8080/8085 软件设计

C·A·泰特斯 等著
张梅岗 桦 纳 译

【下 册】

封面设计：邵　新

科技新书目：109-148・统一书号：15045・总3109-无6349・定价：2.65元

8080/8085软件设计

下　册

C. A. 泰特斯　等著

张梅岗　华　纳　译

符　明　薛家政　校

人民邮电出版社

8080/8085 Software Design by
Christopher A·Titus,David G·Larsen,and
Jonathan A·Titus
Howard W·Sams & CO, INC.

内 容 提 要

本书通俗地讲述 8080/8085 微型计算机软件设计，分上下两册，下册主要讲述系统程序设计方法，内容包括：中断及应用、检索、排序、命令译码程序、系统监控程序、断点和调试程序。书中给出了许多实用性、通用性强的程序例子。通过例子，总结了 8080/8085 软件设计的方法和技巧。

本书可供从事计算机应用的工程技术人员、大专院校有关师生阅读。

8080/8085 软件设计

下 册

C.A.泰特斯 等著

张梅岗 华 纳 译

符 明 薛家政 校

责任编辑：徐修存

*

人民邮电出版社出版

北京东长安街27号

北京印刷一厂印刷

新华书店北京发行所发行

各地新华书店经售

*

开本：787×1092 1/32　　1986年5月第一版

印张：$14^{24}/_{32}$ 页数：236　　1986年5月北京第一次印刷

字数：335千字　　印数：1—8,000册

统一书号：15045·总3109-无6349

定价：2.65元

序　言

随着低成本的微型计算机，例如，Commodore PET 和Radio Shack TRS-80 的出现，许多人将要开始应用微型计算机。这两种微型计算机，虽然都具有汇编语言编制程序的能力，但是，人们通常用 BASIC 语言给它们编写程序。对于这两种微型计算机，由于 BASIC 程序的规模和复杂性日趋增加，因此，越来越多的用户开始认识到 BASIC 解释程序的一些局限性。实际上，有些任务，用汇编语言比用 BASIC 语言更容易完成。

PET 和 TRS-80 微型计算机的一些用户发现 BASIC 的解释程序的能力对他们而言是足够强的；但是，他们还发现；必须用汇编语言编写所用的子程序，BASIC 的解释程序才能和已经连接的微型计算机的外部设备通信。

值得指出，汇编语言程序设计将会长期被人们采用，它仍然是给 8080、Z-80、6800 和 6502 各种微型计算机进行程序设计的有效方法。实际上，汇编语言程序可能解决的问题比 BASIC 语言程序可能解决的问题还多！正是这个原因，我们编了«8080/8085 的软件设计»一书的下册。

在«8080/8085 的软件设计»上册的最后一章，我们讨论了许多输入/输出设备以及它们的相应的软件“驱动器”。那一章所用到的输入/输出设备包括 ASCII 键盘，16 键硬件编码的键盘，扫描式键盘，锁存式发光二极管(LED)显示器和多路转换的 LED 显示器。

本书是一本 8080 的基本指令系统的参考书，书中还给出了算术操作、数制转换和输入/输出程序设计等许多程序实例。上册书中的许多程序例子和有关论述，在下册，我们还要参考。

为了把数据串行化并传送给外部设备，在本书第一章，我们将要阐述怎样使用通用同步接收器/发送器（UART）、通用异步接收器/发送器(USART)与 8080 微型计算机本身进行异步通信的问题。就是这两种器件也可用来从外部设备接收串行数据。读者会看到，利用程序指令来完成数据传送和接收任务，硬件的成本将减至最少。我们还要讨论 8085 所用的两条指令。SIM 和 RIM，8080 指令系统中并没有这两条指令。我们始终坚持给出软件程序实例所需要的接口电子线路，以使这些程序实例能正确地操作。

下面两章讨论中断问题。第二章概括地讨论中断-询问的，向量的，和优先权三种中断方式。这一章所包括的内容是中断服务子程序，以及这些子程序所必须具有的一般结构。第三章讨论中断的应用，其中包括实时时钟，日时时钟和可编程序的时钟。这一章还包括中断驱动键盘（硬件编码的和扫描的）和多路转换 LED 显示器所用的程序和子程序。第三章的最后一节，描述了优先权中断控制器 8214，并且讨论了驱动该集成电路所需要的软件。

本书以下四章讨论数据结构，以及如何对它们进行存取。我们用了简短的一章，定义“数据结构”以及数据结构的一些普通术语。继这一章之后的第五章阐述的内容包括许多子程序，可以用这些程序在表格或表里检索特定的数字值。所列出的子程序可用来检索 8 位的，16 位的和 24 位的数。这一章的最后一节讨论字母数字串的检索方法。我们给出了许多子程序，用于表格中检索存储特定的 ZIP 代码的地址。

第六章讨论“排序”问题，还涉及到数字值和字母数字串。我们采用两种排序法(插入排序法和交换排序法)，讨论某些具体排序问题的最佳情况/最坏情况的时间。

第七章描述对照表问题。这一章给出的程序例子所涉及的问题包括角的正弦的产生，(很快地)以及把字母数字字符在纸带上穿孔。使用 90 个节点（项目）的正弦对照表来产生 0°～360°之间的任何一个角的正弦值，(按 1°递增)，也可以决定正弦值的符号。

不同形式的命令译码程序也许可用于百分之九十的微型计算机程序。因此，我们在本书用了一章的篇幅来讨论命令译码程序，其中包括许许多多的程序例子。这些命令译码程序实例阐述了固定字长和可变字长的命令译码问题。

本书的最后两章，即第九章和第十章讨论系统控制程序和调试程序。生产厂提供的微型计算机，有许多机种就已经编有监控程序或调试程序；例如：荒原 (Heath)公司的 H-8；英特尔(Intel)公司的 SBC 80/10 、SDK-80 和 SDK- 85；国家半导体公司 (National semiconductor corporation)的 BLC 80/10；洛克韦尔国际公司(Rockwell Internatimal)的 BLC 80/10，MOS 技术公司 (MOS Technology Corporation) 的 KIM；和洛克韦尔国际公司的 AIM-65。如果读者兴致勃勃为你现在正在开发的系统编写系统监控程序或系统调试程序，那么，这两章会给你许多启示。这两章的内容包括许多程序和子程序，读者可以把它们用作为系统监控程序或调试程序。

我们编写这本书时有一个明确的目标：对程序实例如何工作给出详细的说明解释。我们不说：“给你程序，你自己去搞清楚它是怎样工作的。”如果采用这种方法，你就学不到什么。相反，我们把程序“开发”出来，从能完成这个具体任务的最简

单的指令序列开始。这些程序常常带有局限性；如果它们确实有局限性，我们就要分别给这些程序添加一些指令，8080 微型计算机才能更好地完成更普通的任务，或者特定的任务。因此，我们只有学习了前面的例子，才能对某个问题设计出最好的解决方案。

关于读者的计算机执行本书给出的程序和子程序例子，是否需要特定的外部设备的问题请读者不必担心。我们在本书给出的程序例所用到的硬件和软件并不局限于某个制造厂商的硬件特性。书中的程序和子程序例子对于下列制造厂家的 8080 微型计算机都同样适合执行：PERTEC(MITS)，英特尔公司，国家半导体公司，荒原公司，控制逻辑公司(Control Logic)和 E&L 仪器公司(E&L Instruments)。当然，这些厂家的 8080 微型计算机系统可能各有不同，这正如系统与系统的外部设备的地址各有不同一样。

在这本书里，我们继续使用常驻编辑程序/汇编程序(TE-A)来产生每行一个字节的程序表。用这种程序表格式，对于许多读者来说，比较容易地、准确地理解多字节指令的地址字节和数据字节应该存储在存储器的什么单元。我们还从过去的经验知道，许多人仍用人工汇编程序，这是因为他们的计算机没有足够的存储器来存储编辑程序/汇编程序，或者他们不能找到可以在他们的系统上工作的编辑程序/汇编程序。如果读者使用我们提供的程序和子程序表，人工汇编是很容易的，因为我们为多字节指令的地址字节和数据字节留有“剩余房间”。在这本书中，我们还讨论了八进制数和十六进制数。我们先列出八进制数，跟在它后面的括号内是十六进制数；例如：125(55)。

我们怀着高度热情向读者推荐这本书，因为它为 8080 基

本指令系统提供了一本很好的参考书。读完本书会对进行8080/8085的软件设计有所帮助。

C.A. 泰特斯
D.G. 拉森
J.A. 泰特斯

目　录

第 四 章

数据结构

线性表-顺序分配-连接分配-循环表

第 五 章

检索

单精度表(8位)-双精度表(16位)-三精度表(24位)-检索子程序的共同特征-ASCII字符串的检索-测试邮政编码存入存储器-检查邮政编码的检索程序-程序的最后一个错误

第 六 章

排序

数字值的排序-字母-数字串的排序

第 七 章

查表

使用更精确的正弦表-纸带字母穿孔程序

第 八 章

命令译码程序

单字母命令译码程序-以表为基础的单字母命令译码程序-用两个表的单字母命令译码程序-多字符命令译码程序-可变字长的命令译码程序

第 九 章

系统监控程序

硬接线前面板--一般系统监控程序的特点-简单系统监控程序-用于非 ASCII 键盘的系统监控程序-用多路转换显示器的系统监控程序-用系统监控程序连接程序

第 十 章

断点和调试程序

断点-断点指令-断点的人工设置和清除－断点的自动设置与清除-保存和打印寄存器的内容-断点操作-寄存器内容非破坏性打印－给调试程序添加一个“继续”命令-单步一次执行一条指令-单步通过控制转移指令-简单的调试程序-关于调试程序的最后几点意见

程序例目录

第一章

第二章

第 三 章

第 五 章

第 六 章

第 七 章

第 八 章

第 九 章

第 十 章

第一章　异步串行通信

如果微型计算机只配备了一个 16 只键或 20 只键的键盘和一些七段显示器，那么，要把一个包含 10 0条或 200 条指令的程序用这种键盘和这些显示器送入微型计算机，则需要花费很长的时间。如果需要 100 或 200 个存储单元来存储的程序表，那么把写入每个存储单元的内容一个一个地进行检查，同样，需要花费很长的时间。为了提高微型计算机的效率和能力，经常把显示终端 CRT（阴极射线管）和电传打字机与微型计算机系统连接。

有两种基本方法，可以用来把外部通信设备与 8080 微型计算机系统连接起来。我们要介绍的第一种方法，它需要大量硬件，而需要很少的软件；第二种方法则需要很少的硬件，而需要很多的软件指令。我们将要对这两种方法进行比较，说明它们各自的优点和缺点。

硬件方法

通用异步接收器/发送器(UART)是一种很复杂的器件，可以用它把微型计算机与电传打字机连接起来，或者与 CRT 连接起来。通用同步/异步-接收器/发送器(USART)是一种比较先进的器件，在以 8080 微型计算机为基础的通信领域里正在得到大量的应用。

通用异步接收器/发送器(UART)是一种有 40 条引线的集成电路封装组件。一种普通的 UART 器件的引线结构如图 1-1 所示。UART 包含完全独立的异步串行接收器和异步串行发送器。这就是说，它可以用一种速度从 CRT的键盘接收数据，同时可以用另一种速度向电传打字机发送不同的数据，UART 的其它一些特性如表 1-1 所示。与之相应的 CMOS (互补金属—氧化物—半导体)UART 用单电源工作，而且功耗较低。

UART 的特性

正如在表 1-1 中所看到的一样，UART可以发送和接收 5、6、7 或 8 位信息。对于用这些字长工作的 UART 来说，为了获得所希望的字长，必须给它编好程序。应该这样编程序：把逻辑 1 或逻辑 0 加在 UART 的引线 37 和 38 上，如表 1-2 所示。这样编程序可以决定 UART 的接收器和发送器的字长。

我们也可以将 UART 的两条引线专用于奇偶校验的产生和选择。如果将使用奇偶校验，则可以选用偶校验或奇校验。为了从发送的数据字或接收的数据字去掉校验位，UART 的引线 35 必须置于逻辑 1。如果这条引线上的电平保持为逻辑 0，那么，奇偶校验位(由引线 39 的逻辑状态所决定的)可用于 UART 的发送器和接收器，如表 1-3 所示。

我们也可以为 UART 的接收器/发送器的停止位的位数编程序。这是由加给引线 36 的逻辑电平来控制的。如果把引线 36 连接到逻辑 0 电平，那么，将发送 1 位停止位；接收器将也要求只接受 1 位停止位。如果把逻辑 1 加给引线 36，那么，发

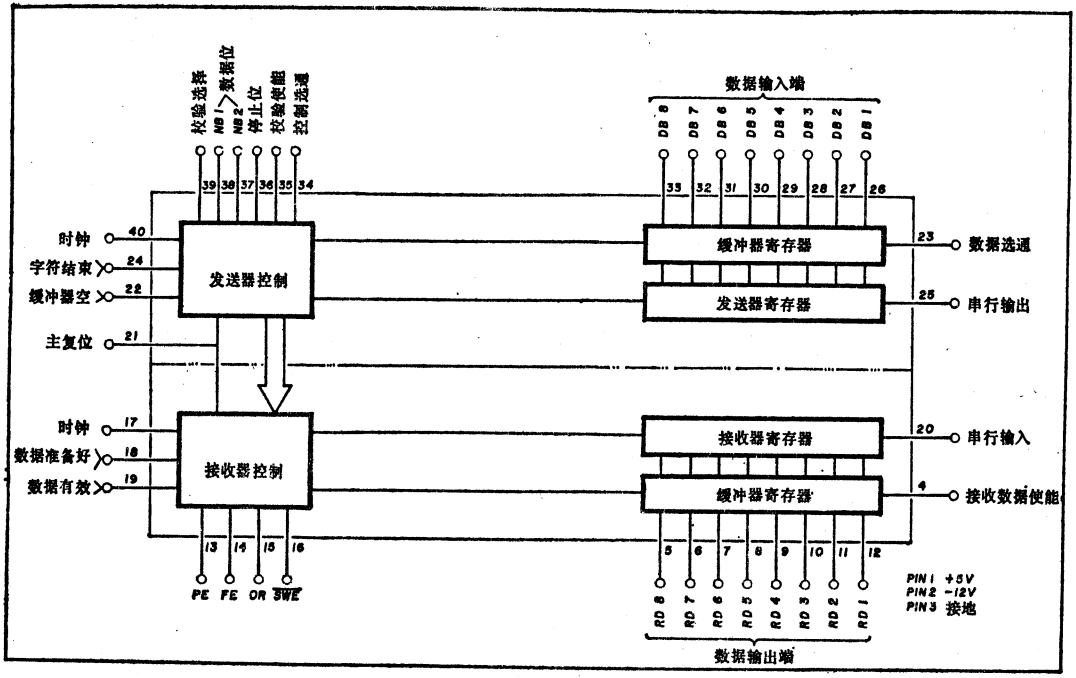


图 1-1 40 条引线 UART 的引线分配

表 1-1 UART 的性能特点

- 独立的接收器和发送器；
- 接收和发送 5 、 6 、 7 或 8 位数据字；
- 可以选择偶校验、奇校验或无校验；
- 用直流电源工作，速度至少为20,000比特/秒；
- 接收器有 3 个错误标识位；
- 三状态输出。

表 1-2 对UART的字长编程序

引 线 37	引 线 38	字长(每个字的位数)
0	0	5
0	1	6
1	0	7
1	1	8

表 1-3 给 UART 的奇偶校验编程序

引 线 35	引 线 39	被选择的奇偶校验
0	0	奇 校 验
0	1	偶 校 验
1	0	无 校 验
1	1	无 校 验

送器将发送两位停止位，接收器要求在它接收的字中有两位停止位。

关于 UART 的字长，奇偶校验和停止位的数目编程序的问

题，应该指出重要的一点：**只有引线34被置成逻辑1，这些逻辑电平才将被装入UART的内部保持寄存器。**

该引线可以一直与逻辑1电平连接，或者用一个输出脉冲来控制，数据总线上的内容才能装入保持寄存器。这个脉冲的最小持续时间必须有300 ns。

最后介绍与UART的接收器相关的三个错误标识位。这三个标识位分别表示奇偶错(PE，引线13)，帧错(FE，引线14)，和溢出错(OE，引线15)。当被接收的数据字的奇偶校验与对UART编程序的奇偶校验不一致时，出现奇偶校验错误。如果出现了奇偶校验错误，则该标识位置到逻辑1。如果接收的字符不包含一位有效停止位，或者停止位的位数不正确，那么，帧错误出现，FE标识位置到逻辑1，予以表示。如果接收最后数据字后，**接收的数据有效标识(引线19)还没有被复位，那么，溢出标识(OR)置成**逻辑1。换句话说，如果在接收器的一个数据字溢出到另一个数据字，那么，溢出标识位置成逻辑1。这些标识位由UART集成电路器件上的三状态缓冲器进行缓冲，所以这些标识位引线可以直接与8080微型计算机的数据总线连接。当引线16，**状态字使能**($\overline{\text{SWE}}$)引线被置成逻辑0时，这些标识位被选通，进入数据总线。可以通过 $\overline{\text{IN}}$(I/OR) 或 $\overline{\text{MEMR}}$ 与该器件的一个地址适当地组合，形成脉冲加到引线16。

串行数据格式

一般地说，我们采用只有四条连线的接口，就能把大多数电传打字机和CRT与UART或USART连接起来。怎样在四

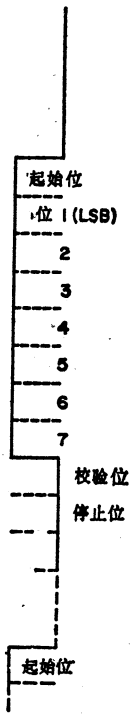


图 1-2 7 位字符加 1 位校验位的串行数据流

条线上发送或接收 5、6、7 或 8 位的字符呢？实际上，UART 用两条线把数据发送给外部设备，用两条线从外部设备接收数据。为了使数据在两条线上传送，一次一位发送或接收数据字。这种方法就叫做串行通信。7 位字符的串行数据流如图 1-2 所示。

这种串行的通信方法为 UART 的发送器和外部设备(电传打字机或CRT)的发送器使用。UART 和电传打字机或 CRT 也一定能够接收串行数据。

因为 UART 是一种复杂的器件，所以，它实际上能发送和接收串行字符。这些串行字符被称为异步串行字符；这就是说，字符发送或接收与时钟信号无关。如果字符是以同步串行的方式接收或者发送，那么，用与时钟信号特定的时间关系发送和接收字符。

硬件 UART 的软件

读者通过学习《8080 / 8085 的软件设计》的上册给出的许多例子可知道：当电传打字机用接口器件 UART 与 8080 微型计算机连接时，把字符发送给电传打字机，或者从电传打字机接收字符，这是很容易的。在本书的上册第三章中，我们曾假设用 UART 把电传打字机和 8080 微型计算机连接起来。为了把

字符发送给异步串行的外部设备， 8080 必须只把 A 寄存器的内容输出给 UART（假设它是累加器输入/输出设备）。

8080 微型计算机的双向数据总线直接连接到 UART*的发送器的输入引线，当执行正确的 OUT 指令时，UART 的引线 23 必须用负脉冲选通，数据总线上的数据才能选通，进入 UART 的发送器。这种情况一出现，UART 自动地开始把这个数据字符串行传送到外部设备。因此，为了把 ASCII 字符“Z”发送给外部设备，可以执行例 1-1 列出的程序。

例 1-1 把 ASCII“Z”发送给异步串行外部设备

```
.
.
MVIA/把 ASCII 字符“Z”装入 A 寄存器
 132  /(132= 5 A， 16 进制)。
OUT  /把该字符输出给
001  / UART(把数据选通引线选通)。
.
.
```

当然，这条 OUT 指令被执行后， 8080 把另一个要发送给外部设备的字符装入 UART。但是，正如我们在前面已看到的，电传打字机和 CRT需要固定的时间来实际接收和打印这个字符。对电传打字机而言，该时间大约为 100毫秒(ms)。为了防止 8080 发送字符的速度比电传打字机所能接收字符的速度快，UART 产生一个标识位， 它表示 8080 可以把另一个字符输出给 UART。 当该标识位是逻辑 1 时， 整个字符已经发送给了 UART 的发送器寄存器， 然后， 另一个字符可以被装入发送

* 这是假设 UART 的三状态输出与微型计算机的定时要求是兼容的。这种情况并非一成不变。请注意检查特定器件性能参数表的三状态输出的定时要求。

器的缓冲寄存器。第一个字符已经被全部发送以后，再把下一个字符从缓冲寄存器发送给发送器寄存器，然后由该寄存器串行发送。

这就是说，8080 必须监控 UART 的发送器的**缓冲器空标识位**的状态，以决定另一个字符何时可以输出给 UART。这个标识位输出给 UART 集成电路的引线 22。该引线通常处于第三种状态（高阻抗状态），但是，当**状态字使能**引线（引线 16）被置到逻辑 0 时，可以在引线 22 上监控该标识位的状态。这就是说，引线 22 可以直接与 8080 微型计算机的双向数据总线连接。只要执行适当的 IN 指令以及选通控制 $\overline{\text{IN}}$ 信号（$\overline{\text{I/OR}}$）和器件的正确地址，就可以把这个标识位的状态输入到 8080 的 A 寄存器。把一个字符实际输出给 UART，等待该标识位置到逻辑 1 之后才返回的子程序如例 1-2 所示。这种子程序在«8080/8085的软件设计»一书的上册中已经使用过多次。

例 1-2 发送一个字符，等待发送器标识位

```
/把 ASCII 字符"Z"发送给电传打字机
/或 CRT，然后等待发送器标识位。
ATRANS, MVIA   /把 ASCII 字符"Z"装入 A，
        132    /(132，16 进制 5 A)
        OUT    /把"Z"输出给 UART。UART 自
        100    /动地开始发送。
WAIT,   IN     /输入包含 UART 发送器标识位
        101    /的数据字。
        ANI    /只把发送器的标识位保存在
        040    /A 寄存器里(040=20，16 进制)。
        JZ     /这个标识位还是 0，
        WAIT   /所以继续等待它
        0      /置成逻辑 1。
```

RET/然后，从该子程序返回。

UART 可以用来把数据发送给电传打字机（该机的速度是 10 字符/秒），同时，可以用来从 CRT（240 字符/秒）接收数据，这是 UART 的优点之一。这种功能之所以可以实现，是因为 UART 接收和发送信息位的速度是由两种分开的与 TTL 兼容的时钟所决定的。我们必须把 UART 的发送器的时钟加给引线 40，必须把接收器的时钟加给引线 17。

位速率（波特）是经常用来描述异步通信的术语之一；可以按照位速率发送或者接收信息。位速率除以每个字符的位数，就能决定每秒可能接收或发送的字符的数目。因为我们已经知道，UART 可以发送或接收 5、6、7 或 8 位的字符。但是，UART 还必须发送一位起始位，至少一位停止位。因此，对于一个 8 位的字符，可以发送如表 1-4 所示的位数给外部设备。从该表可以看到，UART 发送或接收的最长的字符是 12 位。

表 1-4 UART 可以发送的位数

1 位起始位＋8 位数据字＋1 位停止位。
1 位起始位＋8 位数据字＋2 位停止位。
1 位起始位＋8 位数据字＋1 位奇偶位＋1 位停止位。
1 位起始位＋8 位数据字＋1 位奇偶位＋2 位停止位。

发送（或接收）的数据字不论多长，必须发送（或接收）1 位起始位，必须至少发送（或接收）1 位停止位。读者可以看到，在一个微型计算机系统和另一个微型计算机系统之间，比较字符接收或者发送的速率时，你必须小心。由于 UART 的内部结构，

发送器和接收器的时钟频率必须是所要求的位速率的 16 倍。因此，普通位速率和所需要的时钟频率如表 1-5 所示。

表 1-5 普通位率和所需要的时钟频率

位率(位/秒)	时钟频率(kHz)	时钟周期(μs)
110	1.76	586
150	2.40	417
300	4.80	208
600	9.60	104
1,200	19.2	52.1
2,400	38.4	26.0
4,800	76.8	13.0
9,600	153.6	6.51
19,200	307.2	3.26

UART 的接收器的操作正好与发送器的操作相反。接收器必须能够接收一个字符，一次接收 1 位。所需要的数据位被接收后，必须形成 5、6、7、8 位的并行数据字，以便 8080 把这个接收的字符输入到一个通用寄存器。如果采用累加器输入/输出技术，把 UART 与 8080 微型计算机连接起来，那么，必须把这个字符输入到A寄存器里。如果采用存储器映象输入/输出接口技术，那么，8080 可把这个数据字输入到任何一个通用寄存器里。当引线 4，**接收数据使能**(RDE)引线变为逻辑 0，UART 接收的字符将被置于引线 5～12 的引线上，请参考 UART 的引线排列图 (图 1-1)。如果把逻辑 1 加到引线 4，那么，引线 5～12 将处于第三种状态(高阻抗状态)。

UART 的接收器引线可以直接与 8080 微型计算机的双向数据总线连接。发送器的引线也可以直接与双向数据总线连接。实际上，为了把引线 4 置于逻辑 0，读出接收的字符，

8080 可以执行一条 IN 输入 指令。 这就 是说， 用或门把 $\overline{\text{IN}}$ ($\overline{\text{I/OR}}$)信号和译码的器件地址选通 (低 电平 有效)。 当包含 UART 的被选择的器件地址的一条 IN 指令被执行时， 或门的输出将变成逻辑 0 状态。这时， UART 所接 收的 数据 将被选通，进入 8080 的双向数据总线，然后进入 8080 的A寄存器里。

当 UART 发送一个字符时， 必须给 8080 编程序， 让它等待发送器的缓冲器空标识位变成逻辑 1 之后， 才能发送另一个字符。还必须为 8080 编程序， 监控数据接收有效 标识 (引线 19)， 该标识位与 UART 的接收器连接。 当这个 标识位为逻辑 1 时， 此 UART 已经接收了一个字符， 并且可以 把该字符输入。如果这个标识位为逻辑 0 ，则 UART 还没有接收字符。该字符被输入之前， 等待这个 标识位的程 序是简 单的， 如例 1-3 所示。

例 1-3 字符被输入之前， 等待接收器标识位

```
        /监控 UART 的接收器标识。
        /当该标识为逻辑 1 时，已经接收了一个
        /字符，并且可以输入它。
RCVR, IN    /把含有 UART 的接收器
     101    /的标识位的数据字输入
     ANI    /只把接收器标识位保存在 A 寄存
     020    /器(020＝十六进制 10)
     JZ     /这个标识位仍是逻辑 0，所以
     0      /继续等待它成为逻辑 1。
     IN     /该标识位是逻辑 1，所以
     100    /从 UART 输入这个字符，
     RET    /然后把它输入 A 寄存器里，返回。
```

软件不仅能够检测标识位的逻辑 1 ， 而且还必须能以某种

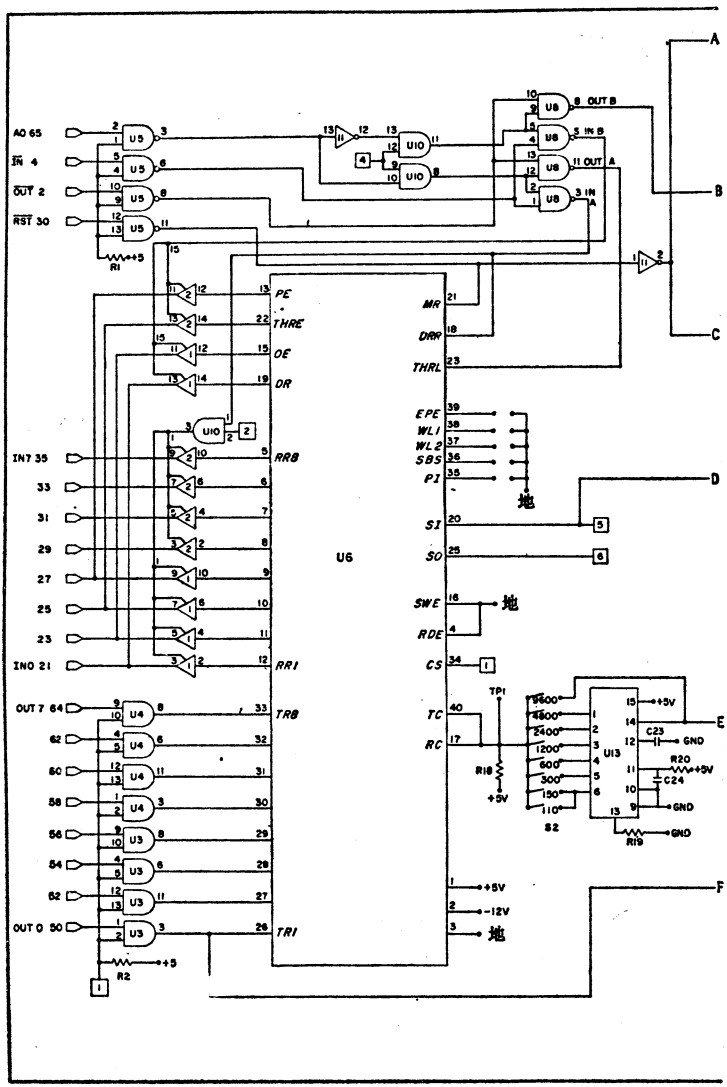
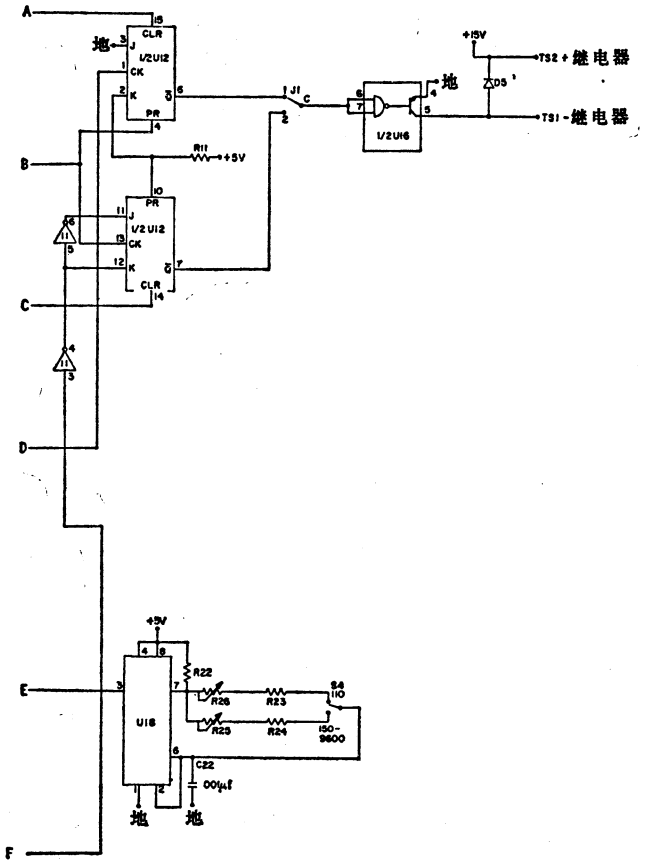

A
B
C
D
E
F
AO 65
IN 4
OUT 2
RST 30
U5
U10
U8
8 OUT B
5 IN B
11 OUT A
3 IN A
R1
+5
U6
PE
THRE
OE
DR
RRB
RRI
TRB
TRI
MR
DRR
THRL
EPE
WLI
WL2
SBS
PI
SI
SO
SWE
RDE
CS
TC
RC
地
IN7 35
33
31
29
27
25
23
IN0 21
OUT 7 64
62
60
58
56
54
52
OUT 0 50
U4
U3
TP1
R18
+5V
U13
9600
4800
2400
1200
600
300
150
110
C23
C24
R20
R19
GND
S2
-12V
R2

图 1-3 以 UART 为


A
B
C
D
E
F
地
CLR
J
CK
K
PR
1/2U12
R11
+5V
J1
C
1/2U16
+15V
D5
+TS2 + 继电器
+TS1 - 继电器
U18
R22
R26
R23
R24
S4
110
150-
9600
C22
.001μF

基础的异步通信接口

方法把该标识位清除为逻辑 0 状态。如果该标识位不能被清零，那么，即便是外部设备只向发送器发送了一个字符，8080 也会多次检测和输入同一个字符。只要把一个负脉冲加到引线 18（数据准备好复位线），使数据有效标识复位，则可以把该标识位置成逻辑 0 状态。

如果巧妙地进行硬件设计，用例 1-3 的程序的第二条 IN 指令，不仅可将 UART 所接收的字符输入，而且还能用来控制 UART 的引线 18，把数据有效标识位清零。这就是说，当电传打字机或 CRT 键盘上的一个键被按下时，8080 将接收的每个字符只输入一次。以 UART 为基础的异步通信接口的电路如图 1-3 所示。这个接口电路的地址译码器如图 1-4 所示。

读者注意，该接口采用多总线设计，它的结构类似于 S-100 型总线。由于添加了一组三态缓冲器，所以对 UART 的三态输出进行了缓冲。这样做是需要的，因为 UART 的输出速度太“慢”，不能与 8080 的定时兼容。

采用 UART 器件的优点和缺点是什么呢？虽然 UART 是一种强功能的集成电路，但是，它仍然需要附加许多集成电路，才能完成电传打字机（或 CRT）和 8080 微型计算机的连接。在图 1-3 中，所采用的 UART 是通用仪器公司（General Instruments Corp·）的产品 Ay-5-1013。该 UART 需要外加－12V 电源，这可能增加一个只需要＋5 V 电源的微型计算机系统的成本。但是，其他 UART 器件，如西方数字产品公司(The Westerm Digital)的TR 1863，只需要＋5 V 电源(它使系统的成本略高一些)，因而是适用的。为了把 UART 的 TTL 输入和输出的逻辑电平变换为 20 mA 或适合许多外部设备所需求的 EIA（电子工业协会）标准接口 RS-232 C，还需

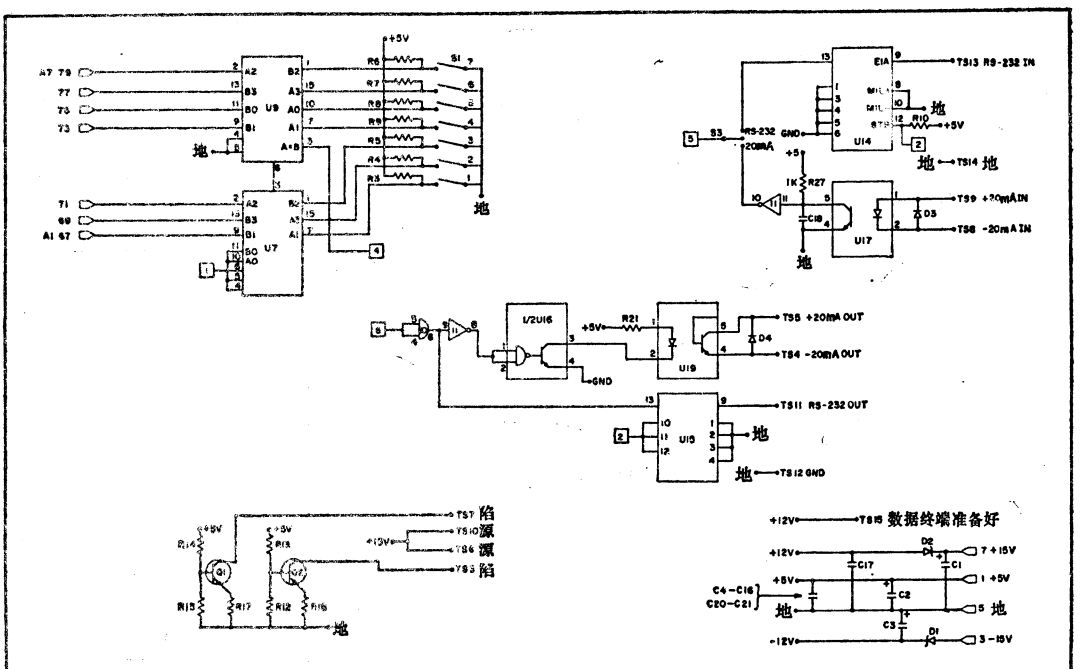


图 1-4　异步通信接口的器件地址译码器

要附加电路，如图 1-4 所示。

如果你对使用 UART 与诸如电传打字机和 CRT 的异步串行外部设备进行通信感到有兴趣的话，那么，请参考《接口与数据通信科学实验》一书。这本书不仅从深度上进一步探讨了 UART 的许多特点，而且包括许多实验供你实际使用，从而使你能熟练地使用 UART 器件。

以软件为基础的 UART

对电传打字机或 CRT 来说，“以软件为基础的”UART 好象一块 UART 芯片。只要认识到软 UART 和硬 UART 的优点和缺点，无论使用哪一种都不会出问题。软 UART 和硬 UART 不同之点在于：用许多条软件指令代替 40 条引线的集成电路。

这些指令把串行的逻辑 1 或逻辑 0 发送给电传打字机，或 CRT，或其它异步串行设备，或从这些设备接收串行的逻辑 1 或逻辑 0。使用 UART 集成电路时，可以给它编程序，发送和接收或只发送，或只接收 5、6、7、8 位的字符。偶或校验位也可以加到发送的字符上，也可以在接收的字符中检测它。只要把适当的电压加到这些可编程的引线上，可以消除奇偶校验位。对于以软件为基础的 UART，发送器或接收器的子程序必须予以增加或改变，5、6、7、8 位的字符才能发送和接收，或者只发送或者只接收。此外，为了发送或接收带有偶校验位、奇偶校验位或无奇偶校验位的字符，必须为8080编程序。为了简单起见，我们将只编写发送和接收 1 位起始位、8 位数据字和 2 位停止位的 UART 程序软件；将不使用奇偶校

验。这就是说，我们可为 8080 编程序，来发送和接收 11 位的字。

如图 1-5 所示，与软 UART 的子程序一起使用的接口电子器件是很简单的。该接口的串行输入和输出线的功能，与 UART 集成电路的串行输入和输出引线的功能是相同的，为了把该接口的 TTL 输入和输出逻辑电平转换为 20 mA，或 CRT 和电传打字机所需求的 RS-232 C 的标准电平，应该需要添加某些附加电路。但是，我们估计，如果采用软件串行技术，大约可以省去 8 块集成电路，包括 40 条引线的 UART 集成电路在内。如果省去了这些集成电路，为了使 8080 能和异步串行设备通信，需要更复杂的汇编语言指令序列。

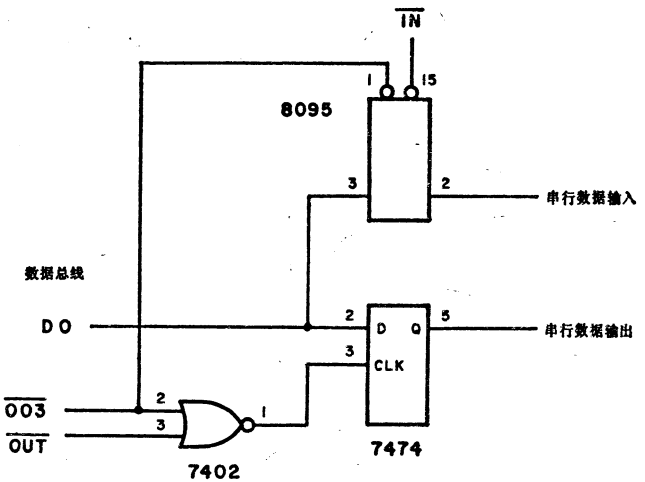


图 1-5　以软件为基础的异步通信接口

简单软件串行发送子程序如例 1-4 所示。该子程序所完成的许多功能与 UART 的发送器所完成的功能相同。例 1-4 所

列出的子程序只与要发送的 8 位字符一起调入；这个八位的字符现在被存储在 A 寄存器里。

该子程序首先保存寄存器对 H 的内容，然后把 013(十进制 11，十六进制 0 B)装入 L 寄存器，这个数是将要发送的启始位(1 位)，数据位数(8 位)和停止位数(2 位)的数目，然后把这个数与 A 寄存器的内容一起进行"或"操作。这并不改变A寄存器的内容，但是它把进位位清零。该逻辑 0 将被用作为启始位，它将用 1 位时间发送(即发送 1 位信息所需要的时间)。读者记住，在串行发送每个数据字开始时，必须发送 1 位启始位。

这条 ORAA 指令被执行之后，A 寄存器的内容和进位循环左移一次。这就把进位(逻辑 0)循环移入 A 寄存器的 D_0 位。然后OUT 指令把 A 寄存器的 8 位内容置于数据总线，但是，接口中的 SN 7474 D 触发器只锁存 D_0 位(如图 1-5 所示)。这位启始位被 D 触发器锁存以后，8080 调用 DELAY 子程序。

例 1-4 以软件为基础的异步串行发送子程序

/该子程序使用 1 位的输出端口

/来发送异步字符。

/送入要发送的 8 位字符和子程序，要发送的字符存储在在 A 寄存器。

```
TRANS,    PUSHH     /把寄存器对 H 的内容保存在堆栈。
          MVIL      /把要发送的数据的位数
          013       /装入 L 寄存器。
          ORAA      /保持 A 的内容不变，只把进位
          RAL       /清零。把进位循环移入 A 的D0位。
NXTBIT,   OUT       /输出D0
          003
```

```
        CALL    /调用产生 1 位时间的
        DELAY   /延时子程序。
        0
        RAR     /现在，把 D1 循环移入 D0。
        STC     /把进位置为停止位。
        DCRL    /位数减1。
        JNZ     /如果位数不等于0，
        NXTBIT  /返回，发送另一位
        0       /数据位
        POPH    /把 H 和 L 原来的值恢复。
        RET     /然后从该子程序返回。
DELAY,  PUSHPSW /把标识位和 A 的内容保存在堆
        PUSHH   /栈，把 H 和 L 的内容保存在堆栈。
        LXIH    /把一个 16 位的定时
        365     /字节装入寄存器对H。
        002
WAIT,   DCXH    /定时字节减 1
        MOVAM   /把这个最高有效字节送到 A。
        ORAL    /把最低有效字节与它相"或"。
        JNZ     /如果这个 16 位数不等于0。
        WAIT    /则执行 JNZ-DCXH 指令。
        0
        POPH    /从堆栈弹出 H 和 L 的内容。
        POPPSW  /和 PSW。
        RET     /从这个延时子程序返回。
```

这个延时子程序产生 1 位时间的延迟。该延迟时间是多长呢？这个子程序所产生的时间延迟的持续时间由 8080 必须传送给外部设备的信息所需要的速率来决定。如果 8080 正在与

每秒发送 10 个字符的电传打字机通信，那么，所要求的传送速率为 110 比特/秒(记住，每个字符为 11 位)。因此，每位的传送时间是 9.09 ms。这就是说，1 位起始位，8 位数据位，和两位停止位都必须由 D 触发器锁存 9.09 ms(9090μs)。在例 1-4 中，DELAY 子程序所产生的延迟时间是 9.09 ms。我们假设，8080 正在向电传打字机(该机的速度是 10 个字符/秒)发送字符；8080 的周期为 500 ns。

当 DELAY 子程序被执行时，A 寄存器、标识位以及寄存器对 H 的内容保存在堆栈。然后把 16 位定时字节装入寄存器对 H 里。8080 将寄存器对 H 的内容减 1，直到它等于 0 时为止。当寄存器对 H 的内容为 0 时，8080 恢复这些寄存器的内容和标识位，然后返回到发送器子程序。为什么在该延时子程序的开始把 PSW(处理机状态字)保存在堆栈呢？这样做是因为，A 寄存器和进位含有正在被发送给外部设备的信息。因此，我们必须把这些信息保存在堆栈。

8080 从 DELAY 子程序返回时，它已经用了 9.09 ms 发送了这位起始位。然后，8080 把 A 寄存器的内容、进位向循环右移 1 位。这样，则把 D_0 移入进位，D_1 进入 D_0，等等。现在，A 寄存器存储了必须发送给外部设备的那个原 8 位的字符。当 STC 指令被执行时，则把进位置于逻辑 1。然后，把存储在 L 寄存器的位数(发送的信息的位数)减 1。如果执行这条 DCRL 指令的结果不等于 0，则 8080 执行 JNZ-NXTBIT 指令。在 NXTBIT 处，8080 把 D_0 位的内容输出，然后由接口中的 D 触发器锁存。8080 然后调入 DELAY 子程序，用 9.09 ms把 D_0 位发送给外部设备。

既然 A 寄存器的内容正在一次 1 位地发送，那么首先发送 LSB(最低有效位)还是 MSB(最高有效位)呢？这位起始位

(始终是0)发送之后，则首先发送 A 寄存器中的 LSB (最低有效位)。A 寄存器中的 MSB 是最后要发送的数据位。如果需要的话，可以改变软件，就能首先发送 MSB。但是，几乎所有的电传打字机和 CRT 都要求首先把数据字的最低有效位发送给它们。

现在，让我们来探讨一下两位停止位是怎样产生的。在 8 位数据字全部发送后，A 寄存器的内容为 377(FF)，因为每次循环移位后，进位被置成逻辑 1，并且 8080 执行这条 RAR 指令，把进位移入 A 寄存器。假设要把字符 107(47)发送给外部设备。当 8080 执行这条 OUT 指令时，A 寄存器的内容和进位如表 1-6 所示。A 寄存器中的 8 位数据内容被发送以后，L 寄存器(位数)仍旧存储 002(02)。因此还必须发送两位附加位。这个子程序总是发送 A 寄存器的 D_0 位，所以从进位循环移入 A 寄存器的逻辑 1，实际上形成了两位停止位。因此，最

表 1-6 字符发送时进位和 A 寄存器的内容

进 位	A 寄 存 器	位
0	10001110	起始位
1	01000111	数据-LSB
1	10100011	数据
1	11010001	数据
1	11101000	数据
1	11110100	数据
1	11111010	数据
1	11111101	数据
1	11111110	数据-MSB
1	11111111	停止位
1	11111111	停止位

后两次通过传送循环，A 寄存器的内容是 377(FF)，而这两位逻辑 1 作为停止位发送给外部设备。

关于以软件为基础的异步串行 UART 的“发送器”，我们最后要说明一点：当UART（硬件的或软件的）没有向外部设备发送字符时，UART 的输出应该是逻辑 1。对于 20 mA电流环路，该逻辑 1 会使电流流过该环路。如果外部设备使用 RS-232 C标准接口，那么，UART 输出的逻辑 1 一定会把 +5 V和 +15 V之间的电压电平输出给外部设备。当电源加到UART芯片时，并且使它复位时，UART 的串行输出变为逻辑 1。当把电源加到图 1-5 所示电路时，SN 7474 D 触发器或者输出逻辑 1，或者输出逻辑 0。至于输出的逻辑电平是什么，则无法知道。因此，使用这种以软件为基础的串行发送器电路，在任何程序的开始，应该执行例 1-5 所列出的指令。这些指令把逻辑 1 装入 D 触发器里，经过适当的电平转换（如把 TTL 电平转换成 20 mA，或 RS-232 C的电平）后，发送给外部设备。

读者知道，采用这种技术，发送 8 位异步串行字符比较容易。使 8080 接收异步串行字符的子程序仅稍微复杂一点。

例 1-5 把串行输出端口置逻辑 1

```
START,   LXISP      /把一个 R/W 存储器地址
         STACK      /装入堆栈指示器。
         0
         MVIA       /然后把逻辑 1 装入
         001        /A 寄存器的 D0 位。
         OUT        /把 D0 位输出给
         003        /D 触发器。
         .          /然后执行该程序的
         .          /其余指令。
```

以软件为基础的接收器所需要的硬件有 1 位输入端口，如图 1-5 所示。当执行适当的输入指令时，1 位信息将被输入到 A 寄存器的 D_0 位。在发送异步串行数据时，正象对一些事件必须精确的定时一样，它们也必须进行精确的定时，8080 才能正确地接收异步串行数据。

当执行例 1-6 的指令时，8080 希望接收的串行数据流包括 1 位起始位，8 位数据位(首先是最低有效位)，和两位停止位。这样，8080 首先检测为逻辑 0 状态的一位输入端口。这样做是为了检测发送给 8080 微型计算机的起始位的开始时间。因此，8080 把该信息输入到 A 寄存器的 D_0 位。ANI 指令把 A 寄存器中的其余各位都置成逻辑 0。如果 A 寄存器的内容不等于 0，那么，现在没有接收启动位，然后，8080 执行JNZ RCVR 指令。当逻辑 0 被检测后，则调入 HALF 子程序，该子程序产生半位时间的延迟。如果以 110 比特/秒的速率接收数据，则一位时间是 9.09 ms，半位时间是 4.54 ms。

例 1-6 以软件为基础的异步串行接收器子程序

```
                /该子程序使用 1 位输入端口，
                /接收 8 位字符。8080 接收的字符存入
                /B 寄存器，然后退出
                /该子程序。该字符的
                /奇偶校验将不被检查。

RCVR,   IN      /从一位输入端口
        003     /输入这个数据(003＝03，16 进制)。
        ANI     /只保存 D 0 位
        001     /(001＝01，16 进制)。
        JNZ     /无数据在发送，甚至起始位也没
        RCVR    /有，所以继续等待
```

```
          0          /起始位。
          CALL       /检测到了 0，所以等待半位时间。
          HALF
          0
          IN         /然后，再测试数据线。
          003        /以保证确实检测了 1 位起始位。
          ANI
          001
          JNZ        /它是噪声，所以返回到该子程序
          RCVR       /的始点。
          0
          LXIB       /它是有效起始位，所以把 B 寄存器
          010        /清零，作为暂存器，C 寄存器置
          000        /010，因为它是位计数器。
NXTBIT,   CALL       /等待 1 位时间后，
          DELAY      /数据值才输入到 A 寄存器
          0          /的 D0 位。
          IN         /正在该位的中间，
          003        /所以输入它，
          ANI        /只保存这一位数据。
          001
          ADDB       /把这个暂存字加到这一位。
          RRC        /把它向右移一位。
          MOVBA      /然后把结果保存在 B 寄存器。
          DCRC       /位计数器减 1。
          JNZ        /它不是 0，所以取另一位。
          NXTBIT     /但是，首先等待 1 位时间，
          0          /再输入这位数据位。
          MOVAB      /取这个字符。
          RLC        /把它循环左移 1 次，
```

```
        MOVBA     /然后把它送回 B 寄存器。
        CALL      /所有的数据位都已被读出，所以，
        DELAY     /等待最后这位数据位通过，
        0         /然后是停止位。
        CALL
        DELAY     /(已经产生了
        0         /两位延迟时间)
        RET       /把该字符存入 B，然后返回。
DELAY,  CALL      /为了产生 1 位延迟时间，
        HALF      /两次执行
        0         /HALF 子程序。
HALF,   PUSHH     /把 H 和 L 的内容保存在堆栈。
        LXIH      /把一个 16 位的定时字节
        172       /装入寄存器对 H，
        001       /它是位速率的一半。
WAIT,   DCXH      /把这个定时字节减量。
        MOVAH     /把 MSBY 送到 A 寄存器。
        ORAL      /把它与 LSBY 进行"或"操作。
        JNZ       /该定时字节不等于 0，所以再
        WAIT      /执行 DCXH
        0         /指令。
        POPH      /该结果为 0，把 H 和 L 寄存器
        RET       /的内容从堆栈弹出，然后退回。
```

当 8080 从 HALF 延时子程序返回时，再一次测试正在被接收的这一位。如果这位数据位仍然是 0，那么，正在检测真正的起始位。如果这一位已经变成逻辑 1，那么，正在被接收的逻辑 0 也许是由于噪声引起的。如果属于这种情况，8080 绝对不能把该噪声看作为串行字符，所以它返回到 RCVR 子程

序。如果起始位首先被检测以后，它在半位时间仍然存在，那么，8080 把 000010 (0008) 装入寄存器对 B。这条指令做两件事：第一，它消除 B 寄存器。这是重要的一点，因为 B 寄存器将被用来暂时存储将要接收(一次接收一位)的字符。第二，它把位数装入 C 寄存器。在这个例子中，把十进制数 8 装入 C 寄存器，所以将接收一个 8 位字符。十进制数 8 是将要被接收的这个字符的数据的位数；它并不会受到要接收的停止位的位数的影响。在 NXTBIT 处，8080 调入 DELAY 子程序。为什么这样做呢？

在 NXTBIT 处，8080 执行 DELAY 调入指令时，用了半位时间，起始位已经被发送给了 8080。现在 8080 等待整位时间，从而它**在下一位时间的中点，能够对数据输入线取样**。如果用这种方式对数据流取样，时钟速率的误差或差别会减至最小。这就是说，在这个例子里，发送速度比 110 比特/秒稍高或稍低，8080 还能正确无误地接收这个数据字。例如，即使 8080 已经被编好了程序，以 110 比特/秒的速率接收数据，实际上可以以 106 比特/秒至 114 比特/秒的速率向 8080 发送数据。如图所示，我们对于串行发送的字符，在字符前面，加了箭头，表示 8080 在 1 位输入端口上实际测试数据的地方。我们在启始位前没有画出箭头，是因为 8080 会以很快的速度反复测试该输入端口的缘故。如果我们画了箭头，那么就会一个箭头压在另一个上面。这个新的图形可以如图 1-6 所示。

人们以类似的方法检测构成这个 8 位的数据字的其余各位。DELAY 子程序被调入以后，输入这个数据字，8080 执行这条 ANI 指令之后，只把从这个输入端口来的 1 位数据保存在 A 寄存器里 (D_0位)。然后把 B 寄存器的内容加到 A 寄存器存储的 0 或 1 上。把相加的结果再循环右移一次，然后送回到 B 寄

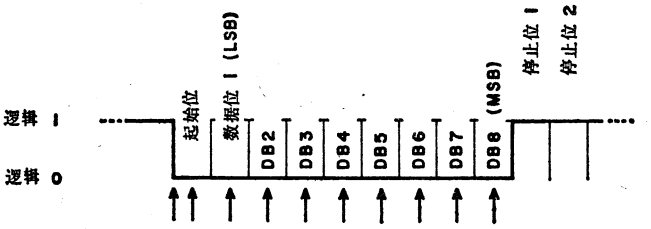


图 1-6 串行数据和取样的时间关系

存器。因为该移位转操作是由 RRC 指令完成的，所以不把进位的内容移入A寄存器。这个值被保存在B寄存器之后，C寄存器的位数减 1。如果从这个输入端口输入的数据字还没有输入八次，则位数不等于 0，因此，808 0执行 JNZ-NXTBIT 指令。8080 已经接收 8 位数据以后，它两次调入 DELAY 子程序，以便 8080 "接收"两位停止位，——虽然这两位不是由 8080 输入和检测的。

为了使 8080 微型计算机能够以 600 比特/秒的速率接收数据，这个 RCVR 子程序应做什么改变呢？该子程序必须改变的唯一一段是时间延时子程序 HALF。请读者记住，该子程序所产生的时间延时等于半位时间。所以，新的定时字节可以计算如下：

$$\frac{1\text{ s}}{600\text{ 位}}=1.66\text{ ms}=\text{位时间}$$

$$\frac{1.66\text{ ms}}{2}=833\ \mu\text{s}=\text{半位时间}$$

8080 执行这个 HALF 子程序的DCXH,MOVAH,ORAL，和 JNZ 四条指令，需要 24 个周期。如果 8080 的指令执行周期为 500 ns，那么，要执行这些指令需要 12 μs。

$\frac{833\ \mu s}{12\ \mu s} = 69_{10}$(HALF 的这个循环执行 69 次)

这就是说，装入寄存器对H的16位定时字节必须等于69_{10}。我们可以使用定时字节 000105(0045)。这个数必须存储在存储器，紧跟在 HALF 子程序的 LXIH 指令的操作码之后。这个定时字节是假设 8080 的周期时间为 500 ns(2 MHz)的情况下确立的。许多其他位速率的定时字节如表 1-7 所示。

表 1-7 不同接收速率(周期500ns；半位时间)的定时字节

接收速率(比特/秒)	十 进 制	八 进 制	十六进制
110	378	001172	017A
300	138	000212	008A
600	69	000105	0045
1,200	34	000042	0022
2,400	17	000021	0011
4,800	8	000010	0008
9,600	4	000004	0004
19,200	2	000002	0002

RCVR 子程序可以用来可靠地接收以 19200 比特/秒的速率发送的异步串行字符吗？不，不能。该数据速率的位时间是 52.08 μs(1 位/，19,200 比特/秒)。这就是说，这个 HALF 时间延时子程序必须产生 26.04 μs 的延迟时间。如果把 000 002 (0002)装入寄存器对H里，HALF 子程序中的时间延时循环程序将产生 24 μs 的延迟时间。但是，执行 PUSHH，LXIH，POPH 和 RET 这四条指令还需要 20.5 μs 时间。这就是说，实际上 HALF 子程序所产生的时延是 44.5 μs。

当以 110 比特/秒的速率接收字符时，这种附加延时程序(“辅助指令”)重要吗？不，并不重要。对于这种数据速率来说，

必须产生 4.54 ms(4545 μs)的延迟时间。增加 20.5 μs 的延时产生的错误只有 0.45%。

$$\frac{20.5\ \mu s}{4545\ \mu s}\times 100\%=0.45\%\ \text{错误}$$

遗憾的是，当这种数据速率越高时，由于这些“辅助指令”所引起的错误则越大。我们把 HALF 子程序所产生的错误编入了表 1-8。

表 1-8 HALF 子程序在不同的位率所引起的错误(周期为500ns)

位 速 率 (位/秒)	错 误
110	0.45%
300	1.23%
600	2.46%
1200	4.92%
2400	9.84%
4800	19.68%
9600	39.36%
19200	78.72%

8080 在什么速率时不能可靠地接收数据呢？我们的意见是：1200 比特/秒是这种软件的上限。请读者记住，表 1-8列出的错误是每一位的错误率。因此，对于一位启始位和 8 位数据字来说，以 1200 比特/秒的速度发送，总错误率是 44.28%。如果需要接收的速率越高，那么，必须参考此程序加以改进，把执行 RCVR 子程序的全部指令，包括 CALL 和 RET 这两条指令所需要的时间考虑进去。但是，这个问题超出了我们现在讨论的范围。

我们可以修改这个接收器子程序，接收 6 位的数据字吗？

这就是说，把1位起始位，6位数据字和1位或多位停止位发送给8080。它所需要的修改是很简单的。正好在NXTBIT之前，有一条LXIB指令。只要改变这条指令的第二个字节（装入C寄存器的数），就可以把8080将要检测的每个字的位数改变。为了接收一个6位的字，应该把这个数从010改变为006（08～06）。

假设读者把这个数据字节从010改变为020（08～10），那么，将会发生什么现象呢？如果我们假设一个16位的字正在发送给8080，8080则接收这个16位的字。但是，因为B寄存器是一个8位寄存器，用它来存储8080接收的这个字符，只能保存这个字的最后8位。这8位信息是被发送的16位字的最高有效字节（MSBY）还是最低有效字节（LSBY）呢？它应该是被发送的数据字的最高有效字节。最后，我们可以修改这个RCVR子程序，以接收字长从1位到8位的数据字。只要改变LXIB指令的第二个字节，就可以实现这种修改。

RCVR子程序（例1-6）改变A、B、C这三个寄存器的内容。为了只改变A寄存器的内容，有什么指令可以执行吗？有。在这个RCVR子程序的开始，在LXIB指令执行之前，可以执行一条PUSHB指令。然后恰好在这条RET指令以前应该执行MOVAB和POPB指令序列。用这条POPB指令恢复B和C寄存器的内容之前，需要使用MOVAB指令，把接收的数据字从B寄存器传送到A寄存器。

当8080在接收或发送数据时，它不能执行任何其他任务，这是以软件为基础的各种串行方法的最严重局限之一。例如，如果8080与硬件UART芯片一起使用，这个程序可能只测试发送器和接收器这两个标识位的状态，或者只测试二者之一的状态。如果这些标识位不是处于所要求的状态，那么8080可

以利用几毫秒时间执行其他任务。在该时间末了，8080可以再检查这些标识位的状态。如果这些标识位处于所要求的状态，则8080可以从接收器输入数据或者把数据输出给发送器。一旦该操作完成后，8080可以做其他任务，因为UART不再需要计算机/UART干预，而自动地发送数据或接收数据。

如果8080采用以软件为基础的串行发送方法，那么，它将或者在执行发送器程序或者在执行接收器程序。当8080开始实际发送或接收数据字时，它不能执行其他任务，这正是因为8080必须精确地为每位信息之间的时间定时。这就是说，8080被输入/输出束缚，即8080致力于处理外部设备传送数据给它的通信，或者从外部设备接收数据的通信。如果采用软件为基础的串行发送技术，8080与外部设备通信时，它不能执行其他任何操作。硬件UART和以软件为基础的串行发送技

表 1-9 硬UART和软UART的比较

特点比较	软UART	硬UART
最大速率(比特/秒)	1200(大约)	DC～50,000
硬件成本(美元)	1～3	20
软件(存储单元)	100	20
电源*	+5	+5,−12
有错误检查?	可能	有
受I/O束缚?	是	不
可靠性	取决于速度	最佳
决定位速率的因素	CPU的速度	外部时钟
同时接收与发送吗?	不是	是
同时与许多异步设备通信吗?	不是	是

* 有些UART工作只需要一种+5 V电源，可是它们的成本比需要两种电源的UART的还要高。

术的比较，如表 1-9 所示。选用软 UART 还是硬 UART 由读者自己决定。两种方法各有利弊。

8085 和 UART

硬 UART 可以与 8085 一起使用，其使用方法与 8080 微型计算机用 UART 的方法完全相同。它们的接口可能是完全相同的；用来与 UART 通信的程序指令也可能是完全相同的。但是，我们为 8085 设计了两条新指令，能更容易使用软件串行发送方法。8085 集成电路有 40 条引线，它的改进还表现在以软件为基础的位串行发送方法所需要的集成电路更少。

8085 集成电路上的两条引线专用于串行通信。这两条引线被命名为串行输入数据线（SID，引线 5）和串行输出数据线（SOD，引线 4）。8085 用了两条新指令，它们或者用来把 SID 引线上的数据输入到 A 寄存器的 D_7 位，或者把 A 寄存器的 D_7 位输出给 SOD 引线。RIM 指令把 SID 的状态读入到 A 寄存器的 D_7 位；SIM 指令把 A 寄存器 D_7 位的状态发送给 SOD 引线。把 SID 和 SOD 这两条引线上的 TTL 可兼容的电平转换为 20 mA，或电传打字机或 CRT 与 8085 进行通信所要求的 RS-232 C 的标准接口电平，正如我们前面的软件和硬件例子所说明的那样，将需要增添附加电路。RIM 和 SIM 指令的操作码如表 1-10 所示。

以软件为基础的发送器子程序列于例 1-7。它比前面以软件为基础的串行传送子程序（例 1-4）稍长一些，比较复杂。**这个子程序将只能在8085微型计算机上工作**。必须把将要发送给外部设备的这个数据字存入 A 寄存器才调入这个子程序。

表 1-10 单字节 RIM 和 SIM指令的操作码

操作码 八进制，十六进制	助记符	操作
040,20	RIM	把SID（串行输入数据）引线的状态读入A寄存器的 D_7 位*。
060，30	SIM	如果A寄存器的 D_6 位是逻辑1，则把 D_7 位的内容传送给SOD（串行输出数据）引线*。

* RIM和SIM 这两条指令使其它操作在 8085的A寄存器和 8085 的某些内部逻辑之间发生。这个问题将在第三章予以讨论

当 8080 调入 TRANS 子程序时，寄存器对H和B 的内容被保存在堆栈。在 DELAY 子程序中，寄存器对H用来存储一个定时字节；寄存器对 B 用来存储要发送的数据字和位数。这条 PUSHB 指令被执行之后，A寄存器的数据字被传送到 B 寄存器里；然后把将被发送的数据位的数目装入C寄存器。只要改变装入C 寄存器的数值就能给 8085 编程序，发送从 1 位到 8 位的数据字。然后，这条 MVIA 指令把A 寄存器的 D_7 位置 0，D_6 位置 1，其余各数据位都置 0。D_7 位代表起始位，它必须在数据发送之前，发送给 SOD。**当 8085 执行 SIM 指令时，A 寄存器的 D_6 位一定是逻辑 1，因为 D_7 位的内容要被发送给** SOD。SIM 指令（正好位于 NXTBIT 之前）然后把A 寄存器的 D_7 位发送给 SOD 引线，因为 D_6 位是逻辑 1。这条 SIM 指令并不改变A寄存器的内容。

一旦起始位已经被发送给 SOD 之后，8085 在 NXTBIT 处调用 DELAY 子程序。这就会使一位时间的延时产生。当8085 从 DELAY 子程序返回时，将要发送给外部设备的 8 位数据字从 B 寄存器传送到A寄存器。这条 ANI 指令把 A 寄存器中的

各位(D_7位除外)都置成逻辑 0 。只是因为 SIM 指令仅仅把 D_7 位发送给 SOD 引线。好象这条指令是不需要的。但是，当 8085 执行这条 SIM 指令时，D_0 位到 D_4 位使其他操作在 8085 集成电路之内出现。这个问题将在下一章进行讨论。

只要把这些位都置逻辑 0 ，这条 SIM 指令就只影响 SOD 的状态。这条 ADI 指令把A寄存器的 D_6 位置逻辑 1。在这个子程序中也可以使用一条 ORI 指令。A 寄存器的 D_6位必须是逻辑 1，当这条S IM 指令被执行时，D_7 位的状态才能发送给 SOD。如果 D_6 位是逻辑 0 ，那么，SIM 指令将不会影响 SOD 的状态。

这条 ADI 指令被执行之后，SIM 指令把 D_7 位发送给 SOD。C 寄存器所存储的位数然后减 1。如果这一操作的结果不是 0 ；则 8085 执行 JNZ NXTBIT 指令。在 NXTBIT 处，8085 调入 DELAY 子程序，因此，刚刚输出到 SOD 引线的信息位将在那儿保存 1 位时间之久。然后，按规则的延时间隔发送每位信息，从而产生串行数据流。

当数据位都被发送以后，（这由C寄存器的内容减为 0 来表示的）8085 不执行 JNZ-NXTBIT 指令。相反，8085 调入 DELAY 子程序，以便最后一位信息用整位时间发送。然后，8085 把 300(C 0)装入A寄存器，接着执行另一条 SIM 指令。它把 SOD 引线置于逻辑 1。这就是 1 位或多位停止位的开始。在例 1-7 中，8085 两次调入 DELAY 子程序，所以把两位停止位发送给串行设备。当发送操作已经完成以后，8085 从堆栈弹出寄存器对 B 和H的内容，然后，它从 TRANS 子程序返回。

例 1-7 8085 专用的软异步串行发送器子程序

```
/这个子程序只能在 8085 上执行。
```

```
/使用这条 SIM 指令，发送 8 位。
/异步字符。

TRANS,  PUSHH  /把寄存器对 H 的内容保存在堆栈。
        PUSHB  /把寄存器对 B 的内容保存在堆栈。
        MOVBA/把这个字符保存在 B 寄存器。
        MVIC   /把每个字符的数据位数。
        010    /装入 C 寄存器。
        MVIA   /把 01000000 装入 A，以便发送 0 启
        100    /始位(D6=1，SIM 指令执行)
        SIM    /把这个 0 输出给 SOD 引线。
NXTBIT, CALL   /调用产生 1 位延时。
        DELAY /的 DELAY 子程序。
        0
        MOVAB/取这个发送字符。
        RRC    /把 DO 循环移入到 D7和进位。
        MOVBA/把这个字符存回 B 寄存器。
        ANI    /只保存发送字的 D7位。
        200    /(200=80十六进制)。
        ADI    /加 01000000，以便 SIM 指令发
        100    /送这位数据位(100，十六进制 40)。
        SIM    /把 D7位发送给 8085 的 SOD 引线。
        DCRC   /位数减 1。
        JNZ    /如果位数不等于 0，
        NXTBIT/则返回，发送
        0      /另一位数据位。
        CALL   /现在用一位时间
        DELAY /发送最后一位数据位。
        0
        MVIA   /现在发送两位停止位
```

```
        300     /(300=CO，十六进制)11000000。
        SIM     /把1发送给 SOD 引线。
        CALL    /调用 DELAY 子程序，
        DELAY   /产生一位时间。
        0       /(这是1位停止位)
        CALL    /第二次调入 DELAY 子程序，以发
        DELAY   /送第二位停止位。
        0
        POPB    /把寄存器对 B 的内容弹出堆栈。
        POPH    /把寄存器对 H 的内容弹出堆栈。
        RET     /然后从这个子程序返回。
DELAY,PUSHPSW/把该标识位和 A 的内容保存在堆
        LXIH    /栈。把一个 16 位的定时字节
        365     /装入寄存器对 H。
        002
WAIT,   DCXH    /这个定时字节减 1。
        MOVAH   /把这个 MSBY 送入 A 寄存器。
        ORAL    /把它与这个 LSBY 进行"或"操作。
        JNZ     /如果这个 16 位的数等于 0，
        WAIT    /则执行 JNZ-DCXH。
        0
        POPPSW  /从堆栈弹出 A 的内容和标识位。
        RET     /然后从这个延时子程序返回。
```

我们怎样修改这个 TRANS 子程序才能只把 1 位停止位发送给外部设备呢？这只要把位于这条 POPB 指令之前和这个子程序末尾的 DELAY 子程序的一条 CALL 指令取消就行了。

改变例 1-7 的程序指令怎样才能节省两个存储器单元呢？这只要删去 DELAY 子程序的PUSHPSW 和 POPPSW 这两条

指令即可。在前面列出的以软件为基础的异步发送器子程序中，这两条指令是需要的，因为A寄存器存储了正在被发送给外部设备的这个数据字。但是，在例 1-7 中，被发送的数据字保存在B寄存器里，所以这两条指令没有用处。

读者可以看到，8085 的这个发送器程序比 8080 能执行的以软件为基础的串行异步发送器子程序长。但是，对于在 8080 微型计算机上将要执行的软件发送方案来说，必须构成一位输入端口和一位输出端口。由于 8085 微处理器集成电路的改进，在 8085 芯片上已经包含了这两个端口。

8085 的以软件为基础的异步串行接收器软件

例 1-6 和例 1-8 这两个接收器子程序的差别并不那么大。实际上，只有三条不同的指令。首先，RIM 指令用来读出 SID 的状态，把它输入到A寄存器的 D_7位。因为 RIM 指令不能把信息位读入 D_0 位，所以必须把 ANI 屏蔽字节从 001 改变成 200(01-80)。最后，因为将被接收的第 1 位数据位是最低有效位(LSB)，所以当接收这些数据位时，必须循环右移。因为每一位数据至少向右移一次（其中包括这个被接收的数据字的最高有效位），所以，必须把一条 RLC 指令加到该子程序的末尾(例 1-8)。读者可以看到，这个子程序与前面介绍的异步串行软接收器子程序(例 1-6)很相似。

例1-8 8085专用的以软件为基础的异步串行接收器子程序

```
/8085 的这个子程序使用 SID 引线
/和这条 RIM 指令来接收 8 位异步
```

```
/串行的字符。当程序控制从这个
/子程序返回时，接收的字符存储在 A 寄存器。

RCVR, PUSHB/把寄存器对 B 的内容保存在堆栈。
RCVR1,   RIM      /把 SID 读入 A 的 D7 位。
         ANI      /只把这个数据位保存在 D7 位
         200      /(200＝十六进制 80 )。
         JNZ      /现在没有发送信息位，甚至
         RCVR 1   /连起始位也没有，所以
         0/继续等待起始位。
         CALL     /检测到了零，所以等待半
         HALF     /位时间。
         0
         RIM      /再读 SID 引线，以便保证
         ANI      /一位起始位已被检测，
         200      /(200＝十六进制 80 )。
         JNZ      /它是噪声，所以返回到
         RCVR 1   /这个程序的起点。
         0
         LXIB     /它是有效启动位，把 B 置
         010      /000,供暂时存储用；把 C 置
         000      /010，因为 C 是位数计数器。
NEXTBIT, CALL     /在一位输入到 A 寄存器的
         DELAY    /D7 位之前，等待一位时间。
         0
         RIM      /现在读 SID 引线，然后
         ANI      /只把 1 位数据保存
         200      /(200＝16 进制 80)。
         ADDB     /把这个暂时存储的字加到这位。
         RRC      /把它循环右移 1 位。
```

```
        MOVBA   /然后，把这个结果保存在 B。
        DCRC    /位计数器减 1。
        JNZ     /它不是 0，所以，取另一位；
        NXTBIT  /但是首先等待 1 位时间，
        0       /才能输入这位数据位。
        CALL    /为了让停止位"通过"，
        DELAY   /现在等待两位时间。
        0
        CALL
        DELAY
        0
        MOVAB   /把这个接收的字符送到 A。
        RLC     /把它向右移一次。
        RET     /然后把它存入 A 而返回。
DELAY,  CALL    /为了产生 1 位时间延时，
        HALF    /两次执行 HALF
        0       /延时子程序。
HALF,   PUSHH   /把 H 和 L 的内容保存在堆栈。
        LXIH    /把 16 位定时字节装入 H 和 L，
        172     /它是半位速率的装入时间。
        001
WAIT,   DCXH    /定时字节减 1。
        MOVAH   /把该最高有效字节传送到 A。
        ORAL    /把这个最低有效字节与它"或"。
        JNZ     /这个定时字节不等于 0，
        WAIT    /所以再执行这条DCXH指令。
        0
        POPH    /该结果是 0，从堆栈弹出 H
        RET     /和 L 的内容，然后返回。
```

在微型计算机系统中，使用 8085 微处理器比使用 8080 微处理器有什么优点呢？如果这个微型计算机必须只与一台异步串行外部设备通信，那么，以软件为基础的一组发送器和接收器子程序与 8085 一道使用，也许会节省几块集成电路。如果使用 8080，那么，将还需要某些电路才能构成 1 位输入端口和 1 位输出端口。如果这台微型计算机必须与许多不同的异步串行外部设备通信，那么，8085 有几点优越于 8080，这只不过因为通信任务是由 UART 或由 USART 器件来执行的。

在表 1-9 里，我们列出了以硬件为基础的 UART 器件的一些特征。但是该表有一点应该予以强调。单独使用以软件为基础的异步串行方案，微型计算机不能同时发送和接收数据。我们假设用这种以软件为基础的方案执行例 1-9 这个程序。用每秒 10 个字符的电传打字机发送或接收数据。键盘每秒可能输入的字符和打字机每秒可能打印的字符最大是多少呢？这个问题与程序是在 8080 微型计算机上执行还是在 8085 微型计算机上执行无关。

例 1-9 软件为基础的接收和发送简单测试程序

```
/这是以软件为基础的串行接收
/简单测试程序

START, LXISP   /把一个读/写存储器地址
       STACK   /装入堆栈指示器。
       0
LOOP,  CALL    /从键盘输入
       RCVR    /一个字符。
       0
       CALL    /然后把这个字符发送给
       TRANS   /打字机。
```

```
0
JMP   /然后再返回到LOOP。
LOOP
0
```

如果使用例 1-9 所列出的程序，每秒钟只能接收和发送 5 个字符。其道理应该是很明显的：8085 微型计算机接收 1 个字符需要 10 0ms；把同一个字符往回发送给打字机也需要 100 ms。因此接收和发送同一个字符需要 200 ms。如果读者要把以软件为基础的串行方案用于通信问题，那么要记住这一点是重要的。随着位速率增至 600 比特/秒或 1200 比特/秒，这个问题就不那么重要了。

硬件器件——USART 与 UART

因为通用同步/异步接收器/发送器 (USART)是比UART更新的器件，所以，它不仅具有 UART 的各种特性，而且还具有其它的特点。USART 通常封装成 28 条引线的集成电路，但是，USART 不象 UART，开始人们已经把它设计得能与微型计算机一起使用。英特尔的 8251 USART 的引线结构和方框图如图 1-7 所示。

8080 用来与 USART 通信，有 8 条数据线 (D_0～D_7)。这些数据线把数据输出给 USART 中的发送器，并从 USART 中的接收器输入数据。8080 也可以输出**一个方式字和一个命令字**给 USART。这个方式字给 USART 的奇偶校验位，停止位位数和波特率系数编程序。

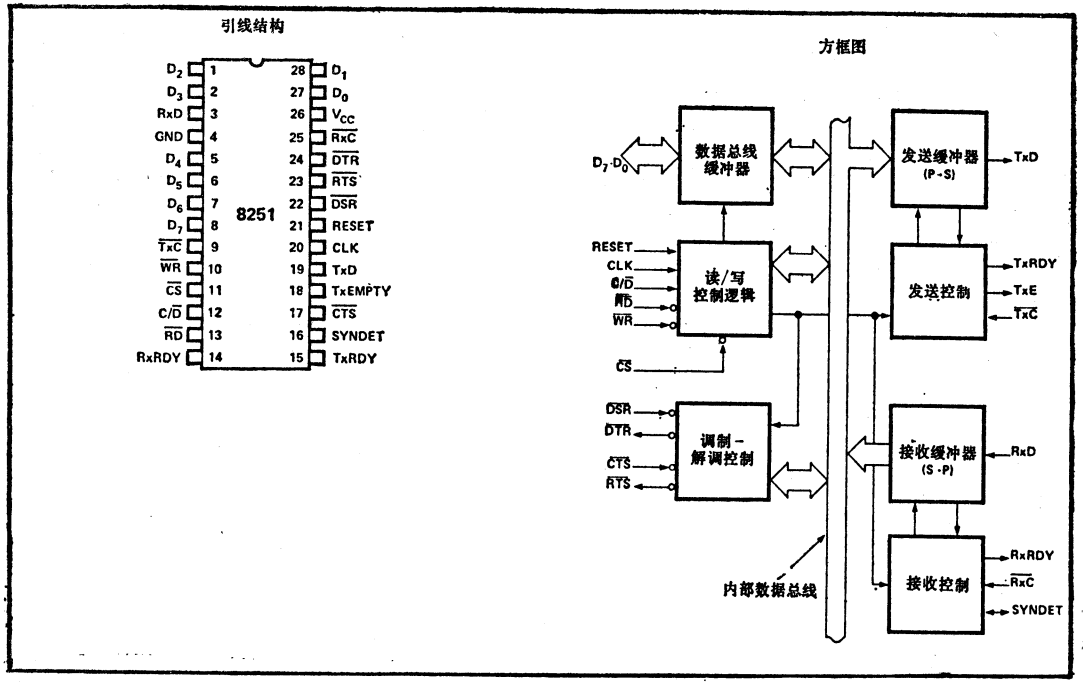


图 1-7　英特尔 8251 USART 的引线结构与方框图

图 1-7　8251 USART 的引线说明

引线名称	引线功能
$D_7 \sim D_0$	数据总线（8 位）
C/D	控制或数据写或读
$\overline{RD}$	读数据命令
$\overline{WR}$	写数据或控制命令
$\overline{CS}$	片　选
CLK	时钟脉冲(TTL)
RESET	复　位
$\overline{TxC}$	发送器时钟
TxD	发送器数据
$\overline{RxC}$	接收器时钟
RxD	接收器数据
RxRDY	接收器准备好（8080 有字符）
TxRDY	发送器准备好（从 8080 准备字符）
$\overline{DSR}$	数据置位准备好
$\overline{DTR}$	数据终端准备好
SYNDET	同步检测
$\overline{RTS}$	请求发送数据
$\overline{CTS}$	清除发送数据
TxE	发送器空
Vcc	+5 伏电源
GND	接　地

对于 UART 来说，它用外部连接电路为校验位和停止位位数编程序。如果使用软件，用波特率系数给 USART 编程序的能力是一种性能特征，它不能直接适应于标准的 UART。波特率是除数。为了得到一定的发送和接收速率，把它加给发送器和接收器时钟。有三种波特率系数可供选择，它们是 ×1，×16，×64。这就是说，加到 USART 的发送器和接收器（引线 9 和 25）的所需要的时钟频率实际上是由所要求的

位速率和给 USART 编程序的波特率系数决定的。波特率是向 USART 编程序的方式字的一部分。“普通”位速率的波特率系数和时钟频率如表 1-11 所示。

表 1-11 不同位速率和波特率所要求的时钟频率

位速率	波特率系数要求的时钟频率		
	×1	×16	×64
110	110 Hz	1.76 kHz	7.04 kHz
150	150 Hz	2.4 kHz	9.6 kHz
300	300 Hz	4.8 kHz	19.2 kHz
600	600 Hz	9.6 kHz	38.4 kHz
1,200	1,200 Hz	19.2 kHz	76.8 kHz
2,400	2,400 Hz	38.4 kHz	153.6 kHz
4,800	4,800 Hz	76.8 kHz	307.2 kHz
9,600	9,600 Hz	153.6 kHz	614.4 kHz
19,200	19,200 Hz	307.2 kHz	1228.8 kHz

如果需要的位速率是 1200 比特/秒，那么，可以把频率 1200 Hz，19.2 kHz 或 76.8 kHz 加到 USART 的时钟引线上。这些时钟频率相应的波特率系数是×1，×16 和×64。

用 USART 与外部设备进行通信之前，必须把 8 位方式字的格式装入 USART；这个 8 位方式字的格式，如图 1-8 所示。这个方式字装入 USART 之后，还必须把一个命令字装入 USART。这个命令字用来启动或禁止 USART 的发送器和接收器。8251 还具有控制调制解调(MODEM)的能力，所以，在这个命令字里的两位可能供调制解调控制用。这个命令字的另一位用来把校验错、溢出错和帧面错这些标识位复位。USART 上的这些标识位的功能和 UART 上的一样。8251 的命令字的格式如图 1-9 所示。

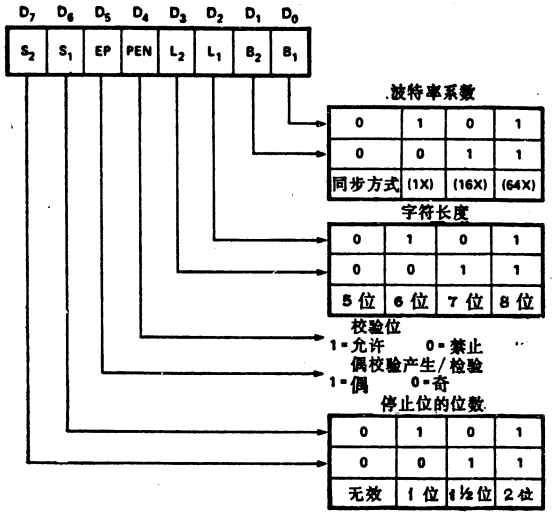


图 1-8 8251 USART 的方式字的格式

请读者注意，可以用这个命令字的 D_6 位来使 8251 USART 复位，回到方式字可编程序步。我们需要这一特征，因为这个方式字和命令字都输出给 USART 的**同一个输出端口**！或者把一个正脉冲加给复位引线（引线 21），或者把这个命令字的 D_6 位置逻辑 1 而使 USART 复位时，则 USART 复位，从而一个方式字可输出给它。

当这个方式字输出到USART 时，它被写入到方式字寄存器里。然后 8251 进行某些内部交换，以便下一次把一个字输出到 USART 的控制部分，这个字写入到命令字寄存器。只有脉冲加到 USART 的 RESET（复位）引线之后，或者当这个命令字的 D_6 位是逻辑 1 时，则把一个新方式字 输出给 8251 USART。这个命令字必须总是跟在这个方式字之后，记住这一

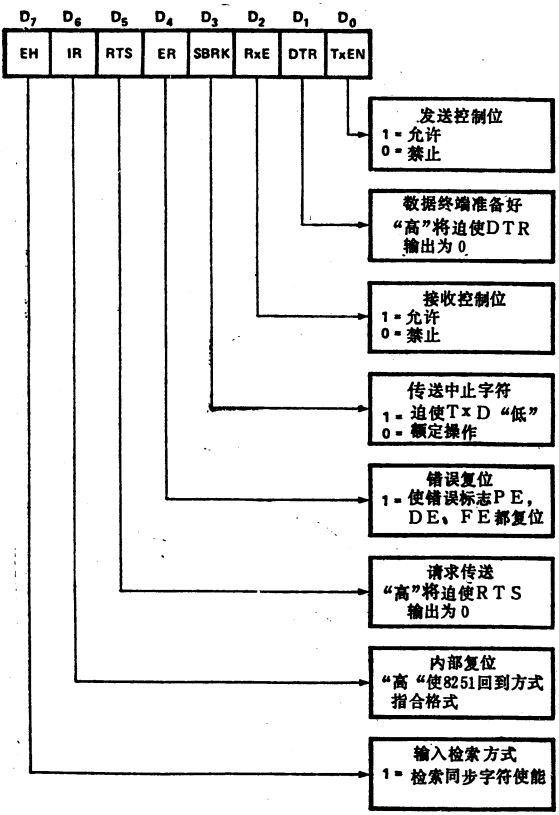


图 1-9 8251 USART 的命令字的格式

点，也是很重要的。当我们已经用一个方式字和一个命令字给 8251 编了程序之后，它可以供同步通信用或异步通信用。大多数 USART 用户把 8251 用于异步通信，所以，不讨论同步通信的问题。

在数据字的发送和接收时，8080 还应该能够监控发送器和接收器的状态，由此决定何时输出另一个数据字给发送器或

从接收器输入一个数据字。为达到此目的，USART 状态字才包含两位 标识位。它还包含三个错误标识位和另外 两个 标识位，我们不讨论这些标识位。另外还有 1 个标识位，即 DSR（数据置位准备好）标识位，供调制解调控制用。这个状态字的格式如图 1-10 所示。

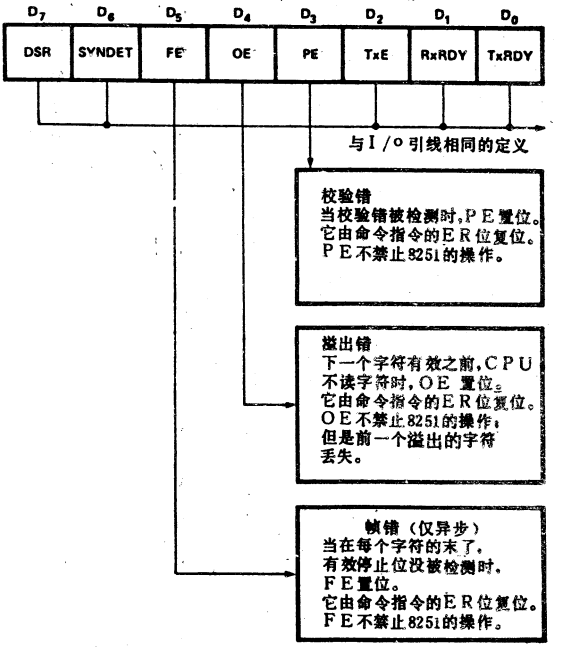


图 1-10　8251 USART 的状态字的格式

正如读者所知，实际上 USART 包含两个输入器件和两个输出器件。可以从 USART 读出这个状态字，或者可以读出一个数据字，并且，可以把一个数据字，或一个方式/命令字输出给 USART。当一个 8 位字被输出给 USART 的方式/命令寄存器部分时，第一个字必定总是方式字；然后，输出给 USART，

必须是命令字。任何数量的不同方式字和命令字都可以输出给USART，但是，方式字必须总是首先输出给 USART。

USART 有四条控制引线，实际控制着 USART 和 8080 之间的数据流。这些控制信号是写（$\overline{WR}$），读（$\overline{RD}$），控制数据（C/D）和片选（$\overline{CS}$）。8251 的这四个输入信号的真值表如表 1-12 所示。我们根据该真值表，能够容易设计 USART 和 8080 的接口（图 1-11）。当下述逻辑电平出现在地址总线的 A_1～A_7（由解码器来执行）时，8251 的 $\overline{CS}$ 输入（引线11）唯一地成为逻辑：

A_7	A_6	A_5	A_4	A_3	A_2	A_1
1	0	1	0	0	0	0

表 1-12 8251 USART 的控制信号的真值表

C/$\overline{D}$	$\overline{RD}$	$\overline{WD}$	$\overline{CS}$	功　　能
0	0	1	0	读接收器的数据。
0	1	0	0	把数据输出给发送器。
1	0	1	0	读状态字。
1	1	0	0	输出方式/命令字
x	x	x	1	USART 没被选；8080 不能与它通信。

x 表示不定状态；逻辑 1 或逻辑 0

我们根据这些地址线的逻辑电平，可以建立真值表，从而容易确定 USART 的输入和输出端口的地址(表 1-13)。

因为 USART 的读($\overline{RD}$)和写($\overline{WR}$)信号线与 $\overline{IN}$ ($\overline{I/OR}$)和 $\overline{OUT}$($\overline{I/OW}$) 连接，所以，我们应该用累加器 I/O 指令与 USART 进行通信。为了输入接收器数据，8080 必须执行一条 IN 240 指令。为了把一个数据字装入发送器，A 寄存器的内

表 1-13 USART 的输入和输出端口的地址

A_7	A_6	A_5	A_4	A_3	A_2	A_1	A_0	$\overline{RD}$	$\overline{WR}$	$\overline{CS}$	功　　能
1	0	1	0	0	0	0	0	0	1	0	读接收器。
1	0	1	0	0	0	0	0	1	0	0	装入发送器。
1	0	1	0	0	0	0	1	0	1	0	读状态。
1	0	1	0	0	0	0	1	1	0	0	装入方式/命令寄存器。
×	×	×	×	×	×	×	×	×	×	1	USART 没被选。

×为不定状态；逻辑 1 或逻辑 0，(A_0 连接到 $C/\overline{D}$。)

容必须输出给输出端口 240。为了把一个方式字或一个命令字输出给 USART，必须把A寄存器的内容输出给输出端口241。为了读出 USART 中的发送器和接收器的状态，必须为 8080 编程序，把信息从输入端口 241 输入到A寄存器里。

如果 8080 装备有 USART，那么，则可以执行例 1-10 所列出的程序。我们用这四条指令已向 USART 编制了什么样的指令序列呢？用二进制表示，这个方式字为 11010111。这个字为 USART 编程序，得到两位停止位和一位奇校验位。请读者注意，选择这位奇校验位，是因为这个校验位(D_5)是逻辑 0，校验使能位(D_4) 是逻辑 1，能使校验产生。这个方式字也给 USART 编程序，发送/接收每个字的 6 位数据位和所选择的波特率系数是×64。然后，8080 输出一个命令字 00010101；这就是把全部错误标识都复位，使 USART 中的发送器和接收器使能。请读者注意，这个方式字和命令字都已经输出给同一个输出端口了。

假设用两个不同的方式字 11101111 和 11001111 给 USART 编程序。如果使用一个或另一个方式字，那么 8251

USART 将会执行不同的功能吗？ 这个命令字是什么则无关紧要。读者可以假设两个方式字是相同的。这两个方式字的唯一差别是：一个用来给 USART 编程序，得到偶校验，而另一个用来给 USART 编程序，得到奇校验。但是因为这两个字的 D_4 位都是逻辑 0，所以 USART 不用校验位。

例 1-10 给 USART 的方式和命令字寄存器编程序

```
.
.
MVIA   /把后面这个数据字节装入A
327    /(16 进制 D7或 2 进制 11010111)。
OUT    /把它输出给 USART,
241    /决定该方式寄存器的状态。
MVIA   /然后输出这个命令字,
025    /(16 进制 15, 2 进制 00010101)
OUT    /该命令字到输出
241    /USART。
.
.
```

USART 的发送器子程序和接收器子程序与 UART 所用的子程序实际上很类似。USART 的这两个子程序如例 1-11 所示。在 UTRANS 子程序里，要输出给 USART 的这个字符必须存储在A寄存器里，然后，才调用 UTRANS 子程序。在检查 USART 的发送器标识位时，这个字符必须被保存在堆栈而不是保存在B寄存器里。

为了实现其功能，USART 还需要几个信号。USART 的清除传送($\overline{CTS}$，引线 17）引线应该接地。如果命令字的 D_0 位也是逻辑 0，那么这样会使发送器处于使能状态。USART 还

有时钟输入(CLK，31 线 20)信号。该时钟输入信号为内部定时用，并且受 USART 控制。**该时钟的频率与 8251 USART 的收送或接收速率无关**。但是，如果 USART 供同步通信用，该时钟频率必须比发送器或接收器的时钟频率至少高 30 倍。对异步通信而言，时钟频率必须比 USART 的位速率至少高 4 倍。该时钟可以从与 8080 一起使用的 8224 时钟发生器 TTL ϕ_2 时钟得到，或者从 8085 的 CLK(OUT)引线得到。

USART，如同 UART 一样，也需要两种与 TTL 兼容的时钟信号，分别供发送器和接收器用。这两种时钟信号分别加到接收器的 $\overline{\text{RXC}}$ 引线和收送器的 $\overline{\text{TXC}}$ 引线。时钟速度可以是位速率的 1 倍，16 倍或 64 倍，这是由方式控制字中的 D_1 和 D_0 位编程序所确定的。根据应用需要，这些时钟信号可以从计算机本机的晶体时基或从分开的振荡电路得到。要说明的最后一点是，8251 只需要一种＋5 V 电源。

例 1-11　累加器 I/O USART 接收器和发送器子程序

```
URCVR,  IN          /输入 USART 的状态字
        241         /(十六进制A1)。
        ANI         /只把接收器的状
        002         /态位(RXRDY)保存在 A。
        Jz          /如果这一位是 0，则还没有接收字符
        URCVR       /所以继续等待。
        0
        IN          /已经接收了一个字符，所以把
        240         /它输入到A寄存器，然后把它
        RET         /存入A而返回。
UTRANS, PUSHPSW     /把A的字符保存在堆栈。
UTRAN 1,IN          /输入 USART 的状态字。
        241
```

```
ANI       /只保存发送器的标识位
001       /(TXRDY)。
JZ        /如果这个标识位是 0，正在发送一个字
UTRAN 1   /符，所以等待它发
0          送完毕。
POPPSW    /弹出这个字符，送回到A。
OUT       /然后把它输出给 USART。
240
RET
```

存储器映象 UART 和 USART

UART 和 USART 不能用存储器映象输入/输出技术和 8080 微型计算机连接通信，这是毫无理由的。我们不使用 $\overline{IN}$ ($\overline{I/OR}$)和 $\overline{OUT}$($\overline{I/OW}$)，而使用 $\overline{MEMR}$ 和 $\overline{MEMW}$ 这两个信号控制 USART 的操作。这也就是说，必须用接口电子器件给 16 位地址译码，从而当适当的 16 位地址由 8080 置于地址总线上时，能启动 USART。

因为 USART 和 UART 所使用的接口非常相似，所以，本书的软件实例都是为 USART 而设计的。在后面的子程序中，我们假设把地址 200240 (80A0) 和 200241(80A1) 分配给了 USART。

例 1-12 的 LXIH 指令把分配给 USART 的方式/命令寄存器的 16 位地址装入寄存器对 H。先把方式字 337(DF)写出到 USART 的方式/命令寄存器，再把命令字 025(15) 写出到该寄存器。请读者注意，这个命令字的存储器地址不能改变。这个方式字为 USART 编程序，提供两位停止位，1 位奇校验，

8 位数据，和×64(波特率系数)。这个命令字把错误标识位复位，并且启动接收器和收送器。

例 1-12 存储器映象 I/O USART 预置初值指令

```
/这些 USART 预置初值指令用来
/给存储器映象 I/O USART 预置初值。

    .
    .
LXIH    /把 USART 的 16 位地址
241     /装入寄存器对 H。
200
MVIM    /把方式字写到
337     /到 USART。
MVIM    /然后把这个命令字
025     /写出到 USART。
    .
    .
```

供存储器映象输入/输出的 USART 一起使用的接收器子程序列于例 1-13。在这个子程序中，寄存器对 H 用来存储 USART 的 16 位存储器地址。因此，该子程序中的第一条指令把寄存器对 H 的内容保存在堆栈。然后把 USART 的 16 位地址 200241(80A1)装入寄存器对 H。这个地址是 USART 内的状态字的地址。然后把接收器标识屏蔽位 002(02) 装入 A 寄存器。然后把 A 寄存器的内容与寄存器对 H 寻址的状态字进行“与”操作。如果接收器标识位是逻辑 0，则执行 JZ—RWAIT 指令。最后，当 USART 接收一个字符后，接收器标识位将置成逻辑 1。

当这个标识位是逻辑1时，8080不返回到RWIAT，而是执行DCXH指令，使寄存器对H的内容从200241减1，成为200240(80A1—80A0)，这个地址就是USART的接收器和发送器的存储器地址。MOVAM指令把USART的接收器的内容传送到8080的A寄存器，然后把寄存器对H的内容从堆栈弹出。8080把接收的这个8位字符存入A寄存器，然后从这个子程序返回。请读者注意，可以用MOVDM指令或MOVBM指令，或其它许多访问存储器指令，把这个接收器字符读入到8080。

把数据写到USART的发送器的子程序是相当简单的，如例1-14所示。这个子程序也使用寄存器对H来存储USART的16位地址。因此，当这个子程序被调入时，寄存器对H的内容和PSW被压入堆栈。A寄存器存储将要写出到发送器的字符。在TWAIT处，把发送器标识屏蔽位装入A寄存器。然后，根据发送器标识位的状态，ANAM指令把8080的零标识位置位或者清零。如果这个标识位是0，则8080返回到TWAIT。

当这个标识位是逻辑1时，USART准备发送另一个字符。因此，DCXH指令使寄存器对H的内容减1，以便它为USART的发送器寻址。这条POPPSW指令把A寄存器的内容和标识位都从堆栈弹出。这样恢复了要写出到USART的那个字符。执行MOVMA指令时，把这个数据字输出给USART。随后，从堆栈弹出寄存器对H的内容，8080返回到曾调入MTRANS的程序段。

例 1-13 存储器映象输入/输出USART接收器子程序

```
/这个接收器子程序与存储器
/映象输入/输出 USART 通信。
```

```
MRCVR,  PUSHH /把寄存器对H的内容存入堆栈。
        LXIH  /然后把分配给 USART 的
        241   /16 位地址装入寄存器对H。
        200
RWAIT,  MVIA  /把标识屏蔽位装入A。
        002
        ANAM  /把状态字装入A。
        JZ    /这个字的D1 是 0,
        RWAIT /所以等待标识位成为逻辑 1。
        0
        DCXH  /然后，输入/输出地址减 1。
        MOVAM/把这个 USART 字读入 A。
        POPH  /从堆栈弹出寄存器对H的内容。
        RET   /把这个字存入A返回。
```

正如读者已经知道的那样，8251 既可以用作为累加器输入/输出器件，也可以用作为存储器映象输入/输出器件。至于应该用累加器输入/输出还是访问存储器指令与 USART 通信，完全取决于所采用的接口技术。

USART 也有三位控制位，它们对读者或许是有用处的。这些控制位是 $\overline{\text{DSR}}$（数据选择准备好）输入、DTR(数据终端准备好）和 RTS（请求传送)输出。这位输入位可以由软件检测。这两位输出位可以用软件控制。如果读者希望检测某个输入端口的状态（开/关)，或者控制某台外部设备；例如电传打字机的纸带阅读机或盒式录音机的马达，那么这些控制信号是有作用的。欲要更详细地了解 8251 集成电路的性能特征，请读者参考«8080 微型计算机系统用户手册» (The 8080 Microcomputer System User's Manual)。

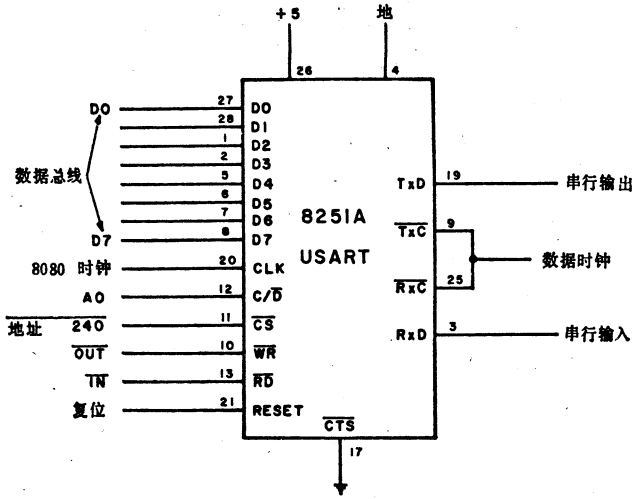


图 1-11　8251 USART 的累加器输入/输出接口

例 1-14　存储器映象输入/输出 USART 发送器程序

/这个发送器子程序与存储器
/映象输入/输出 USART 进行通信。

```
MTRANS,PUSHH    /把寄存器对H的内容保存在堆栈。
       PUSHPSW  /把A和标识位保存在堆栈。
       LXIH     /把 16 位地址装入寄存器对H，
       241      /该地址分配给存储映象 USART。
       200
TWAIT, MVIA     /把标识屏蔽位装入A寄存器。
       001
       ANAM     /再把状态字装入A寄存器。
       JZ       /这个字的 D0 位是 0，
       TWAIT    /所以等待标识位为逻辑 1。
```

```
0
DCXH    /存储器输入/输出地址减 1。
POPPSW  /从堆栈取出这个字。
MOVMA   /输出给 USART。
POPH    /把寄存器对H的内容弹出堆栈。
RET     /从该子程序返回。
```

第二章 中 断

在用了输入/输出设备的许多程序和子程序中，我们已经给8080编制了程序，等待与输入/输出设备相关的标识位。«8080/8085的软件设计»的上册中的许多例子和下册的第一章所列出的电传打字机和CRT的输入/输出子程序就是如此。给8080编制程序，使它等待输入/输出设备的理由是简单的，就是我们假设8080没有其它重要操作（任务）要执行。例如，«8080/8085的软件设计»的上册中的许多程序，只需要几百微秒来"处理"要输入或打印的字符。一般而言，这种处理操作包括比较字符，在存储器中存储字符，或从存储器中取出字符。但是，电传打字机(10个字符/秒)把一个字符送给接口电路的UART，或从它接收一个字符还需要100 ms。所以，8080用了很多时间去等待UART的接收器或发送器的准备好标识位。这就是说，8080不断查询UART的接收器标识或发送器标识。根据查询操作的结果，8080决定是否输入或输出字符。格腊夫(Graf)给查询定义如下：

查询 **1．对共享通信线路的每个终端的周期性询问，决定哪个终端需要服务。多路转换器或控制站（在本例，是8080）发送一个查询标识位，及时询问这个被选择的终端："你有什么需要发送的吗"？2．控制通信线路的手段。通信控制器件将把信号传送给终端，询问："终端A，你有什么需要传送吗？"如果没有，则询问："B终端，你有什么需要传送吗？"如此继续下去。查询是对竞争的选择。它确保任何终端不会长时间等待。**

当微型计算机为设备服务时，它只用该微型计算机的程序所规定的方式与该设备交换数字信息。这种具体程序段通常称之为**软件驱动器**。这种软件驱动器只是在微型计算机和指定的输入/输出设备之间传送信息。

如果有几个电传打字机或 CRT 与同一台 8080 微型计算机连接，则必须轮流查询每一台外部设备。为了查询这些外部设备，8080 可以检测 UART(USART)的接收器和发送器标识位的状态。如果 8080 发现标识位处在表示外部设备需要服务的逻辑状态，那么，8080 停止查询操作，开始按指令序列执行操作，为外部设备服务。为该外部设备服务之后，（一个新字符输出给它，或从它输入一个字符）8080 查询其余的外部设备。

查询操作对那些速度较慢，不需要经常服务的外部设备或者至少可以等待服务的外部设备，是很有用的。我们可以利用 8080 微型计算机和被查询的外部设备的速度的不同的特点，只为每台外部设备服务几分之一秒；8080 微型计算机在 100 ms 时间大约可以执行 20,000 条指令。

因为越来越多的输入/输出设备与 8080 微型计算机连接，所以，这就要花去 8080 微型计算机越来越多的时间查询外部设备。事实上，查询操作的时间太长，可能会丢失数据。请读者记住：UART 或 USART 有一位溢出标识位，用来表示被接收的字已经把前一个字改写了。如果下一个字被接收之前，没有把前一个字读入到 8080 微型计算机，那么，这种溢出错误将会出现。当然，必须避免这种情形。

前面的程序实例没有一个程序按顺序查问两个或两个以上的输入/输出设备。假设必须从两个不同的键盘向 8080 微型计算机输入数据值。从这两个键盘输入的 ASCII 字符必须存储

在读/写存储器的两个不同的存储段。这个具体问题的流程图如图 2-1 所示。执行这些任务所用的汇编语言程序如例 2-1 所示。

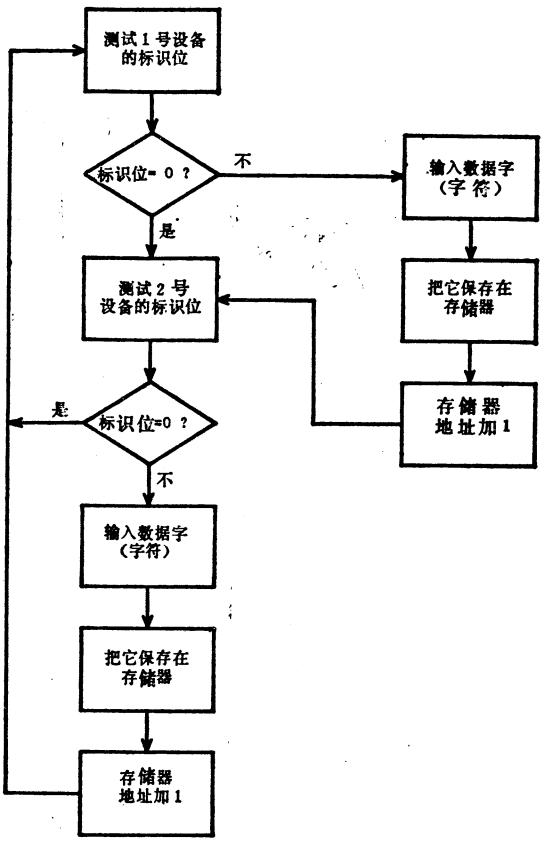


图 2-1 查询两个键盘的流程图

在这个程序的开始，把读/写存储器地址装入寄存器对 H；这个地址用来存储从一个键盘输入的字符。然后把另一个读/

写存储器地址装入寄存器对D；这个地址用来存储从另一个键盘输入的字符。在NODEV处，8080检查（查询）1号键盘的键盘标识位。如果这个标识位是逻辑1，这就是说，一个键被按下，则从键盘输入这个字符，并且把它存储在寄存器对H寻址的存储单元。然后，这个存储地址增1，8080返回到CHK2。这样做，8080才能在再查询第一号键盘之前查询第二号键盘。

如果8080确定第一号键盘的标识位是逻辑0，那么，它就测试第二号键盘的标识位。如果这个标识位也是逻辑0，则8080已经查询了两台设备，没有一台设备请求服务。因此，8080再执行查询指令序列。如果第二号键盘的标识位是逻辑1，8080必须为该键盘服务。8080为这个键盘服务之后，它就返回到NODEV，这样第一号键盘就成为将要查询的下一台设备。

对用查询输入/输出设备的微型计算机设计来说，并不是不可行的。但是，如果有大量的输入/输出设备需要查询，那么，查询这些设备所用的时间比实际为它们服务的时间长。因此，在微型计算机系统中，常常采用**中断**。如果中断硬件的结构适当，中断程序编得合适，那么，开始执行一台设备的服务子程序所用的时间和开始执行另一台设备的服务子程序所用的时间相同。在查询操作中，花去8080微型计算机很长的时间，才能得到8080微型计算机为外部设备服务的指令。

例 2-1 查询两个键盘的程序

```
/这个程序查询两个键盘。
/当标识位是逻辑1时，输入这个键盘字符，
/并且存储在存储器里。产生
/两个独立的字符表。

START,    LXIH    /把一个读/写存储地址装入寄存器对H。
```

```
        BUFFI   /从第一号键盘输入的字符
        O       /存储在这个地址。
        LXID    /然后把可以用来存储第二号键盘输入的字符的
        BUFF2   /地址存入寄存器对 D，
        O       /
NODEV,  IN      /输入第一号键
        001     /盘的标识位(16 进制 01)。
        ANI     /只把该键盘的标识位保存在
        001     /A 寄存器(001＝16 进制 01)
        JNZ     /这个标识位是逻辑 1，所以
        DEVC1   /返回到输入这个字符并且存
        O       /储它的程序段。第一号键盘
CHKZ    IN      /的标识位是逻辑 0，所以，
        003     /检测第二号键盘的标识位。
        ANI     /只把这个键盘标识位保存在 A
        001     /寄存器(16 进制 01)。该标识
        JZ      /位是逻辑 0，所以再检测两
        NODEV   /个标识位。
        0
DEVC2,  IN      /第二号键盘的标识位是逻辑
        002     /1,所以，输入这个字符(
        STAXD   /002＝16 进制 02)。用寄
        INXD    /存器对 D 作为地址，把
        JMP     /它保存在存储器。然后先检
        NODEV   /测第一号键盘的标识位。
        0
DEVC1,  IN      /第一号键盘的标识信号逻辑
        000     /1,所以输入一个字符。使
        MOVMA   /用寄存器对 H 里的地址，把这个字符
        INXH    /保存在存储器。然后，先
```

```
IMP     /检测第二号键盘
CHKZ    /的标识位。
0
```

中断操作

格腊夫给中断定义如下：

中断—1．**在计算机中，中断就是系统或子程序的正常流程的断点，从而在较后的时间，再从这一点起恢复，继续执行该流程。中断源可能是内部的，或者是外部的。**

在微计算机操作中，中断操作是更复杂的操作方式；中断操作比查询操作优越得多。例如，查询操作：

·微型计算机给与它连接的外部设备定序可能花去它很多的时间。

·当某台设备需要服务时，它必须等待到微型计算机查询完毕其它各种设备之后，才轮到为它服务。如果需要，还必须为别的设备服务之后，才能为它服务。

·设备需要服务和被服务的响应时间可能是很严格的，至少应按照微型计算机的标准。相比之下，中断操作：

·微型计算机可能用去其很多时间去处理数据，显示数据；也就是说它只执行一个等待循环，等待设备请求服务。

·可以有中断操作的优先权。如果两台设备同时需要服务，首先给比较重要的设备服务。

·当正在给优先权较低的设备服务时，如果优先权较高的设备需要服务，那么，它可以中断微型计算机，使之为它服务。

·设备需要服务和被服务的响应时间可以很短，甚至按微型计算机标准。对8080微型计算机而言，响应时间通常不大于10 μs(微秒)。

·软件复杂得多。

优先权和响应时间这两条术语定义如下：

优先权——把输入/输出设备根据其重要性排序，从而某些设备排在其它设备的前面，这种情况叫做优先权。

响应时间——一台设备发出中断请求和为它服务的软件驱动器的第一条指令的执行之间的时间。

中断的基本形式

中断的基本形式有三种：**单线，多级和向量中断。**

单线中断——在中断系统中只有一条中断线的中断系统。通过"或门"把多台外部设备连接到微型计算机上。这个"或门"的每一个输入端与一台输入/输出设备连接。当接到中断信号时，微处理器必须查询所有的外部设备，确定是哪台设备发出的中断信号。莫托罗拉公司的M-6800，MOS技术公司的6502和英特西尔（Intersil）公司的6100这类计算机常常采用这种中断方式。

多级中断——这是个中断系统；它有许多中断线与微型计算机连接，每条中断线分别连接到每台输入/输出设备上，这种连接形式的中断系统叫多级中断。这种微型计算机不需要查询各台外部设备，就能确定是哪台外部设备产生了中断。8085具有这种特征。

向量中断——这是个中断系统：中断直接产生程序分支，

转移到为它自己服务的程序段上。矢量中断是最快的中断操作方式。可以连接 8080 和 8085，为矢量中断设备服务。

这三种中断的电路方框图如图 2-2 所示。

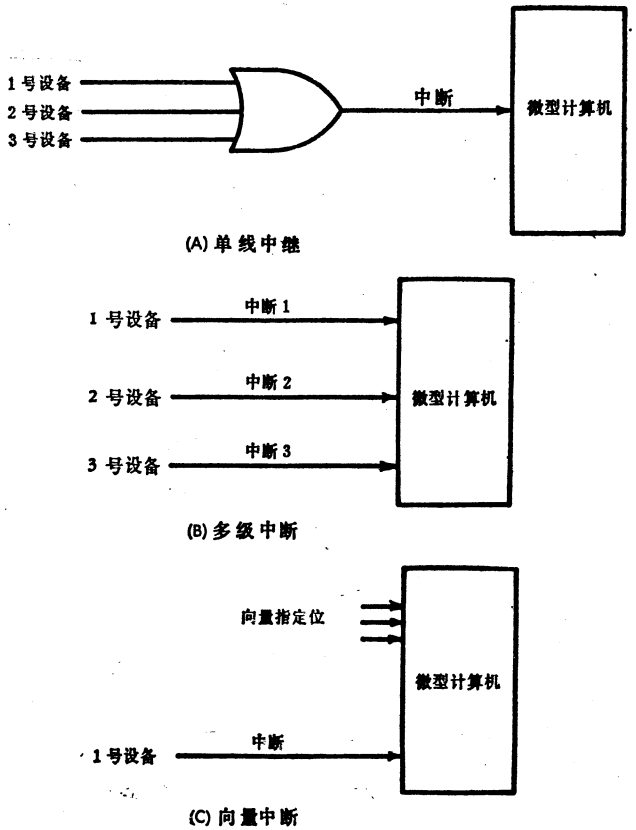


图 2-2　中断微型计算机的三种不同方法

单线中断是微型计算机用普通的中断方式，并且它容易实现“或门”的输出端与一条中断线连接，它的输入端连接外部设备的台数可以不受限制。如果连接三台外部设备，那么其连接

方式如图 2-2 A 所示。但是，一旦检测到中断之后，如果连接的外部设备越多，查询外部设备所需要的时间则越长。即使外部设备是被查询的，外部设备仍然有优先权。这就是说，中断一出现，被查询的第一台设备的优先权最高；最后被查询的一台设备，其优先权最低。

多级中断如图 2-2 B 所示，如果微处理器芯片上有足够数量的中断引线，那么，采用多级中断方式是有效的。可是，这种情况并不多见。现有微处理器芯片上的中断引线（如 8085）不会多于 5 条。

矢量中断方式，如图 2-2C 所示，它允许以现行执行的程序直接转移到为产生中断的外部设备服务的程序上去，立即为该设备服务。这种情况与调入**中断服务例行程序**的子程序很相似。当执行中断服务子程序时，微型计算机可以把数据输出给输入/输出设备，或以输入/输出设备输入数据。

对于 8080 微型计算机系统来说，外部设备首先产生中断信号。实际上，该中断信号就是由外部设备输出的标识位。发出中断信号的外部设备然后可以把一个字节、两个字节或三个字节的指令置于数据总线上，一次送一个字节。第一个字节是指令的操作码，把它写入(压入)指令寄存器 (IR)，然后 8080 把这条指令的其余字节(如果有的话)，置于数据总线、并翻译为该条指令的数据或地址字节。

中断指令

中断出现时，8080 需要以最快的速度为该中断服务。因此，我们可以把 JMP 或 CALL 指令塞入 8080 的指令寄存器

里。如果 JMP 或 CALL 指令的操作码被塞入这个指令寄存器，那么外部设备必须把另外两个字节置于数据总线上，一个字节是被调用的子程序的高位地址字节，另一个是低位地址字节。当三个字节都被置于数据总线之后，8080 就实际执行这条 JMP 或 CALL 指令。我们可以使用这条 CALL 指令的理由之一是因为，返回地址自动地保存在堆栈。这就是说，8080一旦为产生中断请求的设备服务完毕以后，它可以执行一条 RET 指令，然后返回到这个中断曾出现时正在被执行的程序段。

可是，把一条三字节的指令置入 8080 需要相当多的硬件。读者能否想到一、二个字节的，功能与 CALL 指令相当的一条指令吗？可以，再启动(RST)指令就是一例。请读者记住再启动指令(RST)可以当作是单字节的调用指令，其中，被调入的子程序的地址是固定的。表 2-1 概括了被调入的子程序的地址和这些 RST 指令。

表 2-1 再启动指令一览表

指令	操作码		存储器地址(被调入的)	
	八进制	十六进制	八进制	十六进制
RST 0	307	C7	000000	0000*
RST 1	317	CF	000010	0008
RST 2	327	D7	000020	0010
RST 3	337	DF	000030	0018
RST 4	347	E7	000040	0020
RST 5	357	EF	000050	0028
RST 6	367	F7	000060	0030
RST 7	377	FF	000070	0038

* 与硬线复位一致。

这些 RST 指令的操作码的优秀特征之一是：在八条再启动指令中，八位操作码只有三个是不同的。操作码的其余各位

(D_7、D_6、D_2、D_1 和 D_0)都是逻辑 1。这一特征将可以应用于中断硬件的设计。

产生中断信号的外部设备可以把一条指令置于数据总线，然后把该指令写入到指令寄存器，就是因为 8080 已经设计成在中断时间能接收指令。虽然我们通过中断外部设备方式，来只讨论了如何使用再启动指令的问题，但是，当为中断服务时，带有简单的中断硬件的外部设备也能把单字节指令置于数据总线。为中断服务时，我们或许愿意把一条 MOVAB,INRE, DCXE, ADCM 或一条 NOP 指令写入指令寄存器，至于具体写入哪一条指令、应视发出中断的外部设备的功能。执行这一任务所需要的硬件我们将在本章的另一节讨论。读者欲要用中断硬件和 8080 微型计算机具体实践一下，请参考本章末的参考文献 3。

允许与禁止中断指令

我们通常把许多输入/输出设备与 8080 微型计算机系统的中断信号线连接。当某台设备中断 8080 微型计算机时，微型计算机很快为它服务，然后，8080 返回去执行该中断曾出现时，它正在执行的那个任务。但是，如果 8080 正在执行延时子程序，或者 8080 正在从软磁盘读出信息，那么，必须有某些方法防止输入/输出设备中断这些任务的执行。如果 8080 正在执行延时子程序时被中断。那么，会因给发出中断的设备服务，需要一定时间，而使延时增长。如果 8080 微型计算机正在以软磁盘读出信息时而被中断，那么，8080 为中断设备服务时，这个软磁盘将继续旋转。这就是说，8080 返回到读软磁

盘子程序时，它将会从软磁盘的不适当的区段或者软盘的不适当的磁道读出信息。

因此，8080 芯片内有某种电路，另加一条单字节指令，从而使 8080 对各种中断信号成为“聋子”。该电路就是**中断触发器**，或**标识**(这些术语是指同一种概念)，只要微型计算机执行一条单字节指令，就可以把这个电路启动或禁止。

上面出现的一些术语依次定义如下：

中断标识 (interrupt flag) 即中断触发器 (inferrupt flip flop)—8080 微处理器集成电路中的触发器，它能够检测中断信号。它可以被微型计算机软件指令启动和禁止。

禁止中断(disable interrupt)—禁止微处理器集成电路的中断标识位。在这种状态下，中断信号无效。

允许中断 (enable interrupt)：启动微处理器集成电路的中断标识位。微处理器芯片处于这种状态时，中断被接收而加以处理。

8080 微处理器的允许和禁止中断指令如表 2-2 所示。

表 2-2 允许和禁止中断的单字节指令

操作码		助记符	操作
八进制	十六进制		
373	FB	EI	**执行下一条指令**才能启动微处理器的中断线。
363	F 3	DI	**执行这条指令时**，禁止微处理器集成电路的中断。

如果这条中断线被启动，那么 8080 立即为发出中断的设备服务。但是，8080 执行这条 DI 指令，禁止中断，那么，该中断信号无效。当然，如果 8080 的中断输入仍然处在表示中

断需要服务的状态，那么，只要 8080 处于允许中断状态(执行 EI 指令)，它就可以被中断。

8080 实际上是怎样被中断的

外部设备要中断 8080 微型计算机，必须启动 8080 的内部中断标识位。这只要执行 EI 指令就能实现。一旦这样做后，与中断信号线连接的任何外部设备都可以中断 8080。**为了给8080发出信号，告诉它，外部设备需要服务，8080 的中断引线(INT)必须被置到逻辑 1**。但是，这种情况出现时，8080 或许处在把一条指令只执行了一半的状态。**因此，当一个输入/输出设备要中断 8080 时，8080 要把正在执行的这条指令执行完毕**。这就是说，中断 8080 的外部设备不能把这条中断引线置于逻辑 1，简单地把 RST 指令的操作码置于数据总线。而是在把这条 RST 指令置于数据总线，并发送到指令寄存器之前，该接口电子器件必须等待，一直到与前所执行的这条指令执行完毕为止。如果 8080 的指令周期是 500 ns，那么，延时可能是 0～9 μs。

当中断出现时，8080 执行完这条正在被执行的指令，然后产生一个中断响应 $\overline{\text{(INTA)}}$ 信号。当然，如果中断标识位被起动，那么，8080 只产生 $\overline{\text{INTA}}$ 信号。用这个信号进行协调，将操作码送入数据总线，装入指令寄存器。这就是我们已经讨论的**数据交换**的第一个例子。

数据交换——两个或两个以上设备产生信号，并由这些设备识别信号，从而数据按顺序可以在各设备之间传递。

中断响应 ($\overline{\text{INTA}}$) 信号是 8080 的控制逻辑电路产生的，用来把 1 个，2 个或 3 个字节的指令选通到数据总线，然后进入指令寄存器。如要把一条再启动指令压入指令寄存器，那么，8080 的控制逻辑只产生一个 $\overline{\text{INTA}}$ 信号。如果一条三字节的指令的操作码要送入指令寄存器，那么 8080 的外部控制逻辑还必须产生两个 $\overline{\text{INTA}}$ 信号，选通这条指令的第二和第三个字节进入 8080。同时，中断信号把**内部中断标识禁止**，因此，不接收其他任何中断信号。

如果把一条 RST 指令送入 8080，那么，这条指令被执行后，把一个返回地址保存在堆栈。如果中断还没有出现，那么，这个返回地址指示到已被执行的这条指令后的下一条指令的地址。

当为输入/输出设备服务完后，或者中断信号已经被检测之后，**引起中断的输入/输出设备的外部标识（触发器）必须被清零**。这样，将能防止当外部设备只需要服务一次时，8080不会多次被同一个外部设备中断。这个标识位被清零以后，可以执行一条 EI 指令，以便其他的外部设备可以中断 8080，并且为它服务。最后，在输入/输出设备的中断服务子程序的末尾，执行一条 RET 指令，8080 才能返回到已被中断的程序指令。8080 微处理器集成电路的特点之一是：**即使一条 EI 指令被执行，也得在下一条指令被执行之后，才允许中断**。这就是说，如果在 EI 指令之后，8080 执行一条 RET 指令，那么，在另一台外部设备可能中断 8080 微型计算机之前，8080 将从中断服务子程序返回。

8080 在中断服务子程序的开始，还是末尾，执行这条 EI 指令，取决于微型计算机的中断引线所连接的输入/输出设备的性能特点。此外，可以在中断服务子程序的末尾或者开始，把产生中断的标识位清零。

单线中断(查询中断)

在查询中断操作方式中，需要中断微型计算机的输入/输出设备与一条公用中断线连接。当中断出现时，微型计算机必须查询每一台外部设备，确定哪一台设备产生了中断信号，(需要服务)。这种查询很容易进行，只要从高达8个输入/输出设备读标识位(在一个八位的字)即可。可以执行累加器输入/输出指令或存储器映象输入/输出指令来实现单线中断。

采用循环移位指令或屏蔽，很容易确定哪台设备产生了中断信号。以8080微处理器为基础的微型计算机可以采用这种中断方式；但是它应该必须与一个向量中断结合起来使用。莫托罗拉公司的6800，MOS技术公司的6502和英特西尔公司的6100常常使用查询中断方式。

向量中断

大多数微型计算机有许多重要的事情需要做，不只是监控1位标识位。在某些情况下，需要检测数百个标识位。而在另一些情况下，微计算机正在做复杂的算术计算，需要相当多的计算时间。现在假设，通过接口已经把键盘和8080微型计算机连接起来了。问题在于大多数微型计算机与外部设备采用这样一种方式连接，结果，只有当键盘上的一个键被按下时，微型计算机才响应；而在其它时间，对键盘的请求置若罔闻。一个优秀的打字员每秒钟可以打字符5～10个，即每打印一个

ASCII 字符需要 100～200 ms。按照微型计算机的标准来说，这是很慢的；在敲键之间的时间，微型计算机可以做其它许多工作。但是，一旦一只键被按下，微型计算机要立即响应，才是有用的。对 8080 微处理器为基础的微型计算机而言，采用向量中断，可以达到"立即"响应的目的。

向量中断被定义为这样的中断系统：中断引起程序直接转移到为该中断服务的程序。ASCII 键盘与 8080 的中断引线连接，所需要的附加电路如图 2-3 所示。使用 8212"输入端口"，把 RST 5 指令（八进制 357，十六进制 EF）的 8 位操作码压入指令寄存器，这个问题，现在让我们来加以说明。该键盘的"VALID"输出信号用来中断 8080 微型计算机；8080 的控制逻辑产生中断应答信号($\overline{\text{INTA}}$)用来控制 8 位操作码，进入数据总线，然后送入指令寄存器。8212 就是一个 8 位的三状态输入端口。8212 用来控制数据总线上的 RST 5 指令，正如使用 SN 74126、DM 8095 或 SN 74365 集成电路器件一样方便。

8080 微处理器芯片接收一个中断脉冲时，而且，如果该芯片的这个中断标识位已经先被一条允许中断指令 EI 打开了，那么，**8080 执行完了现行指令，然后产生一个中断响应信号**($\overline{\text{INTA}}$)，**该信号被用来控制一个字节指令的 RST 5（八进制 $_3$57），直接送到 8080 芯片内的指令寄存器**。就在这时，利用外部的三状态器件，可以直接把指令输入到指令寄存器，而不是累加器或其它通用寄存器。实际上，把指令的操作码送入指令寄存器。用来把一个指令字节送入指令寄存器的硬件电路框图如图 2-3 所示。它包括一片 8 位的选通驱动器芯片，名叫**中断指令寄存器**。如图 2-3 所示，被送入的指令是 RST 5（八进制 357。十六进制 EF），这条指令使 8080 微计算机调入起始地址为0 00050(0028)的子程序。

让我们来研究一下向量中断系统所需要的软件。因为RSTn指令是一种调入子程序的指令，所以，读者首先必须把一个读/写存储器地址装入堆栈指示器寄存器。然后，执行允许中断指令EI，允许加在8080的INT引线上的外加信号中断8080微计算机。接着，8080转移到我们称之为主任务(MAIMTASK)的存储器段。MAIN TASK与EPROM(电可改写的只读存储器)或ROM的主程序很相似，它将可以周期地被中断。MAIMTASK可以被存储在存储器的任何部分，但是应该最好远离中断服务子程的存储区；MAIN TASK的存储器地址从000000(0000)开始，延续到大约000077(003F)地址单元。但是，请读者注意，RST 7的中断服务子程序可以根据需要多长，就编写多长，因为它将不会"跑入"其它任何再启动指令所调用的地址。

图2-3所示的ASCII键盘可以使用的程序，列在例2-2。

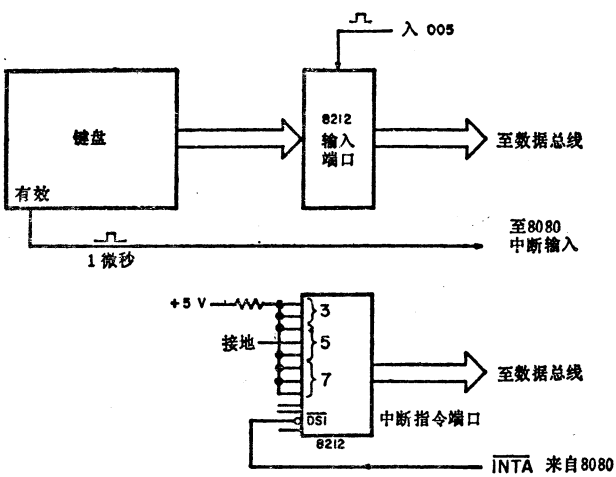


图 2-3 ASCII键盘的简单向量中断

我们已经假设 ASCII 键盘内的电路具有消除键闭合时所产生的颤动的功能。

例 2-2 向量中断服务的 ASCII 键盘程序

```
        *000000
START,  LXISP /把1个读/写存储器地址
        STACK/装入堆栈指示器里。
        0
        EI      /允许中断,
        JMP     /然后执行"MAIN TASK"程序
        MTASK
        0


        *000050
SERVIC, IN      /键盘上的一只键被按下。
        005     /因此,输入这个 ASCII 字符。
        MOVBA   /把这个字符存入 B 寄存器里。
        RET     /然后返回到"MAIN TASK"程序
```

键盘发出的中断将会把一条 RST 5 指令送入指令寄存器，这应该是很清楚的。出现这种情况，是因为 8080 产生了一个 $\overline{\text{INTA}}$ 信号。当执行这条指令时，调用键盘中断服务子程序，它被存储在存储器，起始地址是000050(0028)。该子程序以一条 RET 指令结尾，这条指令允许 8080 微型计算机返回到"MAIN TASK"程序；当该中断发生时，正在执行 MAIN TASK 程序。MAIN TASK 可以是一个简单的控制循环程序，也可以是一个复杂的算术运算程序。因为键盘中断服务子程序很短，所以，8080实际为键盘服务所用的时间很少。读者注意，没有查询键盘标志位。

前面这个程序能工作，但是，如果你要执行这个程序，会看到有许多操作困难。第一，不能多次执行这个键盘中断服务子程序。为什么呢？因为你没法再启动8080芯片内的中断标识位。请读者记住下述规则：

在一个中断机器周期时间，8080芯片上的内部中断标识首先被禁止。然后中断应答信号$\overline{\text{INTA}}$产生，以允许一条指令送入指令寄存器。

这里要指出的关键问题是：8080微型计算机正在为现行中断服务时，这个中断标识位被禁止，防止检测再产生的中断。**如果你希望再启动中断标识位。你必须在中断服务子程序提供一条EI指令。**中断标识位不是自动地被启动的。只有把一个外加的RESET(复位)脉冲施加给8080，中断标志才能被复位或者禁止。

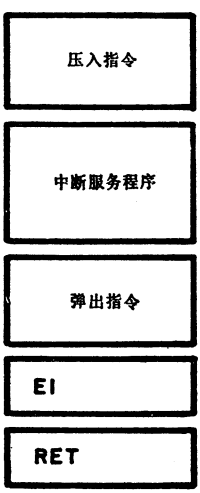


图 2-4 典型的中断服务子程序的方框图

通常是在中断服务子程序的最后一条指令RET之前编一条启动中断指令EI。因为**中断标识位只有在下一条指令被执行完后才启动的。所以8080微处理器在接收另一个中断标识位之前能够返回到MAIN TASK程序。**如果没有这种能力，则也会存在着8080用返回地址去填读/写存储器的危险，因为在8080有时机返回MAIN TASK程序之前，中断标识会把中断服务子程序中断。

在其它各方面，可把中断服务子程序作为正规子程序处理。如果这些寄存器的内容是重要的，你用PUSH和POP这两条指令保存和恢复这些寄存器的内容。典型

的中断服务子程序的方框如图 2-4 所示。请读者注意，这条 EI 指令正好位于 RET 指令之前。

因为在这个向量地址 000050(0028) 和下一个向量地址 000030(0030)之间只有 8 个存储单元，所以，如果不侵占下一个或两个矢量子程序单元，在这个中断服务子程序里，读者将不可能填入四条 PUSH 指令，四条POP 指令，一条 EI 指令，一条RET指令和键盘服务指令。因而你必须安排一条转移指令，把它存储在从000050(0028)地址单元开始的单元；由这条指令把程序执行控制转移到有更多的存储程序的空间的存储区。在中断服务子程序的末尾，RET指令将仍旧要把程序执行控制送回到 MAIN TASK 程序的断点。MAIN TASK 程序(在 000050 (0028)地址单元的向量转移指令)和键盘中断服务子程序之间的关系如图 2-5 所示。这种键盘的接口是比较复杂的；例如，可以使用优先权中断。在前一个例子中，这样做没什么促进作用。

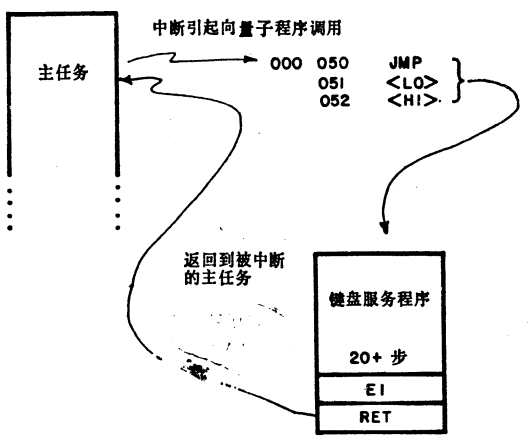


图 2-5 转移到中断服务子程序

向量中断和查询中断

8080微型计算机的许多用户采用向量中断，其中，与这条中断信号线连接的每一台外部设备分别提供专用的 RST 指令。但是，因为8080只能执行 8 条 RST 指令，所以，能使用 RST 指令的外部设备的数量受到限制。而且，为了把适当的 RST指令选通到数据总线，每一台外部设备必须使用一个 8 位的三状态驱动器。因此，有些用户采用查询中断，这正是因为它们所需要的硬件较少。连接三个查询中断设备的接口电路，如图2-

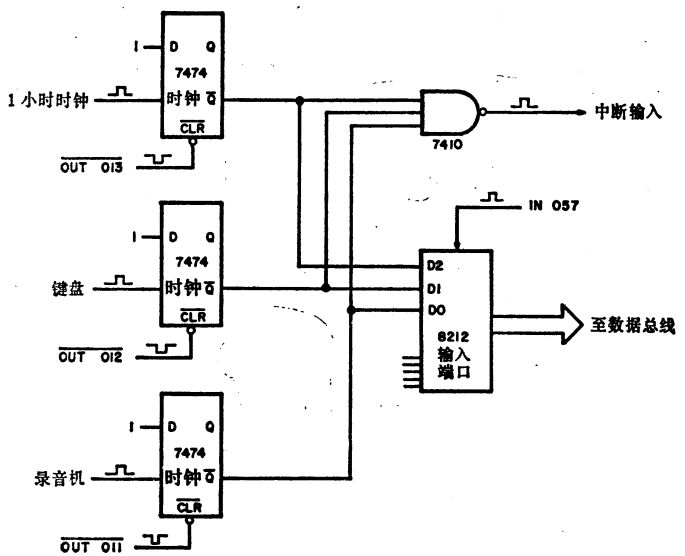


图 2-6　三台查询中断设备的接口(使用一条 RST 指令)

6 所示。我们假设了 ASCII 键盘不再连接到与 8080 的中断引线上。我们还假设“中断指令端口”仍旧与8080的$\overline{\text{INTA}}$信号线连接，这样，中断出现时，一条 RST 5 指令被送入指令寄存器。读者欲知详细情况，请参考图 2-3 。

如图 2-6 所示，共用这条中断输入线的三台外部设备是：高速盒式录音机，中速键盘和极低速时钟。假设，录音机的标识位表示：可以从录音机输入一个 8 位字。键盘标识位表示：一只键被按下。一小时时钟标识位表示：自从时钟最后产生一个时钟脉以来，已经过去了一小时。

查询这些设备的程序，列于例 2-3 。在程序表中，只包括录音机的中断服务子程序。

在 START 处，堆栈指示器装入一个读/写存储器地址。这样做是因为8080响应中断时，将把 RST 5 送入指令寄存器里。执行这条 RST 5 指令时，返回地址保存在堆栈，所以，堆栈区一定是有效的。然后这三条输出指令把三个中断标识位清零。三个标识位被清零之后，8080执行 EI 指令，这条中断线被启动，然后，执行 MAIN TASK 程序。

中断服务子程序被存储在存储器，始于000050(0028)存储器单元。因为当8080响应答断时，接口电子器件把 RST 5 指令送入指令寄存器，所以，中断服务子程序被存储在这里。当中断确实出现时，程序执行控制被引导到000050(0028)地址单元。然后输入标识位，存储在 B 寄存器里。使用这样接口器件，8080经过查询或检查各个标识位，能确定哪一台设备产生了中断。

中断标识位被保存在 B 寄存器之后，第一条 ANI 指令只测试 A 寄存器所包含的录音机标识位。如果该标识位是逻辑 0 ，则执行 JZ-CASSRD(录音机读)指令。如果录音机标识位

是逻辑1，那么，录音机还没有产生中断信号，所以把这些标识位从B寄存器送到A寄存器。然后，下一条ANI指令只检测键盘标识位。如果这个键盘标识位是逻辑0，那么，执行JZ-KEYIN指令执行，因此，输入键盘发出的ASCII字符。如果键盘的标识位是逻辑1，那么，查询1小时时钟的标识位。如果时钟的标识位是逻辑1，那么8080执行HLT指令。如果时钟的标识位是逻辑0，8080则执行JZ-CLOCK指令。

例 2-3 查询三台外部设备的中断服务子程序

```
          *000000
START     LXISP      /把一个读/写存储器地址
          RWMEM      /装入堆栈指示器。
          0
          OUT        /把录音机的中断标识位清零
          011        /(十六进制09)。
          OUT        /把键盘的中断标识位清零
          012        /(十六进制0A)
          OUT        /把1小时时钟的中断标识位清零
          013        /(十六进制的0B)。
          E1         /开中断。
          .          /于是，执行MAIN TASK程序。
          .
          .
          *000050
SERVIC    IN         /输入与中断引线连接的外部设备
          057        /的标识位(十六进制2F)。
          MOVBA      /把这些标识位保存在B寄存器。
          ANI        /只保存录音机的标识位
          001        /(十六进制01)。
          JZ         /标识位是逻辑0，所以
```

```
          CASRD     /以录音机读数据。
          0
          MOVAB     /取回这些标识位。
          ANI       /只保存键盘的标识位
          002       /(十六进制的 0 2)。
          JZ        /该标识位是逻辑 0，所以
          KEYIN     /输入这个 ASCII 字符。
          0
          MOVAB     /取回这些标识位。
          ANI       /只保存 1 小时时钟的标识位
          004       /(十六进制 0 4)。
          JZ        /该标识位是逻辑 0，所以
          CLOCK     /把灯点亮，切断加热器。
          0
          HLT       /什么引起的中断?

CASRD     OUT       /清除录音机的标识位，
          011       /它产生了中断(十六进制 0 A)。
          IN        /输入录音机的数据
          103       /(十六进制 4 3)。
          MOVMA     /把它保存在存储器里。
          INXH      /存储器地址加 1。
          EI        /开中断。
          RET       /然后返回。
```

在为8080编程序时，为什么要在查询程序的末尾要执行一条 HLT 指令呢？因为有些外部设备中断了 8080，所以要执行这条 HLT 指令；但是，当8080查询了外部设备，确定哪台外部设备需要服务时，却没有任何标识位处于适当的状态。这种

情况表示接口硬件有故障。很遗憾，这个问题产生的原因可能是诸如下述情况：电源滤波不当，或不充分，电源屏蔽差，微型计算机输入端有噪声，接地不当，等等。可是给这类问题定位可能很困难。可以通过程序设计不让微处理机暂停，而把UNIDENTIFIED INTERRUPT(未识别中断)打印在CRT上，或者电传打字机上。请读者参考«8080/8085的软件设计»一书的上册第三章(从例3-19开始)。

如果在这个例子中，录音机的标识位是逻辑0，则执行JZ-CASRD指令。在CASRD处，一条输出指令把录音机的中断标识位清零。然后把录音机的这个数据字输入到A寄存器，接着存入存储器。然后这个存储器地址加1。这条EI指令允许中断。但是，只有到这条RET指令已被执行完毕之后，它才真正允许中断。当这条RET指令执行时，8080允许中断，且返回到中断出现时正在执行的MAIN TASK程序段。

当一条指令被送入数据总线时，8080响应该中断。但是，读者可能会问，地址总线上的这个16位地址所寻找的存储器单元的内容，为什么不会干扰与接口电子器件送入到指令寄存器的指令呢？8080响应中断时，与存储器连接的$\overline{\text{MFMR}}$和$\overline{\text{MEMW}}$信号线都没有加脉冲信号。因此，存储器根本没有数据信息置于数据总线上。当中断响应时，前一条指令已经执行完毕。因此当$\overline{\text{INTA}}$信号产生时，8080认为它正在从存储器取一条指令；但是，这条指令实际上是来自中断设备的接口电子器件，是$\overline{\text{INTA}}$脉冲信号控制的而不是$\overline{\text{MEMR}}$！

读者从编写的中断服务子程序(如例2-3)中发现了什么问题吗？当中断出现时，这些问题一定与寄存器的内容相关。当中断出现时，B寄存器或者寄存器对H中的内容是什么？这是没有办法知道的，因为中断可能随时出现。为了解决这个问题，

寄存器对B和H的内容应该保留在堆栈，这取决于KEYIN和CLOCK这两个子程序所用的寄存器。例2-4里所列出的这个中断服务子程序，把这些寄存器对的内容保存在堆栈。

例 2-4 查询三台外部设备用的改进的中断服务子程序

```
          *000050
SERVIC,   PUSHPSW     /把A的内容和标识位保存在堆栈。
          PUSHB       /再把寄存器对B的内容保存在堆栈。
          IN          /把与中断信号线连接的外部设备
          057         /的标识位输入(十六进制的2F)。
          MOVBA       /把这个标识位保存在B寄存器里。
          ANI         /只保存录音机的标识位
          001         /(十六进制的01)。
          JZ          /该标识位是0，所以
          CASRD       /读录音机的数据
          0
          MOVAB       /取回这些标识位。
          ANI         /只保存键盘的标识位
          002         /(十六进制02)
          JZ          /这个标识位是逻辑0，所以
          KEYIN       /输入这个ASCII字符。
          0
          MOVAB       /取回这些标识位。
          ANI         /只保存1小时时钟标识位
          004         /(十六进制的04)。
          JZ          /这个标识位是逻辑0，所以
          CLOCK       /亮灯，切断加热器。
          0
          HLT         /什么引起了中断?
```

```
CASRD,  PUSHH    /把寄存器对 H 的内容压入堆栈。
        LHLD     /把存储在两个连续存储器单元
        POINT    /的一个地址装入
        0        /寄存器对 H 里。
        OUT      /然后，把录音机的标识位清零，
        011      /该标识位引起中断(十六进制的 0 A)。
        IN       /从录音机输入数据
        103      /(十六进制43)。
        MOVMA    /把这个数保存在存储器里。
        INXH     /这个存储器地址加 1 。
        SHLD     /然后把这个新地址保存在
        POINT    /两个连接存储器单元。
        0
        POPH     /把寄存器对 H 的内容弹出堆栈。
        POPB     /把寄存器对 B 的内容弹出堆栈。
        POPPSW   /把A的内容和标识位弹出堆栈。
        EI       /允许中断，
        RET      /然后返回。
```

当中断出现时，8080的程序执行控制被引导到这个新中断服务子程序(例 2-4)，始地址是000050(0028)。8080微型计算机立即把 PSW 和寄存器对 B 的内容保存在堆栈。然后分别查询每台输入/输出设备的标识位。假设录音机请求服务，那么，录音机的标识位将是逻辑 0 。因此 8080 转移到 CASRD。在 CASRD 处，寄存器对H的内容被压入堆栈。然后把16位地址装入寄存器对H，该地址原来存储在读/写存储单元；它被分配给 POINT 符号地址。然后这个中断标识位被清零，从录音机读出这个数据字；接着，把这个数据字存入寄存器对H寻址的寻储器单元。寄存器对H中的地址加 1 ，然后存回到读/写

存储器里。接着从堆栈弹出寄存器对H和B的内容以及PSW，这条中断信号线被再启动。最后，RET指令把程序执行控制送回到原来被中断的MAIN TASK程序段。

中断服务子程序对于已被中断的程序来说，似乎是完全透明的，因为来自MAIN TASK程序的存储在任何寄存器的任何数据都没有扰乱。当8080执行中断服务子程序时，把所需要的数值保存在堆栈，然后，在程序执行控制返回到MAIN TASK之前，这些数据从堆栈弹出。维持"堆栈的适当状态和馈送"始终是很重要的。请读者注意，为了正确操作，还必须用POPB POPPSW，EI，和RET指令序列来结束KEYIN和CLOCK这两个子程序。

优先权在查询程序软件中已经建立了吗？在中断服务子程里，什么设备的优先权最高呢？录音机的优先权最高，就是因为它的标识位首先被检测。时钟的优先权最低，键盘的优先权位于这两种设备之间。如果在8080微型计算机的中断信号线上再连接一个软磁盘，那么它的优先权的等级怎么样呢？因为软磁盘发送/接收数据的速度比录音机的高，所以软磁盘的优先权应该最高。假设把软磁盘的一个标识位分配给被查询的输入端口的D_3位，那么例2-5所列出的中断服务子程序可以用来查询这四台外部设备。

优先权中断

在前面的中断程序例中，每台设备的优先权都是由它被查询问的顺序决定的。优先权等级较高的外部设备较先查询，优先权等级较低的外部设备较后查询。

优先权中断——优先权中断是按照重要性来规定次序进行的中断，所以，有些中断设备优先于其他的设备。当几个中断可能同时出现时，或者有必要确定哪些中断设备是最重要的时，我们就要使用优先权中断。

正如读者已经看到的那样，改变由程序软件查询的输入/输出设备的优先权是很简单的；软件程序查询外部设备的顺序必须予以改变。

例 2-5 给中断服务子程序(例 2-4)增加一台较高优先权设备的程序

```
          * 000050
SERVIC,   PUSHPSW     /把A的内容和标识位保存在堆栈。
          PUSHB       /把寄存器对B的内容保存在堆栈。
          IN          /输入中断信号线连的设备的
          057         /标识位(十六进制 2F)。
          MOVBA       /把标识位保存在B寄存器里。
          ANI         /只保存软磁盘的标识位
          010         /(十六进制08)。
          JZ          /这个标识位是逻辑 0，所以
          FLOPPY      /从软磁盘读出读据。
          0
          MOVAB       /取回标识位。
          ANI         /只保存录音机的标识位
          001         /(十六进制01)。
          JZ          /这个标识位是逻辑 0，所以
          CASRD       /从录音机读出数据。
          0
          MOVAB       /    取回标识位。
          ANI         /只保存键盘的标识位
          002         /(十六进制02)。
```

```
JZ          /这个标识位是逻辑0，所以
KEYIN       /输入B这个ASCII字符。
0
MOVAB       /取回标识位。
ANI         /只保存1小时时钟的标识位
004         /(十六进制04)。
JZ          /这个标识位是逻辑0，所以
CLOCK       /亮灯，切断加热器。
0
HLT         /什么原因引起的中断？
```

硬件优先权中断

使用硬件也可以产生中断优先权。当许多台中断设备与8080微型计算机连接，并且都需要8080较快地为它们服务时，硬件优先权中断是很重要的。所采用的“诀窍”(指程序指令；译者注）是简单的。**每一台中断设备产生它自己的再启动指令RSTn，当RSTn指令被写入8080的指令寄存器时，它把立即向量送到下列存储器单元：000000，000010，000020，000030，000040，000050，000060，000070，(十六进制的0000，0008，0010，0018，0020，0028，0030，0038)**。另外，优先权是由硬件自动地指定的，因此第七号外部设备的优先权比第6号的高，第六号的比第五号的高，等等。换句话说，如果用这个“>”符号表示优先权，则有：7>6>5>4>3>2>1>0。

读者习惯供硬件优先权向量中断用的电路，如图2-7所示。

74148 3-8优先数译码器集成电路芯片有16条引线，其引线结构和真值表如图2-8所示。请读者注意，逻辑低电平时，

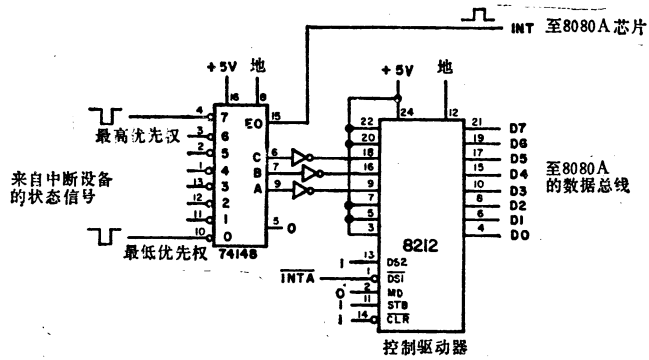


图 2-7　8级硬件优先权，向量中断电路，
（产生 8 条不同的再启动指令）

数据输入和输出是有效的。74148 芯片将从标识位接收 8 个逻辑 0 输入。如图 2-7 所示；**当最高编号的输入端子是逻辑 0 时，74148芯片将输出该输入端的二进制代码**。例如，如果第五号和第七号设备同时发生中断请求，第七号设备的优先权较高。74148芯片和反相器将给 3 n 7 这条再启动指令中间这位八进制数提供八进制数 7 。这样，就把8080引导到000070(0038)号存储器单元。RSTO 指令并不常用，因为它的唯一作用是使8080微型计算机复位和再启动 MAIN TASK 程序。

为了简明起见，图2-7是简化了的电路图。必需的标识位和标识位清零线都没有示出。为了使这个电路更加完善，我们还可以给它增添经改进的硬件。这个电路应该还要包含一块7442译码电路(不用 OUT 指令产生标志位清零脉冲) 和一个屏蔽寄存器，这样，经过外部硬件，几台外部设备可以屏蔽或接通(禁止或允许)。这个改进了的电路如图2-9 所示。它是一个很完善的优先权中断接口，与8080微型计算机连接，用于向量中断，具有很大的灵活性。

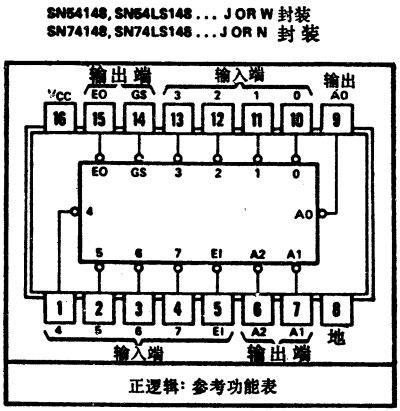


图 2-8　SN 74148 优先权编码器集成电路引线结构与真值表

'148,'LS 148 功能表

输入									输出				
E1	0	1	2	3	4	5	6	7	A2	A1	A0	GS	E0
H	X	X	X	X	X	X	X	X	H	H	H	H	H
L	H	H	H	H	H	H	H	H	H	H	H	H	L
L	X	X	X	X	X	X	X	L	L	L	L	L	H
L	X	X	X	X	X	X	L	H	L	L	H	L	H
L	X	X	X	X	X	L	H	H	L	H	L	L	H
L	X	X	X	X	L	H	H	H	L	H	H	L	H
L	X	X	X	L	H	H	H	H	H	L	L	L	H
L	X	X	L	H	H	H	H	H	H	L	H	L	H
L	X	L	H	H	H	H	H	H	H	H	L	L	H
L	L	H	H	H	H	H	H	H	H	H	H	L	H

为了使用如图2-9所示的中断控制器，读者必须首先确定允许哪些设备能中断8080微型计算机，哪些则不能中断8080。读者应画一个 8 位屏蔽图形，把逻辑 1 分配给发出中断请求的

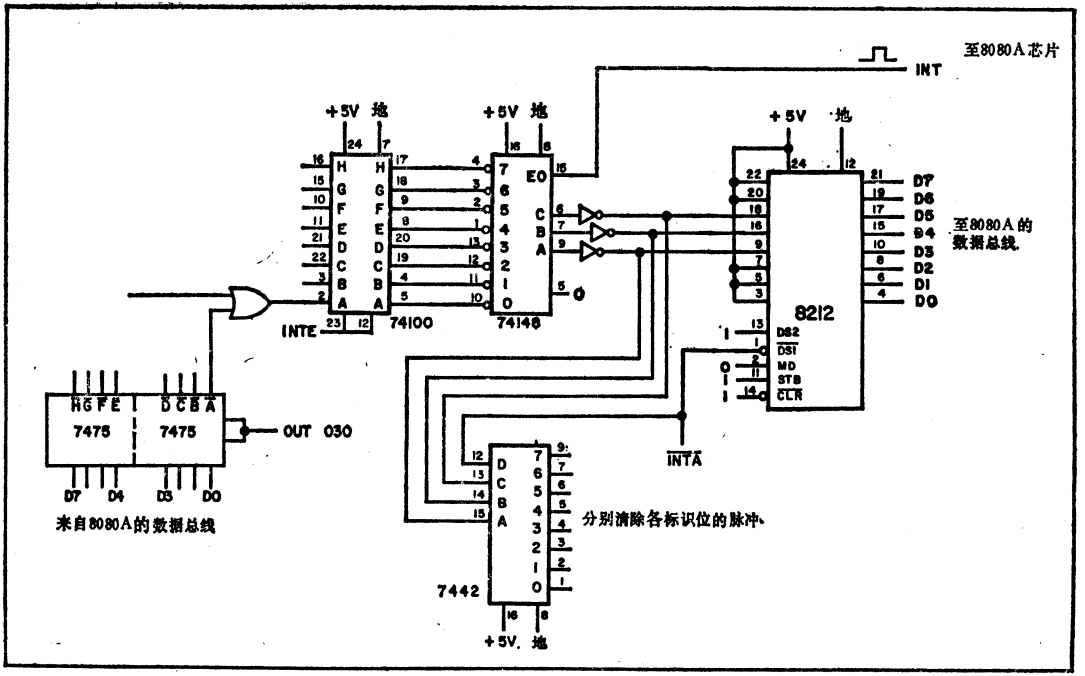


图 2-9　复杂的八级硬件优先权、向量中断控制器

外部设备，逻辑0分配给没发出中断请求的外部设备。然后8080把这8位信息输出给7475锁存器，(如图2-9的左边所示)。我们就是为了这个目的而使用一条OUT指令(030)。D7位对应于第七号中断设备，该设备的优先权最高，能够产生地址向量000070(0038)。如本章前一部分所述，被屏蔽的这些设备将可能使用状态寄存器输入端口，来请求服务(参考有关查询的章节)。

各个标识位的中断请求是由"或门"控制的(图2-9的左边示出了一个"或门")；"或门"的输出端与8位锁存器74100连接。当8080芯片上的中断标志位被启动时，INTE(中断允许，8080芯片输出的信号)是高电平，于是启动74100芯片。74148优先权编码器和中断指令端口的操作，我们在前面已经描述过了。

当中断信号被8080微处理器接收时，它禁止该中断信号，INTE输出为逻辑0，从而把74100芯片上的任何中断信号都锁存起来。$\overline{\text{INTA}}$信号不仅输入RSTn指令的操作码，**而且控制7442译码器，从而产生一个脉冲，将触发器清零；该触发器与正在被服务的向量中断设备连接。**

例2-6所列的程序假设：上述的1小时时钟、键盘和录音机这三台设备仍然与8080微型计算机连接。现在已经分配给每台设备不同的优先权和再启动指令。1小时时钟把一条RST 5送入8080微型计算机，键盘把RST 6送入8080，录音机把RST 7送入8080。这三台设备的中断服务子程如例2-6所示。

在这个程序的开始，把一个读/写存储器地址装入堆栈指示器。然后，8080执行三条OUT指令，把这三个标识位清零，接着允许中断。8080然后执行MAIN TASK(主程序)的其余各条指令。当中断出现时，每台中断请求设备将把它自己的再启动指令送入8080的数据总线，从而8080就能把程序执行控制引导到适当的中断服务子程序。

例 2-6 三台向量中断外部设备的子程序

```
    * 000000
START, LXISP          /把读/写存储器的一个地址
       RWMEM          /装入堆栈指示器。
       0
       OUT            /把录音机的中断标志位清零
       011            /(十六进制09)。
       OUT            /把键盘的中断标识位清零
       012            /(十六进制0 A)。
       OUT            /把 1 小时时钟的中断标志位清零。
       013            /(十六进制 0 B)。
       EI             /允许中断，然后
       •              /执行"MIAN TASK"
       •              /程序的其余指令。

    *  000050
CLOCK, OUT            /把 1 小时时钟的标志位清零
       013            /(十六进制 0 B)。
       EI             /然后向其它设备开中断
       PUSHPSW        /把 A 的内容和标识位保存在堆栈。
       JMP            /然后执行时钟中断服务子程序
       CLK1           的其余指令。
       0
    *  000060
KEYIN1, PUSHPSW       /把 A 的内容和标识位保存在堆栈。
       IN             /输入 8 位 ASCII 键盘代码
       000            /(十六进制00)。
       ANI            /然后从该代码中去掉
       177            /任何奇偶位(十六进制 7 F)。
       JMP            /然后执行键盘输入
```

```
        KEYIN1          /服务子程序
        0               /的其它指令。

        *000070
CASRD,  PUSHPSW         /把 A 的内容和标识位保存在堆栈。
        PUSHH           /把寄存器对 H 的内容保存在堆栈。
        LHLP            /把存储在两个连续
        POINT           /存储单元的一个地址
        0               /装入寄存器对 H。
        OUT             /把录音机的标志位清零，
        011             /该标志位引起中断(十六进制09)。
        IN              /以录音机输入这个数据，
        103             /(十六进制43)
        MOVMA           /把该数保存在存储器。
        INXH            /存储器地址加1。
        SHLD            /然后把这个新地址保存在
        POINT           /两个连接存储单元。
        0
        POPH            /把寄存器对 H 的内容弹出堆栈。
        POPPSW          /把 A 的内容和标志位弹出堆栈。
        EI              /允许中断。
        RET             /然后返回。
CLK1,   IN              /从八个检测开关输入
        034             /这个数据(十六进制 1 C)。
        ANI             /然后只保存这三个开关(D6, D5, D1)
        142             的状态(十六进制62)。
        CPI             /各个开关的状态
        142             /都是逻辑 1 吗?
        JNZ             /不，不是逻辑1。让加热器接通，
        LVEON           /加热箱
```

```
        0               /继续加热。
        OUT             /加热箱足够热了，
        101             /灯点亮，
        OUT             /然后关断加热器
        102             /(十六进制41和42)。
LVEON,  POPPSW          /从堆栈弹出 A 的内容和标志位。
        RET             /然后返回。
KEYIN1, OUT             /把键盘的中断识志位清零
        012             /(十六进制 0 A)。
        CPI             /判明这个键盘代码是否 D 键代码，
        104             /即减 1 命令。
        JZ              /它是减1命令，
        DEC             /所以 H 中的地址减1。
        0
        CPI             /这个键代码是否 1 键代码，
        111             /即加1命令。
        JNZ             /它不是1键，
        NOTINC/         则忽略这个命令。
        0
        INRH
NOTINC,POPPSW           /从堆栈弹出 A 的内容和标识位。
        EI              /允许中断，
        RET             /然后返回。
DEC,    DCRH            /H 中的地址减1。
        POPPSW          /从堆栈弹出PSW。
        EI              /允许中断，
        RET             /然后返回。
```

1 小时时钟的中断服务子程序的始地址是000050(0028)。一小时时钟必须把什么再启动指令送入8080微型计算机呢？ 1

小时时钟必须与中断控制器或接口电路连接，从而才能把一条RST 5 指令送入8080微型计算机。键盘的中断优先权比 1 小时时钟的中断优先权高，所以，它把RST 6 指令送入8080微型计算机。因此，它的中断服务子程序的始地址是000060(0030)。因为录音机的中断优先权最高，所以，它需要服务时，它把8080引导到000070(0038)地址单元。

如果 1 小时时钟中断8080微型计算机，那么该机引导到000050(0028)地址单元。从这个地址开始，8080把 1 小时时钟的中断标志位清零，然后再启动中断。这就是说，如果需要的话，键盘或录音机可以中断8080为时钟的服务。8080允许中断后，把 PSW 保存在堆栈，然后转到在 CLK 1 的时钟中断服务子程序。时钟的整个中断服务子程序不能存储在000050(0028)和000060(0030)之间的地址单元，所以，在这两个地址之间存储一些指令，而其余的指令被存储在存储器的其它部分。请注意，JMP 指令并不存储在000050(0028)地址单元，所以程序执行控制不能立即传送到中断服务子程序。

当8080转移到 CLK 1 时，执行该子程序的其余指令。我们分配给 1 小时时钟的任务是随机选择的。在 CLK 1，八个开关的状态被输入到 A 寄存器里。然后把与 D_6，D_5 和 D_1 连接的三个开关的状态保留在A寄存器。如果这三个开关的状态都是逻辑 1，那么，某个液体箱的加热器已经被接通了足够长的时间。如果一个开关，两个开关或三个开关的状态都是逻辑 0，那么，加热器应该继续接通。

经过多长时间之后，8080才再次检测这些开关的状态呢？经过 1 小时，因为这个时钟 1 小时产生 1 次中断。

这个箱的温度是什么？甚至该箱内是什么都没有讨论的必要。这两个控制功能被执行之后，A 寄存器的内容和标志位都

从堆栈弹出，然后8080返回到 MAIN TASK 程序。

因为1小时时针的中断优先权最低，所以，当8080的程序执行控制引导到000050(0028)号地址单元时，时钟的标识位被清零。该标识位被清零后，再次允许中断。**只有在时钟的标识位被清零之后，才能再次允许中断，认识到这一点是十分重要的。**

与8080微型计算机连接的 ASCII 键盘的中断服务子程序是KEYIN 子程序。当一个键被按下时，8080被中断，它的程序执行控制被引导到000060(0030)号地址单元。然后8080把 A寄存器的内容和这些标识位保存在堆栈，接着，将该键盘的这个8位 ASCII 字符输入到 A 寄存器。这条 ANI 指令把校验位(D_7)置0。这条 JMP 指令把程序执行控制送到该子程序的其余部分。此外，我们不把 JMP 指令正好存储在 000060(0030)号地址单元，而是通过实际执行几条中断服务指令，可以更好地利用几个存储单元。如果简单地把 JMP 指令存储在 000060(0030)地址单元，在 JMP 指令的高位地址字节和下一个优先权较高的设备的中断服务子程序之间，则有五个存储器单元无用。因此，在这个地方，实际执行一些指令是比较有效的。

当8080转移到 KEYIN1时，键盘的中断标识位清零，8080把 A 寄存器的内容与 D 键和 1 键的 ASCII 字符值进行比较。如果键盘上的 D 键已经被按下，那么，H 寄存器的内寄减 1。如果 I 键被按下，H 寄存器的内寄加1。这是在键盘上的 D 键或 I 键被按下之时，我们要8080执行的随机任务。如果这些键都没有被按下，那么，8080把 A 寄存器的内寄和这些标识位都弹出堆栈，再允许中断，然后返回到 MAIN TASK 程序。注意，录音机不能中断8080为键盘的服务，因为 8080执行 EI 指令是在中断服务子程序的末尾。

录音机中断服务子程序与其它两个中断服务子程序略有不同。我们已经给录音机分配了一条 RST7 指令，所以，没有任何外部设备的优先权比它的高。因此，这个中断服务子程序。可以从000070(0038)号地址单元开始，存入存储器。可以从这个地址开始，把整个中断服务子程序都存储在存储器里，因为没有任何外部设备能够把8080引导到比它更高的地址单元。这个 CASRD 子程序与我们在本章前面的章节中讨论过的查询中断服务子程序很相似。唯一的差别是该中断服务子程序没有 PUSHB 和 POPB 这两条指令。

如果图 2-9 所示的中断硬件用来把 1 小时时钟，键盘和录音机这三个外部设备与8080的中断信号线连接，那么，例2-6的程序应该作什么改变呢？8080 应该在 MAIN TASK 程序的开始执行一些指令，清除所有的中断标志位。我们必须这样做，因为当电源首先施加到8080微型计算机系统时，触发器的状态是不确定的。所以，8080执行若干条 OUT 指令，每一条 OUT 指令分别把触发器清零。在中断服务子程序中，不必执行任何 OUT 指令，因为中断硬件7442产生脉冲，这些脉冲能够用来给这些标识位清零。这就是说，每个中断服务子程节省 5 μs时间。

8085与中断

Intel(英特尔)公司设计8085时，在8080的基础上增添了一些优良的中断性能。8085 集成电路芯片上有五条中断输入引线，其中之一是 INTR。INTR 这条输入引线的功能与8080微处理器芯片上的 INT 输入引线的一样。如果执行 EI 指令，

8085允许中断。把这条INTR引线上的逻辑电平变为1，8085将被中断。8085在$\overline{\text{INTA}}$引线上产生一个逻辑0脉冲信号，从而，表示8085将响应该中断。这个信号可以用来把指令选通到数据总线，然后进入到指令寄存器(IR)。

8085还有四条向量优先权中断信号线，它在8085微型处理器集成电路的右边。当这些中断引线上的电平置到逻辑1时，8085调用的中断服务子程序的地址列在表2-3里。

表 2-3 8085 的向量优先权的向量地址

中断	向量地址	
	八进制	十六进制
RST 7.5	000074	003C
RST 6.5	000064	0034
RST 5.5	000054	002C
TRAP	000044	0024

8085的五个中断输入优先权如下：TRAP>RST7.5>RST6.5>RST5.5>INTR。这里，我们的讨论仅限于三个中断：RST7.5，RST6.5和RST5.5。

当8085微处理器集成电路上的这些引线之一的电平被置到逻辑1时，一条再启动指令自动写入指令寄存器，然后执行之。这不需要任何外部中断优先权编码器或中断指令寄存器。然后，执行存储在适当向量地址上的子程序，从而8085便为该中断服务。但是，不象一般的RSTn指令操作，我们不能以一般程序中执行或者应用指令的方式去执行RST7.5，RST6.5，或RST5.5这三条指令。8085微处理器集成电路芯片上的三条中断引线，有一条置到逻辑1时，才能执行这三条指令。但

是，正如你所希望的那样，8085被加到RST 7.5，RST 6.5，RST5.5引线的逻辑1，或者被加到INTR引线的逻辑1中断之前，它必须执行一条EI指令。可是，8085可能被RST 7.5，RST6.5或RST 5.5中断之前，我们还必启动被中断屏蔽的各个中断。

这个中断屏蔽位可以用来启动或者禁止三个RSTn.5中断信号的任何组合。当一条SIM(中断屏蔽置位)指令被执行时，把A寄存器的三个最低有效位装入中断屏蔽寄存器。这条SIM指令是8085微处理器所特有的；在8080的指令系统中没有这条指令。A寄存器的D_0位代表RST5.5的屏蔽位，D_1位代表RST6.5的屏蔽位，D_2位代表RST 7.5的屏蔽位。执行这条SIM指令时，如果A寄存器的D_3位是逻辑1，则只把D_0、D_1、D_2的内容传送到中断屏蔽寄存器。这位D_3位是屏蔽启动控制位。读者从第一章知道，这条SIM指令也可以用来把信息位输出到8085的SOD引线。

表 2-4　8085的中断屏蔽位

D_3	D_2	D_1	D_0	中断允许/禁止
1	0	0	0	允许　RST 7.5，6.5和5.5中断
1	0	0	1	RST 7.5和6.5允许中断，5.5禁止中断。
1	0	1	0	RST 7.5和5.5允许中断，6.5禁止中断。
1	0	1	1	RST 7.5允许中断，6.5和5.5禁止中断。
1	1	0	0	RST 6.5和5.5允许中断，7.5禁止中断。
1	1	0	1	RST 6.5允许中断，7.5和5.5禁止中断。
1	1	1	0	RST 5.5允许中断，7.5和6.5禁止中断。
1	1	1	1	RST 7.5，6.5和5.5禁止中断。
0	×	×	×	不改变中断屏蔽位。

×可以是逻辑1，或逻辑0。

为了启动一个中断，屏蔽控制位必须是逻辑0；为了禁止一个中断，屏蔽控制位必须是逻辑1。

当SIM指令被执行时，如果希望改变中断屏蔽寄存器的内容，A寄存器的D_3位必须是逻辑1，如表2-4所示。

让我们用前一个例子(例2-6)来具体阐明中断屏蔽寄存器是怎样置位的。由于8085的新的中断能力，所以，我们对例2-6这个程序才做了这些改变。新的中断服务子程序，列于例2-7。

例 2-7 给8085的中断屏蔽寄存器编程序

```
          *000000
START,    LXISP        /把读/写存储器的
          RWMEM        /一个地址装入堆栈指示器。
          0
          OUT          /清除录音机的中断标识位，
          011          /(八进制011＝十六进制09)。
          OUT          /清除键盘的中断标识位
          012          /(八进制012＝十六进制OA)。
          OUT          /然后清除1小时时钟的中断
          013          /标识位(八进制OB＝十六进制OB)。
          MVIA         /将这些屏蔽位置0(×××1000)，
          010          /启动这三个中断。
          SIM          /把A的内容传送给中断屏蔽寄存器。
          EI           /8085允许中断线，
          .            /然后执行"MAIN TASK"
          .            /的其余指令。
          .
          *000054
CLOCK,    JMP          /转移到1小时时钟
          CLK 1        /的中断服务子程序。
          0
          *000064
```

```
KEYIN,   JMP/          转移到 ASCII 键盘
         KEYIN 1       /的中断服务子程序。
         0
         * 000074
CASRD,   PUSHPSW       /把 A 的内容和标识位存入堆栈。
         PUSH          /然后把寄存器对 H 的内容存入堆栈。
         .             /从录音机取一个数，
         .             /然后把它存入存储器。
         .
         POPH          /从堆栈弹出寄存器对 H 的内寄。
         OUT           /把录音机中断 8085 的
         011           /标识位清零。
DONE,    MVIA          /现在，利用中断屏蔽寄存器内容
         010           /来启动各中断。
         SIM
         POPPSW        /从堆栈弹出 A 的内容和标识位。
         EI            /8085 允许中断，
         RET           /然后返回。
CLK 1,   PUSHPSW       /存储 A 寄存器的内容和这些标识位。
         MVIA          /然后只启动 RST 7.5
         011           /和 RST 6.5(×××1001)。
         SIM           /设置中断屏蔽位。
         OUT           /时钟中断 8085
         013           /的标识位清零。
         EI            /然后启动 8085 开中断
         .             /现在为这些开关，
         .             /加热器和灯服务。
         POPPSW        /恢复 A 寄存器的内容和这些标识位。
         RET           /返回到"MAIN TASK"。
KEYIN 1, PUSHPSW       /存储 A 寄存器的内容和这些标识位。
```

```
IN          /输入这个八位的
000         /ASCII 字符。
•           /然后，把这个字符与 ASCII 字符 I
•           /和 ASCII 字符D或者二者之一比较。
•           /从而采取适当的操作。
OUT         /键盘中断 8085
012         /的标识位清零。
JMP         /然后装入中断屏蔽寄存器。
DONE        /从堆栈弹出A的内容和标识位，
0           /再启动 8085 的中断 线，然后返回到
                "MAIN TASK"。
```

例 2-7 这个程序，在中断服务子程序中，并没有包括外部设备的许多具体指令。但是，为了用适当的值给 8085 的中断屏蔽寄存器编程序，我们已经给中断服务子程序添加了一些指令。

例 2-7 这个程序的开头，把一个读/写存储器地址装入堆栈指示器。我们让 8085 执行三条 OUT指令，分别使三个中断标识位清零。然后把 00001000 装入A寄存器；8085 执行 SIM 指令时，A寄存器的内容（00001000）将启动 RST 7.5，RST 6.5，和 RST 5.5 这三个中断。但是，只有 8085 执行EI 指令时，外设备才能中断 8085 微型计算机。一旦这一操作完成后，8085 执行主程序(MAIN TASK)的其余指令。当中断出现时，8085 或者被引导到 000054 或 000064，或 000074（002 C，0034 或 003C）去执行程序指令。

如果 1 小时时钟中断 8085 微型计算机，那么，8085 被引导到 000054(002C)，8085 从 JMP 转移到 CLK 1 去执行程序。在 CLK 1，8085 把A寄存器的内容和标识位保存在堆栈，然后，把 00001001 装入A寄存器。8085 执行这条 SIM 指令，把

00001001 写入中断屏蔽寄存器，因为中断屏蔽字的 D_3 是逻辑 1。结果是：RST 5.5 中断被屏蔽掉。这样，就防止了 1 小时时钟中断它自己的中断服务子程序的执行。这是很不可能的，因为时钟这个设备每隔一小时才中断 8085 微型计算机一次。然而，在某些情况下，这可能是个麻烦的问题。关于这个问题，我们将在本章的下节进行更详细的讨论。

不管哪条中断线允许还是禁止中断，这个时候，8085仍然不能被键盘或录音机中断，因为内部的中断标识位还没有被再启动。8085 执行 SIM 指令之后，它执行这条 OUT 013 指令，把时钟的中断标识位清零。8085 执行 EI 指令时，时钟中断服务子程序的执行，可以由键盘或者由录音机中断。一旦中断被再行启动，8085 执行原先所定义的控制功能；如；监控开关，使加热器和灯导通或关断。这些操作已经被完成之后，8085返回到 MAIN TASK(主程序)。

录音机和键盘中断服务子程序与时钟中断服务子程序稍有不同。在录音机子程序和键盘子程序中，录音机或键盘设备已经被服务完毕之后，8085 的中断线才能被再启动。正如读者已经知道的，8085 正在为这两个设备的任何一个服务时，任何一个设备都不能中断 8085。当 8085 已经完成适当的服务操作之后，它把 00001000 装入A寄存器，从而，中断屏蔽才能启动 RSTn.5 各中断。8085 执行这条 SIM 指令，把A寄存器的内容传送给中断屏蔽寄存器；然后，8085 执行这条 EI 指令时，再启动这位内部中断标识位。然后 8085 返回到 MAIN TASK (主程序)。

8085 还可以执行另一条指令，而 8080 的指令系统也没有这条指令。这条指令就是 RIM (读中断屏蔽)。8085 执行这条指令，就能够把这三个中断 屏蔽位的状 态和对于 8085 微处理

器集成电路的 RST 5.5,RST 6.5，RST 7.5 这三个中断输入线的逻辑状态一起读到A寄存器里。在许多应用中，并没有使用这条指令。

我们要讨论的最后一条中断 线是 TRAP。如果 8085 的 引线 TRAP 置到逻辑 1，8085 则被引导到 000044 (0024) 这个地址。**8085 执行一条 DI 指令，或者通过执行一条 SIM 指令，改变中断屏蔽寄存器的内容，这样，则不能禁止 TRAP 中断。**没有办法禁止，这条 TRAP 中断。此外，请读者记住，对8085的各个中断优先权而言，TRAP 的中断优先权最高。根本没有 8085 可以执行的 TRAP 指令，因此，执行在 000044(0024) 地址上的子程序。这是我们要说明的最后一点。这种情况与中断线 RST 7.5，RST 6.5 和 RST 5.5 的情形相同。为了更多地知道 8085 的中断结构，请参考“MCS-85 的用户手册” (MCS-85 User's Manual)。

优先权中断程序定时

作为这一章最后一个讨论题，让我们来讨论执行优先权中断程序所要求的定时问题。假设我们的微型计算机系统只有两台中断设备：优先权较高的中断设备 7 和优先权较低的中断设备 2 。每一个中断设备产生它自己的一个再启动指令字节，用来分别把中断向量引导到 000070(0038)，或者 000020(0010)。我们又假设较高中断优先权设备按照规则的时间间隔,中断MAIN TASK 主程序，并且软件能很快地为它服务。假设设备 2，(优先权较低的设备) 按照不规则的时间间隔产生中断。为中断服务需要相当的时间。例如，设备 2 可能是正在向我们的

微型计算机传送数据块的另一台微型计算机。

最重要的软件是 MAIN TASK 软件。当我们的微型计算机没有为外部设备服务时，执行 MAIN TASK 软件。如果这个软件是不重要的，那么，它就不是 8080 微型计算机执行的主程序。在主程序的开始，我们用一条 LXISP 指令给堆栈指示器定位，然后由 8080 执行一条 EI 指令使 8080 允许中断。

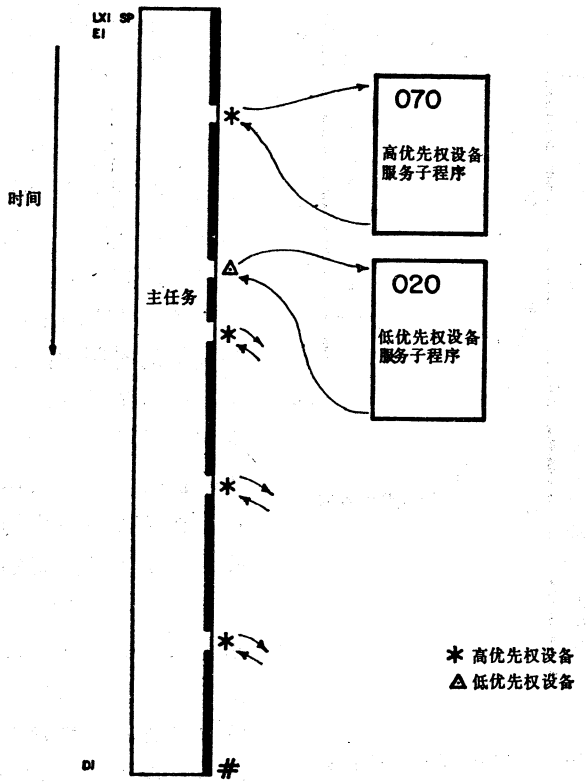


图 2-10 MAIN TASK 程序执行时间图

因为中断信号可能随时出现，所以在中断服务子程序中，需要有 PUSH 和 POP 这两条指令。这样的指令将用来存储和恢复任何寄存器的内容；这些寄存器在这些中断服务子程序中，可以变更。这个程序的执行如图 2-10 所示，可以用时间直线绘图表示。

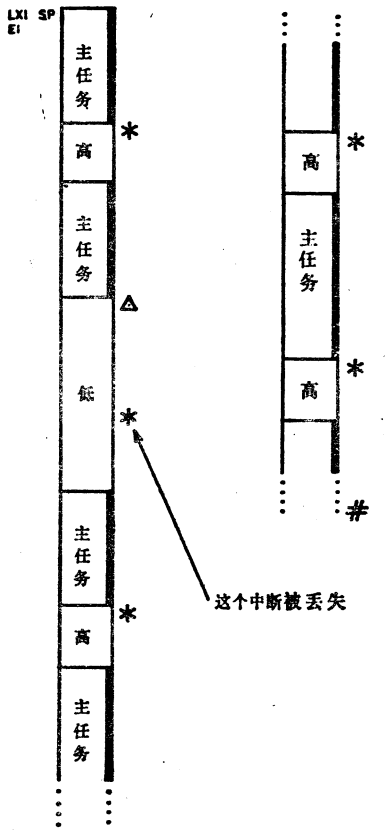


图 2-11 MAIN TASK 程序执行时间图

请注意：高中断优先权的设备已经把主程序中断了四次，而低中断优先权的设备只中断 8080 微型计算机一次。高中断优先权的设备按照额定的时间基准产生中断，如 MAIN TASK 时间线上的空间所示。中断被启动的时间用粗线表示。

图 2-11 示出了一条更加实际的时间线。图 2-10 示出的这条时间线有些不够真实，因为只表示出了主程序的执行时间。表示执行主程序和子程序所用的实际时间，这是比较正确的。在图 2-11 中，MAIN TASK 开始操作，被高优先权中断设备中断。8080 执行高优先权中断设备的中断服务子程序之后，程序执行控制返回到 MAIN TASK，然后，

MAIN TASK 在较后的时间线上，又被低中断优先权中断设备中断。程序执行控制最终被返回到 MAIN TASK；然后，由高优先权中断设备以重复的时间间隔中断 MAIN TASK。很显然，当 MAIN TASK 重复被外部设备中断时，到达主程序的末尾要用去更长的时间。在严格的定时周期时，如果我们将依靠编程序的延时循环，产生时间延期，那么，这样的重复中断会带来灾难。

我们已经假设高优先权中断设备按照规则的间隔产生中断。高优先权中断设备可能试图中断低优先权中断设备的中断服务子程序的执行，如图 2-11 所示。如果高优先权中断设备的中断优先权比低优先权中断设备的高，为什么不出现中断呢？答案是：**8080 在执行中断优先权低的中断服务子程序的时间里，没有启动 8080 芯片内的中断标志位。**我们第一次编写这个中断服务子程序时，忘记考虑这种可能性。结果，在执行优先权低的中断服务子程序时，中断优先权高的设备的数据和信号被丢失了。我们只要在中断优先权低的中断服务子程序的开始处，编入一条允许中断指令 EI，就能够改进我们的程序。我们也可以设计硬件，用来存储错过了一个中断时间内所出现的数据或信号。

我们把这条允许中断指令 EI 移到优先权低的中断服务子程序的开头，也许会碰到一个新问题：中断优先权低的中断服务子程序的执行可能被截断，如图 2-12 所示。为了强调这一点，我们假设优先权高的设备两次中断优先权低的中断服务子程的执行，把优先权低的子程序截成三块。由于优先权低的设备的子程序分裂成这个样子，所以，我们必须查问**在优先权低的设备产生另一个中断之前**，是否能够让 8080 完成执行优先权低的中断子程序。8080 微型计算机仍然企图在为来自优先

权低的设备的最后一个中断请求服务时，低优先权中断设备中断 8080 微型计算机，是完全可能的。当中断响应很快时，实际执行时间可能比一次通过中断服务子程序所需要的时间稍慢。这就是我们可以中断为中断设备服务的原因。微型计算机很容易被中断束缚，即微型计算机用全部时间，检查中断，并且为

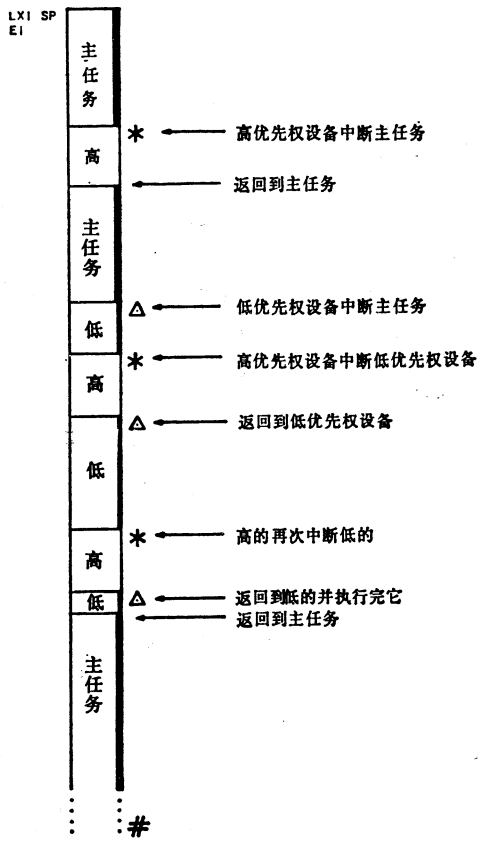


图 2-12　中断一个中断的中断程序执行时间图

它们服务；而根本没有留下时间来执行 MAIN TASK 程序。

在 MAIN TASK 软件中，我们或许希望防止在这样情况下产生中断：延时要求严格的软件，或取决于时间的复杂任务，计算。这条 DI 禁止中断指令使 8080 微型计算机不响应外部中断。这种情形如图 2-13 所示。为了允许执行应急任务，中断标识位被禁止，然后再启动它。可是 MAIN TASK 主程序的执行的时间图表明，高优先权中断设备产生的中断丢失了。由于没有附加的，一般复杂的硬件作为辅助电路，所以当这个中断标识位禁止时，很容易丢失中断设备的信号和数据。这里，需要指出的重要的一点是：**我们并不知道外部设备什么时候中断 MAIN TASK；我们不能肯定，在禁止中断的这段时间，外部设备将不会企图这样做**。怎样克服这个问题呢？克服这个问题是不容易的。这就是为什么我们在使用中断时，必须倍加小心的原因。

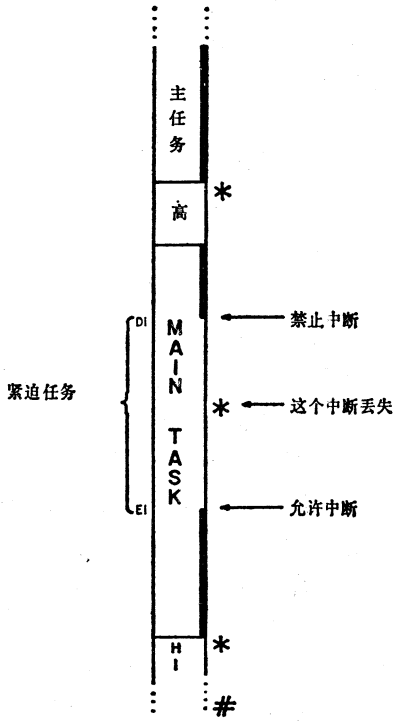


图 2-13 在应急任务时间，DI 和 EI 指令的应用

可能使读者感兴趣的另一种中断是**定时中断**。使用的唯一的中断：时钟。每隔 10 毫秒，或者其它合理的时间周期，时钟

中断 8080 微型计算机。当 8080 微型计算机被中断时，它利用对照表，决定哪些设备要检查，以便弄清它们是否需要服务。有些设备已经被检查了，而其它较慢的设备或许每隔 1000 次检查一次。这种技术是很有用的，但是，为了正常工作，它需要相当多的程序指令。

供 8080 微型计算机用的一种较新的支持集成电路在中断时间，允许把多字节指令写入到指令寄存器。这就是说，可以把完整的三个字节的转移指令或调用指令装入到 8080 微处理器集成电路中去。有"塞入"一条三字节的指令的能力，就没有必要使用再启动指令了，没有必要使用有关的向量单元了；并且，在软件和硬件的应用方面，为读者提供了很大灵活性。把一条三字节的指令送入 8080 微处理器，就是说 8080 微处理器集成电路的支持逻辑电路必须产生三个 $\overline{\text{INTA}}$ 脉冲信号（这条指令的每一个字节一个脉冲）。完成这项任务的逻辑电路是复杂的，所以，并不经常采用它。如果把 8228 系统控制器与8080一道使用，则稍微容易一些。Intel（英特尔）公司生产的 8259 可编程序中断控制器允许读者采用直接调用中断服务子程序的技术，但是 8259 是一种复杂的器件，初学编程序者不宜使用它。

本章结束前，我们最后提出几点注意事项。调试中断是困难的。因为中断几乎随时都有可能出现；应用典型软件调试程序是困难的；因为大多数是无效的。在具体应用中测试中断需要具体的诊断程序。如果能够避免使用中断的话，那就尽量避免。如果可能，把你们宝贵的时间用在非中断的其它方法上面。你们的努力将会得到良好的酬报。

第三章　中断的应用

我们已经讨论了 8080 微型计算机怎样被中断以及为这些中断服务所需要的信号和指令，因此，本章我们将要讨论中断在解决接口问题时的应用。读者将可以看到，这个问题的解决方法有硬件的和软件的。

实时时钟

8080 微型计算机在某些应用方面，可能需要按照预先确定的时间间隔，把数据输出给外部设备；例如：每隔 23 毫秒输出一次数据，或者每隔 13.2 秒输出一次数据。正如我们已经看到的那样，用一串指令给 8080 编程序，产生这样一段时间的延时是能够做到的。产生延时后，一个数据可以输出给外部设备；然后可以开始另一延时周期。但是，对于大多数的应用而言，这并没有实用价值，因为，8080 微型计算机有许多其它的任务将需要完成，计算出完成这些不同的任务所需要的时间是困难的。正是这个原因，我们将使用**实时时钟**把该时钟与 8080 的中断信号线连接。使用实时时钟，就能够以每隔 23 毫秒，或 13.2 秒，或其它的时间间隔产生中断；这就是说，在这个软件中，不再需要延时程序了；不需要计算完成其它任务所需要的时间。

什么叫做实时时钟呢？实时时钟是供计时用的一种外部设

备，**不论微型计算机做什么操作**，它都计时。实际上，实时时钟甚至在微型计算机停机时也可以继续计时，这正是因为它是一种独立于 8080 微型计算机而操作的外部设备。当实时时钟计时完毕，这就是说实时时钟已经给预定的时间间隔计完了时，8080 则被中断。这时，8080 可以为适当的外部设备服务。8080 是否也可以为实时时钟服务，这取决于硬件的设计。

实时时钟器件的“心脏”就是 MOSTEK 公司生产的 MK 5009 金属-氧化物-半导体计数器时基电路，如图 3-1 所示。这块集成电路有四条可编程序的输入端，用来确定它的输出时钟频率(TIMEOUT，定时输出，引线 1)。把 1MHz 的石英晶体与 MK 5009 连接时，用 0000～1000(引线 14～11)的四位二进制字给 MK 5009 编程序，就可以得到表 3-1 所列出的频率。为了得到有关 MK 5009 更多的有关资料，请参考附录 A。

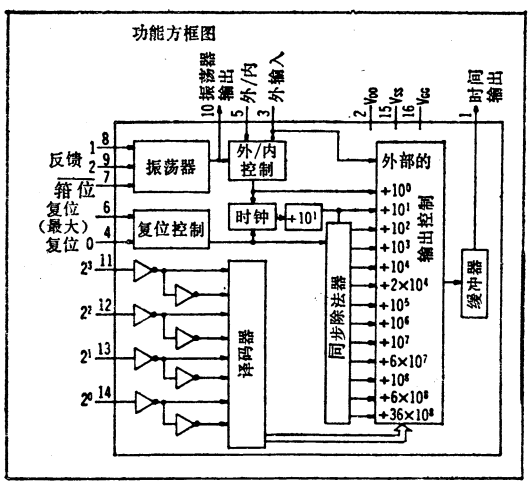


图 3-1 MOSTEK MK 5009 计数器电路功能框图

当然，从我们所知道的为中断服务所需要的最少时间(至

少把所有寄存器的内容都存储在堆栈)，8080 微型计算机不可能为每隔 1 微秒或 10 微秒就中断 8080 微型计算机一次的外部设备服务。

外部设备每隔 100 微秒中断 8080 微型计算机一次，则 8080 能够为这种外部设备服务；但是，只有这个中断服务子程序编写得尽可能短才行。读者可以用来中断你的微型计算机的最高频率(最短周期)实际将取决于中断出现时，被执行的操作的复杂程度。

如果采用 MK 5009，能产生 23 毫秒或 13.2 秒的延时吗？能够，但是解决这个问题需要用硬件和软件。

硬/软实时时钟解决办法

我们用来做实时时钟接口的硬件框图，如图 3-2 所示。读者可以从这个方框图看到，我们使用了一个四位的输出端口(OUT 305)来为 MK 5009 存储一个适当的可编程序数值。我们用 MK 5009 的 TIME OUT 输出端来给 D 触发器定时；D 触发器的输出($\overline{Q}$)驱动优先权编码器为基础的一个中断接口(该接口已经在前一章作了描述)。一旦 8080 为中断服务时，OUT 306 将会使该触发器清零。因为该硬件电路不会每隔 15 毫秒或 20 秒(参考表 3-1)中断 8080 微型计算机，所以，这个中断服务子程序必须以某种方式确定，所要求的持续延时时间是否已经出现了。

如果我们给 MK 5009 编程序，得到操作频率为 1 kHz，那么在 23 个 1 毫秒延时周期应该会产生 23 个中断。经过 200 秒的延时时间，0.1 秒的周期必定会出现 2000 个。例 3-1 所列出的中断服务子程序记录了已经出现的时间间隔的个数；给时间间隔的个数计数，是采用将计数数字减 1 的办法来实现的。

在例 3-1 里，我们把读写存储器地址装入堆栈指示器和寄存器对 H 之后，把 027(十进制数 23) 保存在分配给符号地址“COUNT”和“TEMP”的存储器单元。这个数是外部设备被服务之前中断必定出现的次数。把 003(03)装入 A 寄存器，这个数值输出给实时时钟的锁存器。003(03)这个数(根据表 3-1)给 MK 5009 编程序，以便工作于 1k Hz (1 ms)。第二条 OUT 指令给触发器清零(如图 3-2 所示，这个触发器是由 MK 5009 的输出电平驱动的)；然后，8080 允许中断。接着 8080 执行“MAIN TASK”(主任务)。

表 3-1　使用 1 MHz 晶体的 MK 5009 可编程的输入和时钟频率输出

编程序输入				时钟频率输出	周期
2^3	2^2	2^1	2^0		
0	0	0	0	1 MHz	1μs
0	0	0	1	100 kHz	10μs
0	0	1	0	10 kHz	100μs
0	0	1	1	1 kHz	1 ms
0	1	0	0	100 Hz	10 ms
0	1	0	1	10 Hz	100 ms
0	1	1	0	1 Hz	1 s
0	1	1	1	0.1 Hz	10 s
1	0	0	0	0.01 Hz	100 s

然后，实时时钟从 0 到 1ms，将中断 8080 微型计算机。根据在前一章所看到的中断硬件，我们知道，一条再启动指令被置于数据总线上；当 8080 产生一个中断响应 ($\overline{\text{INTA}}$)信号时，把这条再启动指令写入指令寄存器。因为例 3-1 示出的中断服务子程序的始地址是 000 070(0038)，所以可以假设，图

3-2 所示的触发器的输出被连接到优先权最高的中断线。当8080被中断时，它执行中断服务子程序 RTCISS(例 3-1)的指令。

8080 到达 RTCISS 子程序时，把寄存器对 H 的内容和 PSW(处理机状态字)保存在堆栈。然后，把 COUNT 的存储地址装入寄存器对 H。然后，这个存储单元的内容减 1。如果 8080 执行 DCRM 指令的结果是零，那么，8080 执行 JZ-PSERV 指令。这就是说，已经出现了 23 次中断。如果还没有产生 23 个中断，8080 则不执行 JZ-PSERV 指令，而是从堆栈弹出处理机状态字和寄存器对 H 的内容。然后 8080 把中断触发器(OUT 306)清零，从而再启动中断。8080 然后返回到原来被中断的 MAIN TASK 程序段。

如果已经产生了 23 个中断，那么，8080 转移到 PSERV。到达 PSERV，寄存器对 H 的存储地址加 1，然后把符号地址 TEMP 的内容送入 A 寄存器。然后，把这个存储地址减 1，接着把 A 寄存器的内容保存在 COUNT 存储单元里。这样，就把计数数再恢复到 23。然后，8080 可以把一个数据字输出给外部设备，或者执行其它某个控制功能。8080 为这台外部设备服务完毕之后，它转移到 AGAIN，于是，从堆栈弹出处理机状态字和寄存器对 H 的内容，并且把中断触发器清零。然后启动中断线，8080 返回到 MAIN TASK 程序。

为中断服务，需要多少时间呢？如果 COUNT 的内容减 1，还不等于零时，那么只需要 48μs(微秒)。如果 COUNT 的内容(计数值)减到零时，则至少需要 65μs，另加实际为外部设备服务所需要的时间。现在读者能明白，为什么我们不让外部设备每一次以小于 100μs 的时间间隔中断 8080 微型计算机。如果以更短的时间间隔中断 8080，中断与中断之间的时间很短，

因此，留给执行主程序指令的时间几乎没有了。我们还知道外部设备决不能中断它自己的中断服务子程序。不然的话读/写存储器将会被返回地址填满。

例 3-1 实时时钟的中断服务子程序

```
    *000000
START,  LXISP     /把一个读/写存储器地址
        STACK     /装入堆栈指示器。
        0
        LXIH      /把分配给"COUNT"的
        COUNT     /存储地址装入寄存
        0         /器对 H。
        MVIM      /把十进制数 23(八进制 027)
        027       /存入这个存储单元。
        INXH      /寄存器对 H 的地址加 1。
        MVIM      /把十进制数 23(八进制 027)
        027       /存入"TEMP"的存储单元。
        MVIA      /然后把这个字装入 A 寄存器，
        003       /该字用来给 MK 5009 编程序，得到
        OUT       /1 kHz(1 ms)的工作频率。
        305       /输出这个数。
        OUT       /中断触发器清零。
        306
        EI        /8080 允许中断。
          •       /然后执行"MAINTASK"。
          •
          •
    *000070
RTCISS, PUSHH     /保存寄存器对 H 的内容、
        PUSHPSW   /A 寄存器的内容和标识位。
```

```
        LXIH    /把存储计数数的
        COUNT   /存储器地址装入
        0       /寄存器对 H。
        DCRM    /存储器的计数数减 1。
        JZ      /这个数等于 0，所以
        PSERV   /为这台外部设备
        0       /(已经延期了 23ms)服务。
AGAIN,  POPPSW  /这个数不等于 0，所以，从堆栈
        POPH    /弹出 A 的内容、标识位和
        OUT     /寄存器对 H 的内容。
        306     /把中断标识位清零。
        EI      /8080 允许中断，
        RET     然后，返回到"MAINTASK"。
PSERV,  INXH    /已经出现了 23ms 的延时，
        MOVAM   /所以从"TEMP"取计数数送 A
        DCXH    /然后把该数存回
        MOVMA   /到"COUNT"。
        .       /然后为中断服务。
        .
        .
        JMP     /从堆栈弹出寄存器的内容，
        AGAIN   /给中断标识清零。
        0       /再启动中断，然后返回到"MAINTASK"。

COUNT,  0       /"工作计数数"被存储在这里。
TEMP,   0       /十进制数 23 存储在这里。
```

硬实时时钟的解决方法

如图 3-3 所示，给实时时钟接口增添 6 块集成电路，就可

以提高为实时时钟计数的数量，减少中断服务子程序(例 3-1)的复杂性。如果使用这种结构的硬件，计时到了整个时间间隔，8080 才被中断。8080 为了决定是否应该为外部设备服务，再也没有必要给存储在读/写存储器的计数数字减 1。

如果把三个 4 位的计数器用于实时时钟，那么，中断出现之前，可以计数的次数高达 2^{12}。这就是说，可以给实时时钟编程序，从而能每隔四天中断 8080 微型计算机一次！请读者注意，图 3-3 的较上部锁存器是由 OUT 305 所输出的脉冲控制的，就是这个信号可以用来锁存 MK 5009 的可编程序字。

为了给实时时钟编程序，必须把一个 12 位的计数数输出给三个 SN 7475 锁存器。这三个锁存器连接到 SN 74193 计数器上。该操作完毕后，12 位的数就可以从这三个锁存器发送给 SN 74193 计数器。读者可以看到，最低有效位计数器的时钟输入端是由 MK 5009 的输出驱动的。因为这三个计数器连接成递减计数器，所以，用最高有效位计数器的借位输出为图 3-2 示出的 D 触发器定时，实际上引起 8080 微型计算机中断。

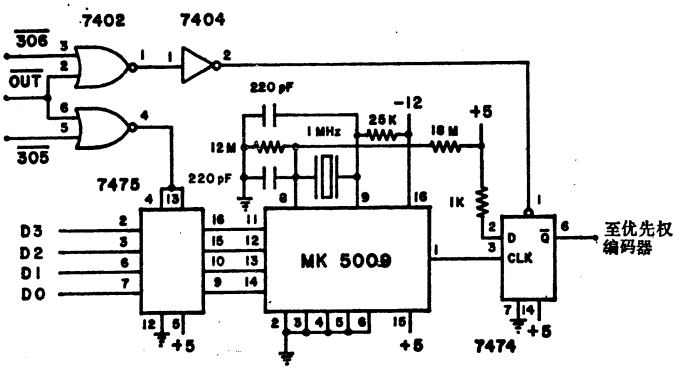


图 3-2 微型计算机可编程序实时时钟

这就是说，这个“计数器链”(SN 74193 这三个计数器)在电气上已经连接在 MK 5009 的输出端和 D 触发器的输入端之间。

可编程序的实时时钟的数据格式

为了给实时时钟编程序，我们必须把一个 12 位的数输出给这些 SN 7475 锁存器，把同一个数从这三个锁存器输出给这三个 SN 74193 计数器。我们还必须给 MK 5009 编程序，从九种可能的频率(周期)得到某一种。我们用两个八位的输出端口来锁存这 16 位信息。给实时时钟编程序的数据的格式如表 3-2 所示。

表 3-2　　可编程序的实时时钟的数据格式

OUT 305		OUT 304	
D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0	D_7 D_6 D_5 D_4	D_3 D_2 D_1 D_0
最高有效位计数器	MK 5009	中间位计数器	最低有效位计数器

根据表 3-2 所示的数据格式，我们必须把什么数值输出给实时时钟，才能使 8080 微型计算机每隔 23 ms (毫秒) 被中断一次呢？为了给时间周期的实时时钟编程序 (23 ms 的时间周期)，可以输出下列数：

OUT 305　　OUT 304

0000 0011　　0001 0111

用这个数值给 MK 5009 编程序，得到 1 kHz (1 ms) 的操作频率(数据值＝0011)；把 0000 0001 0111 这个数装入计数器链。还可以用其它数来给实时时钟编程序吗？可以，可以用表 3-3 的任何一个数给实时时钟编程序，得到 23 ms 的时间间隔。

表 3-3 中的第一个 16 位数给 MK 5009 编程序，得到 1

表 3-3 给实时时钟编程序，每隔23ms 产生中断的数值

OUT 305								OUT 304							
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0
0	0	0	0	0	0	1	1	0	0	0	1	0	1	1	1
0	0	0	0	0	0	1	0	1	1	1	0	0	1	1	0
1	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0

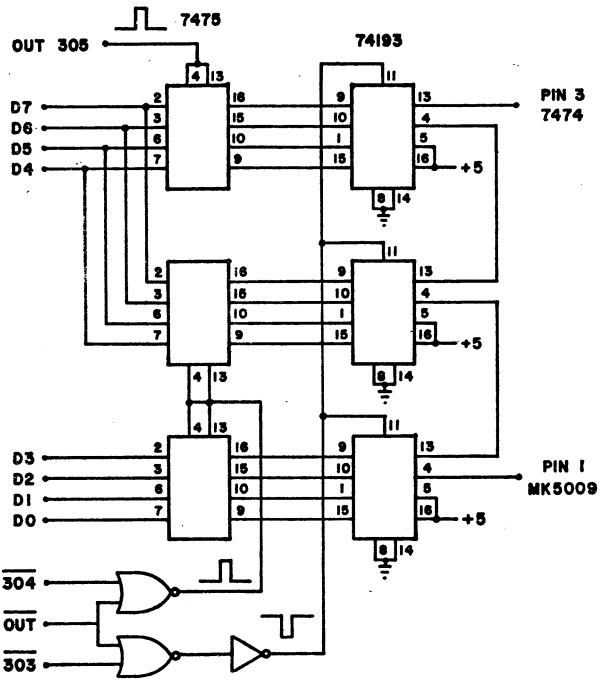


图 3-3 灵活的硬件可编程序的灵活实时时钟

kHz操作频率，十进制数 23 装入这些计数器。第二个 16 位数给 MK 5009 编程序，得到 10 kHz 操作频率，十进制数 230 装

入这些计数器。最后这个 16 位数给 MK 5009 编程序，得到 100 kHz 操作频率，十进制数 2300 装入这些计数器。这样组合将产生高精度的时间，时间的最大误差是 10 μs(微秒)。

用什么样的指令序列来给实时时钟编程序得到 23 ms 的操作呢？假设，我们必须把这个十六位数 00000011 00010111 输出给实时时钟(10 kHz，十进制数 23)，那么，例 3-2 的指令序列正好可以用来完成此任务。

在例 3-2 中，8080 把一个读/写存储器地址装入堆栈指示器，然后把 027 (十进制 23)输出给实时时钟(OUT 304)的两个最低有效位计数器。接着把将给最高有效位计数器和MK 5009 编程序的数装入A寄存器，把 0000 装入计数器，0011 给 MK 5009 编程序得到 1 kHz 的工作频率(输出端口 305)。然后 8080 执行 OUT 303 这条指令，把原先输出给三个锁存器的三个数装入这三个计数器；8080 执行 OUT 306 指令，把中断标识位清零。8080 执行 EI 指令，允许中断线，返回执行 MAIN TASK 程序。

例 3-2 23 ms 的可编程实时时钟

```
        *000000
START, LXISP  /把一个读/写存储器地址
        STACK/装入堆栈指示器。
        0
        MVIA  /然后把十进制数 23
        027     /装入A寄存器
        OUT     /把这个数输出给
        304     /两个最低有效位计数器。
        MVIA  /然后，使最高有效位计
        003     /数器置 0，并且给 MK 5009 编程
        OUT     /序，得到工作频率
```

```
        305     /1 kHz(1 ms)。
        OUT     /把这三个锁存器的输出
        303     /装入这三个计数器。
        OUT     /给中断标识位
        306     (D 触发器)清零。
        EI      允许中断，
        •       然后执行“MAINTASK”
        •
        •
        *000070
RTCISS, •       /现在，为外部
        •       /设备服务。
        •
        OUT     /然后，再装入这些计数器。
        303
        OUT     /清除中断标识位
        306     /(D 触发器)。
        EI      /允许中断。
        RET     /然后返回到“MAINTASK”。
```

23 ms 后，将出现一个中断。中断出现时，8080 微型计算机为这台外部设备服务。由于给实时时钟添加了三个计数器，所以，这个中断服务子程序不需要为计数器做减 1 操作的指令。这就是说，每次中断出现时，就为这个外部设备服务。在为这个外部设备服务之后，8080 执行 OUT 303，把存储在这三个锁存器的数发送给三个 SN 74193 计数器。8080 执行OUT 306 指令，把中断标识位清零，然后允许中断，8080 返回到该中断出现时正在被执行的 MAINTASK 程序段。

硬件实时时钟和软件实时时钟的比较

在软件实时时钟里，必须把计数数字存储在读/写存储器里。每次中断出现时，这个数必须减1。如果这个数减1后不等于0时，则不能为这个外部设备服务。如果这个数减到0时，必须把这个计数数重新装入（从一个存储单元传送到另一个存储单元），然后才能为这个外部设备服务。

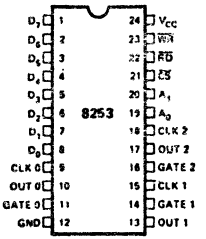
软件实时时钟需要很少很少的硬件；假设已有适当的译码器和中断硬件，则只需要五块集成电路。这种技术的主要缺点是，为中断及其外部设备服务需要的时间较长。为外部设备服务一次，实时时钟要产生23个中断。这就是说（22×48 μs）+65 μs，即需要1.12 ms，才可以为每隔23 ms产生的一个中断服务；另外还要加上实际为外部设备服务的时间。

硬件实时时钟需要八块集成电路。但是，因为，每隔23 ms只产生一次中断，所以，实时时钟的"内务"指令（给三个计数器重编程序和把中断触发器清零）所需要的时间很少。实际上，处理硬件实时时钟的内务指令只需要17 μs。对于要求8080必须尽可能快地执行任务的应用，硬件实时时钟为外部设备服务所需的时间是最少的。

硬实时时钟的选择

许多半导体厂家已经认识到用户极需软件可编程序的计数器和定时器。英特尔公司的8253可编程序时间间隔定时器是这些器件的一种，它有三个独立的16位计数器。每个计数器有一个时钟输入端，一个计数器输出端，和一条控制线（它可以用来启动和停止计数器操作）。8253集成电路的引线结构和方框图如图3-4所示。

引线结构



方框图

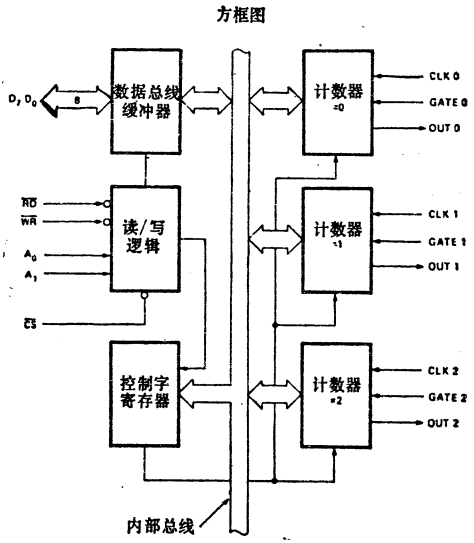


图 3-4 英特尔 8253 可编程定时器

引 线 名 称

D_7D_0	数据总线(8 位)
CLK N	计数器时钟输入端
GATE N	计数器控制输入端
OUT N	计数器输出端
$\overline{RD}$	读计数器信号
$\overline{WR}$	写命令或数据信号
$\overline{CS}$	片选信号
A_0A_1	计数器选择
V_{cc}	十万伏
GND	接 地

对于中断应用而言，计数器的输出(OUT 0，OUT 1或OUT 2)可以与前一章所叙述的任何一个优先权中断接口连接。因为，该集成电路中的这些计数器可以在直流电平上工作，也可以在高达2 MHz的频率上工作，所以，可以把8253的时钟输入端(CLK 0，CLK 1和CLK 2)与TTL时钟连接(TTL时钟是由8080时钟发生器集成电路8224产生的)，或者与8085的CLK(OUT)输出端连接。

如果这样连接，那么一个计数器就可以产生0.5～32,768 μs的延时时间。但是，如果用一个计数器的输出为另一个计数器定时，那么，可以得到高达2^{31}μs的延时时间；将近18分钟。如果把第三个16位的计数器加到计数器链上，那么可以产生大于407天的延时时间。

8253可编程序定时器可以编程序，得到三个不同延时时间，这是8253的突出特点。这就是说，对于三台外部设备的微型计算机系统而言，一个定时器可以用于每台外部设备。如果必须以三个不同的时间间隔为这三台外部设备服务，例如：233 μs，500 μs和1.895 ms，那么8253的这一特点是特别有用的。为了用这个以MK 5009为基础的实时时钟来完成这个任务，我们应该制造三个可编程序的实时时钟。

但是，8253可编程序定时器用于很复杂的定时问题则有一个缺点值得注意。每次定时器产生中断时，或者"定时已完"时，必须给这个定时器再装入1个16位的计数数。完成这一操作，需要执行两条OUT指令，两条MOV类指令，或一条SHLD指令。采用这三种指令哪一种，将由与8253一道使用的接口(累加器输入/输出或者存储器映象输入/输出)决定。由于这个实时时钟是以MK 5009为基础的，所以，8080只要执行一条OUT指令，就可以把一个12位的数装入三个SN 74193计数

器。

小结

正如我们前面所述，让 8080 微型计算机连续不断地查询外部设备，来决定它们是否需要服务，这种方式，效率不高。但是，采用中断方式，外部设备可以告诉 8080 它们何时需要服务。同时，让 8080 微型计算机执行延时子程序，效率可能不高。因为它可能有许多更重要的任务要执行。正是这个原因，我们采用实时时钟。实时时钟节省 8080 微型计算机执行延时子程序的时间。

日 时 钟

我们利用实时时钟，可以编写程序，用小时，分和秒为单位来记录时间。这种程序所完成的功能与硬件时钟所完成的相同。我们的日**时时钟**的第一个程序例子是以 24 小时的格式记录时间的。因此，当日时时钟程序被执行时，下午时间 1:32 实际上将表示为 13:32。我们早就从表 3-1 知道，可以给 MK 5009 编程序，每隔一秒钟中断 8080 一次。当 8080 微型计算机被中断时，它必须使 BCD(二—十进制数)计数数加 1。该数存储在存储器，用它来表示秒数。如果这个数增加到 BCD 60，那么，必须把秒数置零，而把分钟数加 1。如果用 BCD 数表示的分钟数现在等于 60，那么，分钟数必须置零，BCD 表示的小时数增 1。如果 BCD 数表示的时数现在等于 24，那么，时数必须置 0。因为我们使用的是压缩的 BCD 数，所以存储该时间只需要三个读/写存储单元。还需要三个读/写存储单元，

用来存储最大的小时数，分钟数和秒数（24，60 和 60）；这些数字用来表示下一个较高有效数字应该增 1 的时间。所有这些操作都由例 3-3 所列出的程序来执行的。因为 MK 5009 在每一秒钟可以产生一个脉冲，所以，并不需要使用三个 SN 74193 可编程序的低位计数器（图 3-3）。因此，MK 5009 的输出端可以再与D触发器连接，这个D触发器与优先权中断接口连接。

日时钟程序的始地址是 002000(0200)。当 8080 开始执行这个程序时，做装入堆栈指示器的操作，因为当实时时钟的定时数值已完时，中断硬件将把一条再启动指令写入指令寄存器。然后，把 006 装入 A 寄存器里，它是输出给 MK 5009 的。006 这个数用来给 MK 5009 编程序，得到 1 Hz(1 秒)的工作频率。然后把存储时间用的存储地址装入寄存器对 H。再把 003 (03)装入 B 寄存器，以便把 ZERO 循环程序执行三次。在 ZERO，寄存器对 H 寻址的存储器单元置零。然后寄存器对H里的存储地址增 1，B寄存器的内容减 1。这个循环把 BCD 00 存储在三个连续的存储器单元，始于 CURTIM。结果是：时间置到 00:00:00。8080 给时间预置初值后，实时时钟接口的中断触发器清零，8080 允许中断。于是 8080 执行(MAIN TASK)主程序。

例 3-3 中断驱动的日时时钟程序

```
/这是中断服务子程序。中断硬件
/产生一条 RST 4 指令。

    * 000040
CLOCK,    PUSHPSW     /当中断出现时，把各个寄存
          PUSHB       /器的内容都保存在堆栈。
          PUSHD
          PUSHH
```

```
        JMP         /在"中断服务继续"(ISCNT)时
        ISCNT       /继续为该中断服务。
        0
```

这是日时时钟程序

(这是"MAIN TASK"开始)。

```
        *002000
START,  LXISP       /装入堆栈指示器，因为中
        STACK       /断将把一个返回地址保
        0           /存在堆栈。
        MVIA        /把给 MK 5009 编程序，
        006         /使其工作于 1 Hz 频率的字
        OUT         /装入 A 寄存器。
        305         /然后，把该字锁存，输出。
        LXIH        /把存储现行时间(CURTIM)
        CURTIM      /的存储地址装入寄存器对 H
        0           /(现行时间是以 BCD 格式存储的
        MVIB        /把"CURTIM"所用的存储单元
        003         /的数目装入 B 寄存器。
ZERO,   MVIM        /把 000 装入存储单元。
        000
        INXH        /这个存储地址加 1。
        DCRB        /字数减 1。
        JNZ         /如果这个数不等于 0，则把
        ZERO        /另一个存储单元
        0           /置 000
        OUT         /把 MK 5009 的中
        306         /断标识位清零。
        EI          /8080 允许中断，继续
        .           执行"MAIN TASK"。
        .
```

8080 转移到“ISCNT”，继续为
每秒产生一次的中断服务。

```
ISCNT,    LXIH      /把这个表的地址装入寄存器对 H，
          TOPVAL    /该表的内容包括最大的秒数，
          0         /分数和时数。
          LXID      /然后，把存储现行
          CURTIM    /时间的地址装入
          0         /寄存器对D。
          MVIB      /把所包括的存储器单元
          003       /的数目装入 B 寄存器。
UPONE,    LDAXD     /取这个数。
          ADI       /把 1 加到这个数上。
          001
          DAA       /对该结果进行调整。
          STAXD     /保存这个新的时间。
          CMPM      /它太大吗？
          JZ        /是的。然后把这个数置零，
          NEXT      /使表示时间的下两位数字
          0         /增 1。
MIDNGT,   LXIH      /然后，显示现行时间。
          CURTIM
          0
          MOVAM     /表示秒数的 BCD 数取到 A，
          OUT       /把该数输出给
          002       /七段显示器。
          INXH      /这个存储地址加 1，
          MOVAM     /把表示分数的 BCD 数取到 A，
          OUT       /然后，把该数输出给
          000       /七段显示器。
```

```
        INXH    /这个存储地址增 1。
        MOVAM   /表示分数的 BCD 数取到 A，
        OUT     /然后，把这个数输出给
        001     /七段显示器。
        POPH    /该时间是有效的，所以
        POPD    /从堆栈取出各寄存器的内容。
        POPB
        POPPSW
        OUT     /现在把
        306     /与 MK 5009 连接的触发器清零。
        EI      /8080 允许中断，
        RET     /然后返回到"MAIW TASK"。
NEXT,   MVIA    /把 A 寄存器置 000
        000     /(BCD 和十六进制 00)。
        STAXD   /然后，把它保存在现行时间地址。
        INXH    /这个表地址加 1。
        INXD    /这个现行时间地址加 1。
        DCRB    /存储单元的数目减 1。
        JNZ     /这个计数数不等于 0，所以
        UPONE   /下一个连续存储单元
        0       /可以增 1。
        JMP     /表示小时的数等于 24，
        MIDNGT  /所以，是半夜，不使任何存
        0       /储单元增 1。
CURTIM, 000     /这里存储 BCD 数表示的秒数。
        000     /这里存储 BCD 数表示的分数。
HOURS,  000     /这里存储 BCD 数表示的小时数。
TOPVAL, 140     /最大 BCD 秒数＝60。
        140     /最大 BCD 分数＝60。
        044     /最大 BCD 小时数＝24。
```

当实时时钟中断 8080 时，8080 被引导到 000 040(0020)地址单元。我们从这个存储地址开始，把实时时钟的中断服务子程序存入存储器。在 CLOCK(000 040 0020)开始处，8080 的所有的通用寄存器的内容和标识位都被保存在堆栈中。然后，8080转移到在 ISCNT 处的中断服务子程序的其余部分。我们这样编写中断服务子程序，使得其它外部设备能够使用 RST 5 指令作为中断指令。

在 ISCNT 处，把存储表示秒数的"最大"BCD 数的存储器地址装入寄存器对 H。把存储"现行"时间的 BCD 秒数的存储器地址装入寄存器对 D。把用来存储现行时间的存储器单元的数目装入 B 寄存器。在 UPONE 处，把现行时间的秒数从存储器装入到 A 寄存器。把 1 加到这个数上，然后把这个结果进行十进制调整(我们可以这样做，是因为 8080 正在对被压缩的 BCD 数进行运算)。这个结果存储在寄存器对 D 所寻址的存储单元。然后把同一个数与秒数的最大的 BCD 进行比较；表示秒数的最大 BCD 数被存储在 TOPVAL 表里。如果现行时间的秒数不等于 BCD 数 60，8080 则执行 MIDNGT 处的指令。

在 MIDNGT 处，把存储现行时间的秒数的存储地址装入寄存器对 H。然后，把 BCD 秒数输出给输出端口 002，分钟数输出给输出端口 000，小时数输出给输出端口 001。这些输出端口带有锁存器和七段发光二极管显示器(LED)，所以显示出现行时间。为了更多地知道备有 LED 显示器的输出端口的内容，请参考«8080/8085 的软件设计»上册的第七章。显示这个时间后，从堆栈弹出各寄存器的内容，把中断标识位清零，8080的再次允许中断，然后，8080 返回到 MAINTASK。

如果现在时间的 BCD 秒数等于 60，则 8080 执行 JZ-NE

XT 处(在 MIDNGT 之前)指令。在 NEXT 处，秒数置 0。寄存器对 D 和 H 存储的存储地址然后都加 1，B 寄存器存储的计数数从 003(03)减到 002(02)。因为该结果不等于零，所以 8080 转移到 UPONE。

如果秒数已加到 BCD 数 60，那么，现在，分钟数必须加 1。如果现在分钟数加到了 60，8080 则把分钟数置零，然后，把小时数加 1。如果小时数加到了 24，则 8080 把小时数置零。8080 完成该操作之后，它转移到 MIDNGT。在 MIDNGT 处，显示时间，并且，从堆栈弹出各个寄存器的内容。然后，8080 把该中断标识位清零，允许中断，然后返回到 MAINTASK。

用一个固定时间给日时钟预置初值

当日时时钟程序在地址 002 000(0200)处被启动时，日时时钟的时间被置于 00:00:00。假设读者在上午 10:15把这个程序装入到你的微型计算机，那么你能启动这个程序，显示 10:15 吗？8080 微型计算机能够显示这个时间。读者可以采用的方法之一(编程序)，如程序例 3-4 所示。

例 3-4 把时间 10:15:00 保存在读/写存储器

```
          *002 000
START,    LXISP     /装入堆栈指示器，因为
          STACK     /中断将把一个返回地址
          0         /保存在堆栈。
          MVIA      /把用来给 MK 5009 编程序，
          006       /得到 1 Hz 的工作频率的
          OUT       /数装入 A 寄存器。
          305
          LXIH      /把用来存储
```

```
CURTIM      /现在时间的地址（CURTIM）
0           /装入寄存器对 H(BCD 格式)。
MVIM        /然后，把秒数置 00。
000
INXH        /存储分数的地址加 1。
MVIM        /然后把分钟数置成
025         /15。
INXH        /存储小时的地址加 1。
MVIM        /然后，把小时数置成 10。
020
OUT         /把 MK 5009 的中断标识位
306         /清零。
EI          /允许中断，
•           /然后 8080
•           /执行“MAINTASR”。
•
```

在这个程序里，使用了三条 MVIM 指令，就把一个固定时间存储在现行时间读/写存储单元。如果必须把这个程序存入只读存储器，那么这种方法并不特别适合，因为给日时钟编程序所用时间变化。

当然，我们也能为电传打字机编写一个程序，输入这个具体的时间。如果我们这样做的话，那么，8080 必须检查被输入的时间，以便弄清规定的小时数是否比 23 大。还必须保证所规定的分钟数或秒数不能大于 59。使用例 3-5 所列出的这个程序，读者可用电传打字机输入的一个时间，来编日时钟程序。

例 3-5 用电传打字机把时间送入 8080 微型计算机

```
/这是日时时钟程序
```

```
/(这个程序是 MAIN TASK 的起始部分)。
          *002,000
START     LXISP     /装入堆栈指示器，因为
          STACK     /中断将把一个返回地址保
          0         /存在堆栈。
          MVIA      /把用来给 MK 5009 编程序得到
          006       /1 Hz 操作频率的字装入
          OUT       /A 寄存器。
          305       /然后，把这个字锁存，输出。
          LXIH      /把用来存储小时数
          HOURS     /的存储地址装入
          0         /寄存器对 H。
          MVID      /然后，把可以送入的"字"数
          003       /装入D 寄存器。
          CALL      /在 CRT 或电传打字机
          CRLF      /上打印回车符
          0         /和换行。
NXTDIG,   CALL      /取一个两位数的 BCD 数，
          TIMIN     /把这个两位被压缩的 BCD
          0         /字存入 A 寄存器，然后返回。
          MOUMA     /把这个数存入存储器。
          DCXH      /这个存储地址减 1。
          DCRD      /计数数字减 1。
          JZ        /这个数等于 0，所以检查一个
          CHECK     /有效时间是否已被输入。
          0
          MVIA      /否则，在这个两位数字的数
          072       /刚被送入后，打印冒号[:]
          CALL
          TTYOUT
```

```
        0
        JMP
        NXTDIG  /然后，取另一个两位数的数。
        0
CHECK,  LXID    /现在，把被送入的时间数
        TOPVAL  /与最大可允许的数进行比较。
        0
        INXH    /H 和 L 的内容加 1,指向秒数的存储单元。
        MVIC    /把被检查的数的个数
        003     /存入 C寄存器。
AGAIN,  LDAXD   /从"TOPUAL"表取一个数。
        DCRA    /这个数减 1。
        CMPM    /把它与被送入的一个数比较。
        JNC     /送入的这个数比表中
        CHEXT   /的这个数小，所以测试
        0       /已打入的下一项。
        MVIA    /输入的这个数太大，
        277     /所以，打印问号，
        CALL/   /然后，让用户再试。
        TTYOUT
        0
        JMP
        START
        0
CNEXT,  INXH    /下两项数的存储器
        INXD    /地址加 1。
        DCRC    /这三个数都比较完了吗？
        JNZ     /没有，然后比较
        AGAIN   /下两项数。
        0
```

```
        CALL    /被送入的时间是有效的，
        TTYIN   /所以，等待一只键被按下，
        0       /才"启动"时钟。
        OUT     /MK 5009 的中断
        306     /标识清零。
        EI      /允许 8080 中断，继续
        •       /执行"MAINTASK"。
        •
        •
TIMIN,  CALL    /取一个有效的 BCD 数，
        BCDIN   /然后把它存入 A 寄存器，
        0       /返回。
        RLC     /接着，把这个数循环移入
        RLC     /进入高四位
        RLC
        RLC
        MOVCA   /然后，把它存入 C 寄存器。
        CALL    /取下一个数字。
        BCDIN
        0
        ADDC    /C 寄存器的内容与前一个数相加。
        RET     /把该结果存入 A寄存器，
                /然后返回。
BCDIN,  CALL    /从电传打字机
        TTYIN   /取一个字符。
        0
        CPI     /它小于 ASCII 0 吗？
        060
        JC      /是的，则忽略它。
        BCDIN
```

```
          0
          CPI       /它大于 ASCII 9 吗?
          072
          JNC       /是的，则忽略它。
          BCDIN
          0
          ANI       /只保存这四位最低
          017       /有效位。
          RET       /然后，把它存入 A，返回。
CRLF,     MVIA      /把回车字符的 ASCII 值装
          215       /入 A寄存器，
          CALL      /然后打印它。
          TTYOUT
          0
          MVIA      /然后把换行字符的 ASCII 值
          212       /装入A 寄存器，
          JMP       /然后打印它。
          TTYOUT
          0
TTYIN     IN        /输入 UART 的状态位。
          001
          ANI       /只保存接收器的标识位。
          001       /如果 A＝001，则一只键按下。
          JZ        /如果 A＝000，则没有键按下。
          TTYIN     /所以，继续等待一只键
          0         /按下。
          IN        /一只键被按下，所以把
          000       /这个 ASCII 字符输入到 A 寄存器，
          ANI       /然后，把校验位(D 7)
          177       /置 0(177＝十六进制 7 F)。
```

```
TTYOUT,  MOVBA   /把这个字符保存在B寄存器。
TTYO,    IN      /输入UART的状态字。
         001
         ANI     /只保存发送器的标识位。
         004     /如果A=004，发送器（打印机）
                 /准备好
         JZ      /如果A=000，发送器（打印机）不空
         TTYO    /所以，继续等待发送空，
         0
         MOVAB   /然后才能打印A的内容。
         OUT     /把这个字符从B传送到A后，把它
         000     /输出给UART。
         RET     /把这个字符的仍留在A，返回。
```

在例3-5，堆栈指示器被装入之后，为MK 5009编程序，得到1 Hz(1秒)的工作频率，接着，把一个存储地址装入寄存器对H；这个地址将用来存储被压缩的BCD小时数；用它给时钟编程序的，然后，把十进制数3装入D寄存器，因为必须输入三个两位数字的数。电传打字机打印回车符和换行符后，8080才调入在NXTDIG处的TIMIN子程序。8080执行TIMIN子程序，输入两个ASCII数字字符，把这两个数字字符压缩，装入一个八位的寄存器。如果被送入的是两个非数字字符，TIMIN子程序会对它们置之不理。当两个数字字符已经输入并压缩送入A寄存器之后，8080从TIMIN子程序返回。当8080第一次从TIMIN子程序返回时，把A寄存器的BCD小时数保存在寄存器对H寻址的存储单元。然后，把寄存器对H的存储地址和D寄存器的数减1。如果这个数不等于0，那么在电传打字机上打印冒号，8080返回到NXTDIG，

从而能够输入分钟数。当小时数、分钟数和秒数按照该顺序都输入之后，8080 转到 CHECK。

既然表示时间的六位数字已经被送入了，那么，8080 必须决定它是否有效。这就是说：表示小时的数目不能大于 23，分钟数和秒数都不能大于 59。在 CHECK 处，把存储无效秒数的地址装入寄存器对D。当中断出现时，也使用同一个表(TOPVAL)。8080 使这个时间加 1，并且把这个新的时间与该表的项目进行比较。寄存器D被装入之后，寄存器对H 的读/写存储器地址加 1，从而寄存器对H为存储所输入的秒数的存储单元寻址。然后，把存储时间的存储单元的数目装入C寄存器。

这些寄存器被预置初值之后，8080 从 TOPVAL 表读出一个数，然后把它减 1。随即把这个数与用户从电传打字机输入的秒数进行比较。如果被输入的这个数小于或等于 TOPVAL 的被减 1 的那个数，那么，8080 执行JNC-CNEXT。在CNEXT 处，寄存器对D和H所存储的存储地址都加 1，C寄存器的数减 1。如果 8080 还没有对这三者进行比较，那么，8080 执行 JNZ-AGAIN 指令。如果所送入的数不是有效的秒数，8080 则不执行 JNC—CNEXT，而是在电传打字机上打印一个疑问号。然后，8080 返回到 START，从而能够送入六位数字表示的一个新的，希望有效的时间。

如果一个六位数字表示的时间已经送入，那么，8080不执行 JNZ-AGAIN 指令（正好在 CNEXT 之后），而是调用 TTYIN 子程序。为什么要这样做呢？如果读者给 8080 送入这样一个时间：12:05:36，这个时间精确到 1 秒。可是，读者用电传打字机把六位数时间送入到 8080 微型计算机需要占用多长时间呢？需要 4～5 秒。这就是说：用 12:05:36 这个时间给

日时钟编程序，读者应该约在 12:05:31 这一时间开始把时间数 12:05:36 送入，这样才能在该时间之前完成这个时间的输入操作，所送入的时间才能等于实际时间即实时时间。为了使这种同步操作简化，当一个有效六位数时间已经送入以后，8080 才调用 TTYIN 子程序。

这就是说：为了用 12:05:36 这个时间给日时时钟编程序，读者可以在 12:05:01 时，开始把 12:05:36 这个时间送入8080 微型计算机。一旦这个时间已被送入之后，你得等待你的表显示1 2:05:36 这个读数。当这个读数出现时，你可以按下电传打字机的任何一只打印键。这样，8080 把这个 ASCII 字符输入到A寄存器，然后从子程序返回。可是，更重要的是，8080 把中断标识位清零，并且允许中断。TTYIN 子程序调入的理由不是为了可以把一个字符输入到 8080 微型计算机，而是为了在循环中等待，直到送入的时间与实际时间相等。当这两个时间相等时，我们按下一只键，8080 从这个循环程序退出，并且从这个子程序返回。读者如果把例 3-3 这两个程序结合起来，就可以给 8080 微型计算机编程序，它就能作为可编程序的日时钟操作用。

给日时时钟添加上/下午指示器

我们可以对日时钟的程序例子作出一些修改和改进。把时钟格式从 24 小时改变为 12 小时，尽管要增添一个上/下午指示器，但是这是容易修改的形式之一。为了存储上/下午指示器内容，我们要用一个读/写存储单元。如果这个存储单元的内容等于零，那么，存储在 CURTIM 符号地址的时间是上午；如果上/下午指示器是 377(FF)，那么，CURTIM 所存储的时间是下午。如果需要的话，可以锁存一位数据位，这个数据位

输出给发光二极管显示器，表示上午或下午。

使用上午/下午指示器的日时钟程序如例 3-6 所示。为了使这个程序简短，程序中没有包括用于电传打字机的程序指令。添加上/下午指示器，这种修改方式是很简单的。在这个程序(START)的开始，MVIB 指令的数据字节已经从 003 改变为 004(03～04)。这就是说，四个存储器单元将置零：现在的时间（三个存储单元）和上午/下午指示器(一个存储器单元)。这就是说，上/下午指示器存储在读/写存储器，正好位于现行时间的小时数之后。

这个 UPONE 循环程序与前面那些日时钟子程序不同，只有在秒数或分钟数加 1 时才执行它。当小时数必须加 1 时，在 NEXT 之后的指令被执行。8080 从存储器读出小时数，将该数加 1，然后，存回到存储器。可是，如果 8080 把时间刚从 11:59:59 增量到 12:00:00，那么就必须改变上/下午指示器。为了检查这一状态，8080 执行 CPI 0.02，JZ 和 CAMPM 这些指令。此外，也有这种可能性，8080 已经把小时数从 BCD 数 12 增量到 BCD 数 13。如果是这种情况，必须把小时数再恢复到 BCD 数 01。如果在 TOPVAL 表的小时数（BCD 数 13）与 HOURS(小时)的内容相等，那么，8080 不执行 JNZ-MIDNGT 指令，而是把小时数再恢复到 BCD 数 0 1。

例 3-6 日时时钟的上/下午指示器

```
/这个程序是中断服务子程序。
/该中断硬件产生一条 RST 4 指令。

          *000  040
CLOCK,    PUSHPSW   /当中断出现时，
          PUSHB     /把各寄存器的内容
          PUSHD     /都压入堆栈。
```

```
        PUSHH
        JMP         /继续为在 1SCNT（中断服务
        ISCNT       /～继续）处的中断服务。
        0
/这个程序是日时时钟程序
/(它是 MAIN TASK 的开始部分)。
        *002  000
START,  LXISP       /装入堆栈指示器，因为
        STACK       /中断将把一个返回地址
        0           /保存在堆栈。
        MVIA        /把一个字装入A寄存器，用它来给 MK 5009
                     编程序，
        006         /使它工作于 1 Hz。
        OUT         /
        305         /然后把该字锁存，输出。
        LXIH        /把被压缩的 BCD 分钟数和秒
        000         /数装入寄存器对 H。
        000
        SHLD        /把这个时间
        CURTIM      /存入读/写存储器。
        0
        MVIL        /然后把 L 寄存器置到 BCD
        022         /数 12，以便小时数被预置初值
        SHLD        /到 12。上/下午指示器
        HOURS       /恢复成零。
        0
        OUT         /清除 MK 5009 的
        306         /中断标识位。
        EI          /启动 8080 的中断。然后
        .           /继续执行“MAIN TASK”。
```

```
/8080 转移到“ISCNT”，继续为
/每秒产生一次的中断服务。
ISCNT,  LXIH      /把表的地址装入寄存器对 H,
        TOPVAL    /该地址包括最大秒数，最大分数
        0         /最大时数。
        LXID      /然后把一个地址装入寄存
        CURTIM    /器对 D，这个地址
        0         /用来存储现在的时间。
        MVIB      /秒数和分数所用的
        002       /存储单元的数目。
UPONE,  LDAXD     /取这个数 1。
        ADI       /给这个数加 1。
        001
        DAA       /然后把该结果进行十进制调整。
        STAXD     /保存这个新时间。
        CMPM      /新时间太大吗?
        JZ        /是的。然后把该数
        NEXT      /置 0，并且把这个时间的后
        0         /两位数加 1。
MIDNGT,LXIH       /现在，显示现在时间。
        CURTIM
        0
        MOVAM     /取 BCD 秒数。
        OUT       /把这个数输出给
        002       /七段发光二极管显示器。
        INXH      /寄存器对 H 的地址加 1。
        MOVAM     /取 BCD 分钟数。
```

```
        OUT     /然后把这个数输出
        000     /给七段显示器。
        INXH    /这个存储地址加 1。
        MOVAM   /取这个 BCD 小时数。
        OUT     /然后输出这个数
        001     /给七段显示器。
        POPH    /这个时间是有效的，
        POPD    /所以，从堆栈弹出
        POPB    /各寄存器的内容。
        POPPSW
        OUT     /现在把与 MK 5009 连接的
        306     /中断触发器清零。
        EI      /8080 允许中断，
        RET     /然后返回。
NEXT,   MVIA    /把 A 寄存器置 000
        000     /(BCD 数和十六进制 00)。
        STAXD   /然后把它保存在现在时间地址。
        INXH    /这个表地址加 1。
        INXD    /现在时间地址加 1。
        DCRB    /存储单元的数目减 1。
        JNZ     /这个数不等于 0，所以
        UPONE   /可以使下一个连续存储
        C       /单元增 1。
        LDAXD   /现在，取这个小时数。
        ADI     /把这个数加 1。
        001
        DAA     /对该结果进行十进制调整，
        STAXD   /然后把它保存在存储器，
        CPI     /时间已经从 11:59:59
        022     /改变为 12:00:00 吗？
```

```
        JZ          /是的，因此，应该改变
        CAMPM       /上/下午指示器。
        0
        CMPM        /12:59:59 增量到 13:00:00 吗？
        JNZ         /没有，则小时数是有效的。
        MIDNGT      /然后 8080 从中
        0           /断返回。
        MVIA        /是的。这个时间是 13:00:00，所以，
        001         /把该时间改变为 01:00:00。
        STAXD       /把这个时间保存在存贮器
        JMP         /然后，从中断返回。
        MIDNGT
        0
CAMPM,INXD          /这个时间是 12:00:00，所以地址加 1，
        LDAXD       /得到上/下午指示器的存储地址。
        CMA         /取这个数进行比较，
        STAXD       /然后把它存回存储器。
        JMP         /既然上/下午指示器已经改变，
        MIDNGT      /所以，显示这个时间，
        0           /然后，从中断返回。
CURTIM,000          /这儿存储 BCD 秒数。
        000         /这儿存储 BCD 分钟数。
HOURS, 000          /这儿存储 BCD 小时数。
AMPM,  000          /这儿存储上/下午指示器数
                    /(上午＝000，下午＝377)。
TOPVAL,  140        /最大 BCD 秒数＝60。
         140        /最大 BCD 分钟数＝60。
         023        /最大 BCD 小时数＝13。
```

如果必须改变上/下午指示器指示的时间（11:59:59～12:

00:00）那么，8080转移到CAMPM。从CAMPM开始，存储在寄存器对D的地址加1，从而给上/下午指示器寻址。然后把这个存储单元的内容装入A寄存器；将A寄存器的内容取反，该结果存回到同一个存储单元。指示器的第一次开关将在中午出现。8080第二次执行这些指令后的时刻，将是12:00:00(即半夜)。这就是说：指示器必须从下午转换到上午，因此，存储在上/下午存储单元的377（FF)这个数取反成零。

如果需要，当现在时间被送入时（例3-5)，也可以用电传打字机把上/下午指示器重新预置初始值。为了使这个程序尽可能简单，应该把上/下午指示器的存储地址装入寄存器对H。然后，8080应该调用TTYIN子程序，从而可以打入字符A或P。如果送入的字符既不是A也不是P，那么，我们应该给8080编程序，把该字符忽略，然后，等待送入字符A或者P。如果字符A被送入，则上/下午指示器应该置零。如果字符P被送入，则上/下午指示器应该置到377(FF)。这就是说：我们应该A 10:56:03或P 02:13:23（上午10:56:03，或者下午2:13:23）这样的时间输入。

中断驱动键盘

在«8080/8085的软件设计»一书的上册的第七章，我们描述了许多不同的接口和子程序，它们都可以与键盘一起使用。但是，在所有这些程序例子中，我们给8080微型计算机编了程序，等待标识位（参考ASCII键盘的程序例子）或者等待被检测的键闭合（参考4×4和5×5扫描键盘的程序例子)。如果用这种外部设备与中断一起使用，我们就可以把软件程序中

所有的“标识位/键检测”循环程序全部删除。

在前一章，我们采用向量中断，把 ASSII 键盘与 8080 微型计算机连接起来（参考图 2-3）。我们编写了供 ASCII 键盘使用的许多中断服务于程序，并且，对它们的操作过程展开了讨论；请见例 2-2 和例 2-6。如果读者对于采用中断方式，把 ASCII 键盘与 8080 微型计算机连接的问题有兴趣的话，那么，读者可以借鉴这些例子。我们在«8080/8085 的软件设计»上册书中所讨论的另一种键盘是扫描键盘。这种键盘没有任何标识位及其与它有关的硬件编码逻辑。事实上，给键盘扫描，判明是否有键被按下，这是 8080 的任务。8080 还必须为每一只键产生唯一的一个代码。如果采用中断方式，我们可以使键盘扫描程序简化。

中断驱动扫描键盘

4×4 扫描键盘的接口如图 3-5 所示。这个接口电路的一部分与«8080/8085 的软件设计» 的上册的第七章的图 7-5 有点相相似。但是，DM 8095 的 V、W、X、和 Y 输入线也必须与四输入端的“与非”门（SN 7420）连接；这个门的输出经由两个 D 触发器输出，其中第二个 D 触发器的输出送给一个优先权编码器中断接口；这种接口在前一章已阐述过了。我们还给这个接口增添了一个 NE 555 振荡器。由于给这个接口增添了触发器和振荡器，所以使该接口具有抑制键闭合所产生的任何颤动的能力。在«8080/8085 的软件设计»的上册中，我们讨论了利用软件来消除键闭合所产生的颤动问题。但是，我们为什么习惯使用硬件来消除键闭合产生的颤动是有具体理由的；关于这

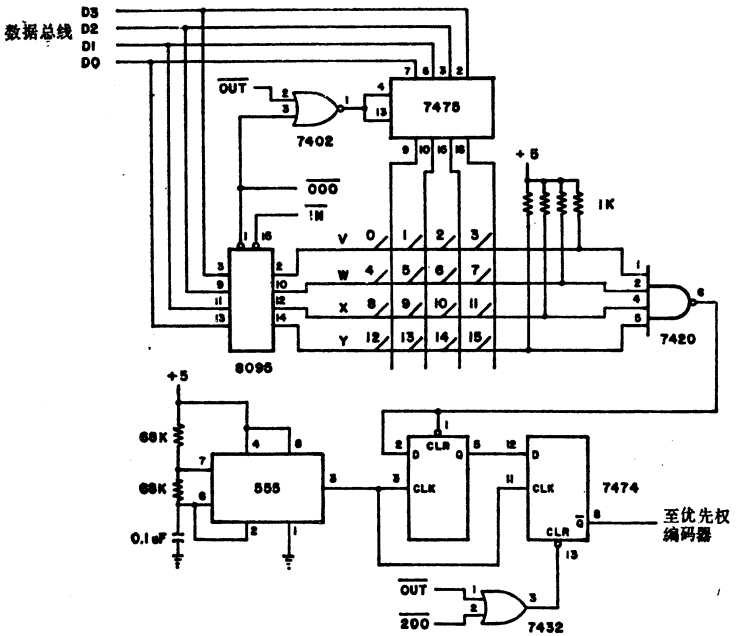


图 3-5　与中断连接的 4×4 矩阵键盘

一点将立刻进行详细讨论。

例 3-7 这个程序包含一个 4×4 矩阵的 ASCII 键盘的中断服务子程序。从这个程序表，读者可以看到，当键盘中断 8080 微型计算机时，中断接口器件必须把一条 RST 5 指令送入指令寄存器。当该键盘中断 8080 时，必须把寄存器对D的内容和 PSW 存储在堆栈，必须把中断服务子程序的其余的指令存储在 SCN 1 的存储单元。

例 3-7　4×4 扫描键盘的中断服务子程序

```
/这段MAIN TASK(主程序)使 4×4 的矩阵
/键盘复原。
```

```
          *000 000
START,    LXISP     /把一个读/写存储地址
          STACK     /装入堆栈指示器。
          0
          XRAA      /把A寄存器的内容置零。
          OUT       /把它输出给键盘
          000       /(SN 7475 锁存器)。
          OUT       /给键盘的中断
          200       /标识位清零。
          EI        /8080 允许中断，然后
          .         /执行"MAIN TASK"
          .
          .
/这个中断服务子程序为
/4×4 矩阵结构（四排，每排四只键）的
/键盘服务。
/把被按下的键的代码
/存储在读/写存储单元。
          *000  050
KEYSCN,   PUSHD     /把寄存器对 D 的内容保存在堆栈。
          PUSHPSW   /再把 A 和各标识位保存在堆栈。
          JMP       /现在，继续对在不同
          SCNI      /存储器部分
          0         /的键扫描
          *030  162
SCNI,     LXID      /把启动一排键的
          376       /测试字和第一只键代码
          003       /装入寄存器对 D。
NXTGRP,   MOVAE     /取这个测试字，
          OUT       /然后，把这个字输出
```

```
          000       /给键盘。
          RLC       /把这个测试字循环左移 1 位。
          MOVEA     /然后把它保存在 E 寄存器。
          IN        /输入四排键
          000       /的数据。
          ANI       /只保存低 4 位，
          017       /它包括这一排键的数据。
          CPI       /把 017(0F) 和这个输入字比较，
          017       /判断是否有键按下。
          JNZ       /这一排一只键被按下，
          NXTKEY    /因此决定它是哪一只。
          0
          DCRD      /被测试的这一排没有键按下，
          MOVAD     /因此使键代码加 1，检查器
          CPI       /排是否已经都被测试
          377       /(377＝十六进制 FF)。
          JNZ       /四排键还没有都被测试，
          NXTGRP    /所以，测试另一排。
          0
          HLT       /是什么引起中断？
NXTKEY,   RRC       /把这一排数据进循环移入进位。
          JC        /进位是逻辑 1，所以
          UPA       /计算可能被测试的
          0         /下一只键的代码。
          MOVAD     /把这只键代码装入 A 寄存器
          STA       /把这个键代码存入读/写存储器
          CHAR
          0
          POPPSW    /从堆栈弹出 A 和状态字。
          POPD      /从堆栈弹出 D 的内容。
```

```
        OUT         /给键盘的中
        200         /断标识位清零。
        EI          /8080允许中断
        RET         /然后返回。
UP 4    PUSHPSW     /否则，把 PSW 保存在堆栈。
        MOVAD       /把 D 存储的键代码增 4
        ADI
        004
        MOVDA       /把这个新键代码保存在 D。
        POPPSW      /从堆栈弹出 PSW。
        JMP         /然后再试，谋求零
        NXTKEY      /进位。
        0
```

这段子程与«8080/8085 的软件设计»上册第七章的程序 例 7—13 很相似。唯一的差别就在 NETKET 附近。假设 8080 被键盘中断，但是执行中断服务子程序时， 8080 没能找到 按 下的任何键。这种情况不会发生，因为大 多 数 人 不 能 用 10 或 20 μs 按下并释放一只键。但是，读者在调试这个中断 服 务子程序或键盘/中断接口硬件的过程中，可能会发生这种 情 况。如果这种情形确实出现了，D 寄存器的内容将从 003 减 到 377 (03～FF)。这一情况表示没有任何键被按下，因此， 8080 就在 NXTKEY 之前暂停。

因为在任何时候都可能有键被按下，当键被按下时，将会产生中断（除非禁止中断），所以，我们不能让 8080 把按下的键的代码存入 A 寄存器而返回。因此，在 NXTKEY 处，如果把零循环移入进位（表示一只键被按下）那么， 8080 则 不 执行 JC—UP 4 指令，而是把 D 寄存器的代码传送到 A 寄存器，

然后把这个数存入读/写存储器。接着 8080 从堆栈弹出 PSW 和寄存器对 D 的内容，并清除键盘的中断标识位。然后再次允许中断，8080 返回到被中断的程序。在稍后一些时间，被中断的程序能够读 CHAR 的内容（被按下去的这只键的键代码），确定应该执行什么操作。

正如读者可以看到的那样，例 3—7 并没有任何指令用来消除键按下或稍后键释放所产生的颤动。相反，在这个接口中，使用硬件来消除各只键闭合所产生的颤动。请记住，采用中断硬件，8080 能够以尽可能快的速度与外部设备进行通信。当8080 与外部设备进行通信完毕之后，它能返回去处理被中断的任务。因此，**我们并不需要让 8080 执行10或 20 ms 的延时子程序来降低 8080 的执行速度，从而用软件消除键闭合所产生的颤动**。正是这个原因，我们才在接口配备了消除颤动的硬件。

必须消除开关闭合和释放的颤动，这是利用软件来消除键闭合所产生的颤动的另一个问题。对于 8080 微型计算机来说，消除开关闭合所产生的颤动是很容易做到的，这正是因为键盘与 8080 的中断线连接的缘故。这就是说，把那些寄存器内容保存在堆栈之后和为键盘“服务”之前，8080 能执行一个 10 或 20 ms 的延时子程序。那么，8080 将怎样测试这只键何时被释放呢？**8080 将必须执行中断服务子程序中的一个循环程序**。如果这只键二、三秒还没有释放时，8080 在键被按的同时将用二、三秒时间执行“键检测”循环。当键最后被按下，可以调用这个延时子程序，从而消除键释放所产生的颤动。然后，8080 返回，执行被中断的任务。显然，我们不需要 8080 去执行任何中断服务子程序的这种循环程序。请读者记住，当 8080 必须尽可能快的速度为外部设备服务时，或以不定的时间间隔为外部设备服务时，应该采用中断技术。所以，不应该让一个子

程序去束缚 8080。相比之下，使用接口硬件去消除键盘的各只键闭合和释放所产生的颤动，这种办法好得多。

中断驱动多路转换的发光二极管显示器

在«8080/8085 的软件设计»上册的第七章，我们还讨论了多路转换的发光二极管显示器。供十位数字的显示管用的典型接口如图 3-6 所示。这个图就是«8080/8085 的软件设计» 上册第七章的图 7-10。可以用来驱动该显示器的程序之一，如例 3-

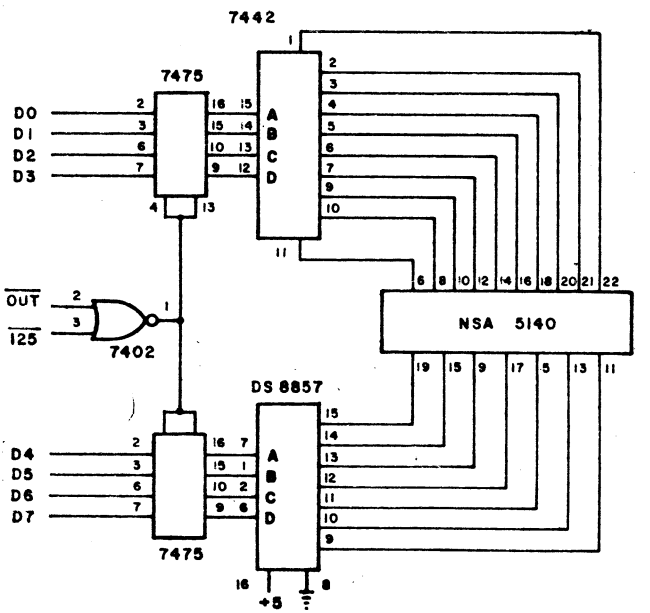


图 3-6 十位数字多路转换的 LED 显示器

8 所示。这个程序没有采用中断；它只是简单地说明数据值是怎样处理的，以及怎样输出给这种多路转换的显示器的。

例 3-8 十个数字多路转换显示器的典型程序

```
/这个程序用来驱动十个数字
/多路转换的发光二极管（LED）
/七段显示器。

DISPLA,  LXIH     /把用来存储 BCD 数字
         120      /的存储地址装入寄存器对 H。
         004      /004 120(十六进制 0450)。
         MVID     /把将要被启动的第一位数字
         000      /装入D寄存器。
DISPL 1, CALL     /显示头两位被压缩
         DIGIT    /的 BCD 数字。
         0
         INXH     /这个存储地址加 1。
         MOVAD    /取这个数字启动字送入 A。
         CPI      /把该字与第 11 位数字启动。
         012      /代码比较。
         JNZ      /十个数字还没有都被
         DISPL 1  /显示完毕,所以
         0        /显示另外两位。
         JMP      /十个数字都被显示完了
         DISPLA   /所以重复上述过程。
         0
DIGIT,   MOVAM    /取被压缩的 BCD 字
         RLC      /送入A。把 4 位最低有效位，
         RLC      /循环移入 4 位最高有效位。
         RLC
```

```
        RLC
        CALL    /然后显示这个数。
        OUTIT
        0
        MOVAM   /再取同一个字。
OUTIT,  ANI     /只保存四位最高有效位。
        360     /(360=十六进制 FO)
        ADDD    /加这个数字启动代码。
        OUT     /输出这个 8 位的数值。
        125
        INRD    /把这个数字启动代码加 1。
        RET     /然后返回。
```

但是，这种显示器需要"时常注意"。这就是说：一次只能在显示器上显示一位数字，每个数字只能显示几毫秒之久。因此，8080 微型计算机不断地改变被显示的数字。假设在某时，8080 微型计算机正在驱动多路转换的显示器时，必须执行一个对时间很灵敏的程序；例如，复杂时间控制程序或延时子程序。如果是这种情况，那么当这些指令正在被执行时，显示器上的一位数字不能连续显示 10 秒或 20 秒钟。«8080/8085 的软件设计»上册讨论了这个问题的解决办法，那就是为显示器提供电源自动开关功能，即给这个接口电子器件增添一块集成电路。如果每隔 10 ms 没有给这个显示器输出一个新数，那么，这块集成电路就把该显示器关断。如果每隔 10 ms 没有输出一个新的数字给这个显示器，那么，整个显示器将熄灭（没有显示一位数字）。这个硬件电路在«8080/8085 的软件设计»上册第七章的图 7-11 可以看到。但是，使用中断，我们可以让 8080 微型计算机去做其它的工作，同时，让数字可以在多路转换的显

示器上显示。

什么将引起 8080 微型计算机中断呢？多路转换的显示器并没有标识位来表示它需要服务。这就是说，我们要给多路转换的显示器的接口增添一个低频振荡器(如图 3-7 所示)。这个振荡器的输出信号给 D 触发器定时；该触发器与向量优先权中断接口连接。这个振荡器经常以多长时间中断 8080 微型计算机呢？为了使显示器不出现闪烁，显示器上十个数码管都必须在 1 秒钟内导通和截止大约 60 次。因为显示器一次只能显示一位数字，所以，在 1 秒钟内将有 600“字次”，即 10 个字×60 次/秒＝600 字次/秒。因此在 16.6 ms 之内，每个数字必须显示 1.66 ms。

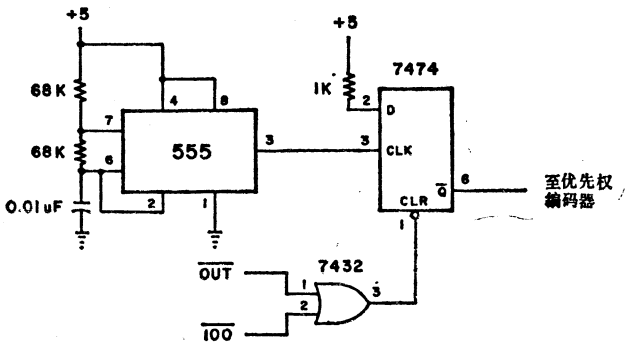


图 3-7　与中断线连接的低频振荡器

这就是说，在每秒钟内，8080 微型计算机可以被中断 60 次；每当 8080 被中断时，十位数字信息都被显示，每一位数字显示的时间为 1.66 ms。可是，如果这样做，8080 微型计算机将几乎没有时间来执行其它任务。这种情况会出现，是因为 8080 微型计算机在 1 秒钟内被中断 60 次（每次为 16.6 ms)；

中断服务子程序需要 16.6 ms 时间才能执行完毕 (10 个数字 × 1.66 ms/数字)。

还有一种方法，是每秒内中断 8080 微型计算机 600 次(每隔 1.66 ms 中断一次)；每次 8080 被中断时，数被显示在显示器的不同数位上。这种方法与上面那种方法一样不佳，就是因为每个数字必须显示 1.66 ms 之久，8080 微型计算机每隔 1.66 ms 被中断一次 (600 次/秒)。但是，我们采用后一种方法，因为这种方法正如读者从例 3-9 所看到的，是把中断用于多路转换的显示器的最佳方法。

在例 3-9 这个程序的开始，8080 就把一个读/写存储地址装入堆栈指示器。然后把一个读/写存储地址装入寄存器对 H，这个存储地址用来存储将要被显示的一个十位数的 BCD 数的最低有效位。因为我们需要显示一个十位数字的数，所以，还需要使用九个连续的存储单元来存储这个数的其余九位 BCD 数字。我们必须把每位 BCD 数字存储在每个存储单元的 4 位最低有效位(D_0-D_3)。这个十位数字的数不是以被压缩的 BCD 数的格式存储的。

一旦把第一位数字的地址装入寄存器对 H 之后，它就被存储在读/写存储器的 ADDR 和 TEMPO 处。因为我们首先需要显示的数字是最低有效位。(但是，首先显示最高有效位，一样容易)，所以，8080 把存储器单元 DIGENB 的内容置零。这将用作显示器的数字启动代码(为了得到显示器操作的更详细的资料，请参考«8080/8085 的软件设计»一书的上册第七章)。除了使用十个存储单元存储要被显示的数之外，还需要用五个存储器单元来存储 ADDR、TEMPO 和 DIGENB 这三个存储地址。当这五个存储单元已经预置初值后，振荡器的中断标识位清零，然后 8080 允许中断。于是，它可以执行该程

序的其余的指令。

在 1.66 ms之内，这个振荡器将中断 8080。中断时，8080 被引向到 000 050(0028)地址单元。在这个地址开始，8080 为显示器服务。在 PISPLA 处，把寄存器对 H 和寄存器对 D,以及处理机状态字(PSW)保存在堆栈。然后 8080 转移到 DISCNT。

在DISCNT，8080 把 ADDR 的内容装入寄存器对 H，ADDR 最初包含 NMB 这个地址，这个存储地址用来存储将要被显示的第一个数字。然后，把第一个数字启动代码(起始为 0)装入 A 寄存器，(数字启动代码存储在 D 寄存器)。最后,用这个数字启动显示器的 1 位数字。在 DISPL1 处，8080 把一个 BCD 数从寄存器对H寻址的存储单元装入到A寄存器。然后把这个数循环左移四次，行入 A 寄存器的D_7～D_4位。8080 执行 ANI 指令，把 A 寄存器的 D_3～D_0 位置零。然后，把 D 寄存器的数字启动代码加到A寄存器所存储的 BCD 数上。

例 3-9： 中断驱动的十位数字多路转换显示器

```
        *000000
START.  LXISP       /装入堆栈指示器，因为要调用子程序，
        STACK       /
        0           /中断可能出现。
        LXIH        /然后把这个存储地址
        NMB         /装入寄存器对H，该地址用来存
        0           /储这个十位数字的数。
        SHLD        /把这个数保存在读/写存储器，
        ADDR        /以便显示器的中断服务子程序能
        0           /够对它进行存取。
        SHLD        /把这个地址保存在另外两个读/
        TEMPO       /写存储器单元。
```

```
        0
        XRAA        /然后把A寄存器置零。
        STA         /把这个数保存在存储单元，作为
        DIGENB      /第一位数启动代码。
        0
        OUT         /清除振荡器
       100          /的中断标识位。
       EI           / 8080 允许中断，然后执行这个
        .           /程序的其余指令。
        .
        .
   /这是十位数字多路转换的显示器
   /的中断服务子程序。
        *000050
DISPLA, PUSHH       /把寄存器对H和D的内容
        PUSHD       /存入堆栈；
        PUSHPSW     /把处理机状态字存入堆栈。
        JMP         /然后转移到显示器的
        DISCNT      /中断服务子程序的其余部分。
        0
        *060131
DISCNT,LHLD         /把这个地址装入寄存器对H，
        ADDR        /这个地址用来存储将要被显示
        0           /的十位数字的数。
        LDA         /把将要被使用的现行数字
        DIGENB      /启动代码装入
        0           /A寄存器。
        MOVDA       /把这个代码保存在D寄存器。
DISPL1, MOVAM       /取要被显示的这个数。
        RLC         /把这个要显示
```

```
        RLC       /的数循环移入A寄存器
        RLC       /的高 4 位有效位。
        RLC
        ANI       /只保存高 4 位有效位。
        360       /(360，十六进制FO)。
        ADDD      /加这个现行数字启动代码。
        OUT       /把该结果输出给
        125       /显示器的接口。
        MOVAD     /把这个数字启动代码取列A。
        INRA      /使这个代码加 1 。
        CPI       /这位最高有效数字，
        012       /刚被显示吗？
        JNZ       /不，没有。然后把这个数字启动
        NOTYET    /代码存回到存储器。
        0
        XRAA      /是的。然后把这个代码置零。
        STA       /只把这个零保存在
        DIGENB    /存储现行数字启动
        0         /代码的存储单元。
        LHLD      /然后，把被显示的十位数字
        TEMPO     /的数的始地址
        0         /装入寄存器对H。
        JMP       /然后，只把它保存在存储器。
        EXIT      /从堆栈弹出这些寄存器的内容，
        0         /允许中断，然后返回。
NOTYET, STA       /只把A寄存器的内容
        DIGENB    /(要用的下一个数字的启动代
        0         /码)保存在读/写存储器。
        INXH      /存储地址加 1 。
EXIT,   SHLD      /然后把这个地址
```

```
ADDR      /保存在读/写存储器。
0
POPPSW    /从堆栈弹出PSW，
POPD      /寄存器对D、
POPH      /和寄存器对H的内容。
OUT       /把振荡器的
100       /中断标识位清零。
EI        /允许中断。
RET       /然后返回。
```

OUT 输出指会把 BCD 数和数字启动代码输出给显示器接口，从而，一个新的数字显示在显示器的另一个数字位上。这个 8 位的数输出给接口以后，8080 把数字启动代码传送到 A寄存器里，该代码在 A 寄存器加 1。这样，就为显示器的下一位较高有效数字产生了数字启动代码。因为显示器有十个数字，所以数字启动代码的值在 000～011 (00 和 09)之间，因此，012 (十进制 10，十六进制 0 A) 这 个数不是一个有效 数字启动代码。如果产生了 012 这个数，则供数字启动代码用，那么，最高有效数字刚刚显示。所以，下一位显示的数字必定是最低有效位数字。数字启动代码加 1 以后，8080 才作出判定。

如果这个数字启动代码不等于 012(0 A)，8080 则执行JNZ -NOTYET指令。在 NOTYE T处，8080 把加 1 后的数字启动代码保存在读/写存储器(DIGENB)，然后，寄存器对 H 所存储的存储地址加 1。这个新地址指到存储器的要被显示的下一位 BCD数；当 8080 执行在 ENXIT处的 SHCD 指令后，把 这个新地址保存在 ADDR。然后，8080 从堆栈取出处理 机 状态字 (PSW)，寄存器对 H 和 D 的内容，然后清除振荡器的中断标识位。8080 再次允许中断，然后返回，执行被中断 的任务。

如果数字启动代码增加到012(0 A)，那么，8080不执行JNZ-NOTYET指令，而是把A寄存器置零，然后把这个值保存在DIGENB。第二次，显示器振荡器中断8080时，用这个数将作为数字启动代码。这就是显示器的最低有效数字的启动代码。然后8080把存储最低有效数字的存储器地址装入寄存器对H：因为这个地址必定是被显示的下一位数字的地址。8080转移到EXIT，寄存器对的内容从堆栈弹出以前，它把这个地址保存在ADDR。寄存器的内容从堆栈弹出以后，中断标识位清零；再次允许中断；然后8080返回去执行被中断的任务。

我们已经计算过，甚至数字启动代码增加到012(0 A)时，数字启动代码和这个存储器地址ADDR必须被再预置初值时，8080执行这个中断服务子程序所需要的最大执行时间是131 μs。

如果是这样，显示器上的1位数字怎样被显示(启动)1.66 ms之久呢？由于在显示器接口(图3-6)有两个4位锁存器(SN-7475)，所以，数字启动代码和数据数被锁存或被存储在该接口，直到8080执行另一条OUT 125指令才被输出。该操作每隔1.66 ms才发生一次，因为这条OUT 125指令在该中断服子程序只用了一次。

因为8080为每隔1.66 ms产生的中断服务所需要的时间只有131 μs，所以，在另一个中断出现之前，8080有大量的时间来完成其它的任务。实际上，8080为显示器服务所用的时间只占该时间的7.8%。当然，如果这个时间太大，8080在执行任何对时间敏感的任务之前，读者必须给8080编上一条DI(禁止中断)指令，让它执行。如果这些对时间敏感的任务包括软磁盘或录音机产生的中断服务的任务，那么，我们必须用

中断硬件(图 2-9) 和中断屏蔽来禁止显示器的中断振荡器。如果以 8085 微处理机为基础的微型计算机正在使用显示器，那么，我们可以让 8085 执行一条 SIM 指令，改变中断屏蔽位的状态。

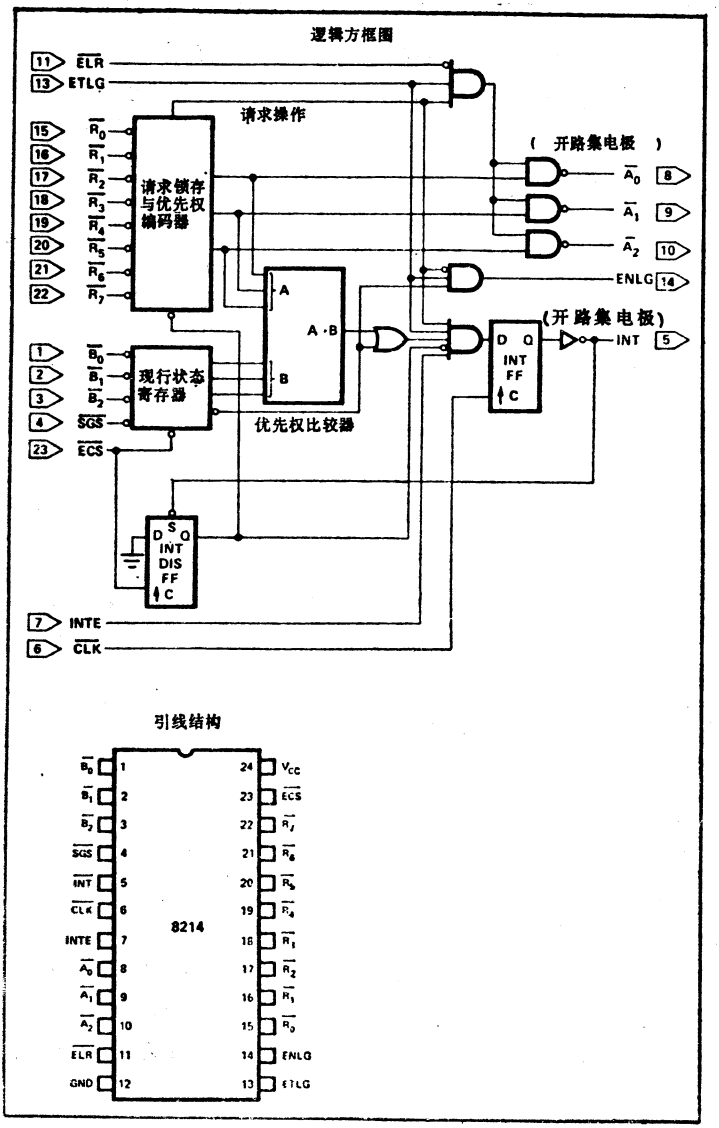
在本章的下一节，我们要讨论 8214 中断控制器的特征，它是一种集成电路的中断控制器，专为 8080 微型计算设计的。

8214 优先权中断控制器

8214 优先权中断控制器(PICU) 所具有的一些特征与图 2-9 所示的那个电路的特征很类似。8214 优先权中断控制器的逻辑电路方框图和引线结构如图 3-8 所示。

8214 优先权中断控制器集成电路包含一个优先权编码器，这个编码器有 8 个有效低输入端子。它们是$\overline{R}_0$～$\overline{R}_7$。输入端子$\overline{R}_7$的中断优先权最高，$\overline{R}_0$的中断优先权最低。这个优先权编码器所产生的代码与**现行状态寄存器**的输出在内部进行比较。现行状态寄存器是可以编程序的。如果这个编码器的输出电平比现行状态寄存器的输出电平高。那么，这个优先权中断控制器的中断输出，引线$\overline{INT}$将置成逻辑 0 。

我们可以把若干个 8214 並联在一起，因此，8080 微型计算机可备有任何数量的优先权编码中断端。正是这个原因，这个控制器电路，才有两个并联输入端子 (ETLG,$\overline{ELR}$) 和一个并联输出端子($\overline{ENLG}$)。对于只需要八个优先权向量中断的系统来说，ETLG引线(引线B)与＋5伏(Vcc，引线 24)连接，$\overline{ELR}$引线 (引线11) 接地。



引线结构

图 3-8 英特尔 8214 优先权中断控制器

引 线 名 称

输入	
$\overline{R_0 \cdot R_7}$	请求电平(R_7中断优先权最高)
$\overline{B_0 \cdot B_2}$	现行状态
$\overline{SGS}$	状态组选择
$\overline{ECS}$	启动现行状态
INTE	中断允许
$\overline{CLK}$	时钟(INT F·F)
$\overline{ELR}$	启动电平读
ETLG	启动电平组
输出	
$\overline{A_0 \cdot A_2}$	请求电平
$\overline{INT}$	中断(有效低)]——开路集电极
ENLG	下一个电平组赋能

当启动 8080 的中断时，中断允许信号 (INTE) 是由微处理器集成电路产生的 (引线 16)。这个信号以及时钟信号必须与 8214 连接。时钟可以是任一高频时钟，可高达 12MHz。如果我们把 8224 时钟发生器与 8080 集成电路微处理器一起使用，那么，8224 所产生的 ϕ_2 (TTL) 信号可以用作为 8214 的时钟。对于 8085 微型计算机系统来说，CLK OUT (时钟出) 信号可以用来给 8214 定时。正如我们从逻辑方框图 (图 3-8) 所看到的那样，时钟信号可以用来选通 8214 集成电路内所包含的五输入端的"与门"的输出，送入D触发器。该触发器的输出用来表示外部设备是否需要服务。

8214 的编码器部分的输出端子只有三个。这就是说，需要增添接口电路，才能为每个中断服务产生一条各不相同的再启动指令。可以用来产生这种再启动指令的一种方法如图 3-9 所示。

8214 的编码输出 ($\overline{A_0}$，$\overline{A_1}$ 和 $\overline{A_2}$) 送到 8212 的三个输入端

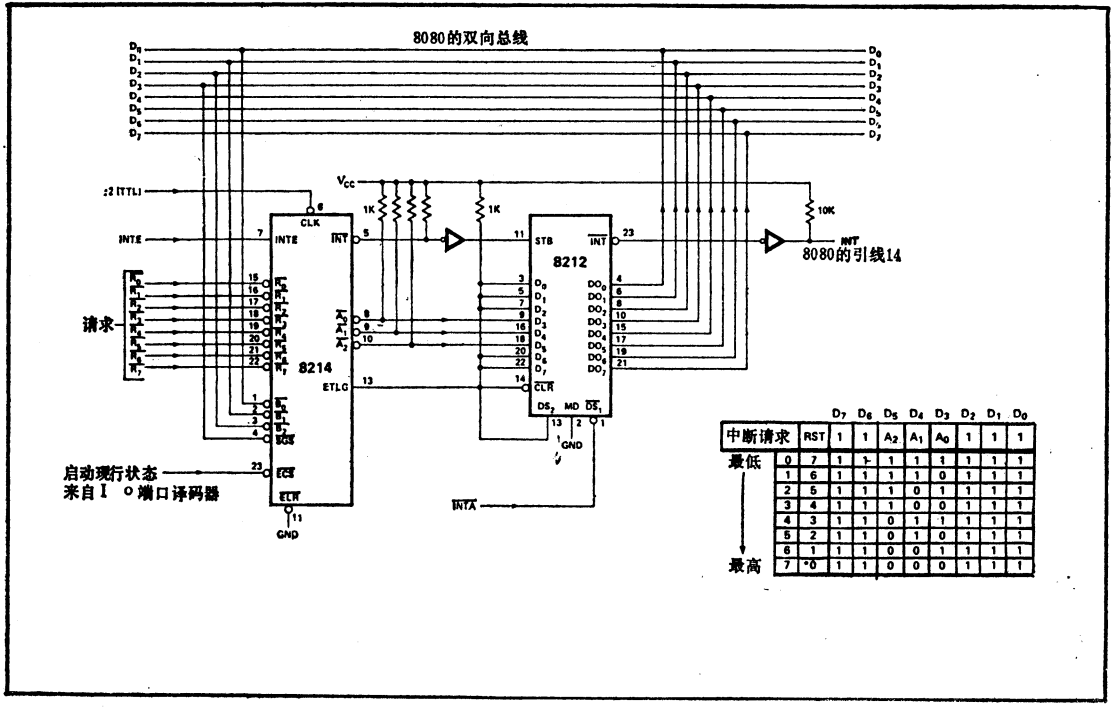


图 3-9 八级中断控制用 8214

* RST 0 把程序计数器引导到 0 号单元，并且调用同一个子程序，作为 8080 的复位输入。根据所调用的子程序，可能使系统再恢复。（系统程序设计员应该警惕）

(D_3～D_5)。8212 只用作为一个 8 位中断指令寄存器。由于 8214 的内部逻辑关系，当八个中断输入端子，其中有一个置逻辑零时，$\overline{INT}$ 输出在一个时钟周期是有效低电平信号。因此，$\overline{INT}$ 输出信号反相，然后连接到 8212 的选通 (STB) 输入端子。这就是说，当 8214 的八个中断输入端子有一个的逻辑电平为 0 时，8212 的 $\overline{INT}$ 输出端子将置成逻辑 0。8214 的四个输出端 ($\overline{INT}$，$\overline{A_0}$，$\overline{A_1}$ 和 $\overline{A_2}$) 各需要一只 1000 Ω 的电阻，因为它们是集电极开路输出端子。

因为 8080 的中断引线 (INT，引线 14) 置成逻辑 1 时，8080 才被中断，所以 8212 的 $\overline{INT}$ 输出必须反相。读者注意，中断响应信号 ($\overline{INTA}$) 仍然用来选通数据总线上的 8 位再启动指令操作码。

8214 集成电路内的现行状态寄存器用来确定哪一个中断输入 ($\overline{R_0}$—$\overline{R_7}$) 将使 8080 中断。只要把一个数值写到现行状态寄存器，在这个指定值以上的中断可以出现。例如，把 004 (04) 写到现行状态寄存器，则可以允许 $\overline{R_4}$，$\overline{R_5}$，$\overline{R_6}$ 和 $\overline{R_7}$ 中断。为了只允许 $\overline{R_7}$ 中断 $\overline{R_7}$，则必须把 001(01) 输出给 8214。遗憾的是，我们不能给 8214 编程序，来只允许 $\overline{R_0}$，$\overline{R_3}$，$\overline{R_4}$ 和 $\overline{R_7}$ 中断，和禁止中断 $\overline{R_1}$，$\overline{R_2}$，$\overline{R_5}$ 和 $\overline{R_6}$。这种情况在功能上与前一章所讨论的中断屏蔽是类似的。但是，对于我们给出的先进的中断控制器 (如图 2-9 所示) 来说，可以允许和禁止任何组合的中断。8085 微处理器集成电路芯片上制造了三条可屏蔽的中断引线 (RST 7.5，RST 6.5 和 RST 5.5)，它们也可以用来允许和禁止任何组合的中断。

如果我们把 007 (07) 输出给现行状态寄存器，那么除了 $\overline{R_0}$ 以外，其余所有的中断线都允许中断，这就是 8214 的不寻常的性能特征之一。用 8214 的 $\overline{SGS}$ 输入端可以用来解决这个

问题。如果$\overline{SGS}$输入端的电平为逻辑1，那么，不管$\overline{B_0}$，$\overline{B_1}$和B_2这三个输入端的状态如何，各个中断输入线（$\overline{R_0}$—$\overline{R_7}$）都允许中断。如果$\overline{SGS}$输入端的电平为逻辑零，那么$\overline{B_0}$，$\overline{B_1}$和$\overline{B_2}$这三个输入信号决定这些中断输入的"截止点"。

为了给现行状态锁存器编程序，它的四条输入线与数据总线连接。只要给**启动现行状态**引线（$\overline{ECS}$，引线23）加给脉冲选通信号，就可以把数据总线上的内容选通，进入现行状态寄存器。所使用的这个脉冲必须是$\overline{OUT}$（$\overline{I/OW}$）和一个译码的设备地址的组合，或$\overline{MEMW}$和一个译码设备地址的组合。正如图3-8的逻辑图所示的一样，不能把现行状态寄存器的内容读回到8080微型计算机。

可以输出给现行状态锁存器的数及其它们允许的中断，我们归纳列入了表3-4。图3-8所示的接口产生的再启动指令，经归纳后，列在表3-5中。表3-5所列出的值并不一定是读者所希望的。在第二章，最高优先权中断设备总是给出一条RST

表 3-4　8214的现行状态寄存器数值

$\overline{SGS}$	$\overline{B_2}$	$\overline{B_1}$	$\overline{B_0}$	效　　果
0	0	0	0	中断输入都是无效的。
0	0	0	1	$\overline{R_7}$是有效的。
0	0	1	0	$\overline{R_7}$和$\overline{R_6}$是有效的。
0	0	1	1	$\overline{R_7}$～$\overline{R_6}$是有效的。
0	1	0	0	$\overline{R_7}$～$\overline{R_4}$是有效的。
0	1	0	1	$\overline{R_7}$～$\overline{R_3}$是有效的。
0	1	1	0	$\overline{R_7}$～$\overline{R_2}$是有效的。
0	1	1	1	$\overline{R_7}$～$\overline{R_1}$是有效的。
1	×	×	×	所有的中断输入都有效。

×为任意；逻辑1或逻辑0。

表 3-5 中断硬件产生的再启动指令

输入接地	优先权	再启动指令
$\overline{R_7}$	最高	RST0
$\overline{R_6}$	•	RST1
$\overline{R_5}$	•	RST2
$\overline{R_4}$	•	RST3
$\overline{R_3}$	•	RST4
$\overline{R_2}$	•	RST5
$\overline{R_1}$	•	RST6
$\overline{R_0}$	最低	RST7

7 指令。如果使用以 8214 和 8212 为基础的接口，那么，最高优先权中断设备（$\overline{R_7}$）产生一条 RSTO 指令。

正如我们已经讨论过的那样，较低中断优先权的设备不能中断较高中断优先权设备的服务；但是较高中断优先权的设备应该能够中断较低中断优先权设备的服务。假设有一个微型计算机系统，该系统通过接口与一个软磁盘和一个电传打字机连接。如果 8080 正在为电传打字机服务，那么，软磁盘能够中断 8080 微型计算机；但是，当 8080 正在为软磁盘服务时，电传打字机则不能中断 8080 微型计算机。为了用 8214 实现这种操作，可以在中断服务子程序执行指令，给现行状态寄存器再编程序。

如果我们把软磁盘与 $\overline{R_5}$ 输入端连接，那么，应该把电传打字机与 $\overline{R_0}$ ～ $\overline{R_4}$ 中的任一个输入端连接。当软磁盘中断 8080 时，002（02）这个数应该输出给 8214，只有较高中断优先权（$\overline{R_6}$ 或 $\overline{R_7}$）设备才能中断 8080 微型计算机。这样，就能防止电传打字机（设备 $\overline{R_2}$）在 8080 还在为软磁盘服务时，中断 8080 微型计算机。当 8080 为电传打字机服务时，（设备

表 3-6 8080正在为中断服务时，现行状态寄存器的数值

中断	现行状态寄存器的新数			允许的中断
	二进制	八进制	十六进制	
$\overline{R_7}$	00000000	000	00	无
$\overline{R_6}$	00000001	001	01	$\overline{R_7}$
$\overline{R_5}$	00000010	002	02	$\overline{R_7}$和$\overline{R_6}$
$\overline{R_4}$	00000011	003	03	$\overline{R_7}$～$\overline{R_5}$
$\overline{R_3}$	00000100	004	04	$\overline{R_7}$～$\overline{R_4}$
$\overline{R_2}$	00000101	005	05	$\overline{R_7}$～$\overline{R_3}$
$\overline{R_1}$	00000110	006	06	$\overline{R_7}$～$\overline{R_2}$
$\overline{R_0}$	00000111	007	07	$\overline{R_7}$～$\overline{R_1}$

$\overline{R_2}$)，应该把什么数输出给 8214 呢？只有把 005(05) 这个数输出给 8214，较高中断优先权设备才能中断 8080（参考表 3-6)。电传打字机和软磁盘被服务时，重新给 8214 编程序的指令如例 3-10 所示。

例 3-10 在为中断设备服务之前改变现行状态寄存器的内容

```
          *000000
START,    MVTA      /把 1000 (SGS=1) 装入现行
          010       /状态寄存器，以便八个外部设
          OUT       /备的任何一个都能中断 8080
          032       /微型计算机。
          EI        /8080 允许中断。
          JMP       /然后执行 MAIN TASK 主程序。
          MT
          0
          *000020
FLOPPY,   PUSHPSW   /保存 A 的内容和标识位。
```

```
        MVIA    /然后，把现行状态寄存器
        002     /的一个新数装入 A 寄存器。
        OUT     /把该数输出给 8214
        032     /（允许 R6 和 R7 中断）。
        EI      /8080 的允许中断。
        .       /当 8080 微型计算机正在给
        .       /软磁盘服务时，只有
        .       /设备 R6 和 R7 可以中断 8080。
        MVIA    /当这样做以后，经改变
        010     /现行状态寄存器的内容（SGS=1），
        OUT     /启动 8 条中断线。
        032
        POPPSW  /恢复 A 的内容和标识位。
        EI      /8080 允许中断。
        RET     /然后返回到"MAIN TASK"。
        *000050
TTY,    PUSHPSW /把处理机状态字保存在堆栈。
        MVIA    /然后把现行状态寄存器的一个新数
        005     /装入 A 寄存器。
        OUT     /输出这个数，R3～R7
        032     /仍然可以中断
        EI      /8080 微型计算机。
        .       /然后，为电传打字机服务。
        .
        .
        MVIA    /当服务完后，经改变现行状态
        010     /寄存器的内容（SGS=1），来
        OUT     /启动八条中断线。
        032
        POPPSW  /然后从堆栈弹出 PSW。
```

```
EI          /8080 允许中断，
RET         /然后返回。
```

如果 8080 正在为中断设备 $\overline{R_7}$ 服务，那么，哪台设备可以中断 8080 呢？当 8080 正在为中断设备 $\overline{R_7}$ 服务时，任何一台设备都不能中断 8080，因为中断设备 $\overline{R_7}$ 的中断优先权最高。8080 微型计算机正在为这台设备服务时，我们可以给 8214 的现行状态寄存器输出一个 0。可是，并不需要这样做。为什么呢？当任何一台设备中断 8080 时，8080 的内部中断自动地被禁止。因此，直到 $\overline{R_7}$ 的中断服务子程序被执行完毕，才再次允许中断。这样，当 8080 正在为这台外部设备服务时，就能防止任何其它的外部设备中断 8080 微型计算机。

最近生产的集成电路中断控制器，8259 可编程序的中断控制器，现在已广泛使用；它除可以完成 8214 的各种功能之外，还可以完成其它一些功能。但是，由于这个器件的复杂性，因此，本书不对它进行讨论。可是，我们应该指出，虽然 8214 是以比较老式的设计为基础的，但是，它也许能够解决 95% 的 8080 和 8085 的中断应用问题。

到现在为止，我们已经讨论了许多中断应用问题。正如读者已经看到的那样，中断服务子程序有些可能是非常简单的，有些也可能是非常复杂的。但是，大多数中断服务子程序都是非常复杂的，因为中断可能随时出现。这就是说，如果中断服务子程序编写得不适当，或者如果中断用的硬件电路有错误，那么对错误定位可能是很困难的。由于这一原因，除非读者绝对需要使用中断的情况下，最好避免使用它们。但是，中断应该用得得当，它们确实十分常用，“因为中断存在”。许多微处理器和微型计算机生产厂有中断控制器出售，可以用它们把

多达 64 台外部设备与中断线连接。如果把许多外部设备与中断线连接，那么，你们将要用去大量的时间，才能使整个系统（包括硬件和软件）工作。处理八台或十六台微型计算机，每一台控制八台或四台中断设备或许是容易的。

第四章　数据结构

关于必须对大量数据进行存取的问题，在前面这几章里，我们还没有碰到过。如何从存储器的内容检索具体数的问题，我们没有讨论；把存储在存储器的数字值排序的问题，我们也没有讨论。所有这些问题，我们将在下面几章进行阐述。因此，可以用来把数据存储在微型计算机的存储器或存储在外部设备的不同的数据结构，我们必须予以讨论。

请读者注意，本章的标题是**数据结构**。柯鲁斯(Knuth)给数据的定义如下：

数据(data)——**(这个字的复数形式原来是"datum"，但是现在，"data"既可以作单数用，也可以作复数用)：以简明的，公式化的语言对于某些事实或概念的表示，常以数字值或字母值的形式出现，并能用计算方法进行处理。**

我们可以从某种计算结果得到数据，或者从软磁盘读出数据，或者从模-数转换器得到数据。我们也可以把数据存入存储器，这些数值用 ASCII 码表示，代表人的名字和年龄。所以，我们将要进行讨论的数据结构可以供数字信息或字母数字信息使用。

当然，8080微处理机(和所有其它的 8 位微处理机)可以处理或使用的信息的最小单位是**位**。位或者可以用逻辑 1 表示，或者用逻辑 0 表示。这样的一位信息可以告诉我们门是开的还是关的，或者告诉我们箱内的液体是不是满的。当 8 位信息一起组合成一组时，就得到了一个 8 位的**字节**，即一个**字**。因为

8080 的存储器的“宽度”只有 8 位，所以，我们可以在每个存储单元存储一个字节或一个字。当然，对于不同的程序，字节可以代表不同类型的信息。例如：11000011这个字节可以代表某个具体子程序已经被调用的次数；它也可以代表ASCII 字符 C；它也可以表示与某台外部设备连接的四个开关处于逻辑 0 状态，另外四个开关处在逻辑 1 状态。把若干字节分组或协调在一起，我们就可以得到**数据表，数据串，数据阵列，数据表格或数据树**。

柯鲁斯这样写道：“计算机程序通常用来处理信息表。在多数情况下，这些表并不是数字值的无定形的简单集合；它们包括数据项之间的重要的结构关系。”表的数据项之间的关系可以包括下列项目之一：n 项比 m 项大，比 0 项小。如果表中的项目是从模-数转换器得到的，那么，在这几个项目之间或许存在着时间关系。这或许可以说，从模-数转换器读出 e 项之前的一些时间，从模-数转换器读出 d 项。

另外，柯鲁斯告诉我们：“表中的信息包括一组节点(有些作者称为‘记录’，‘项’或‘小珠’)”。我们常常用“项”这一名称，而不用“节点”，因为在我们的例子中，“项”稍多地带有说明性。什么叫做“节点”呢？

节点——**每个节点包括一个或多过连续计算机存储器字，分成命了名的若干部分，称之为字段，如果一个节点正好是一个存储器字，则正好只包括一个字的字段；这是最简单的情况。**

如果从模-数转换器读了八位数据值，并且以表的形式存入存储器，那么，每个节点可能包括一个数据字。在这种情况下，我们把字段叫做 ADC DATA(ADC 数据)或简称 DATA (数据)。

存储器单元	字　段
003 100	DATA}节点
031 250	DATA}节点
067 345	DATA}节点

如果把从模-数转换器读数据的时间也存入存储器，那么可以使用两个存储器单元；一个存储器单元用来存储数据（DATA 字段），另一个存储器单元用来存储读这个数据的时间（TIME 字段）。

存储单元	字　段	
003 200	DATA	Node（节点）
003 201	TIME	
031 250	DATA	Node（节点）
031 251	TIME	
067 345	DATA	Node（节点）
067 346	TIME	

当然，如果用一个 16 位数代表 TIME（时间），那么，该 Node（节点）需要三个存储单元；一个存储单元用来存储数据字段，其余两个存储单元用来存储时间字段。为了把一个八位的时间字节与另一个时间字节区别开来，我们可以给时间字节的一部分加上标号“TIME MSB”，给时间字节的另一部分加上标号“TIME LSB”。

存储器单元	字　段	
003 103	DATA	Node（节点）
003 104	TIMZ LSB	
003 105	TIME MSB	

或

003 103	DATA	Node（节点）
003 104	TIME MSB	
003 105	TIME LSB	

“节点的地址也称之为连接地址，指示字或这个节点的参考点，它是节点的第一个字的存储单元。这个地址常常被认为是相对于某个‘基础’单元的……”但是，在有些情况下，读者将也会知道具体节点的绝对地址。例如，下面这条指令

```
LXIH
217
016
```

可以用来把这个表里的第五个节点（项）的地址装入寄存器对H。当然，如果把这个表在存储器中传送必须改变LXIH指令的地址字节（第二和第三个字节），

线 性 表

“线性表是一组n个节点X(1)，X(2)，···X(n)的集合，它的结构特性实质上只包括这些节点的线性的（一维的）相对位置；事实上是：如果n>0，那么X(1)就是第一个节点；当1<K<n，第K个节点X(k)则位于X(k−1)之前，X(n)是最后一个节点。”

顺 序 分 配

“在计算机内，保持线性表的最简单最自然的方法，是把

表的项目按顺序存入存储单元，一个节点接着一个节点存储。因此，就有：

$$Loc(X(j+1))=Loc(X(j))+C,$$

其中C表示每一个节点的字数。（通常C等于1。当C大于1时，为了把节点X(j)的第k个字存储在离X(j)的第一个字的存储单元一定的距离，把一个表分成若干个与C‘并行’的表，有时更为方便。但是，我们将继续假设C个字相临的组形成一个节点。）总之

$$Loc(X(j))=L0+Cj,$$

其中L0为一个常数，叫做基地址，是人为地假设的节点X(0)的存储单元。”

连接分配

“如果我们不把一个线性表的内容按顺序存储在存储单元，我们可利用一个灵活得多的分类表。这个分类表中，每一个节点有一个与这个表的下一个节点的连接，如图4-1所示。图中的A，B，C，D和E是存储器的任意存储单元，符号“∧”表示空连接。在按顺序分配的情况下使用这个表的程序必须有一个附加的变量或常量，其值表示这个表的长度是五项，否则，应该由第五项的或下一个单元的标记代码来指定信息。连接分配的程序必须有一个连接变量或常量，用来指示A，和从A可以找到表中的其它各项。”

标记——**是表中的一个特定的数；例如，给表的边界做标志，是为了让辅助程序能够容易辨认而设计的。**

有时，我们使用“表结束符”这个术语，而不使用“标记”。

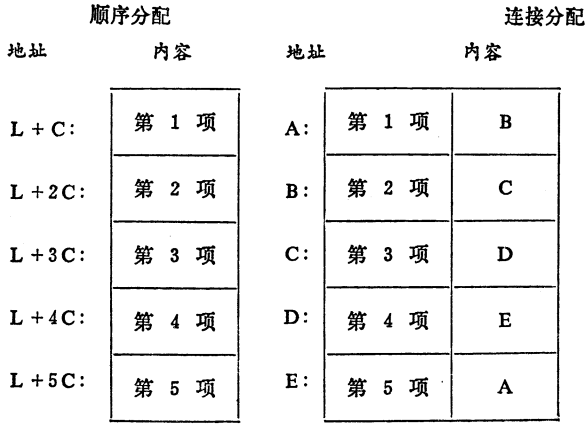


图 4-1 按顺序的表和连接的表

空连接——**空连接是由比较容易辨认的数来表示的；该数不能是某个节点的地址。**

循 环 表

“稍微改变一下连接的方式，就可以为我们提供一个重要的方法；这个方法与前面一节所讨论的方法不同。循环连接的表（简称循环表）的特点是它的最后一个节点与表的首项连接，而不是与“∧”连接。因此，在任何给定的点上开始，都可以存取表的各个项；我们也可以得到很好的对称性。如果习惯了，那么我们不需要考虑这个表有“最后的”节点或“开始的”节点。循环连接表的例子如图 4-2 所示。

连接表或循环表，虽然我们不去讨论它们，但是，核心内容描叙过了。读者如果正在使用那些按顺序存储在存储单元的

循环连接的表

地址	内容	
A:	第 1 项	B
B:	第 2 项	C
C:	第 3 项	D
D:	第 4 项	E
E:	第 5 项	A

图 4-2 循环连接表

线性表，你可以发现，连接的表或循环的表对于一些应用值得考虑。还有许多数据结构，读者应该参考柯鲁斯的著作中有关这个问题的权威性论述。

读者将在本书的其余章节中可见到的其它术语，定义如下：

矩阵——**矩阵是一种表，它常常有n维矩形结构。一维矩阵就是一个线性表。**

数据结构——**是一种包括结构关系的数据表，**

排序——**以特定的顺序重新安排给定的一组客体的过程。**

字符串——**由零或其它许多符号所组成的有限序列。**参考本章线性表一节。

符号处理——**数据处理的广义术语，通常用于非数字的处理；如字符串的处理或代数式的处理。**

第五章 检 索

在某些程序设计中，我们或许要从一个表格或表中找某一特定值，或者找最大或最小的值。例如用计算机对评分。这意味着最终要编写一个程序，使计算机能找出某个学生的名字和打印出该生各门课的分数。在这个例子中，表的一个节点（一项）可包含名字和分数两部分。

在另一种场合，微型计算机可用在炼油厂。微型计算机在24小时内每隔10分钟检查一次分裂蒸馏塔的温度。在24小时周期完了时，微型计算机在其存储器中存入了144个温度数据。如果这些温度数据是依次存入顺序的存储单元，那么，很容易确定何时达到某一特定温度，或蒸馏塔的温度超过300°C的时间。我们也许还希望为微型计算机编程序，以找到出现过的最低或最高温度。这样，就可以把这些温度与从蒸馏塔产生的分馏石油化学产品成分的任何变化联系起来。我们也许还想知道塔温超过或低于某一给定温度的次数。

所有这些问题，只要给8080微型计算机编出程序，用查表的方法，就可以得到解决。这些表的内容可以包括数字的，字母数字的或字母信息。

单精度表(8位)

从表中查出最小或最大的无符号数据值

我们能编写的最简单的子程序之一，可以用来找出表中最小的无符号8位数。为了使这个子程序尽可能具有通用性，我们应能任意规定表的始地址以及表的节点数目(项数或8位数的数目)。例5—1列出的子程序具有这些特性。

例5-1 找出一个表中最小的无符号8位数

```
        /这个子程序将从一个表中找出最小的无符号
        /的8位数。调用它时先把表地址装
        /入寄存器对H，
        /把节点数装入寄存器对D。
        /当8080返回主程序时，最小
        /的无符号数存在B寄存器。
SMU 8,      MOVBM     /把一个数装入B寄存器。
NEXTU 8,    INXH      /表地址加1。
            DCXD      /节点数减1。
            MOVAD
            ORAE
            RZ        /如果节点数为零，则返回。
            MOVAM     /否则，从存储器取一个数到A。
            CMPB      /把这个数与B内的数进行比较。
            JC        /若A寄存器内的数小于
            SMU 8     /B寄存器的数，则
            0         /从存储器取数到B寄存器。
```

```
JMP      /若 B 寄存器内的数较小，
NEXTU 8/则检查表中的
0        /下一个数。
```

调用这个子程序(SMU 8)时必须将表的始地址装入寄存器对 H 和将节点数装入寄存器对 D。子程序的第一条指令用来把表的第一个节点装入 B 寄存器。接着表地址加 1，寄存器对 D 内的节点数减 1。如果寄存器对 D 的内容变为零，8080 将从该子程序返回，同时在 B 内含有找到的最小值。如果寄存器对 D 的内容未减到零，8080 把表的第二个节点传送到 A 寄存器，并与第一个节点进行比较。如果 A 寄存器的内容小于或等于 B 寄存器的内容，8080 执行 JC 指令，转到 SMU 8。这就是说，必须把寄存器对 H 寻址的表节点传送到 B 寄存器，因为它是两数中较小的一个。反之，如果存储器内的节点比 B 寄存器内的节点大，8080 就返回到 NEXTU 8，使表中的下一个节点和 B 寄存器内的节点进行比较。

为了使 8080 能够找出表中的最大节点(而不是最小节点)，子程序应怎样改变呢？最简单的办法是把 JC 这条指令改成 JNC 指令。经过这一改变的子程序如例 5—2 所示。这就是说，当 8080 调用例 5—2 子程序时，如果 A 寄存器的内容等于或大于 B 寄存器的内容，则 8080 只执行 JNC 指令。也就是说，如果两个节点中较大的一个是在存储器中，子程序便把它从存储器传送到 B 寄存器。

例 5-2 找出一个表中最大的无符号 8 位数

```
/这个子程序从一个表中找出最大的
/无符号的 8 位数。调用这个子程序时先把表
/地址装入寄存器对 H，把节点数装入寄存器
```

```
    /对 D。当 8080 从子程序返回时，
    /最大的数存在 B 寄存器。
LRGU 8,   MOVBM/从存储器取一个数到 B 寄存器。
NEXTU 8,  INXH   /表地址加 1。
          DCXD   /节点数减 1。
          MOVAD
          ORAE
          RZ     /如果节点数等于 0，则返回。
          MOVAM/否则，从存储器取一个数到 A。
          CMPB   /把它与 B 中的数比较。
          JNC    /A 寄存器的内容大于 B。
          LRGU 8 /寄存器的内容，所以
          0      /把存储器里的这个数传送到 B。
          JMP    /B 寄存器内的数较大，则
          NEXTU 8/检查表内
          0      /下一个数。
```

同例 5-1 一样，在调用例 5-2 的子程序时，也必须将表的始地址装入寄存器对 H，把节点数装入寄存器对 D。当然，我们可以编写一个程序，使之既能找出表中最小的，也能找出其中最大的无符号 8 位节点，而不必用两个几乎相同的子程序。

找出表中最小的和最大的无符号 8 位节点

例 5-3 的子程序可以用来寻找表中最小和最大的无符号 8 位数节点。正如上两个子程序一样，在调用此子程序之前，必须把表的始地址装入寄存器对 H，把节点数装入寄存器对 D。当 8080 从该子程序返回时，最小的无符号 8 位节点将被装在 B 寄存器里，最大的无符号 8 位节点(数)装在 C 寄存器里。

在例 5-3 的程序中，8080 微处理机把第一个节点（数）装入 C 寄存器和 B 寄存器。然后，表地址加 1，节点数减 1。如果节点数等于 0，8080 从该子程序返回，同时最小的无符号 8 位数存在 B 寄存器，最大的无符号 8 位数存在 C 寄存器。如果节点数减 1 后不为零，则 8080 把表内第二个节点送到 A 寄存器，然后把它和 B 寄存器的内容比较。如果 B 寄存器的内容较大，8080 转到 NEWSML，把两数中较小的一个从存储器传送到 B 寄存器。

例 5-3 找出一个表中最大的和最小的无符号 8 位数

```
/这个子程序从一个表中寻找最大的
/和最小的无符号 8 位数。在调用此子程序时
/先将表地址装入寄存器对 H 和将节点数装入
/寄存器对 D。当 8080 返回时，最大的数存
/在 C 寄存器，最小的数存在 B 寄存器。

SMLRU 8, MOVCM      /从存储器取一个数到 C 寄存器。
NEWSML,MOVBM        /从存储器取同一个数到 B 寄存
NEXTU 8, INXH       /器。表地址加 1。
         DCXD       /节点数减 1。
         MOVAD
         ORAE
         RZ         /节点数为零时返回。
         MOVAM      /从表中取一个数到 A 寄存器。
         CMPB       /把这个数与"最小"的数比较。
         JC         /存储器的内容比 B 寄存器的
         NEWSML     /内容小，所以把存储器的内容
         0          /送到 B 寄存器。
         CMPC       /这个数比 B 寄存器的数大，所
         JC         /以把它与存储器中"最大"的数比
```

```
NEXTU 8/较。存储寄内容小于C寄存器
0          /的内容，所以取下一个数。
MOVCM /把存储器中较大的数送到
JMP       /C寄存器，然后检查表的
NEXTU 8/下一个数。
0
```

如果从存储器传送到A寄存器的节点等于或大于B寄存器的内容，则8080不执行JC指令，而是比较C寄存器中的节点与A寄存器的内容(来自存贮器)。如果A寄存器的内容小于C寄存器的内容，则8080转到NEXTU 8，从而可检查表中的下一个节点。如果A寄存器的内容等于或大于C寄存寄的内容，则8080执行MOVCM指令，把较大的节点从存储器传送到C寄存器，然后转到NEXTU 8，检查表中的下一个节点。

读者可从这个例子看到，同时寻找最大的数和最小的数，和单独寻找最大的数或最小的数相比，几乎一样容易。

寻找一个表中最小的带符号(2补码)的8位节点

当然，在某些应用中，我们必须使用带符号的数。例如与微型计算机连接的模-数转换器可产生带符号的数字值，或者所要处理的温度范围在－100℃～＋340℃之间。表5-1是带符号的8位数的一览表。从该表可见，一个8位数所能表示的带符号数的范围是－128～＋127。我们把－128(八进制200。十六进制80)看成为最小的带符号的8位数，把＋127(八进制177，十六进制7F)看成为最大的带符号的8位数。

如同本章第一节所介绍的情况一样，寻找一个表中最大的节点或最小的节点是最容易的。因此，例5-4的子程序只能

表 5-1 带符号的 8 位数一览表

带符号的数	二进制数	八 进 制 数	十六进制数
+127	01111111	177	7 F
+126	01111110	176	7 E
+125	01111101	175	7 D
•	•	•	•
•	•	•	•
+2	00000010	002	02
+1	00000001	001	01
0	00000000	000	00
−1	11111111	377	FF
−2	11111110	376	FE
•	•	•	•
•	•	•	•
−126	10000010	202	82
−127	10000001	201	81
−128	10000000	200	80

用来寻找表中最小的带符号的 8 位数。

在例 5-4 中，我们假设调用子程序之前，寄存器对 H 内已装有表的始地址，寄存器对 D 内装有节点数。因此，8080 在 SMS 8时把第一个带符号的 8 位数装入 B 寄存器，然后表地址加 1，节点数减 1。如果现在节点数是 0，8080 微处理机便从 SMS 8 子程序返回，此时在 B 寄存器内存有最小的带符号 8 位数。如果节点数不等于 0，则 8080 把第一个节点（在 B 寄存器）传送到 A 寄存器，然后把它与表中的第二个节点进行“异”操作。8080 执行这些指令的目的是确定两个节点（两个数）中是否有一个是负数。如果确有一个是负数（但不是两个），则由于执行 XRAM 指令的结果，A 寄存器的 D_7 位将是逻辑 1。

如果这两个数中有一个是负的(A寄存器的D_7位是逻辑1)，则8080执行JM(负跳转)指令而转到NEG指令。在NEG处，它必须确定这两个数中哪一个是负数。如果B寄存器内的数是负的，8080可以继续向前执行表检查指令。如果存储

例 5-4 从一个表中找最小的带符号(2的补码)8位数

```
/这个子程序将从表中找出最小的2的
/补码(带符号)数。调用它时应把表地址装入寄存器对H
/内，节点数装入寄存器对D内。
/当8080返回时，B寄存器内
/含有最小的带符号8位数。
SMS 8,      MOVBM/将一个节点传送到B寄存器。
NEXT8, INXH   /存储器地址加1。
       DCXD   /节点数减1。
       MOVAD
       ORAE
       RZ     /如节点数为零，则返回。
       MOVAB  /否则取一个节点到A寄存器。
       XRAM   /将它与存储器内的节点相"异"。
       JM     /如符号位为1，则两节点
       NEG    /中有一为负，找出它并
       0      /送B寄存器。
       MOVAB  /两数符号位相同。
       CMPM   /比较两个数。
       JC     /存储器中节点大于B，检查表
       NEXT 8 /内下一项(节点)。
       0
       JMP    /存储器中节点小于B，则将该
       SMS 8  /节点送入B。
       0
```

```
NEG, MVIA     /把 10000000 装入 A 寄存器中。
      200
      ANAM    /A 的内容与存储器相与。
      JNZ     /A=200，存储器内容为负，
      SMS 8   /将它送 B 寄存器。
      0
      JMP     /负数已在 B 寄存器，检查下
      NEXT 8  /一项(节点)。
      0
```

器内的数是负的，那么，必须把这个数传送到B寄存器。所以，在NEG处，8080把200(二进制的10000000，十六进制的80)装入A寄存器，然后把寄存器对H寻址的存储器的内容与它进行"与"操作。如果存储器中的节点(数)是负的，则8080执行JNZ指令而转到SMS 8，这是因为8080执行ANAM指令，使A寄存器的D_7位置为逻辑1的原故。如果存储器的节点是正数，A寄存器的D_7位被清除为逻辑0，8080不执行JNZ指令。相反，由于存储器内的数是正的，8080就转到NEXT 8(因负数已经在B寄存器)。

请读者注意，只有当一个带符号的数是正而另一个带符号的数是负时，8080才执行NEG处的指令。这就是说，负数是两数中较小者，因此，8080应该确保这个数最终存入B寄存器。如果两个数都是正或都是负，则不执行NEG处的指令。

如果两个数都是正或者都是负，8080执行XRAM指令的结果，把A寄存器的D_7位置为逻辑0。因此，8080不执行JM NEG指令，而是把B寄存器中的带符号的数传送到A寄存器，然后，把它与寄存器对H寻址的存储单元的内容进行比较。

如果存储单元的内容比 A 寄存 器(和 B 寄存器)内的节点大，则 8080 跳转到 NEXT 8，从而能 够检查表 中下一个 连续的节点，如果存储单元的内容小于或等于 A 寄存器的内容，则 8080 执行 JMP 而转到 SMS 8，把该节点传送到 B 寄存器。

读者可以看到，寻找带符号的最小 8 位数的子程序，比寻找不带符号的最小 8 位数的子程序(例 5-1) 要稍微复杂一些。8080 检查带符号数表所花的时 间也稍长一些。 在本章的下一节， 我们将要讨论从一个不带符号 的 16 位数的表中找出最小和最大节点以及从一个 16 位的表中找 出最小的 带符号节点的软件。

双精度表(16 位)

从一个表中找出最小的和最大的不带符号的16位数

我们要讨论一个子程序， 它既能从不带 符号的 16 位数表中寻找最小的数，也能找最大的数，而不是只能找最大的数或者只能找最小的数。 例 5-5 所列出的子 程序 能够完成 这个任务。

当调用例 5-5 的子程 序时， 寄存器对 H 内必须 包含了表的始地址，寄存器对 D 必须包含节点数。 当 8080 从该子程序返回时， 最小的不带符号的 16 位数存 在寄存器对 B， 最大的不带符号的 16 位数存在寄存器对 D。

例 5-5 从一个表中找最大的和最小的 不带符号 的 16 位数

```
/这个子程序从表中寻找最小的和最大的
```

```
        /不带符号的 16 位数。调用它时，把表地址装入寄存器对 H，
        /把节点数装入寄存器对 D。
        /当 8080 从该子程序返回时，
        /最小的 16 位数在寄存器对 B 内，
        /最大的数在寄存器对 D 内。
SAL 16, PUSHD       /节点数存入堆栈。
        MOVEM       /从存储器取低位字节到 E 寄存器，
        MOVCM       /把同一字节装入 C 寄存器。
        INXH        /表地址加 1。
        MOVDM       /从存储器取高位字节到 D 寄存器，
        MOVBM       /把同一字节装入 B 寄存器。
NEXT 16, INXH       /地址加 1，指向下一个低位字节。
        XTHL        /表地址入栈、节点数入
        DCXH        /寄存器对 H，
        MOVAH       /节点数减 1。
        ORAL
        XTHL        /表地址又重入寄存器对 H。
        JNZ         /节点数不等于零，
        NOTYET      /检查表中的另一个数。
        0
        POPH        /从堆栈弹出节点数，
        RET         /然后返回。
NOTYET, MOVAE       /取"大的"低位字节到 A。
        SUBM        /减去存储器中的低位字节。
        INXH        /存储地址加 1。
        MOVAD       /取"大的"高位字节到 A。
        SBBM        /减去存储器中的高位字节。
        JC          /存储器的内容大于 D 和 H，所以
        LARGE       /把存储器中的 16 位数传送
        0           /到寄存器对 D。
```

```
        DCXH      /地址减 1，指向低位字节。
        MOVAM     /从存储器取一个低位字节。
        SUBC      /减去"小的"低位字节。
        INXH      /地址加 1，指向高位字节。
        MOVAM     /高位字节送到 A 寄存器。
        SBBB      /减去"小的"高位字节。
        JNC       /存储器的内容大于或等于 B 和 C，
        NEXT 16   /则仅使节点数减 1，如果需
        0         /要，检查下一个节点。
SMALL,  DCXH      /存储器地址减 1。
        MOVCM     /从存储器取低位字节到 C。
        INXH      /表地址加 1。
        MOVBM     /取高位字节到 B 寄存器。
        JMP       /然后判断整个表是否
        NEXT 16   /已检查完毕。
        0
LARGE,  DCXH      /表地址减 1。
        MOVEM      低位字节送到 E 寄存器。
        INXH      /表地址加 1。
        MOVDM     /高位字节送到 D 寄存器。
        JMP       /然后判断是否整个表
        NEXT 16   /已经检查完毕。
        0
```

当 8080 执行 SAL 16 子程序时，它首先把节点数(在寄存器对 D 内)存入堆栈，然后把第一个节点装入寄存器对 B 和 D。即把它的低位字节装入寄存器 E 和 C，高位字节装入寄存器 D 和 B。在 NEXT 16，8080 使寄存器对 H 内的地址加 1，从而指向下一个节点的低位字节。然后，8080 交换寄存器对 H内的地址和存在堆栈内的节点数。节点数减 1 以后，8080 把节点

数的高位字节传送到 A 寄存器，然后把它与节点数的低位字节相“或”。ORAL 指令使标识位置位或者清零。然后，把节点数放回堆栈，表地址送回到寄存器对 H。因为指令 XTHL 不影响任何标识位，所以，如果节点数等于零，则 8080 将节点数从堆栈弹入寄存器对 H，然后从子程序返回。返回时，最小的不带符号的 16 位数存在寄存器对 B，最大的不带符号的 16 位数存在寄存器对 H。

如果节点数不等于零，8080 从目前找到的“最大的”节点(在寄存器对 D 内)减去存储器内节点。因为表包含着不带符号的 16 位数，为了完成 16 位的“比较”，必须进行两次 8 位减操作。如果存储器内的节点比寄存器对 D 内的节点大，则8080 执行 JC-LARGE 指令，把存储器内的节点装入寄存器对 D。在 LARGE 处的指令就干这事。8080 执行了这些指令之后，就转移到 NEXT 16，把存贮器内的下一个 16 位节点与目前找到的“最大的”节点进行比较，如果需要的话，还与已经找到的“最小的”节点进行比较。

如果存储器内的节点小于或等于寄存器对 D 内的节点，则 8080 不执行 JC-LARGE 指令，而是从存储器中的节点减去至今所找到的“最小的”节点(在寄存器对 B 内)。如果存储器中的节点等于或大于寄存器对 B 内的节点，则 8080 执行 JNC-NEXT 16，可以检查下一个 16 位数。如果存储器内的节点小于寄存器对 B 内的节点，则 8080 不执行 JNC-NEXT 16 指令，而是把无符号的 16 位数从存储器传送到寄存器对 B，然后检查表里的下一个数。

从一个表中找最小的带符号(2 补码)的 16 位节点

为了找到一个表中最小的带符号的 16 位数，我们必须执

行寻找表中最小带符号 8 位数的所用的许多操作。但是，8080 微处理器不能进行 16 位数的比较，因而必须用减法运算来完成这种比较。在表中寻找最小的带符的(2 的补码)16 位数的子程序如例 5-6 所示。

例 5-6 从表中找最小的带符号的(2 的补码)16 位数的子程序

```
/这个子程序从表中找最小的带符号
/(2 的补码)16 位数。调用它时，把表地址装入寄存器对 H，
/把节点数装入寄存器对 D。
/当 8080 从该子程序返回时，最小值将存储在
/寄存器对 B 中。
SMS 16,  MOVCM /从存储器取低位字节到 C 寄存器。
         INXH  /表地址加 1。
         MOVBM /从存储器取高位字节（和符号位）到
NEXT 16, INXH  /B 寄存器。表地址加 1,指到下
         DCXD  /一个节点的低位字节。D 寄存器
         MOVAD /中的节点数减 1。
         ORAE
         RZ    /节点数等于 0，返回。
         INXH  /表地址加 1，指到高位字节。
         MOVAB /从 B 寄存器取另一个高位字节到
         XRAM  /A 寄存器。存储器中的高位字节
         JM    /和 A 的高位字节“异或”。如果符号
         NEG 16 /位(D7)是 1，则其中有一个数为负，
         0     /所以找到它，并把它送入寄存器对 B。
SAMES,   DCXH  /两个数符号相同。
         MOVAC /从 C 寄存器取一个低位字节到 A
         SUBM  /寄存器。减去另一个低位字节。
         INXH  /表地址加 1。
```

```
        MOVAB  /取一个高位字节。
        SBBM   /减去另一个低位字节。
        JC     /(存储器)内容大于寄存器对B,
        NEXT16,/则检查表内
        0      /下一个节点。
        DCXH   /寄存器对 H 表地址减 1，指向低位字节
        JMP    /(存储器)内容小于寄存器对 B，则
        SMS 16 /把存储器的内容送入
        0      /寄存器对 B。
NEG 16, MVIA   /把 10000000
        200/装入 A 寄存器。
        ANAM   /A 的内容与存储器的内容相"与"。
        JZ     /存储器中的数是正的，因而较小
        NEXT 16 /的数（负数）已在寄存器对 B 内。
        0
        DCXH   /使这个地址减 1，指到最低有效字节。
        JMP    /负数存在存储器，所以
        SMS 16 /把它传送到寄存器对 B。
        0
```

当 8080 要调用 SMS 16 子程序时，表的始地址和节点数应已分别装在寄存器对H和D。然后，8080 把第一个带符号的 16 位数装入寄存器对 B，并把寄存器对D内的节点数减 1。如果节点数变为 0，则 8080 从子程序返回，同时在寄存器对 B 内装有最小的带符号的 16 位数。即 B 寄存器装这个数的高位字节，而C寄存器装它的高位字节。

如果节点数减 1 后不为 0，则 8080 将寄存器对H内的表地址加 1，使它指向表中第二个节点的高位字节。然后，把寄存器对 B 内的数的（带符号的）高位字节传送到A寄存器，在这

其它与存储器中的数的（带符号的）高位字节进行“异或”操作。如果两数中有一个是负的，则XRAM指令将A寄存器的D_7位置为逻辑1。如果A寄存器的D_7位是逻辑1，则8080执行JM-NEG 16指令。

当8080转到NEG 16时，它必须确定两个数中哪一个是负的。因为负数是两个数中较小的一个，故8080还必须保证把这个数存入寄存器对B。因此，在NEG 16处，二进制数10000000（八进制200，十六进制80）被装入A寄存器，并和存储器中的数的（带符号的）高位字节相“与”。如果存储器中的数是正数，则8080执行ANAM指令的结果是0，所以，8080执行JZ-NEXT 16指令。这就是说，负数已经在寄存器对B内。8080可以检查表中下一个数。如果8080不执行JZ-NEXT 16指令，则负数是在存储里，因此，8080转到SMS 16，把存贮器中负数移到寄存器对B。

如果两个数同为正数或者负数，则8080在执行XRAM指令之后，将执行不同的指令序列。它越过JM-NEG 16指令，而执行它下面的指令，从寄存器对B内的数减去存储器内的数。这一16位“比较”的结果，如果进位位被置为逻辑1，则B寄存器对的内容小于存储器的内容。因此，8080执行JC-NEXT 16，检查表中的下一个数。如果存储器中的数小于或者等于寄存器对B的数，8080执行JMP指令，转到SMS 16，把存储器内的数传送到寄存器对B。

我们在这一章内没有编写能够确定表中最小的和最大的带符号16位数的子程序。这个任务留给读者自己去做。根据前面这些例子，这个子程序应该是不难编写的。

例 5-7 从一个表中找最大和最小的不带符号的24位数

```
/这个子程序将从一个表中找出
```

```
        /最小和最大的不带符号的24位数。在调用此
        /子程序时，寄存器对H内装有表地址，寄存
        /器对D内装有节点数。当8080从此
        /子程序返回时，最大和最小
        /的数存储在读/写存储器里。
SLU24,  PUSHD   /把节点数保存在栈里。
        LXID    /把要存放最小的24位数的读/写
        SMALL   /存储器单元地址装入寄存器对D。
        0
        LXIB    /把要存放最大的24位数的读/写
        LARGE   /存储单元地址装入寄存器对B。
        0
        MVIA    /每个数所用的存储器单元
        003     /的数目装入A寄存器。
INITIL  PUSHPSW /把这个数保存在堆栈。
        MOVAM   /从存储器取表的一个8位字节到
        STAXD   /A寄存器。把它保存在"SMALL"内。
        STAXB   /把它保存在"LARGE"内。
        INXH    /表地址加1。
        INXD    /"SMALL"地址加1。
        INXB    /"LARGE"地址加1。
        POPPSW  /从堆栈弹出存储器单元数。
        DCRA    /把这个数减1。
        JNZ     /如果这个数不等于0，则必须
        INITIL  /从表中将另一个8位字节传送到
        0       /"SMALL"和"LARGE"。
        MVIC    /把每个数的字节数
        003     /装入C寄存器。
TEST24, XTHL    /把字节数取到寄存器对H。
        DCXH    /字节数减1。
```

```
        MOVAH
        ORAL
        XTHL    /把字节数放回堆栈。
        JNZ     /字节数不等于 0，
        TESTIT  /表中还有另一个数待检查。
        0
        POPH    /字节数等于 0，把它从堆栈取出。
        RET     /返回。
TESTIT, LXID    /H 和 L 寄存器指向表的第二个数，
        SMALL   /因此，把"SMALL"的地址
        0       /装入寄存器对 D。
        CALL    /然后，从"SMALL"的内容减去
        SVB 3   /表中的 24 位数。
        0
        JNC     /存储器的内容比"SMALL"
        MOVE    /的内容小，则
        0       /把表中的这个数传送到"SMALL"。这个
        LXID    /数比"SMALL"大。
        LARGE   /它比"LARGE"大吗？
        0
        CALL
        SUB 3
        0
        JNC     /不，它不比"LARGE"大
        NOTLRG  /因此，找表中的下一个数，
        0       /并使节点数减 1。
MOVE,   MOVBC   /把计数值从 C 寄存器送到 B 寄存
MOVE 1, MOVAM   /器。从存储器中取表的一个字节到
        STAXD   /A。把它存入"LARGE"或"SMALL"。
        INXH    /表地址加 1。
```

```
        INXD    /存储器的地址加 1。
        DCRB    /字节数减 1。
        JNZ     /如果字节数不等于 0，
        MOVE 1  /则必须从存储器取表的另一字节。
        0
        JMP     /三个字节都传送完，检查
        TEST 24 /表的节点数是否也
        0       /等于 0 。
NOTLRG, INXH    /数比"SMALL"大。
        INXH    /但是比"LARGE"小，则
        INXH    /找表的下一个数。
        JMP     /然后，检查 8080 是否已到达
        TEST 24 /表的终点。
        0
SUB 3,  MOVBC   /字节数送 B。
        PUSHH   /保存表中数的地址。
        PUSHD   /保存"SMALL"或"LARGE"的地址。
        XRAA    /把 A 和进位置 0。
SUB 3 A, LDAXD  /从"SMALL"或"LARGE"取一个字节。
        SBBM    /减去表中的一个字节。
        INXH    /表地址加 1。
        INXD    /"SMALL"或"LARGE"的地址加 1，
        DCRB    /字节数减 1。
        JNZ     /如果三个字节还没有都被减去。
        SUB 3 A /则减另一个字节。
        0
        POPD    /三个字节都已被减去，
        POPH    /则从堆栈取出原来的地址。
        RET     /返回。
SMALL,  0       /这儿用来存储 24 位最小
```

```
        0           /的数。
        0
LARGE,  0           /这儿用来存储 24 位最大
        0           /的数。
        0
```

三精度表(24 位)

我们将在本章这一节讨论的唯一例子是从一个不带符号的 24 位数的表中找最小和最大的数。例 5-7 列出了这个子程序。因为要用六个寄存器才能存放最小的和最大的不带符号的 24 位数，所以，这个子程序把这两个数存储在读/写存储器。当调用这个子程序时，表的始地址应在寄存器对H中，节点数应在寄存器对D中。

SLU 24 子程序的第一条指令把节点数保存在堆栈里。然后子程序把用来存储最小数的低位字节的地址装入寄存器对D，把用来存储最大数的低位字节的地址装入寄存器对B。然后，把 003 (03) 装入A寄存器，因为存放表中的一个数要用三个存储单元。在 INITIL 处的指令将这个数保存在堆栈里。然后，8080 把表中第一个数的低位字节传送到A寄存器，再存入寄存器对D和B所寻址的存储单元。接着，寄存器对H、D和B内的存储器地址加 1，存储单元数从堆栈弹入A寄存器。然后这个值减 1。如果其结果不等于 0，则 8080 转到INITIL，把减 1 后的存储单元数存入堆栈。其后，8080 往下执行，把表里另外两个 8 位字节送到留着存最小的和最大的不带符号的 24 位数的存储单元。简言之，这些指令把表内第一个数作为

初值送到存储器单元“SMALL”和“LARGE”。

一旦三个字节送入以后，即把 003（03）装入C寄存器。它是一个 24 位数的 8 位字节的个数。在 TEST 24，8080 将节点数减 1，并进行测试。如果节点数不等于 0，则 8080 从 TESTIT 开始执行指令。如果节点数减 1 后等于 0 时，则 8080 从子程序返回，此时最小的不带符号的 24 位数在“SMALL”，而最大的不带符号的 24 位数在“LARGE”。

在 TESTIT 处，8080 把最小数的低位字节的地址装入寄存器对D，然后调用SUB 3 子程序。这个子程序从寄存器对D寻址的三个相邻存储单元的内容，减去寄存器对H寻址的存储单元开始的三个相邻存储单元的内容。当 8080 从该子程序返回时，标识位将反映这一减法操作的结果。请注意，当调用这个子程序时，寄存器对H指向表内第一个数的低位字节。**当 8080 从该子程序返回时，寄存器对 D 和寄存器对 H 内包含的地址与调用这个子程序时它们内部所包含的地址相同。**看起来就好象“SUB 3 ”子程序并没有改变寄存器对D和H所包含的地址。

当 8080 从这个子程序返回时，如果表中的 24 位数比“SMALL”中所存储的数小或者相等，则 8080 将执行 JNC MOVE 指令。MOVE 子程序只是把寄存器对H寻址的存储单元中的三个字节传送到寄存器对D所寻址的三个存储单元。传送完以后，8080 转到 TEST 24，将节点数减 1 并进行测试。如果表中的 24 位数大于“SMALL”内的 24 位数，则 8080 把“LARGE”的地址装入寄存器对D，然后调用 SUB 3 子程序。请记住，因为子程序 SUB 3 不改变寄存器对D或寄存器对H内的地址，所以，寄存器对H仍含有表内第二个数低位字节的地址。

当 8080 这一次从 suB 3 子程序返回时，如果表中的 24 位数小于或不等于 LARGE 内的内容，则执行 JNC 指令后转到 NOTLARG，将寄存器对 H 内存储地址加 1 三次，使之指向表内下一个 24 位数(节点)。8080 然后转到 TEST 24。将节点数减 1，并进行测试。如果表内的 24 位数大于“LARGE”的内容，则 8080 不执行 JNC—NOTLRG 指令，而是从 MOVE 开始执行指令。这些指令把表内的 24 位数传送到“LARGE”所使用的存储单元。

检索子程序的共同特征

本章至此所讨论过的各个子程序具有许多共同的特征。所有**这些子程序都要求把表的始地址装入寄存器对 H，把节点数装入寄存器对 D**。这就是说，如果读者在一个程序中使用 8 位的检索子程序，而且随后还必须把这个程序提高到检索 16 位或 24 位的数，那么，这些子程序的始地址和节点数的要求都相同。

在处理大量数据以后，程序也许会产生若干有待检索的表。有些时候，程序可能产生一个没有任何节点(项)的表。如果发生这种情况，并且调用一个检索子程序，则 8080 要比较 65536 个节点的内容。原因很明显，节点数从 000000(0000)减 1 后变为 377 377(FFFF)，故 8080 从子程序返回之前，必须对 65,536 个节点进行比较。

如果这些子程序偶尔被用来检索只包含一个节点（一个数或一项）的表，那么，读者在解释检索结果时还必须谨慎。这一点在子程序要检索最小节点和最大节点时，特别重要。如果

表内只包含一个节点，那么，在表中找到的最小节点和最大节点具有相等的值。这对于数据处理程序来说有时会产生问题。

正如读者所预料的那样，寄存器对H或寄存器对D内装的数不合适也会出现问题。假设读者不知道节点数，但是知道表的始地址和末地址，那么，还能使用上述的检索子程序吗？是的，这些子程序仍可以用来寻找最小的节点，如果合适的话，还可以找最大的节点。但是，必须给这些子程序编写新的“前端”。例 5-8 列出“前端”的一个典型例子。这些指令不过是从末地址减去始地址，然后使其差值加 1。所得结果存入寄存器对 D，而寄存器对H的始地址仍保持不变。这里假定始地址比末地址小。

例 5-8 由表的始地址和末地址计算节点数

```
        /检索子程序经这样修改之后可计算
        /表的始地址和末地址之差。
ADRCAL, MOVAE    /从E取末地址的低位字节到 A。
        SUBL     /减去始地址的低位字节。
        MOVEA    /相减结果低位字节送入E寄存器。
        MOVAD    /以 D 取末地址的高位字节到 A 寄
        SBBH     /存器。减去始地址的高位字节。
        MOVDA    /结果的高位字节存入 D 寄存器。
        INXD     /差值加 1。
SMU 8,  MOVBM    /从存储器取一个数到 B 寄存器。
NEXTU 8, INXH    /表地址加 1。
        DCXD     /节点数减 1。
        MOVAD
          .
          .
```

注意这些指令是怎样加到 SMU 8 子程序(例 5-1)上的。这

个指令序列只能用在检索单精度(8 位)表的子程序的前面。为什么？因为在单精度表中，每一个节点（数）存在一个存储单元。对于双精度（16 位)表，需要用两个存储单元来存储一个节点。因此，节点的数目等于表所用的存储单元的一半。对于三倍精度（24 位）的表，节点的数目等于表所用的存储单元数目的三分之一。这就是说，如果给检查双精度(16 位)表的检索子程序编写一个新的前端，那么在计算出差值和加 1 以后，还必须用 2 去除寄存器对D的内容。

怎样才能容易地实现寄存器对D的内容除以 2 呢？只要用一条 RAR 指令将D寄存器的内容向右循环移位，然后用一条 RAR 指令，把E寄存器的内容向右循环移位。**要使这种"除以 2"运算可行，D 寄存器内容的循环右移必须先于 E 寄存器。**

有些读者也许会抱怨我们连篇累牍刊载了许多程序设计员不用的程序和子程序，例如例 5-7 所列出的 24 位子程序。但是，我们并不这样认为。我们的目的是给读者和程序设计员提供尽可能多的程序设计例子。即使这些程序和子程序不被使用，但仍有许多有用的方法、思想和说明，最终可为读者在开发自己的程序或者修改别人的程序时提供借鉴。

ASCII 字符串的检索

不仅需要查找表中的数字信息，而且还可能要求从一个表中找一个特定的 ASCII 字符串(关于字符串请参考第四章的数据结构)。检索 ASCII 字符串的程序或子程序的最常见的用途之一是管理发送文件清单。有许多公司出售和出租邮政发送清单，它们利用计算机按地区，州或 ZIP 代码把名字区别开来。

让我们假设读者有一份发送文件清单，它由六个名字和六个地址组成。读者要求 8080 从这个清单中找出所有具有相同邮政编码的名字和它们的地址。在转送清单上所列的名字如下：

1. Apex.Inc
   20 Main Street
   Triangle, VA 24061
2. XYZ Blasting
   22 North Main
   Boom Boom, VA 24060
3. Easy Fire Insuramce
   2280 W. Alpine
   Redhot, VA 24061
4. ABC Pencil Co.
   Arques, Are
   Jeadville VA 24060
5. Clear Plastics, Inc.
   23 Hardy Ave
   Cloudy, VA 24060
6. Ma's Fast Foods
   2105 NE West st.
   Speedy VA 24061

注：编号 1～6 并不存在表内，它们只供我们参考。ZIP—美国邮政分区代码。

读者从上面可以看到，这些地址的邮政分区代码不是 24060，就是 24061。如同多精度表检索程序那样，怎样以 ASCII 字符形式将这些名字和地址存入存储器，也必须事先有某些约定。最简单的做法是把所有的 ASCII 值按顺序存入存储单元。从键盘随时输入的字符都存入存储器，包括回车符。一个地址的邮政号码结束后就存储下一个名字的第一个字符。这种情况如下所示：

名字
•
•
•
A＝101
V＝126
E＝105
•＝056
CR＝015

C＝103
L ＝114
O＝117
U＝125
D＝104
Y＝131
, ＝054
SP＝040
V＝126
A＝101
SP＝040
2＝062
4＝064
0＝060
6＝066
0＝060
CR＝015
M＝115
A＝101
, ＝047
S＝123
•
•
•

CR 二回车符；SP 二空格

在“Ma′s Fast Foods”的邮政编码之后，将存入 003(03)，作为表的结束符。它用来告诉 8080 已经到达了表的末端。003(03)这个数不是一个打印的 ASCII 字符，所以很适合做结束符。之所以需要加入结束符是因为除了对输入到 8080 微型计

算机的字符一一计数之外，没有容易的办法确定包含名字和地址的表有多长。在本章前面的各个例子中，表的节点数是已知的。但是，在现在这种情形里，名字和地址常有增减，要记录该表实际所用的存储单元是不容易的。此刻，我们暂不考虑是怎样把名字和地址送入微型计算机的。而假设名字和地址表已按上述格式存在(微型计算机)存储器里。

一旦六个名字和地址被存入了存储器，读者必须决定要8080检索两个邮政编码(24060和24061)中的哪一个。如果要求8080找出所有地址中的24060，则可以使用例5-9所列的程序。

这个子程序把表的始地址装入寄存器对D。这样，寄存器对D就指向存有“Apex Inc”的第一个字符A的ASCII值的那个存储单元。然后把存储ASCII测试邮政编码（在此例中为24060)的第一位数字的存储器地址装入寄存器对H。这个邮政编码是由五个ASCII字符形成的，因为地址中的邮政编码都是以ASCII字符的形式存在表中。在CHECK处，表的第一个字符传送到A寄存器，然后与数0 03(03)进行比较。请记住，003(03)只是一个任选的非打印字符，用来指示8080何时到达表的末端。003(03)存在存储器中，紧挨着“Ma's Fast Foods”的邮政代码之后。如果8080从存储器读出003(03)，则说明它已经查完整个表，因而返回到称为START的程序。

如果从存储器读出的值不是003(03)，则与测试邮政编码的第一个数值即ASCII 2(8进制062，16进制32)进行比较。如果这两个字符在数值上相等，则8080执行J2—OK 1；如果不等，则8080使表的存储地址加1，准备将表内的下一个字符与0 03(03)进行比较，如果需要，与测试邮政代码的第一个字符进行比较。这意味着，表内的第二个字符也与测试邮政编码的第一个字符相比较。

8080什么时候首次执行J 2—OK 1指令呢？请参考列在本节开头的名字和地址表。当地址“20 Main Street”中的第一个字2首先与邮政编码(24060)中的2进行比较时，8080第一次转到OK 1，开始将名字和地址表中后面四个顺序字符与测试邮政编码中后面四个ASCII字符进行比较。因此，8080在OK 1处把004(04)装入B寄存器。然后在OK 2使寄存器对H和寄存器对D内的存储地址加1，再把“20 Main Street”中的第二个字符0从表装入A寄存器。然后把这个值与测试邮政编码24060中的ASCII字符4进行比较。因为这两个被比较的ASCII值不相等，所以，8080转到AGAIN，把测试邮政代码所用的第一个存储单元地址装入寄存器对H。寄存器对D内的地址不变，仍指向表内的一个字符。

当8080在第一个地址中遇到邮政编码24061的2时，8080第二次执行OK 1及其后面的指令。在OK 1，8080把应该在两个邮政编码之间进行的其余比较次数装入寄存器对B，在OK 2，8080继续比较邮政编码的ASCII字符。如人们预期的那样，8080把两个邮政编码中的ASCII 4进行比较。因为相同，8080不执行JNZ—AGAIN指令，而是使B寄存器中的计数数减1，即从004减至003(04—03)。减的结果不等于0，所以8080返回到OK 2，比较下面两个字符。这个循环一直下去，直到：(1) 测试邮政编码和名字/地址表的一个字符之间出现不一致或 (2) B寄存器内的计数数减为零。在这个例子中，表中的2406和测试邮政编码的2406比较，结果是一致的。但是名字和地址表中的ASCII 1不等于测试邮政编码的ASCII 0，所以，最后执行JNZ—AGAIN指令。

例 5-9 从名字和地址表找邮政编码24060

```
/这个子程序检索存在表内的
```

```
    /地址中的五位邮政编码
START, LXID/寄存器对 D 存储
       LIST/表的始地址。
       0
AGAIN, LXIH   /寄存器对H存储
       ZIPCOD/测试邮政编码
       0      /的第一位数字的地址。
CHECK, LDAXD/取表的一个字符。
       CPI
       003    /已经到表的末端了吗?
       RZ     /是，返回主程序。
       CMPM   /否，把它与测试邮政编码的第一
       JZ     /位数字比较。两者一致，则
       OK 1   /比较其它四位是否一致。
       0
       INXD   /不一致，使表地址加 1。
       JMP    /然后把它与测试邮政
       CHECK  /编码的第一位数字比较。
       0
OK 1,  MVIB   /把还必须做的比较次
       004    /数装入 B 寄存器。
OK 2   INXH   /测试邮政编码的表地址加 1。
       INXD   /名字/地址表地址加 1。
       LDAXD/取下一个地址字符。
       CMPM   /把它与测试邮政分区代码比较。
       JNZ    /不一致，则给邮政编码地址
       AGAIN /(但不是表地址)重新设初值，并再试。
       0
       DCRB   /出现一次一致，B 寄存器中的
       JNZ    /计数减 1。已五次一致吗?
```

```
        OK 2    /没有，试进行另一次比较。
        0
        JMP     /是，找到一个 24060
        AGAIN   /再试名字和地址表中
        0       /其余的项。
ZIPCOD, 062     /这里是测试邮政编码 24060
        064     /ASCII 4
        060     /ASCII 0
        066     /ASCII 6
        060     /ASCII 0
```

什么时候B寄存器的内容才最后减为0呢？在“XYZ Blasting”的地址中找到邮政代码24060时，这时B寄存器第一次减为0。但是，即使8080成功地比较了两个五位邮政代码（即比较结果一致），8080也要返回到AGAIN。

当测试邮政编码和这个表的一个地址出现一致时，这个程序打印表的名字和地址吗？回答是否定的。在START子程序里，没有包括打印指令或子程序。

8080可以不返回到AGAIN，而是执行一串指令，以便将包含测试邮政编码的名字和地址都打印出来。但是，如果需要把这些名字和地址列表，就必须设法先告诉8080，一个地址在何处结束和另一个地址在何处开始。对于存在名字/地址表中的名字和地址来说，邮政编码的ASCII字符是与具体名字有关的最后几个字符；它们的后面就开始另一个名字。这就是说，名字和地址规定按这样形式存储，位于前一个邮政编码之后和另一个名字前。然而，假设下面的名字和地址加到名字/地址表上：

ABC Pencil CO.

Arques Ave
Leadville, VA 24060
ATTN:Erasure Dept.

在这个地址中，邮政编码并不包含在这个地址的最后五个ASCII字符里，相反，ASCII字符串“Dept”位于这个地址的末尾。

当然，如果必须把这样的名字和地址加到名字/地址表上，那么，打字员在打入这些名字和地址时不得不改变顺序，让邮政编码总是在地址的最后面。但是，这可能严重打乱地址。如果不这样做，可以把一个朝上的箭头(↑)存在地址的最末一个字符之后，不论这是个什么字符。几乎任何一个字符都可用作名字和地址结束符或界符，但是，不能使用可能在名字或地址中出现的字符。可以使用ⓐ、$、¢或☆等字符；不要使用#，:，(，)或/这类符号，因为这些符号可以用于名字或地址。选定了名字的结束符后，就应该把它加到名字/地址表上，置于地址的最后一个字符之后。现在，名字/地址表成了下面这个样子：

1. APex Inc.
   20 Main Street
   Triangle, VA 24061↑
2. XYZ Blasting
   22 North Main
   Boom Boom, VA 24060↑
3. Easy Fire Insurance
   2280 W. Alpine
   Redhot, VA 24061↑
4. ABC Pencil Co.
   Arques Ave.
   Leadville, VA 24060↑
5. Clear Plastics, Inc.
   23 Hardy Ave.
   Cloudy, VA 24060↑
6. Ma's Fast Foods
   2105 NE West st.
   Speedy, VA 24061↑

注：编号1—6并不存入表中，仅供参考用。

虽然前面的检索子程序不加朝上箭头(↑)也能正确运行，但是，子程序中在电传打字机或 CRT 上打印或显示名字或地址的那个程序段将使用这个符号。

例 5-9 的检索程序仍可用来检索名字/地址表中的邮政分区代码。但是应该用 JMP-BACKUP 取代 JMP-AGAIN。BACKUP(打印机)指令序列示于例 5-10。

例 5-10 邮政编码检索子程序的打印机指令

```
                    /这些指令加到前述子程序上后，让
                    /8080 使表地址倒退，直到找到前一个
                    /名字末为止。
                    /然后，打印名字、地址和
                    /邮政分区代码。
                      .
                      .
                      .
BACKUP  DCXD        /比较结果，五位数字一致。
        LDAXD       /使表地址倒退，
        CPI         /到前一个名字末。
        136
        JNZ         /在前一个名字的末了找到
        BACKUP      /↑了吗？没有，继续倒退。
        0
PRINT,  INXD        /找到了↑，使地址增 1。
        LDAXD       /从表取一个字符。
        CPI         /它是回车符吗？
        015
        JNZ         /不是，判断它是不是朝上箭头↑
        NPCRLF      /的 ASCII 代码。
        0
```

```
        CALL    /是回车符，则
        CRLF    /打印回车和换
        0       /行符。
        JMP     /然后，从表中取下一个字符。
        PRINT   /并且判明
        0       /它是什么字符。
NPCRLF, CPI     /它是名字末尾的结束符↑吗？
        B6
        JZ      /是。不再打印
        PCRLF   /表中其它字符。打印一个
        0       /CR 和 LF 字符，然后继续检索。
        CALL    /这个字符不是↑，
        TTYOUT  /则打印它。
        0
        JMP     /现在，取另一个字符并对它检查。
        PRINT
        0
PCRLF,  CALL    /找到了↑，在名字和
        CRLF    /地址后打印回车符。
        0
        JMP     /然后检索表的其余部分，
        AGAIN   /找邮政编码 24060。
        0
CRLF,   MVIA    /把回车符的 ASCII 码
        015     /装入A寄存器。
        CALL    /然后，将它在电传打字机上打印
        TTYOUT  /或在 CRJ 上显示。
        0
        MVIA    /然后，把换行的 ASCII 码
        012     /装入A寄存器。
```

```
JMP       /把它打印在电传打字机上
TTYOUT    /或显示在CRT上。
0
```

8080测试邮政编码和表内一个地址间的五个字符的比较结束后，便执行BACKUP处的指令。在BACKUP处，8080寻找前一个地址末了的箭头(↑)。DCXD指令使表地址减1，用LDAXD指令把表的一个字符读入A寄存器。CPI 136指令将朝向箭头字符的ASCII值与从表读出的值进行比较。如果从存储器读出的字符不是朝上的箭头(↑)，则8080执行JNZ-BACKUP指令，检查下一个较低地址上的存储单元的内容。

找到了前一个地址末了的朝上箭头(↑)以后，8080将不再执行JNZ-BACKUP指令，而是执行PRINT处的INXD指令。现在，寄存器对D指向其邮政编码与测试邮政编码相一致的名字和地址的第一个字符。然后，8080把名字的第一个字符从存储器读入到A寄存器，并且与回车符的ASCII值进行比较。如果从存储器读入了回车符的ASCII值，则不执行JNZ-NPCRLF指令，而且调用CRLF子程序，在电传打字机上打印或在CRT上显示一个回车符和一个换行符。然后，8080返回到PRINT，从存储器读出回车符后的字符。

如果8080没有从存储器读出回车符的ASCII值，那么，它转到NPCRLF，把从存储器读出的ASCII值与朝上箭头(↑)的ASCII值进行比较。如果8080从存储器读出了箭头字符，则它转到PCRLF。在这个程序段上，8080打印一个回车符和一个换行符，然后转到AGAIN。这就是说，包含有指定邮政编码的整个名字和地址已打印完，因为刚刚从存储器读出了位于地址末尾的箭头(↑)。于是，8080打印一个回车符和一个换行符，然后继续检索指定的邮政编码。寄存器对D指向

下一个名字的第一个字符；当 8080 转到 AGAIN 时，测试邮政编码第一个字符的地址再度被装入寄存器对 H。

如果从存储器读出的内容既不是回车符，也不是箭头，则 8080 调用 TTYOUT 子程序，在电传打字机或在 CRT 上打印显示字符。当转移到 PRINT 时，8080 可从表中读取下一个字符，然后把它与回车符和箭头符的 ASCII 值进行比较，并进行相应处理。

这个子程序的"打印机"程序段正确运行，为何需要例 5-11 所列指令？当 8080 从存储器读出回车符的 ASCII 值时，这些指令使打印机或 CRT 打印或显示一个回车符和一个换行符。读者从名字/地址表的结构可看到，表内只存有回车符的 ASCII 值，没有换行的 ASCII 值。因此，从存储器读出回车符时，必须同时打印回车符和换行符。表中不存换行符的 ASCII 值的一个原因是，如果表中包含很多的名字，这样做就可以节省少量的存储单元。

例 5-11 打印回车符和换行符的指令

```
      .
      .
LDAXD  /从表中取一个字符。
CPI    /该字符是换行符吗？
015
JNZ    /不是，判别它是不是
NPCRLF /箭头(↑)的ASCII码。
0
CALL   /它是回车符，所以
CRLF   /打印一个回车符
0      /和换行。
JMP    /然后从表中取
```

```
PRINT   /下一个字符，并且判明
0       -/它是什么字符。
  •
  •
```

读者在邮政编码的检索子程序中找出任何错误了吗？问题出在表中的第一个名字。如果第一个名字和地址含有要找的那个邮政编码，那将会得到什么结果呢？子程序的检索部分会正确运行。但是，当 8080 执行 BACKUP 处的指令时，它决不能在上一个地址末尾找到朝上箭头(↑)，因为它前面并不存在任何地址。这个问题有两种解决方法。第一种方法是让打字员在输入任何名字和地址之前，向微型计算机打入一个箭头符号(↑)。当然，有时打字员可能忘记打。第二种方法是编写一段程序，在输入任何名字和地址之前，自动地把箭头符存入存储器。例 5-12 示出了这两种方法。

在例 5-12 的A部分中，打字员必须把箭头作为名字/地址表的第一个字符送入微型计算机。在例 5-12 的B 部分，箭头符自动存入名字/地址表所使用的第一个存储器单元。请注意，为了把箭头符自动地存在表的开头。只要增加三个存储器单元。显然，付出的代价是很小的。

例 5-12 存储朝上箭头(↑)的两种不同方法的比较

(A)让打字员存储箭头符号

```
/在这个程序中，打字员必须把箭头符号
/作为第一个字符存入表中。
ENTER,   LXISP   /设置堆栈指示器
         350
         020     /(等于十六进制10E8)。
         LXIH    /寄存器对H指向
```

```
        LIST    /存储区的起始单元。
        0
NXTCHR, CALL    /从电传打字机取得一个字符。
        TTYIN
        0
        MOVMA /把这个字符存入寄存器。
        DNXH    /表地址加1。
        CPI     /打入回车符了吗?
        015
        .
        .
```

（B）微型计算机自动存箭头符

```
    /在这个程序中，箭头符作为表的第一
    /个字符，自动地存入存储器。
ENTER,  LXISP   /设置堆栈指示器
        350
        020     /(等于十六进制10E8)。
        LXIH    /寄存器对H指向
        LIST    /存储区的始地址。
        0
        MVIM    /把一个箭头(↑)存储在
        136     /表的头一个单元。
        INXH    /表地址加1。
NXTCHR, CALL    /从电传打字机取得一个字符。
        TTYIN
        0
        MOVMA /把这个字符存入存储器。
        INXH    /表地址加1。
        CPI     /打入回车符了吗?
```

```
015
.
.
```

既然我们已经解决了在表的开头存箭头符的问题，检索子程序已能正确运行，下面我们就来完成一项更大的任务：编写一个能让我们将名字和地址存入表内的指令序列。这个指令序列如例 5-13 所示。这些指令把一个名字和地址表送入微型计算机，而且在整个表被输入之后，微型计算机将从表中找指定的邮政编码。如果找到。8080将名字和地址打印在电传打字机上，或者显示在 CRT上。8080查完整个表后，便停止操作。

例 5-13 用来输入和存储名字和地址表的程序

```
/这个程序可用来输入名字，
/地址和邮政编码。
/然后，可从表中找出任何指定的邮政
/编码。测试邮政编码必须存入存储器里，
/其符号地址为“ZIPCOD”。
ENTER,   LXISP   /设置使堆栈指示器
         350
         020     /(等于十六进制的10E8)。
         LXIH    /寄存器对H指向
         LIST    /存储区的起始单元
         0
         MVIM    /把一个箭头符(↑)存
         136     /在表的起始单元。
         INXH    /表地址加1。
NXTCHR,  CALL    /从电传打字机取一个字符。
         TTYIN
```

```
          0
          MOVMA /把这个字符存入存储器。
          INXH  /表地址加1。
          CPI   /打入回车符了吗?
          015
          JNZ   /不，不是回车符。
          NOTCR
          0
CRLFOK,   CALL  /是，它是回车符，所以
          CRLF  /打印一个回车符和一个换行符。
          0
          JMP   /现在，从电传打字机取另一个字符。
          NXTCHR
          0
NOTCR,    CPI   /CTRL/C(终止输入例行程序用)
          003   /打入了吗?
          JNZ   /CTRL/C没打入，
          NXTCHR/从电传打字机取另一个字符。
          0
SRCH,     LXID  /它是CTRL/C，所以D和E
          LIST  /指向名字，地址和邮政编码表。
          0
START,    LXIH  /寄存器对H指向
          ZIPCOD /存储器中的测试邮政编码。
          0     /
CHECK,    LDAXD /从表中取一个字符。
          CPI   /它是被存在表末
          003   /的CTRL/C吗?
          JNZ   不是，判明它是不是邮政编
          NOEND /码的一位。
```

```
        0
        HLT     /它是CTRL/C，停机。
NOEND,  CMPM    /把这个字符与邮政编码相比
        JZ      /较。第一位数字一致。
        OK1     /试比较其余四位是否一致。
        0
        INXD    /不一致，表地址加1，
        JMP     /然后再试。
        CHECK
        0
OK1,    MVIB    /把必须完成的其余的比较次
        004     /数装入 B寄存器。
OK2,    INXH    /一个数字一致，表地址加1，
        INXD    /测试邮政编码的地址加1。
        LDAXD   /从表中取另一个字符。
        CMPM    /把它与测试邮政编码相比较。
        JNZ     /不一致，继续检查表
        START   /的其余部分。
        0
        DCRB    /有一个数字一致，计数减1。
        JNZ     /其余四位还没有检查，
        OK2     /继续试。
        0
BACKUP, DCXD    /比较结果，有一个五位数一致。
        LDAXD   /使表地址倒退
        CPI     /到前一个名字末。
        136
        JNZ     /已经找到了前一个名字末
        BACKUP  /的↑吗？没有，继续退。
        0
```

```
PRINT,  INXD    /已找到↑，使D内的地址加1。
        LDAXD   /从表中取得一个字符。
        CPI     /它是回车符吗？
        015
        JNZ     /不是，判明它是不是
        NPCRLF  /箭头(↑)的ASCII码。
        0
        CALL    /它是回车符，所以打印
        CRLF    /一个回车符和一个换行符。
        0
        JMP     /然后从表取
        PRINT   /下一个字符，判明
        0       /是什么字符。
NPCRLF, CPI     /它是名字末的↑吗？
        136
        JZ      /是的，不再打印表中
        PCRLF   /任何其他字符，
        0       /打印CR和LF，然后继续检索。该
        CALL    /字符不是↑，则打印
        TTYOUT  /该字符。
        0
        JMP     /现在，取另一个字符，并
        PRINT   /检查它。
        0
PCRLF,  CALL    /箭头找到了，
        CRLF    /在完整的名字和地址后面打印回
        0       /车符。
        JMP     /然后检查表的其余部分，看是否
        START   /一致。
        0
```

```
CRLF,   MVIA    /把回车符的ASCII码
        015     /装入A寄存器。
        CALL    /然后把它打印在电传
        TTYOUT/打字机上，或显示在CRT上。
        0
        MVIA    /然后把换行符的ASCII码
        012     /装入A寄存器。
        JMP     /然后把它打印在电传打
        TTYOUT/字机上或者显示CRT上。
        0
```

在例 5-13 程序的开头，8080把一个读/写存储器地址装入堆栈指示器，然后，把名字/地址表的始地址装入寄存器对H。MVIM指令把箭头(↑)符的ASCII码存入存储器，然后，寄存器对H中的存储器地址加 1。在NXTCHR，8080调用 TTYIN 子程序。当电传打字机或 CRT 上的一个键按下时，8080 把这个键的七位ASCII值存入A寄存器然后返回。这个值存入存储器，然后存储器地址加1。如果被按下的键不是回车键，那么，8080 跳转到 NOTCR。如果是回车键，则 8080 调用 CRLF 子程序，打印回车符和换行符。

8080 转到了 NOTCR，就能确定是否输入了 CTRL/C。如果没有输入，8080 返回到NXTCHR，输入另一个ASCII字符，并把它存入表中。如果送入CTRL/C，则 8080 执行 SRCH 处的指令。这意味着，CTRL/C的值被用来结束程序的名字和地址输入部分。CTRL/C只应在所有名字和地址都已送入之后才输入。请注意，CTRL/C的ASCII值存储在存储器。这就是我们前面用作为结束符(例5-9)的那个值。当然，CTRL/C“键”只能按下一次。

当CTRL/C送入时，8080执行从SRCH开始的指令，从而将包含测试邮政编码的名字和地址打印在电传打字机上，或者显示在CRT上。在8080打印邮政编码的过程中，箭头也打印出来吗？不，它不打印箭头，而是打印一个回车符和一个换行符。

测试邮政编码存入存储器

测试邮政编码是怎样被送入存储器的呢？在我们讨论过的程序例子中，没有一个包含能把测试邮政编码存入存储器的指令。只有名字与地址表中的名字和地址可送入存储器。为了输入测试邮政编码，我们可以在例5-13程序的基础上增添几条指令。这个指令序列如例5-14所示。这些指令应该插在例5-13程序的符号地址为SRCH的指令前面。这就是说，先把名字和地址送入表中，再送测试邮政编码。然后，8080才进行表的检索操作

例 5-14 输入测试邮政编码的程序

```
/可用这个程序把测试邮政编码送入
/8080 的存储器。被送入的
/任何字符，只要不是合法数字(0～9)
/都不予理睬。
  .
  .
LXIH        /寄存器对 H 指向
TEST        /ASCII 字符串。
0
CALL        /打印信息，即
```

```
        NXTLET      /"TEST ZIP CODE="
        0           /("测试邮政编码=")
        LXIH        /现在，寄存器对 H 指向将用来存
        ZIPCOD      /测试邮政编码的
        0           /读/写存储区。
        MVIC        /C 为要接收的数字的数目。
        005
ZIPIN,  CALL        /从电传打字机取一个字符。
        TTYIN
        0
        CPI         /这个字符比 ASCII0 小吗？
        060
        JC          /是，置之不理。
        ZIPIN
        0
        CPI
        072         /这个字符比 ASCII 9 大吗？
        JNC         /是，置之不理。
        ZIPIN
        0
        MOVMA       /这个字符是 0～9，则把它存入存
        INXH        /储器。存储器指示器加 1。
        DCRC        /数字计数器减 1。
        JNZ         /已输入了五个数字吗？
        ZIPIN       /没有。则取另一个。
        0
        .           /是的，输入了五个数字，继续执行该
        .           /程序的其余部分。
TEST    215         /这是 ASCII 信息组(CR)。
        212         /换行
```

```
            324         /T
            305         /E
            323         /S
            324         /T
            240         /空格
            332         /Z
            311         /I
            320         /P
            240         /空格
            303         /C
            317         /O
            304         /D
            305         /E
            240         /空格
            275         /=
            240         /空格
            000         /信息组结束符
NXTLET,  MOVAM          /从存储器取一个信息符到A。
            CPI         /这个字符是信息组结束符吗?
            000
            RZ          /是的，返回主程序。
            CALL        /不是，打印这个字符。
            TTYOUT
            0
            INXH        /存储器地址加1
            JMP
            NXTLET      /取另一个字符。
            0
```

把名字和地址输入到表内之后，就输入 CTRL/C。这样就

用 003(03)结束该表，然后 8080 开始执行例 5-14 所列出的指令。首先把 TEST 的地址装入寄存器对 H，这个地址就是存储在存储器中的一串 ASCII 字符的始地址，而这个 ASCII 字符串与名字和地址表是分开并有区别的。这之后，8080 调用 NXTLET 子程序。在 NXTLET，8080 从存储器读一个 ASCII 字符到A寄存器。然后把这个值与 0 比较。如果从存储器读出了 0 值，则 8080 从子程序返回。如果从存储器读出的值不是零，则 8080 调用 TTYOUT 子程序，把这个字符打印在电传打字机上，或者显示在 CRT 上。然后 8080 使存储器地址加 1，并转回到 NXTLET，从存储器读出另一个字符。

根据例 5-14 的程序，读者可以确定打印的是什么信息吗？首先打印的字符是回车符和换行符。接着是“TEST. ZIPCODE =”。请注意，在这个 ASCII 字符串的末了，从存储器读出零时，打印停止。

8080 从 NXTLET 返回之后，把存储邮政代码的读/写存储器地址装入寄存器对H。然后，把 5 存入C寄存器。这个数是邮政编码中包含的数字的个数，也是由下面的九条指令组成的输入循环要执行的次数。C 寄存器装入以后，8080 调用 TTYIN 子程序。把按下的键的七位 ASCII 码存入 A 寄存器，8080 从这个子程序返回。因为只用 0～9 这十个数字组成邮政编码，所以 8080 忽略所有非数字字符。8080 还忽略 ASCII 值小于 060 或大于 072 的任何字符。

一个合法的数字键（0～9）被按下以后，8080 执行 MOVMA 指令，把这个数存入存储器。然后，寄存器对H内的存储器地址加 1；C寄存器内的位数减 1。如果C寄存器的内容不等于 0，则 8080 执行 JNZ-ZIPIN 指令。这一程序循环允许 8080 一个接一个地接收和存储邮政编码的数字，直到输

入了五位代码为止。

一旦测试邮政编码存入了存储器，就可以在名字/地址表中寻找与测试代码相一致的邮政编码。如果找到，则打印整个名字和地址，包括邮政编码。8080查完整个表后便暂停工作。

当8080正在执行输入测试邮政编码的指令时，如果按下了以下的键，那将会发生什么情况呢？

A 12 NEXT 52+9

如果这些键被按下，邮政分区代码12529将作为测试邮政编码，存入连续的读/写存储器的存储单元。

检查邮政编码的检索程序

使用本章这一节中作为例子用的名字/地址和8080微型计算机系统，我们可以得到例5-15的结果。这个程序的一个限制是8080微型计算机的读/写存储器的容量。如果假设每个名字和地址占用60个存储单元，那么，1024(1K)字节的读/写存储器只能存储17个名字和地址。

程序的最后一个错误

在我们的邮政分区代码的检索程序中，最后还有一个错误。读者知道这是什么错误吗？假设在名字和地址表中包含下列的名字和地址：

Ed's Used Car Lot
24060 W. Main St.

San Diego, CA 93451 ↑

如果操作员要求微型计算机找出所有名字和地址中含24060这个邮政编码者，那将会发生什么情况呢？可惜的是，"Ed's Used Car Lot"的地址也将与包含24060邮政分区代码的所有名字和地址一起打印出来。问题在于8080微型计算机不能把五位的邮政编码和五位数字的街道地址，甚至五位数的建筑物编号、批号、雇员号或者订货单号区别开来。而所有这些号码都可能出现在一个名字和地址中。

怎么解决呢？有许多种解决办法。其中之一是在邮政分区代码之后紧跟着存入另一个符号(不是朝上箭头)。当然，打字员必须同箭头符一起打入这个符号。表中的一个地址和名字可如下所示：

Ed's Used Car Lot
24060 W. Main St.
San Diego, CA 93451 $
ATTN: Foreign Car Sales ↑

在这个例子中，我们使用美元符（$）来表示邮政编码的完了。软件唯一需要作的修改是，前面程序中的邮政编码检索部分。在判明五位数字的测试邮政编码与表中的一个邮政分区代码一致以后，程序接着检查邮政编码的最后一位数字之后是否为美元符。这个程序示于例5-16。

这些指令应正好插在BACKUP的前面。只有在五位数字一致时，才执行这些指令。这个程序仅仅是将寄存器对D内的地址加1，使之指向比较数的第五位数字后的字符。如果这个比较数是一个地址中的邮政编码，那么，寄存器对D应指向包含美元符的存储单元。LDAXD指令把这个字符装入A寄存器，并与立即数字节077进行比较。如果从存储器读出的不是美元

符的 ASCII 值，则 8080 执行 JNZ-AGAIN 指令。这就是说，如果一个街道地址或地址中不以美元符结束的任何其他数与测试邮政编码一致，8080 将执行这一转移。如果地址中的邮政编码与测试邮政编码一致，则 8080 不执行 JNZ-AGAIN，而是执行在 BACKUP 的指令。当然，为了使其正确操作，在每个邮政编码之后必须存一个美元符。

还有一些别的方法可用来辨别地址中的邮政编码。读者能想得出吗？有些方法可借助表的结构。例如，回车字符不是正好存在每个街道地址之前和每个邮政编码之后吗？在名字和地址的第二行出现邮政编码的可能性有多大呢？州名的缩写 (NY, CA, VA, MI) 不总是位于邮政分区代码之前吗？这些问题的答案会给读者指明其他可以用来区别邮政编码、街道地址或者其他五位数的方法。

最后要说明的一点是，如果在每个邮政编码后存一个美元符，那么应该把例 5-17 的指令添加到你的程序上。这个指令序列可以防止打印美元符，如果地址中包含所要找的邮政编码的话。

例 5-15 邮政编码检索程序执行示例

```
APEX INC.
20 MAIN. ST.
TRIANGLE. VA 24061↑

XYZ BLASTING
22 NORTH MAIN
BOOM BOOM, VA 24060↑

EASY FIRE INSURANCE
2280 W. ALPINE
```

REDHOT, VA 24061↑

ABC PENCIL CO.
ARQUES AVE.
LEADVILLE, VA 24060↑

CLEAR PLASTICS, INC.
23 HARDY. AVE.
CLOUDY, VA 24060↑

MA'S FAST FOODS
2105 NE WEST ST.
SPEEDY, VA 24061↑

试验# 1：
测试邮政编码=24060

XYZ. BLASTING
22 NORTH MAIN
BOOM BOOM, VA 24060

ABC. PENCIL. CO.
ARQUES AVE.
LEADNLLE, VA 24060

CLEAR. PLASTICS, INC.
23 HARDY AVE.
CLOUDY, VA 24060

试验# 2：
测试邮政编码=24061

APEX INC.
20 MAIN ST.
TRIANGLE, VA. 24061

EASY FIRE INSURANCE
2280 W. ALPINE
REDHOT, VA 24061

MA'S FAST FOODS
2105 NE WEST ST.
SPEEDY, VA 24061

其他测试：
测试邮政编码＝23405
测试邮政编码＝12358

例 5-16 找邮政编码的界符。

```
REALLY, DCXD      /加 1，越过第五位数字。
        IDAXD     /取这个字符到 A。
        CPI       /它是美元符吗？
        077       /(077＝十六进制 3 F)。
        JNZ       /不是美元符，因此
        AGAIN     /在两个邮政编码之间，没有出
        0         /现五位数字的一致，继续检索。
BACKUP, DCXD      /有一次五位数字的一致。
        .
        .
```

例 5-17 防止打印美元符

```
        .
        .
```

```
PRINT, INXD/找到了↑，使地址加 1。
       LDAXD/从表中取一个字符。
       CPI     /它是存储在邮政分区
       077     /代码后的美元符号吗？
       JZ      /是的，则不打印它。
       PRINT   /从表中取另一个字符。
       0
       CPI     /不是美元符，是
       015     /回车符吗？
         •
         •
```

第六章 排 序

所谓排序就是按由大到小、由小到大或按字母的顺序重新排列表、表格、纪录或文件中的数据值。为什么程序设计员要对数据值进行排序呢？如果必须存取的数据值是经过排序的，那么程序往往运行得快得多。现在假定有两盘磁带。一盘存储着上学期有一门课不及格的所有学生的姓名，另一盘存储着所有学生下学期的全部课程表。如果一个学生某门功课不及格，该生就应该在下学期重上这门课。因此，学生的课程表中应该包含不及格的科目。

如果两盘磁带没有按字母顺序排列姓名，那么微型计算机在一盘磁带上找一个学生不及格的功课，然后在另一盘磁带上找学生的课程表，则要花费很多的时间。

在许多情况下，一台微型计算机不会连接两台磁盘驱动器。但是，微型计算机可以用于智能磁带驱动器的控制器中。主计算机可以命令控制器中的微型计算机对磁带上的第十个记录或文件进行排序。如果磁带驱动器中没有用微型计算机，那么主计算机将必须去做排序工作。

表 6-1 包括若干排序的和未排序的列。在这个表中，有一个列（第三个）是按照递升的顺序排列的；另一列(第一个)是按照递减的顺序排列的。如果这个表内的数据值不是按递升次序、递减次序或字母顺序排列，那么，这个表格未经排序（如表 6-1 的第二和第四个表格)。

有许多种不同的排序法，可以用来对数据值排序，其中包

表 6-1 排序的和未排序的表格

存储器单元	排 序 的	未排序的	排 序 的	未排序的
X	5	3	1	4
X + 1	4	2	2	2
X + 2	3	4	3	3
X + 3	2	1	4	5
X + 4	1	5	5	1

括**直接插入法，对分插入法、冒泡法、快速法、堆积法、合并交换法和双路合并法**。总之，所有这些排序法都可以归结为下面五种方法之一：

1. 插入法
2. 交换法
2. 选择法
4. 合併法
5. 分配法

我们将要讨论，插入和交换这两种排序法。

数字值的排序

插入排序(直接插入排序)

我们将要讨论的最简单的插入分类法是直接插入分类法。假设下列的数存储在读/写存储器连续的存储单元：

5 存储在最低存储地址的存储单元，而 1 存储在最高存储地址的存储单元。如果使用直接插入排序法，8080 将把存储

在X+1存储单元的3与存储在X存储单元的5比较。由于3小于5，所以把5向上移动一个存储单元，从X移到X+1，并且把3写入到存储单元X，它原来存放5。现在表的顺序是：

存储单元：	X	X+1	X+2	X+3	X+4
内　　容：	3	5	4	2	1

接着，下一个较高存储单元内的下一个数据值，$X+2$中的4与$X+1$中的5相比较。因为4比5小，但4又比3大，所以5向高端移一个存储单元，即移到X+2。然后8080把4写入X+1存储单元。X+1原来存储5。

存储在X+3存储单元内的2是8080必须检查的下一个节点。因为2比3、4和5都小，所以这三个数据值都往高端移动一个存储器单元。5移到X+3，4移到X+2，3移到X+1，然后，2写入存储单元X。现在这个表就成了下面这个样子：

存储单元：	X	X+1	X+2	X+3	X+4
内　　容：	2	3	4	5	1

当8080检查这个表格的最后一个节点，即1时，它判定这个节点比这个表格中所有其它节点都小。因此，所有其它节点都向上端移一个存储单元，然后把1写入存储单元X。

存储单元：	X	X+1	X+2	X+3	X+4
内　　容：	1	2	3	4	5

因为8080已经检查并根据需要移动了表中最后一个节点，所以现在这个表是排序过的。只有表内所有节点都已被检查之后，才能下这个结论。从这个表的例子可以看到，直接插入法要求8080进行许多次比较。根据这些比较，数据值可移到表的不同位置上。例6-1内的子程序，利用直接插入排序法给不

带符号的八位二进制数排序。

当 8080 调用例 6-1 子程序时，寄存器对H内必须存有表的始地址，寄存器对D内必须存有节点数，即表中包含八位数的数目。

ISORT 处的第一条指令把寄存器对H内的表的始地址保存在堆栈。然后，这个 16 位地址减 1，求反。INXH 指令形成表始地址之前一个地址的对 2 补码。如果表的始地址是040100(2080)，那么，执行 MOVLA 指令后，寄存器对H的内容是什么呢？是 337201(DF 81)。

例 6-1 用直接插入法的表排序子程序

```
/这个子程序用插入排序
/法给存储器中的单精度
/（8 位）不带符号的数排序。进入此子程序时，表
/的始地址存储在寄存器对 H，节点数
/存储在寄存器对 D。
ISORT,   PUSHM   /表地址保存在堆栈。
         DCXH    /地址减 1。
         MOVAH   /现在形成
         CMA     /这个地址的
         MOVHA   /2 的补码。
         MOVAL   /因而 8080 知道
         CMA     /表在低端的界限。
         MOVLA   /1 的补码在寄存器对 H 内。
         INXH    /现在，变为 2 的补码。
         SHLD    /表的低端界限的 2 的
         BOTTM   /补码存入 BOTTM。
          0
UP1      POPH    /现在从堆栈取出地址。
         INXH    /寄存器对 H 内的地址加 1。
```

```
        MOVBM   /从存储器取一个节点（数据值）到 B。
        DCXD    /寄存器对 D 存储的节点数减 1。
        MOVAD   /判别它是不是 0。
        ORAE    /这指明何时整个
        RZ      /表已经排序完毕。
        PUSHH   /保存加 1 后的地址。
DOWN 1, DCXH    /表地址减 1。
        PUSHD   /保存节点数。
        XCHG    /表地址取到 D 和 E 寄存器。
        LHLD    /取低端边界地址。
        BOTTM
        0
        DADD    /把这个地址和最低地址相加。
        MOVAH   /其结果是零吗？
        ORAL
        XCHG    /把地址取到 H 和 L。
        POPD    /从堆栈弹出节点数。
        JZ      /是的，已经到达表的起点，
        FARENF  /所以，B 寄存器内的数。
        0       /存入表内。
        MOVAM   /没到表的起点，取一个节点
        CMPB    /到 A。它比 B 的内容小吗？
        JC      /是的，把 B 的内容存入表中。
        FARENF
        0
        INXH    /地址加 1。
        MOVMA   /把节点保存在较高地址。
        DCXH    /然后使地址减 1。
        JMP     /检查它是否能够向低端
        DOWN 1  /移到表内另一个位置。
```

```
        0
FARENT, INXH    /表内的节点较小，
        MOVMB   /地址加 1，并保存 B 的内容。
        JMP     /检查表内
        UP I    /下一个连续的节点，看看它。
        0       /是否应该送到较低地址上。
BOTTM,  0       /最低地址的对 2 补
        0       /码存在这里。
```

	八进制	二进制	十六进制
始地址：	040 200	00100000 10000000	2080
减 1 后：	040 177	00100000 01111111	207 F
取反：	337 200	11011111 10000000	DF 80
加 1：	337 201	11011111 10000001	DF 81

然后把这个 16 位结果存在读/写存储器的 BOTTM 处。8080 将用这个数来确定它到达表的始点或终点的时刻。

为了说明 ISORT 子程序如何操作，假设 8080 必须对下表进行排序：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内　　容：	5	3	4	2	1

8080 计算了表的始地址下面，紧靠始地址的一个地址的 2 的补码以后，在 UP 1 处，把表的始地址从堆栈取出，送入寄存器对 H。现在，在我们的例子中，寄存器对 H 指向的存储单元内存有 5 这个数。然后，使这个地址加 1 （变为 X＋1），并将 X＋1 地址内的内容(3)传送到 B 寄存器。然后寄存器对 D 内的节点数（一开始为 005，因为这个表有 5 个节点)减 1，并检查结果是否等于 0。因为结果不等于零，所以，8080 不执行

RZ 指令，而是把加 1 后的地址存入堆栈。加 1 后的地址指向 X＋1 存储单元。该单元所存内容是 3。

程序执行到 DOWN 1 时，8080 使寄存器对 H 的内容减 1。因为寄存器对 H 内的地址已减 1，所以，8080 必须确定寄存器对 H 的内容是否小于表的始地址。为此把寄存器对 D 内的节点数存入堆栈，把寄存器对 H 内的减 1 后的地址交换到寄存器对 D。然后，比表的始地址小 1 的地址的对 2 补码装入寄存器对 H，与寄存器对 D 内的表地址相加。如果相加的结果等于零，那么，8080 把表的地址减 1 后，已越过了表的始地址。为了检查这种状态，H 寄存器的内容传送到 A 寄存器，并与 L 寄存器的内容进行或操作。8080 执行完 ORAL 指令以后，把寄存器对 D 中的表地址交换回到寄存器对 H，然后从堆栈取节点数送寄存器对 D。ORAL 指令后没有任何指令影响 8080 的任何标识位。

如果零标识位变成逻辑 1，则 8080 转到 FARENF。但是，此例中，寄存器对 H 的内容已被减为 X，而该单元内存着 5 这个数。因此，808 0并不转到 FARENF。相反它把 5 从存储单元 X 传送到 A 寄存器，并与 B 寄存器内的 3（来自 X＋1）进行比较。3 比 5 小，所以，8080 不执行 JC-FARENF 指令，而是把寄存器对 H 内的存储地址加 1 成为 X＋1，并把 A 寄存器内的 5 存入这个存储单元。此时，X 和 X＋1 这两个存储单元都存着 5 这个数。请记住，3 这个数还在 B 寄存器内。然后寄存器对 H 中的地址减 1 成为 X，8080 转回 DOWN 1。现在这个表成了下面样子：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内　　容：	5	5	4	2	1

（B 寄存器的内容是 3）。

在 DOWN 1 处，寄存器对H内的地址再次减 1而成为X－1。这比表的始地址少 1。当 BOTTM 的 16 位内容加至这个地址时，其结果等于零，因此，8080 转到 FARENF。在 FARENF，这个地址加 1。又回到 X，B 寄存器的内容(3) 存入这个存储单元。现在，表内各节点按照下面的顺序排列：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内容：	3	5	4	2	1

8080 执行 MOVMB 指令以后，转移到 UP 1。可以看到，这个子程序的大多数指令用来保证 8080 不超越表的终点(通过节点数实现)或始点 (通过 BOTTM 内存的对 2 的补码数来实现)。

在 UP 1，从堆栈弹出表地址，加 1 成为 X＋2。现在，寄存器对H指向存有 4 这个数的存储单元。4 从存储器传送到B寄存器，D 寄存器内的节点数减 1，成为 003。这个数不等于零，所以，8080 把地址X＋2 压入堆栈，然后将它减 1，变成X＋1，并把 BOTTM 的内容与它相加。相加的结果不等于零，所以，8080 把X＋1 单元的内容(5) 装入A寄存器，并把它与B寄存器内的 4 进行比较。B 寄存器的内容比A寄存器的内容小，所以，8080 必须把 5 写入存 4 的表地址内。因此，寄存器对H内的地址加 1，成为X＋2，B 寄存器内的 5 存入这个存储单元。5 这个数再次存入表的两个存储单元。

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内容：	3	5	5	2	1

(B寄存器含有 4。)

5 写入存储单元X＋2 以后，寄存器对H内的地址减 1，成为X＋1。8080 转移到 DONN 1。寄存器对H内的存储器地址减 1 而变为X。因为这个地址不小于表的始地址，所以，存储

器单元X中的3装入A寄存器，并与B寄存器中的4进行比较。B寄存器的内容大于A寄存器的内容，所以，8080执行JC—FARENF。在FARENF，寄存器对H的内容加1而成为X+1，然后，把4存入这个存储单元。

存储单元：	X	X+1	X+2	X+3	X+4
内容：	3	4	5	2	1

读者可以推测，当把2从存储单元X+3读入B寄存器并执行ISORT子程序的各指令段时，表将呈如下形式：

首先，我们有

存储单元：	X	X+1	X+2	X+3	X+4
内容：	3	4	5	5	1

然后

存储单元：	X	X+1	X+2	X+3	X+4
内容：	3	4	4	5	1

次后

存储单元：	X	X+1	X+2	X+3	X+4
内容：	3	3	4	5	1

最后有

存储单元：	X	X+1	X+2	X+3	X+4
内容：	2	3	4	5	1

当1从存储单元X+4读入B寄存器时，表内的数据值往高地址方向各移一个存储单元，如下所示：

存储单元：	X	X+1	X+2	X+3	X+4
内容：	2	3	4	5	5
内容：	2	3	4	4	5
内容：	2	3	3	4	5
内容：	2	2	3	4	5

内容：　　1　　2　　3　　4　　5

总之，这种排序方法是从表中读一个节点，并把它写入到8080的B寄存器。然后把这个值与在**较低存储器地址**上的下一个连续的节点进行比较。如果表内的节点大于B寄存器内的节点，那么把表内的节点往高端移一个位置（本例为移一个存储单元）。然后，B寄存器的内容与下一个较低存储单元内的节点进行比较。只有B寄存器内的节点大于表内的节点时，才把它写回到表内。但是，从例子中可看到，如果8080"退回"到表的起点或始地址，则B寄存器的内容也被写回表中。在此例子中，表内的数没有比1小的，所以，最后把这个数存在表的始点。

插入排序法在最佳和最坏状态下的排序时间

假定数值1、5、2、7、6、8、3和4存在一个表内。适当地排列这些数，使ISORT子程序花费最少的时间就能完成这个表的排序。这个时间就是**最佳状态时间**。如果这些数从最低存储地址到最高存储地址排成为1、2、3、4、5、6、7和8，那么，ISORT子程序需要的排序时间最少。换句话说，当调用ISORT子程序时，这个表已经排完序。最坏情况的时间出现在表内的数按递减次序排列时，例如排成8、7、6、5、4、3、2、1。这就是说，ISORT子程序要用最多时间来排序。当然，大多数表("平均"表)需要的排序时间介于最佳状态时间和最坏状态时间之间。

交换排序法(冒泡法)

交换法是与上述插入法不同的另一种排序方法。我们要讨论的交换排序法叫做冒泡法。极简单冒泡法是比较一个表中的

两个节点。如果存储单元X内的节点比存储单元X＋1内的节点大，则两者交换位置。反之，如果存储单元X内的节点小于X＋1内的节点，则不交换位置。这里假设，我们是要按从小到大的顺序排表，而且节点是单精度的(8位二进制数)；在本章的上面一节也作了相同的假设。

不管是否进行交换，微型计算机接着比较X＋1和X＋2这两个存储单元内的节点。如果存储单元X＋1内的节点比存储单元X＋2内的节点大，这两个节点交换位置。反之则不交换。然后，8080对X＋2和X＋3这两个存储单元内的节点进行比较。这一比较与交换过程一直继续下去，直到8080到达表的终点为止。**如果任何时候发生了任何交换，8080就把使一个寄存器或一个存储单元置于某一特定值，表示发生了一次位置交换。**

8080检查完整个表以后，就检查这个寄存器或存储单元的内容，看看是否发生过任何交换。如果有，这个寄存器或存储单元被复位或重新置为某一已知的初值，然后再次检查。只有整个表被检查完毕，并且没有发生任何交换，排序才告结束。

根据上面的叙述，最大的节点(数据值)将存在什么地方？存在表的始点还是终点呢？在表的范围以内，最大的数据值最终将冒到表的顶点。这种冒泡排序过程如表6-2所示。开始，表6-2内的表格包含下列数据值：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内容：	3	2	5	4	1

第一次检查表，即第一遍排序时，3和2相比较。3比2大，这两个数交换位置。因为发生了一次交换，交换指示器被置位。然后，存储单元X＋1内的3与X＋2内的5相比较。

表 6-2 数据表的冒泡排序

存储地址	原来的排列	第一次	第二次	第三次	第四次	第五次
X	3	3	2	2	2	1
X+1	2	2	3	3	1	2
X+2	5	5	4	1	3	3
X+3	4	4	1	4	4	4
X+4	1	1	5	5	5	5
		结果	结果	结果	结果	结果
X	3	2	2	2	1	1
X+1	2	3	3	1	2	2
X+2	5	4	1	3	3	3
X+3	4	1	4	4	4	4
X+4	1	5	5	5	5	5

因为5比3大，不需交换位置。但是，接着5与存储单元X+3中的4相比较，两数需要交换，因此，交换指示器再次被置位。现在，这个表的排列如下所示：

存储单元：	X	X+1	X+2	X+3	X+4
内容：	2	3	4	5	1

存储单元X+3中的5与存储单元X+4中的1相比较并交换位置，交换指示器第三次被置位。正如读者所看到的那样，此时5已上推到表的顶点。

存储单元：	X	X+1	X+2	X+3	X+4
内容：	2	3	4	1	5

因为整个表已检查完，所以，8080检查交换指示器的状态。因为它置成1，所以8080把它清零，然后再次开始检查这个表，即第二遍排序。从表6-2可以看到第二、三、四遍排序的结果。最后，在第五遍排序时，没有任何节点要交换，所

以，交换指示器仍保持为清零状态。8080 读出这种状态，然后返回到调用冒泡排序子程序的主程序。当 8080 从这个子程序返回时，表已是排序了的。

例 6-2 列出了用冒泡排序法给表排序的一个子程序。当调用 BSORT 子程序时，必须把表的始地址（最低的存储单元地址）装入寄存器对H，把表的末地址（最高存储单元地址）装入寄存器对 D。BSORT 子程序的头六条指令的作用是从表的末地址减去表的始地址，其结果存储在寄存器对D。这些指令做些什么呢？它们计算出表的始地址和末地址之差即节点数。

在例 6-1 中，当调用 ISORT 子程序时，表的始地址应在寄存器对H内，表的节点数应在寄存器对D。但是，有时表的始地址和末地址是已知的，而节点数是未知的。因此，BSORT 子程序的头六条指令就用来计算表的节点数。这个指令段也可用于 ISORT 子程序的开头。如果在 ISORT 子程序中使用这些指令，MOVDA 指令之后应增加一条 INXD 指令，如例 6-3 所示。

例 6-2 采用交换排序方法（冒泡排序法）的表排序子程序

```
/这个子程序利用冒泡排序方法对
/存储器中的单精度(8 位)不带符号数
/进行排序。进入子程序时，表的
/始地址(较低的地址)存储在
/寄存器对H，末地址存在寄存器对D。

BSORT,  MOVAE   /从E寄存器取末地址的低位字节
        SUBL    /到A，减去始地址的低位字节，
        MOVEA   /结果存入E寄存器。
        MOVAD   /从D取末地址的高位字节到A。
```

```
        SBBH    /减去始地址的高位字节。
        MOVDA   /结果存入D寄存器。
PASS 1, PUSHH   /始地址存入堆栈。
        PUSHD   /节点数存入堆栈。
        MVIC    /C寄存器用来表示
        000     /是否发生了交换。
UP 1,   MOVAM   /从存储器取表的一个节点到A。
        INXH    /寄存器对H的内容加 1，指
        CMPM    /向下一个节点。
        JC      /A和存储器单元的内容相比较。
        MEXT    /存储单元的内容大于A，则不交换，
        0       /而是检查表内下两个节点。
        JZ      /如果它们相等，
        MEXT    /也不交换。
        0
        MOVBM   /存储单元的内容取到B。
        MOVMA   /A寄存器的内容存入存储器。
        DCXH    /后退一个存储单元，
        MOVMB   /并把B的内容存入其中。
        INXH    /表地址加 1。
        MVIC    /用C寄存器来表示
        377     /发生了交换。
NEXT,   DCXD    /寄存器对D中的节点数减 1。
        MOVAD   /节点数是 0 吗？
        ORAE
        JNZ     /整个表还没有检查完毕，所以
        UP 1    /继续检查。
        0
        MOVAC   /把交换指示器的内容取到
        ORAA    /A寄存器，并使标识位置位。
```

```
POPD    /节点数装入寄存器对D，
POPH    /表的起始地址装入寄存器对H。
JNZ     /发生过交换，所以
PASS 1  /再一次检查表。
0
RET     /没有交换，从子程序返回。
```

BSORT 子程序（例 6-2)在求出节点数以后，把寄存器对H里的表的始地址和寄存器对D里的节点数存入堆栈。然后，C寄存器即“交换指示器”清零。在 UP 1，寄存器对H寻址的存储单元X中的第一个节点移到A寄存器，然后，寄存器对H内的存储器地址加1，成为X＋1，并把这个存储单元里的节点与A寄存器的内容进行比较。如果存储单元的内容比A寄存器的内容大，则 808 0转到 NEXT。在这种情况下，两个节点不交换位置。如果这两个节点相等，8080 也转到 NEXT。

如存储单元X＋1 的内容比A寄存器的内容小，则必须交换位置。因此，8080 不转到 NEXT。假设我们用 BSORT 子程序对表格 6-2 中的表排序。这个表的排列如下所示：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内容：	3	2	5	4	1

这就是说，3 和 2 这两个数必须交换位置。请记住，3 存储在A寄存器里，2 则存储在寄存器对H寻址的存储单元（X＋1)里。为了进行交换，8080 把存储器中的 2 传送到B寄存器，然后，把A寄存器的内容 3 存入寄存器对H所寻址的存储单元（X＋1)。现在，表内的数如下所示：

存储单元：	X	X＋1	X＋2	X＋3	X＋4
内容：	3	3	5	4	1

3 存入存储单元X＋1 之后，寄存器对 H 里的存储地址减

1，并把B寄存器中的2写入这个存储单元X。在存储器地址加1，从而回到X+1之后，377(FF)这个数被装入C寄存器，表示在这遍排序中曾发生交换。8080执行这些交换指令之后，接着执行在NEXT处的指令。

在NEXT，8080确定它是否已经到达了表的顶部。为此，8080将寄存器对D内的节点数减1并检查“零状态”。当然，DCXD指令不影响8080的任何标识位，因而必须执行MOVAD和ORAE这两条指令，以便根据寄存器对D内的16位数将标识位置1或者清零。如果节点数不等于零，8080转到UP 1。在此例子中，8080只对表中3和2进行了比较和交换。所以，8080转到UP 1。

例 6-3 ISORT子程序(例6-1)中计算节点数的指令序列

```
/这个子程序用插入算法对
/存储器中单精度(8位)不带符号数进行排序。
/调用它时，
/表的始地址在寄存器对H内，
/节点数在寄
/存器对D内。

ISORT, MOVAE  /从E取末地址的低位字节到A。
       SUBL   /减去表的始地址的低位字节。
       MOVEA  /差的低位字节存入E。
       MOVAD  /取末地址的高位字节到A。
       SBBH   /减去始地址的高位字节。
       MOVDA  /差的高位字节存入D。
       INXD   /差值加1。
       PUSHH  /寄存器对H内的表地址存入堆栈。
       DCXH   /H内的地址减1。
```

```
MOVAH /现在，形成这个地址的对2的补码
CMA   /(该地址比表的
MOVHA /始地址小1)。
```

当8080这一次执行到UP 1的指令时，它把存储单元X+1内的3与存储单元X+2内的5进行比较。比较的结果没有产生交换。但是，在第一遍排序中，5和4相交换，然后1和5相交换。这之后，寄存器对D内的节点数减到0。因此，8080执行ORAE指令之后不转到UP 1，而是检查交换指示器的状态。如果没有发生交换，则C寄存器的内容是0。如果至少发生了一次交换，则C寄存器的内容是377(FF)。为了确定交换指示器的状态，把C寄存器的内容传送到A寄存器，然后，8080执行ORAA指令，根据A寄存器的内容，把标识位置1或者清零。

执行ORAA指令，使标识位置位或清零；所以，然后，8080从堆栈弹出表的始地址和节点数。如果C寄存器的内容是377(FF)，即在最后一遍排序中至少发生了一次交换，那么，8080执行JNZ-PASS 1。在PASS 1，8080把寄存器对H和D的内容压入堆栈，把C寄存器清零。然后，8080再次检查整个表。

如果C寄存器的内容是0，那么在最后一遍检查表期间没有发生交换。因此，这个表是排好序的，8080便从BSORT子程序返回。根据这一说明，现在读者应该知道，对于表6-2中的表格，为什么8080必须进行五遍排序。到最后一遍，所有的节点都已按适当顺序排列，所以，交换指示器为0。随后，8080从BSORT子程序返回。

冒泡法在最佳和最坏情况下的排序时间

正如读者所预料的那样，8080 要对一个表排序，需要的扫描次数介于 1 和 n 之间，其中 n 代表节点数。如果表已排好序，那么 8080 调用 BSORT 子程序后只要对该表扫描一遍。在这一遍扫描的末了，因为没有发生交换，C 寄存器的内容将是零。因此，8080 将从 BSORT 子程序返回。如果表的节点以完全颠倒的顺序排列，那么，8080 必须对表扫描最多的次数（n 遍）。这和我们在上面讨论的插入排序法的最佳和最坏情况的排序时间相同。

插入排序法和交换排序法的比较

ISORT 和 BSORT 这两个子程序都可以用来给表排序。至于哪一个子程序较好，则完全由所需排序时间的长短决定。为了获得排序时间的数据，我们用 ISORT 和 BSORT 子程序对一个包含 256 个不带符号的 8 位数的表进行排序。这些子程序用来给表排序分三种情况：(1) 已经排好序；(2) 节点按颠倒顺序排列；(3) 包含的数全相同。所得结果如表 6-3 所示。

表 6-3　ISORT和BSORT子程序的排序时间

表的节点	ISORT	BSORT
已排序	11.13 毫秒	3.41 毫秒
颠倒顺序排列	1.05 秒	1.26 秒
数都相同	1.05 秒	4.05 秒

从表 6-3 可以看到，只有当(1)表已经排好序或(2)表内包含的数都相同时，冒泡法才比插入法快。这两种情况都不是很

常见的。当有大量的排序操作(插入或交换)要做时，例如在表中的数以相反的顺序排序的情形里，插入排序子程序(ISORT)所需要的时间(1.05 秒)，比冒泡排序子程序(BSORT)所需要的时间(1.26 秒)要少些。当然，在大多数情况下，排序时间介于已排序表和相反顺序表所需要的排序时间之间。

ISORT 和 BSORT 子程序的局限性

ISORT 和 BSORT 子程序能对之进行排序的表的最多字数有多少呢？65,536 个 8 位字(节点)。这是因为子程序中用寄存器对D来存放节点数。当然，这两个子程序都只能对 8 位的二进制数进行排序。但是，稍用一点技巧，对两个子程序进行修改，就可以完成对 16 或 24 位数的排序。这意味着必须完成 16 位或 24 位数的比较。但是 8080 没有 16 位或 24 位的比较指令，因而必须执行两次或三次 8 位的减操作。当然，这些子程序也可以修改成能对带符号的数或浮点数进行排序。

如果试图用这些子程序对只读存储器(ROM)中的数据值进行排序，那将会发生什么情况呢？我们希望读者绝不要去做这种不幸的尝试，可是，有时可能发生这种情况。如果使用 ISORT 子程序，8080 从子程序返回时并不能改变表的排列顺序，虽然它力图这样做。然而如果使用 BSORT 子程序，则 8080 根本就不会从该子程序返回。

请记住，ISORT 子程序只对表扫描一遍。在 BSORT 子程序中，当 8080 到达表的终点时，它检查C寄存器的内容，确定是否发生了任何交换。如果发生了交换，8080 将进行另一次扫描。因为表是存储在 ROM 中，决不可重新排序，所以 8080 将一遍遍地继续对它扫描。只有一种情况可使 8080 退出 BSORT 子程序，即使表是存储在 ROM 中。读者知道是什么情况吗？

如果ROM中的表已排好顺序，那么，8080只对表扫描一遍以后，就将从BSORT子程序返回。这是因为没有发生任何交换。

字母-数字串的排序

至此读者已经看到了两种不同的数字值的排序法——插入法和交换法。这两种方法也可以用来给字母-数字串排序，即按字母顺序将它们排列好。但是，在例子中，我们只采用交换排序法(冒泡法)。在由ASCII字符组成的字符串排序中遇到的一个困难是：待排序的字符串可包含20或30个字符，而在我们前面所讨论的排序例子中要排序的是单精度数据值。现在，我们还面临着对那些连长度都可能不同的字符串进行排序的问题。假设我们要8080对表6-4中的名字排序，可以看到，表中有三个名字包含五个字母，两个名字包含六个字母，一个名字包含八个字母。

表 6-4　包含有六个名字的表

SMYTHE
SMITH
JONES
LEWIS
PETERSON
PETERS

为对表6-4内的名字排序(按字母次序排列)必须从比较头两个名字开始。因为SMITH在表中应该排在SMYTHE之前，所以这两个名字应交换位置。同时，两个名字所包含的字符数

不同(字符串是不同"长度"的)，所以，最简单的做法是把名字SMYTHE移到暂时存储区，然后把SMITH送到表中原来放SMYTHE的那些存储单元，再从暂时存储区把SMYTHE送回到表中，存放在SMITH的后面。

因为我们是使用冒泡排序法，所以一定要设置一个交换指示器。C寄存器将用作于交换指示器。如果8080到达表的终点时，C寄存器的内容等于377(FF)，那么可以肯定，在最后一次扫描期间，至少发生了一次交换。这就是说，8080必须至少再排序一遍。如果C寄存器的内容等于0，那么表是排好序了的，8080就从子程序返回。

要使子程序正确地对名字排序(即按字母顺序排列)，表中的字符串必须具有一定的格式(结构)。除非使用这种格式，否则就不能把字符串填入表中。我们将要使用的格式规定如下：在每一个字符串最后一个字母或数字字符之后必须存一个零，作为它的最后一个字符，即字符串的结束符。如果字符串是表中最后一个字符串，那么除了在该字符串最后存一个零之外，还应在这个结束符之后存入字符233(9 B)，作为表的结束符。可以用来给字母数字字符串排序的子程序列在例6-4中。

ABSORT子程序一开始，零被装入C寄存器，表示没有发生任何交换。然后，表的始地址，即表内第一个字符串的第一个字符的地址，装入寄存器对H。同时，该地址被存入存储器中FIRST处。8080执行到UP 1，开始确定表内第二个字符串的第一个字符。8080很容易找到这个字符，因为它存储在表内第一个零(第一个字符串的结束符)之后。因为零不是一个可打印的ASCII字符，它不会包含在任何一个名字之中。因此，选它作为字符串的结束符是很合适的。请不要把数字0与ASCII 0混同起来，ASII 0等于八进制260或十六进制B0。

8080 找到了第二个字符串的第一个字符后就把它的地址存入存储单元 SECOND。现在，8080 已经知道头两个字符串的第一个字符存储在存储器的何处。因此，在 AGAIN 之前，第二个字符串的始地址被交换到寄存器对D中，第一个字符串的始地址则装入寄存器对H。

现在，8080 必须一个字符对一个字符地比较这两个 ASCII 字符串。如果是对表 6-4 里的名字排序，8080 将确定 SMYTHE 应排在 SMITH 之后，所以这两个名字在表内的位置应对调。然后，8080 对 SMYTHE 和 Jones 进行比较，交换它们的位置。8080 将继续这种比较和交换操作，直到整个表被排序好（按字母顺序排列）。

这就是说，8080 在 CMPNXT 处把第二个字符串中的第一个字符装入A寄存器。如果它是零，那就找到了第二个字符串的结束符。接着 8080 检查寄存器对H寻址的存储器单元是否含有第一个字符串的结束符 0 。如果两个字符串的结束符都已找到，那么，这两个字符串相等；例如，对表内的名字 PORTER 和 PORTER，或 JONES 和 JONES 进行比较。因此，不应发生位置交换，所以，8080 转到SECBIG，开始对表的第二、三个字符串进行比较。

假设 8080 在第二个字符串的末尾找到了零，但在第一个字符串的同一个“位置”上找到一个字母或数字。例如比较字符串 ABS(第一个字符串)与字符串 AB(第二个字符串)。字符串 AB 的第三个字符是零。但字符串 ABS 的第三个字符是 S 。所以，这两个字符串应该交换位置。注意，仅仅一个字符串比另一个字符串长，还不能决定应该交换位置。如果两个字符串各对应字符相同，但第二个字符串较短，则应该交换位置。8080 通过转到 EXCH 来完成这种交换。

如果 8080 在第二个字符串末没有找到零，那么，它转到 ENDLST，对从第二个字符串读出的数与表的结束符 233(9B) 进行比较。如果两者相等，那么 8080 已经达到了表的尽头。因此，它必须检查C寄存器的内容，确定上一遍排序中是否发生了任何交换。当 8080 转到 DONE 时，就检查C寄存器的内容。如果C寄存器的内容是 0，8080 返回到 ABORT，再检查表内其它字符串。

例 6-4 用交换法对字母数字串排序

```
ABSORT,   MVIC    /把交换指示器
          000     /置 0 (表示没有任何交换)。
          LXIH    /把包含 ASCII 字符串的
          LIST    /表的始地址装入
          0       /寄存器对 H。
NEXT,     SHLD    /把这个地址保存起来，作为第一个
          FIRST   /ASCII 字符串的始地
          0       /址。
UP1,      MOVAM   /从这个字符串取一个字符到 A。
          INXH    /寄存器对 H 内的存储器地址加
          CPI     /1。找到第一个
          000     /字符串的结束符了吗?
          JNZ     /没有，继续找。
          UP1
          0
          SHLD    /找到了，保存这个地址，作为
          SECOND  /第二个 ASCII 字符串的
          0       /始地址。
          XCHG    /上述地址放入D和E。
AGAIN,    LHLD    /第一个字符串的始地址装入H
          FIRST   /和 L。
```

```
          0
CMPNXT,   LDAXD   /从第二个字符串取一个字符。
          CPI     /第二个字符串结束了吗?
          000
          JNZ     /未结束,
          ENDLST  /判断是不是表的末尾
          0
          CMPM    /第一个字符串的末尾也是 000 吗?
          JZ      /是,这两个字符串相等,
          SECBIG  /不交换位置。
          0
          JMP     /在第二个字符串的末尾找到了
          EXCH    /000,但第一个字符串的末尾不
          0       /是 000,第一个字符串>第二个
ENDLST,   CPI     /字符串,交换它们。是表的末
          233     /尾吗?
          JZ      /是的,然后判断在这一轮
          DONE    /排序过程中是否发生
          0       /了交换。
          CMPM    /比较第一个字符串和第二个字
          JC      /符串。第一个比第二个大,
          EXCH    /所以,交换它们。
          0
          JNZ     /它们也不相等,则
          SECBIG  /第二个一定比第一个大,
          0       /所以不交换。
          INXH    /两字符串中的字符相等,
          INXD    /检查下面两个连续的
          JMP     /字符。
          CMPNXT
```

```
          0
DONE,     MOVAC   /取交换指示器内容到 A。
          ORAA    /发生任何交换了吗?
          RZ      /没有，表已排序好。
          JMP     /未排序好，
          ABSORT  /再试。
          0
EXCH,     MVIC    /交换指示器
          377     /置为 377。
          LHLD    /第一个 ASCII 字符串
          FIRST   /的地址装入寄存器对 H。
          0
          XCHG    /这个地址放入 D 和 E。
          LXIH    /把一个可暂存字符串的读/写存
          TMP     /储器地址装入
          0       /寄存器对 H。
          CALL    /D 和 E 的内容传送到 H 和 L。
          MOVE    /返回时 D 和 E 指向第二个
          0       /字符串。
          LHLD    /第一个字符串的地址
          FIRST   /装入寄存器对 H。
          0
          CALL    /现在，把 D 和 E 的内容传送到 H
          MOVE    /和 L，把第二个字符串传送到
          0       /原来存储第一个字符串的读/写存
          SHLD    /储器区。保存 H 和 L，作为新
          SECOND  /的第二个字符串的存储器地址。
          0
          LXID    /H 和 L 指向第二个字符串的地
          TMP     /址，所以把暂存第一个字符串
```

```
          0          /的存储器地址装入 D 和 E。
          CALL       /现在，把存在 TMP 的第一个字
          MOVE       /符串送到原来存第二个字符串
          0          /的地址。
SECBIG,   LHLD       /然后，把新的第二个字符串
          SECOND     /的存储器地址装入 H 和 L。
          0
          JMP        /比较这两个字符串。
          NEXT
          0
MOVE,     LDAXD      /按 D、E 内的地址从存储器取
          MOVMA      /一个数。把它存入(H 和 L 指示
          INXH       /的)存储单元。H 和 L 内的地址
          INXD       /加 1。D 和 E 内的地址加 1。
          CPI        /刚刚传送的是字符串
          000        /的结束符吗?
          JNZ        /否，传送另一个字符。
          MOVE
          0
          RET        /传送了结束符，返回。
```

8080 如果既没有找到 0，也没有找到 233(9 B)，就把寄存器对H寻址的第一个字符串中的字符，与按寄存器对D内的地址从存储器读入A寄存器的第二个字符串的一个字符进行比较。如果第一个字符串中的字符的 ASCII 值大于第二个字符串中同一个"位置"上的字符，这两个字符串就必须交换位置。紧跟在 CMPM 指令之后的 JC 和 EXCH 将完成这种交换。如果第一个字符串中的字符比第二个字符串的字符小，那么，这两个字符串排列的顺序是正确的。如果第一个字符串是 SMITH

而第二个字符串是SMYTHE，两者比较就会出现这种情况，因为字符I的ASCII值小于字符Y的ASCII值。8080转移到SECBIG后，就可以开始对表中第二个字符串和第三个字符串进行比较。

如果两字符串中的字符相等，例如，SMYTHE（第一个字符串）中的S与SMITH（第二个字符串）中的S相比较时，8080只是把寄存器对H和D中的地址加1，然后返回到CMPNXT。当比较SMITH和SMYTHE时，这一指令序列将执行两次：一次是比较两个字母S，另一次是比较两个M。应注意，只有在出现下面五种情况中的一种时，8080才能从CMPNXT循环退出。这五种情况是：(1) 找到了第二个字符串的结束符零，但未找到第一个字符串的结束符(须交换位置)；(2) 同时找到第一个字符串和第二个字符串的结束符(不交换)；(3) 找到表末的233(9 B)(不交换)；(4) 发现第一个字符串中的一个字符比第二个字符串中同一位置上的字符大(须交换)；(5) 第一个字符串的一个字符比第二个字符串同一位置上的字符小（不交换)。

子程序的EXCH指令序列实际上完成交换操作。它有四个基本任务。首先把377(FF)装入C寄存器，表示发生了一次交换（虽然当C寄存器被置位时，实际上还没有执行具体的交换操作)。当8080到达表末时，它将检查C寄存器的内容，确定表是否已排好序。377(FF)装入C寄存器之后，8080把第一个字符串(该字符串由寄存器对H寻址）传送到从TMP开始的一串读/写存储单元。接着，把第二个字符串装入表内原来用于存储第一个字符串的存储区。最后，8080把第一个字符串从TMP送回表内，紧接着刚送入的第二个字符串之后。现在，第一个和第二个字符串的位置颠倒过来了。第二个字符串处于

原来第一个字符串的位置上，而第一个字符串处于原来被第二个字符串所占用的位置上。

两字符串交换位置之后，新的第二个字符串的存储器地址装入寄存器对H。然后，8080转移到NEXT，以使比较表中的第二和第三个字符串。如果是对表6-4内的名字排序，那么当SMYTHE的Y与SMITH的I相比较时，8080将执行ABSORT子程序的EXCH程序段。8080交换这两个字符串的位置之后，接着对SMYTHE和JONES进行比较。后两个字符串也要交换位置。事实上，将把SMYTHE将上推到表顶。

我们在ABSORT子程序(例6-4)中使用了符号地址FIRST和SECOND。读者不应被它们弄糊涂。当第一次调用这个子程序时，FIRST用来存第一个字符串的始地址，SECOND用来存第二个字符串的始地址。但是，如果要对一个包含50个字符串的表排序，那么在某一时刻，FIRST可用来存第23、32或41个字符串的始地址，而SECOND将用来存第24、33或42个字符串的始地址。

字符串排序(ABSORT)子程序的具体说明

可以编写一个实验程序，用它测试ABSORT子程序，测试程序如例6-5所示。执行这个程序，就可以把ASCII字符串送入微型计算机的读/写存储器。所有字符串都送入之后，可对它们排序，然后，在电传打字机或CRT上打印或显示出排好序的表。

在例6-5中，首先将读/写存储器的一个地址装入寄存器对H，然后由电传打字机打印或用CRT显示回车和换行符。当8080执行LXIH指令时，它把存字符串的读/写存储器地址装入寄存器对H。在CHARIN，8080调用TTYIN子程序，以

便从电传打字机或 CRT 显示器输入一个 ASCII 字符到微型计算机。当一只键被按下时，相应的字符也被打印出来。8080 从 TTYIN 子程序返回时，七位的 ASCII 字符存在A寄存器中。如果 RETURN(返回)键被按下，或者产生了一个 CTRL/C，则 8080 执行一些特定的操作。否则，它将把 ASCII 字符存入寄存器对H寻址的读/写存储单元。然后，存储器地址加 1，8080 执行 JMP-CHARIN 指令，以输入另一个 ASCII 字符。

例 6-5 字母数字串排序子程序（ABSORT）的实验程序

```
START,   LXISP    /读/写存储器的一个
         STACK    /地址装入堆栈指示器。
         0
         CALL     /在电传打字机上打印或 CRT
         CRLF     /显示一个换行符和一个回车符。
         0
         LXIH     /表的始地址(读/写存储器地址)
         LIST     /装入寄存器对H。
         0
CHARIN,  CALL     /从电传打字机或 CRT 取一个字符。
         TTYIN
         0
         CPI      /输入了一个 CTRL/C 吗?
         003
         JZ       /是的，则把 233(9 B)存。
         SORT     /在表示，并对表排序。
         0
         CPI       /没有输入 CTRL/C,
         015       /是回车符吗?
         JZ        /是的，则把字符串结束符
         ENDLN     / 0 存在这个字符串的末尾。
```

```
          0
          MOVMA     /既不是回行，也不是 CTRL/C，则
          INXH      /把这个字符存入存储器，然后使
          JMP       /表地址加 1。输入另一个 ASCII 字
          CHARIN    /符。
          0
ENDLN,    MVIM      /输入了回车符，把 0 存入存储器
          000       /中该字符串之后。
          INXH      /表地址加 1。
          CALL      /字符串之后打印一个回车符
          CRLF      /和一个换行符。
          0
          JMP       /然后，取另一个 ASCII 字符。
          CHARIN
          0
SORT,     MVIM      /输入了一个 CTRL/C，所以
          233       /用 233(9 B)结束这个表。
          CALL      /现在，对存在表内的
          ABSORT    /ASCII 字符串排序。
          0
          LXIH      /表已排好顺序，把表的
          LIST      /起始地址装入寄存器对 H。
          0
LINE,     CALL      /首先，打印一个回车符
          CRLF      /和一个换行符。
          0
PRINT,    MOVAM     /从表内取一个字符到 A。
          CPI       /它是字符串的结束符吗？
          000
          JZ        /是的，则在电传打字机打印
```

```
        ENDS    /或 CRT 显示字符串之后打印或
        0       /显示一个回车符和一个换行符。
        CPI     /不是字符串结束符，是
        233     /表的结束符吗？
        JNZ     /不是，则打印这个字符。
        NOTEND
        0
        HLT     /找到了表的结束符，停止。
NOTEND, CALL    /不是字符串或表的结束符，
        TTYOUT  /则在电传打字机上打印或在 CRT
        0       /上显示这个字符。
        INXH    /表地址加 1。
        JMP     /从表取另一个字符，
        PRINT   /并检验它的值。
        0
ENDS,   CALL    /找到字符串末尾的 0 。
        CRLF    /打印回车符和换行。
        0
        INXH    /寄存器对H加 1(越过 0)。
        JMP     /从表取另一个字符。
        PRINT
        0
CRLF,   MVIA    /回车符的 ASCII 值
        215     /装入 A 寄存器。
        CALL    /然后，在电传打字机上打印
        TTYOUT  /或在 CRT 上显示这个字符。
        0
        MVIA    /换行字符的 ASCII 值
        212     /装入 A 寄存器。
        JMP     /然后在电传打字机上打印
```

```
          TTYOUT    /或 CRT 显示这个字符。
          0
TTYIN,    IN        /输入 UART(通过异步收发器)的标识
          001       /字。
          ANI       /只把接收器的标识位
          001       /存入 A 寄存器。
          JZ        /该标识位等于 0，则
          TTYIN     /等待它变为逻辑 1，
          0
          IN        /标识位等于逻辑 1，
          000       /输入 ASCII 字符。
          ANI       /把校验位置为逻辑 0
          177       /(D 7 位)
TTYOUT,   PUSHPSW   /ASCII 字符存入堆栈。
TTY 1,    IN        /输入 UART 的标识字。
          001
          ANI       /只把发送器的标识位
          004       /存入 A 寄存器。
          JZ        /标识位是逻辑 0，
          TTY 1     /等待它成为逻辑 1。
          0
          POPPSW    /标识位是逻辑 1，把
          OUT       /ASCII 字符取到寄存器 A，
          000       /然后输出给 UART。
          RET
```

用 RETURN 和 CTRL/C 的 ASCII 值作为结束符 的值。如果收到了 RETURN 的值，那么 8080 把前一个 RETURN 之后输入的所有 ASCII 字符，看作为一个字符串。因此，当输入 RETURN 值时，8080 转移到 ENDLN， 把零存入存储器（在

字符串最后一个字符之后)。这就是说，一个字符串由一行ASCII字符组成。用CTRL/C向微型计算机指明，所有字符串都输入到了表中。这意味着，在打入表的最后一个字符串之后，其后必须是RETURN，整个表的最后则必须是CTRL/C。CTRL/C将使233 (9 B)存在最后一个ASCII字符串的结束符0的后面。之所以必须在存储器中存0和233 (9 B)，仅仅是因为ABSORT子程序要求这样的表格式。输入CTRL/C和在表末存入233(9 B)之后，8080调用ABSORT子程序，对表进行排序(按字母顺序排列)。

例 6-6 实验程序用于某些样本字符串

(A) 电传打字机输入的未排序字符串

```
DAVE
NANCY
CHRIS
JON
JANE
SARA
LISA
DAVID
BETH
TYCHON, INC.
HOWARD  W. SAMS
```

(B) 电传打字机打印的排了序的字符串

```
BETH
CHRIS
DAVE
DAVID
HOWARD  W. SAMS
```

JANE
JON
LISA
NANCY
SARA
TYCHON, INC

8080 从 ABSORT 子程序返回时，表已排好序，所以8080把表的排好序的内容打印在电传打字机上或者显示在CRT上。字符串从存储器读出时 (LINE)，就在电传打字机上打印或者显示在 CRT 上。8080 一面从存储器读入表的 ASCII，一面检查它是否为 233(9 B) 和 0，如果从存储器读出了 0，则表示整个 ASCII 字符串已打印。因此，8080 调用 CRLF 子程序，在电传打字机或 CRT 上打印或显示一个回车符和一个 换 行符。如果从存贮器读出 233(9 B)，则 8080 停机，因为它已经到达表的末尾，而且所有的字符串都已打印。例 6-6 包含了我们输入微型计算机的一些字符串和打印出来的排序结果。

如果用 ABSORT 子程序对包含字母和数字的字符串排序，那会发生什么情况呢？假设一个表包含 AAA 和 AA 1 两个字符串，排好顺序后，哪个字符串在表的前面，哪个字符串在后面？AA 1 字符串将居前，AAA 其次。这就是说，ABSORT 子程序可以用来给字母和数字串 (ASCII 字母和数字)排序。因为用 0 和 233(9 B) 作为结束符，ABSORT 子程序还可以用来对包含标点符号的字符串排序！

第七章 查 表

在«8080/8085 软件设计» 上册的第五章，我们介绍了许多数学运算子程序。我们曾提到，用这些子程序可以把数据值从一个**数据域**转换到另一个**数据域**。这就是说，用这个子程序可使华氏温度转换成摄氏温度，或者把以英尺表示的长度转换成以厘米表示的长度。众所周知，华氏温度和摄氏温度的关系如下：

$$°C=\frac{5}{9}\times(°F-32)\text{ 或 }°F=\frac{9}{5}\times°C+32$$

可以用乘法、除法、减法和加法子程序来完成这种换算。也可以用查表的方法把一个数据域内的数(例如 °F) 转换成另一个数据域内的数(例如°C)。

所要查的表，它不过是存储在连续存储单元内的节点的集合。这里我们用了“节点”这一术语，是因为表可以包括数据和(或)地址。为用查表的方法把华氏温度转换成摄氏温度，华氏温度应作为地址的一个部分，用来对存有相应摄氏温度的存储单元的寻址。具体地说，为了找到表内存有正确摄氏温度的存储单元，应该把华氏温度加到对照表的**基地址**或始地址上。基地址就是用来存表的第一个节点(或其一部份)的第一个存储单元的地址。温度与基地址相加的结果将产生一个适当的地址。当然，整个过程假设基地址是最低的存储器地址，表的其余部分按顺序存在存储器的较高地址单元。

所要查的表也可用来解决许多软件方面的问题。汇编程序、

BASIC 解释程序和 BASIC 编译程序都可以用它。例如，汇编程序可用对照表确定 JMP 指令的操作码。该指令用存在连续存储单元的 ASCII 字符 J、M 和 P 表示。对于一个汇编程序来说，这个表叫做**符号表**，因为它包含许多符号(实际上是 ASCII 字符串或助记符)的值(操作码)，BASIC 解释程序可利用对照表达到这样一个目的，即当它在用户程序(用 BASIC 语言写成)中碰到 LET、FOR、IF、DIM 或 READ 语句时，就执行相应的指令序列。对照表还有一些其它用途，汇总于表 7-1。在本章的其余部分，我们将描述对照表的结构、应用及其"软件驱动器"。

表 7-1　对照表的可能应用

1. 磅转换成公斤，英里转换成公里，或二进制转换成二-十进制。
2. 汇编程序、解释程序或编译程序字符串的转换。
3. 二-十进制数转换成七段的条型码，供发光二极管显示器显示用。
4. ASCII码转换成莫尔斯电码，或相反。
5. 信息加密和解密。
6. 查找电子锁不同的有效代码。
7. 将 SSAN 与人名联系起来。
8. 计算一个角度的正弦值或者一个数的对数。

在表 7-1 所列的对照表应用中，有一种并不是非常实用的。它是哪一种呢？一般说来，由英磅到公斤，由英里到公里，或者由二进制到二一十进制的转换不用查表法。其原因是在这些情况下，可利用简单的公式，把一种测量单位(数据域)转换成另一种测量单位(数据域)。例如：1 磅＝0.454 公斤，1英里＝1.6 公里。另一方面，要把 ASCII 字符转换成相应莫尔斯电码的点和划，并没有简单的公式可供使用。我们从《8080/8085 软件设计》上册第五章知道，用下面的公式可以逼近一个

角度的正弦值：

$$\sin(x) = x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!} + \frac{x^9}{9!}$$

然而，完成这些计算可能需要一两毫秒的时间。如果采用查表法，在 114μs 之内我们可以求得 0°～360°之间任何一个角的正弦值(角度只能是整数值)。

为什么采用查表法呢？查表法一般只用于两种情况：(1) 两个数据域之间不存在简单的关系；(2) 要以尽可能快的速度进行转换。可能的转换问题包括：ASCII 字符转换成莫尔斯电码字符，角度转换成正弦值，或者二—十进制数转换成七段发光二极管显示器的条型码。

为了确定一个角度的正弦值，可以采用查表方法。事实上，甚至可以在集成电路出售商店买到正弦值对照表！这个表装在只读存储器(ROM)中，并可以象任何其它存储器电路片那样连接到微型计算机系统。国家半导体公司 MM 5220 BM 只读存储中的对照表内容如图 7-1 所示。表中每一个正弦值的"地址"，实际上是用弧度或度表示的角度。每个存储单元的内容是这个角度的正弦值。请注意，图 7-1 所列出的地址用十进制表示，但很容易转换成八进制数或十六进制数。ROM 输出的 B_1 是正弦值的最高有效位(MSB)，B_8 是正弦值的最低有效位(LSB)。还应记住，0°和 90°之间任何一个角的正弦值在 0 和 1 之间。所以，输出各位的有效值为：$B_1=1/2$，$B_2=1/4$，$B_3=1/8$，$B_4=1/16$，$B_5=1/32$，$B_6=1/64$，$B_7=1/128$，$B_8=1/256$。

用这种器件的困难之一是，0°到 90°之间角度的正弦值有 128 个，因此，每一个存储单元表示 0.703°的角度变化。这并不是特别容易处理的倍数。为了用这种表确定一个角度的正弦值，可用下式算出只读存储器(ROM)的适当地址：

$$\left(\frac{X}{90_{10}}\right)\times 128_{10}=\text{存储地址}$$

这就是说，若 X＝45°，存储器地址是 64_{10}；若 X＝30°，存储器地址是 43_{10}。求得适当的地址，就可以确定角的正弦值。这种计算或许要用乘和除，但是，30°角或者71°角的正弦值所用时间仍很少。

为了简化确定一个角的正弦值的过程，较容易的方法是让对照表内的每一个地址表示 1°而不是 0.703°的增量。为产生这种新的表，我们用 BASIC 语言编写了一个程序。由这个程序产生的正弦表如表 7-2 所示。表中二进制值的最高有效位是0。这样做的目的在于，8080 从表读出正弦值之后，可在其中加进一个符号位。因此，表内八位字的格式(符号和幅值，非 2 的补码)如下所示：

S.×××××××

八位字的最高有效位(S)用作七位正弦值的符号位。×××××××代表一个角的七位正弦值。因为二进制小数点位于 D_7 位和 D_6位之间，D_6 位的有效值是 1/2，D_5＝1/4，D_4＝1/8，D_3＝1/16，D_2＝1/32，D_1＝1/64，D_0＝1/128。如果把这些七位正弦值与 ROM MM 5220 BM 正弦对照表 ROM (图 7-1)中的八位正弦值进行比较，可以看到，它们是很一致的。请记住，ROM中的角差为 0.703°，计算机产生的表(表 7-2) 内的角差是 1°。

用前面的格式，30°角的正弦应怎样表示呢？它的符号是正的，值是 0.5。所以应是 01000000。90°角的正弦是 1，符号位是正的，所以，计算机产生的值应是 01111111。181°角的正弦是负值，但其绝对值与 1°角的正弦值相等，即等于 0.0175 或 00000010(1/64 或 0.015625)。所以，181°角的正弦

是10000010。请记住，处理小数二进制数时，可能得不到特别精确的结果。如果使用表7-2，就不能要求角的正弦的精度高于1/128或0.0078125(精度为0.78125%)。这就是说，正弦值精度最高为±0.78125%。这与对1°角的正弦所得到的值是一致的(0.015625+0.0078125=0.0234375)。所以，1°角的正弦精确到±0.78125%。

当然，必须执行一些软件指令，才能存取正弦表中的数据值。最便于使用的方法或许是把二进制数表示的角度装入A寄存器，然后调用一个子程序。在调用例7-1中的SINANG子程序时正是这样做的。

在例7-1，8080首先把A寄存器中的角度值与立即数据字节133(十进制91)进行比较。如果A寄存器的值比91大或者相等，8080带着置0的进位标识从子程序返回。用户也许想在调用SINANG子程序之后立即放一条JNC-ERROR指令，其中ERROR是一个子程序，它告诉用户，微型计算机企图确定大于90°角的正弦。如果A寄存器的内容是在0°和90°之间，8080不执行RNC指令，而是把寄存器对H和B的内容压入堆栈。

LXIH指令把正弦表的基地址装入寄存器对H，MVIB指令把O装入寄存器对B的高位字节。然后，A寄存器内的角度传送到寄存器对B的低位字节，即C寄存器。8080把寄存器

例 7-1 利用正弦表计算0°和90°之间的任何一个角的正弦

```
/这个子程序计算A寄存器内二进制
/角度的正弦值。
/角度必须在0°和90°之间。如果不是这样，
/则当8080从子程序返回时，进位位是逻辑0。
```

```
/如果角度是"有效的"，则当 8080 从子程序返回
/时，角的正弦存在A寄存器。


SINANG, CPI/角与 91°的角比较
        133     /(八进制 133 =十进制 91)。
        RNC     /角太大，返回。
       PUSHH    /寄存器对H的内容压入堆栈。
       PUSHB    /寄存器对B的内容压入堆栈。
       LXIH     /正弦表的基地址。
       SINTAB   /(始地址)装入
       0        /寄存器对H。
        MVIB    /寄存器对B的高位
       000      /字节置 0 。
       MOVCA    /A寄存器内的角度值传送到寄存
       DADB     /器对B的低位字节，并加到基
      MOVAM     /地址上。从正弦表取这个角的
       POPB     /正弦。从堆栈弹入寄存器对B。
       POPH     /从堆栈弹入寄存器对H。
       STC      /使进位标识位置 1。
       RET
SINTAB, 000     /这是 0°角的正弦。
          .     /正弦表的其余部分
          .     /存在这里。
          .
```

对B内的角度加到寄存器对H内的正弦表的基地址上。相加得到的地址留在寄存器对H。然后，从正弦表读出这个角的正弦，送A寄存器。接着，堆栈内保存的值弹入寄存器对B和H，

参考地址	功能输入		代码输出							
	度	弧度	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
0	0.00	0.000	0	0	0	0	0	0	0	0
1	0.70	0.012	1	1	0	0	0	0	0	0
2	1.41	0.025	0	1	1	0	0	0	0	0
3	2.11	0.037	1	0	0	1	0	0	0	0
4	2.81	0.049	0	0	1	1	0	0	0	0
5	3.52	0.061	1	1	1	1	0	0	0	0
6	4.22	0.074	1	1	0	0	1	0	0	0
7	4.92	0.086	0	1	1	0	1	0	0	0
8	5.63	0.098	1	0	0	1	1	0	0	0
9	6.33	0.110	0	0	1	1	1	0	0	0
10	7.03	0.123	1	1	1	1	1	0	0	0
11	7.73	0.135	0	1	0	0	0	1	0	0
12	8.44	0.147	1	0	1	0	0	1	0	0
13	9.14	0.160	0	0	0	1	0	1	0	0
14	9.84	0.172	0	0	1	1	0	1	0	0
15	10.55	0.184	1	1	1	1	0	1	0	0
16	11.25	0.196	0	1	0	0	1	1	0	0
17	11.95	0.209	1	0	1	0	1	1	0	0
18	12.66	0.221	0	0	0	1	1	1	0	0
19	13.36	0.233	1	1	0	1	1	1	0	0
20	14.06	0.245	0	1	1	1	1	1	0	0
21	14.77	0.258	1	0	0	0	0	0	1	0
22	15.47	0.270	0	0	1	0	0	0	1	0
23	16.17	0.282	1	1	1	0	0	0	1	0
24	16.88	0.295	0	1	0	1	0	0	1	0
25	17.58	0.307	1	0	1	1	0	0	1	0
26	18.28	0.319	0	0	0	0	1	0	1	0
27	18.98	0.331	1	1	0	0	1	0	1	0
28	19.69	0.344	0	1	1	0	1	0	1	0
29	20.39	0.356	1	0	0	1	1	0	1	0

续表

参考地址	功能输入		代码输出							
	度	弧度	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
30	21.09	0.368	0	0	1	1	1	0	1	0
31	21.80	0.380	1	1	1	1	1	0	1	0
32	22.50	0.393	0	1	0	0	0	1	1	0
33	23.20	0.405	1	1	1	0	0	1	1	0
34	23.91	0.417	1	1	1	0	0	1	1	0
35	24.61	0.430	0	1	0	1	0	1	1	0
36	25.31	0.442	1	0	1	1	0	1	1	0
37	26.02	0.454	0	0	0	0	1	1	1	0
38	26.72	0.466	1	1	0	0	1	1	1	0
39	27.42	0.479	0	1	1	0	1	1	1	0
40	28.13	0.491	0	0	0	1	1	1	1	0
41	28.83	0.503	1	1	0	1	1	1	1	0
42	29.53	0.515	0	1	1	1	1	1	1	0
43	30.23	0.528	0	0	0	0	0	0	0	1
44	30.94	0.540	1	1	0	0	0	0	0	1
45	31.64	0.552	0	1	1	0	0	0	0	1
46	32.34	0.565	1	0	0	1	0	0	0	1
47	33.05	0.577	1	1	0	1	0	0	0	1
48	33.75	0.589	0	1	1	1	0	0	0	1
49	34.45	0.601	1	0	0	0	1	0	0	1
50	35.16	0.614	1	1	0	0	1	0	0	1
51	35.86	0.626	0	1	1	0	1	0	0	1
52	36.56	0.638	0	0	0	1	1	0	0	1
53	37.27	0.650	1	1	0	1	1	0	0	1
54	37.97	0.663	1	0	1	1	1	0	0	1
55	38.67	0.675	0	0	0	0	0	1	0	1
56	39.37	0.687	0	1	0	0	0	1	0	1
57	40.08	0.699	1	0	1	0	0	1	0	1
58	40.78	0.712	1	1	1	0	0	1	0	1

续表

参考地址	功能 输入		代码 输出							
	度	弧度	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
59	41.48	0.724	1	0	0	1	0	1	0	1
60	42.19	0.736	0	0	1	1	0	1	0	1
61	42.89	0.749	0	1	1	1	0	1	0	1
62	43.59	0.761	0	0	0	0	1	1	0	1
63	44.30	0.773	1	1	0	0	1	1	0	1
64	45.00	0.785	1	0	1	0	1	1	0	1
65	45.70	0.798	1	1	1	0	1	1	0	1
66	46.41	0.810	1	0	0	1	1	1	0	1
67	47.11	0.822	1	1	0	1	1	1	0	1
68	47.81	0.834	1	0	1	1	1	1	0	1
69	48.52	0.847	0	0	0	0	0	0	1	1
70	49.22	0.859	0	1	0	0	0	0	1	1
71	49.92	0.871	0	0	1	0	0	0	1	1
72	50.62	0.884	0	1	1	0	0	0	1	1
73	51.33	0.896	0	0	0	1	0	0	1	1
74	52.03	0.908	0	1	0	1	0	0	1	1
75	52.73	0.920	1	1	0	1	0	0	1	1
76	53.44	0.933	1	0	1	1	0	0	1	1
77	54.14	0.945	1	1	1	1	0	0	1	1
78	54.84	0.957	1	0	0	0	1	0	1	1
79	55.55	0.969	1	1	0	0	1	0	1	1
80	56.25	0.982	1	0	1	0	1	0	1	1
81	56.95	0.994	0	1	1	0	1	0	1	1
82	57.66	1.006	0	0	0	1	1	0	1	1
83	58.36	1.019	0	1	0	1	1	0	1	1
84	59.06	1.031	1	1	0	1	1	0	1	1
85	59.77	1.043	1	0	1	1	1	0	1	1
86	60.47	1.055	0	1	1	1	1	0	1	1
87	61.17	1.068	0	0	0	0	0	1	1	1

续表

参考地址	功能输入		代码输出							
	度	弧度	B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
88	61.87	1.080	0	1	0	0	0	1	1	1
89	62.58	1.092	1	1	0	0	0	1	1	1
90	63.28	1.104	0	0	1	0	0	1	1	1
91	63.98	1.117	0	1	1	0	0	1	1	1
92	64.69	1.129	1	1	1	0	0	1	1	1
93	65.39	1.141	0	0	0	1	0	1	1	1
94	66.09	1.154	0	1	0	1	0	1	1	1
95	66.80	1.166	1	1	0	1	0	1	1	1
96	67.50	1.178	0	0	1	1	0	1	1	1
97	68.20	1.190	1	0	1	1	0	1	1	1
98	68.91	1.203	1	1	1	1	0	1	1	1
99	69.61	1.215	0	0	0	0	1	1	1	1
100	70.31	1.227	1	0	0	0	1	1	1	1
101	71.02	1.239	0	1	0	0	1	1	1	1
102	71.72	1.252	1	1	0	0	1	1	1	1
103	72.42	1.264	0	0	1	0	1	1	1	1
104	73.12	1.276	1	0	1	0	1	1	1	1
105	73.83	1.289	0	1	1	0	1	1	1	1
106	74.53	1.301	0	1	1	0	1	1	1	1
107	75.23	1.313	1	1	1	0	1	1	1	1
108	75.94	1.325	0	0	0	1	1	1	1	1
109	76.64	1.338	1	0	0	1	1	1	1	1
110	77.34	1.350	0	1	0	1	1	1	1	1
111	78.05	1.362	0	1	0	1	1	1	1	1
112	78.75	1.374	1	1	0	1	1	1	1	1
113	79.45	1.387	1	1	0	1	1	1	1	1
114	80.16	1.399	0	0	1	1	1	1	1	1
115	80.86	1.411	0	0	1	1	1	1	1	1
116	81.56	1.424	1	0	1	1	1	1	1	1

续表

参考地址	功能 输入 度	功能 输入 弧度	代码 输出 B_8	B_7	B_6	B_5	B_4	B_3	B_2	B_1
117	82.27	1.436	1	0	1	1	1	1	1	1
118	82.97	1.448	0	1	1	1	1	1	1	1
119	83.67	1.460	0	1	1	1	1	1	1	1
120	84.38	1.473	1	1	1	1	1	1	1	1
121	85.08	1.485	1	1	1	1	1	1	1	1
122	85.78	1.497	1	1	1	1	1	1	1	1
123	86.48	1.509	1	1	1	1	1	1	1	1
124	87.19	1.522	1	1	1	1	1	1	1	1
125	87.89	1.534	1	1	1	1	1	1	1	1
126	88.59	1.546	1	1	1	1	1	1	1	1
127	89.30	1.559	1	1	1	1	1	1	1	1

图 7-1 存储在（国家半导体公司）MM 5220 BM 的 ROM 中的正弦表

表 7-2 角度增量为1°的正弦表

角度	正弦 十进制	正弦 二进制	角度	正弦 十进制	正弦 二进制
0.00	0.0000	00000000	45.00	0.7071	01011010
1.00	0.0175	00000010	46.00	0.7193	01011100
2.00	0.0349	00000100	47.00	0.7313	01011101
3.00	0.0523	00000110	48.00	0.7431	01011111
4.00	0.0698	00001000	49.00	0.7547	01100000
5.00	0.0872	00001011	50.00	0.7660	01100010
6.00	0.1045	00001101	51.00	0.7771	01100011
7.00	0.1219	00001111	52.00	0.7880	01100100
8.00	0.1392	00010001	53.00	0.7986	01100110
9.00	0.1564	00010100	54.00	0.8090	01100111
10.00	0.1736	00010110	55.00	0.8191	01101000

续表

角 度	正 弦		角 度	正 弦	
	十 进 制	二 进 制		十 进 制	二 进 制
11.00	0.1908	00011000	56.00	0.8290	01101010
12.00	0.2079	00011010	57.00	0.8387	01101011
13.00	0.2250	00011100	58.00	0.8480	01101100
14.00	0.2419	00011110	59.00	0.8572	01101101
15.00	0.2588	00100001	60.00	0.8660	01101110
16.00	0.2756	00100011	61.00	0.8746	01101111
17.00	0.2924	00100101	62.00	0.8829	01110001
18.00	0.3090	00100111	63.00	0.8910	01110010
19.00	0.3256	00101001	64.00	0.8988	01110011
20.00	0.3420	00101011	65.00	0.9063	01110100
21.00	0.3584	00101101	66.00	0.9135	01110100
22.00	0.3746	00101111	67.00	0.9205	01110101
23.00	0.3907	00110010	68.00	0.9272	01110110
24.00	0.4067	00110100	69.00	0.9336	01110111
25.00	0.4226	00110110	70.00	0.9397	01111000
26.00	0.4384	00111000	71.00	0.9455	01111001
27.00	0.4540	00111010	72.00	0.9511	01111001
28.00	0.4695	00111100	73.00	0.9563	01111010
29.00	0.4848	00111110	74.00	0.9613	01111011
30.00	0.5000	01000000	75.00	0.9659	01111011
31.00	0.5150	01000001	76.00	0.9703	01111100
32.00	0.5299	01000011	77.00	0.9744	01111100
33.00	0.5446	01000101	78.00	0.9781	01111101
34.00	0.5592	01000111	79.00	0.9816	01111101
35.00	0.5736	01001001	80.00	0.9848	01111110
36.00	0.5878	01001011	81.00	0.9877	01111110
37.00	0.6018	01001101	82.00	0.9903	01111110
38.00	0.6157	01001110	83.00	0.9926	01111111
39.00	0.6293	01010000	84.00	0.9945	01111111
40.00	0.6428	01010010	85.00	0.9962	01111111

续表

角 度	正 弦		角 度	正 弦	
	十 进 制	二 进 制		十 进 制	二 进 制
41.00	0.6561	01010011	86.00	0.9976	01111111
42.00	0.6691	01010101	87.00	0.9986	01111111
43.00	0.6820	01010111	88.00	0.9994	01111111
44.00	0.6947	01011000	89.00	0.9998	01111111
45.00	0.7071	01011010	90.00	1.0000	01111111

STC 指令把进位位置为逻辑 1。执行这条指令的目的在于，使 8080 从 SINANG 子程序返回时不执行 JNC-ERROR指令。当然，在存储器中JNC-ERROR 指令不必存在调用SINANG的指令之后，8080 从 SINANG 返回时，角的正弦值在A寄存器中。

可以看到，8080 产生存储器地址，其方法是将角度值加到正弦表的基地址上。然后就可以按寄存器对H内的地址从存储单元读这个角的正弦，把正弦值送到 8080 的一个通用寄存器。那么，正弦表基地址所寻址的存储单元内应存什么呢？应把 0° 角的正弦存入其中。因为 0°角意味着，调用 SINANG 子程序时A寄存器的内容是 0，而当寄存器对 B 与寄存器对 H 内的基地址相加时，寄存器对 B 的内容将是 0。因此，必须把 0° 角的正弦存在存储器内正弦表的基地址上。

要用多少存储单元来存储正弦表呢？需要 90_{10} 个存储单元，因为 SINANG 子程序可用来确定 0°和 90° 之间任何角的正弦。当然，只有在A寄存器值为 0°到 $90°_{10}$ 间的整数值时，才能调用这个子程序。如果正弦表的基地址是 0400 (0400)，那么表的最后一个存储单元地址是什么呢？是 0404

131(0459)。这就是说，正弦表用的存储单元是132(5A)个。什么数存在004 131(0459)这个存储单元呢？90°角的正弦。

SINANG子程序的一个优点，也即一般查表法的一个特征，是查找表中任何二个值的时间是一样长的。这就是说，8080确定32°角的正弦和确定78°角的正弦所需要的时间相同。

假设必须确定131°或222°角的正弦。如果在调用SINANG子程序时，这两个角等效的二进制数存在A寄存器，而且没有预先将A寄存器的内容与133比较，那么，8080会产生比对照表最后一个地址还大的表地址。这是因为这个表只包含0°和90°间的角的正弦。

可以用曲线表示0°和360°之间任一角度的正弦，如图7-2所示。很容易看出，0°和180°之间的角的正弦是正值，大于180°和小于360°的角的正弦是负值。所以，

$0° < X < 180°$，正弦是正值

$180° < X < 360°$，正弦是负值

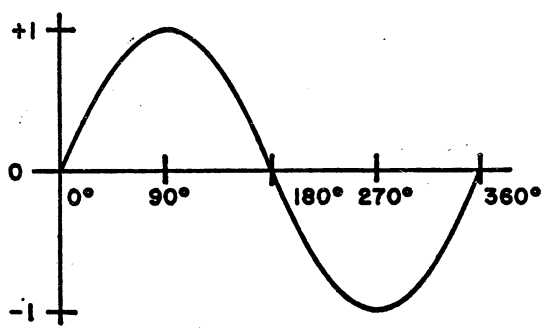


图 7-2　0°和360°间各角的正弦

由图7-2可见，如果不考虑正弦的符号，91°角的正弦与

89° 角的正弦相同，181° 角的正弦与 1° 角的正弦相同。因此，可以得出如下结论。假定角度为 X，则

当 $0° \leqq X \leqq 90°$ 时，$\sin(X)=\sin(X)$

当 $90° \leqq X \leqq 180°$ 时，$\sin(X)=\sin(180°-X)$

或 $\sin(X)=\sin(90°-(X-90°))$

例如：

$$\sin(170°)=\sin(180°-170°)$$
$$\sin(170°)=\sin(10°)$$

或者

$$\sin(170°)=\sin(90°-(170°-90°))$$
$$\sin(170°)=\sin(90°-80°)$$
$$\sin(170°)=\sin(10°)$$

对于同一个角 X，还可以得：

当 $180° \leqq X \leqq 270°$ 时，$\sin(X)=\sin(X-180°)$

当 $270° \leqq X \leqq 360°$ 时，$\sin(X)=\sin(360°-X)$

或 $\sin(X)=\sin(90°-(X-270°))$

例如：

$$\sin(190°)=\sin(190°-180°)$$
$$\sin(190°)=\sin(10°)$$

对于 270° 和 360° 之间的任何一个角如 290°。

$$\sin(290°)=\sin(90°-(290°-270°))$$
$$\sin(290°)=\sin(90°-20°)$$
$$\sin(290°)=\sin(70°)$$

当然，大于 180° 和小于 360° 角的正弦的符号是负的。根据这些结果，可以画一个流程图来说明把 0° 和 360° 间任何角

的正弦归结为 0° 和 90° 间一个角的正弦所需的计算（图 7-3）。

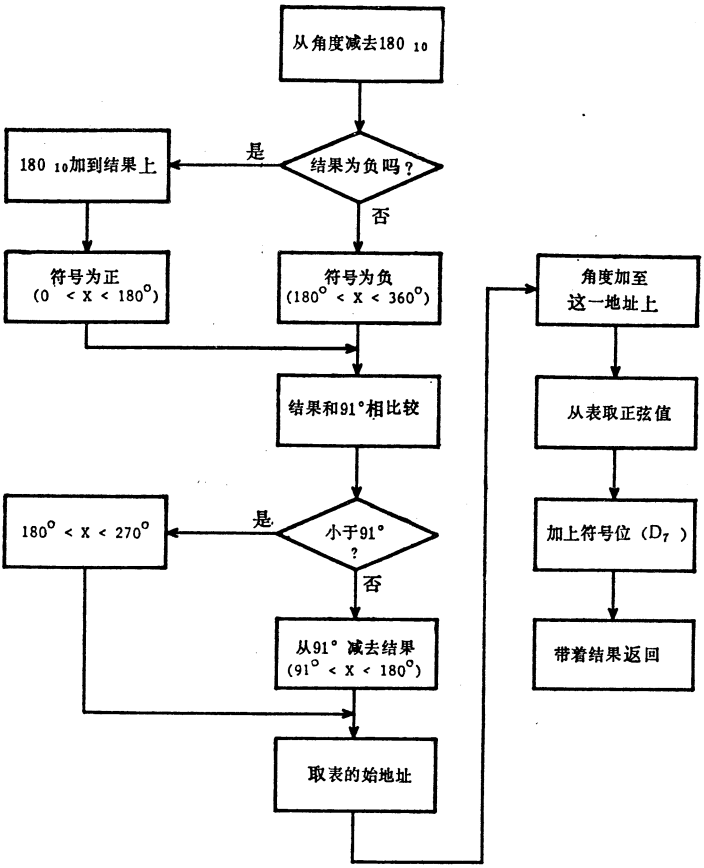


图 7-3　角度-正弦转换子程序的流程图

例 7-2 所列出的子程序可以用来确定 0° 和 360° 之间任何一个角的正弦。它既可确定角的七位正弦值，也可确定它的符号。当 8080 调用 SINANG 子程序时，寄存器对 B 内必须存

有整数二进制角度值(B 寄存器存高字节，C 寄存器存低字节)。角度是 0° 和 360° 间的一个不带符号的数(000000～001 150；0000～0168)。

SINANG 子程序的头两条指令把寄存器对 D 和 H 的内容存入堆栈。然后从寄存器对 B 中的角度减去 180°，以确定正弦的符号。但实际上程序不是采用减指令从寄存器对 B 减去 180°，而是把 180° 的补码与角度相加。所以，377114 (FF 4C)装入寄存器对H，而 DADB 指令完成所要求的加法操作。如果寄存器对 B 中的角介于 0° 和 179° 之间，那么它的正弦的符号是正，8080 执行 JNC-A 0180 指令。如果角大于 180°，则正弦的符号是负，8080 不转移到 A 0180，而是把 B 寄存器的 D_7 位置 1，并转到 NXTSUB。

如果正弦的符号是正，则 8080 执行 JNC-A 0180 指令。如果角在 0° 和 180° 之间，则 B 寄存器的 D_7 位必须置 0。但是，实际上角在 0° 和 180° 之间时，B 寄存器的 D_7 位已为逻辑 0。所以，8080 不必执行任何特殊的操作，来使该位置 0。从 A0180 开始，8080 把寄存器对 B 内的角度压入堆栈，然后，从堆栈中取出这个角度送寄存器对 H。

如果角等于或者大于 181°，则 8080 转移到 NXTSUB，“减”(补码加)的结果留在寄存器对 H。如果寄存器对 B 原来存的角度是 180°，现在，寄存器对 H 内的角度是 1°；如果寄存器对 B 原来含有 271°，那么现在寄存器对 H 内是 91°。总之，不论原来角度的大小(0°～360°)，现在寄存器对 H 内的角是在 0° 和 180° 之间，B 寄存器的 D_7 位含有待找正弦值的符号。

在 NXTSUB，把这个角存入堆栈，然后 90° 的补码装入寄存器对 D。这个值加到寄存器对 H 内的角度值(0°～

180° 之间)上。如果相加的结果，进位位置为逻辑 0，那么，原来的角在 0° 和 90° 之间或者 180° 和 270° 之间。作为例子，注意 30°角的正弦和 270°角的正弦的差别，它们又是符号不同，但值相同，都是 0.5000。因此，8080 转到 OKASIS，把寄存器对 H 的内容加到表的基地址上。这就是说，原来的角在 0° 和 90° 之间，或者说寄存器对 H 的内容是－180° 与 180° 和 270° 之间的一个角相加而得到的。

例 7-2 查表确定 0° 和 360° 之间一个角的正弦

```
/这个程序计算寄存器对 B 内的
/用二进制数表示的角(0° 和 360° 之间)的正弦。
/返回时 A 寄存器内存有带符号的正弦值。

SINANG,  PUSHD    /寄存器对 D 的内容压入堆栈。
         PUSHH    /寄存器对 H 的内容压入堆栈。
         LXIH     /180 的对 2的
         114      /补码(即－180)装入
         377      /寄存器对 H。
         DADB     /寄存器对 B 内的角度值与－180
         JNC      /相加。如果没有进位，则这个角
         A0180    /是在 0° 和 179° 之间。
         0
         MVIB     /这个角等于或者大于 180°，
         200      /则正弦的符号是负的。
         JMP      /B 寄存器存这个符号。
         NXTSUB   /角＋(－180)
         0        /存入寄存器对 H。
A0180,   PUSHB    /符号是正的，取原来
         POPH     /的角(0～180)，送到寄存器对 H。
NXTSUB,  PUSHH    /角度压入堆栈。
```

```
                LXID    /－90 装入
                246     /寄存器对 D。
                377
                DADD    /把这个数加到寄存器对 H 内的
                JNC     /角度值上，如果没有进位，原
                OKASIS  /来的角是在 0° 和 90° 或 180° 和 270°
                0       /之间。
                POPD    /如果角是在 90～180 或 270～360
                MVIA    /之间，则从堆栈弹入 D 和 E。
                132     /从 90 减去角＋(－180)
                SUBL    /的结果。
                MOVLA   /结果存回到 L。
                JMP     /然后，在正弦表中找
                CALCIT  /适当的项。
                0
OKASIS,         POPH    /角取回到 H 和 L。
CALCIT,         LXID    /正弦表的
                SINTAB  /基地址装入
                0       /寄存器对 D。
                DADD    /这个地址加到 H 和 L。
                MOVAM   /角的正弦送到 A 寄存器。
                ADDB    /加上符号(000 或 200)，
                POPH    /从堆栈弹入寄存器对 H 和
                POPD    /寄存器对 D。
                CPI     /结果是负 0 吗？
                200
                RNZ     /不是，则返回。
                XRAA    /是的，则把 A 置为 0。
                RET
SINTAB,         000     /0° 的正弦是 0。
```

```
•          /正弦表的其余部分
•          /存在这儿。
•
```

－90 加到寄存器对 H 的内容上之后，如果进位位是逻辑 1，那么原来的角必定是在 90° 和 180° 或 270° 和 360° 之间。当然，把－180° 加到原来的角上，270° 和 360° 之间的角就能转换成 90° 和 180° 之间的角。因此，如果进位位是逻辑 1，8080 不执行 JNC-OKASIS 指令，而是执行 POPD 指令，从堆栈取角度值送寄存器对 D。然后，90_{10} 装入 A 寄存器。现在，必须从 90° 减去－90 与角度值相加的结果。132(十六进制 5A，十进制 90)装入 A 寄存器之后，从 A 寄存器减去 L 寄存器的内容。这里，我们不必做 16 位的减操作，因为在一个 8 位的寄存器可存放 0° 和 90° 之间的任一角度。

如果 8080 转移到 OKASIS，那么堆栈上的角度值被弹入寄存器对 H。如果刚从 A 寄存器内的 90_{10} 减去 L 寄存器的内容，那么 8080 就跳过这条指令。不论哪一种情况，在 CAL_CIT，8080 总是把正弦表的基地址装入寄存器对 D。然后把它与寄存器对 H 内的角度(0° 和 90° 之间)相加，执行 DADD 指令之后，8080 把七位正弦值从正弦表送到 A 寄存器，再把 B 寄存器内的符号加到这个七位正弦值上。然后，把堆栈内容弹入寄存器对 H 和 D。8080 把 A 寄存器内的带符号的正弦值与立即数据字节 200(80)进行比较。200(80) 等于负零的带符号的正弦值。如果 A 寄存器的内容与这个值不相等，则 8080 返回。如果 A 寄存器的内容与负 0 相等，则 8080 在从子程序返回前把 A 寄存器的内容置成正 0。如果不考虑正 0 和负 0。则可以省去 CPI，RNZ 和 XRAA 指令。如果没有这三条指令，8080 把 180° 的带符号正弦确定为负 0。

使用更精确的正弦表

上面我们讨论了 SINANG 子程序，那么正弦值的精度如何呢？因为角的正弦用七位二进制数(八位二进制数的最高有效位是正弦的符号位)表示，所以，仅精确到 0.78%(1/128)。现假设需要更高的精度，例如 0.003%。怎样才能求出这样高的精度的正弦呢？

最简单的方法理应是把正弦表(SINTAB)从七位改为 15 位。通过修改 BASIC 程序，可使 8080 微型计算机产生 16 位的正弦表，如表 7-3 所示。这些值的最高有效位是 0，所以正弦的符号可加到它上面。

当然，首先必须解决的一个问题是，如何把 16 位值存入存储器。至于每个值的高位字节是先存入存储器还是最后存入存储器，那是无关紧要的，只要 SINANG 子程序能在适当的时候读取高位字节和低位字节就行了。我们假设低位字节跟在高位字节之后存在存储器下一个连续的存储单元(较高的地址上)。1° 和 2° 角的 16 位正弦值象下面那样存在正弦表里：

角		八进制存储器地址		数据	十六进制存储器地址
1°	低字节	015	023	00111011	0D 13
	高字节	015	024	00000010	0D 14
2°	低字节	015	025	01110111	0D 15
	高字节	015	026	00000100	0D 16

015 023(0D13) 这个地址是任选的。一般说来，一旦表的数据格式选定了，所有其它的节点都应以同样的格式存储。因

表 7-3　16位(15位十符号位)的正弦表(精确到0.003%)

角度	正弦		角度	正弦	
	十进制	二进制		十进制	二进制
0.00	0.0000	0000000000000000	45.00	0.7071	0101101010000010
1.00	0.0175	0000001000111011	46.00	0.7193	0101110000010011
2.00	0.0349	0000010001110111	47.00	0.7313	0101110110011100
3.00	0.0523	0000011010110010	48.00	0.7431	0101111100011111
4.00	0.0698	0000100011101101	49.00	0.7547	0110000010011010
5.00	0.0872	0000101100100111	50.00	0.7660	0110001000001101
6.00	0.1045	0000110101100001	51.00	0.7771	0110001101111001
7.00	0.1219	0000111110011001	52.00	0.7880	0110010011011101
8.00	0.1392	0001000111010000	53.00	0.7986	0110011000111001
9.00	0.1564	0001010000000101	54.00	0.8090	0110011110001101
10.00	0.1736	0001011000111010	55.00	0.8191	0110100011011001
11.00	0.1908	0001100001101100	56.00	0.8290	0110101000011101
12.00	0.2079	0001101010011100	57.00	0.8387	0110101101011001
13.00	0.2250	0001110011001011	58.00	0.8480	0110110010001100
14.00	0.2419	0001111011110111	59.00	0.8572	0110110110110111
15.00	0.2588	0010000100100000	60.00	0.8660	0110111011011001
16.00	0.2756	0010001101001000	61.00	0.8746	0110111111110011
17.00	0.2924	0010010101101100	62.00	0.8829	0111000100000100
18.00	0.3090	0010011110001101	63.00	0.8910	0111001000001100
19.00	0.3256	0010100110101100	64.00	0.8988	0111001100001011
20.00	0.3420	0010101111000111	65.00	0.9063	0111010000000001
21.00	0.3584	0010110111011110	66.00	0.9135	0111010011101111
22.00	0.3746	0010111111110011	67.00	0.9205	0111010111010010
23.00	0.3907	0011001000000011	68.00	0.9272	0111011010101101
24.00	0.4067	0011010000001111	69.00	0.9336	0111011101111111
25.00	0.4226	0011011000011000	70.00	0.9397	0111100001000111
26.00	0.4384	0011100000011100	71.00	0.9455	0111100100000110
27.00	0.4540	0011101000011100	72.00	0.9511	0111100110111100
28.00	0.4695	0011110000010111	73.00	0.9563	0111101001101000
29.00	0.4848	0011111000001110	74.00	0.9613	0111101100001010

续表

角　度	正　弦		角　度	正　弦	
	十进制	二　进　制		十进制	二　进　制
30.00	0.5000	0100000000000000	75.00	0.9659	0111101110100011
31.00	0.5150	0100000111101100	76.00	0.9703	0111110000110010
32.00	0.5299	0100001111010100	77.00	0.9744	0111110010111000
33.00	0.5446	0100010110110110	78.00	0.9781	0111110100110011
34.00	0.5592	0100011110010011	79.00	0.9816	0111110110100101
35.00	0.5736	0100100101101010	80.00	0.9848	0111111000001110
36.00	0.5878	0100101100111100	81.00	0.9877	0111111001101100
37.00	0.6018	0100110100001000	82.00	0.9903	0111111011000000
38.00	0.6157	0100111011001101	83.00	0.9926	0111111100001011
39.00	0.6293	0101000010001101	84.00	0.9945	0111111101001100
40.00	0.6428	0101001001000110	85.00	0.9962	0111111110000011
41.00	0.6561	0101001111111001	86.00	0.9976	0111111110110000
42.00	0.6691	0101010110100101	87.00	0.9986	0111111111010010
43.00	0.6820	0101011101001011	88.00	0.9994	0111111111101011
44.00	0.6947	0101100011101010	89.00	0.9998	0111111111111010
45.00	0.7071	0101101010000010	90.00	1.0000	0111111111111111

此，其余各个值的低位字节都必须首先存入存储器的较低地址，紧接着便在下一个较高的连续的存储单元存高位字节。

存储这种新的正弦表需要多少存储单元呢？ $90_{10}\times 2_{10}$ 即 180_{10} 个。应怎样修改SINANG子程序(例 7-2)呢？从 CALCAD 开始，子程序应象例 7-3 所示的那样修改。

例 7-3　修改 SINANG 子程序(例 7-2)，使它处理 16 位的正弦值

```
        .
        .
CALCAD,  LXID     /角存在寄存器对 H 内。
         SINTAB   /正弦表的基地址
```

```
          0          /装入寄存器对 D。
          DADH       /角乘 2。
          DADD       /基地址加到(角 * 2)。
          MOVAM /正弦的低字节传送到 A，再传送到 C
          MOVCA /寄存器。存储器地址加 1。
          INXH       /高、低字节相"或"。
          ORAM       /正弦值为 0，所以不把
          JZ         /符号位加到
          NEGZER /正弦值上。
          0
          MOVAM /正弦的高字节取到A 。
          ADDB       /把符号加到七位的高位字节上。
NEGZER,   MOVBA /正弦的高字节存入 B。
          POPH       /从堆栈弹入寄存器对 H。
          POPD       /从堆栈弹入寄存器对 D。
          RET
```

子程序(例 7-3)的第一部分与前一个程序例子完全相同。正弦的符号位仍用同样的方法进行确定。而且，将用数学方法把 0°和 360°之间的所有角转换成 0°和 90°之间的角。

从 CALCAD 开始，把正弦表的基地址装入寄存器对 D。DADH 指令把 0°和 90° 之间的角度和 2 相乘， 因为存储正弦表中每个节点需要两个存储单元。 这就是说，40° 角的正弦不是存在从正弦表始点起的第 40_{10}个存储单元， 而是在第 80_{10}个存储单元。然后，寄存器对D内的正弦表基地址加到寄存器对H 内倍乘后的角上。角的正弦的低字节传送到A寄存器和C寄存器。 寄存器对H内的存储器地址加 1， A寄存器的正弦值的低字节与存储器中的高字节相或。如果角的正弦等于 0,则 8080

转移到 NEGZER，否则，把正弦的高位字节传送到A寄存器，并加上符号位。在 NEGZER 处，正弦的高位字节与符号位一起存入B 寄存器里。然后，把堆栈的内容弹入寄存器对 H 和 D，8080 从 SINANG 子程序返回。寄存器对B 存有角的正弦值。

象前面的程序例子一样，CALCAD 子程序段包括了防止产生负 0 的正弦值的一些指令。8080 把正弦的高、低字节相“或”，确定此正弦值是否为0。如果是0，则 8080 转到 NEGZER，A寄存器内的 0 存入B 寄存器里。这就是说，符号位(B 寄存器的 D_7位)等于 0。如果角的正弦不是 0，则 8080 不执行 JZ-NEGZER 指令，而是把符号位加到正弦值的高位字节上。

还有一个问题没有讨论：当 8080 调入子程序时，如果寄存器对 B 内的角大于 360°，那 SINANG 子程序会产生什么结果呢？无法预料 8080 返回时，A 寄存器(或寄存器对 B，视正弦表的精度而定) 的内容是什么。原因是明显的：8080 产生的存储器地址，比正弦表所用的最高地址还大。避免这个问题的最简单的方法是在进行任何计算之前，检查寄存器对 B 的内容。如果寄存器对 B 的内容是 360° 或者小于 360°，那么可执行子程序的其余指令。如果寄存器对 B 的内容大于 360°，那么，可以从这个角减去 360°，然后把所得的差与 361° 比较，如果其结果等于或者大于 361°，8080 应继续进行减和比较操作过程，直到寄存器对 B 内的角减到 0°和 360°之间为止。此时，8080 就可以执行 SINANG 子程序的其余指令。这就是说，对于有些角来说，必须从这个角多次地减去 360°。

纸带字母穿孔程序

使用过纸带来存储程序或数据的人都会知道，有时很难在纸带的开头写上一个带说明的题目。大多数的笔在上了油的纸带上写不上字，许多墨水会渗开。如果有一个程序可将从电传打字机或 CRT 键入的信息穿孔在纸带上，这个问题便迎刃而解。例如，按下了 T、E、S 和 T 键，字母数字字符 TEST 便以 5×7 点阵形式穿孔在纸带上。完成这项工作的程序叫做纸带字母穿孔程序，它是应用对照表的另一个程序例子。

当电传打字机或 CRT 上的键被按下时，计算机产生的 5×7 点阵字符如图 7-4 所示。假设从电传打字机或 CRT 键入 N，那在纸带上应穿上什么样的数值呢？字符 N 的 5×7 点阵如图 7-5 所示。

应在纸带上穿孔的第一个数值是 177(7 F)。即使大多数电传打字机能并行穿孔八个数据位，但我们只用八个"道"中的七个，第八道，即最高有效的道不用。所以，它总是逻辑 0，即不穿孔。其后应穿孔的值为 002，004，010 和 177(02，04，08 和 7 F)。纸带是从左向右穿孔，同读带的方向一致。

我们在图 7-4 和 7-5 中假定，一个点代表在纸带上穿的一个孔，那么，177,002,004,010 和 177 (7 F,02,04,08 和 7 F) 这些值存在什么地方呢？这些值与另外 63 个 5×7 点阵一道存储在对照表中。因为每一个点阵有 5 个数据值"长"，所以整个表需要 320_{10}个存储单元才存得下。

如果需要的话，表可以由可变长度的节点组成。这就是说，每一个"字符"不一定非用 5×7 点阵表示不可。这种表示法的

优点是可以节省存储单元。例如，用 5×7 点阵存储惊叹号，就必须存两个 000，一个 137(5 F)，再加两个 000。

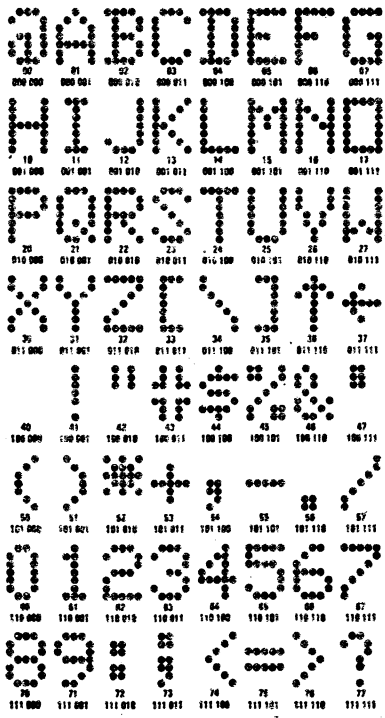


图 7-4　存在对照表中的5×7点阵

如果不用 5×7 点阵，也可以只存 137 (5 F)，后面再加一个结束符。这就是说，只需要两个存储单元，而不是五个。可以使用的另一种存储方法是，把每一个字符所用的存储单元的数目存入一个存储单元，该单元后是用来存字符的存储单元。不论是使用结束符还是使用“节点数”的方法存惊叹号字符，都可以节省三个存储

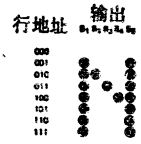


图 7-5　字符 N 的5×7点阵

单元。

可惜的是，ASCII 字符集的64个字符，大多数需要五个存储单元存放其点阵。因此，如果编制一个可变长度的表，许多点阵要占用六个存储单元：五个存点阵，一个存“节点数”或点阵的结束符。我们做过计算，可变长度的表总共需要 349 个存储单元，但如果每个点阵用五个存储单元，存64个点阵只需要

320个存储单元。因此，在纸带上给字母数字字符穿孔的程序，将利用包含64个节点，且每个节点占五个存储单元的对照表。这个程序(例7-4)从电传打字机键盘输入一个字符，在表中找出适当的5×7点阵，并将它穿孔在纸带上。

在这个程序(例7-4)的开头，8080把一个存储器地址装入堆栈指针。然后调用 TTYI 子程序。从子程序返回时，被按下的键的 8 位 ASCII 值存在 A 寄存器中。**电传打字机将不打印这个字符。**

在例7-4，我们用 TTYI 子程序，而不用 TTYIN 子程序。为什么呢？这两个子程序之间存在什么差别呢？TTYI 子程序从键盘输入一个字符，但不在打字机上打印它。TTYIN 子程序也是从键盘输入一个 ASCII 值，但把这个 ASCII 值送回打字机打印。电传打字机的打字机和穿孔机通常机械地连接在一起的。因为我们要在纸带上对一个特定的点序列进行穿孔，所以下按键的 ASCII 值不应送给打字机。这样，这个键的 ASCII 值的相应的5×7点阵穿孔在纸带上。

当8080从 TTYI 子程序返回时，8位 ASCII 值存放在 A寄存器。然后 ANI 指令把奇偶校验位置 0 。现在，8080 应确定所接收的 ASCII 值是否代表其点阵存在对照表中的某个字符。例如，回车符（CR；015，0D），换行符（LF;012，0 A）和报警符(007，07)不能用点阵表示。这可从图7-4看出。这三个 ASCII 字符在图中没有点阵。只有空格（040，20）和下横线(137，5 F)之间的字符可以用5×7点阵表示。所以，8080从 TTYI 子程序返回以后，A 寄存器的内容与177(7 F) 进行“与”操作，把校验位置为0。结果与040(20)进行比较。如果被按下的键的 ASCII值比040(20)小，8080执行 JC-IGNOR。同样，如果被按下的键的 ASCII 值等于或大于140(60)，则 8080执行

JNC-IGNOR。只有所接收的ASCII值在040(20)和137(5F)之间，8080才执行SUI040。这条指令的作用是从任何一个"合法的"ASCII值减去040(20)，从而使A寄存器的内容介于0和077(0和3F)之间。现在，8080应该根据这个数去查找表中适当的5×7点阵。

例 7-4 纸带字母穿孔程序

```
LETTER, LXISP /把一个读/写存储器地址
        STACK/装入堆栈指示器。
        0
IGNOR,  CALL  /从键盘取入一个字符，但
        TTYI  /不打印它。
        0     /(不回送)。
        ANI   /校验位置0。
        177
        CPI   /这个字符比040(20)
        040   /(即空格符的ASCII值)
        JC    /小吗？是的，则忽略
        IGNOR /这个字符。
        0
        CPI   /这个字符值比下横线
        140   /的值大吗？
        JNC   /是的，则也忽略这个字符。
        IGNOR
        0
        SUI   /它是个合法字符，所以从
        040   /ASCII值减去040(20)。
        MOVLA/现在所有的值在000和077(00和
        MVIH  /7F)之间。这个值存入L寄存器，
        000   /H置000。
```

```
        PUSHH   /寄存器对H内的值存入堆栈。
        POPD    /然后，从堆栈送入寄存器对 D。
        DADH    /这个值乘以2。
        DADH    /再乘以 2（总计×4）。
        DADD    /加上原来的数(总计×5)。
        LXID    /现在，把点阵表的始
        TABLE   /地址装入
        0       /寄存器对D。
        DADD    /把这个地址加到×5后的 ASCII
FIRST,  MVIC    /值上。每个字符的点阵占用
        005     /五个存储单元。
NXTCHR, MOVAM   /取一个 8 位字符。
        CALL    /把它穿孔在纸带上，
        TTYOUT  /电传打字机的穿孔机与打字机
        0       /在机械上是互相连接的。
        INXH    /存储地址加1。
        DCRC    /计数减1。
        JNZ     /计数不等于0，
        NXTCHR  /所以从存储器取另一个字符
        0       /并穿孔。
        XRAA    /五行都穿了孔。
        CALL    /现在穿一个空白行
        TTYOUT  /把信息中的字符分开。
        0
        JMP     /从键盘取另
        IGNOR   /一个字符。
        0
TTYI,   IN      /输入 UART 的状态。
        001
        ANI     /只存接收器的状态。
```

```
        001
        JZ      /如果 A=001，有一个键按下。
        TTYI    /如果 A=000，还没有键按下。
        0       /如果没有键被按下，则继续等
        IN      /待。有一个键被按下，所以
        000     /输入这个字符的 ASCII 代码。
        RET     /带着这个字符(在A寄存器内)返回。
TT, MOVBA       /字符存入B寄存器。
TTYO, IN        /输入 UART 的状态字。
        001
        ANI     /只保存发送器的标识位。
        004     /如果 A=004，发送器(打字机)准备
        JZ      /好。如果 A=000，发送器(打字机)不
        TTYO    /空，则继续等待打字机，
        0       /完成其操作，然后打印
        MOVAB   /A 寄存器的内容。
        OUT     /字符从B传送到A后，
        000     /输出给 UART。
        RET     /返回时字符仍在A中。
TABLE, 000      /这是空格字符。
        000
        000
        000
        000
EXCLAM, 000     /惊叹号。
        000
        137
        000
        000
```

现假设按下了电传打字机的空格键，8080收到ASCII值040(20)。因为这个值处在040和137(20和5F)之间，所以8080不返回IGNOR,而是从A寄存器中的ASCII值减去040(20)。减的结果等于0(040-040或20-20),它被存入L寄存器。然后，MVIH指令使H寄存器清零。这就是说，由于电传打字机键盘上按下了空格键，现在寄存器对H内包含了0。如果按下了惊叹号键，那么程序执行到这一点时，寄存器对H的内容会是什么呢？惊叹号的ASCII值是041(21),所以,寄存器对H的内容应是000001(0001)。8080用这个数能从表中找到适当的点阵。

我们已经知道，每个字符用一个5×7的点阵来表示，每个点阵存在五个连续的存储单元。一个可能的点阵表的一部分如表7-4所示。这个表包含了八进制和十六进制的地址与数据。由表可见，当按下空格键时，8080必须产生地址020 000(1000)，当按下惊叹号键时，8080必须产生地址020 005(1005)。做到这一点的最简单的方法是，把减立即数指令的结果乘以5，然后加到表的基地址上。对于空格符来说，乘法操作的结果是0(000 000×005=000 000)。把0加到表的基地址，结果仍等于基地址。对于惊叹号键来说，减操作的结果为1，乘以5得5。把5加到基地址，得020 005(1005)。这个地址应是惊叹号的5×7点阵第一部分所在的存储单元的地址。

在例7-4中，0装入H寄存器之后，寄存器对H的内容压入堆栈，然后又从堆栈弹入寄存器对D。现在，寄存器对H和D包含了同一个16位数。大家知道，某数乘以5等于该数乘以4再加该数。例如：

$$A\times5=A\times4+A$$

表 7-4 点阵表的结构

八进制			十六进制	
地址		内容	地址	内容
020	000	TABLE,000	1000	TABLE,00
020	001	000	1001	00
020	002	000	1002	00
020	003	000	1003	00
020	004	000	1004	00
020	005	EXCLAM, 000	1005	EXCLAM,00
020	006	000	1006	00
020	007	137	1007	5F
020	010	000	1008	00
020	011	000	1009	00
020	012	QUOTE, 000	100A	QUOTE,00
020	013	007	100B	07
020	014	000	100C	00
•	•	•	•	•
•	•	•	•	•
•	•	•	•	•

因此，在寄存器对 H 的内容从堆栈弹入寄存器对 D 之后，接连执行两条 DADH 指令，相当于把寄存器对 H 的内容乘4。然后寄存器对 D 内的原始数与寄存器对 H 内的相乘结果相加，完成乘 5 运算。乘的结果存在寄存器对 H。然后，表的基地址装入寄存器对 D，并用 FIRST 前一条 DADD 指令加到寄存器对 H 内的数上。这时寄存器对 H 存有电传打字机上被按下键的相应的5×7点阵第一个存储单元的地址。现在8080应接连从存储器读出 5 个连续的存储单元，并把它们穿孔在纸带上。

因为每个字符只有五个值要穿在纸带上，所以在 FIRST 处把计数值 5 装入C寄存器。然后，寄存器对H寻址的存储单

元的内容传送到A寄存器，调用 TTYOUT 子程序，把它穿在纸带上。以前我们用 TTYOUT 子程序完成在打字机上打印字符。但是，打字机和穿孔机在机械上相互连接在一起，因而可以用 TTYOUT 子程序把A寄存器的内容穿孔在纸带上。穿孔以后，8080返回到 INXH 指令。

这条指令使存储器地址加1，然后，DCRC 指令使 C 寄存器内的计数值减1。如果没有穿完 5 个存储单元的内容，那么8080转回到 NXTCHR。当所有五个存储单元的内容穿完以后，8080将A寄存器清为零，并把这个 0 值穿在5×7点阵之后，形成一个间隔，从而把每个字符的点阵分隔开。请注意，**空格的ASCII值(040，20)不在纸带上穿孔**。如果对这个值穿孔，则将在打字机上打印一个空格，而在纸带上穿出空格的ASCII值。这样，5×7点阵之间就没有间隔分开。请记住，我们并不关心打印机上打印什么。我们仅关心穿在纸带上的字符。

为确保读者理解8080是怎样计算相应点阵的地址，假设按下了一个Z键。如果表的基地址是 020 000(1000)，那么Z键的点阵存在什么地方呢？

八进制	十六进制
132	5A
−040	−20
072	3A
× 5	× 5
001 042	0122
+020 000	+1000
021 042	1122

Z键的点阵将存在存储器中地址为021 042～021 046(1122到1126)的存储单元。完整的纸带穿孔字符表如表7-5所示。

您想得出一种简化例7-4程序的方法吗？但不能改变表的结构。程序的开头部分可象例7-5所示那样修改。在例7-5，8080在把读/写存储器的一个地址装入堆栈指示器之后，调用TTYI子程序。当8080返回时，ANI指令使A寄存器的D_7置0。然后从A寄存器的内容减去040(20)。这样，合法字符，即空格和下横线之间的字符，现在其值都在000和077(00和3F)之间。然后，CPI指令确定输入字符是否合法。如果A寄存器的内容比100(40)小，8080不执行JNC-IGNOR，而是把数存入L寄存器。如果ASCII字符的值是100(40)或更大，8080将置之不理。

刚才我们描述的程序(例7-4)有时称为**软件字符发生器**。8080使用软件产生了字符(点阵)。然后，这些点阵穿孔在纸带上。这类软件也可以用来在CRT上显示字符，或者在打字机上打印字符。如果是在CRT上显示，则要用不同的点阵使电子束偏转到CRT屏幕的不同的点。如果这种软件与具有七或九根打印针或打印锤的打字机连用，那么只要改变软件所存取的字符表，几乎任何一个字符集都可以用这种打字机打印。

正如第一节的开头所提到的那样，还有一些包含字符表的硬件器件(ROM)。事实上，许多这类器件含有上述那样的表，如Fairchild半导体公司的3257和3258，NS公司的MM4240AA/MM5240AA，Monolithic Memories公司的MM 6061和MM6062，Signetics公司的2513和2516。制造厂家称它们为字符发生器。欲得这些器件更详细的资料，请参考NS公司的使用说明书AN-40和AN-57。

例 7-5 简化纸带字符穿孔程序

```
LETTER, LXISP   /装堆栈指示器。
        STACK
```

表 7-5 完整的穿孔纸带对照表

TABLE,	000	PLUS,	010	SIX,	074	A,	174	L,	177	W,	177
	000		010		112		022		100		040
	000		076		111		021		100		020
	000		010		111		022		100		040
	000		010		060		174		100		177
EXCLAM,	000	COMMA,	130	SEVEN,	141	B,	101	M,	177	X,	143
	000		070		021		177		002		024
	137		000		011		111		014		010
	000		000		005		111		002		024
	000		000		003		066		177		143
QUOTE,	000	DASH,	010	EIGHT,	044	C,	076	N,	177	Y,	003
	007		010		111		101		002		004
	000		010		111		101		004		170
	007		010		111		101		010		004
	000		010		044		042		177		003
NSIGN,	014	PERID,	000	NINE,	006	D,	101	O,	177	Z,	141
	167		000		111		177		101		121
	000		140		111		101		101		111
	167		140		041		101		101		105
	014		000		036		042		177		103
DSIGN,	044	SLASH,	140	COLON,	000	E,	177	P,	177	RSB,	000
	052		020		000		111		011		177
	153		010		066		111		011		101
	052		004		066		101		011		101
	022		003		000		101		006		000
PCENT,	143	ZERO,	000	SCOLON,	000	F,	177	Q,	076	BSLSH,	003
	023		076		000		011		101		004
	010		101		133		011		121		010
	144		101		073		001		041		020
	143		076		000		001		136		140
AND,	044	ONE,	000	LAROW,	000	G,	076	R,	177	LSB,	000
	111		102		010		101		011		101
	126		177		024		101		031		101
	040		100		042		111		051		177
	120		000		101		171		106		000
APOS,	000	TWO,	162	ESIGN,	024	H,	177	S,	042	UPAROW,	004
	007		111		024		010		105		002
	007		111		024		010		111		177
	000		111		024		010		121		002
	000		106		024		177		042		004
RBRAC,	034	THREE,	042	RAROW,	101	I,	000	T,	001	BAROW,	004
	042		101		042		101		001		016
	101		111		024		177		177		021
	000		111		010		101		001		004
	000		066		000		000		001		004
LBRAC,	000	FOUR,	030	QMARK,	000	J,	040	U,	077		
	000		024		002		100		100		
	101		022		001		100		100		
	042		177		131		100		100		
	034		020		006		177		077		
STAR,	025	FIVE,	047	APPRX,	062	K,	177	V,	007		
	016		105		111		010		030		
	037		105		171		024		140		
	016		105		101		042		030		
	025		071		076		101		007		

0

IGNOR, CALL /从键盘取一个字符，

```
TTYI    /但不打印(即
0       /不回送)。
ANI     /奇偶位(D7)置0。
177
SUI     /从所有 ASCII 字符减去
040     /040(20)，合法字符具有000到
CPI     /077(00到3 F)内的值。
100     /大于上面值为非法。
JNC     /值大于077，
IGNOR   /不予理睬。
0
MOVLA   /值存入L。
MVIH    /H 置为000。
000
•
•
```

第八章　命令译码程序

读者通常也许希望微型计算机按照命令执行一定的操作。这个命令可用一组开关置逻辑 1 和逻辑 0 的形式输入，也可通过电传打字机或者通过 CRT 显示器的键盘送入微型计算机。微型计算机必须编好程序，以便接收开关闭合或按键产生的输入信息，然后检查该信息，确定这个命令所规定的适当的操作。最后执行规定的操作。这种类型的软件叫做命令译码程序。

命令译码程序用在什么地方呢？所有的 BASIC 解释程序和编译程序都有命令译码程序，所以读者可以使用或输入诸如 SCR、LIST、LET、FOR、DATA 和 IF 之类的命令。此外，在诸如**系统监控程序、调试程序、编辑程序和汇编程序等程序**中也有命令译码程序。

单字母命令译码程序

当电传打字机或者 CRT 显示器的键盘上的一个数字键或一个字母键被按下时，最简单的命令译码程序可产生一个动作。键盘上的每一只键能够代表一个特定的命令；因此；一个特定的操作序列将被执行。对一个简单系统监控程序来说，表 8-1 所列出的命令也许是有用的。

正如读者可以从表 8-1 所看到的那样，每一个命令都是由一个字母表示的，没有两条命令用同一个字母表示的。例 8-1

所列出的命令译码程序实际上是为8080编程序，使8080能识别（译码）这四个单字母命令。

表 8-1 命令译码程序的命令一览表

命　令	操　作
E	执行始于符号地址ENTER的程序段。
D	执行始于符号地址DELET的程序段。
L	执行始于符号地址LIST的程序段。
M	执行始于符号地址MOVE 的程序段。

在例 8-1 里，8080 首先调用 TTYIN 子程序。在这个子程序中，8080 等待电传打字机或 CRT 的键被按下。当一只键被按下时，8080 将这个 ASCII 字符存入 A 寄存器，然后从这个子程序返回。请记住，TTYIN子程序还把这个 ASCII 字符的 D_7 位置零，清除校验位。此外，8080 还用 TTYOUT子程序来打印这个字符。如欲复习电传打字机的输入/输出子程序，请参考«8080 / 8085 的软件设计»上册的第三章。

例 8-1 系统监控程序的单字母命令译码程序。

```
/这是一个很简单的单字母命令译码程序，它使用了 CPI 指条。

CMDDEC，CALL/从 CRT 或电传打字机取一个字
        TTYIN/符。
0
CPI
115             /这个字符是 ASCII M吗？
JZ              /是的，然后执行 M (MOVE)。
MOVE            / 子程序
```

```
0
CPI
104         /这个字符是 ASCII D 吗
JZ          /是的，然后执行 D (DELET)。
DELET       /子程序。
0
CPI
114         /这个字符是 ASCII L 吗?
JZ          /是的，然后执行 L (LIST)
LIST        /子程序。
0
CPI
105         /这个字符是 ASCII E吗?
JZ          /是的，然后执行E (ENTER)。
ENTER       /子程序
0
JMP         /它不是M、D、L或E。
CMDDEC/所以，忽略这个字符，
0           /并取另一个字符。
```

当 8080 从 TTYIN子程 序返回时，它把 A 寄存 器的内容与 M 的 ASCII 值（八进制 115，十六进制 4 D) 进行比较。如果这两个数相等，这就是说 M 键已被按下，那么，8080 执行 JZ-MOVE 指令。

如果 D 键被按下，则由于 执行 CPI 115 指令的结果， 0 标识位为逻辑 0 。因此，8080 不执行 JZ-MOVE 指令，而执行 CPI104指令。因为A寄存器的内容是D键的ASCII值104，比较的结果是零标识位置为逻辑 1，因此，8080 执行 JZ-DELET 指令。

在存储器的符号地址 ENTER、DELET、 MOVE 和LIST

处必须存有执行所需要完成任务的指令序列。这些任务完成以后，微型计算机应该返回到命令译码程序的始点(CMDDEC)，以便可以输入另一个命令并进行译码。8080将会响应E、D、M或L以外的任何其它命令吗？不。8080对E、D、M或L以外的其它任何字符置之不理。如果8080把输入的键代码与命令译码程序所用的代码一一进行了比较而不一致时，那么，8080将在这个程序的末尾执行JMP-CMDDEC指令。

这种命令译码程序很容易编写和存入8080微型计算机，但是，它确实有一些局限性。如果读者需要20条不同的命令，那么必须要有20条立即数比较指令和20条条件转移指令。存储这些指令，需要多少个存贮单元呢？这个命令译码程序需要106个存储单元，因为每一条命令需要一条两个字节的CPI指令和一条三个字节的JZ指令。此外，还需要六个存储单元来存命令译码程序开头的CALL TTYIN指令和命令译码程序末尾的JMP-CMDDEC指令。

以表为基础的单字母命令译码程序

采用另一种译码技术的命令译码程序如例8-2所示。这种命令译码程序，象本章即将出现的许多程序一样，采用了一个表，表内包含全部键的ASCII代码输入有效命令时，8080要转移的转移地址。这个表的格式如下：每一个命令由一个字母代表，每个字母的ASCII值存储在这个表里，紧跟在这个字母的ASCII值后面的是一个16位地址(两个8位字节)。这个16位地址是在相应的命令被送入时，将要执行的程序的始地址。地址的低位有效字节(LSBY)紧接ASCII值之后存储在表中，低位有效字节

(LSBY)后面是高 位有效字节(MSBY)。 一旦包含所有的有效命令及相应的 16 位地址 的表产生之后， 必须把表结束符 0 存储在最后一个 16 位地址的后面。 在这个例子里， 另外还用了六个存储单元来存储0,因此,如果需要的话,还可以给这个表增添两条命令。

例 8-2 灵活的单字母命令译码程序

```
        /这是一个比较先进的单字母
        /命令译码程序。它具有较高的灵活性。

CMDDEC, LXIH    /寄存器对 H 指向
        CMDTAB/有效命令的命令和
        0       /命令地址表。
IGNOR   CALL    /从CRT或电传打字机
        TTYIN   /取一个字符。
        0
        MOVBA   /把这个 ASCII 字符保存在 B 寄存器里。
CHECK,  MOVAM   /从表中取一字符。
        CMPB    /把它与键盘字符比较。
        JZ
        GETADD  /如果一致，取 8080
        0       /的转移地址。
        INXH    /否则使地加 1，越过该字符。
        INXH    /然后使地址加 2，越过
        INXH    /这个字符后的两个地址字节,
        MOVAM   /从表取一个值。
        CPI     /这个数值是 0
        000     /(表的结束符)吗?
        JNZ     /不是 0，则
        CHECK   /把这个 TTY字符与表内
```

```
        0           /另一个字符比较。
        JMP         /是0，也就是说，
        IGNOR       /整个表已检索完毕，
        0           /则忽视这个 TTY 字符。
GETADD, INXH        /地址加 1，以越过这个字符。
        MOVFM       /把地址的低位字节传送到 E。
        INXH        /再使地址加 1。
        MOVDM       /把地址的高位字节送入 D。
        XCHG        /这个地址现在在 H 和 L 里。
        PCHL        /把这个地址塞入程序计数器 (PC)。
CMDTAB, 101         /命令字符是 ASCIIA。
        125         /转移地址
        315         /是 315125 (CD 55 )。
        102         /命令字符是 ASCIIB。
        232         /转移地址是
        314         / 314232 (CC 9 A)。
        103         /命令字符是 ASCIIC。
        000         /转移地址是
        370         / 370 000 (F 800)。
        104         /命令字符是 ASCII  D。
        000         /转移地址是
        376         / 376 000 (FE 00)。
        000         /这儿还有六个存储单元，用来
        000         /存储数字零。
        000         /增添的两条命令
        000         /及其相应的地址可以存储在这
        000         /些单元。
        000
        000         /这是表结束字符。
```

在例 8-2 的开始，把存放有效命令和 16 位地址的表的始地址装入寄存器对 H 。然后调用 TTYIN 子程序。键盘上的一只键被按下时，8080 输入这只键的 ASCII 值，把 D_7位置 0，并打印这个字符。8080 把这个七位 ASCII 值存入 A 寄存器后从 TTYIN子程序返回。然后，8080 执行MOVBA指令，把这个值保存在B寄存器里。在CHECK处，寄存器对H寻址的存储单元的内容传送到 A寄存器，然后与 B 寄存器的内寄进行比较。这两条指令使输入的ASCII值与存储在表内的第一个ASCII值进行比较。

如果电传打字机或 CRT 上的 B 键按下时，那么，情况又怎么样呢？ B 键产生的 7 位 ASCII 值为 102 (42)。当这个值与存储器中的ASCII值(101,41)比较时，零标识位被清除为零，因为这两个值是不等的。因此，8080 不执行 JZ-GETADD 指令，而把存储在寄存器对 H 的存储地址加 1 三次。寄存器对 H 第一次加 1 后指向表里的 125 (55)。第二次加 1 后指向 315 (CD)，第三次加 1 后指向表中的 102 (42)。102 (42)是存储在这个表中的第二个 ASCII 值。

现在，寄存器对 H 内的存储地址是存储在表内的下一个命令值的地址。因为这个表可能只存储了几条命令及其相应的地址，所以第三条命令和所有其它的命令都要与表的结束符0进行比较。结束符用来告诉 8080 ，它已经检查完了整个表。因为8080 已经执行了MOVAM指令，把下一个命令从存储器取了出来，所以它很容易与输入的ASCII值(该值存储在B寄存器)进行比较。

8080 执行了 CMPB 指令 (寄存器 A＝102，寄存器 B＝102) 和标识位测试指令 JZ-GETADD 之后，转移到符号地址 GETADD。因为寄存器对 H 内的地址是表中找到的那条命令的地址，所以，给寄存器对H的 16 位地址加 1 就可以从存储器读出命令地址的低位有效字节，送入 E 寄存器。寄存器对

H内的地址第二次加1，地址的高位有效字节就能够从存储器读入D寄存器。现在，8080已经把它必须转移到的那个地址（B命令的地址）装入到寄存器对D。8080执行XCHL指令，把这个地址传送到寄存器对H；8080再执行PCHL指令，把这个地址装入程序计数器。程序计数器指示到将要执行的下一条指令。后者是输入B命令时8080必须执行的任务的一部分。

假设键盘上被按下的键是E键，而不是B键，那么，情况将又会怎么样呢？当8080执行在CHECK处的CMPB指令时，它把E键的ASCII值(105，45)与A键的ASCII值进行比较。这两个值不相等，所以，8080不执行JZ-GETADD指令，而把寄存器对H的地址加3，使它指示存储着B键的ASCII值(102，42)的存储单元地址。然后把这个值传送到A寄存器，并且把它与0进行比较。因为这个值不等于零，这就说明，8080还没有检查到表的末尾，所以，8080执行JNZ-CHECK指令。

在CHECK处，8080把E和B这两个键的ASCII值进行比较。它们是不相等的，所以寄存器对H内的存储地址加3。现在，寄存器对H指示存储C键的ASCII值(103，43)的存储单元地址。这两个ASCII值不相等，寄存器对H内的存储地址再次加1，指示存储在表中的D键的ASCII值的地址，如此继续下去，直到8080找到正确的ASCII值或者到达表结束符为止。在这个例子中，表中没有存储E键的ASCII值，所以，最后找到的是表结束符。这个值从存储器传送到A寄存器，然后，8080执行CPI 000指令，把这个值与0进行比较。由于比较的结果零标识置1，8080不执行JNZ-CHECK指令，而是转到IGNOR。这样，E键(命令)便被忽略。然后，8080调用TTYIN

子程序，因此能够送入另一个命令。为什么8080不执行E命令呢？E命令被忽视是因为在表中没有存储E键的ASCII值，这就是说，E命令没有被定义，即没有给E命令指定任何操作。

还有其它哪些命令被忽视呢？除了A、B、C和D命令以外，其它所有的命令都被忽视。给表增添一条新命令，需要增加多少个存储单元呢？只需要增添三个存储单元。一个存储单元用来存储这条新命令的ASCII值，另外两个存储单元用来存储一个16位地址。这个地址是在输入这条命令时，8080必须转到的地址。在例8-2里，有七个存储单元存储了零，所以，能够给这个表增添两条命令。请读者注意，我们还必须把表的一个结束符存在存储器里，它位于最后这个16位地址之后。就是这个原因，在例8-2中，有七个存储单元存储零。

在命令译码程序的末尾，XCHG和PCHL这两条指令使8080"跳转"到指定的存储单元。还有哪两条单字节指令会使8080产生同样的操作呢？PUSHD和RET这两条指令也会使8080"跳转"到相应的地址。这些指令把寄存器对D内的16位地址存入堆栈。8080执行RET指令，把这个16位地址从堆栈弹出，送入程序计数器。

假如只有两条命令要译码，那么，整个命令译码程序需要多少个存储单元呢？等于存储命令译码程序和表所需的存储单元数。不论有多少条命令可译码，基本的命令译码程序需要30个存储单元。对于两条命令来说，需要增加六个存储单元，分别用来存储两个ASCII值和两个16位地址。另外，还必须把表的一个结束符0存入存储器里。因此，如果只有两条命令要译码，这种命令译码程序总共需要37个存储单元。如果用CPI型的命令译码程序(例8-1)，那么，对两条命令译码，只需要

16 个存储单元。有没有使表型的命令译码程序所需要的存储单元比 CPI 型命令译码程序为少的点呢？有。这一点可以用表 8-2 所示的公式计算出来。

根据表 8-2，我们可以确定：如果所需要的命令是 13 条或者更多，那么，采用表方法可以更有效地利用存储器。请读者注意，这两种指令译码程序增加新的命令是比较容易的。在以CPI为基础的命令译码程序中，只要把 0 存入存储器中 JMP -CMDDEC指令后便可。为了给命令译码程序增添新的命令，可写入适当的 CPI 和 JZ 指令，来代替 JMP CMDDEC 指令。再在最后的，由这样的两条指令组成的指令序列之后存储 JMP -CMDDEC指令。对于以表为基础的命令译码程序来说，应该把一些零保存在最后那个 ASCII 值及其有关的 16 位地址之后。这些存储单元然后可以用来存储新命令的 ASCII 值及其 16 位地址。当然，最后一个 16 位地址之后，必须把一个零存到存储器，作为表结束符。

用两个表的单字母命令译码程序

如果需要的话，也可以编写一种命令译码程序，让它存取两个表。一个表存储有效命令的 ASCII 值，另一个表存储有效命令被送入时8080转到的地址。例 8-3 所列的程序就使用了这两种类型的表。

在例 8-3 中，8080 把一个表(包含有效命令的 ASCII 值)的地址装入寄存器对D，把另一个表(包含有效命令的 16 位地址)的地址装入寄存器对H。这个命令译码程序与例 8-2 内的命令译码程序在许多方面都是相同的。唯一的差别是，一个程序使

表 8-2 计算两种不同的命令译码程序所需的存储容量

CPI 方 法	表 方 法
S=6+C×5	S=? (C×3)

C=命令的条数

S=需要的存储单元数

用一个表，而另一个程序使用两个表。

正如读者可以看到的那样，使用两个表的命令译码程序比使用一个表的命令译码程序稍微长一点。其理由之一是，在例8-3命令译码程序的开头，必须执行两条LXI型的指令。但是，在GETADD(例8-3)，在E寄存器可以装入相应的16位地址的低字节之前，寄存器对H内的地址不应加1。这就是说，例8-3所需要的存储单元比例8-2只多了两个。请读者注意，在例8-3，两个表只有一个(CMDTAB)需要表结束符。这是因为如果8080到达这个表末端，它也应该到达另一个表的末端。从这两个例子，读者应该判定，使用两个表的命令译码程序，实际上没有什么优点。

多字符命令译码程序

假设要为EXIT(退出)、ENTER(输入)、EXTRACT(抽出)和EQUALIZE（相等）命令编写一个单字母的命令译码程序。正如读者所预料到的那样，我们不能把E键分配给所有这四个命令。键和命令的一些可能分配如表8-3所示。

例 8-3　利用两个表的单字母命令译码程序

```
/这个命令译码程序使用两个独立的表。
/一个表包含有效命令，另一个
/表包含命令的地址。

CMDDEC, LXID      /寄存器对D指示
        CMDTAB /有效命令表的地址。
        0
        LXIH      /寄存器对H指示
        CMDADD /命令地址表的地址。
        0
IGNOR,  CALL      /从 CRT 或电传打字机
        TTYIN     /取一个字符。
        0
        MOVBA     /把这个键盘字符保存在 B 。
CHECK,  LDAXD     /从命令表取一个 ASCII 字符。
        CMPB      /把这个字符与键盘字符比较。
        JZ
        GETADD /如果比较的结果一致，取 8080
        0         /应该转到的地址。
        INXD      /加 1，指到下一个字符。
        INXH      /加2，指到"CMDADD"
        INXH      /表中的下一个地址。
        LDAXD     /从命令表取一个值。
        CPI       /这个值是 0 (表
        000       /的结束符)吗?
        JNZ       /不为 0 ，
        CHECK     /则将这个 TTY 字符与表内
        0         /另一个字符进行比较。
        JMP       /为0，即
```

```
        IGNOR   /整个表已经检查完毕，
        0       /则不理睬这个 TTY 字符。
GETADD, MOVEM   /把地址的低位字节送入 E。
        INXH    /再使地址加 1。
        MOVDM   /把地址的高位字节送入 D。
        XCHG    /现在这个地址存储在H和L。
        PCHL    /把这个数塞入程序计数器(PC)。
CMDTAB, 101     /命令字符是A。
        102     /命令字符是B。
        103     /命令字符是C。
        104     /命令字符是D。
        000     /此处可放一条增加的命令的 ASCII 代码。
        000     /此处可放另一条增加的命令的 ASCII 代码。
        000     /表结束符。
CMDADD, 125     /A命令的地址。
        315
        232     /B命令的地址。
        314
        000     /C命令的地址。
        370
        000     /D命令的地址。
        376
        000     /用于存一条增加的命令的地址。
        000
        000     /用于存另一条增加的命令的地址。
        000
```

根据表 8-3 所列出的键和命令分配，编写单字母命令译码程序是很容易的。但是，要记住Q键代表 EQUALIZE 命令，N键代表 ENTER 命令，可能是困难的。实际上，E 键和X键

表 8-3 EXIT、EXTRACT、ENTER和EQUALIZE命令的键分配

键	命　　令
E	EXIT
N	ENTER
X	EXTRACT
Q	EQUALIZE

容易混淆，因为 EXIT 和 EXTRACT 这两条命令都是由EX开头的。为了消除这种混淆，我们可以写一个命令译码程序，让它使用四个字符的命令。这样，为了从程序退出，可输入EXIT命令。为了执行 EXTRACT 程序段，可输入 EXTR 命令。表 8-4 汇总了这四种操作用的可能的四个字符的命令。用这些四字符命令的命令译码程序如例 8-4 所示。

表 8-4 四个字符的命令一览表

命　　令	操　　作
EXIT	EXIT
EXTR	EXTRACT
ENTR	ENTER
EQUA	EQUALIZE

读者一眼就可以看清，例 8-4 的命令译码程序比前面任何命令译码程序都要复杂得多。现在，我们只能把这种四个字符的命令存入读/写存储器，而不是存入一串寄存器。命令输入以后，8080必须按照字符输入的顺序在表中检索一个与电传打

字机键盘输入的四个字符都一致的节点。然而这种命令译码程序的一个优点是，输入的命令与微型计算机实际执行的操作是很类似的。

在例 8-4 的第一部分，8080 把一个读/写存储器地址 装入寄存器对H。这个地址已分配给了符号地址 CMD。 然后，把可以从键盘输入，并且从 CMD 符号地开始 存入读/ 写存储器的字符的数目装入C寄存器。在 CMDIN 处，8080 调用 TTYIN 子程序，以便能够输入一个 ASCII 值。当 8080 从 TTYIN 返回时，这个字符已经被打印在电传打字机 上， 它 的 ASCII 值的 D_7位将是 0。然后，8080 把A寄存器的内容 与 CTRL/C 代码的七位值进行比较。如果 CTRL/C 代码被输入， 8080 转移到 CMDDEC。这就是说 CTRL/C 代码用 来废除正在输入的命令。我们通过产生 CTRL/C 代码， 就能开始输入一条 新而正确的命令。 如果输入的值不是 CTRL/C 代码， 则把A寄存器内的 ASCII 值保存在寄存器对H寻址的存储 单元。 然后这个地址加 1， C寄存器内的数减 1。 这就是说， 8080 三 次 执 行 JNZCMDIN 指令，就能够输入四个字符， 并且把它们存储在读/写存储器里。第四次输入一个字符后， C寄存器 的内容减为 0，所以，8080 不再执行 JNZ-CMDIN 指令。

在 DECODE 处，表地址被装入寄存器对H， 每条命令的字符的数目(4)装入C寄存器。然后， 8080 把一个值从寄存器对H定址的表传送到A寄存器，并把它与 0 进行比较。这一指令序列的作用是什么呢？ 检查表是否结束。如果从表中读出一个零，那么，打入微型计算机的命令和表中任何一个节点都不一致；如果读出的不是零，即 8080 还没有到达表 的末尾， 则把键盘输入命令所在的读/写存储单元的始地址装入 寄 存器对D。

在NXTCHR处，8080把从电传打字机输入的第一个字符装入A寄存器。然后，它把这个值与存储在表中的第一个值进行比较。如果这两个值不相等，8080转到NXTCMD，在这里，8080必须找到表中下一条命令的始地址。如果头两个ASCII值是相同的（它们一致），那么，8080将对后面三个紧挨着的字符一次一个地进行检查，看是否一致。8080执行INXH和INXD这两条指令，使两个存储器地址加1，然后，C寄存器内的字符数减1。如果C寄存器的内容不等于零，则8080执行JNZ-NXTCHR指令。这就是说，还没有出现四次一致。如果C寄存器的内容减为0，那么，输入命令的ASCII值和表中一个命令的ASCII值，已经连续四次出现了一致。

当四个字符完全一致时，8080必须从表中取得适当的16位地址，并把程序执行转到这个地址。这个16位地址存在表内四个ASCII值（字符）的后面，低位字节在前，高位字节在后。所以，当出现四个字符完全一致后，后面两个存储单元的内容装入到寄存器对D。XCHG指令，使这个地址装入寄存器对H，PCHL指令把寄存器对H的内容装入程序计数器。于是程序计数器的内容为这条指令引导程序执行转向的那个程序段的始地址。

例 8-4 每个命令四个字母的命令译码程序

```
                    /这是每个命令四个字母
                    /的命令译码程序
CMDDEC, LXIH        /寄存器对H中装入作为存储
        CMD         /打入命令
        0           /的存储器地址
        MVIC        /C寄存器用来给
        004         /字符的数目计数。
```

```
CMDIN,  CALL    /从电传打字机取一个字符。
        TTYIN   /送回 CRT 显示，并把它存
        0       /入A寄存器而返回。
        CPI     /键盘产生的是 CTRL/C吗？
        003     /如果是的，则作废。
        JZ      /再给寄存器对H内
        CMDDEC/的存储地址和
        0       /C寄存器内的字符数预置初值。
        MOVMA /把这个命令字符保存在存储器里。
        INXH    /存储器地址加 1。
        DCRC    /字符计数器减 1。
        JNZ     /4 个字符已经取完了吗？没有，取
        CMDIN   /另一个。
        0
DECODE, LXIX    /命令已经被打入，现在
        CMDTAB /判明它是什么命令。寄存器对H
        0       /给命令表定址。
AGAIN,  MVIC    /寄存器C用作每个命令的字符
        004     /计数器。
        MOVAM   /从表取一个字符。
        CPI     /这个字符是表结束符吗？
        000
        JZ      /是的，整个表已经检查
        CMDDEC  /完毕，没有出现一致，
        0       /所以，忽略这条命令，再另取一条命令。
        LXID    /寄存器对D对存储打
        CMD     /入命令的存储单元寻址。
        0
NXTCHR,LDAXD    /取另一个命令字符。
        CMPM    /把它与表的一个字符比较。
```

```
        JNZ     /两者不一致，找
        NXTCMD  /表中的下一条命令
        0
        INXH    /这两个字母是一致的，试比较
        INXD    /下面两个。两个地址都加 1。
        DCRC    /字符计数器减 1。
        JNZ     /四个字符都比较完了吗？
        NXTCHR  /没有。比较后面的字符。
        0
        MOVEM   /从存储器取地址
        INXH    /到寄存器对D。
        MOVDM
        XCHG    /把这个地址存入H和L。
        PCHL    /转移到那个地址。
NXTCMD, INRC    /由于有两个地址字节，
        INRC    /所以，使字符数加 2。
NOTYET, INXH    /表地址加 1。
        DCRC    /到达下一条命令了吗？没有，
        JNZ     /继续使这个地址加 1。
        NOTYET
        0
        JMP     /H和L指向表里的
        AGAIN   /下一条命令，试比较
        0       /输入命令和这一命令。
CMDTAB, 105     /"EXIT"里的 ASCII E
        130     /X
        111     /1
        124     /T
        125     /低位地址
        315     /高位地址
```

```
105        /"EXTR"里的 ASCII  E
130        /X
124        /T
122        /R
232        /低位地址
314        /高位地址
105        /"ENTR"里的 ASCII E
116        /N
124        /T
122        /R
000        /低位地址
370        /高位地址
105        /"EQUA"里的 ASCII E
121        /Q
125        /U
101        /A
000        /低位地址
376        /高位地址
000        /表结束符
```

在 MOVEM 指令之前 NXTCHR 之后，为什么没有 INXH 指令呢？因为最后一次循环中寄存器对D和H的地址都加了1，然后，C寄存器的内容减为 0。程序执行的这一点上，寄存器对H恰好指向一个存储单元，这个存储单元内有程序执行将要转向的 16 位地址的低位字节。

如果 8080 发现从键盘输入的命令与表里的第一条四个字符的命令不一致，那么，它就执行 JNZ-NXTCMD 指令。在 NXTCMD处，C寄存器的内容加 2。为什么这样做呢？请读者记住，C寄存器内的数只反映命令中的字符的数目。可是，

命令表不仅包含了每条命令的四个字符，而且还包含了两个8位的地址字节。8080越过这两个地址字节，它才能到达命令表的下一条命令。因此，C寄存器的内容加2，以便8080考虑到存储这个16位地址的两个存储单元。这两条INRC指令被执行以后，寄存器对H内的存储地址加1，而C寄存器存储的数减1，从而使寄存器对H指向存储着表中下一条命令的第一个字符的存储单元。

如果EXTR命令被输入，将发生什么情况呢？微型计算机把EXTR命令的E和X与表里第一个节点EXIT进行比较。但是T和I的ASCII值是不相等的，所以，8080执行JNZ-NXTCMD指令。这时，C寄存器存储的内容是002(02)。在NXTCMD处，这个数加2，成为004(04)，然后，寄存器对H的内容加1，指向存储EXIT命令的T的存储单元。然后，C寄存器的内容从004减为003(04到03)。为了跳过EXIT命令之后的16位地址，寄存器对H的内容加1，指到这个地址的低位字节，而C寄存器的内容减为002(02)。为了跳过地址的高位字节，寄存器对H再加1，寄存器C减到001(01)。最后，寄存器对H的地址加1，以指向存储下一条命令的第一个字符的存储单元。这时，C寄存器的内容减到0。于是，8080转移到AGAIN。

在AGAIN处，8080把字符的数目004(04)装入C寄存器，然后判明是否已经到达了表的终点。105(45)这个数值从表传送到A寄存器；因为这个数值不等于0，所以，8080继续检索。在NXTCHR之前，从键盘输入并且被存储在读/写存储器的第一个字符的地址被装入寄存器对D。然后，8080把输入命令EXTR与寄存器对H寻址的命令EXTR进行比较。不出所料，四个字符都一致。然后，EXTR的地址（314232，

CC9A）从表中读入寄存器对D。这个地址与寄存器对H的内容交换，然后装入程序计数器。

可变字长的命令译码程序

对四字符命令译码程序（例8-4）最后可能做出的重要改进是使各条命令的字长可变。也就是说，根据需要这些命令所包含的字符可多可少。以EXIT、EXTRACT、ENTER和EQUALIZE为例，命令译码程序可"使用"这些命令的各个字母表8-5对可变字长和固定字长的命令进行了比较。

表 8-5 可变字长命令和固定字长命令比较

可变字长	固定字长
EXIT	EXIT
EXTRACT	EXTR
ENTER	ENTR
EQUALIZE	EQUA

如果使用可变字长命令，命令译码程序所采用的表的结构必须改变。这就是说，并不是只把一条命令的头四个字母存入表中，而是把一条命令的所有字母都存储在表里。正如读者所看到的那样，各条命令的字母的个数各不相同。如果表的结构改变，那么，命令译码程序也必须予以改变。命令输入程序段也必须改变，以使任何数目的字符都能输入和存入读/写存储器。只有输入了一个特定的结束符，才执行真正的"比较"程序段。当然，命令的字符数目实际上有一定的限度。如果使用长

于 20 个字符的命令，想必是不可行的，因为记住这样长的一条命令是困难的。为了结束输入到微型计算机的字符串，将使用 RETURN（回车）键（ASCII 015，0 D）。这个键是任意选定的。因此，只有在 RETURN 键被按下后，8080 才试图比较输入的字符和表里的字符串。

既然命令的字符长度是可变的，那么，计数方法还可以用来确定何时到达表里的命令末尾吗？可以的，如果我们把**每条命令的字符的数目**存储在该命令之前的话。例如，把 004(04)这个数存贮在 EXIT 命令的ASCII 字符的前面。对于 EXTRACT 来说，007(07) 这个数存储在它的 ASCII 字符之前；对于 ENTER 来说，使用 005(05)这个数；对于 EQUALIZE 来说，使用 010(08)这个数。当然，这种方法要求读者知道每条命令使用多少个字符，因为必须把这个数字存储在表中这条命令的第一个字符的前面。这种新表的一部分如例 8-5 所示。

例 8-5 可变字长的命令的表结构

```
CMDTAB,004      /"EXIT"命令中字母的个数
        105     /ASCII E
        130     /X
        111     /I
        124     /T
        125     /低位地址
        315     /高位地址
        007     /"EXTRACT"命令中字母的个数
        105     /ASCII E
        130     /X
        124     /T
        122     /R
        101     /A
```

```
103/C
124/T
232/低位地址
314/高位地址
•
•
```

在实际比较输入的命令和表里的一个命令之前，必须把字符数从表里送到C寄存器。在其他各方面，影响C寄存器的指令仍旧是相同的。但是，可变字长的命令译码程序会出现另一个问题。假设在表中只有两条命令，它们使用下面这种新格式存储：

```
CMDTAB,002      /"GO"命令中字母的个数
        107     /ASCII G
        117     /O
        107     /低位地址
        232     /高位地址
        004     /"GOOD"命令中字母的个数
        107     /ASCII G
        117     /O
        117     /O
        104     /D
        345     /低位地址
        246     /高位地址
        000     /表的结束符
```

对于每条命令（GO和GOOD）来说，命令的字符数目被存储在命令（"测试串"）的第一个ASCII字符的前面。这二个数分别是002和004。然后把各字符的ASCII值存入表里，

后面紧接着的是一个相应的16位地址。在表的末尾，存储表的结束符。

现在，假设GOOD命令被输入了。根据前面这个表，8080将转移到哪个地址？是232107 (9A47) 还是246 345 (A6E5)？微型计算机将"错误地"转移到232 107 (9A47)这个地址。其理由是：微型计算机把表里的两个字符的GO命令和从键盘送进去的GOOD命令的头两个字母进行比较，结果完全一致。因为在GO命令里只有两个字符，所以C寄存器所存储的字符数将从2减为0。因为该比较是"成功"的，所以，将从存储器里取出232 107 (9A47) 这个地址，作为从键盘输入的GOOD命令的始地址。这个问题有没有简单的解决办法呢？有的。微型计算机所要做的是当它到了表里的ASCII字符串的最后一个ASCII字符时，确定是否已经达到输入命令(存储在读/写存储器)的最后一个字符。在前面这个例子中，微型计算机把表中的GO命令与输入的GOOD命令里的GO比较，结果一致；但是，当微型计算机检查是否已经到达GOOD命令的最后一个字母时，它会发现，它还要比较GOOD命令的OD字母。换句话说，当微型计算机检查输入时存储GOOD命令的存储单元时，它没有找到结束符RETURN（如果输入的命令是GO，微型计算机会找到RETURN。）

可变字长的命令译码程序如例8-5所示。请注意，在该程序的CMDIN程序段，RETURN键的ASCII值被存储在存储器里，位于该命令之后。

正如读者所看到的那样，我们用命令译码程序能够很容易地控制微型计算机执行的操作序列。在小系统中，我们或许只用四五条命令，每条命令用一个字符或数来表示。如果微型计算机必须执行的操作数增加，命令的复杂性（每条命令的字母

或数字字符的个数）也可能随之增加。如果命令的复杂性增加了，那么，或许要使用多字符的命令译码程序。我们已经看到，这种命令译码程序能够对诸如 CLOSE VALVE 3（关闭三号阀门）或 PUMP 1 ON（开动一号泵）这类命令进行译码。

在本章中讨论过的各个译码程序，都可以认为是解释程序。我们把一串字母数字字符输入微型计算机，微型计算机解释它，然后执行这条命令所规定的操作。这些操作执行完毕以后，微型计算机返回到命令译码程序，从而另一条命令可以输入，并且被解释。

例 8-6 可变字长命令的命令译码程序

```
/这是可变字长命令的命令
/译码程序。

CMDDEC,  LXIH    /寄存器对H中装入
         CMD     /存储打入字符
         0       /的存储器地址。
CMDIN,   CALL    从电传打字机取一个字符，送回
         TTYIN   /CRT显示，然后把它
         0       /存入A后返回。
         MOVMA   /把这个命令字符保存在存储器里。
         INXH    /存储器地址加1。
         CPI     /存储在存储器的这个字符
         015     /是结束符（回车符）吗？
         JNZ     /不是。
         CMDIN   /取另一个命令字符。
         0
DECODE,  LXIH    /命令已经打入，现在
         CMDTAB  /判明它是什么命令。寄存器对H
```

```
        0           /指向命令和命令地址表。
AGAIN,  XRAA        /把A寄存器置零。
        CMPM        /把存储器的内容与A的内容比较。
        JZ          /存储器的内容等于0，因而
        CMDDEC      /整个表已经检查完毕，
        0           /没有出现一致，所以取另一条命令。
        MOVCM       /存贮器的内容不等于0，因而它
        INXH        /是字符个数。加1，指到表的第一个字。
        LXID        /寄存器对D中装入存储打入，
        CMD         /命令的存储单元的地址。
        0
NXTCHR, LDAXD       /取一个命令字符。
        CMPM        /把这个字符与表内一个字符比较。
        JNZ         /不一致，寻找
        NXTCMD      /表里的下一条命令。
        0
        INXH        /这些字母是一致的，再试下面两
        INXD        /个字母。使两个地址加1。
        DCRC        /字符计数器减1。
        JNZ         /所有的字符都比较完了吗？没有，
        NXTCHR      /则比较下面两个字符。
        0
        LDAXD       /在刚刚打入的命令的末尾
        CPI         /有回车字符吗？
        015
        JNZ         /没有。因此，在表里没有
        NXTCMD      /找到适当
        0           /的命令。再试下一条命令。
        MOVEM       /从存储器取一个地址，送
        INXH        /寄存器对D。
```

```
          MOVDM
          XCHG     /把这个地址存入H和L。
          PCHL     /转移到这个地址。
NXTCMD,   INRC     /由于有两个字节地址，
          INRC     /所以，把字符数加 2。
NOTYET,.  INXH     /使表地址加 1。
          DCRC     /到达下一条命令吗?没有，
          JNZ      /继续使地址加 1。
          NOTYET
          0
          JMP      /H和L指示到表里的下
          AGAIN    /一条命令，以比较
          0        /这个命令与表的一项。
CMDTAB,   004      /"EXIT"的字母个数
          105      /ASCII E
          130      /X
          111      /I
          124      /T
          125      /低位地址
          315      /高位地址
          007      /"EXTRACT"的字母个数
          105      /ASCII E
          130      /X
          124      /T
          122      /R
          101      /A
          103      /C
          124      /T
          232      /低位地址
          314      /高位地址
```

```
005     /“ENTER”的字母个数
105     /ASCII E
116     /N
124     /T
105     /E
122     /R
000     /低位地址
370     /高位地址
010     /“EQUALIZE”的字母个数
105     /ASCII E
121     /Q
125     /U
101     /A
114     /L
111     /I
132     /Z
105     /E
000     /低位地址
376     /高位地址
000     /这是表结束符
```

第九章　系统监控程序

读者知道，生产厂所提供的 8080 或 8085 微处理器集成电路并不带有 50 条或 100 条指令的固化程序，相反，它需要编程序，以便在从存储器向指令寄存器(IR)读入一条特定指令时执行规定的操作。假设读者已经有一个 8080 微型计算机系统，它有 1024 个字的读/写存储器。这个存储器连接的地址是 000000～003377（000～03FF)。这样，当 8080 的 RESET（复位）引线的逻辑电平为 1 时，8080 的程序计数器清零，存储在 000000（0000）存储单元的指令被读入到指令寄存器（IR），然后执行。现在，读者要开始给微型计算机编程序。遗憾的是，你没有任何外部设备，可以用来把程序输入到微型计算机。

硬接线前面板

微型计算机的设计人员认识到缺乏输入/输出设备可能成问题；解决办法之一是在微型计算机系统里装上硬接线前面板。硬接线前面板能使你直接对微型计算机的存储器进行存取，也就是说，可以使用这种前面板，把信息存入存储器，并对存储在存储器的信息进行检查。前面板与微型计算机的数据总线和地址总线是通过硬线连接的。访问前面板时，不用执行任何输入/输出指令。这就是说，前面板必须产生一个 16 位的存

储器地址，使用户能够把八个开关的状态（逻辑1或逻辑0）存储在一个特定的存储单元。前面板还必须用脉冲信号控制$\overline{\text{MEMW}}$和$\overline{\text{MEMR}}$这两条信号线，以便把信息写入存储器，或者从存储器里读出。

因为前面板要使用8080微型计算机的地址总线和数据总线，那么，当你从前面板把信息存入存储器时，8080能够执行程序吗？不能。事实上，当前面板在使用时，前面板必须**封死**，即防止8080使用这些数据总线。另一方面，8080正在执行程序时，必须阻止前面板使用这些总线。

所有这些功能，需要相当数量的硬件，而且，制作硬接线前面板成本较高。E&L仪器公司生产的DM-1微型计算机的硬接线前面板的方框图如图9-1和9-2所示。PDP-8计算机(DEC公司)、Intellec系列的微型计算机(Intel)和MITS(PERTEC)及IMS合股公司（IMSAI）的某些微型计算机也都采用了硬接线前面板。

用硬接线前面板可以把指令操作代码或数据值直接输入存储器。完成这些任务并不需要任何软件。一旦指令（如果需要，还有数据）被保存在存储器后，8080就能够复位，然后从存储单元000000（0000）起，开始执行存储在存储器里的程序。但是用这种前面板输入50到60条指令的程序，要花去相当多的时间。

为了解决这种硬接线前面板成本高的问题，以及减少向微型计算机输入信息所需要的时间，许多生产厂制造了带软件驱动"前面板"的微型计算机系统。至今这种前面板更为普遍；例如KIM（MOS Technology公司）、MMD-1(E&LInstruments)、SDK-85（Intel公司）和H8（Heath公司）等机器上都可以找到这种前面板。现在，最流行的一种软件驱动的前面板

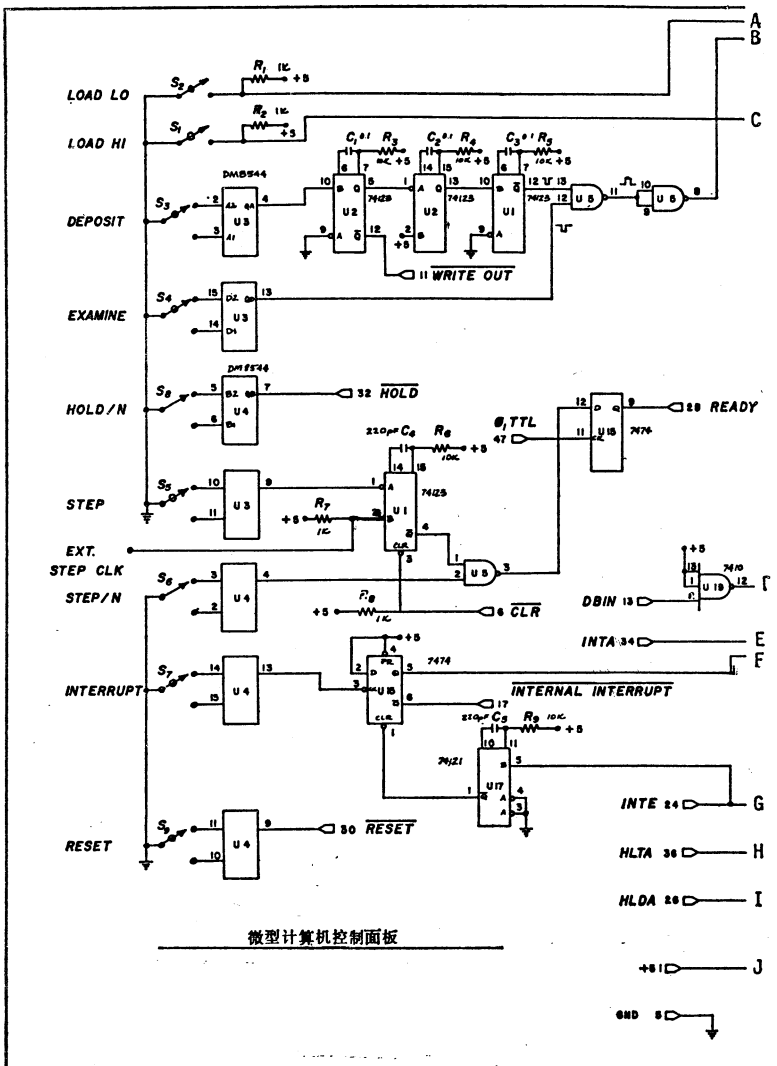
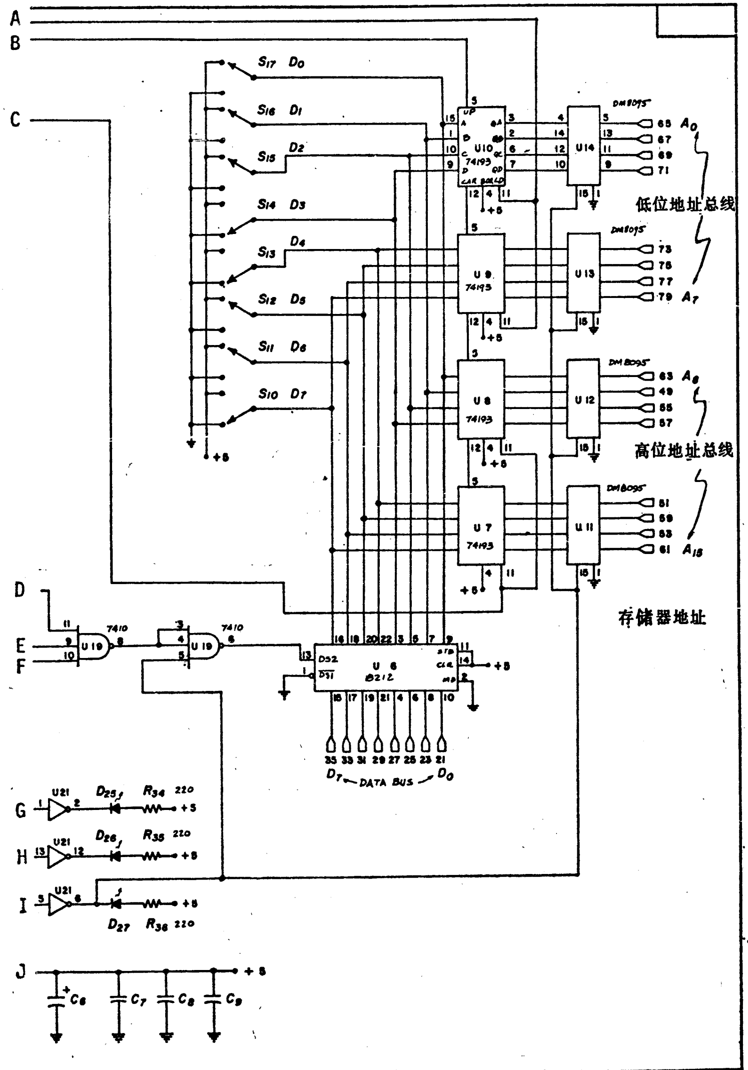


图 9-1　以 8080 为基础的微型计算机 MD-1(E&L仪器公司)的硬接线前面板的方框图



A
B
C
D
E
F
G
H
I
J
S17 D0
S16 D1
S15 D2
S14 D3
S13 D4
S12 D5
S11 D6
S10 D7
U10 74193
U9 74193
U8 74193
U7 74193
U14
U13
U12
U11
DM8095
A0
A7
A8
A15
低位地址总线
高位地址总线
存储器地址
U19 7410
U6 8212
D7 DATA BUS D0
U21
D25 R34 220
D26 R35 220
D27 R36 220
C6 C7 C8 C9

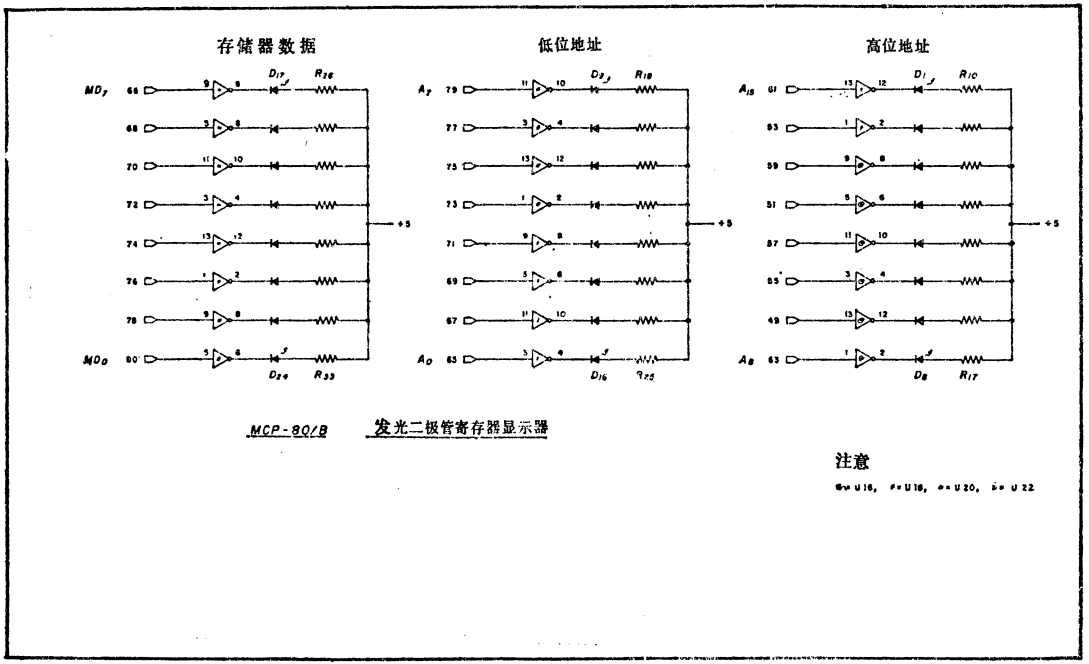


图 9-2　图 9-1　所示的硬接线前面板的 LED 显示器

是由键盘（16～25只键）和一些七段发光二机管显示器所组成的。这些设备与8080微型计算机连接，或者作为累加器输入/输出设备。或者作为存储器映象输入/输出设备。8080微型计算机还必须具有某种非易失性的只读存储器(ROM)，以便让"驱动"键盘和发光二极管显示器的程序常驻在微型计算机系统里。

这就是说，当微型计算机的电源被切断时，存储在ROM里的系统监控程序不会丢失，它仍旧保存在ROM里。如果把系统监控程序存储在读/写存储器里，那么一旦电源被切断，它将消失。读/写存储器需要电能，才能保持被存储的信息；ROM则不然。Commodore公司的PET和Radio Shack公司的TRS-80微型计算机都用ROM来存储BASIC解释程序。如果这样做了，则微型计算机在启动之前不必从磁带录音机装入程序。

系统监控程序构成微型计算机系统的"心脏"或"核心"。如果没有系统监控器程序，你就不能在键盘上把指令操作码或数据值输入微型计算机，也不能把数值显示在发光二极管显示器上或存入读/写存储器。

一般系统监控程序的特点

简单的系统监控程序可以执行的操作归纳在表9-1里。如果一个系统监控程序与软件驱动的前面板结合起来，能够完成所有这些操作，那么，这个系统监控程序就能够完成硬接线前面板能完成的各种操作；但是，它必须是在软件程序的控制下。软件驱动的最简单但未必最便宜的前面板之一可以由一个八位ASCII字符的键盘和六至九个七段发光二极管显示器构

成；如果用八进制数制显示地址信息和数据信息，那么，需要九个显示器；如果用十六进制数显示这些信息，则需要六个显示器。如果采用八（或十六）进制数制，则需要用六个（或四个）显示器来显示一个16位的存储器地址，用三个（或二个）显示器显示这个16位地址内存储的内容。一开始我们将使用ASCII键盘，因为它是一种常用的外部设备。我们将假设键盘的接口逻辑给输入的键代码编码和消除键闭合所产生的颤动。

表 9-1 简单系统监控程序的命令归纳

1. 指定任何一个16位存储器地址。
2. 检查任何存储单元的内容。
3. 改变任何存储单元的内容。
4. 从任何存储地址开始执行程序。
5. 现行存储器地址加1。

我们要求系统监控程序执行什么操作呢？要能指定一个16位地址或者一个八位数据字。数据字或者代表一个指令操作码，或者代表一个具体数据值。为了指明被输入的信息是一个16位地址，我们将必须按下A键，接着按下六位八进制数字或者四位十六进制数字。当整个十六位地址输入之后，它在七段“地址”显示器上显示。然后，必须把显示地址的存储单元的内容显示在七段“数据”显示器上。这时，也许我们对这个存储单元的数据感到满意。如果是这样，则按下N键，从而下一个存储器地址及其内容将被显示。N键代表NEXT命令。如果我们需要改变这个被寻址的存储单元的数值，我们只要输入三位八进制（两位十六进制）数就行了。

完成这一任务的最简单的而并不一定是最好的方法是编写一个系统监控程序，从而在任何时候都可以输入一个八进制数

或十六进制数。一旦数被输入，它就自动地存入 LED 显示器上显示的 16 位地址的存储单元。然后，存储器地址加 1，显示新的存储器地址及其内容。

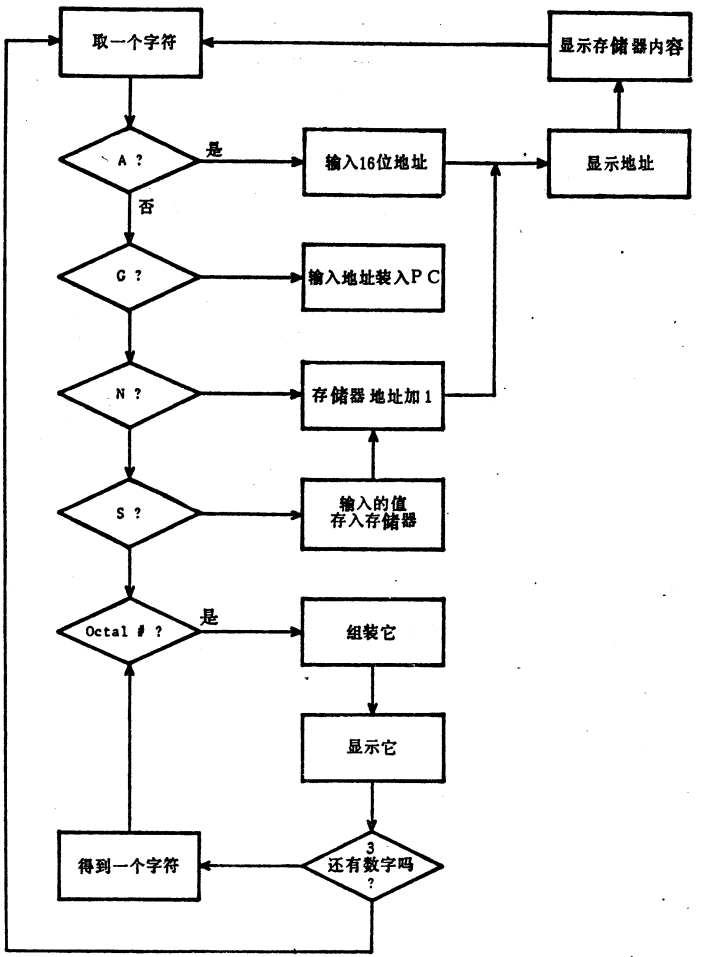


图 9-3　简单系统监控程序的流程图

假设我们必须把 341 (E 1) 存入 005135 (055D) 存储器单元。为此可按下 A 键，接着是 005135，然后再按 341 这些数字键。但是，假设在输入 341 最后一位数字时出了错，把 1 变成了 2，那会发生什么问题呢？342 会被存入 005135 这个存储单元，存储器地址，变为 005136。然后新的存储器地址及其内容被显示。为了改变 005135存储器单元内的 342，应该输入 16 位地址 (A 005135)，然后输入341。如果输入 341 时，又犯了另一个错误，那么应该再重新输入和存入其中正确的数字值。

由此可见，改正错误需要付出巨大的努力。因此，最好让用户根据需要输入八进制数字或十六进制数字。每输入一位新数字，前面的 8 位数向左移，空出的空间用来存储三位的八进制数字或四位的十六进制数字。只有键盘上的 S 键 (代表保存命令 SAVE) 被按下时，实际所显示的八位数才存入指定的存储单元。当这个"保存"操作完成之后，存储器地址加 1，显示新的地址和数据值。因此，当 S 键被按下时，存储器地址自动地加 1，不必再按其他的键。

S 键 (SAVE命令) 也可以用来执行 N 键 (NEXT 命令) 所执行的全部操作。这就是说，S 键不仅可以用来把"显示"的信息保存在存储器里，而且也可以用来检查连续的存储单元。这种简单系统监控程序的流程图如图 9-3 所示。执行这些操作的系统监控程序列在例 9-1。

简单系统监控程序

例 9-1 的系统监控程序有四条命令。这些命令给我们提供下述功能：指定一个 16 位地址 (A 或 ADDRESS命令)；使存

储地址加1(N或NEXT命令)；把8位数据值保存在存储单元(S或SAVE命令)和开始执行存储在存储器的程序（G或GO命令)。当信息保入存储器时，当我们检查存储器的内容时，或者当8080转移到指定的存储单元开始执行程序时，系统监控程序都使用16位地址。

例 9-1 有四条命令的简单系统监控程序

```
        * 000000
START,  LXISP     /把一个读/写存储器地址
        300       /装入堆栈指示器。
        003
CMD,    CALL      /从键盘取
        KEYIN     /一个 ASCII 字符。
        0
        CPI       /是"地址"键A被按下了吗?
        101       /
        JZ        /是的，则A后必定是一个六
        ADDR      /位地址。
        0
        CPI       /是 G(GO)键被按下了吗?
        107
        JNZ       /不是。G键没有被按下。
        NORS      /判明它是不是S键。
        0
        PCHL      /它是G键，把H和L 作为地址。
NORS,   CPI       /S键被按下了吗?
        123
        JNZ       /否，S键没有被按下。判断
        NOTS      /是不是N键被按下。
        0
        MOVMD     /S键被按下了，则把
```

```
        JMP     /D的值保存入存储器里，使
        ADRUP   /存储器地址加 1，然后显示新
        0       /地址及存储器内容。
NOTS,   CPI     /N键被按下了吗?
        116
        JNZ     /否，N键没被按下。
        NMBIN   /判是否正在输入
        0       /一个八进制数。
ADRUP,  INXH    /使存储器地址加 1，
        JMP     /然后显示这个新地址
        ADROUT  /及该地址的存储内容。
        0
NMBIN,  LXID
        003     /把位数存入 E，
        000     /并清除D，以供暂存用。
        CALL    /然后，调用"CONVERSION"(转换)子程序
                 的一
        SKIP    /段，因为A寄存器已经
        0       /存储了键的 ASCII 值。
        JMP     /把被输入的值
        CMD     /存入D寄存器，然后
        0       /返回命令译码程序。
OCTIN,  LXID    /把 003 装入 E 寄存器，
        003     /把 000 装入D寄存器。
        000
1GNOR,  CALL    /从键盘取
        KEYIN   /一个 ASCII 字符。
        0
SKIP,   CPI     /这个 ASCII 字符值
        060     /小于 ASCII 0 吗?
```

```
        JC          /是的，则不理该字符。
        IGNOR
        0
        CPI         /它等于或大于
        070         /ASCII 8 吗？
        JNC
        IGNOR       /是的，不理它。
        0
        ANI         /它是一个八进制数，所以
        007         /处理它。
        MOVBA       /把低三位保存在 B。
        MOVAD       /取前一个数，
        RLC         /然后乘以 8。
        RLC
        RLC
        ANI         /丢失低三位。
        370
        ADDB        /加上刚输入的数。
        MOVDA
        OUT         /把这个数输出到
        002         /"数据"七段显示器。
        DCRE        /使位数减 1。
        JNZ         /如果这个数不等于零，
        IGNOR       /则取另一个数字字符。
        0
        RET
ADDR,   CALL        /A 键被按下了，所以，取
        OCTIN       /一个六个数字的存储器地址。
        0
        MOVHD       /然后把前三位数字保存在 H。
```

```
          CALL      /再取低位地址。
          OCTIN
          0
          MOVLD     /把它保存在L寄存器。
ADROUT,   MOVAH     /再显示高位地址。
          OUT
          001
          MOVAL     /再显示低位地址。
          OUT
          000
          MOVDM
          MOVAM     /取这个地址的存储内容
          OUT       /并显示。
          002
          JMP       /再取另一个命令或
          CMD       /数据字。
          0
KEYIN,    IN        /输入ASCII键盘
          001       /的状态字，
          ANI       /只把键盘标识位
          200       /保存在A寄存器里。
          JZ        /该标识位是逻辑0，所以
          KEYIN     /没有任何键被按下。
          0
          IN        /输入8位ASCII值，
          000       /并清除键盘标识位。
          ANI       /把ASCII值的
          177       /校验位置零。
          RET       /8080把这个ASCII值存入
                    /A寄存器，然后返回。
```

把读/写存储器的一个地址装入堆栈指示器之后，8080调用KEYIN子程序，从键盘取一个ASCII字符值。然后它执行命令译码程序的一些典型指令。如果键盘上的A键被按下了，则8080转移到ADDR。在ADDR处，8080两次调用OCTIN子程序，因此，16位地址就能以两个三位的八进制数的形式输入。这个地址被输入之后，8080把寄存器对H的内容和该寄存器对寻址的存储单元的内容输出给LED显示器(在ADROUT处)。8080执行完这些操作之后，返回到CMD，以便另一条命令可以被输入和译码。

如果代表GO命令的G键被按下，那么，显示的地址（也存储在寄存器对H）被作为程序执行的起始地址，并从这个地址开始执行程序。这以后，程序控制回到系统监控程序的唯一途径是将8080微型计算机复位。

如果S键(SAVE命令)被按下，8080就把D寄存器的内容保存在寄存器对H所寻址的存储单元，然后转移到ADRUP处。在这里，寄存器对H的内容加1。接着，8080转移到ADROUT处，把寄存器对H的内容和寄存器对H寻址的存储单元的内容显示在LED显示器上。然后，8080返回到CMD，从而可以输入另一条命令。

如果N键(NEXT命令)被按下，寄存器对H的内容加1，加1所得的结果与寄存器对H寻址的存储单元的内容一道显示。这些操作执行完毕之后，8080返回到CMD处。如果这四条命令中没有任何一条输入，那么，8080试把这个ASCII值解释成一个八进制数的一部分。因此，8080执行在NMBIN处的指令。

在NMBIN处，8080把000 003(0003)装入寄存器对D，这样D寄存器清零，从而能够用来存储那些以ASCII为基础

的八进制—二进制的转换结果。因为三位八进制 ASCII 数将转换成一个八位的二进制数，所以，把 3 装入 E 寄存器。然后，8080 调用 SKIP 子程序。这个子程序假设 A 寄存器内已有键盘来的一个 ASCII 值；并假设 D 和 E 这两个寄存器已预置初值。实际上，SKIP 子程序是较长子程序 OCTIN(八进制输入）的一部分。如果 8080 从 OCTIN 开始调用这个子程序，那么，它将对起初存储在 A 寄存器的一个 ASCII 值视而不理。8080 调用这个 SKIP 子程序，就能处理 A 寄存器的这个 ASCII 值，然后 8080 两次调用 KEYIN 子程序，共输入三个 ASCII 数。OCTIN 和 SKIP 的子程序的末尾有几条指令，它们用来显示 D 寄存器的现行内容，即由三个数字键输入的八位数值。

假设九位数字的显示器显示地址 012 300，在这个地址上的存储器内容是 203，这个数也显示在显示器上。如果现在按下 ASCII 键盘上 2 这只数字键，那么，显示器将显示什么呢？被显示的地址还是 012 300，但是显示的数据值将是 002。因为 8080 还在执行 OCTIN 子程序中的指令，所以，必须还要按下两只数字键，8080 将才能从这个子程序返回。一旦另外两只键被按下，则 8080 返回。**作为三位 ASCII 数字，输入的八位数值并不存储在 012 300 这个存储器单元，它只存储在 D 寄存器里**。只有当按下 S 键时，存储在 D 寄存器里的数才将存入 012 300 存储单元。

如果在稍后一些时间，7 这只数字键被按下三次，那情况将会怎样呢？则会显示 377 这个数值。同样，如果数字键 5 被按下三次，则显示 155。请读者记住，对于三位八进判数字，显示器只显示一个 8 位二进制数。因此，777 和 555 这两个数分别被截短为 377 和 155。

这个系统监控程序一被启动，如果立即按下 S 键，那会

发生什么现象呢？S 键会使 D 寄存器的内容保存到寄存器对 H 寻址的存储单元。这个存储单元的地址是什么呢？这是没有办法知道的。最简单的方法是在设置堆栈指示器的内容后，8080 立即转移到 ADROUT。这使寄存器对H的内容和寄存器对H寻址的存储单元的内容一起显示。程序指令也把这个存储单元的内容复写到D寄存器里，因此，如果按下S键，那么D寄存器的内容会存回到同一个存储单元。最后的结果是什么是不会改变的。

G键的用途是什么呢？当G键被按下时，寄存器对H的内容装入程序计数器。这就是说，寄存器对H存储着 8080 将要执行的下一条指令的存储器地址。如果从存储器地址 002 145 开始将程序装入 8080 微型计算机，那么，也许要从存储器地址 002145 开始执行这个程序。为了这样做，应该输入A002 145G。假设你只有 1024 个字的读/存储器，该存储器的地址在 000000～003377(0000～03 FF)之间；如果你输入A 200 127G，那情况将会怎么样呢？8080 将“跳转”到不存在的存储器单元 200 127，并从这个单元取出指令操作码 377(FF)，因为数据总线“浮置到逻辑 1”。377(FF)是 RST 7 指令的操作码，所以，8080 将调用在 000 070(0038)地址上的子程序。这个地址上的这条“指令”是 OCTIN(SKIP)子程序的一条 CPI 指令的数据字节 070。

不用说，读者使用 GO 命令时必须注意，要保证被显示的地址确实是你希望 8080 转移到的地址。假设A000 000G被输入到微型计算机，那么情况怎么样呢？8080 转移到存储器地址 000 000，然后开始执行存储在这里的程序。因为这是系统监控程序的始地址，所以，系统监控程序仅仅是被再启动。

例 9-1 的系统控制程序处理 16 位和 8 位八进制数。如果

A键被按下，则必须接着输入16位二进制数(六位八进制数)。如果数据值将被保存入存储器里，那么，它们必须作为八位二进制数(三位八进制数)输入。如果系统监控程序只输入和操作八位数，那么，它可以大大地被简化。我们还可以用它来输入16位地址，只不过现在把16位地址作为两个八位数输入罢了。如果输入的是一个八位数，并且我们要它代表16位地址的高8位，那么我们按H键，即输入**高位地址命令**。然后把这个8位数值传送到H寄存器，显示器的内容被更新，反映出地址的这一变化。

我们先输入一个8位数，然后按L键，即输入**低位地址命令**，则可以改变地址的低8位。一旦显示出适当的16位地址，S键就可用来：① 把新的信息保存在被寻址的存储单元，然后使存储器地址加1；或者(2)只使存储器地址加1。如果非常认真地研究例9-1程序，那么你将会看到NEXT命令(N键)是不需要的。如果从键盘输入一个数并按下S键，那么这个新的数值被保存在寄存器对H寻址的存储单元，并且存储器地址增1。如果存储器里已经存有正确的数值，那么S键最后所起的作用只是使存储的单元地址加1。实际上，无论S键何时被按下，D寄存器的内容总是被写入到寄存器对H寻址的存储单元。但是，当用A命令一指定地址，寄存器对H寻址的存储单元的内容就**传送到D寄存器**。因此，如果按下S键，而且没有打入新的数字信息，那么看起来就好象只有存储器地址加1似的。实际上，D寄存器的内容在存储器地址加1之前存入了存储器。地址加1以后，存储在新寻址的存储单元的数值传送到D寄存器里，以准备存回到这个存储单元，或者在改变后写回它里面。后一个操作最后的结果仍然是使地址加1。这是很重要的一点，因为S命令允许读者检查存储单元和使它不改变或者

改变。因为S命令所做的一切正是N命令所做的，所以我们在例9-2列出的新的简化了的系统监控程序里，只需要4条命令。这些命令是H、L、S和G。

在例9-2中，8080首先给堆栈指示器置值，然后显示寄存器对H里的地址，以及该存储器单元的内容。请注意，存储单元的内容不仅传送到A寄存器，然后输出，而且还传送到D寄存器。在IGNOR处，8080调用KEYIN子程序，从而可以从键盘输入一个ASCII值。然后，8080执行命令译码程序的一些指令，确定是否按下H、L、S或G键。

如果H键被按下，D寄存器的内容传送到H寄存器，然后，显示新的16位地址和数据。如果L键被按下，D寄存器的内容就传送到L寄存器，然后，显示新的地址和数据。如果S键被按下，D寄存器的内容被保存在寄存器对H寻址的存储单元。然后，存储器地址加1，显示新的地址和数据值。如果G键被按下，寄存器对H的内容被装入程序计数器(PC)。

如果这四只键没有一只被按下，那么，8080就把这个ASCII值解释为一位八进制数字。如果这个ASCII值不代表一个有效的八进制数字，那么，8080转移到IGNOR处。如果这个值是有效的，那么，把它转换成相应的三位二进制数值，并与已传送到D寄存器的内容相结合。因为8080转移到DOUT处，所以，显示这个新的数值。

例 9-2 四条命令的简化系统监控程序

```
          *000 000
START,    LXISP   /把读/写存储器
          300     /的一个地址装入堆栈指示器。
          003
DISPLA,   MOVAH/取存储地址的
```

```
        OUT     /高位字节并显示。
        001
        MOVAL   /取存储器地址的
        OUT     /低位字节，并显示。
        000
        MOVAM   /取这个地址的
        MOVDM   /存储内容，然后
DOUT,   OUT     /显示它。
        002
IGNOR,  CALL    /取一个 ASCII 字符。
        KEYIN
        0
        CPI
        110     /H键被按下了吗？
        JZ      /是的，则把输入的
        HI      /数送入高位地址寄存器，再显示
        0       /新地址。
        CPI     /L 键被按下了吗？
        114
        JZ      /是的，则把输入
        LO      /的数送入低位地址寄存器，再
        0       /显示新地址。
        CPI     /S 键被按下了吗？
        123
        JZ      /是的，则把输入的
        STEP    /数保存在寄存器对H
        0       /寻址的存储器单元，再使地址加 1 。
        CPI     /G键被按下了吗？
        107
        JZ      /是的，则转移到寄存
```

```
        GO      /器对H寻址的
        0       /存储单元。
        CPI     /这个数值比 ASCII 0 小吗?
        060
        JC      /是的，则不理它。
        IGNOR
        0
        CPI     /这个值等于或大于
        070     /ASCII 8 吗?
        JNC     /是的，则不理它。
        IGNOR
        0
        ANI     /不，则把这个八进制数保
        007      存在B。
        MOVBA
        MOVAD   /取前一个数值，
        RLC     /然后左移，以腾出空
        RLC     /间，供存储一个新的八进制数字
        RLC     /用。
        ANI     /只保存高 5 位。
        370
        ADDB    /加刚刚输入的数，
        MOVDA   /然后把它保存在 D 寄存器。
        JMP     /显示新数，然后
        DOUT    /取另一个 ASCII 字符。
        0
HI,     MOVHD   /把输入的数
        JMP     /送入H寄存器，
        DISPLA  /然后，显示新的地址。
        0
```

```
LO      MOVLD   /把输入的数送到
        JMP     /L寄存器，然后，
        DISPLA  /显示新的地址。
        0
STEP,   MOVMD   /把输入的数值保存在存储器
        INXH    /里，使存储器地址加1。
        JMP     /然后显示新地址。
        DISPLA
        0
GO,     PCHL    /把寄存器对H的内容装入PC。
KEYIN, IN       /输入ASCII键盘
        001     /的状态字。
        ANI     /只把键盘的标识位
        200     /保存在A寄存器里。
        JZ      /这个标识位是逻辑0，
        KEYIN   /所以，没有任何键被按下。
        0
        IN      /输入8位ASCII值，
        000     /清除键盘标识位。
        ANI     /把ASCII值的
        177     /校验位置0。
        RET     /把ASCII值存入A
                /寄存器后返回。
```

请注意，在例9-2中，虽然还有用来执行ASCII值的八进制—二进制转换的指令，但是没有OCTIN子程序。而且，当一个八进制数的ASCII值被输入时，没有使用数字计数器。在例9-1，8080执行了一条LXID指令，目的在于把3装入E寄存器(3是数字计数数)，把0装入D寄存器。在例9-2，S键

有两个用途。如果一个新的八进制数被输入，S键将使得这个数保存在寄存器对H寻址的存储单元。然后，地址加1，并且被显示。如果没有任何新的数值输入，那么，从存储器读出的数值被写回到同一个存储单元。**S键可以完成这两种功能的唯一的原因是：每次8080执行DIPLA之后的指令时，它执行一条MOVDM指令。**实际上，D寄存器用来把整个系统监控程序"联系"在一起。D寄存器的内容可以用作地址的高八位，或地址的低八位，或者可以保存在存储器里。

用于非ASCII键盘的系统监控程序

前面我们假设一个，ASCII编码的键盘与8080微型计算机连接，我们可以用它来输入数字值和命令。对于微型计算机的某些用户来说，ASCII键盘可能太昂贵，或不实用。于是，他们用一个16或25键的键盘，与他们的8080微型计算机系统连接起来。但是，这些键不能产生例9-2的系统监控程序所希望的ASCII代码，而是产生表9-2所列出的代码。例9-2的系统监控程序可以加以修改，使之能够用这种键盘把命令和数字信息输入到8080微型计算机吗？是的，可以的。我们只需要用一个换算表，把这种键盘产生的键代码转换成ASCII值就行了。系统监控程序的新的KEYIN子程序列于例9-3。

例9-3的换算表用来把这种键盘产生的键代码转换成系统监控程序所需要的ASCII值。这就是说，KEYIN子程序是例9-2系统监控程序中唯一修改过的部分。在例9-3中，8080输入键代码之后，执行ANI 017指令，把D_4～D_7位置零。然后

把寄存器对D和H的内容保存在堆栈，因为在计算换算表地址时，将使用这两个寄存器对。

寄存器对D和H的内容压入栈之后，把四位的键代码从A寄存器传送到E寄存器，D寄存器置零。然后把换算表的基地址装入寄存器对H。8080执行DADD指令，把寄存器对D内的一个四位的键代码加到换算表的基地址上，再把被按下的键的ASCII值从存储器传送到A寄存器。然后，栈顶内容弹入寄存器对H和D，8080带着A寄存器里的ASCII值返回。例9-2中原来的KEYIN子程序的操作正是这样。

表 9-2 16键键盘产生的代码

键	键盘产生的代码
0	012
1	000
2	003
3	013
4	017
5	005
6	001
7	014
H	004
L	007
G	010
S	011

这里要指出的重要的一点是，键代码被用作为**一个地址**。因为每一只键所用的代码是唯一的，而且它所代表的地址因而也是唯一的，所以，与各个键相应的ASCII代码被存储在由键盘的键所"寻址"的存储单元。在表9-2中，键4产生的代码是017。当这个数值被加到表的基地址(000 371)时，所得到的

结果地址等于 001 010。这个存储单元的内容是 4 的 ASCII值，即 064。这样，通过使用换算表，键代码被用来对存储适当的 ASCII 值的存储单元寻址。

例 9-3 用查表法转换键代码的 KEYIN 子程序

```
                                *000341
000 341 333 KEYIN,IN        /输入 ASCII 键盘的
000 342 001         001     /状态字。
000 343 346         ANI     /只把键盘的标识位
000 344 200         200     /保存在 A 寄存器里。
000 345 312         JZ      /标识位是逻辑 0
000 346 341         KEYIN   /所以，没有任何键
000 347 000         0       /被按下。
000 350 333         IN      /输入 8 位的 ASCII
000 351 000         000     /值，清除键盘的
000 352 346         ANI     /标识位。只把 D3、D2
000 353 017         017     /D1 和 D0 保存在 A 寄存器里。
000 354 325         PUSHD   /保存寄存器对 D。
000 355 345         PUSHH   /保存寄存器对 H 的内容。
000 356 137         MOVEA   /把键代码传送到 E。
000 357 026         MVID    /把 D 寄存器置 000。
000 360 000         000     /
000 361 041         LXIH    /把换算表的
000 362 371         TABLE   /始地址装入
000 363 000         0       /存储器对 H 里。
000 364 031         DADD    /把键代码加到这地址上。
000 365 176         MOVAM   /从表里取 ASCII 值。
000 366 341         POPH    /恢复寄存器对 H 的内容。
000 367 321         POPD    /恢复寄存器对 D 的内容。
000 370 311         RET     /把 ASCII 值存入 A 寄存器
```

```
                                    /返回。
                                    /
000 371 061 TABLE, 061      .       /1 此键产生代码 000。
000 372 066                 066     /6 此键产生代码 001。
000 373 377                 377     /没有任何键产生代码 002。
000 374 062                 062     /2 键产生代码 003。
000 375 110                 110     /H 键产生代码 004。
000 376 065                 065     /5 键产生代码 005。
000 377 377                 377     /没有任何键产生代码 006。
001 000 144                 144     /L 键产生代码 007。
001 001 107                 107     /G 键产生代码 010。
001 002 123                 123     /S 键产生代码 011。
001 003 060                 060     /O键产生代码 012。
001 004 063                 063     /3 键产生代码 013
001 005 067                 067     /7 键产生代码 014。
001 006 377                 377     /没有任何键产生代码 015。
001 007 377                 377     /没有任何键产生代码 016。
001 010 064                 064     /4 键产生代码 017。
```

用多路转换显示器的系统监控程序

系统监控程序的最后一个例子，使用一个九位数字的多路转换的七段 LED 显示器和一个十二只键的键盘。多路转换的显示器的接口和显示器的软件说明可以在《8080/8085 软件设计》上册的第七章中找到。我们将使用的这个键盘产生四位键代码，并有一位状态位表明是否有键按下。这种键盘在《8080/8085 软件设计》上册的第七章也描述过。这种键盘不产生 ASCII 代码，其数字键 0～7 的代码是 0000～0111。当输入

键代码时，键盘的状态位不清零，而且，这些键没有消除颤动。状态位只表示何时有键按下，而且只要有一个键被按下时，它将保持在逻辑1状态。为了消除键闭合和释放所产生的颤动，8080必须执行延时指令。

把例9-2的系统监控程序变换成应用这种键盘和显示器的程序，要作许多修改，但是大多数修改涉及多路转换显示器的数据显示，以及检测键闭合、抑制键颤动和输入相应键代码的那些指令。例9-4所列出的系统监控程序可完成所有这些任务。

在例9-4中，8080把读/写存储器的一个地址装入堆栈指示器，然后，把读/写存储器的一个地址装入寄存器对H，并从这个存储单元把一个数据值读入到D寄存器。在AGAIN处，8080调用DISPLA子程序，将寄存器D、L和H的内容显示在九位数字的多路转换显示器上。这些数是从右至左显示的。D寄存器的内容首先被显示。新着是L寄存器的内容，最后显示H寄存器的内容。当一个特定的寄存器的内容正在以八进制七段显示格式显示时，首先显示D_2，D_1和D_0这三位，接着显示D_5、D_4和D_3这三位，最后显示D_7和D_6两位。

在DISPLA处，数字启动代码（显示器中将导通的那一位数字的代码）置零。这个数值存储在E寄存器里。然后，D、L和H这三个寄存器的内容依次装入A寄存器里。每次装入之后，8080调用DIGIT子程序，把每个寄存器的信息显示在七段显示器上。请注意，8080执行MOVAH指令以后，它将继续向下执行，进入DIGIT子程序。这就是说，DIGIT子程序末尾的RET指令，也可以使8080返回到调用DISPLA子程序的程序段。

例9-4 用多路转换显示器和未抑制颤动的12键非ASCII

键盘的系统监控程序

```
START,  LXISP   /把读/写存储器的一个地址
        370     /装入堆栈指示器。
        003
        LXIH    /把一个读/写存储器地址
        000     /装入寄存器对H。
        003
        MOVDM   /从存储器取一个数值到D寄存器。
AGAIN,  CALL    /在多路转换的显示器上
        DISPLA  /显示D、H、L这三个寄存器
        0       /的内容。
        IN      /然后，看看是否有
        000     /键按下。
        ANI     /只保存最高有效位。
        200
        JZ      /没有键被按下，所以
        AGAIN   /显示这些寄存器的内容 13.5
        0       /毫秒之久，然后再检查键盘。
        CALL    /一只键被按下，所以用
        DISPLA  /显示器子程序实现延时，
        0       /消除键闭合的颤动。
        IN      /然后，再次输入键代码。
        000
        ANI     /保留低四位。
        017
        CPI
        014     /H键被按下了吗？
        JZ      /是，则把输入
        HI      /的数送到高位地址寄存器，
        0       /并显示这个新地址。
```

```
CPI     /L键被按下了吗?
015
JZ      /是，则把输入
LO      /的数送到低位地址寄存器并
0       /显示这个新地址。
CPI     /S键被按下了吗?
010
JZ      /是的，则把输入的
STEP    /数存入寄存器对H寻址的存储
0       /单元，然后使地址加1。
CPI     /G键被按下了吗?
013
JZ      /是，则
GO      /传送到寄存器对H
0       /寻址的存储单元。
CPI     /数大于
010     /键7的代码吗?
JNC     /是的。则置之不理。
RELESE
0
ANI     /不。则把这个八进制数
007     保存在B寄存器里。
MOVBA
MOVAD   /取前面的数送A，
RLC     /然后乘以8。
RLC
RLC
ANI     /只保存高5位。
370
ADDB    /加上刚输入的数，
```

```
        MOVDA   /然后把它保存在D寄存器里。
RELESE, CALL    /既然已经
        DISPLA  /发现一只键闭合了，所以等待这
        0       /只键释放。
        IN      /再次输入键盘代码和
        000     /状态位。
        ANI     /只保存最高有效位。
        200
        JNZ     /仍有键处于闭合状态，
        RELESE  /所以等待它释放。
        0
        JMP     /键被释放，
        AGAIN   /等待另一只键被按下。
        0
HI,     MOVHD   /把输入的数移到
        JMP     /H寄存器，然后显示
        NEWCON  /16 位地址和
        0       /该地址上的存储器内容。
LO,     MOVLD   /把输入的数传送到
        JMP     /L寄存器，然后显示
        NEWCON  /16 位地址和
        0       /这个地址上的存储器内容。
STEP,   MOVMD   /把输入的数保存在存储器里。
        INXH    /存储器地址加 1 。
NEWCON, MOVDM   /取存储器的内容，送到D。
        JMP     /显示这数和 16 位地址，
        RELESE  /同时等待键释放。
        0
GO,     PCHL    /把寄存器对H的内容装入 PC。
DISPLA, MVIE    /把数字启动代码置为 000。
```

```
        000
        MOVAD   /首先显示D寄存器的内容。
        CALL
        DIGIT
        0
        MOVAL   /然后显示低位地址。
        CALL
        DIGIT
        0
        MOVAH   /最后，显示高位地址。
DIGIT,  MOVBA   /把字保存在B。
        ANI
        007     /取低三位，
        RRC     /循环右移四次，进入高四位。
        RRC
        RRC
        RRC
        CALL    /显示这个数。
        DIGOUT
        0
        MOVAB   /现在，显示 D5、D4 和 D3
        ANI     /这些信息。
        070
        RLC     /把该信息向左移一次。
        CALL    /显示这个数。
        DIGOUT
        0
        MOVAB   /现在取 D7 和 D6 位到 A。
        ANI
        300
```

```
        RRC     /把该信息向循环右移
        RRC     /两次。
DIGOUT, ADDE    /加数字启动代码。
        OUT     /把这个值输出给
        000     /显示器。
        INRE    /数字启动代码加1。
        MVIC    /现在使显示器增加辉度。
        310
INTENS, DCRC
        JNZ
        INTENS
        0
        RET
```

在DIGIT处，8080把A寄存器的内容保存在B寄存器里。然后把要显示的信息位循环移入A寄存器的D_6、D_5和D_4。这就是说，为了显示D_2、D_1和D_0三位信息，必须执行四条RRC指令；为了显示D_5、D_4和D_3，只要执行一条RLC指令。一旦要显示的信息进入了D_6、D_5和D_4位的位置，8080就执行DISPLA的DIGOUT程序段。在DIGOUT处，把E寄存器内的数字启动代码(只点亮显示器的一位数码)，加到A寄存器所存储的信息之上。然后，把该信息输出给显示器接口。D_6、D_5和D_4是要显示的数值，D_3、D_2、D_1和D_0决定显示器中将显示上述数值的那个位。然后，8080在DIGOUT的末尾时执行延时循环程序，让每个显示管都显示大约1.5毫秒时间。8080用这种方法显示了D、L和H这三个寄存器的内容之后，它从这个子程序返回到AGAIN之后的IN指令。全部显示这九位数字信息大约需要13.5毫秒时间。

位于AGAIN之后的IN指令，把键盘的一位状态位和数据位输入到A寄存器。下面两条指令决定键盘上是否有一只键被按下。如果没有键被按下，则D_7是逻辑0，8080执行JZ-AGAIN指令。这就是说，8080再显示D、L和H三个寄存器的内容；过13.5毫秒之后，8080再执行同一条IN指令。因此，8080每隔13.5毫秒检查一次键盘，看看是否有键被按下。如果有一只键被按下，A寄存器的D_1位是逻辑1，8080调用DISPLA子程序。这样，不仅再度显示数据，而且产生足够长的时间延迟，足以消除被按下去的键的颤动。8080执行了消除颤动的软件之后，将键的代码输入到A寄存器。

然后，8080执行命令译码指令，以便确定H、L、S、G或一个数字键是否被按下。请读者注意，在给8080编程时，为了在CPI指令中写入正确的立即数字节，我们必须知道键盘对每一只键产生的键代码。当H、L、S或G键被按下时，8080执行的操作已经讨论过了。在这一点上，本例子与例9-2很类似。一旦H、L或S命令处理完，8080就转移到NEWCON。在NEWCON处，寄存器对H寻址的存储单元的内容传送到D寄存器里，然后，8080转移到RELESE。

如果是一只数字键而不是H、L、S或G键被按下，则8080执行数制转换操作，把输入的键代码转换成8位二进制数。转换所得到的结果保存在D寄存器。一旦数字信息已经被处理或8080执行了某条命令所要求的操作，那么8080执行在RELESE处的指令。

从8080检测到某只键开始被按下的时间算起，处理键代码所需要的时间仅仅是50～100微秒。因此，8080到达RELESE时，**用户操作员的手指还在这只键上**。因此，8080必须等待用户松开按键，然后才做别的操作。因此，8080调

用 DISPLA 子程序，使得 D、L 和 H 这三个寄存器的内容再次被显示。每隔 13.5 毫秒时间，8080 检查键是否已被释放。如果还没有，那么再次调用 DISPLA 子程序。如果已被释放，那么 8080 转移到 AGAIN；在 AGAIN 处，8080 调用了 DISPLA 子程序。这样一来，键被释放时所产生的颤动就被系统监控程序软件忽略了。

如果键颤动延续了 20 毫秒，那么例 9-4 这个程序应该做什么修改呢？显示每个数字所需要的时间，可以通过改变位于 DISPLA 子程序末尾的 MVIE 指令的立即数字节来增加或者减少。这个子程序所给出的最长的延迟时间大约是 2 毫秒。因此，显示九位数字的信息只需要 18 毫秒。对于消除键颤动来说，这个时间不够长。解决这个问题的办法不是改变 DISPLA 子程序末尾的指令，使较大的数加 1 或者减 1。但是，假设我们这样做了并使显示每一位数字所需要的时间达到 5 毫秒，那情况会怎样呢？这就是说，在 45 毫秒之内应该显示九个数字。结果是整个数每秒钟只显示二十次。读出这种显示是困难的，因为显示器会产生闪烁。

解决这个问题的办法是让 DISPLA 子程序保持不变。但不是在 AGAIN 处、在 RELESE 处和为消除键颤而各调用一次 DISPLA 子程序，而是每一处都调用二次。这就是说，消除键颤动的延时应该是 27 毫秒，而不是 13.5 毫秒。因为有三处调用 DISPLA，所以，为了产生消除颤动 27 毫秒延时，只要增加三条调用指令(九个存储单元)。

用系统监控程序连接程序

到目前为止，我们已经讨论的系统监控程序的例子，都具有两个共同的特点。它们都有相同类型的命令译码程序，都具有某种形式的数制转换子程序或指令序列。但是，并不是所有的系统监控程序都能提供检查和修改(或者二者之一)存储器的内容的能力。实际上，我们可以编写一种系统监控程序，使它只是一个**命令译码程序**。这种系统监控程序可以看作为轮毂，它把存储在微型计算机系统(通常在 ROM 或 PROM)里的其它程序都连接在一起，如图 9-4 所示。

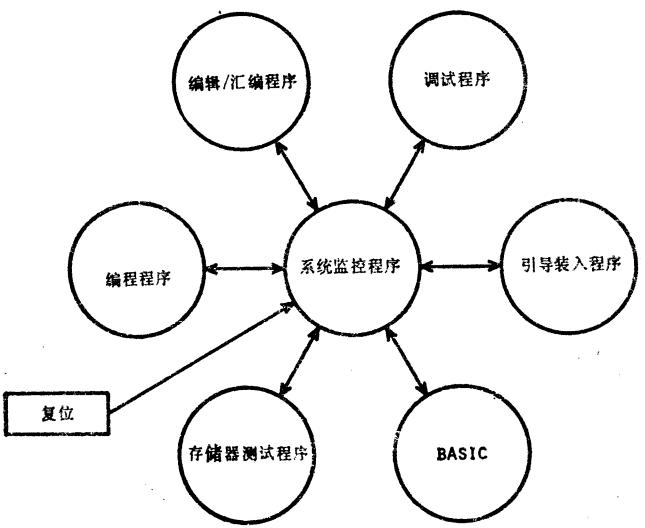


图 9-4　用系统监控程序把其他程序连接一起

在图 9-4 中，读者可以看到，8080 复位时，系统监控程序就将被启动。只要把各种程序的名字输入微型计算机系统，就能和系统监控程序一起，执行存储在存储器里的一个程序。这就是说，在简单的微型计算机系统中，系统监控程序和其它各个程序同时被存入存储器里。常见的这种做法之一是把所有的程序都存入只读存储器(ROM)。因此，当把电源加入微型计算机系统时，许多程序可以立即资用。程序可以存储在读/写存储器。但是，这意味着每次接通电源时，应该把这些程序装入到微型计算机系统。

把所有这些程序"连接"在一起的系统监控程序，可以是很简单的命令译码程序，也可以是很复杂的命令译码程序。一旦用电传打字机或者 CRT 把一个程序的名字输入 8080 微型计算机系统，8080 就在命令和命令地址表中寻找这个名字。如果有，8080 转移到这个程序（存在命令译码程序所用的表里）的始地址。一旦这个程序完成了它所要求的全部任务之后，必须有某种方法，让 8080 返回到系统监控程序。由图 9-4 可见，返回到系统监控程序的最简单的方法是使 8080 复位。但是，经常在所有其它的程序里编入一条命令，这样返回到系统监控器程序就很容易。

系统监控程序内包括的命令和命令地址表，有两个用途。被输入的任何一条命令与这个表里的各项比较。如果出现一致，则 8080 从这个表取一个 16 位地址，然后转移到这个地址。这个表也可以被 8080 用来打印程序目录（有时称为程序单）。这些程序是系统监控程序中的命令译码程序指令所能识别的。该程序目录如例 9-5 所示。打印程序目录要费点功夫，因为所用的表不仅存储构成命令的 ASCII 字符，而且 8 位字符的计数值和每个程序的 16 位始地址也存储在这个表里。

例 9-5 系统监控程序命令表的输出

```
PROGRAMS ON FILE
    BOOT
    DBUG
    DBUGII
    DUP
    LANDM
    LETTER
    TEA
    UPP
    VER
    ENTER A COMMAND, THEN A CR.
    COMMAND=
```

根据例 9-5，微型计算机必须执行的第一个操作是打印信息“PROGRAMS ON FILE:”(文件程序)，因为这没有包括在系统监控程序的命令译码程序所用的命令和命令地址表里。打印这段信息所需要的软件是很简单的，在前面的例子中（例如例 5-14)已经使用过，其中 NXTLET 子程序曾被用来打印“TEST ZIP CODE”。这个信息被打印之后，8080 必须打印存储在命令和命令地址表内的命令(程序的名字)。

为此，8080 必须把一条命令的字符数目读入到一个寄存器，然后把指定数量的字符打印在电传打字机上，或者显示在 CRT 上。然后，8080 必须越过存储在 ASCII 字符之后的那个 16 位地址，再打印回车符和换行(它们并不存储在表里)。通过打印回车符和换行，每个程序的名字将会打印成单独一行。当 8080 从这个表里读出字符计数为零时，它就知道已经把所有的程序名字都打印完了。

然后，8080 打印信息“ENTER A COMMAND, THEN A CR”（输入一个命令，然后输入一个回车符，这之后是回车和换行。接着打印信息“COMMAND＝”（命令＝）。然后，8080 执行系统监控程序内的命令译码程序指令，从而可以用电传打字机或者 CRT 输入一个多字符的命令，并存入读/写存储器。一旦命令被输入并用回车符终结时，8080 确定这个命令是不是有效的。如果是有效的，8080 从表里读出一个 16 位地址，然后转移到这个地址。如果这个命令是无效的，则 8080 再次打印“ENTER A COMMAND, THEN A CR·”和“COMMAND＝”，因此可以输入另一条命令。

除了我们讨论过的操作外，系统监控程序还能够执行其它许多操作。例如，系统监控程序可以用来直接与输入和输出端口通信；还可以用来启动或者禁止外部设备，在这一分钟时间里程序是和行式打印机“通话”，而在下一分钟时间里程序又和软磁盘“通话”。程序不必知道这两个外部设备是接通和断开。系统监控程序也可以用来把数据从一个外部设备传送到另一外部设备，或者用来监控应用程序的操作。

第十章　断点和调试程序

在前一章，我们讨论了系统监控器程序以及它的一些特征。读者知道，我们可以用已叙述过的系统监控器程序把程序装入存储器，并进行检查；如果需要的话，还可以对程序进行修改。一旦一个完整的程序被装入存储器之后，系统监控器程序可以用来把程序执行控制，从系统监控器程序传送到存储器存储的程序。如果读者输入存储器的程序没有给你所期待的结果，那么，情况怎么样呢？这可以说，电传打字机没有打印正确的信息，算术计算的结果是不正确的；也就是 8080 微型计算机没有正确地控制步进电机。

如果我们的问题确实与程序有关，那么可以采取的唯一的措施就是用敏锐的眼睛去检查我们的程序，看看是否能够找出错误。如果找出了错误，那么，我们可以使用系统监控程序来改变该程序的一些指令或者数据，然后再执行这个程序。遗憾的是，为了找出所有的错误，需要把这个程序执行 10 次或者 15 次，如果这个程序每次的执行时间是 5 分钟或者 10 分钟，那么，调试这个程序则需要相当长的时间。如果通过检查程序，还不能找出错误，那么，我们必须编写一些程序来测试几种外部设备，或者主程序中的子程序

因为用这种方法来调试程序可能需要花去很多时间，所以，我们常常使用调试程序来帮助我们找出程序中的错误。汇编语言，BASIC 和 FORTRAN 程序都有调试程序。事实上，可以肯定地说，对已经编写好了的每一种计算机语言来说，都有某

种调试程序。8080 的汇编语言的典型调试程序的命令一览表如表 10-1 所示。

表 10-1 8080 的典型调试程序的命令一览表

1. XXXYYY/OLD	输出存储器单元 XXX YYY 的内容(OLD)
2. XXX YYY/OLD NEW LF XXX YYZ/ABC	把这个新数据字 (NEW) 存入存储器单元 XXX YYY，然后输出下一个连续存储器单元 (XXX YYZ)的内容。
3. XXX YYY/OLD CR	找到存储器单元 XXX YYY 的内容之后，返回到调试程序的命令方式。
4. XXX YYY L	从指定的存储器地址 (XXX YYY)开始，列出存储器的内容。
5. XXX YYY G	执行始于存储器地址 XXX YYY 的程序。
6. P XXX YYY AAA BBB	把存储器单元 XXX YYY—AAA BBB 的内容在纸带上穿孔。
7. R	把纸带上的内容读入存储器。
8. XXX YYY B	把断点置于存储器地址 (XXX YYY)
9. K	从程序中除去最后一个断点。
10. S	执行一条指令，然后在电传打字机上打印标志位的状态以及堆栈和寄存器的内容。
11. X	从最后一个断点地址开始，以全速继续执行程序。

读者从表 10-1 中可以看到，调试程序是面向电传打字机的程序；前一章中所描述的系统监控器程序能执行的许多操作，它都能够完成。为了检查一个存储器单元的内容，我们把进制六位数字存储器地址(XXX、YYY) 输入 8080 微型计算机；排在该地址后面的是斜线符号“/”。然后，把寻址的存储器单元的内容立即打印在斜线符号“/”之后。为了改变这个存储器单元的内容，我们必须把三位八进制有效数输入，然后按下换行键 (CF) 就行了。这样就能把这个八进制新数值保存在存储器里，并能打印出连续存储器单元的地址。该地址的后面紧跟着的是这个存储器单元的内容。如果在存储器单元的内容打印

之后，按下回车符键(CR)，而不是换行键，那么 8080 将返回到调试程序命令，然后，可以输入一条新的命令或者一个新的存储器地址。

因为电传打字机与 8080 微型计算机通信，所以调试程序也具有表特征。这就是说，许多连续存储器单元的内容，不需要我们干与，就可以在电传打字机上列表。调试程序也有一条 GO 命令。当把一个六位八进制数送入以后，则按电传打字机的 G 键。然后 8080 转移到这个地址开始执行在这个地址上的程序。

调试程序还具有保存存储器单元的内容的能力，这是通过把这些内容在纸带上穿孔来实现的。这就是说，我们可以编写一个程序，把它输入存储器里，再对它调试，然后把它存储在纸带上以供日后使用。在稍后一些时间，可用调试程序来读出纸带上的信息输入存储器。虽然不一定使用纸带，但是把程序存储在某种存储器件上的能力是很重要的。在许多微型计算机系统中，音频盒式录音机，软磁盘常用作为大容量存储器装置。

断　　点

下面我们将要讨论的调试程序最后一个特点，也即使调试程序区别于系统监控程序的一个特点是断点。调试程序可在待调试的程序中设置断点。断点是我们可以用来使程序执行到某一特定指令时停止执行的一种方法。随后程序控制(执行)将回到调试程序。在这一点上，调试程序可以用来检查或修改存储器的内容、8080 通用寄存器的内容或指令操作码。断点可由硬件或软件组成，或者软硬结合。我们在此只讨论实现断点的软

件方法。

引起错误的常见原因是，不适当地使用了寄存器或存储单元。一个程序可把两个 8 位字节存入同一个寄存器里，或者当一个 16 位的值应存入两个连续存储单元的时候，存到了两个不连续的存储器单元。为了寻找程序中的这类错误，我们在执行一条特定的指令或一系列指令以后，就得观察一个或数个寄存器的内容，或者一系列存储器单元的内容。

这就是说，我们必须在程序的某一点上设置断点。然后我们令 8080 从起始地址开始执行程序。当执行到断点时，程序控制返回到调试程序。这时，8080 通用寄存器的内容可由电传打字机打印出来。这将告诉我们，当达到断点时，8080 处在什么样的工作状态？将寄存器或某些存储单元的内容与程序正常运行时其中应有的值相比较，我们能够确定程序从开始到断点地址之间有无错误。如果 8080 寄存器中的值等于我们预期的值，则必为下面两种情况之一：(1) 还没有到达出错的地方，(2) 错误出在许多条指令之前。让断点在程序中前后移动，并重新从头开始执行程序，我们最终能够找到错误——引起错误的指令或指令序列。

值得注意的是，调试程序本身并不能找出错误，这是因为 8080 不知道在它的存储器中所存储的指令应该产生什么样的结果。反之，调试程序可给我们显示微型计算机在程序某一特定点上的状态。我们的责任是要确定 8080 的状态是否正确。如果通用寄存器中的某些值是不正确的，那么，断点必须往回移几条指令，并重新执行程序。通过在程序中上下移动断点和观察 8080 在不同断点位置上的状态，我们应能找到错误所在。如果发现在特定断点位置上 8080 的状态是正确的，那么应使断点向前移动一条或多条指令。当然，如果能做到不是从头开始

执行程序，而是令 8080 从最后的断点继续往前执行到下一个断点所在处，那是很有利的。为了使继续执行的指令有用，8080 在继续执行程序之前，应恢复它在达到第一个断点时的状态。

可以看到，调试程序要具有断点特征，就必须完成许多复杂的操作。

下面要讨论的断点特性包括：

1. 适当的断点指令；
2. 人工设置与清除断点；
3. 自动设置与清除断点；
4. 当到达断点时，保存和打印寄存器的内容；
5. 断点操作；
6. 对连续执行命令的要求；
7. 在调试程序控制下的单步执行——一次执行一条指令。

断点指令

当达到断点的时候，我们如何确实地使程序控制返回到调试程序呢？最简便的方法是把一条指令插入被调试的程序之中，当执行到这条指令时控制返回到调试程序。8080 的控制转移指令——转移、调用以及重新启动，能够很好地做到这点。为了在程序中插入这样的指令，调试程序可以把一条返回到调试程序的转移、调用或重新启动指令写在我们要“断开”被执行程序的那一点所包含的指令之上。因此，控制从被调试的程序返回到调试程序的这样的点叫做断点。这一点的地址常称

为断点地址。

要使一条转移或调用指令插入被调试的程序之中，应把三个8位字节写入程序。转移或调用指令的操作码应写入断点地址的存储单元中，而该指令的第二个字节和第三个字节（地址字节）应写入下两个连续存储单元。重新启动指令是单字节指令，因此，如果使用它，只要向被调试的程序写入一个字节。用三字节转移或调用指令作为断点指令（以便把控制转回到调试程序）有一个优点。当执行一条重新启动指令时，8080 总是在 000 000 到 000 070（0000 到 0038）的地址范围内调用一个子程序。但转移指令和调用指令是三字节指令，故可以指定任何地址作为目标地址。因此，如果用转移或调用指令作断点指令，我们就能够立即而方便地“回到”调试程序。如果采用重新启动指令，那么，8080 要从存储器头 56 个存储单元里调出一个子程序，并在这些存储单元中的某一个存储单元（根据所使用的重新启动指令而定），放一条转移指令，以便使控制返回到调试程序。即使单字节指令较容易插入被调试的程序，但还需要有一条三字节转移指令与之配合。此转移指令从重新启动指令的向量地址开始存放。读者如欲更多地了解重新启动指令，请参看本书上册的第四章。

一旦控制返回到调试程序，我们或许就想检查达到被调试程序的断点时，8080 寄存器的内容。实际上，如果不能检查通用寄存器的内容，也就难以调试一个程序。通用寄存器的内容使我们能确定在达到断点时微型计算机干了什么。我们能确定是什么原因导致不同的寄存器中出现“错”值。你或许甚至还没有达到设在被调试程序中的断点位置。这也告诉了你一点情况，在程序开头和断点之间必定有错误。这帮助你“缩小”软件中潜在的问题。

可以用于检查这些寄存器内容的方法主要有两种。第一种方法是，每次到达断点（执行断点指令）时，打印出全部寄存器的内容；第二种方法是，到达断点时，由用户指定要打印的寄存器内容。我个人看法是，最好打印出全部寄存器的内容。这样就可以掌握调试一个程序所需的很多信息。同时，用户与机器的交往减少，因为用户不必指定哪些寄存器要打印。如果寄存器的内容打印在硬拷贝设备上，那么用户以后可以回过头来检查打印结果，并且不必关心其它寄存器里存着什么。

不管使用哪种方法，一旦程序控制传到调试程序，调用一个子程序将寄存器的内容用简明易懂的形式打印出来，那是很容易的事。然而，这种子程序也许使用了三个或四个寄存器来传送寄存器数据，和在电传打字机或 CRT 上打印或显示它们。因此，所有寄存器的内容应怎样保存起来，使它们不致被调试程序中的格式化和输出子程序所改变呢?最简单的解决办法是，执行了断点指令之后，尽快地将全部寄存器压入堆栈。一旦这些值被送入堆栈，就可根据需要从堆栈取出，并以八进制或十六进制的形式打印在电传打字机或显示在 CRT 上。

上面我们阐述了必须完成的操作和能作为断点指令使用的各种指令，现在让我们看看实际使用的程序和子程序。

断点的人工设置和清除

在一个程序中设置和清除断点是较简单的过程。事实上，可以使用前一章中的系统监控程序之一来做这个工作。确定了想要“插入”断点的地址之后，可以把存储器在这个地址上所含指令的操作码以及下面两个连续存储单元的内容记在纸上。接

着，从断点地址开始，依次将转移指令操作码(303，C3)和16位地址写入适当的读/写存储单元。指令第二和第三字节中的16位地址，是存储压入指令的地址。当达到断点时，这些压入指令把寄存器的内容压入堆栈。压入指令之后，则是在电传打字机或CRT上打印或显示寄存器内容的指令。

通过使用转移指令设置了断点之后，就可借助系统监控程序将起始地址输入8080，并开始执行被调试的程序。当达到断点时，8080将转移到先保存后打印寄存器内容的那些指令。此后，控制也许将返回到系统监控程序，以便检查或修改存储器的内容。

如前所述，这种方法的一种改进是用单字节重新启动指令代替三字节转移指令作为断点指令。这就是说，当按断点地址把重新启动指令写入存储器中的时候，只要在纸上记下一个字节。达到断点和打印寄存器内容之后，可用系统监控程序将原来的指令操作码按断点地址送回程序。当然，在实际执行程序之前（用断点地址上的重新启动指令），在存储器中用作断点指令的特定重新启动指令的向量地址上必须存一条转移指令。如图10-1所示。

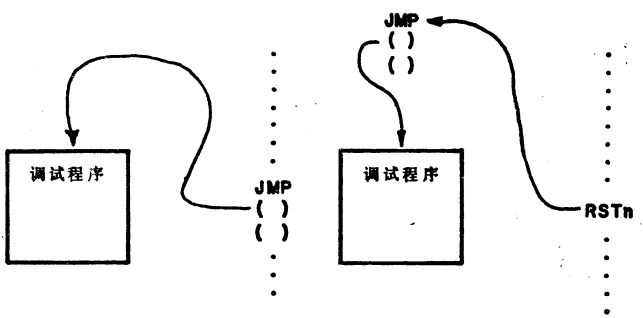


图 10-1 用JMP指令或重新启动指令作断点指令

注意，在图 10-1 中， 如果用 RST 3 指令作为断点指令，那么，在存储单元 000 030，000 031 和 000 032(0018， 0019 和 001A)中必须存放一个转移指令。当 RST 3 指令被执行时，这条转移指令使控制返回到调试程序。如果用另一条重新启动指令作为断点指令，那么，在该重新启动指令的相应向量地址里必须放一条转移指令。

我们刚才讨论的人工放置与清除断点的方法比较费事；也就是说，在断点指令写入以前，要记住断点地址，以及在此地址中所存储的指令操作码。然而，这种方法是可行的。对调试程序的一个明显的改进是让微型计算机按用户规定的断点地址自动设置和清除断点。

断点的自动设置与清除

断点设置

调试程序要进行断点设置，就应在被调试的程序中插入一条重新启动指令。一旦到达断点，微型计算机就应把原来的指令写回程序，代替重新启动指令，这叫做清除或“移去”断点。为了规定断点的 16 位地址，也许要给微型计算机输入004 023 B，BREAK AT 0413，或 BRK 0413 之类命令。不管使用何种方法或格式来规定断点，我们假定，当必须在程序中设置一个断点时，便调用调试程序中的 BREAK 子程序。在这个子程序中，8080 应从用户那里获得 16 位断点地址。 这可以通过调用以 ASCII 码为基础的八进制或十六进制到二进制的转换子程序来实现。这类转换子程序列在本书上册的例 6-3 和例 6-9

中。为使两个 8 位数输入微型计算机，必须两次调用这些特定的子程序。为利用其中的一个子程序来输入一个 16 位 断 点地址，可按例 10-1 所示调用它。

例 10-1：用 OCTIN 子程序输入一个 16 位地址

```
BREAK,   CALL    /取断点地址的高位字节,
         OCTIN   /
         0       /(返回时高位字节在 D 寄存器。)
         MOVHD   /高位字节存入 H。
         CALL    /调用 OCTIN 子程序，取断点低位地
                 /址。
         OCTIN
         0
         MOVLD   /低位地址存入 L。
         .       /然后在这一地址设置断点
         .
         .
```

在例 10-1 中，我们假定首先输入微型机的是 断 点地址的高 8 位，接着是断点地址的低 8 位。注意，如果采用HEXBIN 子程序（见上册例 6-9），则在例 10-1 中必须使用 MOVHA和 MOVLA 指令，而不是使用 MOVHD 和 MOVLD 指令。

例 10-2 用 8080 设断点

```
BREAK,   CALL    /取断点地址的高位字节。
         OCTIN
         0       /返回时地址在 D 寄存器。)
         MOVHD   /高位字节存入 H。
         CALL    /调用 OCTIN 子程序,
         OCTIN   /取断点地址低位字节。
         0
```

```
          MOVLD   /低位字节地址存入 L。
          CALL    /然后在寄存器对 H寻址
          SETBRK  /的存储单元设置断点。
          0
          JMP     /断点设置好后，转回
          CMDDEC  /调试程序的命令译码程序。
          0
SETBRK,   MOVAM   /取"原来"的指令并
          STA     /存入读/写存储器。
          ORGIN
          0
          MVIM    /然后将一条重新启动指令写到
          357     /这一条指令上。
          SHLD    /设置断点的地址
          TEMPO   /存入暂存区"TEMPO"。
          0
          RET     /然后从 SETBRK 返回。
```

请记住，输入的地址是在被调试的程序中想要设断点的地址。因为重新启动指令要"插入"程序中，所以，程序断点地址上原来有的指令先必须从存储器里读出来，并保存在存储器别的地方(高速暂存区)。然后，重新启动指令可写入被调试的程序之中。以例 10-1 作为起点，现在我们能写一个子程序来设置断点和保存待调试程序里被代替的原指令。这个新的子程序如例 10-2 所示。

在例 10-2 中，16 位断点地址用电传打字机或 CRT，以两个三位八进制数的形式送入微型机。这个地址保存在寄存器对 H中。然后调用 SETBRK 子程序，将寄存器 H 所寻址的存储单元内容读入寄存器A。这就是在被调试程序的断点地址中存

放着的指令。随后，这个 8 位指令操作码存入ORGIN(存原来指令的读写存储单元)。RST 5 指令(357,EF) 被写入寄存器对 H 寻址的存储单元。这就把RST 5 指令写入了被调试的程序。

当然，如果中断硬件（见第三章和第四章）已经使用了 RST 5 指令，那就必须用别的重新启动指令作为断点指令。这就是说，SETBRK 子程序（例 10-2）中MVIM 指令的立即数据字节必须改为其它未用的重新启动指令的操作码。同样，由于我们必须将重新启动指令写入被调试的程序中，因而不能用这种设断点的技术来调试只读存储器(ROM)中的程序，包括许多不同的只读存储器（例如 PROM 和 EPROM）在内。

让我们在这里指出很重要的一点：断点只能设在指令操作码所占用的地址上，不能设在数据或地址值占用的地址上。断点只能是用一个操作码代替另一个操作码。如果用断点指令的操作码代替数据或地址值，那么程序将继续往下执行，把断点操作码看作一个新的数据或地址字节。

如果用重新启动指令作为断点指令，那是否还必须完成任何其它辅助操作呢？是的，我们必须在此重新启动指令的向量地址中放一条转移指令，使得在执行重新启动指令的时候，把控制转到调试程序的断点部分。例 10-3 中包含了把一条三字节转移指令(JMP)写入 RST 5 向量地址中的一些指令。

例 10-3 设置断点并在 RST 5 的向量地址中写入 JMP 指令。

```
BREAK,    CALL    /取断点地址的高位字节。
          OCTIN
          0       /(返回时高位地址在 D寄存器。)
          MOVHD   /高位地址存入 H。
          CALL    /调用 OCTIN 子程序，取断点
```

```
          OCTIN   /低位地址。
          0
          MOVLD   /低位地址存入 L。
          CALL    /然后在寄存器对 H 所寻址
          SETBRK  /的存储单元设置断点。
          0
          MVIA    /JMP 指令的操作码被装入
          303     /A 寄存器。
          STA     /将此操作码存入 8080 执行断点
          050     /指令时将转向的存储单元
          000
          LXIH    /JMP 指令中的高地址和低地址字节
          TRAP    /装入寄存器对 H。
          0
          SHLD    /此 16 位地址存入 JMP 指令操作码
          051     /后的存储单元。
          000
          JMP     /断点设置好后，转回到
          CMDDEC  /调试程序的命令译码程序。
          0
SETBRK.   MOVAM   /取"原来"的指令并存入
          STA     /读/写存储器
          ORGIN
          0
          MVIM    /然后将一条重新启动指令
          357     /写到这一条指令上。
          SHLD    /设置断点的地址
          TEMPO   /存入暂存区"TEMPO"。
          0
          RET     /然后从 SETBRK 返回。
```

这里假设地址 000 050，000 051 和 000 052 (0028，0029 和 002A)是在读/写存储器中，并没有让被调试的程序占用。

在例 10-3 中，8080 在设断点之后从 SETBRK 子程序返回，将无条件转移指令 JMP 的操作码装入寄存器 A，再由 A 存入 RST 5 的向量地址 000 050(0028)。然后把 JMP 指令的高、低地址字节装入寄存器对 H 中。这些地址字节接着存入存储单元 000 051 和 000 052 (0029 和 002A)。须注意，例 10-3 被汇编时 (见附录 B)，一个 16 位地址将被指定给符号地址 TRAP，这是调试程序中必须将全部寄存器压入堆栈并打印这些寄存器内容的部分。在本章的另一节里，你将看到调试程序的 TRAP 部分象什么样子。

有可能根本就到达不了程序中所设置的断点吗？有的。假定在例 10-4 中断点插在 003 010(0308)地址。这就是说，RST 5 指令将被写在输出(OUT)指令的上方。用系统监控程序或调试程序的 GO 或 EXECUTE 指令使这个例子从 003 000(0300)地址开始执行，8080 将执行 TTYI 子程序循环，直到电传打字机或 CRT 中有一个键被按下为止。如果按下 Z 键，将会出现什么情况呢？此时微型机将输入 ASCII 值 132。这就是说，8080 将不从 TTYI 子程序返回，而是暂停工作。然而，断点仍设置在存储单元 003 010(0308)。为了清除这个断点，我们假设调试程序有一个 K 命令，能消除未执行的断点，即可以利用 TEMPO 的内容作为存储器地址把 ORGIN(其中存着原来的操作码) 中的内容写回到程序中去。

例 10-4 断点测试程序

```
        * 003 000
DEMO.   LXISP       /读/写存储器地址装入
        STACK       /堆栈指示器，因为即将调用子程序。
```

```
        0
        CALL        /从电传打字机取一个字符。
        TTYI
        0
        ADI         /005(05)加到被按下的键的
        005         /ASCII值上。
        OUT         /将这个值输出到某个七段发光二极管显
        300         /示器。
        HLT         /做完后暂停。
TTYI,   IN          /输入UART的状态
        001
        ANI         /只保存接收器的状态。
        001
        JZ          /如A=001，则有键按下。
        TTYI        /如A=000，无键按下。
        0           /无键按，则等待。
        IN          /一个键被按下，输入该字符的
        000         /ASCII码。
        ANI         /将ASCII值的奇偶位
        177         /(D7)置为0。
        CPI         /电传打字机或CRT上按
        132         /下了Z键吗?
        RNZ         /否，带着寄存器A中的ASCII值返回。
        HLT         /按下了Z键，暂停。
```

断点的清除

在例10-3中，断点的16位地址被保存在读/写存储器的TEMPO开始的存储单元中。因此，当输入调试程序的K命令表示我们要消灭或移去断点时，只要把原先存的操作码，

即 ORGIN 的内容写入 TEMPO 内容所指定的存储单元。这使程序中原来的指令操作码写到 RST 5指令上。

例 10-5 中的指令就是通过这些操作从被调试 的程 序移去断点的。

例 10-5 从某一程序中移去断点

```
CLEARB, LHLD     /断点地址装入
        TEMPO    /寄存器对H,
        0
        LDA      /原来在这个地址的指令操作码
        ORGIN    /装入A寄存器。
        0        /
        MOVMA    /此操作码存在断点地址,
        RET      /然后返回
```

须注意，K 命令用于从一个程序除去断点（见表 10-1）。它可以消除错误设置的断点，或已达到的断点。CLEARB（例 10-5）是一子程序，K 命令可使它被调出，然后控制返回调试程序中的命令译码程序。不久你就会明白，为什么要把 CLEARB 做成一个子程序。

保存和打印寄存器的内容

当 8080 达到了断点并执行 RST 5 指令的时候，便调出在 000 050(0028) 地址上的子程序，使 8080 立即转移到 TRAP（见例 10-3）。

例 10-6 当达到断点时保存寄存器内容

```
                /因为是用RST 5 指令作断点指令,
```

```
                    /8080 执行到断点(即执行RST 5 指
                    /令) 时，便立即转到000 050。
                    /在这个地址上执行JMP 指令，程
                    /序控制即转到TRAP。
TRAP,  XTHL         /H和L压入堆栈，断点地址入H
                    /和L中。
       DCXH         /地址减 1。
       SHLD         /地址保存到"BRKADD"，因为这是到
                    /达断点时的地址。
       BRKADD
       0
       PUSHD        /然后寄存器对D压入堆栈，
       PUSHB        /寄存器对B压入堆栈，
       PUSHPSW      /寄存器A和状态标识压入堆栈。
         .
         .
```

在 TRAP 处，寄存器内容被压入堆栈，还打印或显示在电传打字机或 CRT 上。例 10-6 的这些指令，可在到达断点时将寄存器内容压入堆栈。

当 8080 转移到 TRAP 的时候，XTHL 指令使寄存器对 H 的内容压入堆栈，而栈内的返回地址则送入寄存器对 H。如果断点是在存储器地址003 120(0350) 处，那么寄存器对 H 现在含有003 121(0351)。必须记住，保存在栈内的返回地址比 RST 5 指令的地址大 1。返回地址交换到寄存器对 H 中之后，即被减 1 并存入 BRKADD。这时断点的地址存在BRKADD之中。这个地址采用在调试程序的其它部分。由于 XTHL 指令的作用，寄存器对 H 已压入堆栈。接着所有剩余的 寄存 器对和状态标识也一一存入堆栈。至此，我们可以打印标识位的状态、寄存

器A和其它通用寄存器的内容。

这就是说，在TRAP的某一点上，我们可把堆栈里的程序状态字PSW弹入寄存器对H，寄存器A的内容弹入H寄存器中，而8位的状态字弹入寄存器L。嗣后，寄存器L的内容可以一串(8位) 1和0的形式打印在电传打字机或显示在CRT上，供操作者检查所有五个标识位的状态。H寄存器的内容(即寄存器A的原来内容) 可用8进制数或16进制数打印出来，这要取决于所使用的输出子程序的类型。同样，其余寄存器的内容也可从堆栈取出来打印。例10-7中的指令，当到达断点的时候，把寄存器的内容压入堆栈，同时在电传打字机或CRT上打印或显示通用寄存器的内容及标识位的状态。

例 10-7 在TRAP中保存和打印寄存器的内容

```
                        /当达到
                        /断点(执行到RST5指令)时，8080
                        /转向000 050，因为使用了RST5作断点指令。
                        /在000500处执行JMP指令，使程序控制移到
                        /TRAP。
TRAP,   XTHL            /H和L压入堆栈，断点地址入H
                        /和L中。
        DCXH            /地址减1。
        SHLD            /地址保存到"BRKADD"，
        BRKADD          /因为这是到达断点时的地址。
        0               /
        PUSHD           /然后寄存器对D压入堆栈，
        PUSHB           /寄存器对B压入堆栈，
        PUSHPSW         /寄存器A和状态标识压入堆栈。
        POPH            /A和标识入H和L。
        CALL            /然后以一串1和0的数字形式打印
```

```
          BIT       /L寄存器内的8位状态标识字。
          0
          MOVAH     /H寄存器内容送入A，
          CALL      /以三位8进制数形式打印或显示在
          BINOCT    /电传打字机或CRT上。
          0
          MVID      /寄存器D置为3，因为另有三对
                    /寄存器要打印。
          003
REGOUT.POPH         /AF寄存器对入H和L。
          MOVAH     /高8位取入A，
          CALL      /并以8进制数形式打印或显示
          BINOCT    /在电传打字机或CRT上。
          0
          MOVAL     /低8位取入A，
          CALL      /以8进制数形式打印或显示
          BINOCT    /在电传打字机或 CRT 上。
          0
          DCRD      /寄存器对数目减1
          JNZ       /该数非零，则转回到REGOUT，打
          REGOUT    /印另一对寄存器。
          0
          JMP       /所有的寄存器和状态标识已打印，
          CMDDEC    /取另一个命令。
          0
BIT, MVIC           /待打印的"位"数装入C寄存器
          010
NXTBIT, MOVAL       /取"待打印"的字。
          RLC       /循环移位，最高位移入进位位，
          MOVLA     /并保存该字。
```

```
        MVIA    /ASCII O(十六进制BO)装入A寄存器。
        260
        JNC     /如进位为零，则在电传打字机或 CRT 上打印
                或显示"0"；
        ZERO
        0       /如进位为 1，则
        INRA    /打印"1"。
ZERO,   CALL    /A 寄存器内容打印在电传打字机或显示在
                CRT。
        TTYOUT
        0
        DCRC    /"位"的数目减 1。
        JNZ     /如该数为零，检查L寄存器的另一位。
        NXTBIT
        0
        RET     /8 位都已打印，返回。
```

在例 10-7 中，8080 把所有寄存器存入堆栈之后，便将PSW从堆栈弹入寄存器对H。当调用 BIT 子程序时，L寄存器中含有PSW。在BIT处，010(08) 被装入C寄存器，因为PSW中有 8 位要打印。接着三条指令所起的作用是使L寄存器的内容向左循环移位，L的最高位则移入进位位。接着把 ASCIIO 的值装入A寄存器。请记住，这条指令(MVIA) 并不影响 8080 的任何状态标识位。接着用 JNC 指令检查进位位的状态。如果进位为逻辑 0，则执行 JNC-z ERO 指令，A 寄存器中的 260 (BO) 被传送到电传打字机，并打印出 0；如果进位为逻辑 1，则不执行JNC，而是使 A 寄存器的内容加 1，变为 261(B1)，即 ASCII 1 的值，继而由电传打字机打印出 1。

打印1或0以后，C寄存器中的位计数减1。如果该数非零，则执行JNZ-NXTBIT，使L寄存器中数另一位移入进位位，并对它进行测试。L寄存器中的八位全都以1或0打印出来之后，8080从BIT子程序返回。其后，从堆栈弹入H寄存器的A寄存器的内容，用一个二进制到八进制(以ASCII为基础)的转换程序，打印在电传打字机上。

堆栈中还有其它三个寄存器对要打印。因此，A寄存器的内容打印出来以后，在D寄存器中装入数值3，即仍待打印的寄存器对的数量。然后保存在堆栈中的寄存器对内容弹入寄存器对H（REGOUT），H寄存器和L寄存器的内容以两个三位八进制数的形式打印出来。REGOUT循环每执行一次，D寄存器的内容减1。因此，REGOUT循环执行三次，电传打字机则打印出所有其余寄存器的内容。例10-8中介绍一个典型的BINOCT子程序，它非常类似于本书上册的例6-8。

例10-7中的TRAP有一个严重问题。为了打印堆栈内的值，首先要把它们从堆栈弹入寄存器。这样，在达到断点时，这些寄存器里所存在的值遭到破坏。这就是说，调试程序不可能执行继续（执行）命令。在继续执行被调试的程序以前，调试程序不可能使这些寄存器恢复原有的值。我们将简短地讨论另一种好得多的TRAP方案，用它可执行继续命令。

例 10-8 典型的8位二进制-八进制（以ASCII为基础）转换子程序

```
                    /这个子程序将A寄存器中的二进
                    /制值转换成三位以ASCII为基础的
                    /八进制数。
BINOCT, MOVCA       /二进制值保存在C寄存器。
        ANI         /最高位必须首先打印。
```

```
        300
        RLC         /最高两位循环移入最低二位。
        RLC
        CALL        /调用 BCDOUT 子程序，
        BCDOUT      /将 260 (BO) 加至 A 寄存器并
        0           /打印相加结果。
        MOVAC       /现在必须打印中间一个八进制位。
        ANI
        070
        RRC         /D5、D4和D3移入D2、D1和D0。
        RRC
        RRC
        CALL        /将 260 (BO) 加至 A 寄存器并打印
        BCDOUT      /相加结果。
        0
        MOVAC       /现打印最低一个八进制位。
        ANI
        007
BCDOUT, ADI         /260 (BO) 加至 A 寄存器并打印相
        260         /加结果。
        JMP
        TTYOUT
        0
```

断点操作

假设我们想要利用断点来调试例 10-9 所示的程序，并把断点设在第一条 OUT 指令所在处。用断点代替了这条指令之

后，程序变成例 10-10 所示的那样。

为了用断点来代替 OUT，我们应在命令中指定 OUT 指令操作码的地址，如 003 341 B。一旦设置了断点，我们就能用调试程序命令即 003 337G或 GO TO 03 DF 开始执行程序。当输入此命令时，8080 转移到 003 337 (03 DF)，并执行该存储单元中所存的指令。也就是执行 MVIA 指令，给A寄存器装入立即数据字节 231 (99)，然后执行 RST 5 指令，即断点指令。接着，8080 调用 000 050 (0028) 地址中的"子程序"。在这个存储单元里的 JMP 指令使程序控制转到 TRAP。

例 10-9 测试断点用的程序

```
        * 003 337
TEST,   MVIA    /二-十进制数 99 (八进制 231，十六进制 99)
        231     /装入 A 寄存器。
        OUT     /输出这个值到二个
        101     /七段发光二极管显示器。
        ADI     /将 1 加到这个数。
        001
        DAA     /进行十进制调整。
        OUT     /结果输出到另二个七段
        102     /发光二极管显示器。
        HLT     /暂停
```

例 10-10 插入了断点的程序

```
        * 003 337
TEST,   MVIA    /二-十进制数 99 (八进制 231，十六进制 99)
        231     /装入 A 寄存器。
        RST 5   /*****断点指令*****
        101     /七段发光二极管显示器。
        ADI     /将 1 加到这个数。
```

```
001
DAA     /进行十进制调整
OUT     /结果输出到另二个七段
102     /发光二极管显示器。
HLT     /暂停。
```

在TRAP寄存器对H的内容压入堆栈中，执行RST 5指令时存入栈内的返回地址交换到寄存器对H。然后返回地址减1并保存在BRKADD。这就是断点所在地址。接着寄存器对D和B，以及A寄存器和状态标识存入堆栈。8080在电传打字机或CRT上打印或显示状态标识位的状态和寄存器的内容。此后，8080返回到调试程序中的命令译码程序。用DBUG（8080的解释调试程序[1]）中列出的调试程序获得的打印样本如下：

```
003 341  B
003 337  G
003 341
SZ1 P2   A   B   C   D   E   H   L   M   S   P
01000110 231 000 000 000 000 000 000 024 177 372
           C   S
         107 125
```

调试程序的TRAP部分（见例10-7）把断点从样本程序清除掉了吗？原来的OUT指令代替了断点指令（RST 5）吗？否。当然，为用OUT指令代替RST 5指令（即清除断点），我们可输入K命令（见例10-5）。但是，在用断点来调试一个程序时，有时我们会忘记清除断点。一个极为简单的方法是改写调试程序的TRAP部分，使之在8080转回到指令译码程序（CMDDEC）之前，调用CLEARB子程序来清除断点。例

10-11 REGOUT 循环以后的指令正是干这事的。在 10-11 中，我们没有示出调试程序的全部 TRAP 的内容，而仅列出其中打印寄存器内容的指令序列（最后部分）、清除断点的指令和转回到指令译码程序的指令。

例 10-11 清除执行后的断点

```
        :
        MOVAL   /低 8 位取入 A，
        CALL    /以 8 进制形式打印
        BINOCT  /在电传打字机或显示在 CRT 上。
        0
        DCRD    /寄存器对的数目减 1。
        JNZ     /该数非零，则转回到 REGOUT，
        REGOUT  /以打印另一对寄存器。
        0
        CALL    /把原来的指令写回到
        CLEARB  /被调试的程序，清除断点。
        0
        JMP     /所有的寄存器和状态标识已打印，
        CMDDEC  /取另一命令。
        0
CLEARB, LHLD    /断点地址装入寄存器对 H。
        TEMPO
        0
        LDA     /原来存放在这个地址的指令操作
        ORGIN   /码装入寄存器 A。
        0
        MOVMA   /该操作码存入断点地址，然后
        RET     /返回。
```

寄存器内容非破坏性打印

如前所述，调试程序的 TRAP 部分在打印寄存器内容时破坏了这些寄存器内容。出现这种情况的原因是堆栈内的数据弹入了寄存器。现在再一次利用例 10-9 的程序，假定断点设在第一条 OUT 指令上，并且让 8080 从地址 003 337 开始执行程序。执行到断点时，寄存器的内容被打印出来，断点则被清除（OUT 指令写回到程序中）。然后 8080 返回到调试程序内所包含的指令译码程序。现在我们想要把断点设在第二条 OUT 指令处，并令 8080 从第一条 OUT 指令开始继续执行这一程序。这就是说，我们想从设置最后一个断点的地方继续执行程序。经过这些指令之后，我们就能观察出 OUT 101 指令的作用。

为了达到这个目的，现在需要从头（003 337，03 DF）开始重新执行程序。在我们的测试程序中，要做到这一点并不困难，但在某些程序则很难，因为程序可能很长，而且与实时事件有关。

由于前一个断点的地址放在 TEMPO(两个读/写存储单元)内，微型计算机不难确定从什么地方“重新开始”执行正在被调试的程序。然而，要继续执行程序，还必须恢复到达第一个断点时所有状态标识和寄存器的内容。但是，自堆栈内容弹入寄存器并打印在电传打字机上以后，调试程序的 TRAP 部分(例 10-7）并没有为使 8080 可以继续执行程序而保存这些寄存器的内容。我们必须做到既打印出寄存器的内容而同时仍在栈内保存着它们。为此，我们不是从堆栈取一个 16 位字到两个寄

存器内，把它们打印出来，然后再把同一个 16 位字压回栈内。如果这样做了，那么，很难从堆栈取出任何别的 16 位字。当然，寄存器内容之所以要保存在堆栈内，首先是因为转换和输出子程序要使用它们。“堆栈”实际上是读/写存储器的一部分，因而还可用其它的方法来存取压入栈内的寄存器内容。

例 10-12 中指令将 8080 的寄存器内容压入堆栈，然后将堆栈指示器的读/写存储器地址装入寄存器对 H。现在，8080 用 MOV 型指令便能把一个值从存储器读进一个通用寄存器中，然后根据电传打字机或 CRT 打印或显示的数的形式，调用 BIT、BINOCT 或 BINHEX 子程序。MOV 型指令不影响堆栈，因而很易做到既打印出寄存器内容和状态标识位的状态，而同时又把这些信息保存在栈内。让我们来检查堆栈的内容并看一看寄存器的内容在堆栈中实际存储顺序。我们可编写一个能存取这些值并能将它们打印或显示在电传打字机或 CRT 上的程序段。假设寄存器对 H 是第一个压入堆栈的寄存器对，PSW 最后压入。

在表 10-2 中我们假定在执行调试程序 TRAP 程序段内的任何 PUSH 型指令之前，堆栈指针指向存储器地址 024 100 (1440)。如果现在执行例 10-12 中的指令，那么当执行到 D-ADSP 指令时，寄存器对 H 里含有什么样的存储器地址？

例 10-12 使用寄存器对 H 以存取寄存器

```
/现在调试程序中的“TRAP”程序段将堆栈指示器加到
/寄存器对 H。存储器访问指令可用来存取栈内的寄存器值而不致
/破坏堆栈。
TRAP,   XTHL    /H 和L 入堆，断点地址入 H 和 L。
        DCXH    /地址减 1。
        SHLD    /地址存入“BRKADD”，因为这是断点
```

```
0          /的地址。
PUSHD      /然后将寄存器对 D、B、
PUSHB      /寄存器 A 和标志压入堆栈。
PUSHPSW/
LXIH       /将000000(0000)装入寄存器对 H。
000
000
DADSP      /SR 加到寄存器对 H。
.
.
```

令有 024 070(1438)，即存着PSW的状态标识字的存储单元的地址。从表 10-2 可见，我们不能简单地从寄存器对 H 所寻址的存储单元读一个值，然后打印在电传打字机上并使存储器地址加 1。因为我们要打印的顺序是状态字和寄存器 A、B、C、

表 10-2 寄存器存入堆栈的顺序

八进制堆栈地址		十六进制堆栈地址	
最初的 SP	024 100		1440
	024 077	寄存器 H	143 F
	024 076	寄存器 L	143 E
	024 075	寄存器 D	143 D
	024 074	E	143 C
	024 073	B	143 B
	024 072	C	143 A
	024 071	A	1439
现在的 SP	024 070	状态标识字	1438

D、E、H 和 L。这不是使用 PUSH 指令写入存储器的顺序。存取这些八位的值和以所要求的顺序打印它们的指令序列如例 10-13 所示。

例 10-13 中的指令也许有点难读，但的确能使 8080 按要求的顺序在电传打字机上打印出寄存器内容。要记住的重要之点是，由于用 MOV 型指令从存储器读寄存器内容，故在电传打字机上打印或在 CRT 上显示以后，这些内容仍存在存储器里。同样，堆栈指针的值也没有改变。

例 10-13 当达到断点时，按要求的顺序打印寄存器的内容

```
        /现在调试程序的"TRAP"部分将堆栈指示器加至寄
        /存器对 H。于是可用访问存储器指令访问"堆栈"内的
        /寄存器值而又不致扰乱堆栈。
TRAP,   XTHL    /寄存器对 H 的内容入栈，断点
                /地址入寄存器对H。
        DCXH    /地址减 1。
        SHLD    /此地址存入"BRKADD"，因为这是到达断点时
                 的地址。
        0
        PUSHD   /寄存器对 D 压入堆栈，
        PUSHB   /寄存器对 B 压入堆栈，
        PUSHPSW /寄存器 A 和状态标识位压入堆栈。
        LXIH    /现在数值 000 000(0000)装入
        000     /寄存器对 H。
        000
        DADSP   /堆栈指示器加到寄存器对 H。
        MOVAM   /现在从存储器读状态标识字，并
        CALL    /以 1 和 0 数字形式打印出来。
        BIT     /这是以前见到过的"BIT"子程序的
```

```
          0          /新方案。
          INXH       /寄存器对 H 内的地址加 1。
          MOVAM      /取出 A 寄存器的内容，
          CALL       /以三位八进制数形式打印出来。
          BINOCT
          0
          MVID       /把 003 装入寄存器对 D，
          003        /还有三个寄存器对待打印。
REGPR,    MVIE       /一个寄存器对内的寄存器数(2 个)
          002        /装入 E 寄存器。
          INXH       /存储器地址两次加 1。
          INXH
NXTREG,   MOVAM      /从存储器取一个八位字，
          CALL       /并以八进制数形式打印在
          BINOCT     /电传打字机上或显示在 CRT 上。
          0
          DCXH       / 存储器地址减 1。
          DCRE       /寄存器数减 1。
          JNZ        /如寄存器数不为 0，则打印
          NXTREG     /寄存器对中另一寄存器的内容。
          0
          INXH       /已打印整个寄存器对，
          ZNXH       /故使存储器地址加 2。
          DCRD       /寄存器对的数目减 1。
          JNZ        /如该数不为 0，则应
          REGPR      /打印另一个寄存器对的内容。
          0
          CALL       /全部打印完毕，
          CLEARB     /从程序中清除断点(RST 5)。
          0
```

```
        JMP     /取另一个命令。
        CMDDEC
        0
BIT     MOVDA   /8 位数入寄存器 D。
        MVIC    /待打印的"位"数入寄存器 C。
        010     /
NXTBIT, MOVAD   /取"待打印"的字。
        RLC     /最高位移入进位位。
        MOVDA   /移位结果存入寄存器 D。
        MVIA    /ASCII 0 (十六进制 B 0)的值送寄
        260      存器 A。
        JNC     /如进位位为 0，则在电传打字机
        ZERO    /或 CRT 上打印或显示 0。
        0       /如进位位为 1，则将 260 加 1 而受
        INRA    /为 261(由 B 0 到 B 1)。
ZERO,   CALL    /在电传打字机或 CRT 上
        TTYOUT  /打印或显示寄存器 A 的内容。
        0
        DCRC    /"位"数减 1。
        JNZ     /如该数不为 0，则检查
        NXTBIT  /寄存器 D 中的另一位。
        0       /如寄存器D中的八位都已打印
        RET     /则返回。
```

应记得，POP 型指令的特点之一是，可从存储器里非破坏性地读取数值。即是既可从堆栈取出这些值打印，而同时仍能把它们保存在读/写存储器里。然而，一旦寄存器的内容已被打印后，假如调用任何子程序的话，那么返回地址会存入堆栈，从而破坏寄存器的内容。因此，较为方便的方法是堆栈指示器

加到寄存器对 H(其中置入初值 0)，然后用 MOV 型指令读存储器里的数值。如果在这之后调用任何子程序，那么返回地址将被保存在堆栈中寄存器内容和状态字的下方。

给调试程序添加一个“继续”命令

由于所有寄存器的内容在电传打字机上打印出来以后，仍然保存在堆栈里，继续执行命令，使寄存器恢复到最初的值，然后转移到断点地址重新开始或继续执行被调试的程序。断点地址储存在 BRKADD 中(见例 10-3)。

当“继续”命令输入微型机并被调试程序中的命令译码程序检测到时，8080 转移到例 10-14 中的 CONTIN。在 CONTIN 中，以适当的顺序把 PSW 和寄存器对 B、D 的内容弹出堆栈(这些寄存器是以例 10-13 中的指令序列所确定顺序压入栈里)。达到断点时的寄存器对 H 里的原有 16 位值仍然保存在栈内。然后给寄存器对 H 装入断点地址，即被调试程序中下一条应执行指令的地址。在例 10-9 和例 10-10 中，这是第一个 OUT 指令。记住，当到达断点时，这条指令未被执行；再者，在寄存器的内容被打印出来以后，调试程序的 TRAP 部分清除断点。断点地址被装入寄存器对 H 之后，XTHL 指令把这个地址送入堆栈，并使寄存器对 H 恢复到达断点时所具有的值。然后，RET 指令使 BRKADD 从堆栈弹入程序计数器(PC)中。这样，全部寄存器都恢复到原来的值，8080 即可继续执行被调试的程序。

我们能用 BREAK(例 10-3)、TRAP(例 10-13) 和 CONTIN(例 10-14) 指令序列在离原来断点 1、2 或 10 条指令的地

方设置新的断点，然后利用继续执行指令观察这 1、2 或 10 条指令对寄存器内容的影响吗？是的，这是可能的。让我们来确定必须完成的操作(命令)。在某一指令上设置断点以后，调试程序使程序控制转到被调试程序的开头。当执行到断点时，寄存器的内容被打印出来。然后，用BREAK 指令序列在离最后一个断点几条指令的地方设置一个断点。BREAK 指令序列把设置断点的地址存在 TEMPO中。接着将继续执行命令输入调试程序，开始执行 CONTIN 指令序列。这些指令把堆栈保存的所有寄存器的值从堆栈弹入相应寄存器，接着使程序控制转到最后一个(或者说上一个) 断点地址上所存的那条指令。因此，BRKADD里含有程序继续执行的起始地址，而TEMPO则存放设置断点的那个地址。认清这两个地址之间的差别是非常重要的。

当执行TRAP中的指令(例 10-13) 来保存寄存器的内容，这些寄存器保存在哪一个栈里呢？这是一个非常困难的问题。毫无疑问，调试程序启动以后，它首先执行 LXISP 指令。然而，被调试的程序一开始也可能有一个LXISP 指令。因此，如果用调试程序的GO命令从头开始执行被调试的程序，那么SP中的内容最后将取决于被调试程序中的那条LXISP 指令。因此，在TRAP指令序列中，寄存器将被保存在属于正被调试的那个程序的堆栈。出现这种情况的原因是，被调试的程序执行了第二条LXISP 指令。

例 10-14 使 8080 继续执行程序的指令序列

```
/"继续"命令使 8080 从最后一个断点的地址开始
/继续执行用户程序。
CONTIN, POPPSW  /寄存器 A 和状态标识出栈,
        POPB    /寄存器对 B 出栈,
```

```
POPD     /寄存器对 D 出栈。
LHLD     /寄存器对 H 内装入
BRKADD   /最后一个断点所在的地址。
0        /
XTHL     /该地址压入堆栈，寄存器对 H 恢
RET      /复原来内容，然后转到断点地址。
```

这会产生新的问题。寄存器内容存入堆栈和被打印以后，8080 返回调试程序中的命令译码程序，等待输入和执行另一个命令。这就是说，如果执行任何调用指令的话，那么返回地址将保存在用户程序建立和使用的堆栈内。当执行返回指令的时候，这些返回地址从用户栈弹出。但是，如果在输入“继续”命令之前，调试程序没有从堆栈清除其数据值和返回地址，那么，这些数据值就会在执行 CONTIN 时从堆栈弹出，并被当作为“原来的”寄存器值。当然，一个程序必须保持一个“清洁的”堆栈，但要找出调试程序中的所有错误可能是困难的。你如何调试一个调试程序呢？

调试程序使用用户建立的堆栈所引起的另一个更重要的问题是，用户也许没有给堆栈分配足够的空间，既能存程序的数据值和返回地址，又能存调试程序的数据值和返回地址。假设调试程序需用 15 级堆栈（30 个读/写存储单元）。如果用户只给堆栈分配 10 级（20 个读/写存储单元，这由用户程序中 LXISP 指令的第二字节和第三字节确定），那么，堆栈数据就可能越出栈界而写到用户的程序或数据值上。从而使用户的程序“丢失”。解决这个矛盾的唯一途径是设置两个独立的堆栈，一个供用户程序用，另一个是供调试程序用。但任何时刻只能用其中的一个堆栈。

设置两个独立的堆栈，会增加调试程序断点软件（TRAP

和 CONTIN)的复杂性，但那是必须付出的代价。当执行调试程序时，就使用调试程序的堆栈；当 8080 执行用户程序时，则使用用户堆栈。为此，必须在 TRAP 中执行堆栈交换指令，使计算机从用户堆栈转到调试程序的堆栈，而在 CONTIN 中，使 8080 从调试程序的堆栈转到用户堆栈。这种新的 TRAP 指令如例 10-15 所示。

例 10-15 带有堆栈转换指令的新 TRAP

```
/现在调试程序的 TRAP 部分把所有的寄存器保
/存到堆栈上。堆栈指 示器 SP 也存入读/写存储器。
/然后设置一个新堆栈，专供调试程序用。
TRAP,  XTHL     /H 和 L 入栈，断点地址入 H、L。
       DCXH     /地址减 1。
       SHLD     /此地址存入"BRKADD"，因为
       BRKADD   /这是已达到断点的地址。
       0
       PUSHD    /保存寄存器对 D，
       PUSHB    /保存寄存器对 B，
       PUSHPSW  /寄存器 A 和状态标识入栈。
       LXIH     /000 000 (0000)送入寄存器对 H。
       000
       000
       DADSP    /SP 加到寄存器对 H。
       SHLD     /"用户 SP"存入读/写存储器，以备后用。
       USERSP   /寄存器对 H 仍含有
       0        /SP 的内容
       LXISP    /然后设置一个新的堆栈供调试程序用。
       STACK    /现在再没有任何用户寄存器会
       0        /被越出控制范围的堆栈所冲掉。
       MOVAM    /从存储器读入标识字，并以 1
```

```
CALL    /和0的形式打出它。
BIT     /这是前面见到的"BIT"子程序
0       /的新方案。
.
.
```

在TRAP中增加二条指令，就可在所有寄存器压入用户堆栈(占用四级即八个读/写存储单元)以后，用户堆栈的SP加到寄存器对H，并用SHLD指令存入两个读/写存储单元。执行SHLD指令以后，寄存器对H仍包含有SP的内容，因此，用寄存器对H中的地址，仍然可从读/写存储器读出寄存器的内容。其后，调试程序执行LXISP指令，专门为调试程序建立一个新的堆栈。现在调试程序完成的任何与堆栈有关的操作(即调用、返回、压入或弹出)均将使用这个栈而不用用户栈。值得注意的是，调试程序仍用了用户堆栈保存寄存器的内容。

与此同时，调试程序的CONTIN部分也必须作相应改动，以便从用户栈弹出寄存器的内容之前，调试程序回到这个栈。新的CONTIN指令序列如例10-16所示。在此例子中，用户SP从名为US-ERSP的两个读/写存储单元读入寄存器对H中。当执行SPHL指令时，这个值送入堆栈指示器。然后PSW和寄存器对B和D的内容从堆栈进入各自的目的地。接着BRKADD的内容送入寄存器对H。BRKADD是已达到的断点的地址，也是8080必须由之开始继续执行用户程序的地址。然后，XTHL指令将存在堆栈里的寄存器对H原来的内容和最后一个断点的地址相交换，RET指令又把这个断点地址送入程序计数器。

例 10-16 带有堆栈转换指令的CONTIN

```
/用户堆栈地址已存在读/写存储器，
```

```
/"继续"命令在从堆栈取寄存器和继续执行程序
/之前，必须将这一地址装入 SP。
CONTIN, LHLD     /用户堆栈指示器装入寄存器对 H。
        USERSP
        0
        SPHL     /然后将此值装入 SP。
        POPPSW   /从堆栈弹入 A 和状态标识。
        POPB     /从堆栈弹入寄存器对 B
        POPD     /和寄存器对 D。
        LHLD     /寄存器对 H 内装入断点地址
        BRKADD   /(从此地址开始继续执行)。
        0        /与堆栈内的
        XTHL     /寄存器对 H 交换。
        RET      /断点地址"BRKADD"弹入程序计数器。
```

对 TRAP 的修改导致用用户堆栈的四级（八个存储单元）来保存各种寄存器的内容。但是，在某种场合下，由于缺少可用的读/写存储单元，连这么小的用户堆栈也没法提供。也就是说，寄存器的内容必须保存在第三个堆栈或调试程序的堆栈里。如果这样做了，TRAP 和 CONTIN(例 10-15 和例 10-16)就得修改。最简便的解决方法是用户给堆栈分配更多的存储区。倘若办不到这一点，寄存器的值可存在调试程序的堆栈里。

单步——一次执行一条指令

假设有一个例 10-17 那样的程序存在读/写存储器中。如果我们轻易就能做到一次执行一条指令，然后观察这条指令对通

用寄存器的影响，那会非常方便。这意味着我们把断点设在一个特定指令上，然后执行程序，考察各寄存器的内容。这之后，把断点往前移一条指令，并输入“继续”命令。

例 10-17　被调试的样本程序

```
/这个程序用作单步执行的例子。
        *003 000
START, LXISP    /读/写存储器地址
        300     /装入堆栈指示器。
        003
        LXIH    /存储器地址(043 030=2318)装
        030     /入寄存器对 H。
        043
        MOVAM   /从存储器取一个 8 位值。
        ADDB    /寄存器 B 的值加到这个值上。
        OUT     /结果输出到输出口
        203     /203(83)。
        HLT     /停止执行程序。
```

为了单步(一次一条指令)地执行例 10-17 的程序，我们首先把断点设在存储单元 003 003(0303)里的 LXIH 指令上，并从003 000(0300) 存储单元开始执行这个程序。当执行到断点时，LXISP 指令已设置了用户堆栈。全部寄存器的内容被打印出来。然后，应将断点设在存储单元 003 006 (0306) 的 MOVAM 指令上，并输入“继续”命令。当打印出全部寄存器的内容之后，就可以观察 LXIH 指令对寄存器的影响。为了一步步执行程序的其余部分，断点就应依次设在存 MOVAM、ADDB、OUT 和 HLT 等指令操作码的存储单元。在每条指令上设断点，并使用调试程序的“继续”指令，我们可以单步通过

整个程序（即一次执行一条指令，并观察它对寄存器的影响）。

这个过程的一种限制是，我们必须确定用于存储每条指令的操作码的存储单元的地址。如果不熟悉 8080 的指令操作码，或者调试某个别的程序，那么，这种单步执行指令的过程可能冗长而乏味。比这完善和复杂得多的一种方法是编写一个可用来计算下一条待执行指令的地址的 8080 程序。这就是说，调试程序中的指令必须能够确定一条指令是单字节、双字节还是三字节的。例如，该调试程序必须“知道”MVIA 指令是双字节指令，而 JMP 指令则为三字节指令。

在表 10-3 中，我们列出了 8080 的所有单字节、双字节和 3 字节指令及其八进制操作码。须注意，其中某些指令列为一类指令。例如，全部 MOV 型指令系单字节的指令，因此，把它们统一写成 MOVDS(D 为目的寄存器，S 为源寄存器)。

已经每条指令的“长度”，8080 很易于求出下一个断点应该移动到的地址。设 x 为最后一个断点的地址，我们需要单步通过一两字节的指令，则下一个断点应该在地址 x+2 上。对于三字节的指令，断点地址应为 x+3。这没有考虑执行转移、调用、重新启动、返回或 PCHL 指令的可能性。为使调试程序尽可能简单，我们暂且不管这些具有转移程序控制（程序执行）能力的指令。

如果要单步通过例 10-17 中的程序，我们可将断点设在存有 LXIH 指令操作码的地址 003 003（0303）处，然后利用调试程序的 G 或 GO 命令，从地址 003 000（0300）开始执行程序。执行 LXISP 指令之后便达到断点。因为 LXISP 指令不影响 8080 的任何通用寄存器，我们难以预料到达断点时这些寄存器中所存的内容。当 STEP 命令输入调试程序的时候，首先

表 10-3 8080 的单、双和三字节指令

单 字 节 指 令

助记符	八进制操作码		助记符	八进制操作码
MOVDS	1 DS	D=S=0,1,2,3,4,5,6,7	RLC	007
			RRC	017
ADDS	20 S		RAL	027
ADCS	21 S		RAR	037
SUBS	22 S		HLT	166
SBBS	23 S		NOP	000
ANAS	24 S		EI	373
XRAS	25 S		DI	363
ORAS	26 S			
CMPS	27 S		STC	067
			CMC	077
			DAA	047
INRS	0S4		CMA	057
DCRS	0S5			
PUSHRP	3R5	R=0,2,4,6	XCHG	353
POPRP	3R1	R=0,2,4,6	XTHL	343
INXRP	0R3	R=0,2,4,6	PCHL	351
DCXRP	0R3	R=1,3,5,7	SPHL	371
STAXRP	0R2	R=0,2	RET	311
LDAXRP	0R2	R=1,3		
DADRP	0R1	R=1,3,5,7	条件返回	3X0
RSTn	3N7	N=0,1,2,3,4,5,6,7	X=0,1,2,3,4,5,6,7	

续表

双字节指令			
助记符	八进制操作码	助记符	八进制操作码
IN	333	ADI	306
OUT	323	ACI	316
		SUI	326
MVID	0D6 D=0,1,2,3,4,5,6,7	SBI	336
		ANI	346
		XRI	356
		ORI	366
		CPI	376

三字节指令			
助记符	八进制操作码	助记符	八进制操作码
JMP	303	LXI	0R1
条件转移	3X2		
CALL	315	STA	062
条件调用	3X4	LDA	072
	X=0,1,2,3,4,5,6,7	SHLD	042
		LHLD	052
		R=0,2,4,6	

应在地址 003 006 (0306) 处设断点，然后 8080 执行存在存储器中原断点地址即 003 003 (0303)处的指令。这意味着当 8080 要单步通过指令时，它必须从原断点地址中读指令操作码，并根据这个操作码确定新断点应该在离它 1 个、2 个还是 3 个存

储单元的地方。一旦在新地址放置了新断点，立即执行“继续”指令序列，使所有的寄存器恢复到原来的值，然后执行原断点地址中的指令。

为了使调试程序简单，我们不让它单步通过转移、调用、返回、重新启动或 PCHL 指令。仔细观察 8080 指令的八进制操作码，很易看出 8080 如何确定每条指令的字节数。

例 10-18 中的 STEP 指令序列完成以下任务：确定待执行指令的字节数量，紧接着这条指令之后设置一个断点，恢复用户堆栈和寄存器的值，然后转移到被调试程序中的指令。执行了这条指令以后，8080 被 RST 5 指令引导到 TRAP。当设立了第一个断点和将调试程序的 G 命令输入微型计算机之后，就能使调试程序单步通过一条指令。为此，我们在电传打字机上或 CRT 上按下 S 键（输入单步命令 STEP）。调试程序中的命令译码程序将控制转向 STEP（例 10-18 所示）。于是 8080 确定存在原断点地址上的指令操作码的字节数。如果调试程序不能单步通过某一特定指令，那么电传打字机或 CRT 打出一个疑问号(？)，然后返回到调试程序中的命令译码程序。一旦出现这种情况，就必须用手工方法，使断点越过控制转移指令。值得注意的是，STEP 中的一些指令组成一个指令译码程序，其功能类似于一个简单的命令译码程序。

例 10-18 确定 8080 每条指令的字节数目

```
/调试程序的这一部分确定 8080 每条指令的字
/节数。然而某些指令不能用单步法执行，
/包括转移、调用、返回、重新启动和 PCHL 指
/令，即任何转移程序控制的指令。
STEP,   LDA      /原来存在存储器中断点地址内的
        ORGIN    /指令操作码被装入
```

```
0           /A寄存器。
MOVBA       /还把此操作码装入B。
LHLD        /已达到的断点的地址
BRKADD      /装入寄存器对H。
0
CPI         /这指令是 OUT 吗?
323
JZ          /是的，则它为双字节指令。
TWOBYT
0
CPI         /是 IN 指令吗?
333
JZ          /是的，则它是双字节指令。
TWOBYT
0
CPI         /是 JMP 指令吗?
303
JZ          /是的，则不能单步通过它，
CANTDO      /电传打字机或 CRT
0           /打印或显示?。
CPI         /是调用指令 CALL 吗?
315
JZ          /是的，则不能单步通过它，
CANTDO      /电传打字机或 CRT 打印或显示?。
0
CPI         /是返回指令 RET 吗?
311
JZ          /是的，我们不能单步通过它，
CANTDO      /电传打字机或 CRT 打印
0           /或显示?。
```

```
CPI         /是 PCHL 指令吗?
351
JZ          /是的，则不能单步通过它，
CANTDO      /电传打字机或 CRT 打印
0           /或显示?。
ANI         /仅 D 7、D 6、D 2、D 1
307         /和D0 保留在A寄存器。
CPI         /是重新启动指令之一吗?
307
JZ          /是的，不能单步通过它，
CANTDO      /电传打字机或 CRT 打印或显示?。
0
CPI         /是立即算术逻辑指令吗?
306
JZ          /是的，则为双字节指令。
TWOBYT
0
CPI         /是立即传送指令吗?
006
JZ          /是的，则为双字节指令。
TWOBYT
0
CPI         /是 STA、LDA、SHLD 或 LHLD
002         /(直接送数)指令吗?
JZ          /可能是，再检查。
DLOAD
0
CPI         /是 LXIB、LXID、LXIH 或
001         /LXISP 指令吗?
JZ          /可能是，再检查。
```

```
          LXI
          0
          MOVAB     /操作码取回A寄存器
          ANI       /指令是条件转移、调用或返回指
          301       /令(3X2、3X4、
          CPI       /3X0)吗?
          300
          JNZ       /不是，则为单字节指令。
          ONEBYT
          0
CANTDO,MVIA         /不能单步通过这一特定指令。
          277
          CALL      /电传打字机或CRT打印或显示?
          TTYOUT
          0
          JMP       /然后转到命令译码程序，输入和解
          CMDDEC    /释另一个命令。
          0
LXI,      MOVAB     /可能是LXI型的指令。
          ANI       /再检查。
          010
          JNZ       /不是LXI型指令。
          ONEBYT    /则为单字节指令。
          0
THRBYT,INXH         /是三字节指令。
TWOBYT,INXH         /是双字节指令。
ONEBYT,INXH         /是单字节指令。
SPCSTP,   CALL      /在寄存器对H寻址的存储单元内
          SETBRK    /设置一个断点
          0         /(例10-3)。
```

```
        LHLD    /用户堆栈指示器装入寄存器对H。
        USERSP
        0
        SPHL    /此地址装入堆栈指示器。
        POPPSW  /从堆栈弹入寄存器A和状态标识。
        POPB    /弹入寄存器对B。
        POPD    /弹入寄存器对D。
        LHLD    /新断点地址装入寄存器对H。
        TEMPO
        0
        PUSHH   /新的"BRKADD"压入堆栈。
        LHLD    /程序继续执行的地址
        BRKADD  /装入寄存器对H。
        0
        XTHL    /新的断点地址BRKADD入H和L，老的断点
                 地址入堆栈，
        SHLD    /然后新断点地址存入读/写存储器，
        BRKADD  /供以后使用。
        0
        POPH    /老的断点地址BRKADD从堆栈入H和L。
        XTHL    /将它与寄存器对H交换。
        RET     /老的断点地址入程序计数器PC。
DLOAD,  MOVAB   /取指令操作码送A。
        ANI     /仅保留D3位。
        010
        JN 2    /不是042、052、062或072
        ONEBYT  /(22、2A、32或3A)，则必为
        0       /单字节指令。
        JMP     /是LDA、STA、LHLD或SHLD指令，
        THRBYT  /则作相应处理。
```

在例 10-18 的 STEP 处，8080 向寄存器 A 和寄存器 B 装入断点地址中所存的指令操作码。SETBRK 子程序在 ORGIN 中保存有指令操作码(见例 10-3)。然后程序对断点地址中的指令操作码和一个立即数据字节进行一系列比较。这些立即数据字节代表了 8080 的许多指令的操作码。通过比较可以确定，这条指令能否单步通过，如果可以单步执行，则还要确定其字节数为多少。

前二次比较是对要单步通过的指令操作码和 323、333 进行对比。323 和 333 为 OUT 和 IN 指令的八进制操作码。如果指令操作码等于这二个值中的任何一个，那么 8080 就转移到 TWOBYT，因为 OUT 和 IN 都是二个字节的指令。如果两者不相等，则把指令操作码与无条件转移指令 JMP 的操作码进行比较。如果相等，8080 转移到 CANTDO 中，因为这一个调试程序不能单步通过 JMP 指令。事实上，只要指令操作码等于任何一个无条件或有条件转移、调用或返回指令，8080 就将转移到 CANTDO。同样，如果指令操作码等于重新启动或 PCHL 指令操作码的话，8080 就在电传打字机上或 CRT 打印或显示一个问号(？)，然后再返回到调试程序中的命令译码程序。这向用户表明，8080 不能单步通过这条指令。

值得留意的是，8080 不必将断点地址中的指令操作码同所有无条件和有条件转移、调用和返回指令的操作码进行比较。这是因为 8080 指令的各种操作码有某些“一致性”。例如，所有条件转移指令的操作码中的 D7、D6、D2、D1 和 D0 位是相同的，都为：1、1、X、X、X、0、1、0。将断点地址上的指令操作码和 307 相与，再将所得结果与 302 相比较，如

果指令操作码等于任何一条条件转移指令的操作码，那么零标识位为逻辑1。8080也能够将这条被屏蔽的（即和307相与后的）指令操作码同300相比较，以确定指令是否为条件返回指令，或同304相比较，确定指令是否为条件调用。用八进制表示8080指令操作码，容易检查指令。事实上，通过执行ANI 301和CPI 300指令序列，8080可确定操作码是否属于条件指令（应仔细，这是决窍！）。

如果8080能够单步通过指令，那么它必定执行THRBYT、TWOBYT和ONEBYT三条指令中的一条。因为寄存器对H中含有原来的断点地址（参看STEP中的第三条指令LHLD BRKADD），这些指令将这一地址按要求一次次地加1，从而使寄存器对H指向存储器中下一条"可执行"指令的操作码。然后调用SETBRK子程序，将断点设置在这个存储单元里。这意味着将这个存储单元内的指令的操作码存入ORGIN中，然后在其中设置RST 5指令。

设置断点之后，其地址暂时存入TEMPO，以便可用寄存器对H来保存用户堆栈指针内的地址。用户堆栈含有寄存器B、C、D、E、H、L、A和状态标识位的值。这些值从用户堆栈取出，用来供8080执行"继续"命令用。寄存器对H的值留在堆栈内，因为在8080实际继续执行用户程序之前，还要再次使用寄存器对H。

为将新的断点地址存入存储单元BRKADD和BRKADD+1，恢复寄存器对H的内容和继续执行用户程序，8080使用下列五个不同的程序步：

1．将新的断点地址压入堆栈；

2．给寄存器对H装入老的断点地址（8080继续执行程序的地址）；

3．交换堆栈和寄存器对H的值，把继续执行的地址存入堆栈内，新的断点地址装入寄存器对H之中；

4．接着用新的断点地址代替BRKADD中的老断点地址；

5．继续执行地址置入寄存器对H，然后与栈顶内容进行交换。这样寄存器对H恢复成到达最后一个断点时的值。同时，继续地址存在栈顶，当执行返回指令RET时，8080可转到该地址继续执行。

这是程序中较复杂的一部分，也许要反复读几次才能读通。如果怀疑这一指令序列的可行性，可在纸上设立一个想象的堆栈，将老的断点地址存在BRKADD处，然后在纸上从SPCSTP开始执行程序。

8080返回到用户程序后，在遇到新断点中的RST 5指令以前，它将执行老断点地址中的指令。当执行RST 5指令时，8080再一次转移到TRAP，打印出寄存器内容，从而显示单步执行一条指令的影响。

单步通过控制转移指令

怎样编写一个调试程序，使之能够单步通过上述的控制转移指令？对于这个问题，要在此刻说明白是太难了。然而设想一下它的做法，应该是比较容易的。假如我们从存储器读到一条无条件转移或调用指令（JMP或CALL）的操作码，那么指令的第二和第三字节应用作为下一个断点的地址；假如读出的是RET(无条件返回)指令的操作码，那么存在用户堆栈中寄存器对上面的子程序返回地址，应该作为新的断点地址。对于PCHL指令，寄存器对H的内容（它也保存在用户堆栈里，参见

例 10-15 中符号地址 TRAP 内的指令），应该作为新的断点地址。为了单步通过重新启动指令（不是调试程序断点指令所用的重新启动指令），不难算出 8080 应“转向”的地址（例 10-19）。

例 10-19 计算重新启动指令的向量地址

```
/调试程序的这个部分根据被执行的 RST 指令计算新的断点地址。
RSTIN, MOVAB    /RST 指令操作码传送到 A 寄存器。
       ANI      /只保留 D 5、D 4 和D 3 位，即变为
       070      /00 XXX 000
       MOVLA    /将此值存寄存器 L。
       MVIH     /000(00)置入 H 寄存器。
       000
       JMP      /然后在此地址(指 HL 内的地址)
       SPCSTP   /设置断点和继续执行程序。
       0        /
```

仅当从存储器中的老断点地址上读出八进制操作码 3 x 7（x 可为 0～7 以内的任意值）时，8080 才执行调试程序的RSTIN 部分。RSTIN 的 16 位地址应写入 STEP 程序段内的指令译码程序中，直接置于 CPI 307 指令之后。CANTDO 的地址应改为 RSTIN 指令（见例 10-18）。指令操作码从寄存器 B 读入寄存器A之后，ANI 指令使D7、D 6、D 2、D 1 和 D 0 各位置 0。如果要单步通过 RST 3 指令，那么从存储器读出操作码 337。ANI 指令被执行之后，寄存器A 中存有 030(18)。然后，这个值被装入寄存器 L 中，寄存器H 被清零。当转移到 SPCSTP 时，8080 便将断点设在寄存器对 H 所寻址的存储单元中。这就是当执行 RST 3 指令时，8080 将要“转向”的那个存储单元。由此可见，使用八进制操作码，很容易看出 8080 是如何正确确定下一个断点地址的。

条件转移、调用和返回指令是最难单步通过的指令。可采用的一个方法是设置两个断点0一个紧跟在上述指令之后；另一个断点是设在程序控制能够转到的存储单元中。如果不执行条件转移指令，那么8080将到达转移指令之后的断点；如果执行了条件转移指令，那么8080将到达设置在控制可能转向的存储单元里的那个断点。这种方法比较复杂，因为这意味着调试程序不得不设两个断点，而且当8080到达其中的一个断点时，还得把两个断点同时从程序清除掉。下面我们不准备对这种方法作进一步的讨论。

第二个方法，也许是最出色的方法（我们希望它是最好的方法），就是从条件转移、调用或返回指令操作码中“产生”一个条件转移指令操作码。条件指令转换为条件转移指令后就存入读/写存储器区域里。接着这个操作码之后在存储器中还必须存一个16位地址，这个地址是如果执行条件转移，就要被执行的指令序列的地址。在条件转移指令之后，调试程序还应该在存储器中存储一个无条件转移指令。这条指令的转移地址是不执行条件转移指令时将要执行的指令序列的地址。

接着，8080必须将用户堆栈地址装入堆栈指示器，将用户的状态标识装入PSW(POPPSW)。8080之所以要做这一步，目的是使产生的条件转移指令能测试用户标识。8080执行了POPPSW指令之后，就转移到包含着所产生的条件转移指令的那个读/写存储单元。然后，它不是执行条件转移指令就是执行无条件转移指令。

如果执行条件转移指令，8080就转回到调试程序中的一个特定点。如果8080回到了调试程序的这个部分，那么用户程序中的条件指令就会被执行，要是我们单步通过了这一条件指令的话。因此，8080确定条件转移或调用的目的地址或条件

返回的地址。断点就被设在这个地址上，然后 8080 执行“继续”指令序列。接着，执行用户程序中的条件指令，把 8080 引导回调试程序。

如果 8080 产生的条件指令不被执行，那就执行存在它后面的无条件转移指令。8080 转移回到调试程序，确定用户程序内存在条件指令之后的那条指令的地址。然后，8080 就在这个地址上设一断点，并执行“继续”命令。这样，8080 来执行用户程序中的条件指令，而是到达了设在存储器中该条件指令之后的断点，然后回到调试程序。

调试程序只要执行三条指令就可把全部条件指令转换成条件转移指令，如例 10-20 所示。这是因为条件转移、调用和返回指令的操作码非常相似。所有 24 条这种指令的 D 7 和 D 6 位都是 1，而 D 5、D 4 和 D 3 位代表待测试的条件。也就是说，要测试 8 个状态，即符号、零、进位和寄偶标识位的 1 和 0 的状态。D 5、D 4 和 D 3 三位的各种组合及其代表条件可参见表 10-4。

表 10-4 可测试的条件及其代码

条 件	代 码		
	D 5	D 4	D 3
NZ	0	0	0
Z	0	0	1
NC	0	1	0
C	0	1	1
PO	1	0	0
PE	1	0	1
P	1	1	0
M	1	1	1

例 10-20 全部条件指令转换成条件转移指令

```
/调试程序的这一部分将所有的条件转移、调用和返回指令转成条件转移
指令。
CONVRT, MOVAB/取条件指令的操作码到寄存器 A。
        ANI     /仅保存 D 7、D 6、D 5、D 4 和 D 3 位。
        370     /(370=F 8)。
        ADI     /然后加 002(02)。
        002     /条件转移指令的操作码现在在A
        .       /寄存器中。
        .
        .
```

不论什么样的条件指令(转移、调用或返回)，表 10-4 中的三位表示相同的条件。也就是说，对于 JNZ、CNZ 和 RNZ 指令说来，操作码中的 D 5、D 4 和 D 3 三位有相同的状态。低三位(D 2、D 1 和 D 0)确定了指令是条件转移、调用还是返回指令。表 10-5 列有这三类指令的低三位的组合。

表 10-5 条件指令操作码中表示转移、调用和返回的位

指令型号	D2	D1	D0
条件转移	0	1	0
条件调用	1	0	0
条件返回	0	0	0

作为一个例子，假设我们要单步通过 CNZ 指令。当执行 STEP 指令时，CNZ 的操作码(304，C 4)读入寄存器 A。接着执行 ANI 301 和 CPI 300 指令(见例 10-18)，8080 转移到 CONNRT 中，将 CNZ 指令的操作码与 370 相与，结果 300 留在寄存器 A。再将 2 加到 A，得 302 (C 2)，这就是 JNZ 指令的操作码。8080 将这个操作码同 16 位地址和无条件转移指

令(JMP)一起存入存储器中，结果如下：

```
JNZ
ALPHA
0
JMP
BETA
0
```

这些指令被存储在读/写存储器里之后，调试程序就向 SP 装入用户堆栈的地址并从堆栈弹出状态标识。然后 8080 转移到含有 JNZ 指令的存储单元。

如果 JNZ 指令测试条件得到满足，调试程序便转移到 ALPHA，以确定 8080 转移控制的适当地址。在此例中，因为调试程序单步通过 CNZ 指令，所以 8080 将把新的断点设在 CNZ 指令第二和第三字节所指定的存储单元。

如果测试条件不满足，调试程序将转到 BETA，断点就设在 CNZ 指令后面一个存储单元内所存的下一个指令操作码上。然后 8080 使所有用户寄存器的内容从堆栈弹出并转移到被调试程序中的 CNZ 指令。用户的状态标识位的状态决定了 8080 不执行 CNZ 指令，而是到达 CNZ 指令之后的存储单元中所存的那个断点。

8080 利用这种方法确定测试的条件（状态标识），进而确定条件是否满足。然后在调试程序的控制下判断下一个断点应设在二个可能地址中的哪一个；确定下一个顺序操作码的地址或控制转向的地址。鉴于完成这些操作所需的全部软件非常复杂，本书不准备提供程序清单。

简单的调试程序

鉴于我们已经讨论了简单调试程序需要的各种“模块”或子程序，现在我们将把它们同命令译码程序连接在一起组成一个程序。这个程序仅仅是一个调试程序，它没有提供修改程序和存储器的能力。此调试程序具有 5 条命令，列在表 10-6 中。

表 10-6　简单调试程序的命令

1. 在一地址中设断点；
2. 消除断点；
3. 执行存储器中的程序；
4. 单步通过一条指令(除控制转换指令之外)；
5. 从最后一断点处继续执行程序。

正如我们前面已讨论过的那样，这个调试程序不能单步通过某些指令(如转移、调用、重新启动、返回以及 PCHL)。当执行调试程序的 TRAP 部分时，用户堆栈将被保护起来。可是，全部用户寄存器将保存在用户堆栈里，调试程序将建立它自己用的堆栈。这个调试程序的汇编语言指令列在例 10-21 中。

例 10-21　简单的调试程序

```
TYCHON    EDITOR-ASSEMBLER V-2
          DW  TEMPO   070 106
          DW  USERSP  070 104
          DW  STACK   070 000
```

```
    DW  ORGIN   070 100
    DW  BRKADD 070 102
```

/此调试程序具有5个命令。如输入“S”命令，调试程序将单步操作；如输入“C”命令，8080将从最后一个断点起继续执行程序；如输入“K”命令，则清除断点；在输入“B”命令(设置断点)或“G”命令(开始执行程序)后也可输入16位地址。

```
                    *100  000
100000061   DEBUG,LXISP  /装入堆栈指示器，
100001000   STACK        /建立调试程序自己用的堆栈。
100002070    0
100003076   IGNOR, MVIA  /在电传打字机或CRT上
100004077   “?”          /打印或显示问号“?”。
100005315   CALL
100006104   TTYOUT
100007100   0
100010315   CMDDEC, CALL/打印回车或换行。
100011120   CRLF
100012100    0
100013315   CALL         /从电传打字机或CRT取一个
100014073   TTYIN        /字符。
100015100    0
100016376   CPI          /输入的是“S”命令吗？
100017123   “S”
100020312   JZ           /是的，则转单步。
100021011   STEP
100022101    0
100023376   CPI          /输入的是“C”命令吗？
100024103   “C”
100025312   JZ           /是的，则继续执行程序
```

```
100026375  CONTIN
100027100  0
100030376  CPI          /输入的是"K"命令吗?
100031113  "K"
100032312  JZ           /是的，清除此断点。
100033266  KILLBP
100034100  0
100035000  NOP
100036000  NOP
100037000  NOP
100040000  NOP
100041000  NOP
100042021  LXID         /不是"S"、"C"或"K"，试将ASCII
100043003  003          /值看作一个八进制数。
100044000  000
100045315  CALL         /再输入两个八进制数
100046140  SPCOCT
100047100  0
100050142  MOVHD        /结果送入H。
100051315  CALL         /然后再取一个三位的八进制数。
100052132  OCTIN
100053100  0
100054152  MOVLD        /结果送入L。
100055315  CALL         /然后从电传打字机或
100056073  TTYIN        /CRT取另一个字符。
100057100  0
100060376  CPI          /继六位八进制数后输入的是"B"命令
                        吗?
100061102  "B"
100062312  JZ           /是的，则在寄存器对H寻址
```

```
100063233   BREAK              /的存储单元内设断点。
100064100   0
100065376   CPI                /继六位八进制数后输入的是
100066107   "G"                /"G"命令吗?
100067302   JNZ                /否，则取另一个命令。
100070003   IGNOR
100071100   0
100072351   GO, PCHL           /寄存器对H的内容装入PC。
100073333   TTYIN, IN          /输入UART的状态位。
100074001   001
100075346   ANI                /只保存接收器的标志。
100076001   001                /如A＝001，则有键按下。
100077312   JZ                 /如A＝000，无键按下。
100100073   TTYIN              /等待，直到键被按下。
100101100   0
100102333   IN                 /键被按下，输入ASCII字符
100103000   000                /到A寄存器。
100104107   TTYOUT,MOVBA       /该字符存入B。
100105333   TTYO, IN           /输入UART的状态字。
100106001   001
100107346   ANI                /只保存发送器的标识位。
100110004   004                /如A＝004，则发送器已准备好。
100111312   JZ                 /如A＝000，则发送器忙。
100112105   TTYO               /在打印A的内容之前，等待发送器
                                （打字机）工作结束。
100113100   0
100114170   MOVAB              /字符从B传送到A之后，
100115323   OUT                /输出到UART。
100116000   000
100117311   RET                /返回，字符仍留在寄存器A中。
```

```
100120076  CRLF, MVIA    /回车符的 ASCII 值送
100121215  215           /A寄存器，然后打印它。
100122315  CALL
100123104  TTYOUT
100124100  0
100125076  MVIA          /然后换行符的 ASCII 值送 A寄
100126212  212           /存器并打印它。
100127303  JMP
100130104  TTYOUT
100131100  0
100132021  OCTIN, LXID   /000 003(0003)送寄存器对D。
100133003  003           /E内含 003(03)，
100134000  000           /D内含000(00)。
100135315  OCTINI, CALL  /从电传打字机或 CRT 取一个字
100136073  TTYIN         /符(它将打印在电传打字机
100137100  0             /或显示在 CRT 上)并返回。返回时字符
                         在A中。
100140376  SPCOCT, CPI   /其值小于 ASCIIO 吗？
100141060  060
100142332  JC            /是的，置之不理。
100143135  OCTINI
100144100  0
100145376  CPI           /等于或大于 ASCII 8 的值吗？
100146070  070
100147322  JNC           /是的，置之不理。
100150135  OCTINI
100151100  0
100152346  ANI           /好，它为 ASCII0 到 ASCII 7，
100153007  007           /除 D2、D1 和 D0 三位外，其余各位均
                         置 0。
```

```
100154107   MOVBA          /数暂时存放在寄存器B。
100155172   MOVAD          /取前一位并循环左移三
100156007   RLC            /次。这样将增大其有效值，并
100157007   RLC            /为刚输入的数腾出空间。
100160007   RLC
100161200   ADDB           /加刚才输入的数。
100162127   MOVDA          /新得的二进制数存寄存器D。
100163035   DCRE           /位计数器减1。
100164302   JN2            /计数不为0，
100165135   OCTINI         /取另一个字符。
100166100   0
100167076   SPC, MVIA      /打印空格。
100170240   240
100171303   JMP
100172104   TTYOUT
100173100   0

100174117   BINOCT,MOVCA /数值保存在C。
100175346   ANI
100176300   300
100177007   RLC              /两个最高有效位移入最低位。
100200007   RLC
100201315   CALL             /调用BCDOUT子程序，
100202226   BCDOUT           /加260(B0)，以转换A的内容
100203100   0                /并打印它。
100204171   MOVAC            /现在该打印中间一位。
100205346   ANI
100206070   070
100207017   RRC              /把它循环移入3个最低位。
100210017   RRC
```

```
100211017   RRC
100212315   CALL            /把260(B0)加到A的内容上，
100213226   BCDOUT          /并打印结果。
100214100   0
100215171   MOVAC           /现打印右面的一位。
100216346   ANI
100217007   007
100220315   CALL
100221226   BCDOUT
100222100   0
100223303   JMP             /在数之后打一空格，
100224167   SPC
100225100   0
100226306   BCDOUT, ADI     /把260(B0)加到A寄存器上。
100227260   260
100230303   JMP             /在电传打字机或 CRT 上打印
100231104   TTYOUT          /或显示 ASCII 值。
100232100   0

100233315   BREAK, CALL     /在寄存器对H所寻址的存储
100234254   SETBRK          /单元设置断点。
100235100   0
100236076   MVIA            /JMP指令的操作码送A寄
100237303   303             /存器。
100240062   STA             /将此操作码存在8080在执
100241050   050             /行断点指令时将转向的地
100242000   000             /址内。
100243041   LXIH            /转移指令高位和低位地址
100244304   TRAP            /字节送寄存器对H。
100245100   0               /
```

```
100246042  SHLD              /将此16位地址存入JMP指令
100247051  051               /操作码后的存储单元。
100250000  000
100251303  JMP               /设置断点之后，转回到调
100252010  CMDDEC            /试程序的命令译码程序。
100253100  0

100254176  SETBRK,MOVAM      /取"原来"的指令并把字存入
100255062  STA               /读/写存储器。
100256100  ORGIN
100257070  0
100260066  MVIM              /然后把重新启动指令写到
100261357  357               /这条指令的位置上。
100262042  SHLD              /断点地址存入
100263106  TEMPO             /"TEMPO"。
100264070  0
100265311  RET               /从SETBRK返回。

100266315  KILLBP, CALL      /从程序清除断点，
100267274  CLEARB
100270100  0
100271303  JMP               /然后返回调试程序中的命令
100272010  CMDDEC            /译码程序。
100273100  0

100274052  CLEARB,LXIH       /设有断点的地址送寄存器对
100275106  TEMPO             /H。
100276070  0
100277072  LDA               /将原来在这个地址内的指令
100300100  ORGIN             /操作码装入A寄存器。
```

```
100301070   0
100302167   MOVMA          /将操作码存在断点地址内，
100303311   RET            /然后返回。
/现在调试程序的"TRAP"部分将把SP加到寄存器对
/H，这样，就可用访问存储器指令来存取堆栈内的寄存器
/值而又不致扰乱堆栈。
100304343   TRAP,XTHL      /H和L入栈，断点地址入H和L。
100305053   DCXH           /断点地址减1。
100306042   SHLD           /断点地址入
100307102   BRKADD         /"BRKADD"。
100310070   0
100311325   PUSHD          /然后保存寄存器对D
100312305   PUSHB          /和寄存器对B，
100313365   PUSHPSW        /最后保存A和状态标识。
100314041   LXIH           /000 000 (0000) 送入寄存器对H。
100315000   000
100316000   000
100317071   DADSP          /SP加至HL。
100320042   SHLD           /用户SP存入读/写存储器，以备后
                           用。
100321104   USERSP         /寄存器对H仍存有SP的值。
100322070   0
100323061   LXISP          /然后为调试程序设一新堆栈。
100324000   STACK          /现在用户寄存器
100325070   0              /不再会被越出控制范围的堆栈所冲
                           掉。
100326315   CALL           /在电传打字机或CRT上打印或
100327120   CRLF           /显示回车换行符。
100330100   0
100331176   MOVAM          /从存储器读标识字，并以
```

```
100332315  CALL          /1和0形式打印出来。
100333176  BIT           /这是前面见到"BIT"子程序的
100334101  0             /新方案。
100335043  INXH          /寄存器对H内的地址加1。
100336176  MOVAM         /取A寄存器内容，
100337315  CALL          /以三位八进制数形式打印出
100340174  BINOCT        /来。
100341100  0
100342026  MVID          /003送D寄存器。
100343003  003           /有三个寄存器对要打印。
100344036  REGPR, MVIE   /一个寄存器对内的寄存器数
100345002  002           /送E寄存器。
100346043  INXH          /存储器地址加1。
100347043  INXH
100350176  NXTREG,MOVAM/从存储器取一个八位字，
100351315  CALL          /以八进制数形式打印在电传
100352174  BINOCT        /打字机或显示在CRT上。
100353100  0
100354053  DCXH          /存储器地址减1。
100355035  DCRE          /寄存器数减1。
100356302  JNZ           /如寄存器数不为0，则打印
100357350  NXTREG        /寄存器对内另一寄存器的内
100360100  0             /容。
100361043  INXH          /已打印整个寄存器对，
100362043  INXH          /故使存储器地址加2。
100363025  DCRD          /寄存器对的数目减1。
100364302  JNZ           /如此数不为零，则应打印
100365344  REGPR         /另一对寄存器的内容。
100366100  0
100367315  CALL          /全部打印完，
```

```
100370274   CLEARB          /从程序中清除断点（RST5）。
100371100   0
100372303   JMP             /取另一个命令。
100373010   CMDDEC
100374100   0
/用户SP已存入读/写存储器，在从堆栈接受寄存器和继续执行程序之
  前，"继续"命令应将这个地址送堆栈指示器。
100375052   CONTIN,LHLD     /用户SP的值送寄存器对H。
100376104   USERSP
100377070   0
101000371   SPHL            /将此值装入SP。
101001361   POPPSW          /从堆栈弹出A和状态标识、
101002301   POPB            /寄存器对B和
101003321   POPD            /寄存器对D。
101004052   LHLD            /断点地址（继续执行的开始地址）送
101005102   BRKADD          /寄存器对H。
101006070   0
101007343   XTHL            /寄存器对H与栈顶交换内容。
101010311   RET             /断点地址"BRKADD"弹入程序计数
                             器。

/调试程序的这一部分确定每一条8080指令的字节数。但某些指令不能
/用单步执行，这包括转移、调用、返回、重新启动和PCHL指令（即
/转移控制的任何指令）。

101011072   STEP, LDA       /原来存在存储器中断点地址
101012100   ORGIN           /内的指令操作码取入
101013070   0               /寄存器A。
101014107   MOVBA           /此操作码又送入B。
101015052   LHLD            /已到达的断点的地址
```

```
101016102  BRKADD      /送寄存器对H。
101017070  0
101020376  CPI         /是OUT指令吗?
101021323  323
101022312  JZ          /是，则为双字节指令。
101023133  TWOBYT
101024101  0
101025376  CPI         /是1N指令吗?
101026333  333
101027312  JZ          /是，则为双字节指令。
101030133  TWOBYT
101031101  0
101032376  CPI         /是JMP指令吗?
101033303  303
101034312  JZ          /是，则不能单步通过它。
101035003  IGNOR       /在电传打字机或CRT上打印
101036100  0           /或显示问号"? "。
101037376  CPI         /是CALL指令吗?
101040315  315
101041312  JZ          /是，不能单步通过。
101042003  IGNOR       /在电传打字机或CRT上打印
101043100  0           /或显示问号"? "。
101044376  CPI         /是RET指令吗?
101045311  311
101046312  JZ          /是，不能单步通过。
101047003  IGNOR       /在电传打字机或CRT上打印
101050100  0           /或显示问号"? "。
101051376  CPI         /是PCHL指令吗?
101052351  351
101053312  JZ          /是，不能单步通过。
```

```
101054003  IGNOR      /在电传打字机或CRT上打印
101055100  0          /或显示问号"？"。
101056346  ANI        /仅将D7、D6、D2、D1和D0各
101057307  307        /位保存在A寄存器。
101060376  CPI        /是重新启动指令吗？
101061307  307
101062312  JZ         /是，不能单步通过。
101063003  IGNOR      /在电传打字机或CRT上打印
101064100  0          /或显示问号"？"。
101065376  CPI        /是立即算术或逻辑运算指令
101066306  306        /吗？
101067312  JZ         /是，则为双字节指令。
101070133  TWOBYT
101071101  0
101072376  CPI        /是立即传送指令吗？
101073006  006
101074312  JZ         /是，则为双字节指令。
101075133  TWOBYT
101076101  0
101077376  CPI        /是STA、LDA、SHLD或LHLD（直
101100002  002        /接取数）指令吗？
101101312  JZ         /或许是，再次检查。
101102165  DLOAD
101103101  0
101104376  CPI        /是LXIB、LXID、LXIH或LXISP 指
101105001  001        /令吗？
101106312  JZ         /或许是，再次检查指令操作
101107124  LXI        /码。
101110101  0
101111170  MOVAB      /操作码送回寄存器A。
```

```
101112346   ANI                 /是条件转移、调用或返回指
101113301   301                 /令（3X2、3X4或3X0）吗？
101114376   CPI
101115300   300
101116302   JNZ             /否，则指令必为单字节指令。
101117134   ONEBYT
101120101   0
101121303   JMP                 /是条件转移、调用和返回指
101122003   IGNOR               /令，8080 不能单步通过。
101123100   0
101124170   LXI,  MOVAB         /也许是 LXI 类指令。
101125346   ANI                 /再次检查。
101126010   010
101127302   JNZ                 /不是 LXI 型指令，
101130134   ONEBYT              /故为单字节指令。
101131101   0
101132043   THRBYT,   INXH  /是三字节指令。
101133043   TWOBYT,   INXH  /是双字节指令。
101134043   ONEBYT,   INXH  /是单字节指令。
101135315   SPCSTP,   CALL  /在寄存器对 H 寻址的存储单
101136254   SETBRK              /元设置断点。
101137100   0
101140052   LHLD                /用户 SP 的值送寄存器对 H。
101141104   USERSP
101142070   0
101143371   SPHL                /此地址送 SP。
101144361   POPPSW              /从堆栈弹入 A 和状态标识位，
101145301   POPB                /从堆栈弹入寄存器对 B
101146321   POPD                /和寄存器对D
101147052   LHLD                /才设置了断点的那个地址
```

```
101150106    TEMPO  /送寄存器对H。
101151070    0
101152345    PUSHH  /新设置的断点地址"BRKADD"入
                    栈。
101153052    LHLD   /继续执行的起始地址送寄存
101154102    BRKADD/器对H。
101155070    0
101156343    XTHL   /新断点地址入H和L，老断点地址入栈内。
101157042    SHLD   /然后将新的断点地址存入
101160102    BRKADD/读/写存储器，以备后用。
101161070    0
101162341    POPH   /老断点地址弹入H和L。
101163343    XTHL   /栈顶与寄存器对交换。
101164311    RET    /老的断点地址入PC。

101165170DLOAD  MOVAB /指令操作码入寄存器A。
101166346    ANI    /仅保存D3位。
101167010    010
101170302    JNZ    /不是042、052、062或072
101171134    ONEBYT /(22、2A、32或3A)，则必为单
101172101    0      /字节指令。
101173303    JMP    /是LDA、STA、LHLD或SHLD指令，
101174132    THRBYT /则作相应处理。
101175101    0

101176016  BIT，  MVIC      /待打印的"位"数送寄存器C。
101177010       010
101200127       MOVDA     /数保存在D。
101201172  NXTBIT，MOVAD   /取"待打印的"字。
101202007       RLC       /最高位循环移入进位标识位。
```

```
101203127          MOVDA /然后存这个字。
101204076          MVIA  /ASCII 0 的值（十六进制 BO）
101205260          260   /送 A 寄存器。
101206322          JNC   /如进位为 0，则在电传打字
101207212          ZERO  /机或 CRT 上打印或显示"0"；如
101210101          0     /进位为 1，则打印"1"。
101211074          INRA
101212315ZERO,     CALL  /在电传打字机或 CRT 上打印。
101213104          TTYOOT/A 寄存器内容。
101214100          0
101215015          DCRC  /"位"数减 1 。
101216302          JNZ   /如不为零，则测 L 寄存器的
101217201          NXTBIT /另一位。
101220101          0
101221303          JMP   /打印出八个 0 和 1 后打印一
101222167          SPC   /空格。
101223100          0
```

例 10-21 的调试程序，由于我们想使它尽可能保持简短，因而存在许多"错误"。当 8080 开始执行这个调试程序时，如果输入单步或继续执行命令，会出现什么情况呢？8080 将用最后一个断点地址作为起点，进行单步操作或继续执行程序。这个地址是存在 BRKADD 中的 16 位地址。因为我们刚启动调试程序，无法知道起初存入 BRKADD 中的是什么样的地址。因此，只有在至少使用一次断点之后，才可执行单步和继续执行命令。此外，除非输入了一个调试程序绝对达不到的断点，否则不要输入清除断点命令（K）。

关于调试程序的最后几点意见

在我们讨论过的调试程序中，没有一个能够用来调试只读存储器(ROM)中的程序。这是因为8080不能把断点指令（在我们的例子中为RST 5)写进ROM 程序中。

例10-21中的调试程序也还有许多可改进的地方。例如可以使调试程序打印出已达到的断点的地址。做到了这一点，就易于把寄存器的内容同被调试程序中的具体指令联系起来。

在大多数程序中，寄存器对H用于存储器的寻址。因此，当达到断点的时候，除了寄存器内容外，寄存器对H所寻址的存储单元内容也应被打印出来，这可以简化用存储器存储数据的程序的调试。如果需要的话，也可以将寄存器对B和D所寻址的存储单元内容打印出来。同样，堆栈指示器的16位内容也可以和栈顶上最后一个或二个16位值一起打印出来，不管这些值是表示数据还是返回地址。

在所有这些信息都被打印出来时，如在状态字和寄存器内容的上方打印出状态标识和寄存器的名称，那也会是有益的。我们在本章这一节中介绍了一些特点，其中许多已在DBUG中反映出来。

在例10-22中介绍了简单的汇编语言程序和利用DBUG单步通过此程序的结果。

例 10-22 利用DBUG单步通过一个程序

(A)样本程序的汇编清单

```
TYCHON EDITOR-ASSEMBLER V-2                    PAGE 01-001
        *020 000
```

```
020000061 START, LXISP   /把地址装入堆栈指示器，
020001200        200     /以便在到达断
020002004        004     /点时把寄存器内容保存起来。
020003041        LXIH    /读/写存储器地址送寄存器对 H
020004100        100
020005020        020
020006066        MVIM    /将操作码后的立即数据字节
020007233        233     /存入此存储单元。
020010076        MVIA/立即数据字节送寄存器A。
020011001        001
020012053        DCXH    /寄存器对H内容减1。
020013206        ADDM    /存储器内容加到寄存器A。
020014117        MOVCA   /此值存入C寄存器。
020015014        INRC    /C寄存器加1，
020016014        INRC    /再加二次
020017014        INRC
020020166        HLT     /暂停
```

（B）利用 DBUG 得到的结果

```
020 000 B
020 000 G
020 000
SZ I P 2  A   B   C   D   E   H   L   M   S P    G S
01000110 000 000 000 000 000 000 000 024 004 200 000 135

S 020 003
01000110 000 000 000 000 000 020 100 137 004 200 000 135

S 020 006
01000110 000 000 000 000 000 020 100 233 004 200 000 135

S 020 010
01000110 001 000 000 000 000 020 100 233 004 200 000 135

S 020 012
01000110 00. 000 000 000 000 020 077 375 004 200 000 135
```

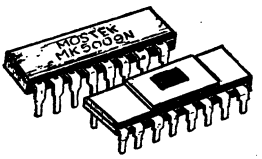
```
S 020 013
10000010 376 000 000 000 000 020 077 375 004 200 000 135

S 020 014
10000010 376 000 376 000 000 020 077 375 004 200 000 135

S 020 015
10000110 376 000 377 000 000 020 077 375 004 200 000 135

S 020 016
01010110 376 000 000 000 000 020 077 375 004 200 000 135

S 020 017
00000010 376 000 001 000 000 020 077 375 004 200 000 135
```

读者知道，当8080机复位时，它就从存储器地址000000开始执行程序。如果系统监控和调试程序从这个地址开始存入存储器，那么在用作断点指令的重新启动指令的地址中就不需要写入转移指令。可是，当系统监控和调试程序存储在ROM中的时候，必须在ROM中编入转移指令。

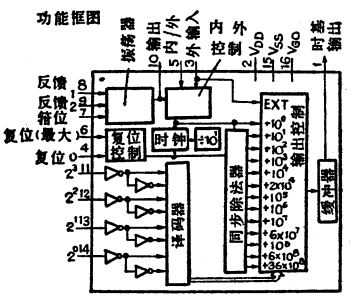
在列举的所有调试程序的示例中，我们用了RST 5指令作为断点指令。这意味着中断设备不能产生RST 5指令。我们在第三章中还曾谈到，RST 0指令不如其他重新启动指令通用，因为它起着硬连线复位的作用。由于这个原因，某些调试程序把RST 0指令作为断点指令使用。这就是说，当8080复位并当到达被调试程序中的断点指令RST 0的时候，它从地址000000开始执行程序。因此，每当微型计算机复位和到达程序中的断点时，将寄存器内容压入用户堆栈，然后打印它们的指令将被打印出来。

还有许多调试程序的特点，我们未加讨论。这包括多个(同时的)断点、中断服务子程序的实时调试（在子程序运行中调试)，和当第341次通过循环时出现“错误”的情况下程序的调试。此外，还有ROM中的程序调试问题。可惜这些专题都已超出本书论述的范畴。

附录 A MOSTEK 公司 MK 5009 计数器时间基准电路



功能框图



MOS 计数器时基电路

- □ 采用离子注入工艺，与 TTL/DTL 完全兼容。
- □ 内时钟工作于：
  - 外部信号
  - 外部 RC 网络
  - 外部晶体
- □ 工作频率：直流～1 MHz 以上。
- □ 频率选择用二进制编码

说明

MK 5009 是一种多功能的 MOS 振荡器和除法链，它是由 MOStek 公司采用耗尽型负载、离子注入工艺和P-沟道技术制造而成的。MOS5009是双列直插式封装组件，有16条引线，提供的分频范围是1～36×10^8。MK 5009 可以分别用三种频率源工作：内部振荡器与外部 RC 电路组合；内部振荡器与外部晶体组合；或者用外加的 TTL 信号。另外，控制输入端为 MK 5009提供多用性，使它具有多种用途，其中包括仪表、定时器和时钟。

MOS 5009 的输入频率高达 1 MHz，所以它能为大多数频率测量仪器提供所需要的时间周期，即 1 μs～100s。如果采用其 1 MHz 的输入频率，则它可适用于 1 分、10 分和 1 小时定时。如果使用1/1.2MHz 输入，MK 5009 P 也可以为袖珍仪表和时钟产生精确的线性频率，提供50/60Hz 的输出频率。

时基输出(Time out)是方波，它的频率是由所选择的计数器分频、振荡器频率或外来输入决定的。输出方波的下降沿可以用来控制外部控制电路。

时 基 输 出

地址输入	不复位	复位		旁路方式		
$2^3\,2^2\,2^1\,2^0$	$R_{最大}=0$ $R_0=0$	复位最大 $R_{最大}=1$ $R_0=0$	复位最小 $R_{最大}=0$ $R_0=1$	方式 1 $R_{最大}=V_{GG}$ $R_0=0$	方式 2 $R_{最大}=0$ $R_0=V_{GG}$	方式 3 $R_{最大}=V_{GG}$ $R_0=V_{GG}$
0 0 0 0	$\div10^0$	$\div10^0$	$\div10^0$	$\div10^0$	$\div10^0$	$\div10^0$
0 0 0 1	$\div10^1$	复位	复位	$\div10^1$	$\div10^1$	$\div10^1$
0 0 1 0	$\div10^2$			$\div10^2$	$\div10^2$	$\div10^2$
0 0 1 1	$\div10^3$			$\div10^3$	$\div10^3$	$\div10^3$

续表

地址输入	不复位	复位		旁路方式		
	$R_{最大}=0$	复位最大 $R_{最大}=1$	复位最小 $R_{最大}=0$	方式1 $R_{最大}=V_{GG}$	方式2 $R_{最大}=0$	方式3 $R_{最大}=V_{GG}$
$2^3 2^2 2^1 2^0$	$R_0=0$	$R_0=0$	$R_0=1$	$R_0=0$	$R_0=V_{GG}$	$R_0=V_{GG}$
0 1 0 0	$\div 10^4$	计数器	计数器	$\div 10^4$	$\div 10^4$	$\div 10^4$
0 1 0 1	$\div 10^5$			$\div 10^2$	$\div 10^5$	$\div 10^2$
0 1 1 0	$\div 10^6$	接时基电路	时基电路	$\div 10^3$	$\div 10^6$	$\div 10^3$
0 1 1 1	$\div 10^7$			$\div 10^4$	$\div 10^7$	$\div 10^4$
1 0 0 0	$\div 10^8$	最高	最低	$\div 10^5$	$\div 10^5$	$\div 10^3$
1 0 0 1	$\div 6\times 10^7$			$\div 6\times 10^4$	$\div 6\times 10^4$	$\div 6\times 10^1$
1 0 1 0	$\div 36\times 10^8$	状态	状态	$\div 36\times 10^5$	$\div 36\times 10^5$	$\div 36\times 10^2$
1 0 1 1	$\div 6\times 10^8$			$\div 6\times 10^5$	$\div 6\times 10^5$	$\div 6\times 10^2$
•	—			—	—	—
1 1 1 0	$\div 2\times 10^4$			$\div 2\times 10^1$	$\div 2\times 10^1$	$\div 2\times 10^1$
1 1 1 1	外部输入	外入	外入	外、内	外、内	外、内

* 不论复位最大和复位0输入端的状态如何，地址1100和1101在输出端都产生逻辑0。

逻辑1＝高＝V_{SS}

逻辑0＝低＝V_{DD}

绝对最大额定值

任何端子相对于 V_{ss} 的电压……＋0.3V～－20V

工作温度范围(环境温度) ……0℃～＋70℃

存储温度范围(环境温度) ……－55℃～＋150℃

推荐的工作条件

(0℃≤T_A≤70℃)

参　　数	最　小	典型值	最　大	单　位	注
V_{SS} 电源电压	+4.5		+5.5	V	
V_{DD} 电源电压	0.0		0.0	V	
V_{GG} 电源电压	−9.6		−14.4	V	
f_{XTAL} 晶体频率	0.1		2.0	MHz	
f_{RC} RC 频率	DC		200	kHz	
f_{EXT} 外部频率	DC		2.0	MHz	
t_{PL} 逻辑 0 脉冲宽度，$\overline{CLAMP}$	—				注 5
外部输入	200			ns	
t_{PH} 逻辑 1 脉冲宽 度外部输入	200			ns	
复位最大	10.0			μs	
复位 0	10.0			μs	
R 反馈电阻	.01		2.5	MΩ	图 1
V_{IL} 输入电压，逻辑 0，复位输入	0.0		0.8	V	
复位(旁路方式)	V_{GG}		$V_{GG}+1.0$	V	注 2
其它逻辑输入			0.8	V	
V_{IH} 输入电压，逻辑1，其它逻辑输入	$V_{SS}-1.0$	V_{SS}	$V_{SS}+0.3$	V	

电气特性

(V_{SS} = +5V±10%；V_{DD} = 0V；V_{GG} = −12.0V±20%；0℃ ≤T_A≤70℃)

参　　数	最　小	典型值↑	最　大	单　位	注
I_{SS}电源电流，V_{SS}		6.0	11.0	mA	注1
I_{GG}电源电流，V_{GG}		6.0	11.0	mA	
I_{IL} 输入电流，逻辑0			−1.6	mA	注2 V_I=0.4
V_{OL} 输出电压，逻辑0			0.4	V	I_{OL}=1.6mA*
V_{OH} 输出电压，逻辑1	2.4			V	I_{OH}=−40μA*
f_{STA} 频率稳定性W/电压变化，RC式		±3.0		%/V	注3
/温度变化RC式		−0.2		%/C	
/晶体式		—			注4
$t_{..}$不稳定性边缘-边缘变化		<15		ns	温度与电源电压不变

V_{SS} 的典型值＝+5V，V_{DD}=0V，V_{GG}=−12V，T_A=25°C

1. V_{SS} 逻辑输入，输出开始；当为逻辑0时，每个逻辑输入(参考注2)使 I_{SS} 增加1.6 mA(最大)。

2. 逻辑输入是：复位最大；复位0；地址输入；外部输入；外部/内部选择和$\overline{\text{CLAMP}}$。

3. 频率的变化只随电源变化。

4. 晶体方式的稳定性取决于晶体。

5. 在$\overline{\text{CLAMP}}$输入端的最小逻辑0时间是振荡器周期的50%。

* V_{OH}，V_{OL}只提供时基输出。

操作说明

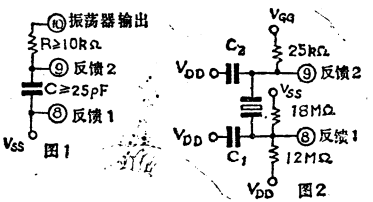
MK 5009 P 基本上是由一串计数器组成的；通过内部多路转换可以选择计数器。$\div 10^1$ 计数器的输出用来为 10^2 到 36×10^8 计数器级产生内部时钟信号；这些计数器级是完全互相同

步的。

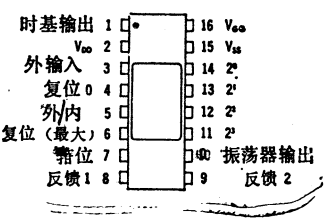
振荡器的控制

RC 振荡器方式操作的实现如图 1 所示。频率 f 大约为 0.8/RC。在RC 振荡器中可用箝位电路，提供一个启动脉冲或精确的启动操作。当$\overline{\text{Clamp}}$(箝位非)为逻辑 0 时，内部电路保持为参考电平，因此$\overline{\text{Clamp}}$复原(回到逻辑1)，振荡器的第一个周期将是全周期。

晶体振荡器方式如图 2 所示。为了得到最佳性能，选择电阻值来为内部电路提供偏置。选择两个电容器作为选择的晶体规定的负载电容(C_L)。建议 $C_1=C_2=2C_L$。



引线连接



复位/旁路控制

MK 5009 兼有两种不同的复位控制。复位 0 的 10 μs 或更长的正向脉冲将使计数器复位，回到它们的最低状态；而复位

最大的正向脉冲使计数器复位到它们的最高状态。不论选择哪种除法链，复位最大控制能使用户建立计数器，在下一个振荡周期或外部负向输入时提供下降沿。

另外，把一个或两个复位输入置到最负电压 V_{GG}，则允许除法链的旁路成分用来测试或作其它用途(参考时基输出表)。

外部和内容频率源

当使用外来信号源操作 MK 5009 P 时，这个信号应该加到外部输入端(引线 3)；外部和内部选择端(引线 5)应该置逻辑 1。

用外部信号操作时，外部/内部选择端子应该置逻辑0。

振荡器输出

振荡器输出(引线10产生的)不是真逻辑输出，但是可以用来驱动高阻抗器件，例如结型场效应晶体管或其它 MOS 电路。

附录 B 微型计算机接口

准 备 程 序

摆在微型计算机许多用户面前的问题之一是为解决具体应用问题准备程序。最后一个表给出的程序例子是够简短的，已经把代码放在一块了，即用人工进行了汇编；也就是说每个助记符翻译成了八进制、十六进制或二进制等效值。因为计算机程序简短，地址按顺序列在草稿上，所以容易增添或改变转移、调用和输入/输出设备的地址。可是，并不是所有的程序准备都是这么容易。许多应用程序可能长达几千条指令。这里从适宜微型计算机的程序开发的助记符开始讨论。

程序开发的最大问题之一就是对问题简要说明以及它的解法。在程序设计开始之前，必须把所有的希望结果、输入、输出和完整的程序流（包括各个判定步）都必须认真考虑。这个程序流可能是概要性的，即方框图形式；但是按照程序流编程序容易得多。典型的程序流程图如图 B-1 所示。

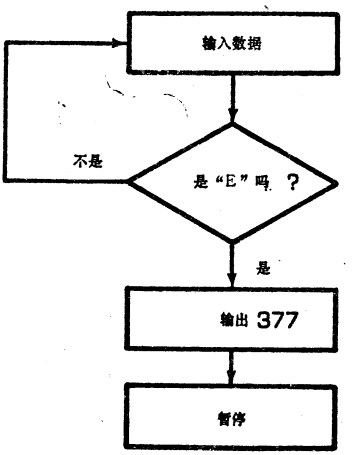


图 B-1 典型流程图

要解决的问题想好了，并且已在流程图里规定了解决办

法之后，则必须进行判定。程序够简短，容易人工翻译吗？在许多情况，特另是程序简单，人工汇编最为适合。在其它情况下，称之为软件开发的工具**编辑程序**和**汇编程序**则更快更有效。为了理解编辑程序和汇编程序是怎样工作的，我们来研究这一栏连同代码的形成过程。

第一步是概要地描述问题，并且用简短的一栏助记符能够具体阐明。然后把手写本打印，做些修改，再打印，也许还要进行最后一次修改和打印。列出实例，并系统地进行具体说明。这一过程叫做编辑过程。编写程序时，最好避免参考诸如"下例"或"下页的表"的做法。当汇编这一栏时，参考具体表或图的办法，进行下去，容易得多。

开发计算机软件的方法大同小异；或者在微型计算机上使用编辑程序，或者在分时系统上使用编辑程序来分别编辑各个程序步。编辑程序可以改正程序步，改变程序步，即插入和删去程序步，正如秘书处理手稿一样。编辑程序一般不知道你在编写计算机程序，因为你可以使用编辑程序来写信，准备邮件单等。当编辑程序来准备助记符形式的程序时，在程序中常常把**符号地址**分配给程序任务。这样，子程序的地址实际值可以参考字母 LOOP，把它作为延时循环的始地址。由于可以给程序步使用符号地址，不管地址的实际数值如何，都可以把程序改变。

汇编程序必须是这种程序：它从编辑程序接收信息，产生与计算机能处理的完全兼容的输出代码形式。汇编程序包含助记符表及其等效的值。例如，8080 的汇编程序把一条 MVIA 指令翻译成076(八进制)。汇编程序也可以把真正的 16 位地址分配给符号地址，例如 LOOP。使用符号地址时，务必使每个符号地址都有一条程序指令。如果使用符号地址，则必须分配

一个地址。同一个“名字”只能分配给一个地址。多数汇编程序能认识别重新定义的符号或无定义的符号，并且产生错误信息，让你知道需要改正什么。

汇编程序的最后输出在纸带上穿孔，或者存入盒式录音机，或者存入磁盘，以备计算机系统执行之。多数汇编程序也可以产生程序表：示出每条指令的地址，每个连续存储器单元的数据，符号地址的名称和助记符及其说明。典型汇编程序输出如例 B-1所示。

例 B-1 表示典型汇编程序输出的程序例

```
                        * 003000
003 000 061  START,  LXISP   /START 的符号地址。
003 001  377         377
003 002  000         000
003 003  333 LOOP,   IN      / 从端口 5 输入数据
003 004  005         005
003 005  376         CPI     /把该数据与026比较。
003 006  026         026
003 007  312         JZ      /如果一致，则转到“DETECT”。
003 010  015         DETECT
003 011  003         0
003 012  303         JMP     /如果不一致，则转到
003 013  003         LOOP    /LOOP，再检查。
003 014  003         0
003 015  171         MOVAC
003 016 323          OUT
003 017 007          007
003 020 166          HLT
```

程序经汇编以后，为了能正确地运行，还需要进行调试。

如果没有别的软件“工具”，那么程序检查和调试工作可能是艰苦的。事实证明，计算机控制面板是有用的，但是用它来读二进制代码可能太费时间；许多计算机没有外部控制设备和读出设备。调试程序作为其替代措施对大多数计算机都是适合的，可以用它改变指令、程序数据块或指令和一次一条指令，单步通过程序。

许多调试程序具有在被测试的程序中建立断点的能力，这是它们的特征之一。计算机达到断点时，执行该地址上的指令；如电传打字机一类输出设备给CPU的重要内部寄存器的内容列表。断点是非常有用的，因为它们不仅表示计算机达到了程序的某点，而且表示它到达某一点时计算机还在做什么。如果在正常程序流上设置断点，如果没有达到断点，则程序有错误。如果是这样，断点会越来越靠近程序的始地址，直到找到错误为止。发现错误时，利用调试程序改变指令、数据等，从而能把错误改正。

程序正确地运行时，调试程序应该有办法把程序存储在纸带上、录音机里或其它媒体中。调试程序也能把程序读回到存储器里。总之，找到错误时，还需要对程序再进行编辑和汇编，才能产生完善的无错误的文件程序。

由于大多数程序都含有错误，所以最好是把调试程序作为计算机的常驻部分。把调试程序存储在只读存储器（ROM或PROM），要小心谨慎，因为正在被测试的“越界”程序可能更改调试程序，因此必须把它再装入。有许多调试程序和控制程序可供使用；英特尔公司的Insite软件库程序至少有四个。编辑程序和汇编程序也可以常驻在PROM里。读/写存储器和PROM芯片的价格很低，因此建议多数用户应该把编辑程序、汇编程序、调试程序等标准系统程序常驻在自己的系统

里；或者存储在纸带上，录音机里和以磁盘为基础的软件块里，在每次应用之前，必须读入到存储器里。

交叉汇编程序也能产生汇编程序，但只供别的计算机使用。例如，PDP-11能够给8080微型计算机的程序进行交叉汇编。交叉汇编程序可能是强有力的程序，因为有些包括仿真程序，也可用来测试程序。

用来测试程序的程序是DBUG，它是C·A太特斯编写的；我们的程序例示出的汇编程序输出是Tychon编辑程序/汇编程序(TEA)产生的。这两种程序常驻在8080系统的PROM芯片内*。

后　记

《8080/8085的软件设计》第一册详细地介绍了8080微处理器的基本指令系统，给出了许多程序实例；第二册是为8080和8085这两种机器的用户用汇编语言编制系统程序而设计的，实用性较强。它不仅向读者提供了在系统上可以运行的现成程序，而且对这些程序进行了详细的说明，从而告诉读者编程序的方法和"捷径"。这是本书不同于其它程序设计的书的突出地方。

第一、三、四、五、六、七、八、九章和A、B两个附录由张梅岗、刘翠霞同志翻译，第二、十章分别由廖星桥、熊永寿同志翻译；全书由符明、薛家政同志审校。

由于译者水平有限，错误之处恳求读者指正。

译者

一九八四年元月

于长沙铁道学院