各种物理设备的总称，如显示器、主板、鼠标、键盘等。软件是运行、维护、管理计算机的各类程序和文档，包括语言处理程序、操作系统、数据库管理系统、应用软件等。

1.1.1　计算机硬件组成

目前计算机的硬件组成仍以经典的冯・诺依曼体系结构为主。该结构可以划分为三个子系统：处理器子系统、存储器子系统和输入/输出子系统。如图 1.1 所示，这三个子系统通过总线连接在一起。

处理器也就是 CPU（中央处理器或处理器）（如图 1.2 所示），是计算机中的核心部件。在 CPU 内部有三个组成部分：算术逻辑单元、控制单元和寄存器组。算术逻辑单元 ALU 即运算器，负责进行算术和逻辑运算；控制单元即控制器，主要是控制程序的执行；寄存器组用来临时存放参与 ALU 运算的各种数据，主要有数据寄存器、指令寄存器和指令计数器等。

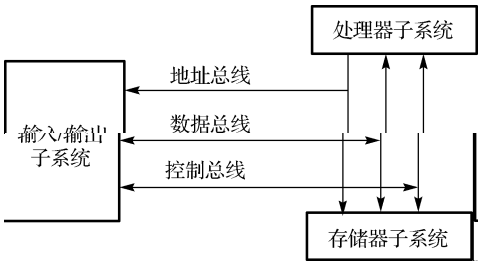


图 1.1　计算机三个子系统和总线的连接

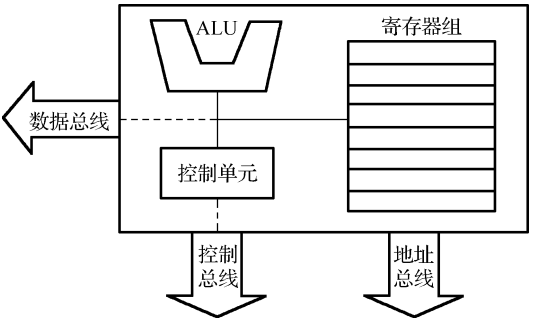


图 1.2　CPU 示意图

存储器是计算机的记忆部分，由主存储器和辅助存储器组成。主存储器简称内存，是计算机内部的存储器，与 CPU 直接进行电路连接。计算机在执行程序时，程序和运行该程序的数据都存储于此。无论 CPU 数据处理的长度是多少，在存储器系统中都是以字节为单位进行组织的，即每个存储器字节都有唯一的标识，叫做存储器地址。CPU 对存储单元内的数据进行存取操作就是通过存储器地址进行的。主存储器有随机存取存储器（RAM）和只读存储器（ROM）两种类型。RAM 是计算机主存储器系统的主要组成部分，对其单元的存取操作是随机发生的，但其中的数据会随系统断电而消失。ROM 是指其中的数据只能读出，而不能写入。这种存储器芯片是为了存放只需读取的数据和程序而设计的，数据和程序在使用之前被写入。辅助存储器简称外存，具有外设的特性，以 I/O 总线的方式和主机连接。与主存储器相比，其存储容量大，存储的信息不会因断电而消失，价格便宜，但存取速度慢。

输入/输出设备包括许多类型的设备（有时简称外设），以及连接这些设备和处理器、存储器进行

数据通信的接口电路。输入/输出设备的功能千差万别，工作速度要比 CPU 和存储器慢许多，因此需要接口在其间起到缓冲的作用，实现主机和外设交换数据速度的匹配。

连接 CPU、存储器和外设（或者外设接口）的总线就是内部总线，内部总线为三总线结构，分别是地址总线、数据总线和控制总线。地址总线是单向的，总是传送 CPU 需要对存储器和外设进行数据读/写的地址信息。CPU 通过存储器单元的地址来寻找需要进行存取操作的对应单元，而对外设（接口）也是通过统一编址的方法，按不同的地址对不同的外设进行操作的。地址总线的数目决定了机器的寻址空间大小。数据总线在 CPU、存储器和外设之间可以双向传输数据，其宽度是计算机处理能力的重要指标，一次存取的数据越多，说明 CPU 的处理能力越强。一般 16 位 CPU 是指数据总线有 16 位，32 位 CPU 是指数据总线有 32 位。控制总线不同于前两种总线，CPU 根据指令操作的类型，对其发出不同的控制信号，控制其他两种总线或其他 I/O 部件。控制总线是单个信号线的集合，在某个操作发生时只有一个或几个控制信号线起作用。对每个信号而言是单一方向的。外存既是输入设备，又是输出设备，如图 1.3 所示。

图 1.4 列出了许多计算机主机部件，你能识别出它们在计算机中分别起什么作用吗？

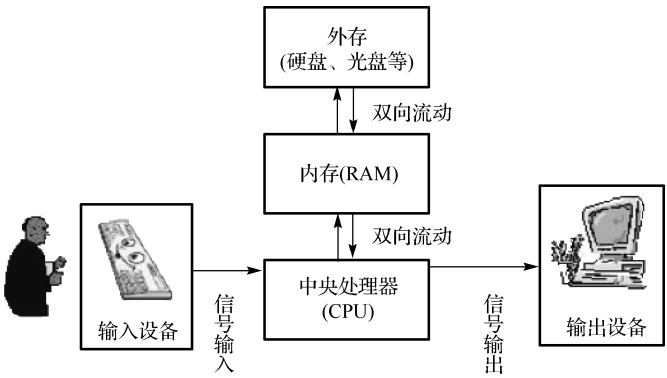


图 1.3 计算机体系结构示意图

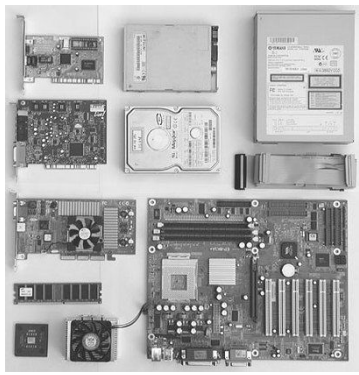


图 1.4 计算机主机部件

1.1.2 计算机软件系统

软件系统（Software Systems）由系统软件、支撑软件和应用软件组成，它是计算机系统中由软件组成的部分。它包括操作系统、语言处理系统、数据库系统、分布式软件系统和人机交互系统等。

操作系统的主要功能是资源管理、程序控制和人机交互等。计算机系统的资源可分为设备资源和信息资源两大类。设备资源指的是组成计算机的硬件设备，如中央处理器、主存储器、磁盘存储器、打印机、磁带存储器、显示器、键盘输入设备和鼠标等。信息资源指的是存放于计算机内的各种数据，如文件、程序库、知识库、系统软件和应用软件等。操作系统位于底层硬件与用户之间，是二者沟通的桥梁。用户可以通过操作系统的用户界面输入命令，操作系统对命令进行解释，驱动硬件设备，实现用户需求。

操作系统是一个庞大的管理控制程序，大致包括 5 方面的管理功能：进程与处理机管理、作业管理、存储管理、设备管理、文件管理。目前个人计算机上常见的操作系统有 Windows、Linux、MAC OS、UNIX、DOS、OS/2、XENIX、Netware 等，如图 1.5 和图 1.6 所示。

支撑软件是支撑各种软件的开发与维护的软件，又称为软件开发环境。它主要包括环境数据库、各种接口软件和工具组。著名的软件开发环境有 MyEclipse、Visual Studio.NET、SQL Server、Oracle 等。

图 1.5　Windows 操作系统界面

图 1.6　MAC 操作系统界面

应用软件是专门为某一应用目的而编制的软件，较常见的有以下几类。

（1）文字处理软件

用于输入、存储、修改、编辑、打印文字材料等，如 Word、WPS 等。

（2）信息管理软件

用于输入、存储、修改、检索各种信息，如工资管理软件、人事管理软件、仓库管理软件、计划管理软件等。这种软件发展到一定水平后，各个单项的软件相互联系起来，计算机和管理人员组成一个和谐的整体，各种信息在其中合理地流动，形成一个完整、高效的管理信息系统，简称 MIS。

（3）辅助设计软件

用于高效地绘制、修改工程图纸，进行设计中的常规计算，帮助人们寻找好的设计解决方案，如 AutoCAD 等。

（4）实时控制软件

用于随时采集生产装置、飞行器等的运行状态信息，以此为依据，按预定的方案实施自动或半自动控制，安全、准确地完成任务。

（5）其他软件

除上述之外的其他应用软件，如游戏软件、浏览器等。

1.2　信息的表示与存储

1.2.1　进制与进制转换

1．进制

进制对于现代计算机的设计具有重要的意义。人们最熟悉的是十进制数系，但是，几乎所有的计算机采用的都是二进制数系，所有的外界信息在被转化为不同的二进制数后，计算机才能对其进行传送、存储和加工处理。当我们进行程序设计时，与二进制数之间进行转换比较方便的八进制数系、十六进制数系表示法也经常使用。无论是哪种数系，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”，而数制中每个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：

$$256.47=2\times10^{2}+5\times10^{1}+6\times10^{0}+4\times10^{-1}+7\times10^{-2}$$

对任意一个 R 进制的数 X，其值 $V(X)$可表示为：

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

式中，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 X_i 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。表 1.1 所示为几种进位数制。

表 1.1　几种进位数制

进　制	基　数	进 位 原 则	基 本 符 号
二进制	2	逢 2 进 1	0,1
八进制	8	逢 8 进 1	0,1,2,3,4,5,6,7
十进制	10	逢 10 进 1	0,1,2,3,4,5,6,7,8,9
十六进制	16	逢 16 进 1	0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

其中，十六进制数 A～F 分别对应十进制数的 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 2 进 1 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

几乎所有的计算机都采用二进制的数系，采用二进制码表示信息，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很多，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 10 种稳定状态的物理电路则是非常困难的。

（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。

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。

2．进制转换

（1）R 进制数转换为十进制数

基数为 R 的数，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。例如：

$$(11111111.11)_2=1\times2^7+1\times2^6+1\times2^5+1\times2^4+1\times2^3+1\times2^2+1\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2}$$
$$=(255.75)_{10}$$

$$(3506.2)_8=3\times8^3+5\times8^2+0\times8^1+6\times8^0+2\times8^{-1}=(1862.25)_{10}$$

$$(0.2A)_{16}=2\times16^{-1}+10\times16^{-2}=(0.1640625)_{10}$$

从以上几个例子可以看到：当从 R 进制数转换到十进制数时，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。对于二进制数来说，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，二–十进制转换是最简便的，同时也是最常用的一种。

（2）十进制数转换为 R 进制数

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。

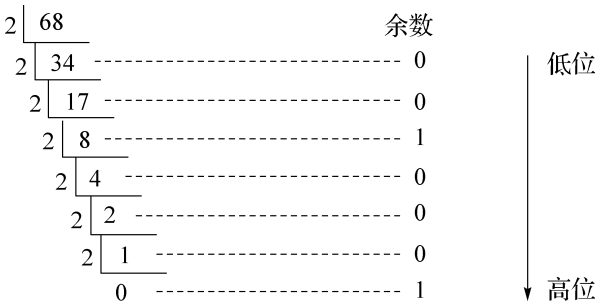
① 十进制整数转换成 R 进制的整数

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，其余数即为相应 R 进制数的各位系数。此方法称之除 R 取余法。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。以此类推：$Q_i=X_i+(X_{i+1}+X_{i+2}R^1+\cdots+X_{n-1}R^{n-2-i})R=X_i+Q_{i+1}R$，$Q_i$ 除以 R，则商为 Q_{i+1}，余数是 X_i。这样除下去，直到商为 0 时为止，每次除 R 的余数 X_0，X_1，X_2，…，X_{n-1} 即构成 R 进制数。例如，将十进制数 68 转化为二进制数，用除 2 取余法：



所以$(68)_{10}=(1000100)_2$。而将$(168)_{10}$转换为八进制数，则要用除 8 取余法：

所以$(168)_{10}=(250)_8$。

② 十进制小数转换成 R 进制小数

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”。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，得到的 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。

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

小数部分再乘以 R，得到的 X_{-2} 是整数部分，即 R 进制数小数点后第二位。依次乘下去，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 0）。

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

例如，将$(0.3125)_{10}$转换成二进制数：

高位

$0.3125 \times 2 = 0.625$

$0.625 \times 2 = 1.25$

$0.25 \times 2 = 0.5$

$0.5 \times 2 = 1.0$

所以$(0.3125)_{10} = (0.0101)_2$。

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），因为有换算误差的存在。若将十进制数 68.3125 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：$(68.3125)_{10} = (1000100.0101)_2$。

（3）二、八、十六进制数的相互转换

二、八、十六进制数的权值有内在的联系，即每位八进制数相当于三位二进制数（$2^3 = 8$），每位十六进制数相当于 4 位二进制数（2^4=16）。

二进制数，从小数点开始，向左往右分别按三（4）位为一个单元划分，每个单元单独转换成为一个八进制（十六进制）数，就完成了二进制数到八、十六进制数的转换。在转换时，位组划分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。

八（十六）进制数的每一位，分别独立转换成三（4）位二进制数，除了左边最高位，其他位如果不足三（4）位的，要用 0 来补足，按照由高位到低位的顺序写在一起，就是相应的二进制数。例如：

$(1000100)_2 = (\underline{001}\ \underline{000}\ \underline{100})_2 = (104)_8$

$(1000100)_2 = (\underline{100}\ \underline{0100})_2 = (44)_{16}$

$(1011010.10)_2 = (\underline{001}\ \underline{011}\ \underline{010}\ .\ \underline{100})_2 = (132.4)_8$

$(1011010.10)_2 = (\underline{0101}\ \underline{1010}\ .\ \underline{1000})_2 = (5A.8)_{16}$

$(F)_{16} = (\underline{1111})_2$

$(7)_{16} = (\underline{0111})_2$

$(F7)_{16} = (\underline{1111}\ \underline{0111})_2 = (11110111)_2$

1.2.2 信息存储单位

在计算机内部，各种信息都是以二进制编码形式存储的，因此有必要介绍一下信息存储的单位。信息的单位通常采用“位”、“字节”、“字”。

（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）：一字节由 8 位二进制数字组成（1byte = 8bit）。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）通常也是以字节多少来表示它的容量的。常用的单位有：

K 字节　　1K = 1024byte

M 字节　　1M = 1024K

G 字节　　1G = 1024M

（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，它的含义取决于机器的类型、字长及使用者的要求。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。

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，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，如 32 位、64 位等。

1.2.3　数值的表示

1．码制

一个数在机器内的表达形式称为“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的。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，8 位二进制数 $A = (+1011011)$ $B = (-1011011)$，则它们在机器中可以表示为：

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，人们研究了符号数的多种二进制编码方法，其实质是对负数表示的不同编码。下面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。如果用一字节存放一个整数，其原码表示如下：

$$X = +0101011 \qquad [X]_{原} = 00101011$$

$$X = -0101011 \qquad [X]_{原} = 10101011$$

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，它的原码表示是把小数点左边一位用做符号位。例如：

$$X = 0.1011 \qquad [X]_{原} = 0.1011$$

$$X = -0.1011 \qquad [X]_{原} = 1.1011$$

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…0，$[-0]_{原}$=100…0。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，不妨先介绍。

① 正数的反码与原码表示相同。

② 负数的反码与原码有如下关系：负数反码的符号位与原码相同（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，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（或取余）运算（mod）”。“%”是 C 语言中求除法余数的算术运算符。例如：14%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？可以有两种方法：把表往后拨 2 小时，或把表往前拨 10 小时，效果是一样的，即：

$$8-2=6,\ (8+10)\bmod 12=6$$

在模数系统中：$8-2=(8+10)\bmod 12$

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，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：

$8+(-2)=8+10$（在对 12 取模的情况下）

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。例如：

$X=-0110$ $[X]_{补}=24+(-0110)=1010$

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

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。例如，求 $X=-1010101$ 的补码：

$[X]_{原}=11010101$

$[X]_{反}=10101010$

$[X]_{补}=10101011$

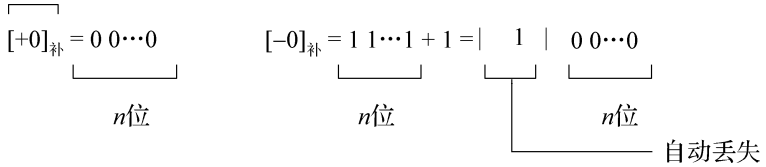
例如，求 $X=-0.1011$ 的补码：

$[X]_{原}=1.1011$ （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：



这种简便的求补码的方法经常被简称为“求反加 1”。本书不对此做推导和证明，读者只要初步了解补码的表示方法，在学习后续章节时对内存中数据的存储形式不感到费解就可以了。

③ 补码运算规则

采用补码表示的另一个好处是，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，符号无须单独处理；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。

例如，计算 67–10，我们看一下计算机中的运算过程（这里用下脚标的方式表明数的进制）：

$$
\begin{array}{lll}
[+67_{10}]_{原} = (01000011)_2 & & [+67_{10}]_{补} = [+67_{10}]_{原} \\
[-10_{10}]_{原} = (10001010)_2 & & [-10_{10}]_{补} = (11110110)_2 \\
\quad (0\,1\,0\,0\,0\,0\,1\,1)_2 & & [+67_{10}]_{补} \\
+ \quad (1\,1\,1\,1\,0\,1\,1\,0)_2 & & [-10_{10}]_{补} \\
\hline
1 \quad (0\,0\,1\,1\,1\,0\,0\,1)_2 = (57)_{10} & &
\end{array}
$$

最高位的进位自然丢失

由于字长只有 8 位，因此加法最高位的进位自然丢失，达到了取模效果（即丢掉一个模数）。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。例如，对于 10–67：

$$
\begin{array}{ll}
[+10_{10}]_{原} = (00001010)_2 = [+10_{10}]_{补} & \\
[-67_{10}]_{原} = (11000011)_2 & [-67_{10}]_{补} = (10111101)_2 \\
\quad (0\,0\,0\,0\,1\,0\,1\,0)_2 & \\
+ \quad (1\,0\,1\,1\,1\,1\,0\,1)_2 & \\
\hline
\quad (1\,1\,0\,0\,0\,1\,1\,1)_2 &
\end{array}
$$

$[结果]_{补} = (11000111)_2$，$[结果]_{原} = (10111001)_2$。

所以结果的真值为–0111001，十进制数为–57。以上两个例子是否可以说明补码运算的结果总是正确的呢？下面再看一个例子：$(85)_{10} + (44)_{10}$

$$
\begin{array}{ll}
\quad (0\,1\,0\,1\,0\,1\,0\,1)_2 & \\
+ \quad (0\,0\,1\,0\,1\,1\,0\,0)_2 & \\
\hline
\quad (1\,0\,0\,0\,0\,0\,0\,1)_2 &
\end{array}
$$

从结果的符号位可以看出，结果是负数。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。

2．小数点的处理——定点数与浮点数

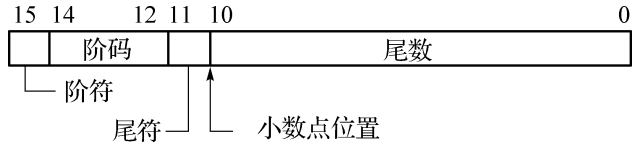
数值数据既有正、负之分，又有整数和小数之分，本节将介绍小数点如何处理。在计算机中通常都采用浮点方式表示小数，下面介绍数的浮点表示法。

一个数 N 用浮点形式表示（即科学表示法），可以写成：$N = M \times R^E$。

式中，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。在人工计算中，一般采用十进制，10 就是基数。在计算机中一般用二进制，因此以 2 为基数。E 表示 R 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，假设有一台 16 位机，其二进制浮点数组成为阶码 4 位，尾数 12 位，则浮点数格式如下：



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
1	1 0 1	0	110 … 0

表示 $(-0.11\times10^{10})_2$

表示 $(0.11\times10^{-11})_2$

3．数值的范围

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 $[X]_{补}=100\cdots0$，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）。对补码的表示范围，本书不做证明，读者如果感兴趣，可以自行验证。

例如，设 $M=8$，则原码表示范围是–127～+127，反码的表示范围也是–127～+127。补码的表示范围是–128～+127。

一个 n 位定点小数，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：

$$-(1-2^{-n})\times2^{(2^{r-1}-1)}-1\sim+(1-2^{-n})\times2^{(2^{r-1}-1)}$$

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字长来表示一个浮点数。

1.2.4　非数值信息表示

在计算机内部，非数值信息也是采用 0 和 1 两个符号来进行编码表示的。非数值数据又可划分为文字、多媒体两大类。

（1）西文字符

西文字符的编码，ASCII 码是“美国信息交换标准代码”的简称，在这种编码中，每个字符用 7 个二进制位表示，即从 0000000 到 1111111 可以给出 128 种编码，可用来表示 128 个不同的字符。一个字符的 ASCII 码通常占用一字节，由 7 位二进制数编码组成，故 ASCII 码最多可表示 128 个不同的符号。

由于 ASCII 码采用 7 位编码，未用到字节的最高位，故在计算机中一般保持为“0”，在数据传输时可用做奇偶校验位。

汉字的编码：我国使用的是“国家标准信息交换用汉字编码”（G132312—1980 标准），该标准码是二字节码，用两个 7 位二进制数编码表示一个汉字，并收入了 6763 个汉字。

汉字在计算机中的表示有多种编码，如汉字输入码，输入码进入计算机后，必须转换成汉字内码，才能进行信息处理。为了最终显示、打印汉字，再由内码转换成汉字字形码。此外，为使不同的汉字处理系统之间能够交换信息，还必须设有汉字交换码。

（2）图像数据

位图是指存储在计算机中的由图像中的许多点构成的点阵图。构成位图的这些点称为像素，用以描述图像中各图像点的亮度与颜色。

图像分辨率是指图像点阵中行数和列数的乘积。

屏幕分辨率是指计算机显示器屏幕上的最大显示区域以水平和垂直方向的像素个数的乘积。

像素分辨率是指一个像素的长和宽的比例。

图像的颜色深度是指图像中可能出现的不同颜色的最大数目。颜色深度值越大，图像的色彩越丰富。位图中每个像素都用一位或多位二进制位来描述其颜色的信息。

图像文件的大小是指存储整幅图像所需的磁盘字节数，计算式为：

图像文件大小=图像分辨率×颜色深度÷8

（3）视频数据

视频信号经数字化处理之后，以视频文件格式存储在计算机中。视频信号也可视为图像数据中的一种，由若干有联系的图像数据连续播放而形成。计算机所播放的视频信号是数字信号，与电视上播放的模拟视频信号是不一样的。由于视频信号的数据量很大，所以在存储和传输数字视频过程中，要采用压缩编码技术。

（4）音频数据

音频数据在计算中可分为数字音频文件和 midi 文件。数字音频文件是将声音信号数字化处理后的数据文件。midi 文件是通过一串时序命令，用于记录电子乐器键盘弹奏的信息，包括键名、力度和时值长短等，是对乐谱的一种数字式描述。

1.3　现代计算机的发展

在推动计算机发展的众多因素中，电子元器件的发展起着决定性的作用；另外，计算机系统结构和计算机软件技术的发展也起着重要的作用。从生产计算机的主要技术来看，计算机的发展过程可以划分为 4 个阶段。

1.3.1 第一代：电子管时代

第一代（1946—1958 年）计算机的特征是采用电子管作为计算机的逻辑元件，内存储器采用水银延迟线，外存储器采用磁鼓、纸带、卡片等。运算速度只有每秒几千次到几万次基本运算，内存容量只有几千个字。用二进制表示的机器语言或汇编语言来编写程序。由于体积大、功耗大、造价高、使用不便，此类计算机主要用于军事和科研部门进行数值计算。代表性的计算机是 1946 年美籍匈牙利数学家冯·诺依曼与他的同事们在普林斯顿研究所设计的存储程序计算机 IAS，本意是要预测天气变化，虽然在预测天气方面还不够准确，但是 IAS 成功地完成了氢弹设计的复杂计算工作。它的设计体现了“存储程序原理”和“二进制”的思想，产生了所谓的冯·诺依曼型计算机结构体系，对后来计算机的发展有着深远的影响。电子管如图 1.7 所示。

图 1.7 电子管

1.3.2 第二代：晶体管时代

第二代（1958—1964 年）计算机的特征是用晶体管代替了电子管；大量采用磁芯做内存储器，采用磁盘、磁带等做外存储器；体积缩小、功耗降低，运算速度提高到每秒几十万次基本运算，内存容量扩大到几十万字。同时计算机软件技术也有了很大的发展，出现了 Fortran、ALGOL-60、COBOL 等高级程序设计语言，大大方便了计算机的使用。因此，它的应用从数值计算扩大到数据处理、工业过程控制等领域，并开始进入商业市场。代表性的计算机是 IBM 公司生产的 IBM-7094 机和 CDC 公司的 CDC-1604 机，机型如图 1.8 所示。

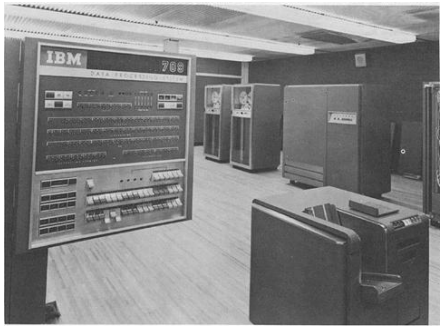


图 1.8 IBM 推出的 IBM-7094 大型计算机

1.3.3 第三代：集成电路时代

第三代（1964—1975 年）计算机的特征是用集成电路（Integrated Circuit，IC）代替了分立元件，集成电路是把多个电子元器件集中在几平方毫米的基片上而形成的逻辑电路。第三代计算机的基本电子元件是每个基片上集成几个到十几个电子元件（逻辑门）的小规模集成电路和每个基片上集成几十

个元件的中规模集成电路。第三代计算机已开始采用性能优良的半导体存储器取代磁芯存储器，运算速度提高到每秒几十万次到几百万次基本运算，在存储器容量和可靠性等方面都有了较大的提高。同时，计算机软件技术的进一步发展，尤其操作系统的逐步成熟，是第三代计算机的显著特点。多处理机、虚拟存储器系统及面向用户的应用软件的发展，大大丰富了计算机软件资源。为了充分利用已有的软件解决软件兼容问题，出现了系列化的计算机。最有影响的是 IBM 公司研制的 IBM-360 计算机系列。这个时期的另一个特点是小型计算机的应用。DEC 公司研制的 PDP-8 机、PDP-11 系列机及后来的 VAX-11 系列机等，都曾对计算机的推广发挥了极大的作用，如图 1.9 所示。

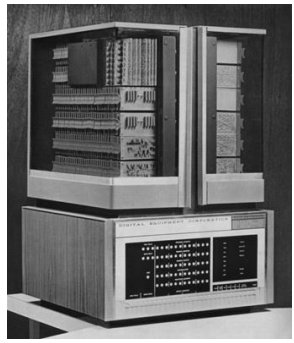


图 1.9　DEC 公司推出的 PDP-8 型计算机

1.3.4　第四代：大规模集成电路时代

第四代（1975 年—）计算机的特征是以大规模集成电路（每个基片上集成成千上万个逻辑门，Large-Scale Integration，LSI）来构成计算机的主要功能部件，主存储器采用集成度很高的半导体存储器。运算速度可达每秒几百万次甚至上万亿次基本运算。在软件方面，出现了数据库系统、分布式操作系统等，应用软件的开发已逐步成为一个庞大的现代产业。第四代计算机外观中的笔记本电脑效果如图 1.10 所示。

图 1.10　笔记本电脑

当然，人类探索的脚步不会停止，最新一代机器也正在研制之中，它是一种采用超大规模集成电路的智能型计算机。这一代的基本体系结构与前四代有很大不同。前四代基本属于冯 • 诺依曼型的，即通常说的五官型（存储器、运算器、控制器、输入和输出设备）；而第五代机器将采用分布的、网络的、数据流的体系结构。在硬件方面，它由推理机、知识库和智能接口机组成；在软件方面，将由一个程序分别对硬件三大部分进行操作管理。它的主要特点是采用平行处理、联想式检索、以 PROLOG 为“机器语言”、以应用程序为用户呈现，因此，智能化程度显著提高，是一种更接近于人的计算机系统。

1.4　计算机程序设计

计算机程序（Program）是人们为解决某种问题用计算机可以识别的代码编排的一系列加工步骤。计算机能严格按照这些步骤完成对数据的处理。程序的执行过程实际上是对程序所表达的数据进行处理的过程。一方面，程序设计语言提供了一种表达数据与处理数据的功能；另一方面，编程人员必须按照语言所要求的规范（语法要求）进行编程。

1.4.1　程序与指令

计算机中最基本的处理数据的单元是计算机的指令。孤零零的一条指令本身只能完成计算机的一个最基本的功能，如实现一个加法运算或实现一个大小的判别。计算机所能实现的指令的集合称为计算机的指令系统。

虽然计算机指令所能完成的功能很基本，并且指令系统中指令的个数也很有限，但一系列指令的组合却能完成一些很复杂的功能，这也是计算机的奇妙与强大所在。计算机一系列指令的有序组合就构成了程序。

假设某台虚拟的计算机指令系统由以下几条指令构成，其中指令的第一部分是指令名（如 Store，Add 等），随后的几个部分是指令处理所涉及的数据（如 x，y，z，p 等）。

（1）指令 1。Input x：将当前的输入数据存储到内存 x 单元中。

（2）指令 2。Output x：将内存 x 单元的数据输出。

（3）指令 3。Add x y z：将内存 x 单元的数据与 y 单元的数据相加，并将结果存储到内存 z 单元。

（4）指令 4。Sub x y z：将内存 x 单元的数据与 y 单元的数据相减，并将结果存储到内存 z 单元。

（5）指令 5。BranchEq x y p：比较 x 与 y，若相等，则程序跳转到 p 处执行，否则，仍按一般顺序执行下一条指令。

（6）指令 6。Jump p：程序跳转到 p 处执行。

（7）指令 7。Set x y：将内存 y 单元的值设为 x。

上述简单的 7 条指令通过不同的组合就可以完成不同的功能。例如，下面是一段由上述指令组成的虚拟程序，完成的功能是：输入三个数，将它们相加，最后输出结果。

① Input A; 输入第一个数据到存储单元 A 中；

② Input B; 输入第二个数据到存储单元 B 中；

③ Input C; 输入第三个数据到存储单元 C 中；

④ Add A B D; 将 A、B 相加，并将结果存在 D 中；

⑤ Add C D D; 将 C、D 相加，并将结果存在 D 中；

⑥ Output D; 输出 D 的内容。

又例如，下面一段程序通过不断执行加法运算（Z=Z+A）实现两个整数（A、B）的乘法功能，即把 A 累加 B 次就可以获得 A*B 的结果。

① Input A; 输入第一个数据到存储单元 A 中；

② Input B; 输入第二个数据到存储单元 B 中；

③ Set 0 X; 将 X 设为 0，X 用以统计 A 累加的次数；

④ Set 0 Z; 将 Z 初始值设为 0，Z 用以存放 A*B 的最后结果；

⑤ BranchEq X B 9; 判别 X 与 B 是否相等，若相等，说明 A 已累加了 B 次，程序跳转到第 9 条指令，输出结果；

⑥ Add Z A Z; Z=Z+A；

⑦ Add 1 X X; X=X+1；

⑧ Jump 5; 程序跳转到第 5 条指令，继续循环执行第 6、7 条指令；

⑨ Output Z; 输出 Z 的值，该值等于 A*B。

上述程序中，A、B 用于存储准备相乘的两个数（第 1、2 条指令）；Z 用于存储乘法运算的结果，开始时初始化为 0（第 4 条指令），随后不断将 A 累加到 Z 上（第 6 条指令），而累加次数的控制通过 X 来实现；X 开始时设为 0（第 3 条指令），随后 A 每累加一次，就将 X 加 1（第 7 条指令）；但 X 被累加 B 次时（第 5 条指令），Z 的值就是 A*B 的结果了，最后输出所产生的结果（第 9 条指令）。

一般情况下，程序是按指令排列的顺序一条一条执行的。但稍微复杂的程序往往需要通过判断不同的情况执行不同的指令分支，第 5 条指令（BranchEq）就是这种情况。另外，在许多情况下，有些指令需要被重复地执行，如第 6、7 条指令，所以也需要有指令能强行改变程序中指令执行的顺序，如第 9 条指令（Jump）。

实际的程序在计算机中是用由 1、0 组成的指令码来表示的，也就是说，程序也是 0、1 组成的序列。当然，计算机能够识别这个序列。实际上，程序与数据一样共同存储于存储器中。当程序要运行时，当前准备运行的指令从内存被调入 CPU 中由 CPU 处理这条指令。这种将程序与数据共同存储的思想，就是目前绝大多数计算机采用的冯・诺依曼模型的存储程序概念。

然而，为什么程序一定要由计算机中的指令所组成呢？一方面，通过定义计算机可直接实现的指令集使得程序在计算机中的执行变得简单，计算机硬件系统只要实现了指令，就能方便地实现相应的程序；另一方面，需要计算机实现的任务成千上万，如果每个任务都相对独立，与其他程序之间没有公共的内容，编程工作将十分困难。这就是计算机科学中很重要的一个概念——“重用”的体现。

如果程序设计直接用 0、1 序列的计算机指令来写，那将是一件难以忍受的事。所以，人们设计了程序设计语言，用这种语言来描述程序，同时应用一种软件（如编译系统）将用程序设计语言描述的程序转换成计算机能直接执行的指令序列。

1.4.2　程序设计语言的功能

程序设计语言是程序员用来编写程序的手段，是程序员与计算机交流的语言。程序员为了让计算机按自己的意愿处理数据，必须用程序设计语言来表达所要处理的数据，同时用程序设计语言来表达数据处理的流程。因此，程序设计语言必须具有表达数据和处理数据（称为控制）的能力。

1．数据表达（Data Representation）

世界上的数据有多种多样，而语言本身的描述能力总归是有限的。为了使计算机程序设计语言能有效地、充分地表达各种各样的数据，一般将数据抽象为若干类型。数据类型（Data Type）是对某些具有共同特点的数据集合的总称。例如，平常所说的整数、实数，就是数据类型的例子。对数据类型来说，涉及两方面的内容：该数据类型代表的数据是什么（数据类型的定义域），能在这些数据上做些什么（操作或称运算）。例如，整数类型所包含的数据是{…，–2，–1，0，1，2，…}，而+、–、*、/就是作用在整数上的运算。

在程序设计语言中，一般都事先定义好几种基本的数据类型，供程序员直接使用，如整型、实型（浮点型）、字符型等。这些基本数据类型在程序中的具体对象主要有两种形式：常量（又称常数 Constant）与变量（Variable）。常量值在程序中是不变的，例如，123 是一个整数常量，12.3 是一个实型常量，‘a’是一个字符常量。变量则可对它做一些相关的操作，改变它的值。例如，可以通过“int i”来定义一个新的变量 i，然后可以对该变量进行某种操作，如赋值（如 i=20;）。

同时，为了使程序员能更充分地表达各种复杂的数据，程序设计语言还提供了构造新的具体数据类型的手段，如数组（Array）、结构（Structure）、文件（File）、指针（Pointer）等。例如，在 C 语言中，可以通过“int a[10]”来定义一个由 10 个整数组成的数组变量。这样变量 a 所代表的就不是一个整数，而是由 10 个整数组成的有序序列，其中的每个整数都称为 a 的分量。

程序设计语言提供的基本数据类型及构造复杂类型的手段，如数组、结构等，为有限能力的程序设计语言表达客观世界中多种多样的数据提供了良好的基础。

2．流程控制（Flow Control）

程序设计语言除了能表达各种各样的数据外，还必须提供一种手段来表达数据处理的过程，即程序的控制过程。程序的控制过程通过程序中的一系列语句来实现。

当要解决的问题比较复杂时，程序的控制过程也会变得十分复杂。一种比较典型的程序设计方法是：将复杂程序划分为若干相互独立的模块（Modular），使完成每个模块的工作变得单纯而明确，在

设计一个模块时不受其他模块的牵连。同时，通过对现有模块进行积木式的扩展，就可以形成复杂的、更大的程序模块或程序。这种程序设计方法就是结构化的程序设计方法（Structured Programming）。C 语言就是支持这种设计方法的典型语言。

在结构化程序设计方法中，一个模块可以是一条语句（Statement）、一段程序或一个函数（Function）（子程序）等。

一般来说，从程序流程的角度看，模块只有一个入口、一个出口。这种单入单出的结构为程序的调试（查错，Debug）提供了良好的条件。多入多出的模块结构将使得程序的调试变得异常困难。

按照结构化程序设计的观点，任何程序都可以将模块通过三种基本的控制结构进行组合来实现。这三种基本的控制结构如下。

（1）顺序控制结构（Sequential Control Structure）：一个程序模块执行完，按自然顺序执行下一个模块。

（2）分支控制结构（Branch Control Structure）（又称选择结构）：计算机在执行程序时，一般是按照语句顺序执行的，但在许多情况下需要根据不同的条件来选择所要执行的模块。例如，检测某种条件是否满足，如果条件满足，执行某些指令，否则，执行另外一些指令。例如，周末时，根据天气情况决定去郊游还是在房间里学习，就是一种分支控制。

（3）循环控制结构（Loop Control Structure）：有时经常需要重复地执行某些相同的处理过程，即重复执行某个模块。当然，重复执行这些模块一般是有条件的。也就是说，检测某些条件，如果条件满足，就重复执行相应的模块。

顺序结构是一种自然的控制结构，通过安排语句或模块的顺序来实现。所以，对一般程序设计语言来说，需要提供表达分支控制和循环控制的手段。

以上三种控制方式称为语句级控制。它实现了程序在语句间的跳转。

例如，对于 1.4.1 节中求两个整数（A、B）相乘的例子，该程序实际上存在着问题：当 B 为负数时，该程序将终止不了！如果 B 是负数，可以将 A、B 均乘上–1，再应用 1.4.1 节中的思路求解，就可得到正确的结果。其求解过程大致可描述如下。

① 分别输入两个数到 A、B 两个变量；

② 如果 B 是负数，那么 B=B*(–1)，A=A*(–1)；

③ 设 X=0，Z=0；

④ 当 X 不等于 B 时，重复执行以下操作：

a）Z=Z+A；

b）X=X+1；

⑤ 输出 Z。

上述处理过程基本上是从步骤 1 顺序执行到步骤 5，其中步骤 2 就是一种分支控制，而步骤 4 就是循环控制。

当要处理的问题较复杂时，为了增强程序的可读性，以及便于程序维护，往往将程序分为若干相对独立的子程序。例如，在 C 语言中，子程序的作用是由函数来完成的。函数通过一系列语句的组合来完成某种特定的功能（如求整数 *n* 的阶乘）。当程序一些地方需要相应功能时，不用重新写一系列代码，可直接调用，并根据需要传递不同的参数（如求 *n* 阶乘函数中的 *n*）。同一个函数可以被一个或多个函数（名括自己）调用任意多次。函数调用时可传递零个或多个参数（Argument），函数被调用的结果将返回给调用函数。这种涉及函数定义和调用的控制称为单位级控制。所以，程序设计语言的另一个功能就是提供单位级控制的手段，即函数的定义（Definition）与调用（Call）手段。

1.4.3　程序设计语言的语法

程序员利用程序设计语言来编写程序以处理相应的问题。在程序中，可能要表达数据，包括定义相应的用于存储数据的变量；还要用程序设计语言来描述需要的数据处理过程，包括语句间的控制和子程序间的控制。为了让计算机能理解程序员在程序中所描述的这些工作，用程序设计语言所写的程序必须符合相应语言的语法（Grammar）。

编写程序就像用某种自然语言（如中文）来写文章，首先语法要通，即要符合语言所规定的语法规则。当然，语法通了并不意味着文章就符合要求了，有可能词不达意、离题万里。后者就是在程序调试（查错）时需要发现的事，即找出程序中的错误（非语法错误）。这是一个非常需要耐心和经验的过程。而语法错误的检查则要相对容易得多。

一般，把用程序设计语言所编写的未经编译的程序称为源程序（Source Code，又称源代码）。从语法的角度看，源程序实际上是一个字符序列。这些字符序列按顺序分别组成了一系列“单词”。这些“单词”包括语言事先约定好的保留字（Reserved Words，如用于描述分支控制的 if、else，用于描述数据类型的 int 等）、常量（Constant）、运算符（Operator）、分隔符及程序员自己定义的变量名、函数名等。

在这些“单词”中，除了运算符（如+、–、*、/）、普通常量（如–12、12.34、‘a’）、分隔符（如“;”、“（”、“)”等）外，其他主要是有关用来标识（表示）变量、函数、数据类型、语句等的符号，这些标识符号称为标识符（Idendifier）。任何程序设计语言对标识符都有一定的定义规范，即只有满足这些规范的字符的组合，才能构成该语言所识别的标识符。

“单词”的组合形成了语言有意义的语法单位，如变量定义、表达式（Expression）、语句、函数定义等。一些简单语法单位的组合又形成了更复杂的语法单位，最后一系列语法单位组合成程序。就像在作文中，单词组合成主语、谓语、宾语等，主语、谓语、宾语组合成句子，简单的句子又可以组合成更复杂的句子，句子又组合成段落，段落又组合成文章（相当于程序）。

计算机要理解程序，首先要识别出程序中的“单词”，继而识别出各种语法单位。当计算机无法识别程序中的“单词”或语法单位时，说明该程序出现了语法错误。这些识别工作是由编译程序来做的。

下面就以 C 语言为例，简要说明 C 语言最主要的语法要素。

1．C 语言的主要“单词”

（1）标识符。C 语言的标识符规定由字母、数字及下画线组成，且第一个字符必须是字母或下画线。如_name1 是一个合法的标识符，而 1eft&right 就是非法的。在 C 语言中，标识符中字母的大小写是有区别的。最主要的标识符是：保留字和用户自定义标识符。

（2）保留字。也称关键字，它们是 C 语言规定的、赋予它们以特定含义、有专门用途的标识符。这些保留字也主要与数据类型和语句有关。如 int（整数类型）、float（浮点数类型）、char（字符类型）、typedef（类型定义），以及与语句相关的 if、else、while、for、break 等。

（3）自定义标识符。包括在程序中定义的变量名、数据类型名、函数名及符号常量名。一般来说，为了便于程序阅读，经常取有意义的英文单词作为用户自定义标识符（如前面程序中定义的 n、fact 等）。

（4）常量。常量是有数据类型的，如整型常量 123、实型常量 12.34、字符常量‘a’、字符串常量“hello world!”等。

（5）运算符。代表对各种数据类型实际数据对象的运算。如+（加）、–（减）、*（乘）、/（除）、%（求余）、>（大于）、>=（大于等于）、==（等于）、=（赋值 Assignment）等。绝大多数运算为双目运

算（涉及两个运算对象），也有单目（涉及一个运算数）和三目（三个运算数）运算，如 C 语言中的条件运算“?:”就是一个三目运算。

（6）分隔符，如“;”、“[”、“]”、“(”、“)”、“#”等。

2．C 语言的主要语法单位

（1）表达式。运算符与运算对象（可以是常量、函数、变量等）的有意义组合就形成了表达式。如 2+3*4、i+2<j 等。表达式中可以包含多种数据类型的运算符，不同运算符间有不同的运算优先顺序。如 i+2<j 中，+比<先进行运算。

（2）变量定义。变量也有数据类型，所以在定义变量时要说明相应变量的类型。变量的类型不同，它在内存中所占的空间大小也会有所不同。变量定义的最基本形式是：“类型名 变量名;”。如“int i;”就定义了一个整型变量 i。

（3）语句。语句是程序中最基本的执行单位，程序的功能就是通过对一系列语句的执行来实现的。C 语言中的语句有多种形式。

- 最简单的语句：表达式加“;”。在 C 语言中，赋值也被认为是一种运算，如“i= j+2”（把 j 加 2 的结果给变量 i）就是一个包含“+”和“=”两种运算的表达式，“+”优先级较高。在上述表达式后加“;”就组成了一个执行赋值过程的语句。
- 分支语句：实现分支控制过程，即根据不同的条件执行不同的语句（或语句模块）。具体有两种形式：双路分支的 if-else 语句与多路分支的 switch 语句。如下列 if-else 语句就实现了求变量 a、b 的较大值，并把它赋给 x 的功能。这个 if-else 语句通过判别 if 后面的表达式（a>b）是否成立，来决定执行“x=a;”（如果成立）还是执行“x=b;”（如果不成立）。

```
if (a > b) x = a;
else x = b;
```

- 循环语句：C 语言实现循环控制的过程有三种形式，即 while 语句、for 语句、do-while 语句等。例如，下列 while 循环语句就是实现求 1 到 100 的和，并把结果存于变量 sum 中。在这个循环中，(i<=100)是循环执行的条件，即只要这个条件满足，一对“{…}”中的循环体就会一直循环执行。应该注意到，由于循环体每循环一次，i 被加 1（i=i+1），所以，当循环到一定时，i 的值就会超过 100，即循环条件（i<=100）不再满足，此时，循环结束。

```
sum = 0; /*初始化 sum 和 i */
i = 1;
while (i <= 100) { /*通过循环把 1，2，…，100 分别加到 sum 中*/
    sum = sum + i;
    i=i+1;
}
```

- 复合语句（Compound Statement）：通过一对花括号“{ }”将若干语句顺序组合在一起，就形成了一个程序段。如前面 while 语句中的“{sum=…}”。

（1）函数定义与调用。函数是完成特定任务的独立模块，是 C 语言中唯一的子程序形式。通常，函数的目的是接收 0 个或多个数据（称为函数的参数），并至多返回一个结果（称为函数的返回值）。函数的使用最主要涉及函数的定义与调用。函数定义的主要内容是通过编写一系列语句来规定其所完成的功能。完整的函数定义涉及函数头和函数体。其中，函数头包括函数的返回值类型、函数名、参数类型；而函数体是一个程序模块，规定了函数所具有的功能。函数调用则通过传递函数的参数并执