微型計算機原理及應用

葛纫秋 马世伟 主编

兰州大学出版社

微型计算机原理及应用

葛纫秋　马世伟　主编

兰州大学出版社

（甘）新登字第 08 号

内 容 简 介

本书以使用广泛的 8086/8088 微处理器为背景，较为系统地介绍了 16 位微型计算机的体系结构、指令系统和微型计算机系统的组成原理，并以此为基础着重讨论了汇编语言程序设计方法和输入输出技术，以及典型接口芯片的原理和应用。

在编写指导思想上，编者特别注重突出对微型计算机基本工作原理和基础知识的说明。全书共分七章，第一章介绍计算机中数的表示方法和运算规则；第二至五章全面介绍 Intel8086/8088 系统的组成原理，结构，存贮器组织，指令系统，宏汇编语言程序设计；第六、七章介绍了输入/输出技术，中断，DMA 以及接口电路和典型芯片。本书可作为高等院校本科教材，也可供从事微型计算机应用工作的工程技术人员学习和参考。

微型计算机原理及应用

葛纫秋　马世伟　主编

兰州大学出版社出版

（兰州大学校内）

甘肃静宁印刷厂印刷　　甘肃省新华书店发行

开本：787×1092 毫米 1/16　印张：18.625

1994 年 8 月第 1 版　1994 年 8 月第 1 次印刷

字数：442 千字　　印数：1—4000

ISBN7—311—00786—Q/T・13　　定价：18.50 元

前　言

本书是为无线电电子学、现代通信、自动控制及计算机应用专业本科编写的“微型计算机原理与应用”课程教材。其中第一、二、四、五章还可作为“8086 汇编语言程序设计”课程教材。

由于计算机技术发展十分迅速、机型和芯片更新换代周期短，因此在编写教材时应注意其通用性，力求使教材具有较为广泛的适应性。我们注意突出对微型计算机工作原理及基础知识的说明。以 8086/8088 微处理机及其外围芯片作为典型实例，介绍有关微型计算机的基础知识和基本概念，为学生以后更新知识、学习和掌握新技术提供较为充实的基础。

全书共分为七章。

第一章是基础部分。介绍计算机中数的表示方法与运算规则，运算电路和产生特征标志的电路。在此基础上，介绍一个简单的模型机的总体结构，指令编码格式，电路实现及工作过程。这一章具体地说明了微型计算机各个部件的电路结构与工作原理，是深入理解以后各章内容的基础。

第二章介绍 8086/8088CPU 的结构、引脚与时序，以及其存贮器组织和 I/O 组织的基本特点。

第三章介绍半导体存贮器及 8086/8088 系统中存贮器的组成。

前三章说明了微型计算机的工作原理及基本系统(不包括输入输出接口)的组成。

第四、五章介绍 8086/8088 的指令系统及汇编语言程序设计方法。为了使学生能尽快上机实习，第五章比较系统地介绍了宏汇编语言的基本语法及顺序程序、分支程序和循环程序的设计方法。所举例题均经过上机调试，可作为学生上机练习的实例。

第六、七章为输入输出 技术。包括输入输出接口电路的结构，典型接口芯片及应用举例。这一部分内容安排与例题选择也充分注意与实验课配合。

本教材参改学时为 72 学时，通过本课程的学习，应使学生掌握有关微型计算机的基础知识与基本概念，包括：微型计算机的工作原理，基本系统的构成，汇编语言程序设计方法和输入输出的基本技术。使学生能运用所学知识，完成一般的软、硬件设计，并具有进一步学习，更新知识的能力。

兰州大学教务处，兰州大学电子与信息科学系及有关领导、教师对本书的编写与出版给予了大力支持与帮助；席先觉教授亲自审阅书稿，并提出宝贵意见；李柏年等同志也为本书的出版作了大量的工作。在此一并表示衷心的感谢。

由于编者水平有限，本书难免存在错误和不当之处，敬请读者批评指正。

编　者

1994 年 7 月于兰州大学

目　录

第一章　概述

电子数字计算机的发明和应用标志着人类文明进入了一个新的历史阶段。

从1946年世界上第一台电子计算机问世至今，不过四十多年的历史，在此期间，计算机的发展经历了四代：

从1946年到1957年是电子管计算机时代。其主要特点是使用电子管作为逻辑元件，存贮器用延迟线或磁鼓，软件主要是机器语言，符号语言开始使用。1946年出现的第一台计算机ENIAC，使用了1880个电子管，占地150平方米，重30吨，耗电150千瓦，价值40万美元，加法运算速度500次/秒，各种性能均无法与今天的微型计算机相比。但是，它却奠定了电子计算机技术的基础。

从1958年至1964年是晶体管计算机时代，这一代计算机的主要特点是用晶体管取代了电子管，使用了磁芯存贮器。软件方面出现了高级程序设计语言。如ALGOL，FORTRAN，还提出了操作系统。与第一代计算机相比，第二代计算机在性能与可靠性方面均提高了一个等级。

从1965年至七十年代初，数字集成电路的出现使计算机再次出现重大进步，产生了以中、小规模集成电路为基础，配置更完善软件的第三代计算机。在存贮容量，运算速度和可靠性等方面比第二代计算机又提高了一个数量级。

七十年代以来，随着大规模，超大规模集成电路的诞生，电子计算机技术更是突飞猛进地发展，大规模集成电路取代了中小规模集成电路，代替了磁芯存贮器，从而有可能将计算机的主机集成在一块硅片上，成为微处理器(CPU)或单片机。目前微型计算机的发展趋势一方面是集成电路规模越来越大，性能越来越高，从70年代初的4位机到70年代中期的8位机，以及80年代的16位机和32位机，时钟频率也从4004的750KH$_z$到80486DX4的100MH$_z$。另一方面是将组成一个微型计算机或应用系统所需的存贮器，I/O接口，定时/计数器及A/D转换器全部集成在一个芯片上的单片机不断发展，功能日趋完善，字长也从8位发展到16位。

微型计算机的出现开拓了计算机普及的新纪元。由于它的体积小，性能好，价格低而使其有可能渗透和占领各个技术领域，甚至进入人们的日常生活。目前微机技术仍在飞速发展，在提高硬件性能的同时，各种功能更强的软件系统不断研制出来，而且微型计算机普及应用的势头正在进一步扩大。可以予料它必将渗透到所有的行业和部门，其发展前景不可限量。

1.1　数的表示法及二进制数的机器运算

我们首先研究数的一般表示方法及计算机中所采用的表示方法。计算机采用2进制计数法，并且将符号数表示为补码形式。采用这样的表示方法可以很方便地用数字电路来实现算术运算。

1.1.1 数的位置表示法及各种进位制的数

我们习惯于用一组约定的符号来表示数。这些符号可以是数符(0～9)，也可以是字符(A、B、C、D、E、F 等)或其它符号(Ⅰ、Ⅱ、Ⅲ、Ⅳ等)，表示一个数的一组符号中的每一个符号所代表的量不仅取决于这个符号本身，还取决于它所在的位置。例如，十进制数 33：右边的 3 代表 3，左边的 3 却代表 30，这就是位置表示法。

在位置表示法中，对每一数位赋以一定的位置，称为权。每个数位上的数字所表示的量是这个数字和该位的权的乘积。如我们学过的 8421 码，2421 码等等。

如果相邻两位中高位的权与低位的权之比总是常数，则称此常数为基数。

如果数 $A_{n-1}A_{n-2}\cdots A_1A_0A_{-1}\cdots A_{-m}$ 的基数为 X，则它所代表的数值 N 可表示为：

$$\begin{aligned}N &= A_{n-1}X^{n-1}+A_{n-2}X^{n-2}+\cdots+A_0X^0\\&\quad+A_{-1}X^{-1}+A_{-2}X^{-2}+\cdots+A_{-m}X^{-m}\\&=\sum_{i=-m}^{n-1}A_iX^i\end{aligned}$$

式中从 A_0X^0 起(包含该项在内)，左边的部分为该数整数部分，而右边的部分则为小数部分。整数部分最低位为第 0 位，每向左一位，位数增加 1，每向右一位，位数减 1。

基数 X 可以取不同的值，对应可得到不同进位制的数的表达式。

若 X=10，则可得到十进制的表达式为：

$$(N)_{10}=\sum_{i=-m}^{n-1}A_i10^i$$

其中系数 A_i 可为 0～9 中的任一个。每个数位的权则是 10 的对应次幂。幂的次数即该数位的位数。即整数最低位的位权为 $10^0=1$，次低位为 $10^1=1$，若最低位为 9，再加 1 时该位将变为 0，同时使次低位增加 1，向上各位间的关系与此相似。因此十进制数运算时遵守“逢十进一”、“借一当十”的规则。

例如 $(1392.67)_{10}=1\times10^3+3\times10^2+9\times1^1+2\times10^0+6\times10^{-1}+7\times10^{-2}$

当 X=2 时，得到二进制数的表达式为：

$$(N)_2=\sum_{i=-m}^{n-1}A_i2^i$$

此时系数 A_i 只能取 0 或 1，每个数位上的权是 2 的对应次幂。二进制数运算时遵守“逢二进一”或“借一当二”的规则。

例如 $(1011.011)_2=1\times2^3+0\times2^2+1\times2^1+1\times2^0$

$$+0\times2^{-1}+1\times2^{-2}+1\times2^{-3}$$

二进制数的每一位只有 0 和 1 两种可能的取值，特别适合于用两态器件来存贮运算，是电子计算机内部采用的计数制。

在计算机技术中，经常用到的还有八进制数和十六进制数，它们通常在书写程序时被采用。

表 1.1 列出四种进位制中数的表示法。其中 B(Binary)，Q(Octal，用 Q 代替 O 以与数字 0 区别)，H(Hexadecimal)作为后缀，分别表示其前面的数为二进制数，八进制数或十六进制数。十进制数的后缀 D(Decimal)通常省略。

1.1.2 计算机中对符号数的表示方法

一、原码

机器数包括正数和负数。在计算机中，通常只能识别高低两个不同的电平，对于正逻辑而言，高电平代表二进制数“1”，低电平代表“0”。无论是数位还是符号，都只能用“0”或“1”来表示。

在原码表示法中，用最高位表示数的符号(“0”表示正号，“1”表示负号)，其余各位表示数值本身。例如，设X=+1011100，Y=−1011100，则

$$[X]_{原} = 01011100 \qquad [Y]_{原} = 11011100$$

表1.1 十进制、二进制、八进制、十六进制数码对照表

十进制	二进制	八进制	十六进制
0	0000B	00Q	0H
1	0001B	01Q	1H
2	0010B	02Q	2H
3	0011B	03Q	3H
4	0100B	04Q	4H
5	0101B	05Q	5H
6	0110B	06Q	6H
7	0111B	07Q	7H
8	1000B	10Q	8H
9	1001B	11Q	9H
10	1010B	12Q	AH
11	1011B	13Q	BH
12	1100B	14Q	CH
13	1101B	15Q	DH
14	1110B	16Q	EH
15	1111B	17Q	FH
16	10000B	20Q	10H

对一个具体的计算机，它的一次操作所能处理的二进制数的长度是由其内部结构决定的，通常采用的长度有八位、十六位、三十二位，对应的计算机则被称为“8位机”、“16位机”和“32位机”，因此机器数必须按规定的位数读或写，不能省略无效的0。例如，对

于8位机，若X＝＋10111，Y＝－10111 则其原码表示为：

$$[X]_{原} = 00010111 \qquad [Y]_{原} = 10010111$$

通常定义八位代码组成一个字节，而十六位代码则为一个字。若实际码长超过一个字节，则可用多个字节来表示它。

原码法表示数时，最大的优点是直观，而最大的缺点是运算处理过程非常繁琐。例如，为了实现两数相减，需要首先判断它们是同号数还是异号数。如果是异号数，可以两数的绝对值相加符号不变。如果是同号数，又需要比较其绝对值的大小关系，用大的绝对值减去小的绝对值，最后，还要根据两数的符号与绝对值，决定计算结果的正负。为了简化运算，人们在实践中总结出另一种数的表示法——补码法。采用补码表示时，符号可以与数一同参加运算，从而大大简化处理过程，也使计算机的结构大为简化。

二、反码

当 $X \geqslant 0$ 时，$[X]_{反}=[X]_{原}$

当 $X<0$ 时，$[X]_{反}=\overline{[|X|]_{原}}$

说明：符号$\overline{X}$表示对二进制代码的各位取反的结果。

三、补码

定义 $[X]_{补}=2^n+X$

其中X为带符号的n位二进制数，但其最高位为0(实际上只有(n－1)个数值位)，符号位用‘＋’、‘－’号表示。$[X]_{补}$是用n位二进制代码表示的X的补码，其最高位是符号位，‘0’为正号，‘1’为负号。

以下讨论求补码的方法。

当 $X \geqslant 0$ 时，2^n 是第n位的“1”，n位二进制数的最低位为第0位，最高位为第(n－1)位，而 2^n 在第n位，是进位位，它对n位二进制数本身没有影响。因此，在只保留n位二进制代码的情况下，有：

$$\begin{aligned}[X]_{补} &= 2^n + X \\ &= [X]_{原} \qquad (X \geqslant 0)\end{aligned}$$

如果 $X<0$，那末：

$$\begin{aligned}[X]_{补} &= 2^n + X \\ &= 2^n - |X|\end{aligned}$$

例如，若n＝4，x＝－0101，则$|X|$＝0101。此时

$$\begin{aligned}[X]_{补} &= [-0101]_{补} \\ &= 2^4 - 0101 \\ &= 10000 - 0101 \\ &= 1011\end{aligned}$$

现在讨论如何从X求得$[X]_{补}$。我们知道如果X为n位二进制代码，那末

$X+\overline{X}=2^n-1$ （n位全1）

$X+\overline{X}+1=2^n$ （n位全0和最高位的进位）

$2^n-X=\overline{X}+1$

现在X是n位二进制负数，因此，|X|是n位二进制代码，所以：

$$|X|+\overline{|X|}+1=2^n$$

即

$$\overline{|X|}+1=2^n-|X|$$

$$=[X]_{补}$$

$$[X]_{补}=\overline{|X|}+1$$

由$[X]_{补}=2^n-|X|$，又可得到

$$|X|=2^n-[X]_{补}$$

$$=\overline{[X]_{补}}+1$$

结论1 将一个负数(其最高数值位应为0)的绝对值的各位取反后再加1，其结果即为该负数的补码。

结论2 将一个负数的补码的各位取反后再加1，其结果即为该负数的绝对值。

例如，若X=−01011011，则

$$|X|=01011011$$

$$[X]_{补}=\overline{01011011}+1$$

$$=10100100+1$$

$$=10100101$$

若$[X]_{补}=10100101$，则由于它的最高位为1，所以X为负数，且

$$|X|=\overline{[X]_{补}}+1$$

$$=\overline{10100101}+1$$

$$=01011010+1$$

$$=01011011$$

若$[X]_{补}=00100101$，则由于它的最高位为0，所以X为正数，且

$$|X|=[X]_{原}$$

$$=[X]_{补}$$

$$=00100101$$

四、由$[X]_{补}$求$[-X]_{补}$

若$X\geqslant 0$，则$[X]_{补}=[X]_{原}$，且$-X\leqslant 0$，$|-X|=X=[X]_{原}$。

根据前面的结论，因为$-X<0$，所以

$$[-X]_{补}=\overline{|-X|}+1$$

$$=\overline{[X]_{原}}+1$$

$$=\overline{[X]_{补}}+1$$

若$X<0$，则$[X]_{补}=\overline{|X|}+1$，$|X|=\overline{[X]_{补}}+1$，且$-X>0$，$|X|=-X=[-X]_{原}$，因为$-X>0$，所以

$$[-X]_{补}=[-X]_{原}$$

$$=|X|$$

$$=\overline{[X]_{补}}+1$$

结论：不论X是否大于0，均有

$$[-X]_{补}=\overline{[X]}_{补}+1$$

将$[X]_{补}$取反后加1，其结果即为$[-X]_{补}$。

微型计算机中，求补指令对应的操作是将操作数的各位取反后再加1，我们通常将这种操作称为求补。

对一个正数的补码，执行求补指令后，得到的结果是与它的绝对值相同的负数的补码。

对一个负数的补码，执行求补指令后，得到的结果是它的绝对值，也可以说是与它的绝对值相同的正数的补码。

1.1.3 二进制数的运算与运算电路

一、带符号数的运算

如果X和Y是两个带符号二进制数，那末(Y±X)仍为带符号二进制数。根据补码的定义，有：

$$[Y+X]_{补}=2^n+(Y+X)$$
$$[Y-X]_{补}=2^n+(Y-X)$$

不考虑最高位的进位时，可以得到：

$$\begin{aligned}[Y+X]_{补}&=2^n+(Y+X)\\&=(2^n+Y)+(2^n+X)\\&=[Y]_{补}+[X]_{补}\end{aligned}$$

$$\begin{aligned}[Y-X]_{补}&=2^n+(Y-X)\\&=(2^n+Y)+[2^n+(-X)]\\&=[Y]_{补}+[-X]_{补}\\&=[Y]_{补}+\overline{[X]}_{补}+1\end{aligned}$$

二、无符号数的运算

1. 两个n位无符号二进制数相加的结果即为这两个二进制数之和，如果和大于或等于2^n，则产生最高位的进位。

2. 为了计算两个n位无符号二进制数之差，我们分析以下运算关系：

$$\begin{aligned}2^n+Y-X&=Y+2^n-X\\&=Y+\overline{X}+1\end{aligned}$$

如果$Y\geqslant X$，则$Y-X\geqslant 0$，$2^n+Y-X\geqslant 2^n$，即$Y+\overline{X}+1$运算结果的低n位(0—(n—1)位)为$Y-X$，并且产生最高位(第n位)的进位。

例如，当n=8时

$$11101011-01100011=10001000$$

而
$$\begin{aligned}11101011+\overline{01100011}+1&=11101011+10011101\\&=\underset{\text{进位}}{1}\quad 10001000\end{aligned}$$

如果$Y<X$，则$Y-X<0$，$2^n+Y-X=(2^n+Y)-X<2^n$，即$Y+\overline{X}+1$运算的结果是从高位借1后$Y-X$的结果。

例如：

$$\begin{array}{r} 01100011 \\ -\ 11101011 \\ \hline 01111000 \\ \text{借 1} \end{array}$$

而

$$01100011+\overline{11101011}+1=01100011+00010101$$
$$=01111000$$

结论：$Y-X=Y+\overline{X}+1$。

若 $Y+\overline{X}+1$ 有进位，则运算结果为 $Y-X$ 的结果，且没有向最高位的借位；

若 $Y+\overline{X}+1$ 无进位，则运算结果为 $Y-X$ 的结果，但有向最高位的借位。

三、运算电路

综合前面的分析，得出结论如下：

如果 n 位二进制代码 X，Y 同为无符号二进制数或同为带符号二进制数的补码，则可用 $Y+X$ 的运算求得 Y 与 X 之和；用 $Y+\overline{X}+1$ 的运算求得 Y 与 X 之差。

如果 X，Y 均为无符号二进制数，则上述运算结果仍为无符号二进制数。

如果 X，Y 均为带符号二进制数的补码，则运算结果仍为带符号二进制数的补码。

这就是说，借助于取反操作和加 1 运算，就可以用加法来实现两数相减的运算。图 1.1 所示的电路就是按这个原则设计的，它既能完成加法运算，又能完成减法运算。图中 L_A、L_B、E_V 和 N 均为控制信号，以后我们会看到，这些信号由计算机的控制电路根据指令规定的操作按一定时间顺序产生。8 位寄存器 $A_7\sim A_0$ 和 $B_7\sim B_0$ 中存放两个参加运算的数 A 和 B。

图 1.1 所示即为可实现加法和减法运算的电路的结构。其中 L_A，L_B，E_V，N 均为控制信号，它们由计算机的控制电路根据需要进行的操作在适当的时间产生。

在进行加法操作时，控制电路使 $N=0$，于是 $B_i'=b_i$，全加器的输出为

$$\sum = A + B' + N$$
$$= A + B$$

在进行减法操作时，$N=1$，于是 $B_i'=\overline{B}_i$，

$$\sum = A + B' + N$$
$$= A + \overline{B} + 1$$

若 A、B 均为无符号数，则 $\sum=A\pm B$；若 A、B 为带符号数，则 $\sum=[A\pm B]_{补}$

图 1.1 所示运算电路的各部件(寄存器 A、B 和全加器)的对应位之间分别通过一根公用的数据线相连接。这种结构被称为总线结构，而这样一组连接系统中各个部件的公共信息通道即为总线。显然，在任一时刻总线上只允许存在由一个部件输出的信号。如果同一时刻有两个以上的部件将信息输出至总线，则必然会出现信息混乱。因此在总线上的各个部件的输出端与总线之间均应有控制电路以决定是否允许将其输出送上总线。图 1.1 中用三态门构成线或逻辑。而控制信号(E_V、E_A)为三态门的控制信号，由控制部件保证在同一时刻 E_A 和 E_V 之中至多只有一个为 1，绝不允许二者同时为 1。当 $E_A=1$，

$E_V=0$ 时，$W_i=A_i$.

当 $E_A=0$，$E_V=1$ 时，$W_i=\sum_i$。若此时 $L_A=1$，则由时钟 CLK 同步，将全加器输出 $\sum_7\sim\sum_0$ 送入 $A_7\sim A_0$。即将 A 的内容和 B 的内容相加或相减的结果送入 A 中。

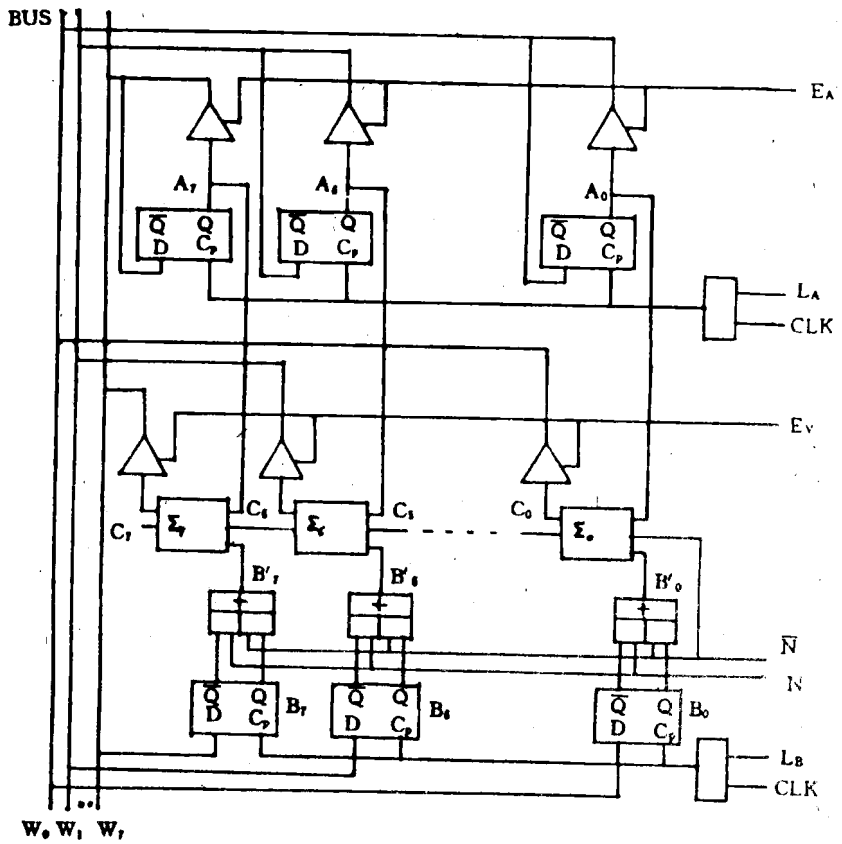

图 1.1　补码运算电路

四、溢出

关于带符号二进制数的运算，还需要讨论溢出问题。

八位带符号二进制数的补码所能表示的数位范围是－128～＋127，其中$[-128]_{补}=10000000$。如果运算结果超出这个范围，则称为产生了溢出。

任何一种运算都不允许产生溢出，除非只是利用溢出进行判断而不使用运算的结果。所以当出现溢出时，应该使计算机停止或输入检查程序找出溢出原因，然后作相应处理。

计算机中常用的溢出判别法是双高位制判别法，为了便于叙述，我们讨论 8 位带符号二进制数补码运算时的情况。只有两个同符号数相加或两个不同符号数相减时才会产生溢出，以下通过两个例题进行讨论。

例 1.1　(＋90)＋(＋107)

$(+90)_{补}=01011010B$

$(+107)_{补}=01101011B$

$$\begin{array}{r} 01011010B \\ +)\ 01101011B \\ \hline 11000101 \end{array}$$

$C_7=0 \quad C_6=1$

在本例中，真正的运算结果是+197，这个数超出了八位二进制补码的表示范围(−128～+127)，也就是产生了溢出。在这种情况下，数值部分需要用八位二进制代码来表示，即数值部分已含有D_7位上的"1"，而运算结果的D_7位是"真正"的符号位与数位部分在D_7位上的"1"之和。如果$D_7=1$，则"真正"的符号位是0；如果$D_7=0$，则"真正"的符号位是1。

我们用C_6表示D_6位上的进位，用C_7表示D_7位上的进位，在本例中，$C_7\neq C_6$。这个结论具有普遍性，即：如果在补码运算中$C_7\neq C_6$，则产生溢出。以下通过一些例题来验证这个结论。

例1.2 (−110)+(−92)

$[-110]_{补}=10010010B \qquad [-92]_{补}=10100100B$

$$\begin{array}{r} 10010010B \\ +)\ 10100100B \\ \hline 1\ 00110110B \end{array}$$

$C_7=1 \quad C_6=0$

本例同样产生了溢出，此时$C_7=1$，$C_6=0$。

由这两个例题不难看出，当$C_7\neq C_6$时，产生溢出。那末，不溢出的情况又怎样呢？分析以下两个例题：

例1.3 两个正数相加，且其和小于+127时：

$$\begin{array}{rll} 0010 & 1101B & [+45]_{补} \\ +)\ 0010 & 1101B & [+45]_{补} \\ \hline 0010 & 1010B & [+90]_{补} \end{array}$$

$C_7=0 \quad C_6=0$

例1.4 两个负数相加，其和大于−128时：

$$\begin{array}{rll} 1111 & 1110B & [-2]_{补} \\ +)\ 1111 & 1110B & [-2]_{补} \\ \hline 11111 & 1100B & [-4]_{补} \end{array}$$

$C_7=1 \quad C_6=1$

可以看出，当$C_7=C_6$时两数相加而不产生溢出。

以上分析了两数相加时，溢出情况的判断，两数相减时，溢出情况的判别完全可以归类于上述各种情况。即减数变补与被减数相加后，若$C_7=C_6$，则无溢出，而当$C_7\neq C_6$时，即为有溢出情况。

五、特征标志