微型计算机通讯

第四集

WEIXING JISUANJI
TONG XUN

科学技术文献出版社重庆分社

编者按：1979年12月我们出版了《微型计算机》——Z-80专集，对该机的硬件和指令系统作了全面介绍。在此基础上，本专集第一部分选编了《Z 80微型计算机手册》中的软件部分，着重介绍Z-80汇编语言程序设计以及如何有效地利用Z-80指令系统等问题。为了系统性起见，还对以Zilog为主的五家公司生产的Z-80产品，从单板机直至开发系统作了简介。本专集第二部分选编了Z-80开发系统软件用户手册。

由于水平所限，在选材、译校和审编工作中，一定会有这样那样的错误和不当之处，切望读者批评指正。

微型计算机通讯 第四集

中国科学技术情报研究所重庆分所　编辑
科学技术文献出版社重庆分社　出版
重庆市市中区胜利路91号
四川省新华书店重庆发行所　发行
重庆印制第一厂　印刷

开本：787×1092毫米1/16　印张：9$\frac{1}{4}$　字数：24万
1981年第一版　1981年2月第一次印刷
科技新书目：167—128　印数：0001—4580册

书号：15176·431　定价：1.00元

目　　录

Z-80微型计算机手册(软件部分)

第九章　Z-80汇编程序

在我们出版的专集《微型计算机Z-80》中已对该机的硬件和指令系统作了全面介绍。本文将讨论如何有效地利用此指令系统，来建立一组执行乘、除、双精度和多精度算术运算、表和字符串操作的指令。为便于编写软件程序，常使用汇编程序。该程序为把较高级的符号汇编语言自动汇编成机器语言指令，提供了方便的途径。

机器语言

机器语言是任何程序的最基本形式，它由实际机器语言操作码和执行程序指令所需的操作数组成（用二进制或十六进制数表示）。例如，假定要编一个短程序，把1到10以内各数加起来。下面便是完成这一任务所需的极低效的方法：

```
XOR   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
```

此程序由清除A寄存器指令（XOR）和一串立即指令组成，这一串立即指令用来把1到10以内的数加到累加器的内容上。程序是用Zilog公司用来表示相应机器语言码的记忆符、所用的寄存器和8位的立即数据值编写的。为汇编成等效的机器语言码，应查出操作码的十六进制表示形式和指令格

机器码	指令格式	指　令
AFH	10101 | 111	XOR A
C6	C6H	ADD A,1
01	01H	
C6	C6H	ADD A,2
02	02H	
C6	C6H	ADD A,3
03	03H	
C6	C6H	ADD A,4
04	04H	
C6	C6H	ADD A,5
05	05H	
C6	C6H	ADD A,6
06	06H	
C6	C6H	ADD A,7
07	07H	
C6	C6H	ADD A,8
08	08H	
C6	C6H	ADD A,9
09	09H	
C6	C6H	ADD A,10
0A	0AH	

图1.1　由手工汇编得到的程序1

式，并把它们写在每条指令的记忆符的旁边，如图1—1所示。由图可见，XOR A是$10101RRR_2$形式的单字节指令，其中R表示所用寄存器。在这种情形里$R=111_2$，表示寄存器A。ADD是一双字节指令，由11000110_2和后随的一个8位字段组成，这个字段代表8位的立即操作数0以ADD A，8指令为例，与它等效的机器码指令是操作码11000110_2或C6H，后跟00001000或08H。

由1到10十个数的整个加法程序，可以通过控制面板（如果有的话）或监督程序输入到Z-80微型计算机，并加以执行。用按键输入的实际数字是图1—1最左边的一排数字，共21个8位的机器码字节。

现再举一个程序例子，来说明机器语言的汇编过程。这次将以略为不同的方式来编写和汇编从1到10十个数的加法程序。和上例相同，我们用寄存器A保存和数，但用寄存器B存放当前的待加数。因为我们是以相反的顺序进行操作，也即先加10，后加9，再加8，等等，最后加1，所以B寄存器内的数将从10变到1。再往下我们将检测到待加数为0，于是程序停止执行。用Zilog记忆符写的程序有如下形式：

```
        XOR     A           清除寄存器A
        LD      B, 10       将计数值置为10
LOOP    ADD     A, B        加下一个数
        DEC     B           准备下一个数
        JP      NZ,LOOP     如该数不为零则
                            转LOOP
        HALT                停机
```

首先XOR A指令将寄存器A清零。然后通过LD B,10指令，将10装入寄存器B。下面的三条指令组成一个循环。只要B内的数是10到1，B的内容就将加到A(ADD A, B)。然后B的内容减1(DEC B)，并转移到标有“LOOP”（作为返回参考点的）程序循环的第一条指令。若寄存器B减1的结果为0，则状态标志Z置1；若不为0，则置0。如果寄存器B不为零（而是9到1以内的任何数），JP NZ, LOOP指令将检测到非零（NZ）并转回到LOOP。如果寄存器B内容为0，则Z标志置1，且不产生条件转移，而是使CPU执行下一条指令（HALT）。

这个程序的手工汇编机器码比第一个例子略为复杂。首先，第一个程序的指令中不包含地址，因而它可以浮动或装入存储器的任何位置。但第二个程序包含有地址（指令JP NZ,LOOP必须在其第二、三字节中指定LOOP的地址）。因而必须做出决定，这个程序应从存储器的何处执行。我们任选第0100H单元作为起点。下一步是计算每条指令以字节数计算的长度，并写在每个记忆符边上。这一步完后程序有如下形式：

```
存储单元   长度   指令
0100H      1              XOR A
           2              LD B, 10
           1      LOOP    ADD A, B
           1              DEC B
           3              JP NZ, LOOP
           1              HALT
```

现在利用这个长度，就可以补上每条指令的存储单元。后者总是指存放指令第一个字节的存储单元：

```
存储单元   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
```

为检验这一步是否做得正确，可从存储单元这一列求得程序的总长度（0109H—0100H＝9字节）和从长度一列内求出总字节数（9），并将它们进行比较。下一步是填入指令格式，如图9—2所示。唯一困难的指令是JP NZ, LOOP。这是一条三字节指令，最后二字节指定条件转移地址。因为是转移到0103H单元内的LOOP指令，这个

存储单元	长度	机器码	指令格式		指令
0100H	1	AF	10101 111		XOR A
0101H	2	060A	00 000 110	0000 1010	LD B,10
0103H	1	80	10000 000	LOOP	ADD A,B
0104H	1	05	00 000 101		DEC B
0105H	3	C20301	11 000 010	0000 0011 0000 0001	JP NZ, LOOP
0108H	1	76	0111 0110		HALT
0109H					

图 9—2　程序 2 的手工汇编过程

地址必须象习惯所做的那样（或至少自8008开始做的那样）以相反的次序（03H、01H）放入第二和第三字节。

虽然可以用手工方法汇编长程序，但这样做是极不经济的。在计算存储单元、填入指令字段和按格式放地址时，很易出错。除此以外，还有若干其它因素使机器语言操作难以实行。其中最重要的是可浮动性。第二个程序只可能从0100H开始执行。为从另一个存储单元执行，转移指令中的地址应作修改。在较大的程序中，有许多地址得重新计算和手工汇编。第二个因素是编辑的方便性。程序很少是一次就通过的，大多数程序在通过之前要进行若干次迭代，每次迭代包括增、删或修改程序中的指令，重新计算指令地址。

汇编过程

由于手工汇编有其固有的限制，所有的微型机制造厂都提供了汇编程序，用来完成从符号汇编语言到机器语言的自动翻译。在许多情形里汇编程序可以由微型机本身执行，这称为驻留汇编程序；有时汇编程序要在另一台计算机上执行，这称为交叉汇编程序。不论在何种情形里，汇编程序将迅速汇编用Z-80或其它源汇编语言编写的程序而产生代表机器码的目标程序块，并以汇编语言和机器语言两种形式列出被汇编程序。汇编程序的特点是：

1．存储单元、操作码和变元的符号表示；

2．注释部分和符号形式的指令可混在一起；

3．向前和向后引用符号存储单元时的自动汇编；

4．各种基数的自动表示；

5．表达式的计算；

6．定义存储单元、使两符号相等、保留存储器和执行其它适当功能的伪操作或不产生目标码的汇编程序指令。

汇编语言格式

本文通篇都用记忆符来表示指令。这是一种简便的书写指令的方法，因为写“ADD A，B”比之写”将寄存器B内容加到寄存器A的内容”，要简单得多。文中所用的Z-80记忆符和Zilog公司所用的基本相同，但寻址类型的表示方法略有区别。第五章的表或附录C列出了所有指令记忆符和可能的寻址格式。

本书所用的标准汇编语言格式示于图9—3。共有四列：标号列、操作码列、变元列和注释列。Z-80指令的每一种表示方式都必须具有操作码。大多数指令都有变元（操作数），如“LD (HL)，R”，其中(HL)和R是二个变元。EI或HALT等指令

列号→	标　号		操作码		变　元		注　释
	1　　6	7	8　　11	12	13　　27	28	29　　64
	名字12		LD		(HL),R		这是一行源程序的实例
	名字13		EI				开放中断
			JMP		NZ,STOP		转STOP,等待,以作中断处理
	名字，可有可无，由1-6个字符组成	空白	操作码，2-4个字符组成	空白	变元，可按需要填入	空白	注释，可有可无

图 9 — 3 典型的Z-80汇编语言格式

没有变元。标号段可有可无。如果有标号，则它可以是一到六个字母或数字符号，且第一个符号必为字母。注释列也是可有可无的，它说明指令的功用（见图 9 — 3）。这四列组成汇编语言的一行。一般说来、汇编语言一行的长短，取决于输入设备如电传打字机和穿孔卡读出机的行长度。实际上，就象在后面讨论的汇编程序中那样，每行的结束是由回车换行符或类似的面向外设的符号表示的。

一般说来，每一行汇编语言（或称源语言行）表示一条 Z-80指令的全部信息。每一行将产生代表该指令的一到四个字节。这一规定有若干例外，其一是注释行，它从分号开始，是非再生性的，也即它不产生机器语言代码，而仅作参考用。图 9 — 4 示出的，是从一页典型的汇编程序清单中取出的一部分。打印在最左边的是源程序的行号，

```
105  ;BXASH-00-00
106  ;
107  ;FUNCTION:THIS SUBROUTINE CONVERTS AN 8-BIT BINARY VALUE
108  ;IN THE C REGISTER TO TWO ASCII HEX DIGITS
109  ;(功能，此子程序将寄存器C中的8位二进制值变换为二个ASCII 16进制数字)
110  ;
     ;调用顺序：(HL)=缓冲区指针；(C)=待变换的8位值；
     ;        调用 BXASH（字/符号送回缓冲区，缓冲区+1和HL加2）

118   1036  3EF0    BXASH  LD    A,F0H
119   1038  A1             AND   C       MASK 1
120   1039  0F             RRCA
121   103A  0F             RRCA
122   103B  0F             RRCA          ALIGN FOR
123   103C  0F             RRCA          CONVERSION
124   103D  CD4710         CALL  CVERT   CONVERT
125   1040  3E0F           LD    A,FH    MASK 2
126   1042  A1             AND   C       GET SECOND CHARACTER
127   1043  CD4710         CALL  CVERT   CONVERT
128   1046  C9             RET
129
130   1047  C630   CVERT  ADD   A,30H    CONVERT TO 0-15
131   1049  FE10           CP    10      TEST FOR 0-9
```

图 9 — 4 典型的Z-80清单

其次是指令第一个字节的存储单元，和代表该指令机器语言代码的八个十六进制数字（二个十六进制数字表示一个字节）。这是汇编程序产生和列出的信息。这右边是源程序行本身。另外在清单上还打印有指明汇编错误的诊断信息，如汇编程序访问到未定义或重复定义的符号，非法变元（如非法的十六进制数字等）。因为清单的格式取决于微型计算机系统和汇编程序的类型，所以此处的讨论仅提供一般的情况，以说明典型的清单是怎么个样子，而不作为详细的指南。

符号表示

源程序行的标号列表示存储单元的名字。程序可以写成仅访问绝对地址的形式，

```
10AAH       LTHAN
                ↓
1202H       CMPARE      CP    B           比较A和B
1203H                   JP    Z,EQUAL     A=B转EQUAL
1206H                   JP    M,LTHAN     A<B转LTHAN
1209H       GREATR
                ↓
14AFH       EQUAL
```

箭头表示未定义的指令，左边的存储单元是汇编后得到的存储单元。大多数汇编程序要扫描两遍。第一遍对记忆符译码，尽可能地组成指令，计算出指令字节，和构成符号表，将程序中所有标号和符号列入其中。第二遍用符号表判别所有地址。所以要有二遍，是因为在见到符号之前不能完成向前引用。第一遍扫描上述程序之后将得到下列的符号表：

符号	数值
CMPARE	1202
EQUAL	14AF
GREATR	1209
LTHAN	10AA

第二遍扫描，EQUAL和LTHAN的值将被填入1203H和1206H处的转移指令内。

Z-80系统中保留着一些符号，是程序员所不能使用的。汇编程序准备用这些符号来指定寄存器或寻址方式。这些符号中有许多出现在附录C的指令格式中。Z-80系统保留的字包括：

寄存器名字：A、B、C、D、E、F、H、L

寄存器对名字：AF、BC、DE、HL、IX、IY、SP、AF′

条件码标志：C、NC、Z、NZ、M、P、PE、PO

数基的表示

所有汇编程序都具有的另一个特点是能把一种数基交换成另一种数基。这就是说，指令的变元可以用最方便的数基表示。例如，指令ADD A，N是将8位的立即数加到A寄存器的内容上。在N的二进制、十进制和十六进制值的后面加上后缀B、不加后缀或加H，就可以指定三种数基中的任何一种："ADD A，100"，"ADD A，64H"和"ADD A，01100100B"全都指同一件事，

如"LD A，123A H"。但在这种情况下就要确切知道待用的存储单元，必须规定变量和常数用的数值地址。写"LD A，RESULT"比之指定绝对地址，要方便得多。汇编程序将自动地把符号"RESULT"转变成相应的机器码地址。对符号的引用可以分为对已定义符号的引用和对待定义符号的引用，前者称为向后引用，后者称为向前引用。下面通过一个短程序来说明汇编程序是怎样分辨符号的。

下面的程序通过比较指令比较A寄存器和B寄存器内容，然后根据A<B、A=B或A>B而分别转移到三个存储单元LTHAN、EQUAL或GREATR。LTHAN是向后引用，而EQUAL和GREATR是向前引用。

即把100_{10}加到A寄存器的内容。这三个后缀将用在本文的例子中，但特定的Z-80汇编程序实际用的格式，无疑会有差别。

表达式求值

大多数汇编程序具有有限的表达能力。表达式可以由符号和字母数据组成，在较完善的汇编程序中可有绝对和浮动符号。表达式的算符可作加、减、乘操作，有时还可作除和移位操作。算符通常可用预定的符号表示，如“+”、“−”、“.”和“/”表示加、减、乘和除。复杂的表达式很少用在汇编语言程序中，在某些情况下可能超过汇编程序的能力所及，但是较简单的表达式可以汇编表的长度，计算系统参数，和产生数据字内的字段，这一章和其它章将提供例子。

伪操作

在每一个源程序行中，负责产生指令操作的是操作码。但是还有一些汇编程序操作记忆符是不产生机器语言指令的，而是告知汇编程序做某些特殊的操作。这些操作称为伪操作，因为它们并不是代表合法机器语言指令的真正的操作码。本文所讨论的伪操作和所有汇编程序中的伪操作相似。如下面所示，括号表示可有可无的标号。

标号（如果有的话）	伪操作	变元（如有的话）
	ORG	N
	END	
NAME1	EQU	NAME2
(NAME1)	DEFB	N
(NAME1)	DEFW	N
(NAME1)	DEFS	N
(NAME1)	TXT	STRING

伪操作ORG建立程序原点。例如在第一行源程序之前写上“ORG 1200H”，则汇编程序指令计数器将置为1200H。以后的每条指令将使指令计数器增加一个数，这个数等于该指令的字节数，这样汇编程序可以记录符号位置和现行指令的位置。ORG还可以在任何时刻出现在程序中，使汇编从新的位置进行。

END伪操作码是程序的最后一个语句。它告知汇编程序开始第二遍扫描或结束汇编过程。

EQU使一个标志和另一个标志或数值相等。这个伪操作通常用来把常数或表达式赋给可识别的名字。下面是表示表长度的EQU的例子。其中“$”代表现行的汇编程序存储单元(汇编程序指令计数器的内容)

```
存储单元                    源程序行
0100H     TABLE
0101H
0102H
0103H
0104H
0105H
          LENGTH    EQU   $-TABLE

103FH               LD    IX,LENGTH
```

在此例中表长度是0106H（现指令计数器内容）—0100H(表的起点)或6个字节。EQU并不产生代码，而是把数值6记入符号表“LENGTH”的名下。以后在程序中遇到16位立即指令LD IX,LENGTH时，汇编程序将搜索符号表，从中找出符号LENGTH，得知其值为6。执行LD IX,LENGTH指令将把6取入IX寄存器。

伪操作DEFB和DEFW定义程序中的常数和变量。DEFB的变元是8位(二进制)的数值或符号表达式，DEFW的变元则应是16位。两种伪操作都是必不可少的，否则汇编程序无法产生数据表、常数表或变量的存储单元。以下几行源程序产生十个字节的表，每个字节表示1到10以内的数据。

```
0100    01    TABLE   DEFB   1
0101    02            DEFB   2
0102    03            DEFB   3
0103    0405          DEFW   0405H
0105    06            DEFB   6
```

```
0106    0708        DEFW  0708H
0108    09          DEFB  1001B
0109    0A          DEFB  AH
010A
```

DEFS是保留一定数量的字节用的伪操作。在许多情形里需要留出一块存储区而不将有意义的数据填入其中，如象在分配I/O缓冲区和工作存储区时那样。DEFS的作用是使汇编程序指令计数器增加一数值，这个数值等于要留出来的字节数。在汇编之后由装配程序将汇编好的目标模块装入存储器时，DEFS分配的存储块不受影响，且在装入之前存储区内将保持无意义的数据。保留存储器的另一种方法是应用ORG伪操作。下面两个语句从存储单元1234H开始保留22H个字节。

```
1234H   BUFFER   EQU   $
1234H            DEFS  22H
1256H   NEXT1   LDD
```

↓

```
1234H   BUFFER   EQU   $
1256H            ORG   $+22H
1256H   NEXT1    LDD
```

现在讨论最后一个假操作TXT，它同DEFB和DEFW相似，产生供程序用的数据。在这种情况下数据是ASCII文本数据。ASCII码用于大多数输入/输出设备，示于附录E。字母、数字和特殊符号在输出给输入/输出设备打印、显示或穿孔之前应编成ASCII码格式。伪操作TXT对变元字符串中的每一文本字符产生一个ASCII字符。操作数字符串以任一字符打头，并以该字符结束。最好用特殊字符作为界符：

```
0100    43555253    TXT    $CURSE YOU RED BARON$
0104    4520594F
010B    55205245
010C    44204241
0120    524F4E00
0124
```

（$为界符）

上述伪操作是最常见的几种，在以后的例子中将用到它们。但Z-80微型计算机软件实际用的伪操作有所不同，读者应参考制造厂的文献资料，查得所用的记忆符和格式。

汇编原理

程序写好以后，实际的汇编是十分简单易行的。源语句用键盘输入，源程序行的复本记录在某种类型的输入/输出介质，如纸带、磁带或软盘。在很多情形里是用称为编辑程序的应用程序将键盘输入传送到存储介质。程序复制在介质上以后，将汇编程序装入微型计算机（如果用驻留汇编程序的话），或装入宿主计算机（如果用交叉汇编程序的话）。然后汇编程序扫描第一遍，从存储介质读出源程序。如用纸带或盒式磁带作存储介质，则可能要用手工方法对这些介质重新定位，使之处于记录的源程序的起始点上。在用其它的介质时，系统将自动从输入介质的起始点重新启动。接着，汇编程序执行第二遍扫描，产生图9-3所示的清单和目标模块。目标模块实质上是具有特殊装配程序格式的机器语言代码。目标模块可以存在纸带、磁带或软盘。现在汇编程序输出的目标模块可用装配程序装入微型计算机的存储器中，以备执行。装配程序也是一种应用程序。

如前所述，几乎没有一个程序是一次就通过的，而往往要多次地重新汇编、装入和执行，才能得到最终的形式。每一次迭代（在某些系统中这种迭代可多达几十次），汇编程序将大大简化编码过程。

［薛家政译　沈志同校］

第十章　数据传送——取数、数据块传送和交换指令组

这一章讨论任何计算机系统中最基本的操作之一，即CPU寄存器和外存间或外存两个区域之间的数据传送。一次可传送8位或16位。传送包括将数据从一个存储单元传到另一存储单元，将源存储单元的内容复制入目标存储单元，或者是交换两存储单元的内容。某些传送包括从作堆栈用的一部分外存取出和存储数据。最复杂的传送中，一条指令可传送多达64K字节。

八 位 传 送

八位取数指令组可通过不同寻址方式将数据从CPU寄存器传到存储器或相反从存储器传到CPU寄存器。向或从A寄存器传送数据是该指令组中的特殊子集。寄存器A被给于优先使用的地位，因为它在8080和8008中是用作算术、逻辑和移位操作的基本寄存器，Z-80沿用了这种做法。

用LD R，R'或LD R，N指令可以分别将CPU寄存器内容或立即数取入CPU的通用寄存器。下面的代码分别将0到4各数取入寄存器A、B、C、D和E，然后用LD R，R'指令颠倒存放的顺序（从4到0）。

```
LD A, 0        取0
LD B, 1        取1
LD C, 2        取2
LD D, 3        取3
LD E, 4        取4
LD H, A        存A
LD L, B        存B
LD A, E        E到A
LD B, D        D到B
LD D, L        B到D
LD E, H        A到E
```

当数据要从存储器传送到CPU寄存器时，可采用几种方法来实现。这些方法同样地适用于从CPU寄存器往回向存储器传送数据，因而说明传送操作的最好方法，是说明数据如何由一存储块传往另一存储块。显然，实现这类传送的最简单方法是用数据块传送指令，但这类指令我们留待下一章来讨论。要一下将八位数据从存储器送到CPU寄存器或反之由寄存器送到存储器，通用的方法是：

1. 用任何一个CPU寄存器和HL作指针（寄存器间接方式）；
2. 用变址寻址和任一CPU寄存器
3. 用直接寻址（扩充型）和A寄存器
4. 用BC或DE寄存器对作间接寻址和A寄存器。

我们将依次讨论上述四种方法，每种方法都举出一种简短的程序以说明八位的数据是如何向或从CPU寄存器传送的。

用HL寄存器的八位数据传送

下列程序是把四个变量从标有VAR1、VAR2、VAR3和VAR4的存储单元取到A、B、C和D寄存器。首先用一条16位取数指令将四字节数据块的起始地址送入作指针用的寄存器对HL。每次取入一个变量后，HL寄存器加1，指向数据块的下一个字节。

```
LD    HL, START    指向起始字节
LD    A, (HL)      取入第一个变量VAR1
INC   HL           指向START+1
LD    B, (HL)      取第二个变量VAR2
INC   HL           指向START+2
LD    C, (HL)      取VAR3
INC   HL           指向START+3
```

```
        LD   D, (HL)      取VAR4
START   EQV  $            把起始字节的地址赋给START
VAR1    DEFS 1
VAR2    DEFS 1            这些变量是在程序执行过程中的某个时刻填
VAR3    DEFS 1              入数值的。
VAR4    DEFS 1
```

注意，上述方法是很适用的，因为四个变量一个接一个地位于同一存储块内。如果变量是随机地放在存储器中，则如下列短程序所示，要少许多做一点工作。此短程序是把A、B、C和D的内容存入标号为STOR1、STOR2、STOR3和STOR4的四个存储单元。每次新寄存器的内容被存入存储器后，必须将新的地址装入HL寄存器对。这是因为不能用加1或减1的简单做法。虽然在Z-80中有许多其它方法解决此问题，但为8008编写的程序必须用这种方法来访问随机数据，因为8008中只有HL寄存器对可用作指针。

```
        LD   HL, STOR1    STOR1的地址取入HL
        LD   (HL), A      存A
        LD   HL, STOR2    STOR2的地址取入HL
        LD   (HL), B      存B
        LD   HL, STOR3    STOR3的地址取入HL
        LD   (HL), C      存C
        LD   HL, STOR4    STOR4的地址取入HL
        LD   (HL), D      存D
        ↓
STOR1   DEFB0
        ↓
STOR2   DEFB0             这些变量一开始置为0，然后填入A—D
        ↓
STOR3   DEFB0
        ↓
STOR4   DEFB0
```

利用变址寄存器的八位传送

Z-80中的变址寄存器IX和IY类似于HL。每一变址寄存器同样地是一数据指针，但有一重要的差别，即有效地址等于8位的位移量与变址寄存器内容之和。这就是说，在每条指令内，都可以将一个位移量加到指针上，从而可在256字节的存储块内存取数据。如图10—1所示，此存储块自LOCN—128（变址寄存器指示值减128字节）开始，以LOCN+127结束。

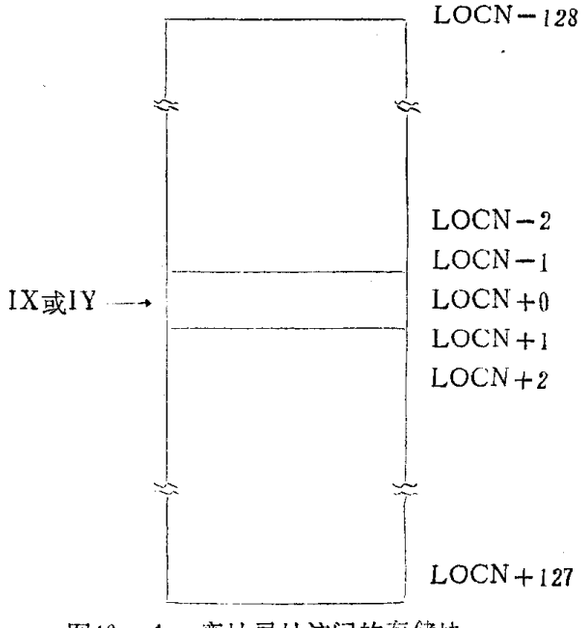

图10—1　变址寻址访问的存储块

假定要把寄存器A、B、C、和D的内容分别存入存储单元BLOCK—4、BLOCK、BLOCK+4和BLOCK+8。下列指令可以做这件事：

```
LD IX, BLOCK        指针指向BLOCK
LD (IX—4), A        A存入BLOCK—4
```

```
LD(IX+0), B      B存入BLOCK
LD(IX+4), C      C存入BLOCK+4
LD(IX+8), D      D存入BLOCK+8
```

指令第三字节中的位移量分别为－4、0、4和8。用这种方法将数据存入256字节的数据块，比起上例中用HL寄存器对作指针来，要有效得多。难道不是这样吗？现让我们比较两个程序的长度和占用时间。第一个程序（用HL寄存器）有四条三字节指令（LD HL, STORX）和四条单字节指令（LD(HL), D），共十六字节和占用17微秒。上述程序用五条三字节指令，共15字节和22.5微秒！看来第一种方法较快，但占用存储单元略多一点。这里仅举一例说明，当涉及到程序效率时，应如何比较执行速度和占用的存储空间。

如果是用IX寄存器而不是用IY寄存器，指令格式实际上是一样的，只是要用'IY'代替'IX'。面向变址寄存器的指令，可以象在下列程序中那样有效地用来传送数据。这个程序是把存在BLK2到BLK2+2内的三个字节传送到BLK1到BLK1+2中。传送是以逆方向进行，从BLK2+2和BLK1+2开始。IX保存源指针而IY保存目标指针。两变址寄存器分别用DEC IX和DEC IY指令减1。

```
        LD    IX, BLK2+2     置入源起始地址
        LD    IY, BLK1+2     置入目标起始地址
NXT1    LD    B, (IX)        同(IX+0)
        LD    (IY), B        同(IY+0)
        DEC   IX             指向下一源字节
        DEC   IY             指向下一目标字节
NXT2    LD    B, (IX)        取下一字节
        LD    (IY), B
        DEC   IX
        DEC   IY
NXT3    LD    B, (IX)        取下一字节
        LD    (IY), B
```

用上述代码进行存储器存数是低效的，因为同样的基本操作要重复许多次。在NXT1、NXT2和NXT3处的传送几乎相同。如要传送100个字节，则自然地要重复操作100次，这是荒唐的。实现重复操作最有效的途径是循环执行同一组指令，循环次数N按需要决定。下列程序是做这件事的，其中IX和IY仍用作源和目标指针，且将100个字节从BLK2传送到BLK1。起始的计数值N=100保存在C寄存器中，并逐次地减小到0。只要N在100～1之内，Z标志就不置1，就执行条件转移JP NZ, LOOP，如此LOOP循环执行100次。每经一次循环，IX和IY加1，指向存储块中的下一存储单元。

```
                LD    IX, BLK2     取源存储块起始地址
仅执行一次      LD    IY, BLK1     取目标存储块起始地址
                LD    C, 100       置N=100
LOOP            LD    B, (IX)      取字节
                LD    (IY), B      存字节
100次
                INC   IX           }指针加1
                INC   IY           }
                DEC   C            N－1
                JP    NZ, LOOP     如未做完(Z≠1)则转LOOP
DONE                               做完
```

循环中前五次和最后五次迭代中IX、IY、C和B内的值示于图10—2。

IX	IY	C	B	
BLK_2	BLK_1	100	—	置初值
BLK_2+1	BLK_1+1	99	第1字节	第一次迭代（之后）
BLK_2+2	BLK_1+2	98	第2字节	第二次迭代
BLK_2+3	BLK_1+3	97	第3字节	第三次迭代
BLK_2+4	BLK_1+4	96	第4字节	第四次迭代
BLK_2+5	BLK_1+5	95	第5字节	第五次迭代
	≈			≈
BLK_2+95	BLK_1+95	4	第96字节	第九十六次迭代
BLK_2+96	BLK_1+96	3	第97字节	·
BLK_2+97	BLK_1+97	2	第98字节	·
BLK_2+98	BLK_1+98	1	第99字节	·
BLK_2+99	BLK_1+99	0	第100字节	第一百次迭代

图10-2 变址寻址例子

用A寄存器和扩充寻址的八位传送

可以用扩充寻址方式将数据取入寄存器A或从它存入存储器。在这种情形里，指令本身规定了地址，不用对任何指针或变址寄存器置值就可以做到完全随机的寻址。这条指令可能是最常用于通过A寄存器将八位的数据传送到CPU寄存器或存CPU寄存器的内容于存储器的指令，如图10—3所示。在这种情形里，A寄存器是所有其它CPU寄存器的通路。

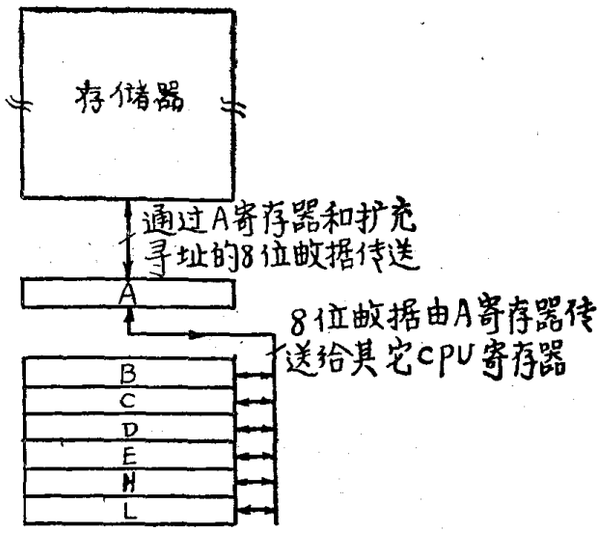

图10—3 用于随机寻址的A寄存器

下列程序首先用这种寻址方式将寄存器内容存入STRA、STRB、STRC和STRD，然后将VAR_1、VAR_2、VAR3和VAR4取入A、B、C和D。

```
LD(STRA), A          存A
LD A, B
LD(STRB), A          存B
LD A, C
LD(STRC), A          存C
LDA,D
LD(STRD),A           存D
LD A,(VAR4)          取VAR4，以送D
LD D,A
LD A,(VAR3)          取VAR3送C
LD C,A
LD A,(VAR2)          取VAR2送B
LD B,A
LD A,(VAR1)          取VAR1送A
```

用A寄存器和BC或DE寄存器间接寻址的8位数据传送

LD A,(DE)、LD A,(BC)、LD (DE), A和LD (BC), A四条指令用BC或DE作指针，就象上述数据传送中用HL作指针一样。只要待取出的数据是连续地存在数据块或表中，这种寻址方式就是一种行之有效的方式。前面曾举例说明用变址寄存器来将数据从这一块传送到另一块。下面的程序做同样的事。可以看出，其作用实际上是相同的。其中BC指向源数据块，DE指向目标数据块，L含有计数值。

```
        LD BC,BLK2      源数据块起点
        LD DE,BLK1      目标数据块起点
        LD L, 100       置N=100
AGAIN   LD A, (BC)      取字节
        LD (DE), A      存字节
        INC BC          BC 内容加1
        INC DE          DE 内容加1
        DEC L           N－1
        JP NZ, AGAIN    如未做完（Z≠1）则转
DONE      ↓
```

十六位数据传送

上述数据传送是一次传送八位。但Z-80有许多指令是用来一次传送二字节或16位数据的。以这种宽度传送的数据在寄存器对BC、DE和HL、寄存器SP、IX和IY和存储器之间被取或被存。十六位数据操作有：

1. 立即取入BC、DE、HL、SP、IX、IY

2. 从存储器传到BC、DE、HL、SP、IX或IY，或者相反。

3. 从HL、IX或IY传送到SP

4 将BC、DE、HL、AF、IX或IY压入堆栈，或自堆栈弹出。

当然，寄存器对、SP或变址寄存器的取数，包括了取存储器地址。十六位可以存放Z-80机所有的64K外存地址。这一组指令是专为处理与地址有关的数据而设置的，但都可以用于取和存非地址操作数，如16位的双精度值或ASCII字符数据。

立即取十六位数据

本章的前面已介绍了立即取数的许多例子。一般是将包含待处理数据的数据块起始地址取入BC、DE、HL、IX和IY，例如：

```
        LD   BC, DATA1      取数据1的地址
        LD   DE, DATA2      取数据2的地址
        LD   HL,DATA3       取数据3的地址
        LD   IX, DATA4      取数据4的地址
        LD   IY, DATA5      取数据5的地址
          )
          ↓
DATA1   DEFS 100            100字节的数据块
DATA2   DEFS  50            50字节的数据块
DATA3   DEFS  20            20字节的数据块
DATA4   DEFS  60            60字节的数据块
DATA5   DEFS 100            100字节的数据块
```

堆栈指针寄存器SP几乎总是指示被分配作堆栈用的存储区，而不是指示预定的数据块。SP通过LDSP,NN指令预置到某一地址值——栈顶。因为SP总是指向堆栈中最后用的那个单元，而且在完成数据存储之前减1，所以应将与堆栈第一个存储单元地址加1相应的地址值装入SP。例如，若堆栈存储区占用自地址3FFFH到地址3F00内的100H个字节，则SP将象下面那样被置初值：

```
0100                LD SP, 4000H          取栈顶
                       或
                    LD SP, TOPS
                       ↓
3F00                DEFS 100H             定义256字节的
4000    TOPS        EQU $                 堆栈区
```

随后进行压入堆栈的操作时（因在这之前没有数据存入堆栈，所以没有弹出操作），每次存储前将使SP减1。第一个数据字节存在3FFF，第二个存在3FFE，余者依次类推。

向和从存储器传送十六位数据

用这一组指令可以将数据从存储器取入BC、DE、HL、SP、IX、IY或SP寄存器，或相反地存入存储器。作为例子，假定BC、DE和HL寄存器中应取入三个存储块的地址，但其内容应加以保存和恢复，以供后用。作为存入堆栈的另一种方法（本章稍后面要谈到），可用下列指令来存三个寄存器对的内容：

```
                    LD  (SAVB), BC        存BC
                    LD  (SAVD), DE        存DE
                    LD  (SAVH), HL        存HL
                       )
                       ↓
SAVB        DEFS  2                       存BC的存储单元
SAVD        DEFS  2                       存DE的存储单元
SAVH        DEFS  2                       存HL的存储单元
```

注意，为每一寄存器对保留的存储单元应为二个字节。以后当要将起始值装入寄存器对时，可以用下列指令：

```
LD BC,(SAVB)        恢复BC
LD DE,(SAVD)        恢复DE
LD HL,(SAVL)        恢复HL
```

如在许多指令组中那样，汇编语言变元的格式极为重要。在下面的代码中，LD HL, SAVL是取SAVL的地址（1000H）而LD HL (SAVL) 是取SAVL的内容：

```
                    LD    HL, SAVL        取1000H
                       ↓
                    LD    HL, (SAVL)      取123AH
                       ↓
1000    SAVL        DEFW  1234H           1000H的内容是1234H
```

十六位数据向堆栈传送

Z-80可以将数据从HL、IX和IY寄存器传到栈指针寄存器，但不能以相反方向传送。这类传送的例子是：

```
LD SP, HL        HL 存堆栈
    )
    ↓
LD SP, IX        IX 存堆栈
    )
    ↓
LD SP, IY        IY 存堆栈
```

十六位栈操作

这一小节的标题是不贴切的，因为所有堆栈操作都包含了一次传送16位或2字节数据的操作。Z-80不象其它微型机那样可把八位数据压入堆栈，或从中弹出。因此，当只有一个寄存器要暂存到堆栈时，会付出稍多

```
LD      SP, 1000H
  ↓
PUSH    AF
PUSH    BC
PUSH    DE
PUSH    HL
PUSH    IX
PUSH    IY
```

SP预置到1000H

A压入0FFFH，F压入0FFEH
B压入0FFDH，C压入0FFCH
D压入0FFBH，E压入0FFAH
H压入0FF9H，L压入0FF8H
IX的15-8位压入0FF7H，7-0位压入0FF6H
IY的15-8位压入0FF5H，7-0位压入0FF4H

就象读者会预料到的那样，在堆栈操作中，F标志寄存器被当作8位的低位寄存器来处理。

当数据从堆栈弹出时，过程恰好与此相反。首先，低位字节从栈顶拉出，放入F、C、E、L、IX_{low}或IY_{low}寄存器，然后SP加1，高位字节放入寄存器对的高位字节或IX或IY寄存器对的高位字节。

栈存储器的用途有：

1. 在处理中断时保存现场
2. 暂时存储CPU寄存器内容

```
PUSH    BC
PUSH    IY
PUSH    DE
PUSH    IX
POP     DE
POP     IX
POP     BC
POP     IY
```

现在堆栈中有BC
现在堆栈中有BC、IY
现在堆栈中有BC、IY、DE
现堆栈中有BC、IY、DE和IX
IX入DE
DE入IX
IY入BC
BC入IY

堆栈寄存器还可以用来实现数据串处理，虽然在这样做时，对待堆栈指针应小心行事。作为这方面的一个例子，假定存储单元1777H到1700H内有一串ASCII字符，且第一个字符放在1700H单元，最后一个放在1777H单元。（用增量指令易于做到以这种方式存储数据。）下列代码逐一处理这些字符，这里假定在处理过程中堆栈不再作任何一点代价。但这并非严重的缺点。在Z-80中BC、DE、HL、AF寄存器对和IX、IY寄存器内容可以压入存储器堆栈，或从其中弹出。每次压入堆栈时，寄存器对高位字节中的数据放入（栈顶－1）单元，而低位字节中的数据放入（栈顶－2）单元。在每一字节压入前，SP寄存器减1。下面说明压入寄存器对IX和IY的内容时，堆栈的工作情况：

3. 作为在CPU寄存器之间传送数据的一种途径
4. 供子程序用

第一和第四项应用，将留待后面讨论。另两种应用则是比较显而易见的。在任何时候，执行一条压入指令，数据就可以从寄存器对IX或IY之一存入堆栈。其后可用弹出指令取出数据。并没有规定弹出的数据非重新存入同样的寄存器对不可，因而堆栈可方便地用于寄存器间传送数据，就如下例（交换BC和IY及DE和IX寄存器内容）所示。