

实用微型计算机程序设计

INTEL 8080

[美]W. J. 威勒 A. V. 沙蔡尔 H. Y. 奈斯著

水利水电科学研究院自动化研究所译

水利电力出版社

内 容 提 要

本书是一本微型计算机汇编语言程序设计专著。使用 Intel 8080汇编语言,通过多种实例,较系统而全面地介绍了微型计算机汇编语言程序设计的方法和若干具体的程序设计技巧。书中主要内容包括微型计算机程序设计的基本知识,各种常用的算术运算程序,堆栈使用和子程序调用,微型计算机上的码制变换,输入/输出和外部设备管理,实时数据收集中的模/数转换,中断处理,以及程序调试等。全书特点是实用性较强。

本书可供应用微型计算机的工作者阅读,也可供高等院校有关专业的师生参考。

实用微型计算机程序设计

——INTEL 8080

[美]W.J.威勒 A.V.沙蔡尔 H.Y.奈斯 著
水利水电科学研究院自动化研究所 译
(根据电力工业出版社1982年版本重印)

*
水利电力出版社出版

(北京德胜门外六铺炕)

新华书店北京发行所发行·各地新华书店经售

水利电力印刷厂印刷

*
850×1168毫米 32开本 8.25印张 216千字

1982年2月第一版

1983年9月新版 1983年9月北京第一次印刷

印数00001—16540册 定价 1.05元

书号 15143·5195

译者的话

本书是一本微型计算机汇编语言程序设计专著。它以 Intel 8080微处理器为举例对象,使用8080的汇编语言,较系统而全面地介绍了微型计算机程序设计的基本方法,并介绍了一些具体的程序设计技巧。书中实例较多,对于开始使用汇编语言进行程序设计的微型计算机工作者,是一本有用的参考书。

原书附录B中的交叉汇编程序文本部分和附录C——俄文字母位格式和等值表两部分,考虑对绝大多数读者用处不大,为了精简篇幅,译文未列入。少数读者如有兴趣,可查阅原文。原书序言最后一段,是作者向一些个人和单位致谢的名单,译文也未列入。

全书由水利水电科学研究院自动化研究所部分同志集体翻译。参加翻译工作的同志为:王金生(序言及第一章),钟道国(第二、三、四章),陈咸铭(第五、六章及附录B一部分),张颖珠(第七、八章及附录B一部分),屠明德(第九章),黄培振(第十章),陈美林(第十一章),孙惠英(第十二章),柳宇强、赵云峰(第十三、十四章),苏开佛(第十五章),梁力(第十六、十七章),李述青(第十八章及附录A)。全书由王金生校订。

由于水平所限,时间又较仓促,误漏难免,欢迎读者批评指正。

梁合庆同志审阅了译稿,提出了很好的意见,特在此致谢。

译者

1981.5.

目 录

译者的话

作者序言

第一章 二进制算术运算及逻辑运算1

两状态码 2的幂 二进制及十进制数的互相转换 十六
进制表示法 二进制加法 负二进制数 一的补码 二
的补码 “与”功能 掩蔽 “或”功能 合并 “异
或”功能 移位 逻辑移位 旋转移位

第二章 计算机的组织与结构.....14

存储器定义 字及字地址的定义 存储器容量 RAM和
ROM存储器 算术和逻辑单元 寄存器 CPU定义
程序计数器 指令寄存器 指针概念 计算机基本指令执
行周期

第三章 机器语言程序设计和汇编程序18

写二进制指令 二进制指令的助记符 用汇编程序翻译符号
程序 源程序 标号和操作数 汇编表 符号表 目
标程序 输入程序 汇编错误标志 未定义和多重定义的
符号 伪指令 交叉汇编程序

第四章 用8080指令传送数据.....25

8080的工作寄存器 在寄存器之间MOV的用法 ADD和
SUB的指令 ORA和XRA AND的用法 在寄存
器和存储器之间传送数据——LDA和STA 交换存储单元
的内容 用MOV在存储器和寄存器之间传送数据 H-L
寄存器对的用途 DATA和DBL伪指令 LHL D的用法
增值的地址指针的应用 SHLD指令 LXI的用途
B-C和D-E寄存器对作指针用 LDAX和STAX
INX和DCX

第五章 8080的二进制算术运算.....35

算术特征位——进位、零、符号的定义 F寄存器 溢出的
定义 进位的定义 溢出与进位的区别 溢出检测与8080
多倍精度数 双精度加法 双精度减法 MVI的使

用	INR和DCR	JMP指令	条件转移	循环
	CMP的用途	伪指令JEQ、JNE、JAL、JGE		
	代数值的比较	RLC和RRC指令	RAL和RAR	
	指令	伪指令LLA和LRA		
第六章	乘法和除法		51	
	以连续相加作乘法	以软件作乘法的过程	乘法循环	CMA
	与TCA伪指令的应用	双精度数求负	特殊因子的乘法	
		以软件作除法的过程	被除数、除数、商和余数的定义	
	除法失效状态	软件除法循环	多倍精度数的乘法与除	
	法	特殊因子的除法		
第七章	堆栈指针的应用		64	
	作为数据存取指针用的栈指针	装栈指针	数据进栈——	
	PUSH	F寄存器	程序状态字	取出栈数据——POP
		使用PUSH和POP交换寄存器对的内容	使用POP	
	直接装入双精度数据	SPHL指令	栈溢出问题	
第八章	子程序		70	
	公用的指令序列	使用LXI和PCHL作子程序调用		
	CALL指令	RET指令	乘法子程序	除法子程序
	子程序自变量传送	自变量直接在调用程序中	自变量地址	
	在调用程序中——ROM的要求			
第九章	数组和表		84	
	数组定义	伪指令RES的用法	在数组中查找最大值和最	
	小值	数组传送	伪指令EQU	数组在存储器中排列次
		顺序的颠倒排序过程——冒泡法	冒泡排序子程序	查找字符
	串数组	伪指令ASC的应用	EQU的进一步应用	查
	表过程	数学函数查表的应用——三角函数正切子程序		
第十章	代码转换成二进制		101	
	十进制形式——BCD码和ASCII码	从ASCII码中		
	分离出BCD码	乘以十的特殊乘法	用连乘法将两位十进	
	数转换为二进制	乘法和加法子程序	有效性检验	合法
	十进制范围的检验	转换中的溢出检验	双精度数转换	
	十六进制转换为二进制			
第十一章	对二进制数进行转换		111	
	用除法转换到十进制	用连续减法转换到十进制	利用进位	
	位的判断转换无符号数	利用符号位的判断转换带符号数		

双精度二进数的转换 双精度二进数转换子程序 转换成十六进制 二进数转换成十六进数子程序 二进数转换成外部数字串

第十二章 基本输入输出——与终端的通信 125

基本 I/O 的功能 (控制、读出和数据传送) 外部设备地址 外部设备接口 IN 和 OUT 的用法 外部设备状态字 利用就绪/未就绪标志的软件延时 用 EQU 定义设备地址
字符打印子程序 将 A 及 P SW 保存在堆栈内 打印串
串打印子程序 多行输出 回车和换行的次序 输入状态字 读字符子程序 读数字串子程序 等待循环问题
作为就绪标志监视的中断系统的概念 I/O 设备接口初始化

第十三章 控制一台复杂的外部设备——Victor

矩阵打印机 140

Victor 矩阵打印机简介 控制的基本元件 6820 外围接口适配器功能介绍 6820 用于 LED 显示 I/O 口的初始化
Victor 打印机口的设置 打印机的驱动 英文文本与符号垂直分辨极限 Cyrillic 字母位格式——RUSKII 码
Cyrillic 字符的查表过程 将子程序综合成完整的打印机驱动程序 Cyrillic 字母样本文本 由 Victor 矩阵打印机打印的全部 Cyrillic 字母

第十四章 在 8080 机上进行十进制运算 167

十进制对二进制的计算 二十进制加法 BCD 的进位 辅助进位位 DAA 指令 BCD 的和数的调整 两位 BCD 和数举例 四位 BCD 和数举例 关于 BCD 减法 九的补码计算 十的补码计算 BCD 减法举例 十进制计算的困难

第十五章 与物理世界的通信 175

数字量、模拟量的输入和输出的定义 由开关寄存器读数字数据 数字输入量变化的检测——逻辑差 数字同时输入输出举例 输出到光显示器 用于诊断的装置 模数变换器介绍 多路器 采样和保持缓冲寄存器 多路器的控制
转换时间 将 A/D 读数转换成工程单位 读 A/D 的子程序
物理测量举例——用热敏电阻测量温度 读数的算术操作

第十六章 中断驱动过程 I: 实时时钟 187

可中断过程 中断的性质 允许中断的任务优先权的方案

被中断任务的状态保护 事件计数中断 实时时钟 开放
 中断的含义 设备待命的含义 能出现中断的情况 中断
 指令 88-VI 的功能 保存8080的寄存器与特征位的次序
 使时钟待命 延时十秒的例子 软件日历钟 复制时钟的
 限制 记载恒星时间 各种类型中断方案的堆栈要求

第十七章 中断驱动过程 II：输入和输出 204

输入中断和输出中断的分类 中断的条件 通过数据传送清除就绪标志 电源接通时待命/解除待命状态的不确定性
 对专用字符监控键盘举例 清除VI的就绪标志 优先权中断系统 输出中断 字符串输出中断服务子程序 在中断控制下同时输入并输出 同时进行两向数据传送的限制 来自不同设备的中断的矛盾

第十八章 调试程序 215

为便于调试进行程序设计的要点 某些错误实践 调试程序
 用调试程序检查和修改内存 从调试转回到执行程序 调试
 伪寄存器 设置伪寄存器 F寄存器各位的说明 设置寄存器对——栈指针H和L、B和C、D和E 设置中断开放/关闭标志 用R命令显示寄存器和特征位 从断点返回的性质 用于调试目的的二进制指令说明 用断点来分步执行程序 校正错误指令举例 应用断点的注意事项 断点返回指令引起的错误 使用陷阱指令插入遗漏指令 RST概述
 调试的经济性 估计错误来源的一些通用步骤

附录 A 调试程序 231

附录 B LSI-2对INTEL8080的交叉汇编程序和ALTAIR 8800的目标装入程序(说明部分) 242

第一章

二进制算术运算及逻辑运算

计算机元件具有处于通和断两状态的性质。对这两种状态无论是称作通和断、一和零、阴和阳或置位和复位是没有多大关系的。计算机的每个操作运算都表现出仅有两种状态这个特点。人们被迫在一个仅有两个数字的系统中思考。通常我们将这两种状态称作 1 和 0。

用两种状态码的方法来表达信息的概念并不是一个新的概念，它比计算机出现得早。只允许用两种状态来表达一个信息的情况，对读者来说是并不陌生的，可以在许多场合遇到^①。摩尔斯码就是一种两状态的例子，这种码仅用点和短横。对于更复杂的信息，像字母，就是用两种状态的组合来构成的，例如第二次世界大战盛行的……码，是字母 V 的代码。

以类似的方法，可将 0 和 1 用于仅有这两个数的数字系统即二进制系统中，来构成更复杂的数字信息。

像我们所熟悉的十进制系统一样，二进制系统也用数串中某一个数的位置来表示其值。例如十进制数 329 表示 9 单位加上两倍该数值系统的基值（即 10），再加上三倍基值的平方（ 10^2 或 100）。数字的位置在十进制中有其特殊意义，在二进制中也是这样。十进制数各位置的值正是其基值从零开始的连续幂。 $10^0 = 1$ （任何数的 0 次幂是 1）。基值的一次幂就是基值本身 $10^1 = 10$ ，二次幂 $10^2 = 100$ ，等等。与此相似，在二进制数中各位置的值就是 2 的连续幂。开始的几项是：

^① 此处删去一个例子——译者。

2 的 幂

$$2^0 = 1$$

$$2^1 = 2$$

$$2^2 = 4$$

$$2^3 = 8$$

$$2^4 = 16$$

$$2^5 = 32$$

$$2^6 = 64$$

$$2^7 = 128$$

$$2^8 = 256$$

一个二进制数的十进制值就是把出现1的各列的值直接相加。二进制数 110 在 2^0 的位置为 0，在 2^1 和 2^2 的位置均为 1，故其值为 $2^2 + 2^1 = 4 + 2 = 6$ 。另一个向十进制数转换的例子，示于例1-1。

例 1-1

将二进制数11010011转换为十进制数。

先写出一组 2 的幂的数值，然后将二进制的数串，写在相应的幂下：

128	64	32	16	8	4	2	1
1	1	0	1	0	0	1	1

在 1，2，16，64和128各列处均为 1，十进制数就是这些 2 的幂的和：

$$1 + 2 + 16 + 64 + 128 = 211$$

当然，2 的幂并不是在 128 处结束，而是无限地延续下去。如同十进制数一样，二进制数也可为任意位长。在许多小型机和微型机手册中载有 2 的幂的更为详尽的表。

注意，在例 1-1 中，要用 8 位二进制数字来表达三位的等值十进制数。这一点就计算机而言不是什么缺陷，但对程序设计人员来说，显得很累赘，抄写长串的数字还容易出错。为此发展了

一些缩写的办法，如八进制系统和十六进制系统。这些系统之所以方便是基于下列事实：即要将二进制数转换成基值为2的整数幂的系统时，仅需将二进制数的位重新组合即可。这个过程很简单。要把上面的数转换为八进制，可将该数从右开始按三位分组：

$$11 \ 010 \ 011$$

这些组即可作为独立量读出。上面的数读为3-2-3。在写这种数时，应设法和十进制数相区别，以免混淆。可以采用加下标的办法，下标表明其基值。这里的基值是8，于是上述数字写为

$$323_8 = 11010011_2$$

处理二进制数更为方便的方法是以16为基值的十六进制，因为在这种系统中，8080的8位数组被分成等位数的两组。16是2的四次幂，所以以四位作一组：

$$1101 \ 0011$$

各组作为独立的数读出。上面右边的一组读起来没有问题，是3。但左边的一组，1101没有相对应的十进制数。其值为 $8 + 4 + 1 = 13$ ，到此为止我们还无法用一个单一的符号来表示13。为此，我们规定用英文字母前六个字A到F来表示十六进制中不能用普通数字表示的那六种可能的组合。十六进制的值为

十 六 进 制 数 值

二进制	十进制	十六进制符号
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9

1010	10	A
1011	11	B
1100	12	C
1101	13	D
1110	14	E
1111	15	F

这些符号开始用时可能觉得复杂，但很快即可熟悉。上表可用于将任意二进制数与十六进制互相转换，如例1-2所示。

例 1-2

将二进制数1100110101001011转换为十六进制。
先将数写成四位一组：

1100 1101 0100 1011

各组对应的十六进制数值可从表上查出。最左边一组是12，即符号C；接下去是13，符号是D；再下一个是4；最右边一组是11，符号是B。故二进制数的十六进制的值为

C D 4 B

用二进制数作算术运算所用的方法与十进制的十分相似。加法是从右边开始，先加第一位的两个数，和数写在下面。如果其和大于一位数的值，则向左一列“进位”一个1。在二进制中一位数所能表示的最大值是1，故任何一列中的和大于1将产生向下一列的进位。二进制的加法过程示于例1-3。

例 1-3

将二进制数00001001和00000011相加。同十进制数加法相同，一数写在另一数的下边：

$$\begin{array}{r} 00001001 \quad \text{被加数} \\ + 00000011 \quad \text{加数} \\ \hline \end{array}$$

两数最右边的一位加起来得 $10_2(2)$ ，一列容纳不下，将 0 写在本列，而 1 进位到下一列。进位以在第二列上边打一个撇来表示：

$$\begin{array}{r} \text{'} \\ 00001001 \\ + 00000011 \\ \hline 0 \end{array}$$

现在加第二列，包括前一列进位，其和为 10_2 。同样本列又写 0，1 进位到第三列，如下：

$$\begin{array}{r} \text{''} \\ 00001001 \\ + 00000011 \\ \hline 00 \end{array}$$

再将第三列与进位相加，其和为 1。此时不产生进位，故直接写出和：

$$\begin{array}{r} \text{''} \\ 00001001 \\ + 00000011 \\ \hline 100 \end{array}$$

再加第四列，这里只有加数与被加数两个数字，第三位并无进位过来，其和为 1，故有：

$$\begin{array}{r} \text{''} \\ 00001001 \\ + 00000011 \\ \hline 1100 \end{array}$$

因其余各列全是 0，其和也全是 0，最后结果为：

$$\begin{array}{r} \text{''} \\ 00001001 \\ + 00000011 \\ \hline 00001100 \end{array}$$

这个和数在 8 及 4 的位置上为 1，故其值为 12_{10} 。
原始数字为 9 和 3，求得的和是正确的。

二进制减法可用同十进制减法相似的方法进行运算，但在计算机中一般并不这样作。计算机减法是用加上减数的负值的方法完成的。为此，我们首先来研究如何表示一个负数。

在一些机器中，字的最左位专门表示符号。这叫作符号-数值系统。在这种系统中，正 3 还是像例 1-3 那样写成：

00000011

负 3 则为

10000011

两数除最左一位外均相同。这一系统前些年用得很广泛，现在用得差不多了。

另一种表示负数的方法是用一的补码的形式。在此系统中，一个数的负数是将该数的每一位均取反。例 1-3 中的负 3 为

11111100

1 换成 0，0 换成 1。在此系统中，减法就是加上一个负数。在运算过程中若高位产生进位，则把它转回来加到字的最低位。为什么这样作是颇有点费解的，但实际上是这样作的，见例 1-4。

例 1-4

用一的补码表示负数作减法 $7 - 4$ ：

将 4 的各位取反，得到 -4：

00000100 = + 4

11111011 = - 4

按照例 1-3 的作法，将 -4 加到 7 上：

00000111 + 7

+ 11111011 - 4

仅将最后一个进位用 1 标出，其和如下：

$$\begin{array}{r}
 1 \\
 00000111 \quad + 7 \\
 + 11111011 \quad - 4 \\
 \hline
 00000010 \quad + 2
 \end{array}$$

这个答案当然是错的。但若将进位转回到低位端并与原结果相加则得：

$$\begin{array}{r}
 00000010 \quad + 2 \\
 + \quad \quad 1 \\
 \hline
 00000011 \quad + 3
 \end{array}$$

这是正确的答案，即 $7 - 4 = 3$ 。这种终端返回是一的补码系统的特点。

在结束一的补码系统之前，我们还应指出这一系统的重要缺点。在此系统中，零有两种不同的表示方法。如果求 00000000 的补码，得到的是 11111111。因为零被认为是无符号的，当检验零值时，这多少有点麻烦，因为对两种情况都要检验。

多数新式计算机，包括 8080 在内，用二的补码作算术运算。在这种系统中，一个数先求对一的补码，然后加 1，就形成这个数的负值。与这样求得的负数相加，可以有效地作减法运算，而高位的进位则直接略去。这个过程见例 1-5。

例 1-5

用二的补码作 $7 - 4$ 的减法运算。

求负 4：先对各位取反，与求一的补码相同：

$$\begin{array}{r}
 00000100 \quad \text{转换为} \\
 11111011
 \end{array}$$

再加上一个 1：

$$\begin{array}{r}
 11111011 \\
 + \quad 1 \\
 \hline
 11111100
 \end{array}$$

为二的补码表示的 - 4

再大到 7 上:

$$\begin{array}{r}
 1 \\
 00000111 \\
 + 11111100 \\
 \hline
 00000011
 \end{array}$$

左上方的 1 表示进位。这时不需终端返回进位答案便是正确的。最高位的进位则直接弃掉。

二的补码系统和一的补码系统相比较，其优点是零的表示方法仅有一个。如果按上面所讲的那样对零取负值，得到的还是零。

$$\begin{array}{r}
 00000000 \quad \text{零} \\
 11111111 \quad \text{每位求反}
 \end{array}$$

再加 1，得二的补码

$$\begin{array}{r}
 1 \\
 11111111 \\
 + \quad 1 \\
 \hline
 00000000 \quad \text{结果仍为零}
 \end{array}$$

高位的进位直接弃掉。

在往下进行之前，还必须对一的补码系统、二的补码系统和符号的特点作一些必要的说明。在本章讨论过的符号-数值系统中，代数符号位从其它位隔离出来置于最左边一位。在一的补码和二的补码系统中，符号必须被看作是填满最高有效位左边所有的位的。如果这一点未了解清除，在以后的一些章节中可能会产生某些混乱。有效位对于正数是 1，对于负数是 0。现考虑二进制数 00001011 和 11111110，第一个数是十一，第二个数是二的补码系统中的负二。十一的最高有效位是在数组中 8 的位置上的 1，负二的最高有效位是在数组中 1 的位置上的 0。所以符号是被看

作填满最高有效位左边所有各个位的。一般应用中，可认为数的最左一位是符号位，但是在处理一的补码系统和二的补码系统时，要记得上面所讲的各点。

在本书中，以后使用“负”这个字时，意味着是二的补码。而“补”这个字，指的是一的补码。一的补码也叫作逻辑补码，二者是同义词。

最后，二的补码系统还有一个值得注意的特点，就是它的负数比正数多一个数。8位数的最大正数为 $01111111_2 = 127_{10}$ ，最大负数是 $10000000_2 = 128_{10}$ 。这个情况偶尔会造成一些麻烦，在本书中出现这种可能会造成麻烦的场合时，将作一些说明。

8080以8位数作运算，运算结果会超出8位。考虑下述加法：

$$\begin{array}{r} 01111000 \quad 120_{10} \\ + 00001001 \quad 9_{10} \\ \hline 10000001 \end{array}$$

这两个数正确的和为 129_{10} 。问题是其和已溢出进入到最左边的符号位。如果这个结果被解释为带符号的数，其值应为 -127_{10} ，这是不对的。产生这个问题的原因在于所求得的是8位数所不能容纳的。这种情况称为溢出。它与高位的进位不同，如果把它当作带符号的数来解释，那结果一定是错的。8080并不包括自动检查溢出的手段。它有检查进位的设施，但无检查溢出的设施。程序设计人员有责任在其程序设计中采取措施，使运算结果能包含在规定的空间内。很多小计算机有硬件的溢出检查，这种检查是靠监视进位位进出最左边一位的情况来完成的。如果进入符号位的进位值与符号位送出的进位值不同，则是产生了溢出，其结果应无效。以后可以看到，这将是一个不必要的隐患。

在进行讨论计算机本身之前，还需了解其它几个二进制运算，即所谓逻辑运算：“与”，“或”及“异或”。

“与”功能的工作和乘法相似，有时称作逻辑乘。两个二进制数的“与”，仅当相应位的两个数均为1时其结果才是1。为了清楚起见，还是用例子来解释。“与”功能示于例1-6。