CROMEMCO微型计算机

硬件资料汇编

（二）

史嘉权　　编译

清华大学出版社

CROMEMCO 微型计算机硬件资料汇编(二)

Z80 汇编语言程序设计

史嘉权 编译

清华大学出版社

内容提要

本书以Z 80指令系统为基础对汇编语言的程序设计作了较系统、较全面的介绍。全书共分十四章。一、二、三章是一些基本概念，分别介绍了汇编语言、Z80 指令系统和伪指令。四～十章结合典型程序对Z80 各类指令在程序中如何使用作了说明，並介绍了程序的基本结构。十一章介绍宏指令和条件汇编。十二、十三章结合 CROMEMCO 系统的机器介绍了系统调用、汇编程序库以及汇编语言程序的上机操作。十四章是汇编语言程序设计综合举例，对汇编语言程序设计的步骤、方法和技巧作了小结。另外，还介绍了可编程序接口的概念，並举例说明其使用方法。最后附有Z 80 指令表，並和 8080 指令作了对照。

本书是以讲义的形式写的，内容由浅入深，循序渐进。

本书可供 CROMEMCO 微型计算机系统用戶和从事微型计算机系统教学、科研及应用等方面的大专院校师生和科技人员参考。

CROMEMCO 微型计算机硬件资料汇编(二)

Z80 汇编语言程序设计

史嘉权　编译

☆

清华大学出版社出版

北京 海淀 清华园

岳各庄印刷厂印刷

北京　　丰台

新华书店北京发行所发行　　各地新华书店经售

☆

开本：787×1902 1/16　印张：15 3/4　字数：403千字

1982年11月第一版　　1983年2月第一次印刷

印数：1～35000

统一书号：15235·54　定价：1.95 元

编 者 的 话

随着大规模集成电路技术(LSI)的发展，在七十年代，出现了在单片硅片上能完成一个运算器和与其相联的控制器功能的微处理器和接口芯片，取得了突飞猛进的发展，用这些大规模集成电路组成微型计算机系统，也迅速发展起来，逐渐形成了计算机技术领域的一个新的技术分支。微型计算机系统，具有功能强、价格低廉、使用方便、体积小等特点，在一般工业企业控制、小型工商企业管理、小规模科学计算及教学等方面得到广泛应用。

美国 CROMEMCO 公司生产的八位微型计算机系统，采用了 Z80(或 Z80A)CPU 作为中央处理器，因此，它具备 Z80(或Z80A)CPU 在成组数据交换和检索、位处理、三种中断方式(特别是向量中断方式)等方面的优点；在接口技术上采用大规模集成电路芯片，如通用双通道(串行和并行通道口)异步收发器 TMS—5501，软磁盘控制器和格式器 FD—1793 (或 FD—1771)，使得接口插件板具有多功能的特性。由 ZPU、64KRAM、TU—ART、16FDC (或 4FDC)等功能插件板及 8″ 软磁盘驱动器、3102 型终端显示器、双向快速打印机 3709(或 3703)等外围设备组成微型计算机基本硬件系统。此外，该公司还提供了各种用途的可扩充系统硬件功能的插件板，如 D+7A A/D—D/A 转换控制板，32K 字节保存器插件板，8 PIO 并行通道接口板等可供用户任意选用。在软件方面，该系统采用了 CDOS 磁盘操作系统，它是在 CP/M 系统软件基础上搞出来的，近年又配置了多用户操作系统软件，在程序语言软件方面，它配置了 Z80 浮动宏汇编和扩展 BASIC、多用户 BASIC，FORTRAN Ⅳ，COBOL 等高级语言。

我校在 1978 年上半年引进一套 CROMEMCO CS—Ⅲ 系统，从 1979 年开始我们对该系统硬件进行剖析，翻译消化技术资料及说明书，查阅了有关技术资料及文献，在此基础上进行了教学工作和技术培训，并根据讲稿初步整理了“CROMEMCO 微型计算机硬件资料汇编”参考资料。

这本参考资料共分四册，第一册 Z80 微处理器原理，第二册 Z80 汇编语言程序设计，第三册接口技术及芯片汇集，第四册 CROMEMCO 微型计算机系统原理分析。该书是由清华大学计算机工程与科学系的计算机车间的同志们集体讨论、分别负责编写的。第一册由乌振声负责编写，朱家维校订；第二册由史嘉权负责编写，朱家维校订；第三册由杨森标负责编写，张公忠、涂连华校订；第四册由陈在勤负责编写，杨森标等同志校订。

本套书第一、二册承蒙林定基同志校阅指正，对此编者表示深切谢意。

由于编者技术业务水平有限，难免出错，敬请广大读者批评指教。

编 者

1982 年 4 月 25 日

前　言

近年来，微型计算机在研制和生产上得到迅速发展，在各个领域得到越来越广泛的应用。硬件和软件紧密结合是微型计算机技术的一个重要特点。而汇编语言程序设计既和硬件有密切联系，又是系统软件和应用软件的基础。因此，学习和掌握汇编语言程序设计的方法和技巧是研制和使用计算机系统，特别是微型机系统，所不可缺少的。

目前，研制、生产和使用最多的是8位微型机，而大部分8位微型机以Z 80-CPU为中央处理器，使用Z80指令系统。Z80指令系统内容丰富、功能较强，中断灵活，兼容8080指令系统。因此，以Z80指令系统为基础来介绍汇编语言的程序设计是有典型意义的。本书正是本着这个原则对汇编语言的程序设计作了较系统、较全面的介绍。内容由浅入深，程序从简单到复杂，尽量做到循序渐进，通俗易懂。

指令系统是汇编语言的基础。本书首先介绍了各类指令的基本功能和通常的使用场合。要分析和设计较复杂的程序，必须对程序的结构有清晰的了解。本书介绍了顺序结构、循环和多重循环结构、简单分支和多路分支结构、子程序、子程序嵌套、递归调用以及中断结构等。可为分析和设计复杂的程序打下较好的基础。分析典型程序是学习程序设计的捷径。本书本着这一观点，以大量的篇幅介绍了许多不同类型的程序。通过典型程序把指令功能、程序结构和程序设计的方法、技巧有机地结合起来。在分析典型程序过程中，使用了程序设计中最常用的方法——流程图法，可使读者对程序的结构和功能有完整的了解。

本书最后对程序设计的步骤、方法和技巧进行了归纳和总结，其中着重介绍了程序结构设计的基本概念和方法，并通过几个综合性的例题作了具体的介绍。例题中用到了微型机系统中必不可少的可编程序接口电路。

本书的大部分内容对于以Z80-CPU为中央处理器的微型机都是适用的，并不局限于具体的机型。只有系统调用命令、汇编程序库子程序以及上机操作的具体键盘命令是以CROMEMCO系统为背景，但所涉及的概念、原则以及步骤、方法对其他微型机系统也同样适用。

本书作为"CROMEMCO微型计算机硬件资料汇编"的一个分册，是由于和其他三个分册有内在的联系，但是，汇编语言程序设计作为软件的基础，又有其自己的独立性。所以，本书从内容上，又自成体系，保持其本身的完整性。只要对计算机有一定的基础知识便可单独阅读。

一至四册总目录

目　录

第一章　汇编语言概述

大家都知道，用计算机可以解决科学计算、过程控制、信息处理和事务管理等多方面的问题。然而计算机总是把各种复杂的问题最终归结为一系列最基本的操作，即执行一条条的机器指令。识别并执行由二进制代码构成的机器指令是计算机的最基本功能。用二进制代码编写的程序叫目的程序。因为二进制表示太繁琐了，而一般的微型计算机系统可由键盘用16进制输入目的代码，所以微型计算机的资料中用十六进制表示。用二进制或十六进制表示指令和地址称为机器语言。机器语言对人们来说，很难识别和记忆，编程序时容易出错，这给程序的编写和阅读带来极大的困难。于是，人们就用汇编语言代替机器语言：用记忆符代表指令的操作码和操作数，用标号或符号代表地址、常数或变量。记忆符一般都是操作码的英文字母的缩写，便于识别和记忆，因此，用汇编语言编写和修改程序比用机器语言方便多了，但用汇编语言编写的程序(称为源程序)机器不能直接识别和执行，必须把它翻译成目的程序，这个翻译过程叫做汇编。用来把汇编语言源程序自动翻译成目的程序的程序叫做汇编程序。

图 1—1　汇编程序功能示意图

在汇编语言中有一类特殊的指令，它们告诉汇编程序如何进行汇编，而在汇编以后，它们本身并不转换成目的代码，我们称这类指令为伪指令。关于伪指令将在第三章中详细介绍。

源程序由语句组成。一般来说，汇编语言的一个语句和机器语言的一条指令相对应。汇编语言和机器是密切相关的，是面向机器的语言。不同系列的机器有着不同的汇编语言。如DJS——130，PDP——11 和 Z80 的汇编语言各不相同。

还有另一类程序设计语言，称为高级语言，这是面向过程的语言。这种语言更接近英语和数学表达形式，一般的用户更容易掌握。高级语言的一个语句相当于许多条汇编语言的语句或机器语言指令。因此，对于同样的问题，用高级语言排程序往往要比用汇编语言简便得多。当然，高级语言程序更不能直接为机器所识别和执行，而需要通过专门的程序翻译成目的程序。现在世界上已经有许多种高级语言，用得比较多的有 BASIC，FORTRAN，COBOL，PASCAL，ALGOL 等。高级语言不依赖于计算机的结构和指令系统，所以，同一种高级语言在不同的机器上基本是一样的，可能会有些细小的差别。

既然有了高级语言，汇编语言是不是可有可无了呢？其实不然。因为计算机的许多系统程序是用汇编语言编写的，因此，要分析、研究以及自己设计系统程序，就要熟练地掌握汇编语言。另外，和高级语言相比，汇编语言有它自己的优点：得到的目的程序比较短，节省内存容量，执行速度快，能准确计算执行时间，适于实时控制，便于管理接口电路等。特别是微型计算机成本低，可靠性高，体积小，可广泛地用于控制系统中，因此，掌握微型机的汇编语言更有它特殊的重要性。

1.1 机器语言和人工汇编

机器语言是任何程序的最基本的形式，它包括操作码和操作数两部分。操作码和操作数都是以2进制或16进制数表示的。例如，假设需要编一个短的程序，把一到十的数相加。下面就是一个完成此任务的效率很低的办法：

```
XOR     A       ; 清除A
ADD     A, 1
ADD     A, 2
ADD     A, 3
ADD     A, 4
ADD     A, 5
ADD     A, 6
ADD     A, 7
ADD     A, 8
ADD     A, 9
ADD     A, 10
```

这个程序包含一条清除A寄存器的指令(XOR A)和十条立即数型指令，把一到十的数和累加器的内容相加。指令是用记忆符表示的，操作数由寄存器及8位立即数组成，这里使用的

地址	指令	机器代码
0100H	XOR A	AFH(即10101111)
0101H	ADD A, 1	C6H
		01H
0103H	ADD A, 2	C6H
		02H
0105H	ADD A, 3	C6H
		03H
0107H	ADD A, 4	C6H
		04H
0109H	ADD A, 5	C6H
		05H
010BH	ADD A, 6	C6H
		06H
010DH	ADD A, 7	C6H
		07H
010FH	ADD A, 8	C6H
		08H
0111H	ADD A, 9	C6H
		09H
0113H	ADD A, 10	C6H
		0AH

图 1—2 人工汇编处理的程序1

记忆符是 Zilog 公司规定的，它和机器语言代码等效。要汇编出等价的机器语言代码，先要查手册找出 16 进制形式的操作码和指令格式，从而得到机器代码，然后把机器代码写在用记忆符表示的每条指令的旁边，如图 1—2 所示。XOR A 是一字节指令，形式是$(10101\text{-}r\text{-})_2$，其中 r 是指令中用到的寄存器。在这种情况下，$r=111_2$，表示 A。ADD 是 2 字节指令，操作码是 11000110_2，后面跟着 8 位的立即数。例如，ADD A，8 的等价机器语言操作码是 11000110_2，即 C6H，第二个字节是 00001000_2，即 08H。

表示一到十的加法运算的整个程序可以由键盘输入到 Z80 系列的微型计算机中，然后执行。送入的实际数字在图 1—2 的右侧给出，共 21 个字节，这是表示这段程序的机器代码。

让我们再编一个程序来进一步说明汇编成机器语言的过程。这是以另一种方法实现一到十的加法运算的程序。我们仍用 A 寄存器存放累加和，但用 B 寄存器存放下次要加的数。运算按相反的顺序进行，从十加到一，即先加 10，然后加 9，再加 8……一直加到 1。当检测出下次要加的数是 0 时，就停下来。用 Z80 记忆符表示的程序如下：

```
        XOR     A          ; 清除 A
        LD      B, 10      ; 设置计数值为 10
LOOP    ADD     A, B       ; 加下一个数
        DEC     B          ; 准备下一个数
        JP      NZ, LOOP   ; 若不为 0，则转移
        HALT               ; 暂停
```

首先，通过 XOR A指令把寄存器 A 清0。然后，通过 LD B，10 指令把 10 装入 B 寄存器。下面三条指令构成一个循环。当 B 中的数是 10 到 1 时，B 的内容将加到 A(ADD A，B)，然后 B 的内容减 1 (DEC B)，再转移到这个循环的第一条指令——标有“LOOP”的指令，LOOP 作为转移的入口。当 B 的内容减 1 后，如果结果为 0，则零标志(即 Z 标志)置 1，如果结果不为 0，则零标志置 0。如果 B 寄存器不为 0 (9 到 1)，JP NZ, LOOP 指令将检测到非 0 (NZ)，而转移到 LOOP。如果 B 寄存器为 0，则零标志为 1，转向 LOOP 的条件不满足，于是 CPU 执行下一条指令(HALT)。

对于这个程序，要人工汇编成机器代码，比前面的例子稍微复杂些。首先，前面的程序可以装在存贮器的任何位置，因为指令中不包含地址，而这个程序中包含地址 (JP NZ，LOOP 指令必须在指令的第二、三字节规定 LOOP 的地址)，因此必须确定程序存放在存贮器的什么位置。我们假定选择 0100H 单元作为起点。汇编过程的第一步是以字节来计算每条指令的长度和写出每条指令相应的记忆符，这步做完后，程序表示如下：

地址	长度	指令		
0100	1		XOR	A
	2		LD	B,10
	1	LOOP	ADD	A,B
	1		DEC	B
	3		JP	NZ,LOOP
	1		HALT	

汇编过程的第二步是根据指令的长度来填写每条指令所在的地址。这里标出的每个地址都是指令的第一字节的地址。

地址	长度		指令	
0100H	1		XOR	A
0101H	2		LD	B,10
0103H	1	LOOP	ADD	A,B
0104H	1		DEC	B
0105H	3		JP	NZ,LOOP
0108H	1		HALT	
0109H				

作完这一步之后，从"地址"这一栏所得到的整个程序的长度（0109 H－0100 H＝9 字节）和从"长度"得到的整个字节数(9)应该是一致的。现在，可以把指令格式填满，如图 1—3 所示。唯一麻烦的指令是 JP NZ, LOOP。这是一条三字节指令，其中，后两个字节指出条件转移的地址。现在要转移到 LOOP 即0103H 单元，这个地址必须以相反的次序装入第二和第三字节。实际上这种格式由来已久(从前的 8008 就是这种格式)。

地址	长度	机器代码		指令	
0100H	1	AF		XOR	A
0101H	2	060A		LD	B,10
0103H	1	80	LOOP	ADD	A,B
0104H	1	05		DEC	B
0105H	3	C20301		JP	NZ,LOOP
0108H	1	76		HALT	
0109H					

图 1—3 人工汇编程序2的过程

虽然用人工的方法汇编较长的程序也能做得到，但是非常繁琐。在计算存贮单元、填充指令字段和按格式编地址时出现错误的机会太多了。除了容易出现错误以外，还有一些其它因素使得不能用机器语言编写程序。其中，最重要的是可浮动性的问题。程序 2 只能从 0100H 单元开始存放，要从其它单元开始存放，就必须改变转移指令的地址。这样，在比较大的程序中，许多地址都必须重新计算。第二个因素是不易于编辑。在新的程序按照预期的方式运行以前，往往需要进行若干次调试，而每次调试都可能增加、删掉或修改程序中的指令，这样就必须重新计算程序中所使用的地址。对于用机器语言编写的程序，要处理这些问题是极不方便的，而且，稍有疏忽，又会带来新的错误。

1.2 汇编语言的格式和规则

在汇编语言中，指令用记忆符表达式来表示。记忆符表达式是写指令的简单而方便的方法，因为写"ADD A, B"比写"B 寄存器的内容和 A 寄存器的内容相加"简单得多。这本书中使用的记忆符都严格地按照 Zilog 公司的规定。在附录 A 中列出了所有的指令记忆符和可能的寻址方式。

汇编语言由语句组成。每个语句可包括一至四段：标号段，操作码段，操作数段和注释段。Z80 汇编语言的格式如图 1—4 所示。

列数	标号	操作码	操作数	注释
	1 8	9 16	17 24	25 80
	LOOP	LD	B, VALUE	; 得到数值
		ADD	A, B	; A 加 B
		EI		; 开中断

图 1—4 典型的 Z80 汇编语言格式

每个 Z80 指令都必须有一个操作码，大多数指令有操作数，如："LD B, VALUE"指令，B 和 VALUE 是两个操作数项。也有的指令，如 EI，没有操作数。标号段是可有可无的。当有标号时，它可以是一至六个字母或数字组成的字符串，其中第一个字符必须是字母。注释段说明一条指令或一段程序的功能，也是可有可无的。各段之间用空格、逗号、冒号、分号等分界符分开。

汇编语言的一个语句占一行，表示一条 Z80 指令。行的长度通常取决于输入设备上一行的长度(如键盘显示终端一行通常是 80 列)，汇编程序可识别的行结束标记一般是回车、换行代码。

关于 4 个段再进一步说明一下：

1. 标号

标号一般用于表示地址，但也可以表示数据。用户使用标号编写汇编语言程序会更加方便，而且会减少发生错误的机会。

标号是任选的(可有可无的)，可由一个或几个字符的字符串组成，字符串中间不能有分界符，并且只有前 6 个字符是有效的。例如，标号"longname" 和 "longnamealso"被认为是同样的标号。如果标号后面紧跟着冒号，则标号可以从任何一列开始；如果标号从第一列开始，就可以不加冒号。大写和小写可表示不同的标号，如 LABEL 和 label 表示两个不同的标号。

指令的操作码、伪指令、寄存器和寄存器对的名字以及条件标志在汇编程序中是作为关键字保留的，不能作为标号。其中寄存器的名字有：A、B、C、D、E、F、H、L、I 和 R，寄存器对的名字有：AF、BC、DE、HL、IX、IY、SP、AF′，条件标志有 C、NC、Z、NZ、M、P、PE、PO。

下面举几个正确标号的例子：LOOP, L1, HERE, ABC:。

下面是几个不正确的标号的例子：123:（以数字开头），LO OP（中间有空格），ADD（操作码），EQU（伪指令）。

如果标号多于 6 个字符，仅取前 6 个，如"INSTRUCTION" 将作为"INSTRU"。

因为标号表示指令地址，故同一标号不能代表两个不同地址。如：

```
HERE    JP      THERE
          ⋮
THERE   LD      C, D
          ⋮
```

THERE CALL SUB1

在这段程序中，标号 THERE 的含义不清楚，汇编程序无法确定 JP 指令的转移地址。

2. 操作码

用记忆符表示指令的操作码，指出要执行的操作。如 LD 表示数据的传送，而不管数据的传送是在不同的寄存器之间进行，还是在寄存器和存贮单元之间进行。

在操作码段后面必须至少有一个空格。

如 HERE JP THERE 是正确的，而

HERE JPTHERE 是错误的。

3. 操作数

操作数提供汇编程序所需要的信息，以便和操作码一起确定指令要执行的操作。根据操作码的情况，操作数段可以是空白、一项或用逗号分开的两项。

有 4 种类型的信息可以作为操作数项：寄存器、寄存器对、立即数、16 位存贮器地址。这 4 种信息共有 9 种表示方法：

a) 16 进制数据

b) 10 进制数据

c) 8 进制数据

d) 2 进制数据

e) 程序计数器当前值 ($)

f) ASCII 常数

g) 赋予值的标识符

h) 标号

i) 表达式

这些表示方法可分成几类说明如下：

1) 数制

上面的前 4 种表示方法表明汇编程序允许源程序中使用四种不同的数制。这就是说，程序中的常数和地址可以用最方便的数制来规定。数都要以数字开头(若以字母开头，则前面应补 0)，并可以有规定数制的单个字母跟在后面："B" 表示二进制，"O"或"Q"表示八进制，"D" 表示十进制，"H"表示十六进制。如果没有规定数制的字母，就约定是十进制。

```
如：HERE    LD    C, 0BAH          ; 16 进制数 BA 送入寄存器 C
    ABC     LD    E, 105           ; 10 进制数 105 送入寄存器 E
    LABEL   LD    A, 72Q           ; 8 进制数 72 送入累加器
    NEW     LD    B, 11110110B     ; 把二进制数 11110110 送入寄存器 B
```

又如，十进制数 92 可用四种数制分别表示如下：01011100B, 134Q, 134O, 92, 92D, 5CH。

2) 符号 $

美元符号 "$" 表示程序计数器当前值，即表示当前指令的第一字节地址。这是汇编语言程序中常见的一个符号。

```
如：   GO   JP  $+6              ; 由当前字节向下移动 6 个字节
```

3) ASCII 常数