DJS100系列机

程序设计基础

清华大学电子系程序系统专业

1979.3.

目　录

注：目录中章、节标题是完整的，小节标题仅列出了较重要的一部分。

前　言

程序设计是伴随着电子计算机的出现而产生的一门学科。简单地说，程序设计就是把人们需要计算机所做的工作写成一种计算机能接受的“指示书”（程序），计算机能根据它进行工作以完成一定的任务。设计的好坏最后就体现在计算机工作的效果上。一个好的设计不仅使得计算机给出正确的结果，而且工作周期短。一个不好的设计往往包含着这样或那样的错误，修改不方便，操作繁琐，工作周期长。所以程序设计便成了一门必要的学科，以研究如何正确地、科学地进行程序设计。

随着计算机程序工作的发展，程序设计的方式也不断改善和多样化。这体现在“指示书”（程序）所用的语言从简单到复杂，从低级到高级。所谓低级语言是指机器语言和汇编语言它们是与具体所用的计算机指令系统相关的。所谓高级语言是指如 FORTBAN、BASIC、ALGOL、PL/1 等程序语言，它们的用法与具体的计算机指令系统无关。目前计算机的使用者基本上只要掌握高级语言就可以了，但对于计算机软件设计和维护人员来说，机器语言和汇编语言必须作为基本知识来掌握，因为它们是软件的基础，高级语言是不可能完全代替它们的。鉴于如上所述机器语言和汇编语言是与具体计算机型号有关的，因此叙述时不得不选定某个计算机型号为对象，不但要介绍它的指令系统，而且要介绍有关的硬部件逻辑功能。因为只有详细地了解硬件的逻辑功能，才能正确、灵活地进行程序设计。那末根据某一特定计算机来进行教学是不是会失去一般性呢？实际上，就程序设计思想来说是有共同性的，而且计算机的功能设计也是大同小异的，所以从学习的角度，只要学会某一种计算机上的汇编语言程序设计，则对其它计算机而言是可以举一返三的，只是需要重新熟悉一下指令系统和有关硬件逻辑功能。

本册教材所介绍的是 DJS100 系列计算机的机器语言和汇编语言的基本知识。基本上为四部分内容：① 计算机基本知识和工作原理（第一、二章）；② 指令用法（第三、四、五章）；③ 汇编语言用法（第六章）；④ 浮动程序的联系和装配（第七章）。属于软件工作人员的入门知识。

DJS100 系列计算机是我国生产的一种小型多功能计算机。所谓系列机是指某一种类型的计算机，在这一类计算机中分成若干档，同一档的计算机是相同的，不同档的计算机有所差别。其差别可以在所用的元器件方面、在机器线路和结构设计方面，在运行速度指标和功能方面等，但各档计算机的指令系统都必须是一致的。

到目前为止在 DJS100 系列中包括有 DJS100—10，DJS100—20，DJS100—30 和 DJS100—40 四档，这四档的运行速度有区别，低档速度低、高档速度高。它们的功能也有所差别，DJS100—40 的功能较其它几档强些。因为它们的指令系统是一致的，所以凡低档机器能运行的程序同样可以在系列的高档机器上运行，这称为“向上兼容性”。在高档能运行的程序只要不涉及高档机器上的特殊功能则同样也可以在低档

机器上运行。所以对一个计算机系列来说其程序系统是通用的，这就是系列机的好处——程序系统的通用性。

本教材中涉及一些 DJS100 系列机的基本软件，如基本汇编，改进基本汇编，扩充汇编，各种引导程序和查错程序等，学习时可参考 DJS130 联合设计组编写的《DJS-130 计算机使用说明》和有关软件资料。（注：DJS130 机是 DJS100—30 档中的一个型号，在同一档中还有 DJS131 和 DJS135 等型号。）

第一章　计算机的基本知识

§ 1.1　计算机的结构

电子数字计算机是一种快速的计算工具。它具有很高的运算速度，每秒钟能执行几十万到上亿次的运算，它还具有自动化程度高、存贮量大，精确度高，使用范围广等特点。但是不管它的速度多高、结构多复杂，它的工作原理和人们打算盘或者使用台式计算机的过程还是很相似的。为了掌握计算机的使用，必须了解它的基本工作原理和结构。为此我们首先来分析一下大家所熟悉的打算盘的过程。

例如要计算：

$$325 \times 17 + 57 \times 24$$

计算步骤如下：

① 把 325 打在算盘的左边；
② 把 17 打在算盘的右边；
③ 左边和右边的数相乘；
④ 把乘的结果记录在纸上；
⑤ 把 57 打在算盘的左边；
⑥ 把 24 打在算盘的右边；
⑦ 左边的数和右边的数相乘；
⑧ 把记在纸上的上次的乘积和当前算盘上的乘积相加；
⑨ 记录运行结果；
⑩ 停止。

计算时，操作人员根据事先编排好的这一计算顺序一步一步地进行操作。

由于计算过程的每一步都是在人的操作控制下进行的所以实际打算盘时，人们并不一定列好计算顺序。但使用电子计算机就必须把整个计算过程交给它，才能由它自动连续地进行计算，因此，人们必须予先根据计算问题，按一定的规则，列好计算顺序。我们把这由人事先编排好的计算过程的序列称为程序。程序中指示每一步操作的命令称为指令。

电子计算机要能模拟人们的手工的计算过程并能自动连续地进行计算，必须包括以下组成部分：①运算器；②存储器；③控制器；④输入器；⑤输出器。这些部分的功能将在后几章详细介绍。在本章中仅对它们的作用作一概述。

1.1.1　存储器的作用

计算机要能高速地进行计算，就不能每运算一步都由人来控制一次并指示下一步的

操作，而必须由计算机自动连续地进行计算。为此，需要把人事先编排好的程序和数据存放在计算机中。用以存放程序和数据的部件称为存储器。它相当于手工计算时纸的作用。要用的数据从存储器中取出来，计算的中间结果存进去。

存储器能存放成千上万个数据或操作指示命令，象一个大楼住成百上千个人一样。一个大楼分成许多小房间，每个房间有一个编号。为了便于找到大楼里的某一个人，必须知道他住的房间的号码，也就是说要知道他住的地址。根据这个道理，为了便于寻找存储器中的数据，把存储器分成很多小单元，每个单元给以编号，称为地址。DJS 100系列机的存储器最多可有 $2^{15}=32768$ 个单元。

存储器和楼房不同的是：当从存储器某一单元取出一个数后，这个数在这个单元并没有消失，可以不断地从一个单元取出同一的数。只有当一个新的数被送入单元，才使原来的数消失，而为新的数代替。

存储器使计算机具有“记忆”的功能。这正是电子计算机区别于其它简单计算工具的一个显著特点。

1.1.2 运算器的作用

计算机必须能执行若干种基本的运算和操作。如加、减、取数、记结果、条件判断、停机等。执行这一部分功能的部件称为运算器。它相当于算盘的作用。它具有能寄存数据的寄存器和一套运算电路。

1.1.3 控制器的作用

控制器起着相当于人工计算过程中人的控制作用。它从存储器逐条取出程序中的指令，加以分析，根据指令要求发出各种控制讯号，控制机器各部分协调地工作，以完成指令所规定的操作。逐条指令执行，直至程序结束，整个计算过程得以完成。

1.1.4 输入器的作用

输入器用来把程序和数据输送到存储器。纸带输入器和电传打字机是我国目前常用的输入手段。程序和数据予先用穿孔方式记录在纸带上，通过纸带输入机送入存储器，或者直接由电传打字机送入机器。

1.1.5 输出器的作用

输出器用来把计算结果从计算机里送出来。常用的输出设备有电传打字机，行式打印机、纸带凿孔机，绘图仪等。

输入输出设备在人和机器之间起桥梁的作用。人通过输入设备向计算机发送信息，控制它的工作，计算机通过输出设备向人报告工作情况和结果。

电子计算机工作过程大致如下：先由人根据任务编好程序，把程序和数据用穿孔方式记录在纸带上，用这条穿好孔的纸带通过纸带输入机把程序和数据送入存储器，然后启动计算机执行程序，控制器便根据程序逐条执行指令，直到遇停机指令为止。

计算机只能执行若干种简单的基本操作，而通过程序可以使计算机完成相当复杂的工作。不仅是完成计算任务，而且可以帮助人们进行各种事务性的工作。如文档的分类和检索、报表数据的分析和处理、仪器仪表的控制等等。所以电子计算机的用途日益广泛，“计算机”这个名称实际上已经名不符实了。

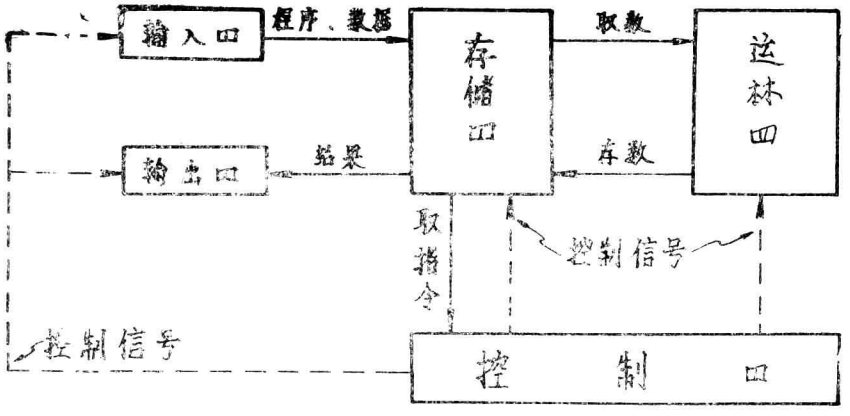

图 1—1 计算机各部分联系简图

§ 1.2 计算机中数的表示

劳动人民在长期的生产实践过程中，根据不同需要，采用了各种不同的计数方法。十进制计数法就是一种人们最习惯的常用计数法。十进制用十个数码 0、1、2、3、5、6、7、8、9、 来计数。计数规则是逢十进一。如人民币的十分为一角、 十角为一元，就是用的十进制。

此外，在日常生活中还遇到别的计数法。如老秤的十六两为一斤是十六进制；钟表里六十秒为一分、六十分为一小时则是六十进制。所以在实际生产和生活中，并不只是采用十进制计数法，而是根据不同情况，采用不同的计数方法。十进制成为人们最常用最熟悉的一种，这也是因为劳动人民在长期的生产实践活动中经常使用自己的双手——这个具有十个手指的“计算机”来计数形成的。

1.2.1 二进制数与八进制数

在电子计算机中一般都采用二进制计数法，即逢二进一的计数法。在二进制中只有两个数码：0 和 1。所有的整数都只用 0 和 1 的序列来表示。下面我们对比一下二进制和十进制的计数法：

二进制计数法（逢二进一）	十进制计数法（逢十进一）
0 + 1 = 1	0 + 1 = 1
1 + 1 = 10	1 + 1 = 2

10 + 1 = 11	2 + 1 = 3
11 + 1 = 100	3 + 1 = 4
100 + 1 = 101	4 + 1 = 5
101 + 1 = 110	5 + 1 = 6
110 + 1 = 111	6 + 1 = 7
111 + 1 = 1000	7 + 1 = 8
1000 + 1 = 1001	8 + 1 = 9
1001 + 1 = 1010	9 + 1 = 10
⋮	⋮

二进制计数与十进制计数方式虽然不同但二进制数与十进制数之间是有着一一对应关系的。我们从上面的对比就可以理解这一点。

二进制数与十进制数的对应表：

十进制数	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	…
二进制数	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000	…

我们可以写成：

⋮

$(101)_2 = (5)_{10}$

$(110)_2 = (6)_{10}$

$(111)_2 = (7)_{10}$

$(1000)_2 = (8)_{10}$

⋮

其中（　　）$_2$表示括号中的数是二进制的；（　　）$_{10}$ 表示括号中的数是十进制的；（　　）$_2$ =（　　）$_{10}$ 表示左边二进制数与右边十进制数相等（或说对应）、意即同一个数的两种不同表示法。

根据逢二进一的原理，我们有：

$(1)_2 = (1)_{10} = (2^0)_{10}$

$(10)_2 = (2)_{10} = (2^1)_{10}$

$(100)_2 = (2 \times 2)_{10} = (2^2)_{10}$

$(1000)_2 = (2 \times 2 \times 2)_{10} = (2^3)_{10}$

$$(10000)_2=(2\times2\times2\times2)_{10}=(2^4)_{10}$$

$$\vdots$$

我们利用这个性质可以很容易地将一个二进制数化成它所对应的十进制数。例如：

$$(10111)_2=(1\times2^4+0\times2^3+1\times{}^22+1\times2^1+1\times2^\circ)_{10}=(23)_{10}$$

对于小数怎么办呢？根据逢二进一原理我们同样可以得到：

$$(0.1)_2=(2^{-1})_{10}$$

$$(0001)_2=(2^{-2})_{10}$$

$$(0.001)_2=(2^{-3})_{10}$$

$$\vdots$$

所以，例如二进制数 101.11 便可用下式化成十进制数：

$$(101.11)_2=(1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+1\times2^{-2})=(5.75)_{10}$$

计算机为什么要采用二进制呢？因为二进制每一位只有两种状态 0 和 1，在机器中表示两种状态是很方便的。例如，电压的高或低，信号的有或无，纸带上有孔或无孔等都可以分别表示 1 或 0 。但要表示十种状态就较困难。 另外， 二进制由于只有两个数据，所以运算比较简单。

二进制数有个缺点，写起来很长也难于识别，如果把一个二进制数自个位起向左每三位为一组进行分组，每一组的状态用一个数码表示。这样写起来可以比较简短，也容易识别。由于三位二进制能表示八种不同的状态，所以我们可以用八个数 0 ～ 7 来对应它们如下：

0 0 0	0 0 1	0 1 0	0 1 1	1 0 0	1 0 1	1 1 0	1 1 1
0	1	2	3	4	5	6	7

不难理解，如果我们把一个二进制数的每一组（三位二进制）用一位对应的数来表示，那末这样表示的数便是八进制数，它的计数规则是“逢八进一”。

从上面的讨论我们可以直接列出二进制和八进制数之间的一一对应关系：

八进制数	0	1	2	3	4	5	6	7	10	11	12	13	14	15	16	17	20	…
二进制数	0	1	10	11	100	101	110	111	1000	1001	1010	1011	1100	1101	1110	1111	10000	…

一个二进制数要写成八进制形式只要从个位起向左每三位用其对应的八进制数代替即得；一个八进制数要写成二进制形式只要将它的每一位分别写成对应的三位二进制形式即得。例如：

$$(\underbrace{1\,0}\quad\underbrace{0\,0\,1}\quad\underbrace{1\,0\,1})_2$$

$$=(2\qquad1\qquad5)_8$$

$$(3\ 7\ 7)_8$$

$$=(\underbrace{1\ 1}\ \ \underbrace{1\ 1\ 1}\ \ \underbrace{1\ 1\ 1})_2$$

$$(3\ 7\ 7+1)_8=(400)_8$$

$$=(\underbrace{1\ 1}\ \ \underbrace{1\ 1\ 1}\ \ \underbrace{1\ 1\ 1}+1)_2=(\underbrace{1\ 0\ 0}\ \ \underbrace{0\ 0\ 0}\ \ \underbrace{0\ 0\ 0})_2$$

八进制数可以看作是二进制数的一种缩写，所以我们今后往往就用对八进制数的讨论来代替对二进制数的讨论。

1.2.2 八进制数与十进制数之间的转换

由于计算机内部采用二进制，而人们习惯的是十进制，因此在数据输入输出时要进行数制的转换工作。输入计算机前的数是十进制形式，输入计算机后需要转换成二进制形式，这个转换简称为“十化二”；在数据输出时又需将二进制形式的数转换成十进制形式，这个转换简称为“二化十”。这些转换工作都是由计算机执行程序来实现的。

下面我们来讨论这些转换工作的算法。讨论时用八进制代替二进制，所以看到的是八进制与十进制之间的转换。

1.2.2.1 八进制整数转换成十进制整数

与 1.2.1 中我们对二进制的讨论类同，可以得到这样的结论： 任何一个八进制整数都可以表示成 8 的幂（8^0，8^1，8^2，　8^3，……）的倍数之和， 以得到对应的十进制数。例如：

$$(251)_8=(2\times8^2+5\times8^1+1\times8^0)_{10}=(169)_{10}$$

1.2.2.2 十进制整数转换成八进制整数

我们用 1.2.2.1 中的结论和待定系数的办法进行逆运算。

例：将十进制数转换成八进制整数。

假定 169 所对应的八进制数为 $a_ka_{k-1}\cdots\cdots a_1a_0$

即 $$(169)_{10}=(a_ka_{k-1}\cdots\cdots a_1a_0)_8,$$

则有 $$(a_ka_{k-1}\cdots\cdots a_1a_0)_8=(a_k\times8^k+a_{k-1}\times8^{k-1}+\cdots\cdots+a_1\times8^1+a_0\times8^0)_{10}$$

$$=8\times(a_k\times8^{k-1}+a_{k-1}\times8^{k-2}+\cdots\cdots+a_1)_{10}+a_0$$

得 $$(169)_{10}=8\times(a_k\times8^{k-1}+a_{k-1}\times8^{k-2}+\cdots\cdots+a_1)_{10}+a_0$$

这个等式告诉我们：如果用 8 去除$(169)_{10}$，则其商为 $(a_k\times8^{k-1}+a_{k-1}\times8^{k-2}+\cdots+a_1)$，而其余数即为 a_0。既然 $(169)_{10}/8=(21)_{10}+1$，

则得 $$a_0=1$$

$$(21)_{10}=(a_k\times8^{k-1}+a_{k-1}\times8^{k-2}+\cdots+a_1)_{10}$$

同理，如再用8去除 $(21)_{10}$，则其余数即为 a_1。这过程继续下去便可得到全部 a_i（$0 \leqslant i \leqslant K$）。现将过程续完：$(21)_{10}/8 = (2)_{10} + 5$

得 $$a_1 = 5$$

$$(2)_{10} = (a_k \times 8^{k-2} + a_{k-1} \times 8^{k-3} + \cdots + a_2)_{10}$$

再用8除前商 $$(2)_{10}/8 = (0)_{10} + 2$$

得 $$a_2 = 2$$

由于商已得0，过程结束。随得：

$$a_2 = 2, \quad a_1 = 5, \quad a_0 = 1;$$

$$(169)_{10} = (251)_8$$

上述过算过程可以简单地用连除式来表示：

```
8 | 1 6 9   | 1 ——— 第一次余数
  8 | 2 1   | 5 ——— 第二次余数
    8 | 2   | 2 ——— 第三次余数
      0     |
```

把所有余数按序自右向左排便得八进制数：251。

任一十进制整数均可通过逐次“除八取余”的方法得到对应的八进制整数。

1.2.2.3 八进制小数转换成十进制小数

根据八进制“逢八进一”的计数原则，可以得到这样的结论：任何一个八进制小数都可以表示成8的负指数幂（8^{-1}, 8^{-2}, 8^{-3}, ……）的倍数之和，以得到对应的十进制数。例如：

$$(0.54)_8 = (5 \times 8^{-1} + 4 \times 8^{-2})_{10} = (0.6875)_{10}$$

1.2.2.4 十进制小数转换成八进制小数

我们用 1.2.2.3 中的结论和待定系数的办法进行逆运算。

例：将十进制小数 0.2875 转换成八进制小数。

假定 0.6875 所对应的八进制数为 $0.a_1a_2\cdots a_k$，则有

$$(0.6875)_{10} = (0.a_1a_2\cdots\cdots a_k)_8$$

$$= (a_1 \times 8^{-1} + a_2 \times 8^{-2} + \cdots + a_k \times 8^{-k})_{10}$$

我们用8乘之，则得

$$(0.6865)_{10} \times 8 = (5.5)_{10} = 5 + (0.5)_{10}$$

$$= (a_1 \times 8^{-1} + a_2 \times 8^{-2} + \cdots + a_k \times 8^{-k})_{10} \times 8$$

$$= a_1 + (a_2 \times 8^{-1} + \cdots + a_k \times 8^{-k+1})_{10}$$

观察等式，

$$5+(0.5)_{10}=a_1+(a_2\times 8^{-1}+\cdots+a_k\times 8^{-k+1})_{10},$$

等式两边的整数和小数应分别相等。因此有

$$a_1=5$$

$$(0.5)_{10}=(a_2\times 8^{-1}+\cdots+a_k\times 8^{-k+1})_{10}$$

再用8乘小数部分，取整数可得到 a_2。重复这样的过程，直到小数部分等于0或已求到所要求的位数时为止。将所得的全部整数按序从左向右排列，并在最前面加小数点便得八进制数。

现将本例过程续完：

$$\begin{aligned}(0.5)_{10}\times 8&=4+(0.0)_{10}\\&=(a_2\times 8^{-1}+\cdots+a_k\times 8^{-k+2})_{10}\times 8\\&=a_2+(a_3\times 8^{-1}+\cdots+a_k\times 8^{-k+2})_{10}\end{aligned}$$

得

$$a_2=4$$

$$0=(a_3\times 8^{-1}+\cdots+a_k\times 8^{-k2+})_{10}$$

过程结束。随得

$$a_1=5;\quad a_2=4$$

$$(0.6875)_{10}=(0.54)_8$$

上述计算过程可以简单地用连乘式来表示：

$$\begin{array}{rl}&0.6875\\&\times\quad 8\\\hline\text{第一个整数}\longrightarrow&5\,|\,.5000\\&\times\quad 8\\\hline\text{第二个整数}\longrightarrow&4\,|\,.0000\end{array}$$

把所得整数按序自左向右排列，并在最前面加上小数点便得八进制数：.54

任一个十进制小数均可通过逐次“乘八取整”的方法得到对应的八进制小数。

以上讨论了整数的及小数的八进制与十进制之间的转换方法，这些讨论具有一般性，也适用于其它进制与十进制的转换，只须将计算过程中的8换成相应进位制的基数即可。譬如，整数十化二的规则只需将十化八的“除8取余”规则改成“除2取余”即得。

对于任意数的八进制与十进制之间的转换请读者自行讨论（譬如，结合例子：$(251.54)_8=(169.6875)_{10}$）。

1.2.3 正负数的表示法

在计算机中正数和负数如何区分呢?

在计算机中数的正负号也是用二进制的一位来表示的，因此正负号也数码化了。通常，用最左的一位表示正负号，“0”表示正，“1”表示负，这一位称为数的“符号位”。

在计算机中负数的表示方法有若干种，它们各有特点，程序设计者根据其需要来选择某一种作为数的表示的依据。根据对负数表示的不同我们把数的表示法分成几种，一般说是三种——原码表示法，补码表示法和反码表示法。

1.2.3.1 原码表示法

一个数 x 的**原码表示**，记为 $[x]_{原}$。数 x 可正可负，例如 x = 10110；x = −10100 等。在原码表示式中，当 x 为正时，最左边的符号位用“0”表示；当 x 为负时，用“1”表示。数值部分不变。

假定符号位处于整数第 n 位（即从个位起向左第 n 位），则 $[x]_{原}$ 可用下式来定义：

$$[x]_{原}=\begin{cases}|x|; & \text{当 } x\geqslant 0\\ 2^{n-1}-x; & \text{当 } x\leqslant 0\end{cases}$$

例：将 +110001 和 −110001 表示成连同符号位共 8 位的原码表示式。

当 x = +110001，则 $[x]_{原}=00110001$

当 x = −110001，因 n = 8，所以：

$$[x]_{原}=2^{7}-(-110001)=10110001$$

在原码表示式中，“0”有两种表示形式，即 +0 和 −0 是分别表示的：

$$[+0]_{原}=00\cdots0$$

$$[-0]_{原}=10\cdots0$$

原码表示简单，但在计算机中作加减运算时不太方便。比如，两个数相加，如果是同号，则数值相加正负号不变；如果是异号，数值部分应相减，还应比较两个数那个绝对值大，以确定谁减谁及结果的正负号。这就带来一些麻烦。下面介绍的补码表示法在这点上就比原码方便。

1.2.3.2 补码表示法

我们用钟表的例子来说明补码的概念。

假定时钟的响声告诉你现在是 4 点正，而你的表却指示七点正，你发现原来你的表停了。这时，你应该将表的分针逆时针方向转（反转）3 圈（−3），就可得到

$$7-3=4$$

但是如果你不是反转 3 圈，而是按顺时针方向转（正转）9 圈（+9），也可以指到 4

点。我们作这样的分析：

$$7+9=16=12+4$$

因为 16 已超过了 12，表盘上表示不出来，因此 12 就自动消失了，过程又从 0 开始，于是我们得到最后结果 4 。

钟表的计时特点是：任何数不超过12，满 12 则又归零。这个特点可利用“模”的运算来描述。所谓“模”是指某一给定的数。所谓模的运算是说所有参予运算的数均用给定的模数除之，取其余数而运算，运算结果也以模数除之，取其余数。这样，钟表的计时特点便归结为一种模的运算，模取 12。上面的讨论便可用下面两式表示：

$$7-3=4 \quad (\mathrm{mod}12)$$

$$7+9=4 \quad (\mathrm{mod}12)$$

其中（mod12）表示是模运算，模数取 12。

设模数为 C，A＜C，B＜C，若有

$$A+B=C=0 \quad (\mathrm{mod}C)$$

则我们称 A、B 互为补码（对模 C 而言）。

例如，3和9在模 12 时互为补码，即 3 是 9 的补码，9 是 3 的补码（mod12）。

模的运算有个很重要的特点：减某一个数就相当于加此数的补码。例如，

$$7-3=7+9=4 \quad (\mathrm{mod}12)$$

$$0-3=0+9=9 \quad (\mathrm{mod}12)^*$$

另外，若用本数的补码来代替本数的负值，则结果不变。例如，

$$7+(-3)=7+9=4 \quad (\mathrm{mod}12)$$

$$(-7)+(-3)=5+9=2 \quad (\mathrm{mod}12)$$

因此，如果在模的运算中用补码表示负值，减法用加补码代替，则加、减运算可以简化，无需根据数的正负号和数值来制定实际应进行什么操作。

在电子计算机中，数的大小受二进制位数的限制。如果二进制整数部分有 n 位(包括符号位)，则就以 2^n 为模。

一个数 x 的补码表示，记为 $[x]_{补}$。则 $[x]_{补}$ 可用下式来定义：

$$[x]_{补}=\begin{cases}|x|; & 当 0\leqslant x<2^{n-1}\\ 2^n+x; & 当 -2^{n-1}\leqslant x<0 \ (\mathrm{mod}2^n)\end{cases}$$

假定 n＝8，则根据补码表示的定义，当 x＝0 时，$[x]_{补}$＝00000000，当x＝1111111 时（即 2^7-1），$[x]_{补}$＝01111111；当 x＝－10000000 时（即 -2^7），$[x]_{补}$＝

* 等式 $-3=9(mod12)$是成立的，因为等式两边若同加以 3，则等式两边都得 0 。

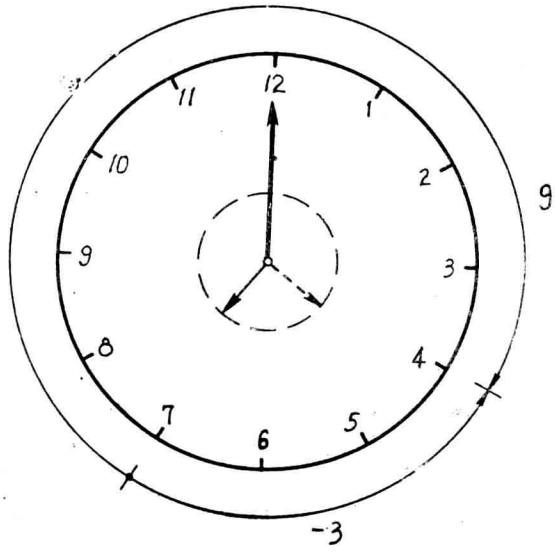

图 1—2　钟表计时——模 12 计数

10000000；当 $x=-1$ 时，$[x]_{补}=11111111$。

由补码表示的定义，可以直接列出几条关于补码表示的基本性质。

性质 1　当数 **x** 为正时，表示式仍为 x。

性质 2　当数 x 的表示式已定，则 $-x$ 的表示式即为 x 的补码（模 2^n）。

性质 3　所有正值的表示式最左位为 0，所有负值的表示式最左位是 1。

上面在钟表计时的例子中已经讨论过补码表示的本质；就是为了把正、负数的关系转换成一种纯数值的关系。从而使加、减运算变成加和求补的纯数值运算，避免了运算中对数的正负号进行判断的复杂过程。而性质 3 告诉我们表示式最左位起着正负的作用，实际上这一位不仅起符号位的作用，它在运算中确实是起着一位数的作用，对于运算来说根本不考虑它的正负号意义。

例：求 $-59+61$ 的值。

现用二进制补码来表示：

$$(-59)_{10}=11000101 \quad (\mathrm{mod}2^8)$$

$$(61)_{10}=00111101 \quad (\mathrm{mod}2^8)$$

$$\begin{array}{r} 11000101 \\ +\ 00111101 \\ \hline 1\,|\,00000010 \end{array}$$

溢出消失——　　——结果

$$00000010=(2)_{10}$$

验证：

$$-59+61=2。$$

所以　　$11000101+00111101=00000010(\mathrm{mod}2^8)$

这样的纯数值运算，完全反映了正负数运算的要求。

因为 $A-B=A+(-B)$，根据性质 2，可得到又一性质。

性质 4　用补码表示的数的减运算可以用加以减数的补码来代替。

设数 x，则其补码应为 $2^n-x(\mathrm{mod}2^n)$。而

$$2^n-x=(2^n-1-x)+1。$$

2^n-1 在二进制表示中是 n 位 1 即 $\underbrace{1\ 1\cdots1}_{n\text{位}}$。x 也用二进制表示，若把其每一个二进位上的数都用相反的值代之，则称为 x 的**反码**。例如，$x=10101101$，则 01010010 是 x 的反码。显然，$\underbrace{1111\cdots1}_{n}-x$ 便得 x 的反码。所以 $(2^n-1-x)+1$ 的意思是：x 的反码加 1。由此我们得到下一性质。

性质 5　一个二进制数 x 的补码，可以用 x 的反码加一的办法得到。

例：求 $x=10101101$ 的补码。

解　x 的反码 $\bar{x}=01010010$，

x 的补码 $\hat{x}=\bar{x}+1=01010010+1=01010011\quad(\mathrm{mod}2^8)$

根据补码表示的定义知道当模为 2^n 时，数的表示范围是 $-2^{n-1}\leqslant x<2^{n-1}$，所以当 $m>n$ 时，所有在模 2^n 能表示的数，在模 2^m 时也能表示，反之则不成立。例如当 $x=+101$ 时，

$$[x]_{补}=0101\quad(\mathrm{mod}2^4)$$

$$[x]_{补}=00101\quad(\mathrm{mod}2^5)$$

$$[x]_{补}=00000101(\mathrm{mod}2^8)$$

当 $x=-101$ 时，

$$[x]_{补}=1011\quad(\mathrm{mod}2^4)$$

$$[x]_{补}=11011\quad(\mathrm{mod}2^5)$$

$$[x]_{补}=11111011(\mathrm{mod}2^8)$$

由此可见，将一个数的二进制补码表示式最左位值向左**延伸**，所得到的仍是此数的二进制补码表示式，只是表示所取的模增大了。这一点从补码表示定义也很容易看出：当 x 为正时，结论是显然的，当 x 为负时，$[x]_{补}=2^n+x(\mathrm{mod}2^n)$；$[x]_{补}=2^m+x(\mathrm{mod}2^m)$，当 $m>n$，则有

$$[x]_{补}\mathrm{mod}2^m=2^m+x=2^{m-1}+2^{m-2}+\cdots+2^n+2^n+x$$

$$=2^{m-1}+2^{m-2}+\cdots+2^n+(2^n+x)$$