中国计算机软件专业技术资格和水平考试自学丛书

CASL 汇编语言程序设计

王世业　编著

清 华 大 学 出 版 社

(京)新登字 158 号

内 容 简 介

本书是为计算机软件专业人员学习 CASL 汇编语言,参加技术资格和水平考试而编写的一本自学教材。它是根据作者多年来在辅导班上的讲稿整理而成。有关汇编程序设计的基本概念及方法,是根据考试大纲的要求,吸收作者在大学里授课的内容编写的。

本书概念明确,叙理严谨,内容充实,并配有各种类型的例题。在分析问题过程中,注意到读者所关心的程序设计方法和技巧。第 6 章又为读者选取或改编了日本及中国近年来具有代表性的试题,可作为读者应考前测试自己能力的借鉴。考虑到自学的特点,在附录 D 中,给出了各试题的解答说明。

本书可作为参加水平考试考生的辅导教材,也可作为高等院校汇编语言程序设计课程的教学参考书。

由复旦大学计算机科学系研制的《CASL 执行环境》软件是一个 CASL 汇编语言的编辑、汇编、连接、运行及调试等多种功能的集成软件,同时由清华大学出版社出版、发行。

图书在版编目(CIP)数据

CASL 汇编语言程序设计/王世业编著 .—北京:清华大学出版社,1994
(中国计算机软件专业技术资格和水平考试自学丛书)
ISBN 7-302-01500-7

Ⅰ.C… Ⅱ.王… Ⅲ.微型计算机,CASL-汇编语言-程序设计 Ⅳ.TP312CA

中国版本图书馆 CIP 数据核字(94)第 03824 号

出版者:清华大学出版社 (北京清华大学校内,邮编 100084)
印刷者:顺义振华印刷厂
发行者:新华书店总店北京科技发行所
开 本:787×1092 1/16 印张:12.25 字数:285 千字
版 次:1994 年 8 月 第 1 版 1994 年 8 月 第 1 次印刷
书 号:ISBN 7-302-01500-7/TP·605
印 数:0001—5000
定 价:12.00 元

目　　录

前　言

本书是由中国计算机软件专业技术资格和水平考试中心组织编写，列为中国计算机软件专业技术资格和水平考试自学丛书之一。

CASL 是 1987 年日本计算机应用软件人员全国统考委员会采用的一种抽象的计算机汇编语言。根据中国计算机软件考试中心颁布的考试大纲规定，CASL 汇编语言是程序员级选考和高级程序员级必考的程序设计语言。

本书主要是为软件专业人员应考、学习 CASL 汇编语言程序设计的读者编写的。书中的内容是作者多年来在软件人员水平考试辅导班上的讲稿。有关程序设计的基本概念及方法，主要吸收我在大学里授课的内容。因此，本书是计算机软件专业技术人员的一本很好参考书。

本书的第 6 章，绝大部分内容是精选日本近年来 CASL 汇编语言试题，有一部分是日本或国内曾以 CAP 汇编语言写的试题，经过相应改编后列出的。选编这部分试题时，作者有意在正文中暂时不附上解答，目的是给那些应考心切的读者实际测试一下自己的实力，然后再听辅导课的讲解是颇为有益的。为适应广大自学读者的需要，这次正式出版时，第 6 章题解思想及答案在附录 D 中给出。

由于各类人员情况迥异，可按以下两类情况选择：

第一类，已有一般计算机汇编语言基础的，重心可放在第 4、5、6 章。即阅读例题及程序以熟悉 CASL 的特点及程序设计技巧。尽可能地做第 6 章的题选，以提高应试的能力。

第二类，是对汇编语言知之甚少，则应系统阅读本书。在熟悉 CASL 程序设计基本概念及方法之后，再逐步做第 6 章的题选。

为适应 CASL 汇编语言程序设计用户的上机实习之需，复旦大学计算机科学系研制的《CASL 执行环境》，是一个 CASL 汇编语言的编辑、汇编、连接、运行及调试等多种功能的集成软件，具有良好的用户界面，使用方便，是学习 CASL 的有力助手，本软件由清华大学出版社软件部同时发行，需要者可与清华大学出版社联系。

王世业

93 年 7 月于复旦大学

第 1 章　数制及运算系统

本章所论及的内容是汇编语言程序设计所需要的必备知识。如果读者业已熟悉它们，可以跳过而进入第 2 章的学习。

无论计算机从外部世界接受的是什么样的数据，例如十进制数，字符常数，甚至图示，它们在计算机的内部形式都是二进制代码。计算机采用二进制数来表达数据是因为具有两个稳定的物理状态，在电子线路上容易实现的缘故。计算机中的数据也可以是指令的编码，指令经过译码，产生一列操作，实现各种运算。计算机只有通过指令的执行，把内部形式的代码解释成特定的含义，才能将各种代码当作有意义的信息来使用。

计算机要进行数据的加工，必然涉及到数据在不同进位计数制下的表示方法和它们之间的转换，不同编码方式的数据在计算机上的形态，以及它与通常意义下数的表示之间的关系。要进行数据的运算，还涉及到采用的何种运算系统。

1.1　进位计数制

进位计数制用于计算机及人们日常事务中。同一含义的数，在不同进位制下有不同的表示方法。虽然都是由一串数码组成，但数制基不同，就有不同的解释。例如，数码串 101，在二进制下它的值是 5，但在十六进制下它的值是 257，因此有必要研究进位计数制及其性质。

假定整数 N，它在一般进位计数制下，每一个数位都对应一个权 R^i，这里的 R 称为数制的基数或底。因此，N 在这样数制系统中，它的一般形式为：

$$N = a_{p-1}a_{p-2}\cdots a_1a_0$$

a_i(i= 0, l, …, p- l)是下列多项式的系数：

$$N = a_{p-1}\times R^{p-1} + a_{p-2}\times R^{p-2} + \cdots + a_1\times R^1 + a_0\times R^0$$

$$= \sum_{i=0}^{p-1} a_iR^i$$

在 R 进位计数制系统中，每个 R 进位制数具有以下性质：

(1) 有 R 个数码。即 a_i 可为 0, 1, …, R- 2, R- 1 中任何一个数码，组成 R 进制数的数值部分。

(2) 逢 R 进 1。任意位置上的 R 个单位，构成高一位置上的 1 个单位。同样，任一位置上的 1 个单位，是它低一位置(右边) 上的 1 个单位的 R 倍。

(3) 数值唯一。当基数 R 确定时，则一个 R 进制表示的数，在十进制记数系统中的数值是唯一确定的。因为

$$N = \sum_{i=0}^{p-1} a_iR^i$$

其中, $0 < a_i < R-1 (i = 0, 1, \cdots p-1)$, 且 $a_{p-1} \neq 0$。

人们习惯使用十进制数, 而几乎所有计算机都是用二进制, 势必要了解数的二进制表示及其性质, 进而懂得二进制算术运算的规则和特点。由于二进制书写很不方便, 人们又引进八进制或十六进位计数制。以八进制或十六进制形式来表示数, 是因为它们的基数恰好是 2 的乘方, 特别适合于表达二进制数。这样在十进制数和二进制数之间的转换, 藉助于八进制或十六进制表示数是很自然的了。大多数计算机软件不是用八进制就是用十六进制描述机器中的二进制数。例如 PDP-11 是用八进制, 而在本书 COMET 计算机用的是十六进制。选用八进制还是十六进制表示完全是一种文件的习惯。选用什么对硬件的影响, 仅在于计算机操作面板上的灯和开关是排成三个一组还是四个一组。

在懂得一般进位计数制的性质之后, 对于特定进位制, 只需对基数 R 给出确定的值 (十进制记数系统意义下), 就类似地给出它的表示和它的性质。不妨看一下人们熟知的十进制数的表示。

十进制数是以 10 为基, 它具有:

(1) 有 10 个数码。即用 0, 1, 2, 3, 4, 5, 6, 7, 8, 9 这十个数码表示十进制数的数值部分。

(2) 逢 10 进 1。这是因为十进制系统中以 10 为基, 加法的满 10 进 1, 减法的借 1 当 10, 数值 825 最左边的数字 8 是“百倍”, 具有 $8 \times 100 = 800$ 的含义, 都是由这个性质决定的。

(3) 数值唯一。一个数 N 在以 10 为基, 可用以下多项式表达:

$$N = \sum_{i=0}^{p-1} a_i (10)^i$$

其中系数 $a_i (i = 0, 1, \cdots, p-1)$ 组成

$$N = a_{p-1} a_{p-2} \cdots a_1 a_0 \qquad (\text{假定 } a_{p-1} \neq 0)$$

就是数 N 的十进制表示。按 10 的幂和相应 a_i 相乘, 逐个相加, 其和就是数 N 唯一确定的值。例如

$$\begin{aligned} N &= 8 \times 10^3 + 1 \times 10^2 + 9 \times 10^1 + 2 \times 10^0 \\ &= 8192 \end{aligned}$$

这说明任何一个数 N 都可分解成一个以 10 为基的多项式, 在十进制记数系统中, 经过运算, 其值 8192 唯一确定。

二进制数是以 2 为基, 它具有:

(1) 有两个数码, 即 0, 1 表示二进制数的数值部分。

(2) 逢 2 进 1。基于这一特性, 人们运用逻辑代数原理, 进行计算机的逻辑设计。

(3) 任何一个数 N, 总可以用 2 为基的多项式表达:

$$N = \sum_{i=0}^{p-1} b_i (2)^i$$

其系数 $b_i (i = 0, 1, \cdots, p-1)$ 组成

$$N = b_{p-1} b_{p-2} \cdots b_1 b_0 \qquad (\text{假定 } b_{p-1} \neq 0)$$

就是 N 的二进制表示。同样原理, 任何一个二进制数

$$B = b_{k-1}b_{k-2}\cdots b_1 b_0$$

展开成以 2 为基的多项式

$$B = \sum_{i=0}^{k-1} b_i (2)^i$$

在十进制记数系统中,进行以 2 为幂的乘加运算,就可唯一得到它的数值。例如:

$$\begin{aligned} B &= 11001010 \\ &= 1\times 2^7 + 1\times 2^6 + 0\times 2^5 + 0\times 2^4 + 1\times 2^3 + 0\times 2^2 + 1\times 2^1 + 0\times 2^0 \\ &= 128 + 64 + 8 + 2 \\ &= 202 \end{aligned}$$

由性质(3),实际上也给出一个十进制数如何表达成二进制数的方法。

1.2 二进制数的运算

根据二进制数性质(2),可以类似于十进制系统建立二进制运算规则。只要记住“逢 2 进 1,借 1 当 2”,就可进行二进制数的算术运算。二进制数的加减手算法是和我们在小学学过的算术一样的,只不过得到的加法和减法表不同而已。表 1.1 是二进制数的加法和减法表。

表 1.1 二进制加法和减法表

C_{in}, b_{in}	X	Y	$X+Y+C_{in}$	C_{out}	$X-Y-b_{in}$	b_{out}
0	0	0	0	0	0	0
0	0	1	1	0	1	1
0	1	0	1	0	1	0
0	1	1	0	1	0	0
1	0	0	1	0	1	1
1	0	1	0	1	0	1
1	1	0	0	1	0	0
1	1	1	1	1	1	1

按照此表,两个二进制数 X 和 Y 的相加,实际上是将它们的最小有效位和初始进位(C_{in})零相加,产生和($X+Y+C_{in}$)与进位(C_{out})。接着从右到左运算每一位,计算时要把每一列的进位加到下一列的和中去。十进制加法和相应的二进制加法的例子表示如下,进位用一个位串 C 表示。

进位	C		101111000
被加数	X	190	10111110
加数	Y	+ 141	+ 10001101
相加	X+Y	331	101001011

进位	C		001011000
被加数	X	173	10101101
加数	Y	+ 44	+ 00101100
相加	X+ Y	217	11011001

减法是类似的,但要用借位(b_{in}和b_{out})代替进位:

借位	B		001111100
被减数	X	229	11100101
减数	Y	- 46	- 00101110
相减	X- Y	183	10110111

在计算机中,减法通常的用途是比较两个数。例如,若(X- Y)的运算产生一个超出最高有效位的借位,则 X 小于 Y;否则 X 大于或等于 Y。

加法和减法表可扩展到八进制和十六进制数,也可扩展到任何一种基数。然而,由于种种理由,几乎没有计算机科学工作者费心去记忆这些表。减法表完全不必要,因为减法总可以用补码加法来实现,这将在下一节中说明。甚至加法也很少用手算,特别是在高级语言程序设计中。

COMET 计算机的终端上输出格式是采用十六进制数形式来表示机内二进制代码,这在调试程序或需要观察程序在机内的代码,需要做一点十六进制加或减法的运算。以下仅给出一个十六进制数相加的示例。

两个十六进制数相加的心算步骤如下:

进位	C	110 0	1	1	0	0
被加数	X	19B 9_{16}	1	9	11	9
加数	Y	+ C7E 6_{16}	12	7	14	6
相加	X+ Y	E19 F_{16}	14	17	25	15
			14	16+ 1	16+ 9	15
			E	1	9	F

二进制数的乘法和除法,在许多计算机中是用机器指令来实现的。像有些小型计算机为了节省硬件而省略了乘除指令,COMET 计算机没有设置这类指令,因此必须在软件中用加法或减法指令和移位来实现。

在小学我们就学过乘法,它是由乘数的每一位数分别和被乘数相乘得到,如下所示:

```
    11                1011        被乘数
×   13        ×       1101        乘数
    33                1011
   11                 0000        移位的被乘数
   143               1011
                    1011
                   10001111        乘积
```

在二进制乘法中得到移位的被乘数是很简单的,因为乘数的值只可能是 0 和 1。

不过,在计算机中不是列出所有的移位的被乘数再相加,而是采用将每一次得到的移位的被乘数和部分积相加的办法做乘法。用这种方法再次重复上述例子示于表 1.2 中。

表 1.2　用部分积的方法做的乘法

1011	被乘数
× 1101	乘数
00000000	部分积
1011	移位的被乘数
00001011	部分积
0000	移位的被乘数
00001011	部分积
1011	移位的被乘数
00110111	部分积
1011	移位的被乘数
10001111	乘积

最简单的二进制除法算法也是建立在小学学过的方法的基础上,如表 1.3 所示。在十进制和二进制中,我们都先将余数和除数的倍数进行比较,以便确定从余数中可以减去多少倍移位的除数。在十进制的例子中,我们第一步选 11 作为 21 的 11 的最小倍数,然后选 99作为小于 107 的 11 的最小倍数。在二进制中,选定除数倍数要简单一些,因为只有两种选择,不是 0 就是除数本身。但是为了选择适当的移位除数,比较操作仍然是需要的。

表 1.3　长除的例子

19	商		10011	商
11 217	被除数	1011	11011001	被除数
11			1011	移位的除数
107			0101	减后的被除数
99			0000	移位的除数
8	余数		1010	减后的被除数
			0000	移位的除数
			10100	减后的被除数
			1011	移位的除数
			10011	减后的被除数
			1011	移位的除数
			1000	余数

在计算机中,不带符号的除法是和乘法互补的。一个典型的除法算法接收一个双字节的被除数和一个单字节的除数,产生单字节的商和余数。如果除数为 0 或商大于一个字节,则该除法将产生溢出。第二种情况仅发生在除数小于或等于被除数的高位字节时。

以上所讨论的二进制数的加、减、乘、除的规则,是针对无符号的二进制代码而言的。有符号数的加减法,虽然计算机视为无符号的二进制代码进行加减,但要求程序给出适当的解释。有符号数的乘除与无符号数的乘除算法是有差异的,以后再去讨论它们。

1.3 数制之间的转换

程序员需要把一个机内数输出,首先是把这些数表示为十进制形式,那将方便得多。有时需要察看机内代码,用十六进制形式输出,不仅读起来方便,而且还与二进制之间保持有良好的对应关系(十进制不存在这种对应),这对了解指令编码或机内特征位是很必要的。同样,把十进制数或十六进制数转换为二进制形式也是需要的。一般来讲,两种基数之间的转换不能用简单的置换法来实现,需要通过一定的运算。

首先给出十进制、二进制、八进制、十六进制的数码之间的对应关系,然后再分别讨论它们数的表达形式之间的转换方法。

表 1.4 二进制、十进制、八进制、十六进制对照表

二进制	十进制	八进制	十六进制
0	0	0	0
1	1	1	1
10	2	2	2
11	3	3	3
100	4	4	4
101	5	5	5
110	6	6	6
111	7	7	7
1000	8	10	8
1001	9	11	9
1010	10	12	A
1011	11	13	B
1100	12	14	C
1101	13	15	D
1110	14	16	E
1111	15	17	F

1. 二进制→十六进制的转换(记为 2→16)

二进制的基是 2,十六进制的基是 16,恰好是 2 的 4 次乘方,即 4 位二进制数正好对应 1 位十六进制数。这样我们从最低位(右边)开始,把二进制数分成 4 位一组,每组都用一个等效的十六进制数字代替即可。

例如:二进制数　0110　1010　1101　1111

十六进制数　6　A　D　F

上述过程中应注意到,在二进制数左边(高位)可根据需要随意地加 0(对于无符号数而言),以便使总的位数正好是 4 的倍数。

2. 十六进制→二进制的转换(记为 16→2)

和 2→16 过程正好相反,16→2 是用等效于十六进制数字的 4 位一组二进制数来代替一个十六进制数字,连起来一串二进位数字组成一个二进制数。

例如: 十六进制数　　1　　2　　A　　6

二进制数　　0001　　0010　　1010　　0110

应该注意,不管表示一个十六进制数字实际上需要多少二进位,重要的是每个十六进制数字,必须被 4 位二进位所代替,如数字 2,是用 4 位 0010 替代,而不是仅用 2 位的 10 来替代,否则得不到正确的二进制结果。

3. 十六进制→十进制的转换(记为 16→10)

十六进制数转换成十进制数,从本质上说是将它表达成以 10 为基数的多项式

$$N_{(16)} = \sum a_i \times (10)^i \qquad (i = 0, 1, \cdots, p-1)$$

则 $a_{p-1}a_{p-2}\cdots a_1a_0$(设 $a_{p-1} \neq 0$)即为 $N_{(16)}$ 的十进制数,但是求出这个多项式的各项系数 a_i($i = 0, 1, \cdots, p-1$)不是很方便。在十进制记数系统运算以 16 为基的 $N_{(16)}$ 的多项式的值就是 $N_{(16)}$ 的十进制表示的数。这种方法简便,只要将数的各位按 16 的幂次展开,然后按十进制规则,逐次进行乘加运算即得。

例如: $1BE8_{(16)} = 1 \times 16^3 + 11 \times 16^2 + 14 \times 16^1 + 8 \times 16^0$

$= 4096 + 2816 + 224 + 8$

$= 7144_{(10)}$

16→10 的另一种方法,可查十六进制和十进制转换表(见表 1.5),形如 $h_4h_3h_2h_1$ 的四位十六进制数,分别从表 1.5 中查出各位对应的十进制值,然后做个十进制加法即可得到。例如:十六制数 7FDE,查表如下:

$h_4 = 7$	对应	28672
$h_3 = F$	对应	3840
$h_2 = D$	对应	208
$h_1 = E$	对应	14

其和为: 32734

这种方法对较大的十六进制数,只需查表做简单的加法即可得到对应的十进制数。

表 1.5　十六进制和十进制转换表

4		3		2		1	
十六	= +	十六	= +	十六	= +	十六	= +
0	0	0	0	0	0	0	0
1	4, 096	1	256	1	16	1	1
2	8, 192	2	512	2	32	2	2
3	12, 288	3	768	3	48	3	3
4	16, 384	4	1, 024	4	64	4	4
5	20, 480	5	1, 280	5	80	5	5
6	24, 576	6	1, 536	6	96	6	6
7	28, 672	7	1, 792	7	112	7	7
8	32, 768	8	2, 048	8	128	8	8
9	36, 864	9	2, 304	9	144	9	9

续表

4		3		2		1	
十六	= +	十六	= +	十六	= +	十六	= +
A	40, 960	A	2, 560	A	160	A	10
B	45, 056	B	2, 816	B	176	B	11
C	49, 152	C	3, 072	C	192	C	12
D	53, 248	D	3, 328	D	208	D	13
E	57, 344	E	3, 584	E	224	E	14
F	61, 440	F	3, 840	F	240	F	15

4．十进制→十六进制的转换(记为 10→16)

在进位计数值一节中，根据性质(3)，取基数 R = 16，则任何一个十进制数 N，可唯一表达成

$$N = h_m \times 16^m + h_{m-1} \times 16^{m-1} + \cdots + h_1 \times 16 + h_0$$

试想，假定用 16 来除这个公式中的各数，发生了什么情况。由带余除法定理，得到：

商　$Q = h_m \cdot 16^{m-1} + h_{m-1} \cdot 16^{m-2} + \cdots h_2 \cdot 16 + h_1$

余数　$R_0 = h_0 \quad 0 \leq h_0 < 16$

于是 h_0 就可以看作为用 16 对已知的十进制数 N 相除所得的余数。另外商 Q 和原先公式有相同的形式。因此不断地除以 16，就会得到 N 的从右到左相继的每一位数字，直到最后一次商为 0 时，余数 h_m 就是十六进制数的最高位数码。由此我们导出 10→16 的规则是：除 16 取余，直到商为零。

例如，将十进制数 483 转换为十六进制数形式，有以下过程：

	商	余数	
16	483	3	h_0
16	30	14	h_1
16	1	1	h_2
	0		

所以　$483_{(10)} = 1E3_{(16)}$

10→16 第二种方法也可以应用表 1.5，在十六进制和十进制转换表中找到小于该十进制数的最大表值，找到对应的十六进制数字所在的列，求差，再去查表，直到低位。

5．二进制→十进制的转换(记为 2→10)

像 16→10 那样，二进制数转换成十进制数的最直接方法，是把二进制数按基数 2 的幂次展开，按十进制记数系统的规则，逐项求和，其结果就是该数的十进制表示。例如：

$$\begin{aligned} 10101_{(2)} &= 1\times 2^4 + 0\times 2^3 + 1\times 2^2 + 0\times 2^1 + 1\times 2^0 \\ &= 16 + 4 + 1 \\ &= 21_{(10)} \end{aligned}$$

6．十进制→二进制的转换(记为 10→2)

按 10→16 的原理，十进制数转换为二进制数方法是进行 2 的多次除法，从低位到高位逐个取余数，直到商为零止，最后的余数就是二进制数的最高位，等等。

2→10 和 10→2 的第二种方法是通过 16→10 与 10→16，即先把二进制数转换成十六进制数，再由十六进制数转换成十进制数，它的逆运算也如此。如果任意一个 4 位二进制码和十六进制数字对应熟悉的话，第二种方法不失为一个简便的方法。

7．实用的 10→2

程序中需要的数据，程序员可利用汇编语言的伪指令语句“ DC n ”，它由汇编程序自动将 n 所代表的十进制数或十六进制数转换成机内二进制数。

如果程序中需要动态地从输入设备上(例如键盘) 输入十进制数，又是怎样的呢?

我们注意到上述的数制间的转换过程，所有数据是作为数值本身的转换。实际上程序员从键盘上输入的数据，例如一个十进制数是作为字符形式的数字串而存在的。要把一个十进制数字串 $a_{p-1}a_{p-2}\cdots a_1a_0$，变成为真正数值来用，要经过如下两步:

第一步: 将输入的数字字符换成该数字的面值。因为一个字符的 ASCII 编码由 7 位二进制位组成。十进制数字字符 ASCII 码的低 4 位即为其面值，截下 4 位即为它对应的二进制值。

第二步: 从高位到低位，利用下列公式进行“ 乘加 ”操作，即得到 $a_{p-1}a_{p-2}\cdots a_1a_0$ 所代表的二进制值。

$$N = (\cdots((a_{p-1})\times 10 + a_{p-2})\times 10 + \cdots + a_1)\times 10 + a_0$$

其中每次相乘的基数 10 应是二进制数 1010 参加运算。

这个公式看来似乎并没有太大的价值，但它却是给出一个实用的 10→2 算法。

1.4 数在计算机上的表示

在计算机内的一切数据都是二进制代码，单纯的数据本身并无实际意义，这些代码只有给出特定解释，才能作为有用的数据使用。现在要问，各式各样的数据以怎样的方式放进计算机中? 到目前为止，我们只涉及了正整数，确切地说是无符号整数。无符号整数的二进制形式和机内二进制形式是完全一致的。在日常事务中，我们常常用到带符号的数值系统，这就是讨论负数在计算机中如何表达。负数的表达有许多种方法，但是多数计算机采用的是补码系统，因此本节重点是讨论数的补码表示及补码运算系统的一般特性。

1.4.1 有符号整数的表示

有符号数在计算机中的表示，关键是用什么方式将它的符号放进去。根据二进制数码具有 0、1 两个状态，我们可以约定正号“ + ”和负号“ - ”分别用数码 0、1 两个状态来代表，这叫符号的数码化。既然符号要占去一个二进位，那末一个 n+ 1 位字长的数值部分最多只能是 n 位了。为了区别原来的数与它在计算机中表示的数，把经过“ 符号数码化 ”的数称为机器码，原来的数称为机器码的真值。

1．符号-幅值表示法

用符号-幅值表示的机器码称为原码，其定义是：假定整数 X 是 n 位二进制数码的数，即

$$X = \pm X_1X_2\cdots X_n$$

$$[X]_{原码} = \begin{cases} X & 当\ 2^n > X \geq 0 \\ 2^n - X & 当\ 0 \geq X > -2^n \end{cases}$$

按此定义，我们看十进制数+ 43 和- 127 的原码是怎样的？不妨假定把它们用一个 8 位二进制位(一个字节) 来表达：

$$A = +43_{(10)} = +0101011_{(2)}$$

$$[A]_{原码} = 00101011_{(2)}$$

$$B = -127_{(10)} = -1111111_{(2)}$$

$$[B]_{原码} = 11111111_{(2)}$$

原码用二进制数表示很容易，只需在数值前面以一个附加符号位置 0 或 1，当数值为正时置 0，数值为负时置 1，既方便又容易读出它的真值，若要这个数变号，只需改变它的符号位。

原码系统具有相同数量的正数和负数。一个 n 位的有符号数一定包括在- (2^n - 1) 到十(2^n - 1) 的范围内，包括 0 的两种可能的表示法。

原码系统有两个缺点：

一是 0 的不唯一性，由定义：

$$[+0]_{原码} = 00\cdots0 \quad (n+1\ 个\ 0)$$

$$[-0]_{原码} = 10\cdots0 \quad (n\ 个\ 0)$$

二是符号位仅作正负标志，不能参加运算。若要做原码的加减，要判别符号位等额外操作。这带来逻辑线路复杂性，是目前计算机不采用原码系统的主要原因。

2．补码表示法

补码定义：假定具有 n 位二进制数码的数：$X = \pm X_1X_2\cdots X_n$

$$[X]_{补码} = \begin{cases} X & 当\ 2^n > X \geq 0 \\ 2^{n+1} + X & 当\ 0 \geq X \geq -2^n \end{cases}$$

记$[X]_{补码} = y_0y_1\cdots y_n$ 是 n+ 1 位机器码。不妨设 n= 7，看十进制数+ 125 和- 109 的 8 位机器码表示的补码是什么？由定义，

令 $A = +125 = +1111101,\ A > 0$

$[A]_{补码} = 01111101$

令 $B = -109 = -1101101 \quad B < 0$

$[B]_{补码} = 2^8 + (-1101101) = 10010011$

用补码表示的机器码，正数和它的真值是一致的，但负数的补码和它的真值的数值部分差异甚大，似乎不直观，不方便。但是，补码系统却有许多很好的性质，目前大多数计算机是采用补码系统。

真值的正负属性由机器码的符号位 y_0 反映，$y_0 = 0$，真值为正，$y_0 = 1$，真值为负。零的补码表示是唯一的，并且确认为正的。因为

$$[+0]_{补码} = 00\cdots00 \qquad n+1\ 个\ 0$$