微处理机的程序设计和软件研制

〔英〕F·G·邓肯 著 白英彩 译

上海科学技术文献出版社

科技新书目：86-234
统一书号：15192·217
定　价：2.00 元

微处理机的程序设计和软件研制

〔英〕F·G·邓肯 著
白英彩 译

上海科学技术文献出版社

内 容 提 要

《微处理机的程序设计和软件研制》系根据 F. G. Duncan 所著 "Microprocessor Programming & Software Development" (Prentice-Hall international, Inc. 1979 年版)一书翻译的。

该书属于论述微处理机的程序设计和软件研制方面的专著。全书共包括微处理机及其系统、指令、指令系统、算术运算的程序设计、非数值操作的程序设计、用高级语言的程序设计、软件的组织与语言和结构、小型 8080 系统用的第一个软件及其文本等八章。书中运用了类似于计算机硬件描述语言使多种型号的微处理机指令有了统一的书写格式，对于微型计算机的教学和应用颇有意义。

该书适于从事微型机的设计、生产和应用的广大工程技术人员以及大专院校和中等专科学校有关专业师生阅读。

译　序

随着计算技术的飞速发展，计算机的应用也日趋普及，特别是微型计算机的应用尤其广泛。要想很好地应用微型计算机，就必须了解其软件。所谓微型计算机的软件是指它的各种程序。它能使微型计算机充分发挥效能，为用户使用机器提供方便。因此，微型机的软件是其重要组成部分。不管是系统软件还是应用软件，对于设计、生产和应用微型计算机来说，都是非常重要的。为了更好地适应计算技术的发展形势和促进计算机应用事业，特翻译了《微处理机的程序设计和软件研制》一书。

本书是在作者及其同事们多年来从事微型机程序设计的教学和运用各种型号微处理机的基础上写成的。该书最大特点是通过建立一种符号系统（或称硬件描述语言）来表示指令，能使不同厂家的各种微处理机的指令助记符归纳为一种统一的表示法。这对于微型机的应用和教学是很有意义的。借助本书所介绍的方法，可以把为某种型号（如 6800）的微处理机编写的程序很快地变换为其它机种（如 8080/8085、Z80 等）的程序。也就是说，使得我们运用某种型号微处理机而获得的经验具有普遍意义。

该书的另一特点是比较系统地阐述了一个软件包中的各个程序段，指出了它的编程要领和技巧。书中提供的各有关机种的指令索引表格，对广大工程技术人员是颇有裨益的。

运用本书提供的指令书写格式和所介绍的编程技巧，能进一步地扩大现有微型机的功能和用途，便于深入地掌握微型机

软件的基本概念和理论，提高编程能力，对推广普及微型机应用是有益处的。

本书内容深入浅出、通俗易懂、理论联系实际，是一本较实用的教学和专业参考书，值得一读。

本书稿付印前承蒙邱百光副教授悉心审阅；全部译稿经刘寿和、顾良士和吕宗祺等工程师仔细校对。他们提出了不少宝贵意见。在此一并致谢。由于译者水平有限，错误和不妥之处在所难免，恳请批评指正。

译　者

1982 年 2 月

原　序

《微处理机的程序设计和软件研制》一书旨在向初用微处理机的人员介绍一些程序设计知识和基本的软件设计技术。说得准确一点，本书将作为在微处理机方面希望深入理解编写和使用程序的一些新手的教科书和参考书。

笔者所以写本书是试图把几种不同型号的微处理机归纳起来。随后又与承担同样任务的师生进行了广泛的讨论，并根据有许多背景和兴趣各不相同的听讲者参加的许多课程和研究班的教学经验进行了充实。

本书主要供大专院校计算机科学系学生作为教材。在阅读本书时，假设读者尚不具备程序设计方面的知识，所需的逻辑设计方面的知识并未超出学生在一、二年级中所学得的知识。在布里斯托尔(Bristol)大学，本课程是对三年级学生讲授的。将来，当大学教学大纲广泛采用微处理机作为教学手段时，此书将作为一年级的教材。

有些阅历深的学生已熟悉了本书内容。譬如，有些学生是通过夜校和假期课程学习班，有些学生是通过自学而学完的。他们提出的非常有价值的批评和建议在本书中已予以考虑。因此，希望本书对于未来的微处理机用户有所裨益，这些用户已能胜任本领域的工作，但需要学习程序设计为其将来的工作以及确切地理解其他人的工作而打下扎实的基础。这类读者将会看到本书的主要内容是完善的，如在阅读第一章遇到困难时，则可以查阅有关逻辑设计方面的基础课本。这样一来就能较好地理

解微处理机系统的结构，并根据其需要来运用微处理机。

本书的叙述方法是实践性较强，虽然许多材料都是比较通用的，但所有详细的讨论都是以 Motorola6800、Intel 8080 和 8085、Zilog Z80 等四种用得很广的处理机为基础进行的①。作为本书的一个组成部分，给出了一个完整的 8080 软件包，并作了全面的说明。

第一章从程序员的角度出发，概括地介绍了微处理机及其系统的硬件。

第二章详尽地描述和讨论了机器的指令系统及其作用。本章还引入了一种指令系统的通用表示法，在一定程度上可帮助读者从已掌握的机种顺利地过渡到另一种机种。这种表示法与构成由制造厂设计的"汇编语言"的所谓"助记符"系统迥然不同。它有很多优点，特别是对于不希望局限于一种机器的用户来说更有好处。对某些类型的指令(例如加法和减法)的用法作了详尽的讨论。如读者觉得这部分内容已超出了自己的需求，则在阅读第一遍时就可略过这一段，留待以后用到时作为参考。

第三章给出四种处理机(6800、8080、8085 和 Z80)的通用形式的指令系统表，以及将各指令系统中的指令(第二章已作介绍)按字母的顺序排列的索引表。这些表面上看来令人望而生畏的大块参考资料对于学习是很有帮助的。读者在后一阶段将会发现在编写程序时有这些材料和表格是很方便的。

第四章通过一系列由浅入深的例子介绍了程序设计技术。开头的一些例子为学习简单的输入、输出以及字符和数字处理方面的内容打下基础，其后的例子相当于软件包中的子程序，以

① 当然，本书内容对其它处理机的适用程度将与这种处理机与上述四种处理机之间的差别有关。虽然其中有些材料对非 8 位字长的处理机也是适用的，但许多材料未必能有效地应用于位片式器件。

后各章将予以详细介绍。

第五章由一系列多少有点脱节的小节组成。唯一的共同之处是各节所讨论的课题都不是处理数量的问题。有些课题(例如排序问题)相当详细地讨论了不止一个实例；但是有些课题(例如文件和记录问题)由于篇幅所限只作了简要论述。

第六章介绍了有助于程序设计的一系列基本软件手段，详细地叙述了使用这些软件手段(包括在软件包里)的用户须知。本章末讨论了如何通过进一步的使用软件提高程序员的表达能力的问题。

第七章对微处理机软件的发展历史作了必要的说明，提及了可供使用的软件包，并讨论了将来各种可能的发展趋势。如果读者在做本书的程序设计练习以外，还使用某种高级语言的编译程序则，应该可以参加这种讨论，也许能达到这样的程度，即根据自己的思路来设计和补充。提供新的语言或其它更好的表达方式。

第八章包含有说明程序设计技术时所用的软件包的完整文本(附有详细的注释)，而这些软件的使用须知已在第六章进行了介绍。这个软件包需占用 6K 字节的可编程只读存贮器(PROM)，由一个监督程序、一个 PROM 编程程序、一个汇编程序和一个反汇编程序以及一组算术子程序(包括 32 位的浮点运算程序)组成。汇编程序至少需要用 1K(最好是 2K)字节的随机存取存贮器(RAM)作为工作空间。然而，正如第八章所说明的那样，部分软件包所需的 PROM 和 RAM 可以少得多。这个软件已使用了一段时间，因此对于读者来说将是有用的。当然，它也不是完美无缺的，未必能长久地满足一些要求高的用户的需要。读者既然能发现这个软件包对自己的用处、限制因素及其不足之处，以及仔细地研究了这个软件包并了解其结构和缺陷，

也就能自行设计和实现好的软件及其它程序。

程序设计是一种实践技能，只有在具体的机器上通过实践才能获得。显然，如果读者拥有能运行本书中的软件包的 8080、8085 或 Z80 系统，则从本书所学到的东西，将比那些只使用其它某种机器的读者所学到的来得多，而后者又比无法接触任何机器的读者好得多。这三种读者聚集在一起，学习为他们的机器编制程序，则都将会有所收益。众所周知，懂两种语言的程序员远比只能用一种语言的程序员能更好地运用新的机器和语言。能为自己购买设备的初学者使用不同的两个小系统所学到的东西远比从一个较大的系统中学到的东西多得多。

本书对有关交叉汇编程序、交叉编译程序、具有很强的诊断能力的调试工具或开发系统方面的内容叙述得很少。这些东西很繁杂，会妨碍初学者的学习，即使是有经验的程序员使用起来也会感到容易混淆。这些东西当然非常方便，很有吸引力。但是正确的方法是要在其上运行某个程序的系统是研制该程序的最合适的系统；虽然错误是不可避免的，但是，混乱和不必要的复杂化则不是不可避免的；在程序设计和任何其它设计工作中，简单、设计良好并容易理解的工具比精巧、复杂而不可预言的机构更为可取。

书中肯定会有错误，会有不应该有的混乱和太复杂化的地方，这都是由于笔者的学识浅薄，书中表达的全部见解也都只是作者的一孔之见。如果书中有任何值得一提的地方，那功劳也主要是属于我的许多同事、教师、学生和朋友，他们有意和无意地给与了许多帮助。由于为数众多，只能在此一并致谢了。

附言(1979 年 6 月)

笔者撰写本书时，第八章的软件已经过改编供一8085 系统使用，其中

的 PROM 在0000-1FFF(4×2716), RAM 在2000以上。§ 8.7提出的一些改进建议已经实现了。使用起来稍微方便一些的汇编程序是在一只2K的PROM中(1800-1FFF)；具有新的行式打印机命令的监督程序、PROM 编程程序和反汇编程序位于另一个区域(0000-07FF)；算术和输入-输出子程序位于第三个区域(0800-0FFF)；而第四个区域(1000-17FF)供存贮盒式磁带机程序和正在研制的高级软件用。这个系统的用户应该注意下述变换：

8080	8085	8080	8085
0360, 03C0	0371, 03C9	0D6F-0D7F	086F-087F
07E6-07FF	0886-089F	0DCA-0FFF	08CA-0AFF
0BE0-0BFF	08A0-08BF	2400-27FF	0C00-0FFF
0CF1-0CF7	08C1-08C7	2B80-2BFF	0B80-0BFF

F. G. D.

目　录

第一章　微处理机及其系统

§1.1　基本概念

就内部结构而论，微处理机是一种超大规模集成电路，它相当于在4平方毫米左右的硅片上集成了数千个分立元件，封装成40条(典型值)引脚的双列直插式组件。就逻辑原理而论，微处理机是一种由时钟信号控制的时序电路，也即能用时钟脉冲同步地改变其内部状态，从而改变其输出信号的一种电路。时钟脉冲通常(但不是全部)是由外部产生的。微处理机各个内部状态的变化都是由其当时的内部状态以及一组输入信号共同决定的。上述说明同样也适用于JK触发器之类的器件，其差别只是复杂程度不同而已，因为微处理机可能出现大量的输入信号组态、内部状态以及输出信号组态。

然而，微处理机的工作原理却相当简单，实质上只是内部状态的重复循环。这种循环称为指令周期，指令周期总是由下述阶段组成：

(i) 将一组输入信号(一条指令)读入(锁存入)处理机的一个内部寄存器(指令寄存器)。

(ii) 处理机要经过一系列由该指令的各位所决定的状态。可能还要读入其它输入信号(数据)或产生输出信号(结果)。这就是指令的执行阶段。

(iii) 最后，产生一组输出信号(下一条指令的地址)，供外部电路(一般是存贮器)决定要送入处理机的下一条指令。

使处理机完成某个特定任务所需的一串指令称为程序。要

执行的程序指令保存在存贮器里(从概念上来说,存贮器是一个很大的双稳态元件阵列)。当处理机需要一条指令时,处理机就产生这条指令的相应地址(决定指令在存贮器中的位置的一组二进制数字),并送给存贮器。然后,将指令取出来送入处理机,而存贮器的内容保持不变。因此,可把一个地址看作是一个二进制数,而下一条指令一般是指其地址跟在现行指令地址后面的那条指令(从数量概念上来说,下一条指令的地址比现行指令的地址大)。但是,程序所执行的下一条指令的地址不一定顺序放在现行指令地址的后面,即可以要求执行其它地址中的指令。执行转移指令就会发生上述情况。也就是说,采用转移指令时程序中的某些指令串可能反复执行多次,而另一些指令串根本没有执行。这取决于程序所处理的数据(存贮的指令数并不代表执行的指令数或程序运行的时间)。

微处理机与存贮器一起构成了基本的微处理机系统。一般还有其它的系统部件,称为外围设备。通常,至少有一台输入设备,用于把数或其它量(例如指令或仪表读数)输入存贮器或处理机;至少还有一台输出设备,用于把数(一组数字)从存贮器或处理机送至外界(例如送去打印或控制机器工作)。典型的微处理机系统如图 1.1 所示。

各个部件由几组线连接起来:

(i) 数据总线,在任何给定的时刻,该总线可能传送下列内容:

(a) 将指令从存贮器传送至处理机;

(b) 将一个数从存贮器或输入设备传送至处理机;

(c) 将一个数(结果)从处理机传送至存贮器或输出设备;

(d) 在存贮器的两个部分之间,存贮器与外围设备之间或两台外围设备之间传送一个数。

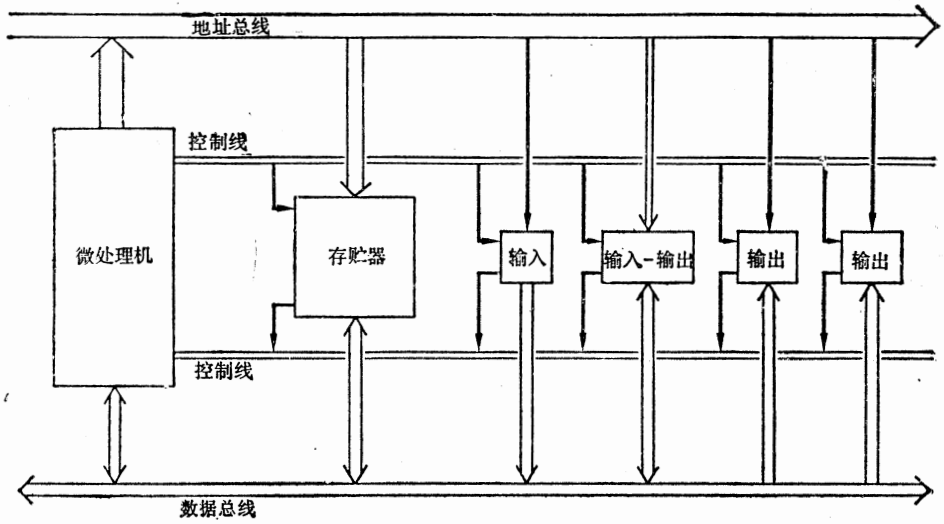


图 1.1　一个典型的微处理机系统

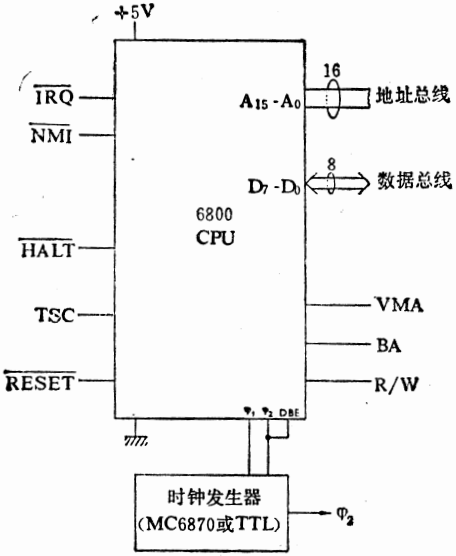


图 1.2　6800 的总线和控制连线

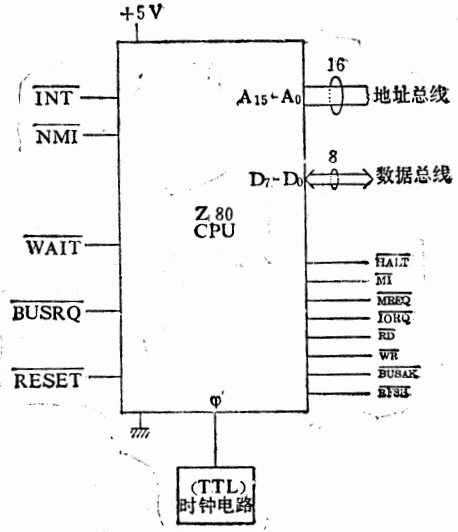


图 1.3　Z80 的总线和控制连线

(ii) 地址总线，在任何给定的时刻该总线可能传送下列内容：

(a) 处理机需执行的下一条指令的地址；

(b) 处理机进行计算所需的操作数的存贮地址；

(c) 准备写入处理机所计算出来的数的地址；

(d) 准备将一个数发送给处理机的某台输入设备的地址(或设备号)；

(e) 准备从处理机接收一个数的某台输出设备的地址(或设备号)；

(f) 与上述(i)(d)条的传送有关的地址。

(iii) 控制线，传送保证计算机同步和协调的定时和控制信号，从而保证正确地执行(i)条所列的各项信息传送操作。

有些微处理机的结构与这种基本结构稍有不同。地址总线和数据总线可以公用一组引脚(8085)；或者可将数据引脚用作其它用途，例如作为传送外部设备与处理机之间的控制信号用(8080)。

图1.2至1.5分别简单地说明了6800、Z80、8080和8085的总线和控制连线。8080和8085的连接图说明，这两种微处理机需外接其它元件对总线信号进行锁存和缓冲，因此使一般的存贮器器件可接至总线上去(也有某些特殊器件不用这些附加元件就能与8080或8085一起工作)。若需使用适量的存贮器，则6800和Z80也需要使用总线缓冲器(参阅§1.3.3)。

表1.1列出了各种控制信号，且粗略地按功能进行了分类并作了说明。列出这些信息只是想帮助程序员理解这些微处理机是如何与存贮器和外围设备进行通信的。详细的技术说明可

参阅有关的手册①，有关比较性的说明可参阅 Hilburn 和 Julich 等人编著的课本②。

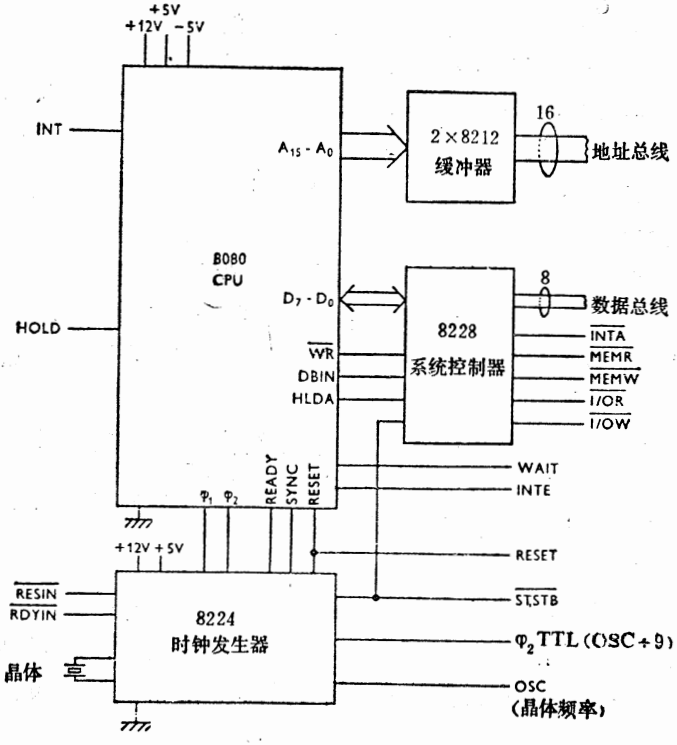


图 1.4　8080 的总线和控制连线

① MCS-80　用户手册(有对 MCS-85 的介绍)。英特尔公司 1977.10。
8085　微型计算机系统用户手册——序言。英特尔公司 1976.11。
M6800　微型计算机系统设计数据。莫托洛拉半导体产品公司 1977.3。
Z80-CPU　Z80A-CPU 技术手册。Zilog 公司，1977。

② John L. Hilburn 和 Paul M. Julich: «Microcomputers/Microprocessors: Hardware, Software, and Applications»1976 年出版。

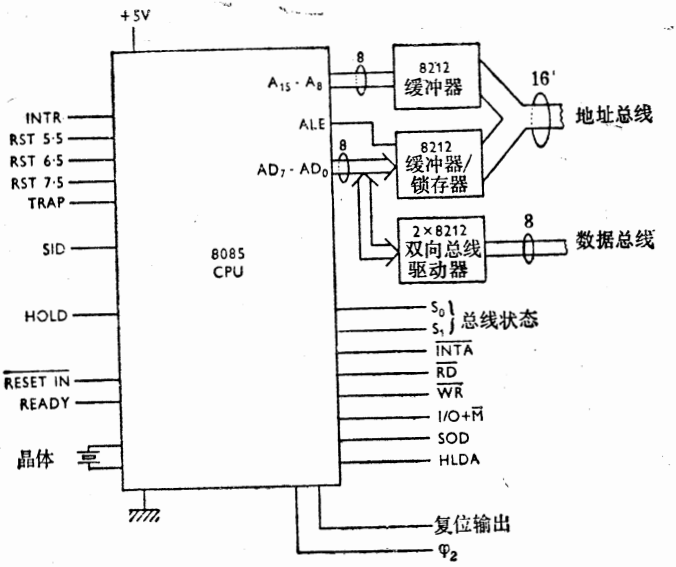


图 1.5　8085 的总线和控制连线

表 1.1　控制信号和总线信号

		6800	8080†	8085	Z80	
	复　位	$\overline{\text{RESET}}$	$\overline{\text{RESIN}}$	$\overline{\text{RESETIN}}$	$\overline{\text{RESET}}$	异步复位信号（见第三章）
	准备好		$\overline{\text{RDYIN}}$	READY	$\overline{\text{WAIT}}$	机器进入等待状态
控制	总线控制	TSC	HOLD	HOLD	$\overline{\text{BUSRQ}}$	使各条总线脱离 CPU
输入		$\overline{\text{HALT}}$				机器停机，各条总线脱离 CPU
		DBE				数据总线脱离 CPU

（续表）

		6800	8080†	8085	Z80	
控制输入	中断请求	$\overline{IRQ}$	INT	INTR	$\overline{INT}$	普通的请求信号
				RST5.5 RST6.5 RST7.5		中间请求（见§2.9和第三章）
	特殊信号	$\overline{NMI}$		TRAP SID	$\overline{NMI}$	高优先级请求 “串行输入数据”（见第三章）
总线	地址	A_{15}-A_0	A_{15}-A_0	A_{15}-A_8	A_{15}-A_0	地址端
	地址/数据			AD_7-AD_0		多路转换的地址/数据端
				ALE		地址选通
	数据	D_7-D_0	D_7-D_0		D_7-D_0	数据端
控制输出	数据传送控制	R/W VMA	$\overline{MEMR}$ $\overline{MEMW}$ $\overline{I/OR}$ $\overline{I/OW}$	$\overline{RD}$ $\overline{WR}$ I/O+$\overline{M}$	$\overline{RD}$ $\overline{WR}$ $\overline{MREQ}$ $\overline{IORQ}$	这些输出信号经组合后允许信息沿着数据总线在规定的单元和设备之间进行传送，并进行定时
	中断计时		$\overline{INTA}$	$\overline{INTA}$	$\overline{M1}$	从数据总线读取中断指令用的定时信号（见§2.9）
	CPU状态		WAIT INTE	SOD	$\overline{HALT}$	关于CPU状态的信息（INTE、SOD信号见第三章）

（续表）

		6800	8080†	8085	Z80	
控制输出	总线状态	BA		$\overline{HLDA}$ S_1S_0	$\overline{BUSAK}$	关于数据总线状态的信息（总线是否可用，或当前的传送操作的方向和性质）
	特殊信号				$\overline{RFSH}$	动态RAM的再生定时脉冲(见§2.10)
	同步的输出信号		RESET	RESETOUT		供接口使用（见§1.4.1）

8080†＝8080＋8224＋8228(见图1.4)

§1.2 微处理机的内部结构

典型的微处理机的内部结构如图1.6所示。并非所有的微处理机都有图中所示的全部部件——特别是可能没有“其它的寻址寄存器”和“其它的寄存器”。在某些情况下，某个寄存器既可作算术运算寄存器，又可作寻址寄存器。有些处理机还有图中未画出的一些内部单元，例如寄存器堆栈或小容量的ROM、PROM或RAM。

图中的每条连线表示若干条并行线，线的数目可视特定的处理机型号而定。

1.2.1 电源

尽管有些处理机只需要一组电源(例如6800、8085、Z80和2650只需一组＋5V电源)，但由于工艺不同，其它一些处理机

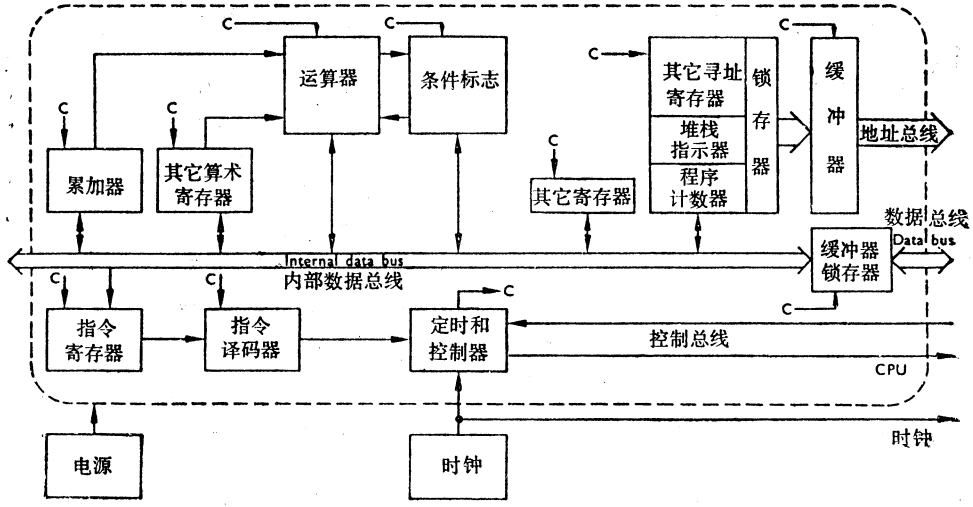


图 1.6　典型的微处理机的内部结构

则需要三组电源(例如 8080 需要 +12V、+5V、−5V 三组，CP1600 需要 +12V、+5V、−3V 三组)。微处理机的功耗一般在1～1.5W 之间，有时可能还要小得多(例如 2650 为 525 mW)。

1.2.2 时钟

目前大多数微处理机都需要用外部时钟(规则的脉冲源)进行驱动。但是，也有一些具有内部时钟电路，只需外接一个晶体来决定时钟的频率。

在使用外部时钟的机器中，Z80 需要一个单相的 TTL(晶体管-晶体管逻辑)电平的方波；6800 需要两个互不重迭的 TTL 电平的脉冲串；而 8080 需要两个互不重迭、幅度为 12V 的脉冲串。但是，在 8080 系统中可用专用的时钟发生器集成电路(8224)来产生时钟信号，8224 只需外接一块晶体。

在大多数情况下，时钟脉冲的宽度和频率必须保持稳定，但是允许的频率范围相当宽(例如 CP1600 为 0.5MHz～5MHz)。有些处理机(例如 2650)允许时钟频率一直下降到手动的频率——这就大大简化了"单步"操作问题。

1.2.3 定时和控制器

从逻辑上来说，定时和控制器是一个由时钟驱动的同步时序电路。其输入信号包括：

(i) 外部控制信号　　例如预置用的"复位"信号、外围设备"准备好"信号以及这些设备的"保持"和"中断"请求信号。

(ii) 指令译码器(见下文)的输出信号　　由这些信号决定指令周期。

其输出信号包括：

(i) 内部控制信号　　触发其它的内部单元，并在微处理机内部建立数据通路，对数据传送操作进行定时。

(ii) 外部控制信号　例如"保持"和"中断"请求用的"允许"和"响应"信号；总线定时脉冲；以及给出有关处理机和总线的状态以供外部部件使用的信号。外部控制信号通过控制总线进行传送，出入于系统其它部件；内部控制信号(在图 1.6 中用 C 来表示)则分配至处理机所有其它的内部部件去。

定时和控制器正规的功能是对每条经过译码的指令作出响应，从而发出信号以保证指令的正常执行。这些信号驱动寄存器、运算器和有关的外部设备。例如，可作下列操作：

(i) 若经译码的指令是一条寄存器间的传送指令，则应通过适当的允许和开门信号，打开一条从源寄存器经过内部数据总线到目的寄存器去的通路，传送操作则由定时信号(选通脉冲)完成。

(ii) 对于算术运算指令而言，控制信号允许数据从存有操作数的两个寄存器(或寄存器和存贮单元)进入运算器(见下文)；这些信号能保证执行所需的操作，允许结果进入一个累加器或其它的结果寄存器，甚至可送回存贮器。

(iii) 若涉及到对存贮器的读写操作，则存取数据的地址将保存在一个寻址寄存器(例如变址寄存器或堆栈指示器)中，并会被送至地址总线。此外，有关的数据寄存器将被连接至数据总线。如前所述，数据沿着数据总线在寄存器与存贮器之间进行的传送将由定时脉冲实现。

定时和控制器除了控制每条指令的执行过程以外，还必须不断前进或用新值取代程序计数器中的内容，以获得要执行的下一条指令的地址。然后才能从存贮器中取出这条指令，并送至指令寄存器去。

1.2.4 指令寄存器和译码器

指令寄存器依次接受每条指令，并在整个执行过程中保存

该指令。指令译码器原理上是一种组合电路，因此与其它译码器并无区别。其输出信号供定时和控制器用来决定执行指令寄存器中指令的过程。

1.2.5 运算器(ALU)

该部件包括执行一些(组合)运算的电路，这些运算的操作数可以是一个，也可以是两个。这类运算有加法、减法以及同时对一个操作数中的所有位或两个操作数中对应各位执行的各种逻辑运算。在大多数情况下，其运算结果要保存在累加器中；几乎所有的情况都会影响条件标志。

1.2.6 条件标志

大多数(如果不说是全部的话)算术和逻辑运算都会影响一个或几个条件标志。每个标志都可看作是一个触发器，其置位和复位的状态由最后执行的算逻运算的结果决定(在6800中，简单的数据传送操作也会影响条件标志)。例如，有一个标志将记录最后的结果是零还是非零，而另一个标志将记录它是正的还是负的(即其最高有效位是0还是1)。由条件标志决定条件转移指令的结果。在大多数处理机中，都有将条件标志各位的值送入另一个寄存器或送入存贮器的指令，以及需要时恢复其原值的指令。所以这组条件标志可以当作一个寄存器("条件码寄存器"、"条件标志寄存器")来处理。此外，还可用专用指令对个别或全部条件标志进行置位或复位，而无视运算器的运算结果如何。

1.2.7 累加器

累加器基本上就是一个寄存器，能接受和保持其原先值与其它某个数的算术和。换言之，累加器能形成一个累加和。广而言之，也可用累加器这个词来表示对加法以外的其它操作起同样作用的寄存器。所有的微处理机都至少有一个累加器。沿

着数据总线与处理机交换数据时通常要用一个累加器作为寄存器。

1.2.8 程序计数器

程序计数器是一个寄存器，存有接下来要执行的指令地址。通常，这个新地址是由现行指令的长度（一、二、三或四个字节）加到现行指令的地址上而产生的。但是，在无条件转移指令中，程序计数器的内容由指定的地址所取代（这一代换就完成了转移操作）。在条件转移指令中，若规定的条件标志（或条件标志值的函数）具有指定的值，则也产生这样的代换，否则将正常地逐步增加。在相对转移（或分支）指令中，转移指令指定的数将被加至程序计数器上。

程序计数器的内容也会受子程序调用、中断和返回指令的影响。有些处理机还有明确地对程序计数器进行操作的指令。

1.2.9 堆栈指示器

堆栈操作是许多微处理机都具有的一种特点。堆栈本身一般建立在存贮器中，通过将其基地址送入堆栈指示寄存器就可建立堆栈。每当新的一项内容被压入堆栈时，堆栈指示器的值就前进；每当从堆栈中取走一项内容时，堆栈指示器的值就缩回。“前进”一般意味着“减1”，“缩回”意味着“加1”。因此基地址是一个高地址，而堆栈在地址空间中“向下”发展。一般的微处理机都有明确地对堆栈指示器进行操作的指令，也可能用一些办法在执行中断或子程序时保存堆栈指示器的值。

1.2.10 其它寻址寄存器

寻址寄存器是指其内容可用作存贮器读、写操作中的地址的寄存器。变址寄存器就是具有相应的“加1”和“减1”操作的寻址寄存器，可用于对连续的存贮单元进行存取。

§1.3 存 贮 器

1.3.1 主存贮器和辅助存贮器

大型系统的存贮器由两部分组成：(i)主存贮器和(ii)辅助存贮器(或后援存贮器)。小型系统则只有主存贮器。

辅助存贮器包括磁带和盒式磁带、磁盘以及纸带或穿孔卡片。对辅助存贮器的存取是通过外围设备进行的。

微处理机的主存贮器一般是由半导体器件构成的，每块集成电路能存贮数千位信息。主存贮器中每个字节都有一个唯一的地址。若采用 16 位的地址总线，则处理机可寻址至 65536 个字节。

为了使微处理机能够根据时钟频率所允许的速度迅速工作，存贮器件的存取时间应该与时钟频率相匹配。若存取时间太长，则可能会在存贮器能提供有效的数据之前就出现将数据取入处理机的定时信号。有时可通过干涉时钟、或使处理机周期进入"等待状态"等方法来解决这个问题，但是这些方法都远不如选择适用的存贮器件来得好。而且肯定有足够多的器件品种可供选用。

虽然微处理机都设计得在采用半导体存贮器时，工作最有效，但是也能使用磁芯存贮器作为主存贮器。不过这样容易出现上面提到的定时问题，必须设法加以解决。

1.3.2 半导体存贮器

半导体存贮器是以标准双列直插式封装集成电路的形式提供的。每块集成电路的容量用含有多少"位"来表示，位数几乎总是 2 的乘幂。这些位可以单独地进行存取，或以 4 位或 8 位为一组进行存取。因此，某种型号的存贮器可能包含 1024×1 位，而另一种是 256×4 位，还有一种是 128×8 位。这几种存贮

器都有1024位，但是：

(i) 1024×1的器件有10条地址引脚，需要用10位的地址来规定其中某一位的位置。

(ii) 256×4的器件有8条地址引脚，需要用8位的地址来规定4位一组的数据的位置。

(iii) 128×8的器件有7条地址引脚，需要用7位的地址来规定每个字节的位置。

半导体存贮器有ROM、PROM和RAM等三种主要类型，下面将分别进行介绍。

1.3.2.1 ROM(只读存贮器)

这个术语是指含有固定信息的半导体存贮器。这些信息是在制造ROM的最后阶段制作在器件内的。这些信息不能改变；这类器件安装在系统中时，处理机只能对其进行读出。因此，ROM的应用是相当专门的。例如可用ROM存放标准的代码转换表。有些含有程序的ROM也可购得，但是读者必须注意：ROM中的程序是不可修改的。

典型的1024×8的ROM有10条地址引脚和8条数据引脚。数据引脚可直接连接至数据总线。ROM中有三态缓冲器③，其输出一般呈高阻状态，但是可通过向"启用"引脚送入一个信号(处理机来的定时脉冲)来启用这些缓冲器。只有当所需的数据位于某片存贮器件之中，而不在其它器件内时，才能启用这片器件。因此，启用信号可能必须用高六位地址(在现在这种情况下)的组合来进行选通。有些ROM为便于这样使用而备有"选

③ 就我们的讨论范围而言，三态缓冲器就相当于电子式通断开关。当其被控制信号启用时，其输出信号的逻辑值(0或1)与其数据输入信号一样；当其被禁止时，其输出信号与数据输入信号无关。在后一种状态(第三状态)下，数据输入与输出之间插入了一个很高的阻抗，其实际作用相当于开路。

片”引脚，在启用某片器件之前必须先使其各选片端处于规定的逻辑电平。

1.3.2.2 PROM（可编程只读存贮器）

当PROM安装在系统中时，其作用完全与ROM相同，使用的方法也相似。ROM和PROM之间的显著区别是PROM可以“重编程序”，即PROM能从计算机系统中拆下来，擦除其中的信息，并将新的信息存入其中。“重编程序”过程需使用专用的仪器（例如用紫外辐射源进行擦除，用电压相当高的脉冲源进行重写）。微处理机系统中正常的信号不会破坏或改写PROM中的信息。

值得注意的是，“可编程序”、“重编程序”、“程序设计”等等词汇用在本书中是很遗憾的，因为“程序”是计算机必须执行的一组指令，“程序设计”就是编写程序。有些用户误认为只有PROM才能用来保存程序了。

显然，可用PROM来存贮与ROM中属于同一类的信息。但是，与ROM不同的特点是，PROM更适于作固定程序存贮器，如果可获得适用的“PROM编程器”和“PROM擦除器”的话，甚至可用于存贮正在研制中的可能要修改的程序。本书后面介绍的软件是以PROM的形式进行研制的，现在都存贮在PROM中。

1.3.2.3 RAM（随机存取存贮器）④

这种半导体存贮器的形式是最接近于磁芯存贮器的。信息可在系统（程序）控制下写入RAM，或从RAM中读出。目前，绝大多数可以获得的RAM在断电后都将丢失所保存的信息。

④ 用“随机存取存贮器”这个术语来表示读/写存贮器，是不太合适的，因为就所有各位可平等地由寻址机构进行存取这一点而言，不仅RAM是“随机存取”的，而且ROM和PROM也是“随机存取”的。

但是，有些 RAM 有一种“备用”状态。在这种状态下，RAM 可在相当小的功耗下保持信息，但不能读出或写入。不消耗功率而能永久地保持信息的半导体 RAM 正在研制中。

与 ROM 和 PROM 一样，所有的 RAM 也都有供选择所需的特定位的位置用的地址引脚。某些 RAM 有两组数据端，一组供写入用，另一组供读出用。但是，目前的发展趋势是只用一组数据端，并可直接接至数据总线上。在这种情况下，输出级肯定是三态的。

通常所需的控制信号是：

(i) 决定下一个操作的读/写信号；

(ii) 控制传送时序的启用信号(脉冲)；

(iii) 一个或几个选择信号；为使(i)、(ii)起作用，选择信号必须具有规定的电平。

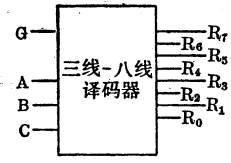
RAM 主要有静态的和动态的两种类型。静态 RAM(仍然)较动态 RAM 贵，且速度也较慢(但对于许多微处理机应用而言，速度是足够的)。只要电源不切断，则写入静态 RAM 中的信息就能永久保持。动态 RAM 的信息存贮密度较静态 RAM 高(最常见的动态 RAM 是 4K 位的，静态 RAM 是 1K 的)，这一点对于大容量存贮器系统来说是很重要的。但是，除非用某种操作进行刷新(再生)，否则动态 RAM 中的信息在写入约数毫秒后就会丢失。这是因为动态 RAM 是以电容效应作为存贮手段的，电容上的电荷会漏失，因此必须在逻辑“1”状态变得无法与“0”状态进行区别之前加以“补充”。在大多数情况下，这种存贮器都按行和列的阵列进行组织，并安排成对任何位的读或写操作都会使同一列中的所有各位进行刷新。因此，所谓“刷新”问题就是保证在每 2ms 的周期(或规定的周期)内对每一列进行存取。在微处理机系统内，一种刷新办法是迫使处理机处于等待

状态，而对整个 RAM 中的各列连续进行刷新；另一种方法是利用处理机不对 RAM 进行正常的存取操作的机器周期实现“每次一列”的刷新。在 Z80 内部已制备了以后一种方式进行刷新的电路。其它的处理机则可使用专用的集成电路。若用 TTL 电路进行设计，则刷新电路的体积可能相当大。

在实际进行程序设计时，往往没有必要考虑动态和静态 RAM 之间的区别。但是，在 Z80 中有一个“刷新寄存器”，这是一个保存动态 RAM（如果有的话）当前“刷新地址”的计数器。可把这个寄存器的值送入累加器或将累加器中的数值预置入这个寄存器以供检查。指令系统中也有进行这一操作的指令。普通的程序是不用这些指令的。

1.3.3 地址的缓冲和译码

考虑一个准备使用 4K 字节的 PROM 和 1K 字节的 RAM 的小型系统。PROM 占用的地址是 0000-0FFF（十六进制），RAM 占用的地址是 1000-13FF（十六进制）。准备用四块 1024×8 位的 PROM 集成电路构成 PROM，用八块 128×8 位的 RAM 集成电路构成 RAM。如何将这些电路连接至地址总线才能满足：(i)每个字节都可与其它字节毫不相关地进行寻址；(ii)不改变现有的连接状态就可对存贮器进行扩充呢？



G	A	B	C	R_0	R_1	R_2	R_3	R_4	R_5	R_6	R_7
0	x	x	x	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	0	0	0
1	0	0	1	0	1	0	0	0	0	0	0
1	0	1	0	0	0	1	0	0	0	0	0
1	0	1	1	0	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	1	0	0	0
1	1	0	1	0	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0	1	0
1	1	1	1	0	0	0	0	0	0	0	1

图 1.7　三线-八线译码器

解决这个问题的一种简单办法就是使用三线-八线译码器，其布局和真值表如图 1.7 所示。

图 1.8 是解决办法之一(图中未画出 RAM 用的启用信号和读/写信号)。在这种方案中，PROM 和 RAM 中的每个字节都象规定的那样有一个独特的地址。译码器 1 和 2 未连接的输出端供存贮器扩充用，最多可完全扩充至 65536 个字节。这种扩充并不影响现有的任何连线。

许多设计人员(包括那些设计市售套件的设计者)都有轻视地址译码机构设计问题的倾向。若给定了这个小型系统的规格，他们就会完全省掉 #1 译码器。这样就使得地址空间只能再扩充 3K(由 #2 译码器未用的输出端来表示)。其原因是去掉了 A_{15}、A_{14}、A_{13} 三位地址，地址总线减少到 13 位。13 位的总线只能对 2^{13}(8192)个字节进行寻址。

由于任何系统都无法避免可能在某个阶段要进行扩充，所以最初留有充分的扩充余地是个聪明的办法。这样做所化的代价(元件或空间、功耗或费用)即使与很小的系统相比也是微不足道的，所以不这样做是"得不偿失"的。

任何对逻辑设计稍有经验的人都不难从图 1.8 中发现，无论容量多大的存贮器都将给地址总线(特别是低位)增加相当重的负载。出于这一原因，除了最小的系统以外，都须对地址总线进行缓冲⑤。借助于 Intel 8212 之类的器件实现地址缓冲是很简单的。把两块这样的器件安排在处理机附近并接以适当的控制信号，就可使其成为简单的缓冲器。

---

⑤ "负载"和"缓冲"这两个术语是从工程的角度而不是从程序设计的角度而言的。不熟悉这些术语的程序员可把"负载"看作是对输入信号源的"负荷"；把"缓冲器"看作是输入一个(吃满负载的或很弱的)输入信号而产生逻辑状态相同、但能承受很大负载的输出信号的器件。

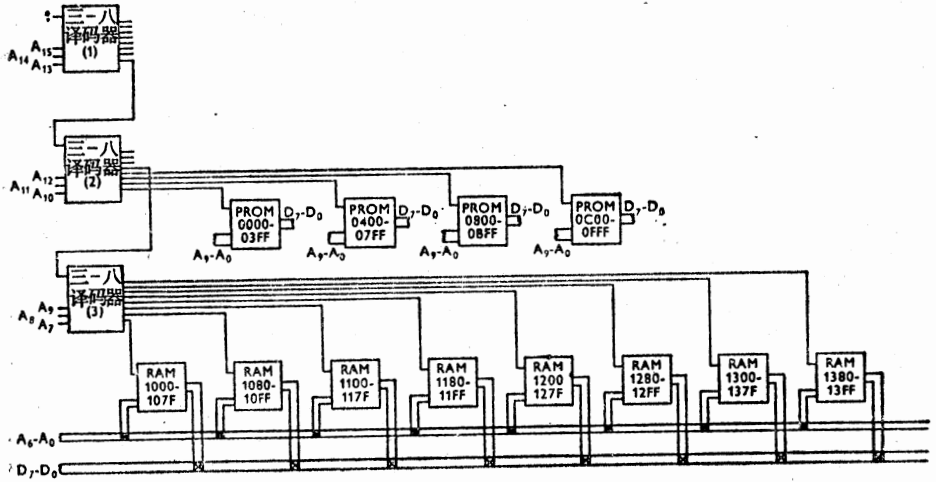


图 1.8　对地址译码机构的说明

§1.4 外围设备

即使最简单的微处理机系统也至少包括一个输入设备和一个输出设备。这些设备可能只是一组开关和一组指示灯，也可能是由电传打字机或带键盘和监视屏幕的目视显示部件(VDU)组成的复杂系统。

其它的外围设备可包括辅助存贮设备(盒式磁带机、软盘驱动器等)、信息输入/输出用的专用设备(纸带输入机、图形处理部件等)、控制其它系统或从其它系统获取信息用的机构(继电器、模/数转换器等)或便于计算机本身工作的辅助设备(如PROM编程器)。

1.4.1 接口

所有的外围设备都会向系统设计员提出这样的要求：使外围设备的速度和时序与计算机的速度和时序相适配。此外，还可能要求对不同特点的信号进行处理以使其相互适配，以及管理串-并和并-串转换操作。为解决这些问题而设计的介乎计算机系统与外围设备之间的电路称为接口。接口设计本身就是一个很大的课题，超出了本书的范围。但是，微处理机制造厂研制了能简化许多接口设计任务的集成电路。例如，若需要一个"并行接口"(一组数据位可同时通过接口来传送就称为"并行")，就可使用 Motorola 6820 (外围设备接口转接器，PIA)、Intel 8255 (可程序控制外围设备接口，PPI)和 Zilog 公司的 Z80PIO (并行输入-输出)等器件。电传打字机或目视显示部件是以串行形式逐位地接收和发送信息的。这时接口就需将各组8位数据(出现在数据总线上)变换成串行形式供外围设备使用，并把顺序的8位数据(由键盘提供)变换成并行的8位数据供系统数据总线用。Motorola 6850 (异步通信接口转接器，ACIA)、Intel

8251(可程序控制通信接口-通用同步/异步接收/发送器，USART)和Zilog公司的Z80 SIO (串行输入-输出)等器件可非常方便地完成这些变换操作。

设计供给定的一台外围设备用的接口通常都可有几种不同的方法。通过接口传送信息所需的程序，必须根据实际所用的接口而采用不同的办法。§2.8讨论了解决电传打字机或目视显示部件的接口问题的三种不同的办法，及其与程序员的关系。

第二章 指 令

§2.1 基本概念

我们已经知道，微处理机的工作就是重复进行“从存贮器中取出指令、执行指令并继续下去”这样的循环。下一条指令一般就是占用存贮器中“下一个”单元的指令，但是在某些情况(转移、子程序调用和返回、中断)下，下一条指令也可在存贮器中的其它地方。

每条指令基本上包括一个机器字，一个机器字在6800、8080、8085和Z80中都是一个字节8位。许多指令(值得一提的是执行寄存器传送和许多算逻运算用的指令)只需要一个字节；但是还有些指令后面总是紧跟着一个或两个附加的字节，这些附加的字节一般也被看作是指令的一部分，形成了一个“立即数”(数据或地址)。在所有的情况下，指令第一个字节常称为“操作码”。例外的是Z80还有长度为四个字节的指令，后面将会进行介绍。

若操作码有8位，则可能有$2^8=256$种不同的操作。这些操作由微处理机的指令译码器进行区分。实际上，6800只有197种规定的操作(执行过程)；8080有244种；8085有246种；而Z80则有全部256种，但其中有4种只是表示还有其它的操作码字节。

各种不同定义的操作就构成了处理机的指令系统。程序就是为了使处理机能完成某个任务而需执行的一串指令。应该记住的是：程序中的指令有两种顺序。一种是静态顺序，另一种是

动态顺序。静态顺序就是这些指令存在存贮器中的顺序。动态顺序就是这些指令执行时的顺序。动态顺序将由转移、调用、中断等过程决定，一般会牵涉到跳越和重复等操作。由于是否执行条件转移完全取决于计算结果，而计算结果又取决于输入数据，所以执行程序之前一般无法预知程序的动态顺序。

对信息（或数据）执行的处理过程常常被认为是由程序决定的，因此"信息（或数据）处理"这个术语就表示执行一个程序的结果，更新一些的术语"处理机"或"微处理机"则表示信息处理用的装置。而"计算"这个词含有处理数量信息的意思，但是"计算机"现在似乎还表示从简单的控制机构到大型信息处理系统等等之类的任何东西。目前，一般认为微型计算机是由微处理机、存贮器和各种外围设备构成的一个系统。

§2.2 指令的表示方式

计算机中的指令无论是在存贮器中还是在微处理机的指令寄存器中，都只是依次排列的一组二进制数字。显然，指令书写时可用一组顺序排列的0和1来表示，实际上也有觉得这样写方便的情况。但是，一般都乐意采用较简练的表示法。

考虑一个字节01010011。

在八进制表示法中，这些二进制数字从右边开始每三位一组进行分组（$8=2^3$），每组用一个能代替其值的数字来表示。上述字节就可写成123或123_8。值得注意的是，最左边（最高）的八进制数字只能是0、1、2或3，因为该数字只代替2位二进制数字。

在十六进制表示法中，这些二进制数字从右边开始每四位一组进行分组（$16=2^4$）。上述字节就可写成53或53_{16}或53H。0000，0001，…1001，1010，…1111等十六个值被写成0，1，…9，

A, …F。

再举几个例子：

二进制	八进制	十六进制
10101100	254	AC
11111111	377	FF
00000000	000	00
01010101	125	55

在本书的其余部分里除非作特殊规定，否则在书写指令时都采用十六进制表示法。

表 2.1 是 8 位二进制数字值的八进制、十六进制和十进制表示法的对照表。

显然，任何一组二进制数字（无论是指令、操作码还是数）都能写成十六进制或八进制形式。

虽然用十六进制表示输入数据是一种非常简便的方法，并且与二进制表示法相比还有不易写错的优点，但是仍有一个很大的缺点：即无法从写下来的数字看出一条指令的含义。

若字节 01010011(53) 进入 6800 的指令寄存器，就会使一个累加器（累加器 B）所有的各位均取反。而同一个字节若进入 8080、8085 或 Z80 的指令寄存器，则会使一个寄存器（寄存器 D）中的内容被另一个寄存器（寄存器 E）中的内容所取代。从“53”中是看不出这些操作的含义的，二进制表示法“01010011”的提示能力就更差了。

生产这些机器的制造厂采用由字母和其它符号组合而成的、称为“助记符”的符号来表示这些指令，现在许多用户也采用这种表示法。例如：

6800：　53 是 COMB　B 取反

8080, 8085：　53 是 MOV D, E　E 送入 D（原文如此）

表 2.1 八位二进制数字值的十六进制、八进制、十进制（带符号和不带符号）表示法对照表

十六进制	八进制	十进制 带符号	十进制 不带符号	十六进制	八进制	十进制 带符号	十进制 不带符号	十六进制	八进制	十进制 带符号	十进制 不带符号	十六进制	八进制	十进制 带符号	十进制 不带符号
00	000	(+)0	0	40	100	+64	64	80	200	−128	128	C0	300	−64	192
01	001	+1	1	41	101	+65	65	81	201	−127	129	C1	301	−63	193
02	002	+2	2	42	102	+66	66	82	202	−126	130	C2	302	−62	194
03	003	+3	3	43	103	+67	67	83	203	−125	131	C3	303	−61	195
04	004	+4	4	44	104	+68	68	84	204	−124	132	C4	304	−60	196
05	005	+5	5	45	105	+69	69	85	205	−123	133	C5	305	−59	197
06	006	+6	6	46	106	+70	70	86	206	−122	134	C6	306	−58	198
07	007	+7	7	47	107	+71	71	87	207	−121	135	C7	307	−57	199
08	010	+8	8	48	110	+72	72	88	210	−120	136	C8	310	−56	200
09	011	+9	9	49	111	+73	73	89	211	−119	137	C9	311	−55	201
0A	012	+10	10	4A	112	+74	74	8A	212	−118	138	CA	312	−54	202
0B	013	+11	11	4B	113	+75	75	8B	213	−117	139	CB	313	−53	203
0C	014	+12	12	4C	114	+76	76	8C	214	−116	140	CC	314	−52	204
0D	015	+13	13	4D	115	+77	77	8D	215	−115	141	CD	315	−51	205
0E	016	+14	14	4E	116	+78	78	8E	216	−114	142	CE	316	−50	206
0F	017	+15	15	4F	117	+79	79	8F	217	−113	143	CF	317	−49	207
10	020	+16	16	50	120	+80	80	90	220	−112	144	D0	320	−48	208
11	021	+17	17	51	121	+81	81	91	221	−111	145	D1	321	−47	209
12	022	+18	18	52	122	+82	82	92	222	−110	146	D2	322	−46	210
13	023	+19	19	53	123	+83	83	93	223	−109	147	D3	323	−45	211
14	024	+20	20	54	124	+84	84	94	224	−108	148	D4	324	−44	212
15	025	+21	21	55	125	+85	85	95	225	−107	149	D5	325	−43	213
16	026	+22	22	56	126	+86	86	96	226	−106	150	D6	326	−42	214
17	027	+23	23	57	127	+87	87	97	227	−105	151	D7	327	−41	215
18	030	+24	24	58	130	+88	88	98	230	−104	152	D8	330	−40	216
19	031	+25	25	59	131	+89	89	99	231	−103	153	D9	331	−39	217
1A	032	+26	26	5A	132	+90	90	9A	232	−102	154	DA	332	−38	218
1B	033	+27	27	5B	133	+91	91	9B	233	−101	155	DB	333	−37	219
1C	034	+28	28	5C	134	+92	92	9C	234	−100	156	DC	334	−36	220
1D	035	+29	29	5D	135	+93	93	9D	235	−99	157	DD	335	−35	221
1E	036	+30	30	5E	136	+94	94	9E	236	−98	158	DE	336	−34	222
1F	037	+31	31	5F	137	+95	95	9F	237	−97	159	DF	337	−33	223
20	040	+32	32	60	140	+96	96	A0	240	−96	160	E0	340	−32	224
21	041	+33	33	61	141	+97	97	A1	241	−95	161	E1	341	−31	225
22	042	+34	34	62	142	+98	98	A2	242	−94	162	E2	342	−30	226
23	043	+35	35	63	143	+99	99	A3	243	−93	163	E3	343	−29	227
24	044	+36	36	64	144	+100	100	A4	244	−92	164	E4	344	−28	228
25	045	+37	37	65	145	+101	101	A5	245	−91	165	E5	345	−27	229
26	046	+38	38	66	146	+102	102	A6	246	−90	166	E6	346	−26	230
27	047	+39	39	67	147	+103	103	A7	247	−89	167	E7	347	−25	231
28	050	+40	40	68	150	+104	104	A8	250	−88	168	E8	350	−24	232
29	051	+41	41	69	151	+105	105	A9	251	−87	169	E9	351	−23	233
2A	052	+42	42	6A	152	+106	106	AA	252	−86	170	EA	352	−22	234
2B	053	+43	43	6B	153	+107	107	AB	253	−85	171	EB	353	−21	235
2C	054	+44	44	6C	154	+108	108	AC	254	−84	172	EC	354	−20	236
2D	055	+45	45	6D	155	+109	109	AD	255	−83	173	ED	355	−19	237
2E	056	+46	46	6E	156	+110	110	AE	256	−82	174	EE	356	−18	238
2F	057	+47	47	6F	157	+111	111	AF	257	−81	175	EF	357	−17	239
30	060	+48	48	70	160	+112	112	B0	260	−80	176	F0	360	−16	240
31	061	+49	49	71	161	+113	113	B1	261	−79	177	F1	361	−15	241
32	062	+50	50	72	162	+114	114	B2	262	−78	178	F2	362	−14	242
33	063	+51	51	73	163	+115	115	B3	263	−77	179	F3	363	−13	243
34	064	+52	52	74	164	+116	116	B4	264	−76	180	F4	364	−12	244
35	065	+53	53	75	165	+117	117	B5	265	−75	181	F5	365	−11	245
36	066	+54	54	76	166	+118	118	B6	266	−74	182	F6	366	−10	246
37	067	+55	55	77	167	+119	119	B7	267	−73	183	F7	367	−9	247
38	070	+56	56	78	170	+120	120	B8	270	−72	184	F8	370	−8	248
39	071	+57	57	79	171	+121	121	B9	271	−71	185	F9	371	−7	249
3A	072	+58	58	7A	172	+122	122	BA	272	−70	186	FA	372	−6	250
3B	073	+59	59	7B	173	+123	123	BB	273	−69	187	FB	373	−5	251
3C	074	+60	60	7C	174	+124	124	BC	274	−68	188	FC	374	−4	252
3D	075	+61	61	7D	175	+125	125	BD	275	−67	189	FD	375	−3	253
3E	076	+62	62	7E	176	+126	126	BE	276	−66	190	FE	376	−2	254
3F	077	+63	63	7F	177	+127	127	BF	277	−65	191	FF	377	−1	255

Z80: 53 是 LD D、E E 打入 D(原文如此)

在这四种处理机中，“将寄存器 B 加入寄存器 A”(A:=A+B)这一操作都由一条单字节指令实现。制造厂的助记符是：

6800(十六进制 1B)	ABA
8080, 8085(十六进制 80)	ADD B
Z80(十六进制 80)	ADD A, B

这三种不同的记号完成的是同样的操作!

“助记符”是从六十年代的大型系统中沿用下来的。在那些系统中这本来就不太令人满意；使用前必须先学习这些助记符，而且用助记符书写的程序很难理解。

本书中不使用这种类型的助记符，而用下面将介绍的一种符号系统来表示指令。这种办法有两个明显的不利因素：(i)这是一种新的表示法；(ii)还没有权威人士使用过这种表示法。鉴于这种表示法具有下述优点，第二个不利因素迟早会消除的。它的优点是：

(i) 在大多数 ASCII 码键盘所能提供的字符范围内，可协调地和简洁地表示各条指令。

(ii) 这种表示法是以(国际信息处理联合会——IFIP) ALGOL 组的论文提出的概念为基础的，有训练的程序员只需作很少的解释就可阅读和理解。

(iii) 若几种机器有同样的操作，则可使用相同的表示式。

(iv) 这种表示法在实践中已表明它能成为程序设计语言的良好基础。后面将会介绍供我们讨论所用的机种使用的这类语言。占用 $2\frac{1}{4}$ K PROM 的 8080 汇编程序已从 1977 年 3 月开始使用。第八章列出了完整的文本。这个汇编程序可改进成只需 2K 存贮器而完成同样的功能，或扩充至 3K 或 4K 而提供其它的功能。与此相应的 6800 和 Z80 汇编程序正在研制中。一

个学生在三星期里就写成了一个供 6800 用的 ALGOL W 交叉汇编程序。有关实现高级语言的可能性的讨论可参阅第七章。

(v) 反汇编程序是一种用于打印机器中程序的服务程序(被打印的程序用本书中的这种表示法书写)。反汇编程序可以编得非常紧凑(对于 8080、8085 和 6800 而言,都可小于1K——见§ 6.1 和§ 8.4)。这是一种非常有用的"监视"手段,可使程序以易读的形式打印出来。

§2.3 通用的表示法

2.3.1 指令各字节的表示方法

除了以 CB、DD、ED、FD 开头的 Z80 指令以外,其它指令的各字节的表示方法是:

(i) 指令的第一个字节用 I① 表示。假定该字节存在主存贮器的第 n 号单元内。

(ii) 二字节或三字节指令的第二个字节用 J 表示,存在第 $(n+1)$ 号单元内。

(iii) 三字节指令的第三个字节用 K 表示,存在第 $(n+2)$ 号单元内。

而以 CB、DD、ED、FD 开头的 Z80 指令中各字节的表示方法是:

(i) 第一个字节用 I_0 表示,存在第 n 号单元内。

(ii) 第二个字节用 I_1 表示,存在第 $n+1$ 号单元内。

(iii) 第三个字节(如果有的话)用 J 表示,存在第 $n+2$ 号单元内。

(iv) 第四个字节(如果有的话)用 K 表示,存在第 $n+3$ 号

① 符号 I、J、K 还有其它的意义(分别表示"中断标志"、"转移"和"进位标志"),但是从上下文总可分清其两种含义)。

单元内。

2.3.2 寄存器

单字长(8 位)寄存器(或其内容)用一个单字母标识符来表示。例如在 8080 中，A、B、C、D、E、H、L 分别代表一个 8 位的寄存器或寄存器中的内容。

双字长(16 位)寄存器(或其内容)用双字母标识符来表示。例如 6800 的 IX、SP、PC 等就是双字长寄存器。

2.3.3 赋值(信息的复制或传送)

符号“:=”(“变为”)的意思是左边规定的寄存器、标志、存贮单元或输出设备接受符号右边规定的数值。

例如，A:=B 表示“寄存器 A 接受 B 中的信息”。A 中原来的信息将丢失，而 B 中的信息保持不变。简言之就是“A 变为 B”。

A:=12 意思是“寄存器 A 接受数值 12”。A 中原来的信息将丢失(12 是十六进制数——见下文)。

在 6800 中，这些操作还能设定某些条件标志值，而在 8080、8085 和 Z80 中则没有这种功能(见“条件标志”一节，以及§ 3.2 中有关 6800 指令系统的说明)。

2.3.4 信息交换

符号“:=:”的意思是符号左边和右边规定的寄存器或存贮单元的内容相互交换。

例如，HL:=:DE 意思是“HL 存入 DE 中原来的值，DE 存入 HL 中原来的值”。这个操作符是互逆的。若无此操作的话，要达到同样的目的需用一个辅助存贮单元和三次赋值操作，即：

$$\left.\begin{array}{l}XY:=DE\\DE:=HL\\HL:=XY\end{array}\right\}\quad 或\quad \left\{\begin{array}{l}XY:=HL\\HL:=DE\\DE:=XY\end{array}\right.$$

在 8080、8085 和 Z80 中都可使用交换操作符。

2.3.5 · 数值

数值就是一组(一个或多个)二进制数字。每位数字可以是 0 或 1。四位一组的数值(有时称为"半字节")可表示成一个十六进制(Hex)数字,因此有下列对应关系:

二 进 制	十六进制	二 进 制	十六进制
0000	0	1000	8
0001	1	1001	9
0010	2	1010	A
0011	3	1011	B
0100	4	1100	C
0101	5	1101	D
0110	6	1110	E
0111	7	1111	F

一个"字节"是 8 位(两个"半字节"),即一个机器字①。一个字节可以表示成两个十六进制数字:

例如:

01001110	4E
00000000	00
11111111	FF

(注: 在赋值操作 C:=12 后,寄存器 C 中就存有 00010010;
在赋值操作 C:=BC 后,寄存器 C 中就存有 10111100。
第二个例子也不会产生混淆,因为尽管有标志符为 BC 的双字长寄存器,但是左边的是单字长寄存器的标识符,所以右边必须是单字节的数值。)

① 由于 6800、8080、8085 和 Z80 中操作数的长度一般是 8 位,所以 8 位就是一个机器字。其它处理机可有其它长度的机器字,例如 4、12、16、24、27 (原文如此——译注)、32、40、48 位。"字节"这个词以前有时表示 6 位的数据,但是现在似乎专门用来表示 8 位的数据。

一个字节中的各位从右边开始分别编成第 0～7 位。

Z80 中有对个别位进行操作的指令。

用来规定某一位或某一位的位置的表示方法是一“点”后面紧跟着一个位号。

例如：A.0:=1　意思是规定寄存器 A 中最右边的一位是“1”(不影响 A 中其它位的内容)。

A.7:=0　意思是规定寄存器 A 中最左边的一位是“0”。

A.0, A.7 在表格和说明中写成 A_0、A_7。

双字长(16 位，两个字节)数值可表示成四个十六进制数字。

例如：在 SP:=147A 操作之后，SP 中存有 0001010001111010。

除非另有说明，否则所有明确的单字长和双字长数值都将用十六进制数字表示(表 2.1 是十六进制与八进制和十进制表示法的对照表)。即使前一位数字是 0 也要写明，因此单字长的数总是两个数字，双字长的数总是四个数字。

2.3.6 地址和存贮单元

给定一个 16 位的地址，就能对 2^{16}(十进制是 65536)个存贮单元之一进行访问。

若给定的地址是 1234，对应的存贮单元就表示成 M1234 (M 表示“主存贮器”)。

在实际的系统中，有些存贮单元是空的(即没有安装存贮器件)，有些配有 ROM 或 PROM，有些则配有 RAM②。

A:=M1234 意思是“将存贮单元 M1234 (地址为 1234 的存贮单元)中的数值送入寄存器 A”。若该单元是 ROM、PROM

② 参阅 §2.8“输入和输出”。

或 RAM 的一个单元，则可执行这个操作。

M1234:=A 意思是"将寄存器 A 中的数值送入地址为 1234 的存贮单元"。这个"写"操作只有对 RAM 单元才能执行，对 ROM 和 PROM 不能执行该操作。

A:=M[DE] 意思是"将由双字长寄存器 DE 中的内容决定的存贮单元中的数值送入寄存器 A"。(DE:=1234 后面跟 A:=M[DE] 的与 DE:=1234 后面跟 A:=M1234 的作用相同)。

记号 MM1234 表示一对存贮单元 M1234 和 M1235。因此：

HL:=MM2345 意思是"将 M2345 和 M2346 两个单元中的双字长数值送入双字长寄存器 HL"。

MM2345:=IX 意思是"将双字长寄存器 IX 中的双字长数值送入一对存贮单元 ——M2345 和 M2346"。

在这类操作中，哪一半寄存器与两个存贮单元中哪一个相对应则视具体的处理机而定。单字长寄存器的各位编号是从 0 (最右边)编到 7(最左边)，与存贮单元中各位的编号一致。在单字长赋值操作中，数据"源"中的一位被送入"目的地"中对应位中去。双字长寄存器的各位编号是从 0(最右边)编到 15 (最左边)。0～7 位是"低字节"，8～15 位是"高字节"。

HL:=MM2345 是一条在 8080、8085 或 Z80 中有意义的指令。双字长寄存器 HL 实际上是由两个单字长寄存器 H 和 L 拼成的，H 是高字节，L 是低字节。双字长赋值操作相当于两个单字长赋值操作③ H:=M2346 和 L:=M2345。应该注意的是，高字节与高地址存贮单元相对应。

MM2345:=IX 是一条在 6800 中有意义的指令。若 IX 的高、低两半部分分别记作 IX_U 和 IX_L，则双字长赋值操作相

③: 这个操作没有独立的指令。

当于两个单字长赋值操作④ M2345:=IX_U 和 M2346:=IX_L。应该注意，这时高字节与低地址存贮单元相对应。

对于这两种习惯用法持赞成和反对意见的都有。不幸的是，各种处理机在这方面采用的是不同的用法，程序员们对此非常不满。

本节中所有的例子都是三字节指令，只有 A:=M[DE] 例外，该指令是一条单字节指令。下表列出了各种指令内容以及能使用这些指令的处理机中相应的机器码。在除 A:=M[DE] 外的所有例子中，其寻址方式被称为是对存贮器直接寻址的，地址是指令的一部分(字节 J 和 K)，在书写时明确地写明。在 A:=M[DE] 中，其寻址方式被称为间接寻址，因为地址不是由指令本身给出，而必须从寄存器(DE 是一个"寻址寄存器"，见 § 1.2.10)中取得。

指令的书写形式	处 理 机	机器码 I	J	K
A:=M1234	6800	B6	12	34
	8080/8085/Z80	3A	34	12
M1234:=A	6800	B7	12	34
	8080/8085/Z80	32	34	12
A:=M[DE]	8080/8085/Z80	1A	—	—
HL:=MM2345	8080/8085/Z80	2A	45	23
MM2345:=IX	6800	FF	23	45

注意：指令 HL:=MM2345 不要与 HL:=2345 混淆。后者的意思是"给双字长寄存器 HL 赋值 2345"。

除了上述一般形式的直接寻址方式外，6800 还有一种特殊的直接寻址方式。操作 A:=M0012 可由三字节指令"B6 00 12"

④：这个操作没有独立的指令。

实现，但是也能由双字节指令“96 12”(这里写作“A:=MZ12”)实现。0000—00FF(256个单元)范围内的地址特殊一些，因为它有操作码提供前面两个“0”，只剩下两个十六进制数字(一个字节)要明确地写出(J字节)。因此，

M0034:=A 表示 B7 00 34——普通的直接寻址⑤

MZ34:=A 表示 97 34——特殊的直接寻址⑥

在某些微处理机中还可能出现的一种寻址方式是变址寻址。这时，所需的地址是变址寄存器中的数值与指令明确规定的数值(字节J或K)相加之和(见§2.4.3.4)。例如，在6800和Z80中，用M12X表示“地址为变址寄存器IX的内容与明确给出的数值12之和的存贮单元”。

考虑两条连续的指令 IX:=3456, M23X:=A。按此顺序执行这两条指令的结果是寄存器A的内容送入了第3479(3479=3456+23)号存贮单元，变址寄存器IX中存有3456。A的内容不变。相加而形成变址地址的两个数值有时被称为“基地址”和“位移量”或“偏移量”。到底哪个是“基地址”、哪个是“位移量”，则由程序员自己决定。

转移指令中的地址需单独进行讨论。程序计数器是一个16位的寄存器，其内容就是将要取出的下一条指令的地址。程序计数器的值一般是按当前执行的指令的字节数逐渐递增的，以便给出下一条指令的地址。但是，当执行无条件转移指令或执行条件转移指令且条件满足时，要求用一个新的数值取代程序计数器中的数值。

这个新的数值可能是转移指令本身给出的数值(类似于直接寻址)，也可能是从一个寄存器得到的数值(类似于间接寻址)。

⑤：制造厂称之为“扩充寻址”。

⑥：制造厂称之为“直接寻址”。

还有一种可能性（类似于变址寻址）是将转移指令给出的正值或负值（见§2.4.3.4）与程序计数器增加后的值相加而得到这个新值的，这就是相对寻址。在动态顺序中，下一条指令可在正常情况（遵循静态顺序）的下一条指令之前或之后相隔许多字节处。

若出于某种原因，一串指令（一段程序）要搬到存贮器的另一个部分，从新的单元开始执行，则该段程序中的所有转移指令都必须改变，其中包括从这串指令中的一条指令直接或间接转移至该串指令中另一条指令去的转移指令。但是这一串指令内的相对转移指令却不必改变。

表示这些形式的转移指令用的记号可用下述例子来说明：

J2345　无条件转移到地址为2345的单元中的指令去。

J[HL]　无条件转移到单元地址为寄存器HL中的内容的指令去。

J+12　无条件转移到比静态顺序中下一条指令的地址大12的地址中的指令去（注：J+00是一条空指令！）。

若假定程序计数器（PC）按指令的字节数正常增加的操作在执行指令之前完成（微处理机中就是这种情况），则这些指令的作用可表示如下：

指令	正常的增量	执行结果	最后结果
J2345	PC:=PC+3	PC:=2345	PC:=2345
J[HL]	PC:=PC+1	PC:=HL	PC:=HL
J+12	PC:=PC+2	PC:=PC+12	PC:=PC+14

（参照：

A:=B　PC:=PC+1　A:=B　PC:=PC+1; A:=B）

若象最后一栏那样用“PC:=…”来表示转移指令（当然也能正常地执行），则必须表示出所有其它指令中程序计数器的正常增

量值。用"J"记号不仅可避免表示正常的增量值，而且正象在打印单上一样一眼就可认出转移指令来。用"J"表示转移（书写时总是指令的第一个符号）不应与表示第二个字节的"J"相混淆。

2.3.7 条件标志

条件标志是一个一位的存贮器（触发器），在执行某些机器指令时会被置位（置1）或复位（置0）。

每个条件标志用一个单字母标识符来表示，例如"Z"表示"零标志"。$Z=1$ 一般意味着执行的最后一条算术或逻辑运算的结果为零（所有的各位均为0），$Z=0$ 表示结果非零（至少有一位为1）。还可能有给个别标志赋值（规定值）的机器指令。不同的处理机对条件标志的处理方法也不同。例如，8080、8085或Z80中的指令"A:=00"并不影响任何条件标志，但在6800中却会影响条件标志，特别是会使Z置1。各种微处理机处理条件标志的方法是程序员必须仔细研究的内容之一，这样才能明确其真正的意义（也许应根据处理标志信息的方法来判断一台微处理机）。

可将一组条件标志（一般小于8位）看作为一个条件标志寄存器。也有指定这一组标志的值或复制其内容的指令。

例如在6800中，六个条件标志H、I、N、Z、V、K与两个常数位一起构成了一个单字长寄存器C，6800的指令系统中包括有A:=C和C:=A等操作。

词头N表示一个条件标志值的反码，例如NZ表示$\overline{Z}$（零标志不置1）。

§2.4 算术和逻辑运算

2.4.1 一个字节的意义

存贮器或寄存器中的一个字节可被处理机程序员当作若干

种不同类型的量来进行处理，这些量包括：

(i) 八个互不相关的位；

(ii) 0～255(十进制)范围内的无符号二进制整数；

(iii) −128～+127(十进制)范围内的带符号二进制整数；

(iv) 操作码；

(v) 16 位地址的一半；

(vi) 一个外围设备的设备号；

(vii) 某种外围设备采用的字符代码中的一个字符。

很难一下就说出存在任意存贮单元中的一个字节最终将被机器当作何种量进行处理。若该字节被送入指令寄存器，则将被当作一个操作码来处理。若进入一个累加器，则会被当作一个数来处理。若进入一个寻址寄存器，则将被当作一个地址的一半来处理，如此等等。写得不正确的转移指令会使程序转移到一条指令的第二个或第三个字节去。若发生了这种情况，则程序将迷路，其执行顺序就无法预言了。在进行程序测试时必须记住这一点。

构成一个字节的各位规定如下：

d_7	d_6	d_5	d_4	d_3	d_2	d_1	d_0

在赋值操作(例如 A:=B, M12X:=A)中，进入目的地的各位信息的相对位置将保持不变。

2.4.2 带符号和无符号的数值

一个字节作为无符号二进制整数时，所表示的数值为：

$$\sum_{i=0}^{7} d_i 2^i$$

例如：10101100(AC) 表示 $2^7+2^5+2^3+2^2=128+32+8+4$
$=172$ (十进制)

00000000 (00) 表示 0 (十进制)
01111111 (7F) 表示 127 (十进制)
10000000 (80) 表示 128 (十进制)
11111111 (FF) 表示 255 (十进制)

该字节作为带符号二进制整数时，所表示的数值为：

$$-d_7 2^7+\sum_{i=0}^{6} d_i 2^i$$

因此：10101100(AC)表示 $-2^7+2^5+2^3+2^2=-128+32+8+4=-84$(十进制)

00000000 (00) 表示 0 (十进制)
01111111 (7F) 表示 +127(十进制)
10000000 (80) 表示 −128(十进制)
11111111 (FF) 表示 −1 (十进制)

这里 d_7 称为"符号位"；若 $d_7=1$，其值为负；若 $d_7=0$，其值为正。这种表示带符号数的方法称为二的补码表示法。与另一种(也许更直观的)"符号与模数"表示法相比，二的补码表示法的优点是相当多的。在"符号与模数"表示法中，数值表示成：

若 $d_7=0$，则为 $\sum_{i=0}^{6} d_i 2^i$；若 $d_7=1$，则为 $-\sum_{i=0}^{6} d_i 2^i$。

在我们讨论的微处理机中，不采用"符号与模数"表示法。

若一个字节表示的是带符号的或无符号的数，则 d_7 是其"最高位"(MSB)，d_0 是其"最低位"(LSB)。

也可用一对字节表示一个无符号或带符号的二进制整数，此时高字节(参阅§2.3.6)的各位要重新编号。高字节的最高位变成 d_{15}，高字节的最低位变为 d_8。

因此，若为一个无符号的二进制整数，这对字节表示的值就是：

$$\sum_{i=0}^{15} d_i 2^i \quad (\text{范围是十进制 } 0\sim 65535)$$

若为带符号二进制整数，这对字节表示的值就是：

$$-d_{15}2^{15}+\sum_{i=0}^{14} d_i 2^i \quad (\text{范围是十进制 } -32768\sim +32767)$$

注意：在这种情况下，高字节的最高位是符号位。

因此，1010101111001101(ABCD)，作为一个无符号数则表示 43981(十进制)；而作为一个带符号数则表示 −21555(十进制)。

注意： AB 表示 171 (无符号，十进制)或 −85(带符号，十进制)。CD 表示 205(无符号，十进制)或 −51 (带符号，十进制)。因此，作为一个无符号数：

ABCD 表示 $171\times 2^8+205=43776+205=43981$(十进制)；作为一个带符号数：

ABCD 表示 $-85\times 2^8+205=-21760+205=-21555$ (十进制)。

ABCD 并不表示 $-85\times 2^8-51$。因为只有数的最高位才是符号位，所以只有高字节才是带符号数。

很清楚，“多字长数”可以由任意个字节组成。若这个数是带符号的，符号位就是最高字节的最高位。

2.4.3 加法

2.4.3.1 单字长加法

(i) 两个无符号单字长数的加法

由于两个无符号单字长数都在 0～255 (十进制)的范围内，所以其和在 0～510(十进制)的范围内。这个范围要用 9 位二进制来表示，其中 8 位可放在单字长寄存器(累加器)中，第 9 位(最高位)存在进位标志 K 中。若执行加法以后 K=0，和数就

在0～255的范围内(即为单字长数)。若K=1，和数的范围是256～510，累加器中存的是比和数小256的数。

(ii) 两个带符号单字长数的加法

由于两个带符号单字长数都处在 −128～+127 (十进制)的范围内，因此其和数处在 −256～+254 的范围内。处理机将象处理无符号数那样用累加器和进位标志把两个数加起来。此时必须区分两种情况：累加器中的结果代表真正的算术和的带符号数的情况以及与此相反的情况。还必须考虑进位标志的意义。

(a) 若两个操作数都是正的，则其代数和也是正的。但是，若和数超过 +127(不会超过 +254)，则累加器中的结果将出现负数(因为其最高位是1)。在任何情况下进位标志总是0。所以，当相加两数均为正时，若累加器的 d_7 位出现1，则表示溢出——结果超出了单字长寄存器的容量。

大多数微处理机都有条件标志N(负)。当操作后结果的第 d_7 位为1，则该标志置位(N:=1)；若结果的 d_7 位为0，则该标志复位(N:=0)。6800和Z80还有一个标志"V"(溢出)，而8080和8085则无此标志。出现上述溢出情况时，该标志置位(V:=1)，否则就复位(V:=0)。

(b) 若两个操作数都是负数，其代数和也是负数，范围为 −256～−2。若和数处在 −128～−2 内，则其值可正确地存在单字长寄存器中，进位标志将置位(K:=1)。但是，若代数和在 −256～−129 范围内，进位标志将置位(K:=1)，但是累加器中存的数表面上却成了一个正数(N:=0)。因此，当相加两数均为负数时，若累加器的 d_7 位出现0，则表示溢出(V:=1)。

(c) 若操作数为一正一负，则其代数和肯定在−128～+126的范围内，累加器中将存有正确的结果。若结果为正 (≥0)，

则进位标志将置位(K:=1)；若结果为负(<0)，则进位标志复位(K:=0)。如果有溢出标志的话，则也将被复位(V:=0)。

总之，若两个带符号单字长数之和是一个带符号单字长数，则单字长累加器就能给出正确的结果。若和数超出了带符号单字长范围(只有当两数符号相同时才有可能)，则累加器的符号位(或用N标志也一样)与操作数的符号位不同，或者如果有溢出标志的话该标志置位，都表示结果是不正确的。

加法用符号"+"来表示。符号":+"表示加入累加器。

因此，A:+B (这实际上是 A:=A+B 的简写)表示"将累加器A的内容与B的数值相加，其和取代累加器A的内容"，或简称为"B加入A"。该操作还自动完成一些附加动作，如设置适当的条件标志以及使程序计数器增量等。

A:+B表示一条单字节指令；第二个操作数存在一个寄存器(B)中。

A:+43表示一条双字节指令；第二个操作数(43)在J字节中，被称为立即数。

2.4.3.2 双字长加法

(i) 两个16位无符号数的加法

双字长数 p 可由两个无符号单字长数 p_1 和 p_0 表示，即 $p=2^8p_1+p_0$。这里 p_1 是高字节，p_0 是低字节。

若 $p=2^8p_1+p_0$，$q=2^8q_1+q_0$，且 $r=p+q$；则 $2^8(p_1+q_1)+(p_0+q_0)=2^8r_1+r_0$。

由于 $0\leqslant p,\ q\leqslant 2^{16}-1$，就有 $0\leqslant p+q\leqslant 2^{17}-2$，所以 r 一般需要十七位。

若形成 p_0+q_0，则会得到一个9位的结果。很清楚，r_0 是低8位——单字长加法后累加器的内容。在形成 r_1 时，第9位(其值在进位标志中，称为 k_0)也必须考虑进去。必须形成和数

$p_1+q_1+k_0$。这个和数的低8位(累加器的内容)是 r_1 的低8位，第9位(其值在进位标志内，称为 k_1)就是 r_1 的第9位，即 r 的第17位。

这个过程可概括如下：

$2^8k_0+r_0$ 是 p_0 与 q_0 之和；

$2^8k_1+r_1$ 是 p_1、q_1 与 k_0 之和；

$r=2^{16}k_1+2^8r_1+r_0$，式中 r_1、r_0 是单字长数。

为便于进行双(多)字长加法，有一条称为"带进位加"的指令。这个操作表示成":++"，例如 A:++B。

A:++B (是 A:=A+B+K 的简写)意思是"B 与 K 加入 A"。

利用微处理机提供的这种操作，就可用下述指令串来完成双字长加法：

A:=p_0

A:+q_0　(执行该指令后 A 中为 r_0，K 中为 k_0)

r_0:=A
A:=p_1　} 这些指令不影响 K

A:++q_1 (执行该指令后 A 中为 r_1，K 中为 k_1)

r_1:=A　(若 K 被置位，即 k_1="1"，则结果超出双字长的范围)。

(ii) 两个16位带符号数的加法

无论带符号数有多长，都只有一个符号位。一个带符号双字长数 p 可用一个带符号单字长数 p_1 和一个无符号单字长数 p_0 来表示，表示的方法是：

$$p=2^8p_1+p_0$$

若

$$p=-2^{15}d_{15}+\sum_{i=0}^{14}d_i2^i,$$

则 $$p_1=-2^7d_{15}+\sum_{i=0}^{6}d_{i+8}2^i \quad (\text{带符号部分})$$

和

$$p_0=\sum_{i=0}^{7}d_i2^i \quad (\text{无符号部分})$$

带符号双字长数的范围是 $-2^{15}\leqslant p\leqslant+2^{15}-1$（十进制 $-32768\leqslant p\leqslant+32767$）。

若两个带符号双字长数 p、q 的代数和处于这一范围内，则利用上述无符号数双字长加法用的程序也可得到正确的带符号双字长结果 $r=p+q$。若其和超出了双字长范围，则可由溢出标志（如果有的话）或由对操作数的带符号部分执行“带进位加”的结果来表示。

上文(i)中的程序中所用的指令在 6800、8080、8085 和 Z80 中都有（p_0、p_1、q_0、q_1 的存贮单元要受一定的限制）。8080、8085 和 Z80 还有一条执行双字长加法用的指令（例如 HL:+DE）。结果的第 17 位在进位标志 K 中。

若两个双字长整数分别存在 MM1234 和 MM1236 中，要求将其和存在 MM1238 中，则可用下述指令串之一实现加法：

(i) (6800):A:=M1235
A:+M1237
M1239:=A
A:=M1234
A:++M1236
M1238:=A
(18 个字节，使用寄存器 A)

(ii) (6800):IX:=1234
A:=M01X
A:+M03X

M05X:=A

A:=M00X

A:++M02X

M04X:=A

(15个字节，使用寄存器A、IX)

(iii) (8080、8085、Z80)：

HL:=MM1234

HL:=:DE(见§2.3.4)

HL:=MM1236

HL:+DE

MM1238:=HL

(11个字节，使用寄存器D、E、H、L)

2.4.3.3 *多字长加法*

简单地将双字长加法程序加以扩充就可达到这一目的。最低一对字节之和由简单的加法形成，其它各对对应的字节(从低到高)之和则用带进位加法形成。逻辑设计人员将会发现，"简单加法"类似于半加器的作用，"带进位加法"类似于全加器的作用。

Z80有双字长带进位加法指令，可使字长较长的加法所需的步骤减少一半。例如HL:++BC。

2.4.3.4 *字长不等的数的加法*

地址计算就是字长不等的数的加法之一。

(i) 在变址寻址中，将由一个双字长数(在变址寄存器中)与一个单字长数(J字节)加起来形成地址。这两个数基本上是无符号的。整个操作可看作先给单字长数补上由8位"0"组成的高字节使其形成双字长数，然后再执行一般的双字长加法。

(ii) 在相对寻址中，单字长数被看作是带符号的，因此转移

既可向前(J为正)也可向后(J为负)。加法操作可看成是给单字长数补上由8个与低字节的最高位相同的位组成的高字节，使其变成双字长数，然后执行一般的双字长加法。

一般说来，长度不等的两个数相加时，若先给较短的字的左边补上足够的"填充位"，使其与较长的字长度相等，则就可执行加法。若该数是带符号数，则填充位必须与其最高位相同(见§4.5.1)。

2.4.4 取反和取负

在无符号表示法中，字节 $d_7d_6d_5d_4d_3d_2d_1d_0$ 表示一个数

$$p=\sum_{i=0}^{7} d_i 2^i$$

现在考虑将该字节各位取反(即0变1，1变0)形成的字节 $\bar{d}_7\bar{d}_6\bar{d}_5\bar{d}_4\bar{d}_3\bar{d}_2\bar{d}_1\bar{d}_0$。

上述取反后的字节所表示的数是：

$$\bar{p}=\sum_{i=0}^{7}\bar{d}_i 2^i=\sum_{i=0}^{7}(1-d_i)2^i=(2^8-1)-p$$

在带符号表示法中，若原来的字节表示为：

$$q=-2^7d_7+\sum_{i=0}^{6} d_i 2^i$$

则取反后的字节表示为：

$$\bar{q}=-(1-d_7)2^7+\sum_{i=0}^{6}(1-d_i)2^i$$

$$=-2^7+(2^7-1)+2^7d_7-\sum_{0}^{6} d_i 2^i=-1-q$$

每种微处理机都有一条对累加器内容取反的单字节指令，这条指令可写成：

A:＃　(意思是 $A:=\bar{A}$)

现在假设在取反后的字节上加01(十六进制)。在无符号

表示法中，就有

$$\bar{p}+1=(2^8-1)-p+1=2^8-p$$

只有当原字节是00(十六进制)时进位标志才会被置位。

在带符号表示法中，加法后可得：

$$\bar{q}+1=-1-q+1=-q$$

只有当原字节是80(十六进制，十进制为 -128)时，结果才会超出单字长的范围。在这种情况下结果不变(因为80取反后得7F，再加01又得80)，但是出现一个溢出状态，如果有溢出标志的话就会被置位。因此，除非A中的值为80(十进制为-128)，否则A:# 后面跟一条A:+01指令就可使A中的带符号数取负。如果原数不为00，那么用这两条指令对无符号数进行操作就给出该值相对于2^8的补码(即原数为p，则得2^8-p)。如果原数为00，则进位标志被置位。

6800和Z80都有一条单字节指令，其作用相当于A:# 后面跟一条A:+01。这条指令写作A:−(是A:=−A的简写)。而8080和8085则没有这样的指令。

双字长(或多倍字长)数的负数可这样来形成：每个字节都取反，然后再加1(注意，取反必须在加1之前完成)。

2.4.5 减法

2.4.5.1 单字长减法

(i) 两个无符号单字长数的减法

由于两个无符号单字长数p、q的范围均在0～255(十进制)之内，因此其差$p-q$的范围也在-255～(+)255之内。处理机中的减法操作要使用一个累加器和进位标志。若$0\leqslant p-q\leqslant 255$，则累加器中的结果是正确的，进位标志被复位(K:=0)。

若$-255\leqslant p-q\leqslant -1$，则将借入256($2^8$)，累加器中的数是$p-q+256$。由于$-255\leqslant p-q\leqslant -1$，就有(+)$1\leqslant p-q+256$

$\leqslant(+)255$，这个量处在无符号单字长数的范围内。借位状态则通过设置进位标志(K:=1)来保存。

(ii) 两个带符号单字长数的减法

由于两个带符号单字长数 p、q 的范围均在 $-128\sim+127$ (十进制)之内，因此其差 $p-q$ 的范围也在 $-255\sim+255$ 之内。可以象在讨论带符号单字长数的加法时那样考虑各种可能性。机器将把 p 和 q 当作无符号数进行处理。

(a) 若 p、q 均为正数，则 $-127\leqslant p-q\leqslant+127$，累加器中的结果是正确的。若结果为负，则将进位标志置位；若结果为正，则将进位标志复位。

(b) 若 p 为正数、q 为负数，则 $+1\leqslant p-q\leqslant+255$。若 $+1\leqslant p-q\leqslant+127$，则累加器中的结果是正确的；若 $+128\leqslant p-q\leqslant+255$，则累加器中将存有一个负数$(p-q-256)$，如果有溢出标志的话将被置位。

(c) 若 p 为负数、q 为正数，则 $-255\leqslant p-q\leqslant-1$。若 $-128\leqslant p-q\leqslant-1$，则累加器中的结果是正确的；若 $-255\leqslant p-q\leqslant-129$，则累加器中将是一个正数$(p-q+256)$，如果有溢出标志的话将被置位。

(d) 若 p、q 都是负数，则 $-127\leqslant p-q\leqslant+127$，累加器将存有正确的结果。若结果为负，则将进位标志置位；若结果为正，则将进位标志复位。

总之，若两个带符号单字长数之差是一个带符号的单字长数，则单字长累加器中就是正确的结果。若差值超出了带符号单字长数的范围(只有当操作数符号相反时才会发生这种情况)，可由累加器的符号位(或 N 标志也一样)与减数(q)的符号相同，或者由溢出标志(如果有的话)的置位状态来表示结果不正确。

减法用符号“－”来表示。符号“:－”表示减“去”累加器(即从累加器中减去)。因此，A:－B(A:＝A－B的简写)的意思是“从A的值中减去B的值，其差值取代A中的值”。与此相比，A－B的意思只是“形成两者之差，但不取代A或B中的任何一个”，见§2.4.7“测试和比较操作”。与其它的算术运算一样，对标志和程序计数器的附加操作也是自动完成的。

A:－B表示一条单字节指令，第二个操作数在一个寄存器(寄存器B)中。

A:－43表示一条双字节指令，第二个操作数(43)在J字节中，是一个立即数。

2.4.5.2 双字长减法

(i) 两个16位无符号数的减法

采用与双字长加法一节对应的表示法，并写作 $s=p-q=2^8s_1+s_0$，则从 p_0 中减去 q_0 可得 -2^8k_0 (进位标志)和 $+2^8k_0+p_0-q_0$ (在累加器中)。在第二阶段，必须从 p_1 中减去 q_1 和进位标志值。所以，利用带进位减法(也称为带借位减法)指令可得 -2^8k_1(进位标志)和 $+2^8k_1+p_1-q_1-k_0$ (在累加器中)。

因此，
$$s_0=2^8k_0+p_0-q_0$$
$$s_1=2^8k_1+p_1-q_1-k_0$$

可得
$$s=-2^{16}k_1+2^{16}k_1+2^8p_1-2^8q_1-2^8k_0+2^8k_0+p_0-q_0=2^8(p_1-q_1)+(p_0-q_0)$$
$$=p-q \quad \text{(所需的结果)}$$

微处理机都有“带进位减法”或“带借位减法”操作，这里用“－－”来表示。例如A:－－27(是A:＝A－K－27的简写)就是实现该操作的指令。利用这种类型的指令就可通过下述指令串来完成双字长减法：

A:=p_0

A:−q_0　(执行此指令后 A 中为 s_0, K 中为 k_0)

s_0:=A
A:=p_1　} 这些指令不影响 K

A:−−q_1 (执行此指令后 A 中为 s_1, K 中为 k_1)

s_1:=A　(若 K 被置位，即 k_1=1 表示结果超出双字长的范围)。

(ii) 两个 16 位带符号数的减法

若两个带符号双字长数 p、q 的代数差 $p-q$ 在带符号双字长数的范围内，则上述无符号双字长数减法用的程序可给出正确的带符号双字长结果 $s=p-q$。若两者之差超出了带符号双字长数的范围，则可用溢出标志(如果有的话)或对操作数带符号部分的"带进位减"的结果来表示。

上一段(i)中的程序所用的指令在 6800、8080、8085 和 Z80 中都有。Z80 还有一条"带借位双字长减法"指令，例如 HL:−−BC(十六进制 ED42)。若保证在执行指令之前进位标志被复位，则这条指令也可用于简单的双字长减法 (没有 HL:−BC 指令)。执行该指令后 K 中的值与 (i) 中的程序所设置的值相同，因此也与 V 中的值相同。

2.4.5.3　多倍字长减法

这种操作完全与多字长加法相类似(§ 2.4.3.3)。

2.4.5.4　字长不等的数的减法

有关字长不等的数的加法一节(§ 2.4.3.4)中最后一段的结论同样适用于减法。

2.4.6　逻辑(布尔)操作

有关逻辑取反(非)、逻辑乘(与)、逻辑加(或)和不等(异或)的操作定义如下。X、Y 是逻辑变量(单个的位)。

取反

X	非X
0	1
1	0

(在大多数课本里写作 $\neg X$ 或 $\overline{X}_0$)

逻辑乘

X	Y	X与Y
0	0	0
0	1	0
1	1	1
1	0	0

(常写作 $X \wedge Y$, XY, X.Y 或 X&Y)

逻辑加

X	Y	X或Y
0	0	0
0	1	1
1	1	1
1	0	1

(常写作 $X \vee Y$ 或 X+Y)

异或:

X	Y	X异或Y
0	0	0
0	1	1
1	1	0
1	0	1

(常写作 $X \forall Y$, $X \oplus Y$, $X \neq Y$ 或 $X \neq Y_0$)

如对8位的字执行这样的操作，则规定为是对各位("非")或对应的各位(与、或、异或)平行地进行操作。因此:

若 p 为 10110010(B2)

且 q 为 00101111(2F)

则 $\overline{p}$ 为 01001101(4D)

$p \wedge q$　　为 00100010(22)

$p \vee q$　　为 10111111(BF)

$p \forall q$　　为 10011101(9D)

微处理机都有执行这些 8 位逻辑操作的单字节指令，这些指令要使用一个累加器，且操作的结果取代累加器的内容。因此：

对于“非”操作就有　$A:=\overline{A}$

对于“与”操作就有　$A:=A \wedge q$

对于“或”操作就有　$A:=A \vee q$

对于异或操作就有　$A:=A \forall q$

这里 A 表示累加器中的数，q 可以是另一个寄存器或一个立即数。

普通的控制台键盘上都没有 ∧、∨、∀、⊕、¬ 等符号，也无法在字上画线；而“+”号则用于加法。所以这里采用下述记号：

A:#　表示 $A:=\overline{A}$

A:&q　表示 $A:=A \wedge q$

A:∪q　表示 $A:=A \vee q$

A:#q　表示 $A:=A \forall q$

8080、8085 和 Z80 都有将进位标志值取反的指令，这条指令将写作：

K:#

2.4.7　“测试”和“比较”操作

上面介绍的单字长算术指令都会影响进位标志 K 和(6800、Z80 中的)溢出标志 V，还会影响另外两个标志：Z (零)和 N (负)。在执行这些操作以后：

若操作结果为零(00000000)，则 Z=1；

若结果非零，则 Z=0；

若结果为负，即 $d_7=1$，则 N=1；

若结果为正或零(非负，$d_7=0$)，则 N=0。

上一节介绍的逻辑运算也会影响这些标志。但是各种处理机对标志的影响是不同的，例如 8080、8085 和 Z80 的指令 A:# 不影响这些标志，但在 6800 中这条指令会影响标志。有关的详细情况可参阅具体的说明(第三章)。

常常会遇到下述这种情形：程序员希望知道两个量之间是否有某种关系——例如两个字节是否相同或一个数是否大于另一个数。

假定已算好一个带符号数存在 A 中，而某处存有另一个带符号数 q。若 A 中的数小于 q，则继续执行程序时必须采用计算出的值；若 A 中的数大于或等于 q，则必须用 q 取代 A 继续进行计算。这种要求在"高级"语言(例如 ALGOL)中表达如下：

$$\text{if A}\geqslant q \text{ then A}:=q \qquad \text{(i)}$$

处理这一问题的一种方法是(每行代表一条微处理机指令)：

```
A:-q          q 减"去" A(从 A 中减去 q)
  Jump if N=1 若结果为负则转移
  A:=q        q 存入 A
  Jump        无条件转移
  A:+q        恢复 A 中的原始值
  (继续)
```

另一种办法是：

```
B:=A
A:-q
```

```
┌─Jump if N=1
│  B:=q
└→A:=B
(继续)
```

由于在这些办法中必须处理计算值与 q 之间的实际差值，所以是很笨拙的。好在我们并不需要研究这个差值，而只需考虑其符号。因此可以用所谓的"比较"操作来解决这个问题。该操作执行减法，并根据其结果设置条件标志，但是结果并不送入累加器。

借助于完成这一操作的指令(所有的微处理机都有)，上例可简化为：

```
    A−q                根据 A−q 的值设置标志
┌─Jump if N=1
│  A:=q
└→(继续)
```

读者可以将这种情形与高级语言表示式(i)比较一下。

表示式 A−q (无冒号)指明结果不送入 A，因此 A 中保持操作前原有的值。这种表示法要比"比较 A、q"的缩写优越，因为前者肯定了所要求的差值是 A−q，而不是 q−A。

读者应该记得，只有当 A−q 的值在带符号单字长数的范围内时，N 才能正确地表示 A−q 的符号(见§2.4.5.1)。若 A、q 的值是无符号的数，则差值 A−q 的符号由 K 给出(若差为正或零，则 K=0；若差为负，则 K=1)，而并非由 N 给出。

在 6800 和 Z80 中，还可使用溢出标志 V。在 6800 中值得注意的还有：根据 K∨Z、N∀V、Z∨(N∀V)及其反码等条件实现的相对转移操作。在执行带符号减法以后，只有当实际差值为负(小于零)时，条件 N∀V 才成立(N∀V=1)(见第三章)。

8080 或 8085 没有溢出标志。在带符号算术运算中若 A－q 可能超出范围，则 K 和 N 就不足以给出其符号。有一种技巧能避免采用双字长减法，即先形成 A＋80 和 q＋80，然后在比较了(A＋80)－(q＋80)以后若 A－q 为负，则将进位标志置位(K＝1)；否则将其复位(K＝0)。只有当 A－q 在单字长范围内时，标志 N 才是正确的。

在 6800 中，还有一些操作可按照某个值设置条件标志。

在写作"A&q"形式的 6800 指令中，会形成"A&q"值，但是不送入 A，且会影响 N 和 Z 的标志。

在简单地写成 A 或 M1234 的"测试"指令中，将根据规定的值(这里就是指 A 中的值或 M1234 中的值)设置标志 N 和 Z。V 和 K 被清除，其它标志均不改变。

还有一个"比较变址寄存器"操作，为简单起见写成 IX－q。读者应该注意，该操作并不涉及双字长减法。若 IX 的高、低两半分别写成 IX_U 和 IX_L，数值 q 的高、低两半写作 q_U 和 q_L，则该操作就好象计算了两个单字长数之差 IX_U-q_U 和 IX_L-q_L 一样。因此，若两个差都是零，则 Z 标志置 1；若 IX_U-q_U 为负，则 N 标志置位；若计算 IX_U-q_U 时发生溢出，则将 V 标志置位；其它标志不受影响。所以实际上只有 Z 给出的是有关双字长差值 IX－q 的直接可用的信息。

2.4.8 "移位"和"循环移位"操作

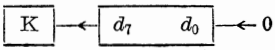
一些有经验的程序员习惯于用一条指令把 32 位或 64 位的数在任一方向上移动 n 位。而在我们讨论的这些微处理机中，一条移位指令只能将一个数移动一位(作为例外，也有移四位的)。所以这些程序员可能会觉得很难办。此外，还会感到奇怪，什么场合需要使用九位信息循环移动一位的循环指令呢？即使在进行了大量的程序设计工作以后，对给定的这几种微处理

机为什么提供这些指令仍然有些莫明其妙。不过，即使有时必须借助于计数循环操作完成移位，但至少这些指令能实现移位并能获得合理的结果。

在6800、8080、8085和Z80这四种微处理机中有九种不同类型的移位或循环移位操作，下面将分别进行介绍。介绍之后还列出了每种处理机所具备的移位操作的一览表。

这四种处理机的进位标志都起了很重要的作用。

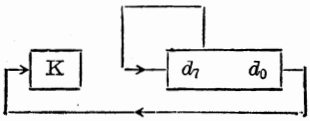
1．*算术左移*(单字长数乘2)



此时符号位 d_7 移入进位标志；d_6 位移入 d_7 的位置；d_5 位移入 d_6 的位置；…；d_0 位移入 d_1 的位置；最后在 d_0 的位置上补以"0"(见上图)。若移位后进位位和符号位相等，移位结果就是一个等于原值两倍的带符号数。若原值被看作是一个无符号数，则状态 K=1 表示原值乘以2后超出了单字长的范围。

这个操作用于单字长量 q 时写作 q:*2。

2．*算术右移*(单字长截尾除2)



此时 d_0 位移入进位标志；d_1 位移入 d_0 的位置；…；d_7 位移入 d_6 的位置；d_7 位上的数字保持不变(操作完成后 $d_7=d_6$)。

这个操作在带符号表示法中相当于除以2；商取代了原来的带符号数，余数(0或1)在进位标志中。由于余数必须为正，所以(举例而言)就有：

3除以2得商为1　　余数为1

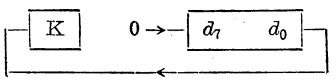
−3除以2得商为 −2　余数为1

−1 除以 2 得商为 −1　余数为 1

−2 除以 2 得商为 −1　余数为 0

商总是单字长数。该操作用于单字长量 q 时写作 q:/2。

3. "逻辑"右移



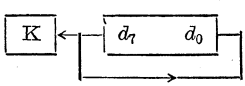
该操作与上述算术右移操作相似，只是最后在 d_7 的位置上补以"0"。

若被移位的量是个无符号整数，则该操作相当于除以 2，用商取代原数，余数(0 或 1)存在进位标志中。

在把被移位的量看作一系列相互无关的"位"的情况下，该操作也许更常用。可利用该操作(顺序地)逐位取出各位数字。

该操作用于单字长量 q 时写作 q:−>。

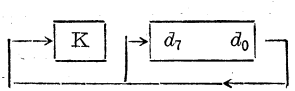
4. 循环左移



在该操作中，K 中移入原来 d_7 位的值，操作数中所有 8 位都向左移一位，原来的 d_7 位还移入 d_0 (见上图)。

这个操作用于单字长量 q 时写作 q:@L。符号 @ 是 ASCII 码之一，是一个长期沿用下来的字符，电传打字机和其它控制台设备上都有这个字符。其形状可提示读者循环移位的概念。

5. 循环右移



在这个操作中，原来 d_0 位的值移入 K 中，操作数中所有 8

位都向右移一位，d_7 位移入原来 d_0 位的数字(见上图)。

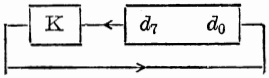
该操作用于单字长量 q 时写作 q:@R。

(练习：下述操作的作用如何?

(i) q:@L 后面紧跟着 q:@R;

(ii) q:@R 后面紧跟着 q:@L。)

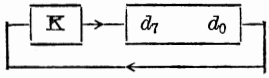
6．九位循环左移



此时 K 被当作操作数的第 9 位，处在 d_7 和 d_0 之间。所有各位均左移一位，K 位进入 d_0 位置，d_7 位进入进位标志。

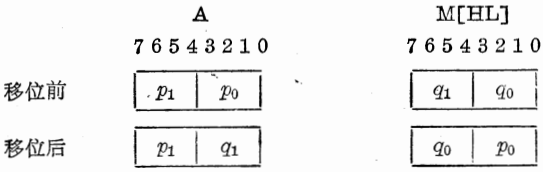
该操作用于单字长量 q 及 K 时写作 qK:@L。

7．九位循环右移



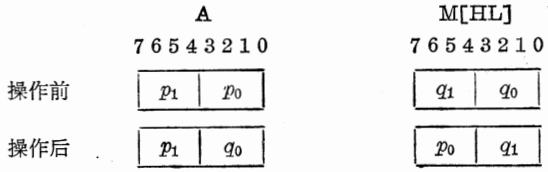
如图所示，该操作恰好与九位循环左移操作相反。用于单字长量 q 及 K 时写作 qK:@R。

8．四位字节左移(只有 Z80 才有)



在此操作(以及"四位循环右移"操作)中，有两个单字长量 p (在寄存器 A 中)和 q(在 HL 的内容规定的存贮单元中)。其作用是 p 的高四位字节(图中的 p_1)保持不动，而其它三个四位字节循环左移四位。该指令被写作 A3M:@4L (可向读者提示 A 的 0～3 位和 M[HL]所有各位都循环左移 4 位)。

9. 四位字节右移(只有 Z80 才有)



如图所示，这个操作是"四位字节左移"的反操作，写作 A3M:@4R。

各机种提供的移位功能如下：

	6800	8080/8085	Z80
1. q:*2	(i)	(ii)	(iv)
2. q:/2	(i)	—	(iv)
3. q:−>	(i)	—	(iv)
4. q:@L	—	(iii)	(iv)
5. q:@R	—	(iii)	(iv)
6. qK:@L	(i)	(iii)	(iv)
7. qK:@R	(i)	(iii)	(iv)
8. A3M:@4L	—	—	(v)
9. A3M:@4R	—	—	(v)

(i) q 可在任何累加器(A 或 B)或以直接(JK)或变址(IX)寻址方式进行寻址的任何存贮单元中。

(ii) 无此指令，但是 A:+A(单字长加法)或 HL:+HL(双字长加法)可起同样的作用。

(iii) q 必须在累加器 A 中。

(iv) q 可在任何单字长寄存器 (A、B、C、D、E、H、L)中，或在由 HL 间接寻址的存贮单元中，或在由变址(IX、IY)寻址方式进行寻址的存贮单元中。

(v) 操作数只能在 A 和 M[HL]中。

—无此操作。

2.4.9 其它算术和逻辑操作

2.4.9.1 加一和减一

"加一"和"减一"操作(即"加法计数"和"减法计数")执行得很频繁,因此所讨论的四种处理机都有对所有累加器,实际上是对所有的存贮单元(仅受寻址能力限制)执行加一和减一操作的专用指令。

现分别将这些操作写成":+1"和":−1"。例如,使用这些操作的指令可以是:

6800	A:+1(4C)	SP:−1(34)	M1234:−1(7A 12 34)
8080/8085/Z80	B:−1(02)	HL:+1(23)	M:+1(1C, M[HL]:+1)
Z80	M12Y:−1	(FD35 12)	IX:+1(DD23)

8080、8085 和 Z80 对单字长量执行加一和减一操作的指令会影响常用的标志位,但不影响 K 的状态;对双字长寄存器执行这些操作则完全不影响标志值。

在 6800 中,对标志的处理也一样,只是对变址寄存器(IX)执行加一或减一操作时会影响零标志 Z。

[注意:指令 A:+01 (其第二个操作数由两个十六进制数字表示,成为指令的 J 字节)表示一般的加法,执行时会影响进位标志 K。而指令 A:+1 (只有一个数字)对 A 所起的作用与上面那条指令相同,但是不会影响进位标志。这一点同样也适用于A:−01和 A:−1 以及涉及其它单字长或双字长累加器的各对相应的指令。]

2.4.9.2 十进制调整

这种奇怪并有点复杂的操作在这四种处理机中都由一条单字节指令实现。这个操作用来协助完成二-十进制编码(BCD)的算术运算。在二-十进制算术运算中,一位十进制数字由其值在 0000(0)～1001(9)内的四位字节来表示,因此单字长寄存器可存入两个这样的数字。假设有两对这样的数字,每对代表一

个0～99之内的无符号十进制整数。应如何将这两对数字加起来形成二-十进制的和数呢？首先来看一下对给定的两位二-十进制数执行简单(二进制)加法的效果。

若 p 是　　01010011 (53)
　q 是　　01000110 (46)
二进制加法得　　10011001 (99)　(正确)
但是,若 p 是　　00110101 (35)
　q 是　　01000110 (46)
二进制加法得　　01111011 (7B)　(不是所要求的结果)
　结果应是　　10000001 (81)
　又,若 p 是　　00111001 (39)
　q 是　　01011000 (58)
二进制加法得　　10010001 (91)　(不是所要求的结果)
　结果应是　　10010111 (97)
　若 p 是　　10011001 (99)
　q 是　　01100111 (67)
二进制加法得(1) 00000000 (K, 00)　(不是所要求的结果)
　结果应是(1) 01100110 (K, 66)，进位标志表示有100(十进制)的溢出。

在上述这些情况下,都要在二进制加法后立即执行一条"十进制调整"指令,以完成把简单二进制加法的结果变换成所需的二-十进制结果。调整操作依靠专用的条件标志H(半进位)来实现，半进位表示二进制加法是否在累加器的 d_3 和 d_4 之间产生了进位。此外还取决于K—(全)进位标志。因此，取入累加器的二进制数,或除两个二-十进制数的加法以外的其它任何过程产生的二进制数，一般都不能单独由十进制调整操作转换成二-十进制数。

Z80 的“十进制调整”指令也可用在减法(带借位或不带借位均可)或取负操作之后。这样做必须增加一位标志位(称为S),由减法指令置位,而由其它所有的指令复位。而 6800、8080 或 8085 则无此特性。

在执行了有效的十进制调整操作后,进位标志 K 的意义与无符号二进制加法或减法后 K 的意义类似。

若 K=0, 累加器中的结果是正确的。

若加法经调整后 K=1, 真正的结果就比累加器中的结果大 100 (十进制)。

若减法经调整后 K=1(只限于 Z80)真正的结果就比累加器中的结果小 100, 所以是负的。

十进制调整的过程可总结如下。在任何情况下要调整的值都应在累加器 A 中。将 A 中高、低四位字节分别写作 a_1 和 a_0。

二进制加法后所需的调整是:

若 $a_0>9 \vee H$, 则 A:+06; 若 $a_1>9 \vee K$, 则 A:+60。

二进制减法后所需的调整是:

若 H=1, 则 A:−06; 若 K=1, 则 A:−60。

若二进制加法或减法使 K 置位了,则在整个调整过程中将保持置位状态。若在调整之前 K 是 0,则可被某个调整步骤所置位,一经置位就不会再被清除了。

对于不熟悉上述“ALGOL类”表示法的读者来说,上式可翻译成:加法后若 $a_0>9$ 或 H 被置位,则将 06(十六进制)加入 A;若 $a_1>9$ 或 K 被置位,则将 60(十六进制)加入 A;这两个修正步骤都是必须执行的。在减法之后,若 H 被置位,则从 A 中减去 06;若 K 被置位,则从 A 中减去 60;这两个修正步骤也都是必须执行的。

读者可以发现,若以本节开始时所举的加法例子以及类似

的减法例子进行调整步骤的练习，则很有好处。

这条指令在本书中的书写形式是 DEC。

其它的二-十进制操作将在§4.7中讨论。

§2.5 堆栈操作

堆栈的概念非常简单，但对计算机设计和程序设计却有着重要的、深远的影响。使用堆栈很有经验的程序员将会发现，如果没有堆栈则将会受到很大的束缚。这里只能介绍一下堆栈的结构及其实现方法，而在后面几章说明其应用⑦。

对我们来说，堆栈就是一堆数。可将一个新的数放到堆栈顶上，堆栈顶上的数也可被取走。只有当堆栈中某一个数之上的所有的数逐个地被取走之后才能对该数进行存取。这个规则就是"后进-先出"。

单字长数的堆栈可借助于一个寻址寄存器用主存贮器(RAM)来实现，该寻址寄存器将用作"堆栈指示器"。首先把一个数值置入堆栈指示器，从而决定堆栈的基地址。这个基地址一般是一个高地址，堆栈向低地址扩展。因此，把一个数放到堆栈上去的操作("压入")就是：

(i) 将该数送入地址为堆栈指示器之值的存贮单元；

(ii) 使堆栈指示器的值减一。

从堆栈中取出一个数("弹出")并将其送入(例如)累加器中的操作是，

(i) 使堆栈指示器的值加一；

(ii) 将存贮单元(其地址在堆栈指示器寄存器中)的内容送入累加器。

---

⑦：IEEE 计算机汇刊 1977 年 5 月号 (Vol. 10, No. 5)化了很大篇幅介绍堆栈计算机。

因此，若堆栈指示器寄存器为 SP，要用的累加器是 A，且任意规定堆栈的基地址为 7FFF，则：

建立堆栈的操作是：　　SP:＝7FFF

压入操作是：　　M[SP]:＝A; SP:－1

弹出操作是：　　SP:＋1; A:＝M[SP]。

注意，如这样安排的话，堆栈指示器总是指向“高于”栈顶的那个存贮单元——将要存入下一个新元素的空单元⑧。

6800 的堆栈就与这种结构完全相同，用一个 16 位的寄存器 SP 作为堆栈指示器，“压入”和“弹出”指令可对两个累加器 A、B 之一进行操作。

可借助于写作 SP:＝7FFF（十六进制 8E 7F 7F）的指令把规定的数值送入堆栈指示器。

S:＝A（十六进制 36，意思是把 A 的内容送上堆栈）的作用是 M[SP]:＝A; SP:－1。

A:＝S（十六进制 32，意思是“将栈顶的值送入 A”）的作用是 SP:＋1; A:＝M[SP]。

对于累加器 B 则用 S:＝B（十六进制 37）和 B:＝S（十六进制 33）。

在 8080、8085 和 Z80 中，其堆栈的安排则与 6800 的大不相同。主要的差别是：涉及的堆栈操作是双字节量而不是单字节量。这些机器都有对各个双字长寄存器（除程序计数器以外）执行堆栈操作的“压入”和“弹出”指令。这些指令都可使堆栈指示器的值加二或减二。另一个差别是堆栈指示器总是指向实际的栈顶元素，而不是其上的第一个空单元。

---

⑧：有些读者也许熟悉一些其它堆栈结构，很难把弹出一个堆栈元素与使堆栈指示器加一的操作联系起来。那末可以这样来理解：堆栈画在一页纸上，其各行按常规进行编号，最低号码在顶上，而最高号码在底部。

因此就有：

SP:＝17F4（十六进制 31 F4 17）将规定的数值放入堆栈指示器；

ST:＝BC（十六进制 C5）是“压入”指令，相当于 SP:－2；MM[SP]:＝BC；

BC:＝ST（十六进制 C1）是“弹出”指令，相当于 BC:＝MM[SP]；SP:＋2。

类似的指令也可对其它的双字长寄存器进行操作。

这些机器还有与栈顶单元进行交换的交换操作（见§2.3.4）。

指令 HL:＝:ST（十六进制 E3）将 HL 中的值与栈顶的值进行交换，而对堆栈指示器的值没有影响。

所有的机器都有对堆栈指示器进行“加一”和“减一”操作的指令。在所有的情况下都只对堆栈指示器加一或减一。

在进入一个子程序（见§2.7）时，就把一个数值（返回地址）放上堆栈。脱离子程序时就把该值移出堆栈。用堆栈保存返回地址是完全合理的，但是遗憾的是没有专门供这一目的使用的堆栈或堆栈指示器。因此，就无法方便地用堆栈把数值送入子程序或从子程序中取出。若在一个包含有子程序的程序中使用几个堆栈，则必须非常仔细才行。

§2.6 转　　移

在“地址和存贮单元”一节（§2.3.6）中曾以无条件转移指令为例作过介绍。这条指令的作用就是把一个转移地址（新值）送入程序计数器；从而可从该转移地址取出待执行的下一条指令，而不是顺序地取出下一个存贮单元的指令。程序计数器的新值可由该指令本身给出（字节 JK），从而形成直接转移；或者

可放在一个寄存器(间接转移)中;还可将一个带符号数(J 字节)与程序计数器的原值相加而得(相对转移)。

只要满足某种规定条件(这种条件一般是指定的某个标志具有规定的值),条件转移指令也能执行转移操作。要是条件不满足,则条件转移指令就只能使程序计数器执行正常的增量操作,按原顺序执行下一条指令。

条件转移的例子是:

JZ 1234　若 Z=1, 则转至 1234(PC:=1234);否则,不做任何操作(PC:+3)

JNK+12　若 K=0, 向前越过 12 个字节(十进制是 18 个!)(PC:+14);否则不做任何操作(PC:+2)。(参阅§2.3.6)

§2.7 子 程 序

子程序是在程序执行过程中随时都可能执行的一段程序。例如,由于微处理机没有乘法指令,就必须用一个子程序来完成乘法(见§4.5.3)。可以用一段程序计算给定在单字长寄存器中的两个单字长数的乘积,并将其放在一个双字长寄存器中。因此,在需要进行乘法时,主程序可把两个因数分别放入两个单字长寄存器中,然后转至乘法子程序的入口地址。执行完子程序后,必须用一条转移指令返回原来跟在"转子"指令后的那条指令。这条指令将是使用留在双字长寄存器中的乘积的指令串中的第一条指令。

存贮器中只有一份乘法子程序。从主程序的若干地方转至子程序的第一条指令去是没有什么困难的。问题是每次调用子程序后必须正确地返回主程序。微处理机是用堆栈来解决这个问题的。就在转至子程序的第一条指令去之前,程序把程序计数器的"正常"值压入堆栈。该值是在静态顺序中紧跟着转移指

令的那条指令的地址，也即在动态顺序中执行完子程序后，必须通过的指令的地址。所以，子程序末尾必须放一条把返回地址“弹出”堆栈并送入程序计数器(PC:=ST)的指令。

每种微处理机都有转子程序(子程序调用)用的专用指令。例如，存在0123号单元中的指令JS1234的作用是把0126(0123+3)压入堆栈，然后转至1234。

每种微处理机也都有从子程序返回的专用指令。例如，若从1234开始的子程序执行到RET(返回)指令时，就把堆栈中的数值(0126)送入程序计数器(相当于转移至0126)。

编写一段短程序也许更说明问题。若要求计算 $x=a\times b+c\times d+e\times f$ 的值，式中 a、b、c、d、e、f 都是单字长数，x 是双字长结果。所用的机器是8080。有一个乘法子程序，其第一条指令存在1234号单元之中。该子程序将给定在寄存器B和C中的数相乘，其乘积放在HL中。子程序(在动态顺序中)以RET指令结束。整个程序从2000号单元开始。

```
1234: —  ┐
         —
         —
         —  } 乘法子程序(HL:=B×C)
         —
         —
         —  ┘
      RET
      . . .
2000: B:=a
      C:=b
      JS1234          形成 a×b
```

HL:=:DE	将 $a\times b$ 放入 DE
B:=c	
C:=d	
JS1234	形成 $c\times d$
HL:+DE	形成 $a\times b+c\times d$
HL:=:DE	将 $a\times b+c\times d$ 放入 DE
B:=e	
C:=f	
JS1234	形成 $e\times f$
HL:+DE	形成 $a\times b+c\times d+e\times f$
x:=HL	存贮结果。

在这个子程序中，程序员假定该子程序不会影响 DE 的内容。

上述“转子”指令还有一些变相形式。6800 具有采用相对地址或变址地址的子程序调用指令。8080、8085 和 Z80 的子程序调用只能采用直接寻址方式，但是每种处理机都有一组与条件转移对应的条件子程序调用指令以及一组条件返回指令。这些机器还有八个特殊的地址(0000、0008、0010、0018、0020、0028、0030、0038)。若子程序的第一条指令位于这八个地址之一内，则可用一条单字节指令来调用。因此，JSS1 (十六进制 CF，一个字节)就相当于 JS0008(CD 08 00，三个字节)。

出于某种莫明其妙的原因，这些特殊的子程序调用指令在制造厂的手册里被称为“再启动”指令(见§2.9)。

上述简单程序并未解释为什么要用堆栈保存返回地址。在任何复杂一些的程序中，子程序本身可能需要调用其它子程序，这些子程序还可能再调用其它子程序。在任何时候，栈顶的返回地址总是最后调用的子程序的返回地址。栈顶之下的返回地

址将按要求的顺序一个一个地出现。

现在就可以明白为什么需要用第二个堆栈指示器(不受程序员控制)来保存返回地址了。在实际的安排中,很难借助于堆栈与子程序交换数据。由于返回地址放在堆栈中,就使得再将新值压入堆栈指示器变得很危险。

有了第二个堆栈指示器后,返回地址就可存放在它专用的堆栈中,同时程序员可完全自由地按问题所需的性质借助于若干个堆栈之一与子程序交换数据。

§2.8 输入-输出

在6800的指令系统中没有专用的输入或输出指令。这并不是说6800无法使用外围设备,而只能说明外围设备必须作为存贮器的一部分(占用一个或多个存贮单元)而与6800系统相接口。这样,执行把这些地址单元内的值送入处理机寄存器的指令(例如A:=M1234)就可完成输入操作。相反,将数值送入这些单元(例如用M1234:=A之类的指令)就可完成输出操作。

在讨论半导体存贮器时(§1.3.2),强调了PROM和RAM的存取时间与处理机的时钟频率适配的重要性。同样,有关外围设备的第一个问题就是它们都属于低速设备,甚至要进行人工操作(如对键盘进行操作)。第二个问题是系统欲与外界传送的信息的性质,可能需与系统内信息(以8位的字节为基础)的性质相适应。

让我们来考虑用电传打字机键盘作为系统的输入设备的情况。

图2.1是供程序员用的框图。在接口的微处理机一边,执行读指令(例如A:=M1234)时,先要将地址(1234)送到地址总线上,并向接口提供一个时序信号(控制信号),然后就把接口的

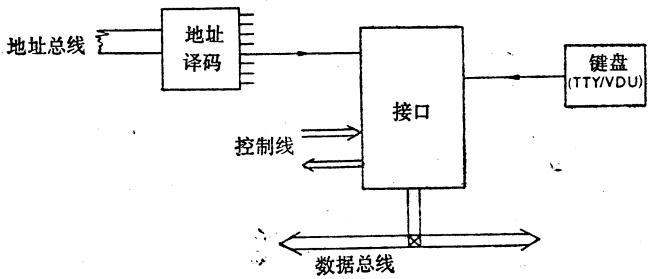


图 2.1 键盘接口

数据总线端上的信息读入。

在键盘一边，每当按下一个键时，接口就从键盘得到一串二进制信息，通常是一个起始位、八个信息位和两个停止位。这些信息是以一定的波特率(每秒 110～330 位)发出的。在两次键盘操作之间则没有其它信息可供接收。

详细讨论接口设计已超出了本书的范围⑨。接口可以有几种实现方法，因此设计出来的程序也不尽相同。

下面分别介绍几种接口实现方法：

(i) 逐位串行

在这种接口方式中，接口的功能是顺序地使由键盘发来的各位信息(包括起始位和停止位)变成数据总线上的各位信息后再提供使用。可用一个子程序来读入字符，每当要读入一个字符时，就调用该子程序。该子程序从一个循环程序开始，该循环程序读入接口数据(例如 A:＝M1234)——送入 A 的 8 位数据中只有一位有意义。然后对该位进行测试，并用条件转移使循环重复执行直至出现起始位时为止。当检测出起始位以后，就不

⑨ 见 D. Zissos, System Design with Microprocessors Academic press, 1978。

再进行循环。经数毫秒(3或9毫秒)之后，才能得到第一个信息位，所以将用一个"延时"循环来消磨时间。然后再读入接口数据，此时第一位信息位就会存入A。出于安全起见，将该位信息放入另一个寄存器，然后重复执行"延时"循环并读取第二个信息位。将该位放在第一位信息的旁边。这个过程一直进行到8个信息位在第二个寄存器中装配成一个字节为止。此后，还必须执行"延时"循环，以便接收停止位。然后子程序才能结束，并能立即或随时再次调用该子程序以处理其它的输入字符。

这个子程序不太容易编写或测试。但是一旦写成后，就可永久地存贮在只读存贮器中，此后读取字符操作就只是调用该子程序罢了。

莫托洛拉公司设计的MEK 6800套件中就有这种类型的接口。该接口包括一个存在ROM中的子程序，占用65个字节，每次读一个字符。

(ii) 串行至并行

在这种接口方式中，接口设计得能接收并保持键盘所发送的一个字符的所有各位，并在接口的数据端上将8个信息位提供给系统。

只有当8位信息都可用时，处理机才能执行"读"指令。所以，"读字符"子程序从一个循环开始，该循环重复执行到一位控制信息("数据准备好")变零为止。数据准备好信号出现在接口的第二个地址上。

这类接口用的子程序比(i)中所介绍的子程序简单得多，相当多的工作要由接口的硬件来完成。英特尔公司设计了一种具有这种规格的接口，配合其8251通用同步/异步收发器使用。该接口所需的子程序只有8个字节。与此类似，莫托洛拉公司采用6850异步通信接口转接器(ACIA)所设计的接口，也能完

成同样的功能。这是微处理机系统接收和发送串行数据最常用的办法。

(iii) "等待"接口

在这种接口方式中，接口硬件承担了所有的工作。程序中只需用一条读数据指令(例如 A:=M1234)读取字符，而不需要子程序。(ii) 中的"数据准备好"位并不出现在数据总线上供处理机读入，而是作为一个电路的输入信号。只要该信号为零，这个电路就使处理机处于"暂停状态"。一旦"数据准备好"信号变成 1，就允许处理机执行读指令。执行读指令后，"数据准备好"信号立即复位，以防同一个字符被读入两次。

8080、8085、Z80 和 6800 都可采用这些类型的接口，所有的机种都可以同样的方式使用地址。(ii)和(iii)型接口也可以设计得由双字长读取指令一次读入 16 位(两个连续的字节)数据。还可设计出与此类似但以相反的方式工作的接口，供电传打字机和目视显示部件之类的外围设备使用。

8080、8085 和 Z80 还有一个特性：除了 16 位地址总线提供的 65536 个地址外，还有 256 个特殊地址("外围设备编号")。这些特殊地址是专供执行处理机的输入和输出指令时用的。每个特殊地址可由一个输入-输出转接口占用。写作 A:=G23 (十六进制 DB 23)的指令(意思是"把 #23 转接口上出现的数值送入 A")的执行过程如下。把特殊地址放到地址总线上，但是处理机发出的不是进行"读存贮器"操作的时序脉冲，而是进行"读输入设备"操作的时序脉冲。因此不会影响任何存贮单元，但是允许从输入转接口来的输入字节进入寄存器 A。

相反的操作可把一个字节输出至一个输出转接口(例如#24 转接口)，写作 G24:=A(十六进制 D3 24)。

Z80 还有其它一些指令，可间接地通过一个单字长寄存器

(寄存器 C)规定特殊地址,而数据寄存器可以是任何寄存器(包括寄存器 A)。Z80 的"单步"和"循环"操作(见§2.10)可直接在输入-输出转接口与一个存贮单元或一组存贮单元之间传送数据。每次传送时转接口和存贮单元都是间接寻址的。

§2.9 中　　断

假定存贮器中有两个程序(A 和 B)。处理机正在执行程序 A，而不执行程序 B。突然，出于微处理机系统之外的某种原因，有必要尽快执行程序 B。但是程序 A 已经做了一半的工作当然也不能丢掉。为解决这个问题就要"中断"程序 A 的执行，转而执行程序 B，然后再从断点开始继续执行程序 A。

由于处理机是由时钟驱动的时序电路，执行任何程序都必须与时钟同步,时钟本身不受中断的影响。但是,处理机是通过重复状态数不同的指令周期而工作的。若允许在任何时钟脉冲内进行中断,即允许在指令周期的任何阶段进行中断,则安排继续执行已部分执行过的指令(任何指令，在任何阶段被中断)就是一个极端困难的问题。显然,若使程序 B 等待到现行执行周期完成以后再由处理机执行,则问题就简单得多了。

基本的中断过程可概括如下:

(i) 向处理机发一个信号,表示需要中断正在执行的程序。

(ii) 处理机执行完一条现行指令。

(iii) 处理机保存好程序计数器的内容(可将返回地址压入堆栈)。

(iv) 处理机(从数据总线)接收信息(程序 B 的起始地址),使处理机能开始执行中断程序 B。

(v) 然后正常地执行程序 B。若 B 以"返回"指令结束,则被中断的程序立即从其断点开始继续执行。

这个过程类似于以 A 作为主程序，B 作为子程序时的子程序调用过程——只是调用 B 是由一个外部信号，而非 A 中的指令引起的。若欲不损害 A 的执行过程，则中断后使 A 能继续正确执行下去所需的全部信息（寄存器内容、状态标志）在执行 B 时必须予以保存，并在继续执行 A 之前加以恢复。

在某些情况下，可能不希望中断一个程序（或某一段程序）。例如，若上一节(i)中介绍的"逐位串行"子程序在执行时被中断了，则键盘发送出来的字符中的若干位就会得不到该子程序的处理。继续工作时子程序会试图读入这后几位信息，但为时过晚了。因此，必须修改中断过程，以便考虑到这种可能性。若处理机中有一个触发器(I)可由程序中的指令置位或复位，则可用该触发器来控制允许还是推迟中断，其过程如下：

(i) 由一个外部信号请求中断目前执行的程序。

(ii) 若 I=1，则程序可被中断；完成现行指令周期并采取上述 (iii)、(iv) 和(v)中的步骤。但是，若接收到中断请求时 I=0，则必须拒绝这一请求，只有当 I 被程序 A 置成 1 时才允许发生中断。

8080 就采用上述那种中断方式，它有一个"中断标志"(I)。该标志可被置位(I:=1，十六进制 FB)，表示"开中断"(允许中断发生)；或被复位(I:=0，十六进制 F3)，表示"关中断"(禁止中断发生)。这个标志的状态由处理机的输出信号 INTE ("中断开放")给出。处理机接收中断请求信号 INT，若 I=1，则在指令周期结束后由处理机给出 INTA("中断响应")信号。出现这个信号就迫使一条指令(一般是子程序调用指令——一或三个字节)被送到数据总线上。再把这条指令取到指令寄存器内，开始执行中断程序。

指令 I:=0 和 I:=1 是 8080 指令系统中仅有的与中断有

关的指令，但还可参阅下文有关“停机”(HALT)一节的内容。程序员必须确保：中断处理程序要把被中断的程序继续正确执行所需的全部信息保存起来，而在最后再加以恢复。

如上文(iii)所建议的(实际上微处理机也是这样做的)，若用堆栈保存返回地址，则原则上中断程序本身还可被中断(就象子程序本身可调用子程序那样)。这就提出了这样一种可能性——中断请求可来自一个以上的中断源，且当 I=0 时可能有若干个这样的中断等待着处理。当再次允许中断时(I:=1)，应以什么顺序处理这些中断请求呢？是以到达的迟早(排队)，还是按照某种相对优先级的概念进行处理？中断处理方案必须由硬件实现，这些硬件将接受若干个中断源来的中断请求，产生送至处理机的“INT”信号，并把有关的指令放到数据总线上，从而允许第一个中断起作用。

被中断的程序 A 与待执行的程序 B 之间的关系可以有几种：

(i) 这两个程序可相互独立，用存贮器的不同部分作为暂存区，尽管它们很可能要调用同一个子程序。

(ii) B 可对 A 获得的结果进行处理。例如，B 的目的可能是根据操作员的请求打印和报告 A 的进程。

(iii) A 可对 B 获得的结果进行处理。例如，B 的目的是尽可能快地向 A 提供 A 所需的新信息。若该信息来自一个数据源(也许是连接监控某个化学过程的仪器的外围设备)，且只在很短的时间内有效，则中断请求就很急迫。如果不足够快地允许这个请求的话，就会错过机会。

在第(i)种情况下，编写程序 A 时可不必考虑中断问题。程序 B 则必须编写得在其执行结束时保持处理机和任何与 A 公用的存贮单元的状态与其开始执行前一样。在(ii)、(iii)两种

情况下，编写程序 A 时必须考虑到 B 的活动。例如在(ii)的情况下，若 A 连续计算若干组结果，就可能希望只有当能提供一组完整的新结果时才允许 B 报告进程，而在计算一组结果的过程中，则不允许 B 报告。这时可使用标志 I 来进行管理。在第(iii)种情况下，可能在有些时间内 A 能接受新的信息，而在另一些时间内不能接受新的信息。这时仍可用标志 I 加以控制。但是由于有紧急请求的可能性，所以在逻辑上实现起来可能有些困难。

当所有的可能性相互结合——中断又被中断、有不同紧迫程度的请求和有人工控制的程序时，程序设计就变得非常困难。有关这个问题的全面讨论已超出了基础课本的范围。

在 6800、8085 和 Z80 的指令系统中，有关中断的安排比 8080 巧妙得多。

在 6800 中，有一个条件标志 I，但是其意义与 8080 中的 I 正相反。在 6800 中，I=0 意味着中断请求将被接受；I=1 意味着普通的请求不被接受。普通的请求是指“IRQ”(“中断请求”)端接收的请求。另一个输入端——“NMI”(“不可屏蔽中断”)端接收的请求无论 I 的状态如何都将在当前指令周期结束时被接受，而且优先级比 IRQ 端的请求高。这既给程序设计带来了好处，也带来了困难。

在接受一个中断时，6800 处理机的动作比 8080 的动作复杂得多。所有寄存器的内容(累加器 A 和 B、条件标志寄存器 C 和变址寄存器 IX)都与返回地址一起存入堆栈。相应地，有一条“由中断返回指令”(RI，十六进制 3B)可用于中断的结尾处，该指令会在转回至堆栈内的返回地址之前，将存入堆栈的值送入寄存器。

中断过程(将寄存器内容存入堆栈后)以间接转移转至

MMFFF8(IRQ)或MMFFFC(NMI)单元中的地址作为结束(参阅上述第(iv)步)。因此,必须使MMFFF8或MMFFFC能给出中断例行程序的第一条实际指令地址。

有的程序(例如第(iii)种情况中的A)可能要到程序B中断了它,并供给它新信息之后才能继续执行下去。6800的设计者预计到会有这种困难,因此,用了一条"等待中断"指令WI(十六进制3E)。这条指令能有效地使程序处于暂停状态,直至发生中断并完成中断处理为止。简而言之,其作用如下:首先,该指令将其(静态顺序中的)下一条指令的地址存入堆栈作为返回地址,并将寄存器的内容存入堆栈。然后处理机处于等待状态,直至接收到预期的请求(IRQ)为止。一当接收到这一请求,处理机转移至MMFFF8中的地址去,从而结束WI的执行。

8080中出现类似的情况时将使用停机(HALT,十六进制76)指令。执行这条指令就将使机器进入暂停状态,直至接收到一个输入信号(RESET、INT、HOLD)为止。若机器停机时接收到一个INT信号,就允许中断。在中断完成后,被中断的程序将从(静态顺序)HALT指令后的那条指令开始继续执行。

6800还有一个特点,即由SI(十六进制3F)引起的"软件中断"。这并不是普通所说的中断,而把它看作是一种特殊形式的子程序调用更为确切。执行SI指令就将返回地址和寄存器内容存入堆栈,然后转移到存在MMFFFA这一对单元之内的地址去。用这种转移方式调用的子程序不应该用普通的返回指令RET(39)而应该用RI(3B)指令结束,因为必须把寄存器的内容从堆栈中取出来。

Z80有两个中断请求输入端,一个是普通中断请求用的INT输入端,另一个是紧急请求用的NMI输入端(与6800中的一样)。Z80对NMI端的请求作出的响应是完成现行指令,然

后不管 I 的状态如何都按通常的方式将返回地址存入堆栈并转至 0066 号单元(总是这个单元)。Z80 对 INT 端的请求作出的响应取决于其当时的"中断模式"。中断模式可以是 0、1 或 2。在复位(RESET)时该模式被置成 0,除此以外其它任何时间都可根据需要用 IM:=0 (ED 46)、IM:=1 (ED 56)或 IM:=2 (ED 5E)等指令之一来设置中断模式。

在 0 模式时,处理机对 INT 端的请求作出的响应与 8080 的相同。即若 I=1,就将返回地址存入堆栈,并执行由中断设备放到数据总线上的(任何)指令。

在 1 模式时若 I=1,则将返回地址存入堆栈,并转至地址 0038。

在 2 模式时若 I=1,则将返回地址存入堆栈,并转至存在 MM[Q // p]单元中的地址去。这里 p 是由中断设备放入数据总线的一个数值为偶数的字节;Q 是一个专用的单字长寄存器,该寄存器的内容可从寄存器 A 传送过去(见第三章 Z80 的指令系统)。Q // p 表示一个 16 位的值,其高字节是 Q 中的值,低字节是 p。

若 I=0,则对 INT 端的请求不予理睬,其停机(HALT)指令(76)与 8080 的一样。

8085 的中断

8085 有五个中断请求端,即 INTR、RST5.5、RST6.5、RST7.5 和 TRAP。每个指令周期内都要检查这五个输入端,检查的顺序是 TRAP、RST7.5、RST6.5、RST5.5、INTR。无论 I 的状态如何,TRAP 端的请求都会被接受,其作用是将返回地址存入堆栈并转至 0024 号单元。RST 端的请求可通过设置"中断屏蔽"(IM)字的对应各位(见第三章)而加以屏蔽(即关闭)。只有当 I=1 且屏蔽位为 0 时才接受 RST 请求。其作用

是将返回地址存入堆栈并转至003C(RST7.5)、0034(RST6.5)或002C(RST5.5)号单元。对INTR端的请求的处理过程完全与8080的一样。当接受一个中断请求时 I:=0。

§2.10 Z80的其它指令

辅助寄存器

Z80除了寄存器A、B、C、D、E、F、H、L外，还有一套辅助寄存器A′、B′、C′、D′、E′、F′、H′、L′。有两条交换指令(见§2.3.4)与这些辅助寄存器有关：

AF:=:XX (十六进制08)的作用是：

A:=:A′; F:=:F′

BL:=:XX (十六进制D9)的作用是：

B:=:B′; C:=:C′; D:=:D′;

E:=:E′; H:=:H′; L:=:L′。

其它指令都与辅助寄存器无关。所以，辅助寄存器的基本用途是，在执行需使用普通寄存器的子程序、中断或其它指令串时，用来安全可靠地保存普通寄存器的内容。

位操作

§2.3.5介绍了个别位的表示法，即用 $q.i$ 表示寄存器或存贮单元 q 的第 i 位。在下文中，q 可以是A、B、C、D、E、H、L或M(代表M[HL])，或 $M_{j_1j_0}X$ 或 $M_{j_1j_0}Y$ (采用变址地址的存贮单元，见§2.3.6)之一。

Z:=#$q.i$ 若 $q.i=0$ 则将零标志置成1。

$q.i=1$ 则将零标志置成0。

$q.i$:=0 将 $q.i$ 置成0，不影响 q 的其余各位。

$q.i$:=1 将 $q.i$ 置成1，不影响 q 的其余各位。

计数转移

在 Z80 中，“B:－1; JNZ+j_1j_0”(三个字节)这对指令可由一条只用两个字节而作用相同的指令来代替。这条指令最主要的用途是对指令循环进行循环次数计数，所以转移可以是向后的(偏移量为负)。

这里将该指令写作

B:－1, JNZ+F8

式中的逗号表示这是一条指令而不是两条，F8 是 J 字节的值(也可以是其它值)。

这类指令只能涉及寄存器 B 和条件标志 NZ，没有对其它寄存器和其它标志进行操作的类似指令。

单步和字组操作

Z80 有十六条对连续的存贮单元进行操作的指令，按下面介绍的那样以四条为一组。其中每条指令都有两个字节的操作码(I_0、I_1)，而无 J 或 K 字节。

(i) 复制(Copy)

指令 ED B0 (十六进制)的作用如下：

W:M[DE]:=M[HL];
DE:+1;
HL:+1;
BC:－1;
if BC≠0000 **then go to** W;

因此，若 BC 中的原始值是 n($1\leqslant n\leqslant 65535$，十进制)，则将把 n 个字节从连续的一“块”单元复制到另一块去。注意，若 BC 中的原始值为零，则该循环将执行 65536 遍，而不是零遍!

指令 ED B8 (十六进制)的作用与此相同，只是循环时 DE 和 HL 中的值每次减一。

指令 ED A0 (十六进制)的作用是执行 ED B0 的一步，即：

M[DE]:=M[HL];

DE:+1;

HL:+1;

BC:−1。

指令 ED A8 (十六进制)的作用是执行 ED B8 的一步。

(ii) 搜索 (Search)

指令 ED B1 的作用是:

W:A−M[HL];

HL:+1;

BC:−1;

if BC≠0000∧Z **then go to** W

在执行这条指令后，若标志 Z=0，且 BC=0000；说明检查的 n 个单元内没有与 A 的内容相同的数值。

但是，若执行后 Z=1，则表示发现了内容与 A 中内容相同的存贮单元。该单元的地址为 HL 中的值减一。

指令 ED B9 (十六进制)的作用与此相同，只是 HL 中的值每步减一，因此是"向后"搜索地通过一块存贮单元。

指令 ED A1 (十六进制)的作用是执行 ED B1 的一步:

A−M[HL];

HL:+1;

BC:−1。

执行这条指令后，标志 N、Z 的设置状态与执行了减法操作一样。且若 BC≠0 则标志 P (一般的溢出标志)被置 1；若 BC=0 则被置 0。

指令 ED A9 的作用是执行 ED B9 的一步。

(iii) 输入 (Input)

指令 ED B2 (十六进制)的作用是:

W:M[HL]:=G[C];

HL:+1;　　B:−1;

if B≠0 **then go to** W。

若 B 中的原始值是 $n(1\leqslant n\leqslant 255)$，则将从输入转接口 C (即设备号存在 C 中的转接口)读入 n 个字节，并送入存贮器中连续的单元内。若 B 中的原始值是 00，则将读取 256 个字节。

就象对待其它输入、输出指令一样，使用这条指令时也必须十分小心。处理机在 4MHz 时钟频率下全速运行时，两次连续的读操作之间的时间间隔将为$5\frac{1}{4}$微秒。电传打字机只能每隔 100000 或 33333 微秒提供一个字符，每秒 1000 个字符的纸带输入机也只能每 1000 微秒提供一个字符。除非转接口 C 上的外围设备能每隔 $5\frac{1}{4}$ 微秒产生一个新的有效数值，否则若使用这条指令的话就必须采用 §2.8 所述的"等待"接口，或者利用中断技术(如下所述)。

指令 ED BA 的作用与此相同，只是 HL 中的值每步减一。

指令 ED A2 的作用是执行 ED B2 的一步：

M[HL]:=G[C];　HL:+1;　B:−1;

而指令 ED AA 是执行 ED BA 的一步。

(iv) 输出(Output)

指令 ED B3 (十六进制)的作用是：

W:G[C]:=M[HL];　HL:+1;　B:−1;

if B≠0 **then go to** W。

这条指令的作用与 ED B2 正相反。

ED BB (十六进制)的作用与此相同，只是 HL 中的值每步减一。

指令 ED A3 的作用是执行 ED B3 的一步：

G[C]:=M[HL];　HL:+1;　B:−1;

而 ED AB 的作用是执行 ED BB 的一步。

与"输入"指令一样，使用这条指令时也必须十分小心。

中断

八条字组处理指令 (ED B0、ED B1、ED B2、ED B3、ED B8、ED B9、ED BA、ED BB)有一点是特殊的，即可在其执行过程中的任何一步之后接受中断。在执行这些指令期间发出的中断请求都会在完成当前这一步之后立即被接受。从中断返回后将继续执行被中断的指令。字组输入与字组输出指令的中断问题需特别仔细地进行考虑。

附注

这些指令的简短具体的表达形式(与前面的表示法一致)列表于下。但是，用这种表达形式作为程序设计语言的指令则太长了。为此，可使用更方便的由三个字母组成的缩写，各个字母的含意是：C——复制；S——检索；I——输入；O——输出；S——单步；B——字组；U——加计数；D——减计数。这组缩写在右边一栏给出。

表 2.2

机器码	作用	缩写
ED A0	M[DE:+1]:=M[HL:+1],BC:−1	CSU
ED A1	A−M[HL:+1],BC:−1	SSU
ED A2	M[HL:+1]:=G[C],B:−1	ISU
ED A3	G[C]:=M[HL:+1],B:−1	OSU
ED A8	M[DE:−1]:=M[HL:−1],BC:−1	CSD
ED A9	A−M[HL:−1],BC:−1	SSD
ED AA	M[HL:−1]:=G[C],B:−1	ISD
ED AB	G[C]:=M[HL:−1],B:−1	OSD
ED B0	repeat M[DE:+1]:=M[HL:+1],BC:−1 until BC=0000	CBU
ED B1	repeat A−M[HL:+1],BC:−1 until BC=0000∨Z	SBU
ED B2	repeat M[HL:+1]:=G[C],B:−1 until Z	IBU
ED B3	repeat G[C]:=M[HL:+1],B:−1 until Z	OBU
ED B8	repeat M[DE:−1]:=M[HL:−1],BC:−1 until BC=0000	CBD
ED B9	repeat A−M[HL:−1],BC:−1 until BC=0000∨Z	SBD
ED BA	repeat M[HL:−1]:=G[C],B:−1 until Z	IBD
ED BB	repeat G[C]:=M[HL:−1],BC:−1 until Z	OBD

第三章 指令系统

§3.1 说 明

本章将以相同的形式和相同的表示方法完整地列出 6800、8080、8085 及 Z 80 这四种微处理机的指令系统表格。

正如读者将看到的那样，每种机型的说明都从有关寄存器与标志的表格开始。若寄存器是由标志或其它寄存器构成的，则将注明。符号"‖"表示连接或并列；例如 HL≡H‖L 表示 HL 就是 H 与 L 相连，连接方法是 H_0 位紧接在 L_7 位的左边。

本章还将定义各种专用符号。然后是一张详尽的表格，列出了每条指令的二进制指令码以及本书软件部分所采用的书写形式，并对其作用作了说明。作用一般分三个部分：(i)对寄存器、存贮器、转接口以及独立的标志的影响；(ii)对标志寄存器中的标志的影响；(iii)使程序计数器增量。接下来的一栏是对第二章中有关的说明材料的索引。表格的各行是按照最右边一栏给出的分类法排列的。

在指令的二进制码一栏中有许多位由 m、r′、op 或 cond 之类的符号占据着，每组符号可以有 1～4 位。对每种符号都有一张辅助表格(代换表)，列出了可能在书写形式或说明中代替这种符号的各种组合情况。实际使用指令时必须用其中的一种组合取代该符号。例如，在 6800 的表格中有：

二进制指令码	十六进制码	书写形式	操作说明
76543210			
0011001r	3–	r:=S SP:+1; r:=M[SP]	

而辅助表格(代换表)为：

r	书写形式	说明
0	A	A
1	B	B

所以，总表中每一行代表两行：

00110010	32	A:=S	SP:+1; A:=M[SP]
00110011	33	B:=S	SP:+1; B:=M[SP]

这两条指令是“弹出”或“出栈”操作(见§2.5)。

在所有的表格中，j_1 与 j_0 分别表示指令中J字节的高半与低半(十六进制数字)，k_1 与 k_0 表示K字节的高半与低半。在二进制指令码一栏中将用四位二进制码代替这些符号，在其余各栏中都将用相应的十六进制数字来代表这些符号(见§2.3.1、§2.3.5)。

在空间窄小的表格中，需作出注解之处将注明，然后将注解列于表后。在标志栏中，带撇号的量(例如 A'_7)表示标志取结果中规定位的值；没有撇号的量表示应取指令执行前的值。注解之后是一张表示该机指令系统的卡诺图。这些图的格局如表3.1所示。每个方块内的值与相邻方块内的值只有一位不同，而且与关于图中各条轴与其对称的各方块内的值也只相差一位。为便于说明，读者可看00方块，与该方块只差一位的八个值是01、02、04、08、10、20、40和80；与AF方块只差一位的一组方块是AE、AD、AB、A7、BF、8F、EF和2F。这样的图将各组有关的指令集中了起来，非常清楚地说明了整个指令系统的结构。在实际编写程序时用这种图作为简明参照表是非常有用的。

卡诺图之后是一张按操作码(I)的数值顺序排列的指令表，

表 3.1 八位操作码用的卡诺图的格局

		$I_6=0$ $I_2I_1I_0$								$I_6=1$ $I_2I_1I_0$							
	$I_5I_4I_3$	000	010	110	100	101	111	011	001	001	011	111	101	100	110	010	000
$I_7=0$	000	00	02	06	04	05	07	03	01	41	43	47	45	44	46	42	40
	010	10	12	16	14	15	17	13	11	51	53	57	55	54	56	52	50
	110	30	32	36	34	35	37	33	31	71	73	77	75	74	76	72	70
	100	20	22	26	24	25	27	23	21	61	63	67	65	64	66	62	60
	101	28	2A	2E	2C	2D	2F	2B	29	69	6B	6F	6D	6C	6E	6A	68
	111	38	3A	3E	3C	3D	3F	3B	39	79	7B	7F	7D	7C	7E	7A	78
	011	18	1A	1E	1C	1D	1F	1B	19	59	5B	5F	5D	5C	5E	5A	58
	001	08	0A	0E	0C	0D	0F	0B	09	49	4B	4F	4D	4C	4E	4A	48
$I_7=1$	001	88	8A	8E	8C	8D	8F	8B	89	C9	CB	CF	CD	CC	CE	CA	C8
	011	98	9A	9E	9C	9D	9F	9B	99	D9	DB	DF	DD	DC	DE	DA	D8
	111	B8	BA	BE	BC	BD	BF	BB	B9	F9	FB	FF	FD	FC	FE	FA	F8
	101	A8	AA	AE	AC	AD	AF	AB	A9	E9	EB	EF	ED	EC	EE	EA	E8
	100	A0	A2	A6	A4	A5	A7	A3	A1	E1	E3	E7	E5	E4	E6	E2	E0
	110	B0	B2	B6	B4	B5	B7	B3	B1	F1	F3	F7	F5	F4	F6	F2	F0
	010	90	92	96	94	95	97	93	91	D1	D3	D7	D5	D4	D6	D2	D0
	000	80	82	86	84	85	87	83	81	C1	C3	C7	C5	C4	C6	C2	C0

并给出了本书所用的书写形式。书写形式中出现的"12"是表示典型的单字节值，意味着该指令还有一个 J 字节。显然，可用 00～FF 之间的任何十六进制值取代"12"。同样，"1234"表示典型的双字节值，意味着指令还有 J 字节与 K 字节；该值可用任何有四位十六进制数字的量来代替。在程序设计时用这张表作为参照表也是很有用的。

Z80 扩充的指令系统必须用比其它机种的总指令表更充实的表格来表示。而且 Z80 的卡诺图有五个部分，一个通用部分(与 8080 的卡诺图相同，只是空格中填入了内容) 以及 I_0 值为

CB、DD、ED、FD 时的四个部分。

最后有一张包括这四种处理机的所有指令，按字母顺序排列的指令汇总表。

读者起先很可能会认为这些表格太多了，但是随着时间的推移这些表格会变成一组很容易查阅的参考资料。无论哪种处理机，主要的说明文件都是总指令表；卡诺图与按数值排列的指令表仅仅是形式上的概括，每种表格都将为某些读者的某些应用提供方便。指令汇总表以另一种形式进行了概括，这种形式也可能很有用，而指令的执行时间在其它表格中是不容易排进去的。随着读者的经验与兴趣的增长，这些参考材料的实际使用情况也会有所变化。

§3.2 Motorola 6800

6800 微处理机

字长：8 位　　地址：16 位

单字长寄存器：A、B、C

双字长寄存器：IX、SP、PC

条件标志：　H≡C.5(条件码寄存器第 5 位，"半进位")

I≡C.4(中断禁止)

N≡C.3(负)

Z≡C.2(零)

V≡C.1(溢出)

K≡C.0(进位)

(注：C.7≡1，C.6≡1，都是常数)

堆栈指示器：SP

变址寄存器：IX

程序计数器：PC

累加器：　　A，B

双字长寻址：　　MM[x]≡M[x]‖M[$x+1$]

复位信号的作用：　　I:=1；PC:=MMFFFE

保留地址：　　FFF8、FFF9：中断请求（见注4及下文）

FFFA、FFFB：软件中断（见注5）

FFFC、FFFD：不可屏蔽中断（见下文）

FFFE、FFFF：复位（见上文）

对中断请求信号的响应：

(i) IRQ：完成现行指令后，若I=0，则：

MM[SP−1]:=PC+1；

MM[SP−3]:=IX；

M[SP−4]:=A；

M[SP−5]:=B；

M[SP−6]:=C；

SP:−7；

I:=1；

PC:=MMFFF8

(ii) NMI：执行完现行指令后：

MM[SP−1]:=PC+1；

MM[SP−3]:=IX；

M[SP−4]:=A；

M[SP−5]:=B；

M[SP−6]:=C；

SP:−7；

I:=1；

PC:=MMFFFC

表3.2 6800指令系统——总表

I 76543210	十六进制码	书写形式	操作说明	标志(注1) H	I	N	Z	V	K	PC:+	索引	分类
00000001	01	NULL		·	·	·	·	·	·	1		空操作
1rα0110	-6	r:=α	r:=α	·	·	α_7	α	0	·	1,2,3	2.3	赋值 单字长 寄存器
1rα'0111	-7	α':=r	α':=r	·	·	r_7	r	0	·	1,2,3	2.3	
00010110	16	B:=A	B:=A	·	·	A_7	A	0	·	1	2.3	
00010111	17	A:=B	A:=B	·	·	B_7	B	0	·	1	2.3	
01β1111	-F	β:=0	β:=0	·	·	0	1	0	0	1,2,3	2.3	
00000111	07	A:=C	A:=C	·	·	·	·	·	·	1	2.3	
00000110	06	C:=A	C:=A	A_5	A_4	A_3	A_2	A_1	A_0	1	2.3	
0011001r	3-	r:=S	SP:+1; r:=M[SP]	·	·	·	·	·	·	1	2.5	堆栈
0011011r	3-	S:=r	M[SP]:=r; SP:-1	·	·	·	·	·	·	1	2.5	
1ργ1110	-E	ρ:=γ	ρ:=γ	·	·	γ_{15}	γ	0	·	1,2,3	2.3,2.5	双字长
1ργ'1111	-F	γ':=ρ	γ':=ρ	·	·	ρ_{15}	ρ	0	·	1,2,3	2.3,2.5	
00110000	30	IX:=SP+1	IX:=SP+1	·	·	·	·	·	·	1	2.3,2.5	
00110101	35	SP:=IX-1	SP:=IX-1	·	·	·	·	·	·	1	2.3,2.5	
0000101b	0-	V:=b		·	·	·	·	b	·	1	2.3.7	标志
0000110b	0-	K:=b		·	·	·	·	·	b	1	2.3.7	
0000111b	0-	I:=b		·	b	·	·	·	·	1	2.9	
01β1100	-C	β:+1	β:+1	·	·	β'	β'	β'	·	1,2,3	2.4.9	加一/减一 单字长
01β1010	-A	β:-1	β:-1	·	·	β'	β'	β'	·	1,2,3	2.4.9	
00001000	08	IX:+1	IX:+1	·	·	·	IX'	·	·	1	2.4.9	双字长
00001001	09	IX:-1	IX:-1	·	·	·	IX'	·	·	1	2.4.9	
00110001	31	SP:+1	SP:+1	·	·	·	·	·	·	1	2.4.9	
00110100	34	SP:-1	SP:-1	·	·	·	·	·	·	1	2.4.9	
01β0011	-3	β:≠	β:=¬β	·	·	β'_7	β'	0	1	1,2,3	2.4.4	算术 逻辑 一元操作
01β0000	-0	β:-	β:=-β	·	·	β'_7	β'	⊙	⊛	1,2,3	2.4.4	
1rα *op*	—	*ropα*	*ropα*	⊛	·	r'_7	r'	r'	r'	1,2,3	2.4	
00010001	11	A-B	A-B	·	·	√	√	√	√	1	2.4.7	二进制
00011011	1B	A:+B	A:+B	A	·	A'_7	A'	A'	A'	1	2.4.3.1	
00010000	10	A:-B	A:-B	·	·	A'_7	A'	A'	A'	1	2.4.5.1	
00011001	19	DEC	(注2)	·	·	A'_7	A'	A'	⊙	1	2.4.9	特殊
01β1101	-D	β		·	·	β_7	β	0	0	1,2,3	2.4.7	
10γ1100	-C	IX-γ	(注7)	·	·	⊙	√	⊙	·	1,2,3	2.4.7	
01β0100	-4	β:->	(注6)	·	·	0	β'	⊙	β_0	1,2,3	2.4.8	移位和环移
01β1000	-8	β:•2		·	·	β_6	β'	⊙	β_7	1,2,3	2.4.8	
01β0111	-7	β:/2		·	·	·	β'	⊙	β_0	1,2,3	2.4.8	
01β1001	-9	βK:@L		·	·	β_6	β'	⊙	β_7	1,2,3	2.4.8	
01β0110	-6	βK:@R		·	·	K	β'	⊙	β_0	1,2,3	2.4.8	
01101110	6E	Jj_1j_0X	PC:=IX+J	·	·	·	·	·	·	–	2.6	控制传递 转移 调用返回特殊
01111110	7E	$Jj_1j_0k_1k_0$	PC:=JK	·	·	·	·	·	·	–	2.6	
0010*cond*	2-	*Jcond*+j_1j_0	(注8)	·	·	·	·	·	·	–	2.6	
10001101	8D	JS+j_1j_0	(注9)	·	·	·	·	·	·	–	2.7	
10111101	BD	JS$j_1j_0k_1k_0$	(注10)	·	·	·	·	·	·	–	2.7	
10101101	AD	JSj_1j_0X	(注11)	·	·	·	·	·	·	–	2.7	
00111001	39	RET	(注12)	·	·	·	·	·	·	–	2.7	
00111011	3B	RI	(注3)	·	(注3)					–	2.9	
00111110	3E	WI	(注4)	·	⊛	·	·		·	–	2.9	
00111111	3F	SI	(注5)	·	⊛	·	·		·	–	2.7	

6800 代换表

表 3.3a

r	书写形式	说明
0	A	A
1	B	B

ρ	书写形式	说明
0	SP	SP
1	IX	IX

b	书写形式	说明
0	0	0
1	1	1

α	书写形式	说明
00	j_1j_0	J
01	MZj_1j_0	M[0‖J]
10	Mj_1j_0X	M[IX+J]
11	$Mj_1j_0k_1k_0$	M[JK]

α′	书写形式	说明
00		(未定)
01	MZj_1j_0	M[0‖J]
10	Mj_1j_0X	M[IX+J]
11	$Mj_1j_0k_1k_0$	M[JK]

β	书写形式	说明
00	A	A
01	B	B
10	Mj_1j_0X	M[IX+J]
11	$Mj_1j_0k_1k_0$	M[JK]

γ	书写形式	说明
00	$j_1j_0k_1k_0$	JK
01	$MMZj_1j_0$	MM[0‖J]
10	MMj_1j_0X	MM[IX+J]
11	$MMj_1j_0k_1k_0$	MM[JK]

γ′	书写形式	说明
00		(未定)
01	$MMZj_1j_0$	MM[0‖J]
10	MMj_1j_0X	MM[IX+J]
11	$MMj_1j_0k_1k_0$	MM[JK]

(注：在上述表格中，0‖J 表示 J 字节的八位跟在全零字节后。IX+J 表示 IX+0‖J。)

表 3.3b

cond	书写形式	说　明
0000		1
0001		(未定)
0010	NKZ	¬(K∨Z)
0011	KZ	K∨Z
0100	NK	¬K
0101	K	K
0110	NZ	¬Z
0111	Z	Z
1000	NV	¬V
1001	V	V
1010	NN	¬N
1011	N	N
1100	NY	¬(N∀V)
1101	Y	N∀V
1110	NYZ	¬(Z∨(N∀V))
1111	YZ	Z∨(N∀V)

op	书写形式	说　明
0000	:−	:−
0001	−	−
0010	:−−	:−−
0011		(未定)
0100	:&	:∧
0101	&	∧
0110		(见 r:=α)
0111		(见 α′:=r)
1000	:#	:∀
1001	:++	:++
1010	:U	:∨
1011	:+	:+
1100		(见 IX−γ)
1101		(见 JS..)
1110		(见 ρ:=γ)
1111		(见 γ′:=ρ)

6800 注解

1．如表中所示，每个操作都要影响条件标志 H、N、Z、V、K：

✓　其值由操作的结果决定

0，1　操作后的实际值(无论操作数为何值)

·　操作前后数值不变

α_7　与数据位有关，标志取操作前该位的值

α'_7　与数据位有关，标志取结果中该位的值

α　与数值有关，若该值为零，则 Z:＝1；若不为零，则 Z:＝0

α'　与结果有关，标志 H、Z、V、K 的取值可见第二章的定义

①　只有当 $\beta'=80$(十六进制)时 V:＝1；否则 V:＝0

②　只有当 $\beta'=00$(十六进制)时 K:＝1；否则 K:＝0

③　移位后 V＝N∀K

④　K:＝K∨(A＞99)

⑤　H 只受 :＋ 与 :＋＋ 影响；不受其它操作符干扰。

⑥　见注 4 与注 5

⑦　见注 7。

2．十进制调整指令 19(十六进制码)写作 $A=2^4a_1+a_0$，相当于：

if a_0＞9∨H　**then**A:＋06；

if a_1＞9∨K　**then**A:＋60；(见§2.4.9)。

3．从中断返回，相当于：

SP:＋7；C:＝M[SP－6]；B:＝M[SP－5]；A:＝M[SP－4]；IX:＝MM[SP－3]；PC:＝MM[SP－1]。

4．等待中断，相当于：

MM[SP−1]:=PC+1;

MM[SP−3]:=IX;

M[SP−4]:=A;

M[SP−5]:=B;

M[SP−6]:=C;

SP:−7;(等待 IRQ 信号);

I:=1; PC:=MMFFF8。

(注意：除非 I=0，否则机器不接受 IRQ 信号。)

5. "软件中断"——相当于：

MM[SP−1]:=PC+1;

MM[SP−3]:=IX;

M[SP−4]:=A;

M[SP−5]:=B;

M[SP−6]:=C;

SP:−7; I:=1;

PC:=MMFFFA。

6. 这个操作的说明见§2.4.8。

7. §2.4.7 介绍了该操作。标志 N 设置得与从 IX 的高半中减去 γ 的高半形成的结果的最高位一样。若在上述部分减法中产生溢出，则标志 V 置位(如§2.4.5 讨论的那样)。若 γ 的十六位值与 IX 的值相同，则标志 Z 置位。不影响其它标志。

8. **if** cond **then** PC:=PC+J+2**else** PC:+2。

9. MM[SP−1]:=PC+2; SP:−2; PC:=PC+J+2。

10. MM[SP−1]:=PC+3; SP:−2; PC:=JK。

11. MM[SP−1]:=PC+2; SP:−2; PC:=IX+J。

12. SP:+2; PC:=MM[SP−1]。

6800 卡诺图

表 3.4 6800 卡诺图

γ,γ'	α,α'								
				06 C:=A			07 A:=C		01 NULL
		10 A:-B		16 B:=A			17 A:=B		11 A-B
		30 IX:=SP+1	32 A:=S	36 S:=A	34 SP:-1	35 SP:=IX-1	37 S:=B	33 B:=S	31 SP:+1
	J cond+[J]	20	22 NKZ	26 NZ	24 NK	25 K	27 Z	23 KZ	
	J cond+[J]	28 NV	2A NN	2E NYZ	2C NY	2D Y	2F YZ	2B N	29 V
				3E W̃I			3F SI	3B RI	39 RET
								1B A:+B	19 DEC
		08 IX:+1	0A V:=0	0E I:=0	0C K:=0	0D K:=1	0F I:=1	0B V:=1	09 IX:-1
JK	J	88	8A	8E	8C	8D JS+[J]		8B	99
MMZ[J]	MZ[J]	98 A:#α	9A A:∪α	9E SP:=γ	9C IX-γ		9F	9B A:+α	99 A:++α
MM[K]	M[K]	B8	BA	BE	BC	BD JS[K]	BF γ':=SP	BB	B9
MM[J]X	M[J]X	A8	AA	AE	AC	AD JS[J]X	AF	AB	A9
MM[J]X	M[J]X	A0	A2	A6	A4	A5	A7		A1
MM[K]	M[K]	B0	B2	B6	B4	B5	B7 α':=A		B1
MMZ[J]	MZ[J]	90 A:-α	92 A:--α	96 A:=α	94 A:&α	95 A&α	97		91 A-α
JK	J	80	82	86	84	85			81

								β / α,α'	γ,γ'
	43	47		44	46		40	A	
	53	57		54	56		50	B	
	73 β:#	77 β/2		74 β:->	76 βK:⊕R		70 β:-	M[K]	
	63	67		64	66		60	M[J]X	
69		6F	6D	6C	6E J+[J]	6A	68	M[J]X	
79		7F	7D	7C	7E J[K]	7A	78	M[K]	
59 βK:@L		5F β:=0	5D β	5C β:+1		5A β:-1	5B β:•2	B	
49		4F	4D	4C		4A	48	A	
C9	CB				CE	CA	C8	J	JK
D9 B:++α	DB B:+α	DF			DE IX:=γ	DA B:∪α	D8 B:#α	MZ[J]	MMZ[J]
F9	FB	FF γ':=IX			FE	FA	F8	M[K]	MM[K]
E9	EB	EF			EE	EA	E8	M[J]X	MM[J]X
E1		E7	E5	E4	E6	E2	E0	M[J]X	MM[J]X
F1		F7 α':=B	F5	F4	F6	F2	F0	M[K]	MM[K]
D1 B-α		D7	D5 B&α	D4 B&α	D6 B:=α	D2 B:--α	D0 B:-α	MZ[J]	MMZ[J]
C1			C5	C4	C6	C2	C0	J	JK

6800 指令表(按操作码的数值排列)

表 3.5 6800 指令表

00			40		A:−	80	12	A:−12	C0	12	B:−12
01		NULL	41			81	12	A−12	C1	12	B−12
02			42			82	12	A:− −12	C2	12	B:− −12
03			43		A: #	83			C3		
04			44		A:− >	84	12	A:&12	C4	12	B:&12
05			45			85	12	A&12	C5	12	B&12
06		C:=A	46		AK:@R	86	12	A:=12	C6	12	B:=12
07		A:=C	47		A:/2	87			C7		
08		IX:+1	48		A:•2	88	12	A: #12	C8	12	B: #12
09		IX:−1	49		AK:@L	89	12	A:+ +12	C9	12	B:+ +12
0A		V:=0	4A		A:−1	8A	12	A:U12	CA	12	B:U12
0B		V:=1	4B			8B	12	A:+12	CB	12	B:+12
0C		K:=0	4C		A:+1	8C	12 34	IX−1234	CC		
0D		K:=1	4D		A	8D	12	JS+12	CD		
0E		I:=0	4E			8E	12 34	SP:=1234	CE	12 34	IX:=1234
0F		I:=1	4F		A:=0	8F			CF		
10		A:−B	50		B:−	90	12	A:−MZ12	D0	12	B:−MZ12
11		A−B	51			91	12	A−MZ12	D1	12	B−MZ12
12			52			92	12	A:− −MZ12	D2	12	B:− −MZ12
13			53		B: #	93			D3		
14			54		B:− >	94	12	A:&MZ12	D4	12	B:&MZ12
15			55			95	12	A&MZ12	D5	12	B&MZ12
16		B:=A	56		BK:@R	96	12	A:=MZ12	D6	12	B:=MZ12
17		A:=B	57		B:/2	97	12	MZ12:=A	D7	12	MZ12:=B
18			58		B:•2	98	12	A: #MZ12	D8	12	B: #MZ12
19		DEC	59		BK:@L	99	12	A:+ +MZ12	D9	12	B:+ +MZ12
1A			5A		B:−1	9A	12	A:UMZ12	DA	12	B:UMZ12
1B		A:+B	5B			9B	12	A:+MZ12	DB	12	B:+MZ12
1C			5C		B:+1	9C	12	IX−MMZ12	DC		
1D			5D		B	9D			DD		
1E			5E			9E	12	SP:=MMZ12	DE	12	IX:=MMZ12
1F			5F		B:=0	9F	12	MMZ12:=SP	DF		MMZ12:=IX
20	12	J+12	60	12	M12X:−	A0	12	A:−M12X	E0	12	B:−M12X
21			61			A1	12	A−M12X	E1	12	B−M12X
22	12	JNKZ+12	62			A2	12	A:− −M12X	E2	12	B:− −M12X
23	12	JKZ+12	63	12	M12X: #	A3			E3		
24	12	JNK+12	64	12	M12X:− >	A4	12	A:&M12X	E4	12	B:&M12X
25	12	JK+12	65			A5	12	A&M12X	E5	12	B&M12X
26	12	JNZ+12	66	12	M12XK:@R	A6	12	A:=M12X	E6	12	B:=M12X
27	12	JZ+12	67	12	M12X:/2	A7	12	M12X:=A	E7	12	M12X:=B
28	12	JNV+12	68	12	M12X:•2	A8	12	A: #M12X	E8	12	B: #M12X
29	12	JV+12	69	12	M12XK:@L	A9	12	A:+ +M12X	E9	12	B:+ +M12X
2A	12	JNN+12	6A	12	M12X:−1	AA	12	A:UM12X	EA	12	B:UM12X
2B	12	JN+12	6B			AB	12	A:+M12X	EB	12	B:+M12X
2C	12	JNY+12	6C	12	M12X:+1	AC	12	IX−MM12X	EC		
2D	12	JY+12	6D	12	M12X	AD	12	JS12X	ED		
2E	12	JNYZ+12	6E	12	J12X	AE	12	SP:=MM12X	EE	12	IX:=MM12X
2F	12	JYZ+12	6F	12	M12X:=0	AF	12	MM12X:=SP	EF	12	MM12X:=IX
30		IX:=SP+1	70	12 34	M1234:−	B0	12 34	A:−M1234	F0	12 34	B:−M1234
31		SP:+1	71			B1	12 34	A−M1234	F1	12 34	B−M1234
32		A:=S	72			B2	12 34	A:− −M1234	F2	12 34	B:− −M1234
33		B:=S	73	12 34	M1234: #	B3			F3		
34		SP:−1	74	12 34	M1234:− >	B4	12 34	A:&M1234	F4	12 34	B:&M1234
35		SP:=IX−1	75			B5	12 34	A&M1234	F5	12 34	B&M1234
36		S:=A	76	12 34	M1234K:@R	B6	12 34	A:=M1234	F6	12 34	B:=M1234
37		S:=B	77	12 34	M1234:/2	B7	12 34	M1234:=A	F7	12 34	M1234:=B
38			78	12 34	M1234:•2	B8	12 34	A: #M1234	F8	12 34	B: #M1234
39		RET	79	12 34	M1234K:@L	B9	12 34	A:+ +M1234	F9	12 34	B:+ +M1234
3A			7A	12 34	M1234:−1	BA	12 34	A:UM1234	FA	12 34	B:UM1234
3B		RI	7B			BB	12 34	A:+M1234	FB	12 34	B:+M1234
3C			7C	12 34	M1234:+1	BC	12 34	IX−MM1234	FC		
3D			7D	12 34	M1234	BD	12 34	JS1234	FD		
3E		WI	7E	12 34	J1234	BE	12 34	SP:=MM1234	FE	12 34	IX:=MM1234
3F		SI	7F	12 34	M1234:=0	BF	12 34	MM1234:=SP	FF	12 34	MM1234:=IX

§3.3 Intel 8080 与 8085

8080 微处理机

字长：8 位　　地址：16 位

单字长寄存器：　A、B、C、D、E、F、H、L

双字长寄存器：　BC≡B ‖ C、DE≡D ‖ E、HL≡H ‖ L、AF≡A ‖ F、SP、PC

条件标志：(i) 条件标志寄存器中：

N≡F.7("负")　Z≡F.6("零")

H≡F.4("半进位")　P≡F.2("偶校验")

K≡F.0("进位")

(注：F.5≡0, F.3≡0, F.1≡1, 都是常数)

(ii) 独立的标志：

I(中断开放)

堆栈指示器：　SP

程序计数器：　PC

累加器：　A、HL

双字长寻址：　MM[x]≡M[x+1] ‖ M[x]

复位信号的作用：I:=0;　PC:=0000

对中断请求信号 INT 的响应：

执行完现行指令后，若 I=1，则：

I:=0; SP:=−2; MM[SP]:=PC

然后执行在 INTA 时放在数据总线上的一条单字节指令

(若使用"系统控制器" 8228，则可将任意指令放到数据总线上去)。

专用地址(不属于 M0000～MFFFF)：

00～FF 供 G00～GFF 输入-输出转接口用。

表 3.6 8080 指令系统——总表

I 76543210	十六进制码	书写形式	操作说明	标志(注1) N Z H P K	PC :+	索引	分类
00000000	00	NULL		· · · · ·	1		空操作
00 r 110	—	r:=j₁j₀	r:=J	· · · · ·	2	2.3	传值 单字长
00 ρ 0001	-1	ρ:=k₁k₀j₁j₀	ρ:=KJ	· · · · ·	3	2.3	传值 双字长
01 r₁ r₂	—	r₁:=r₂	r₁:=r₂	· · · · ·	1	2.3	传值 单字长
00110010	32	Mk₁k₀j₁j₀:=A	M[KJ]:=A	· · · · ·	3	2.3	传值 双字长
00111010	3A	A:=Mk₁k₀j₁j₀	A:=M[KJ]	· · · · ·	3	2.3	传值 双字长
00000010	02	M[BC]:=A	M[BC]:=A	· · · · ·	1	2.3	传值 单字长
00001010	0A	A:=M[BC]	A:=M[BC]	· · · · ·	1	2.3	传值 单字长
00010010	12	M[DE]:=A	M[DE]:=A	· · · · ·	1	2.3	传值 单字长
00011010	1A	A:=M[DE]	A:=M[DE]	· · · · ·	1	2.3	传值 单字长
00100010	22	MMk₁k₀j₁j₀:=HL	MM[KJ]:=HL	· · · · ·	3	2.3	传值 双字长
00101010	2A	HL:=MMk₁k₀j₁j₀	HL:=MM[KJ]	· · · · ·	3	2.3	传值 双字长
11111001	F9	SP:=HL	SP:=HL	· · · · ·	1	2.3, 2.5	传值 双字长
00110111	37	K:=1		· · · · 1	1	2.3.7	标志
11111011	FB	I:=1	(见注 5)	· · · · ·	1	2.9	I/O
11110011	F3	I:=0		· · · · ·	1	2.9	I/O
11011011	DB	A:=Gj₁j₀	A:=G[J]	· · · · ·	2	2.8	I/O
11010011	D3	Gj₁j₀:=A	G[J]:=A	· · · · ·	2	2.8	I/O
11ρ′0001	-1	ρ′:=ST	ρ′:=MM[SP]; SP:+2	* * * * *	1	2.5	堆栈
11ρ′0101	-5	ST:=ρ′	SP:−2; MM[SP]:=ρ′	· · · · ·	1	2.5	堆栈
11101011	EB	HL:=:DE	HL:=:DE	· · · · ·	1	2.3.4	交换
11100011	E3	HL:=:ST	HL:=:MM[SP]	· · · · ·	1	2.3.4, 2.5	交换
00 r 100	—	r:+1	r:=r+1	r′ r′ r′ r′ ·	1	2.4.9	加一/减一
00 r 101	—	r:−1	r:=r−1	r′ r′ r′ r′ ·	1	2.4.9	加一/减一
00ρ0011	-3	ρ:+1	ρ:=ρ+1	· · · · ·	1	2.4.9	加一/减一
00ρ1011	-B	ρ:−1	ρ:=ρ−1	· · · · ·	1	2.4.9	加一/减一
10 *op* r	—	A*op*r	A*op*r	A′ A′ A′ A′ A′	1	2.4	算术-逻辑 单字长
11 *op* 110	—	A*op*j₁j₀	A*op*J	A′ A′ A′ A′ A′	2	2.4	算术-逻辑 双字长
00101111	2F	A:≠	A:=¬A	· · · · ·	1	2.4.6	算术-逻辑 单字长
00100111	27	DEC	(见注 2)	A′ A′ A′ A′ A′	1	2.4.9	算术-逻辑 单字长
00000111	07	A:@L	(见注 4)	· · · · A₇	1	2.4.8	算术-逻辑 单字长
00001111	0F	A:@R		· · · · A₀	1	2.4.8	算术-逻辑 单字长
00010111	17	AK:@L		· · · · A₇	1	2.4.8	算术-逻辑 单字长
00011111	1F	AK:@R		· · · · A₀	1	2.4.8	算术-逻辑 单字长
00ρ1001	-9	HL:+ρ	HL:=HL+ρ	· · · · ⊙	1	2.4.3.2	双字长
00111111	3F	K:≠	K:=¬K	· · · · 反	1	2.3.7	标志
11101001	E9	J[HL]	PC:=HL	· · · · ·	–	2.6	转移
11000011	C3	Jk₁k₀j₁j₀	PC:=KJ	· · · · ·	–	2.6	转移
11*cnd*010	—	J*cnd*k₁k₀j₁j₀	(见注 6)	· · · · ·	–	2.6	转移
11001101	CD	JSk₁k₀j₁j₀	(见注 7)	· · · · ·	–	2.7	调用
11*cnd*100	—	JS*cnd*k₁k₀j₁j₀	(见注 8)	· · · · ·	–	2.7	调用
11 s 111	—	JSs	(见注 9)	· · · · ·	–	2.7	调用
11001001	C9	RET	(见注 10)	· · · · ·	–	2.7	返回
11*cnd*000	—	R*cnd*	(见注 11)	· · · · ·	–	2.7	返回
01110110	76	HALT	(见注 3)	· · · · ·	(1)	2.9	特殊

8080 代换表

表 3.7　8080 代换表

r r_1 r_2	书写形式	说明
000	B	B
001	C	C
010	D	D
011	E	E
100	H	H
101	L	L
110	M	M[HL]†
111	A	A

p	书写形式	说明
00	BC	BC
01	DE	DE
10	HL	HL
11	SP	SP

p′	书写形式	说明
00	BC	BC
01	DE	DE
10	HL	HL
11	AF	AF

†若 $r_1=r_2=110$，则不定（见 HALT）

op	书写形式	说明
000	:+	:+
001	:+ +	:+ +
010	:−	:−
011	:− −	:− −
100	:&	:∧
101	:≠	:∀
110	:U	:∨
111	−	−

cnd	书写形式	说明
000	NZ	¬Z
001	Z	Z
010	NK	¬K
011	K	K
100	NP	¬P
101	P	P
110	NN	¬N
111	N	N

s	书写形式	说明
000	0	0
001	1	1
010	2	2
011	3	3
100	4	4
101	5	5
110	6	6
111	7	7

8080 注解

1. 如表中所示，每个操作都要影响条件标志 **N、Z、H、P K**：

• 操作前后数值不变

* 除非是 AF:＝ST(十六进制 F1)操作，否则不受影响。

A_7 标志取操作前该位的值

A_7' 标志取结果中该位的值

对 Z 标志：r′ 表示若操作后 r＝00，则 Z:＝1；否则 Z:＝0

对 H 标志：r′ 表示若操作产生 r_3 向 r_4 的进位，则 H:＝1；否则 H:＝0

对 P 标志：r′ 表示 P:＝¬ $(r_7'\forall r_6'\forall r_5'\forall r_4'\forall r_3'\forall r_2'\forall r_1'\forall r_0')$（即若操作的结果 r′ 有偶数个“1”，则

P:=1)

对 K 标志：r′ 表示若操作在 r_7 产生进位，则 K:=1

① 表示若操作在 H_7 产生进位，则 K:=1

在 :#、:U 操作后 K=0、H=0

在 :& 操作后 K=0、H=1

ρ:+1 或 ρ:−1 不影响任何标志

K 不受 r:+1 或 r:−1 影响

只有 K 可能受 HL:+ρ 影响。

2．十进制调整指令 27(十六进制码)写作 $A=2^4a_1+a_0$，相当于：

if $a_0>9\vee H$ **then** A:+06;

if $a_1>9\vee K$ **then** A:+60;（见 § 2.4.9)。

3．HALT 指令 76(十六进制码)：停止执行指令周期，直至接收到一个(硬件)信号 RESET 或 INT 为止。

4．“循环移位”：该操作的说明见 § 2.4.8。

5．执行完下一条指令后 I:=1。

6．**if** cnd **then** PC:=KJ **else** PC:+3

7．SP:−2; MM[SP]:=PC+3; PC:=KJ

8．**if** cnd **then begin** SP:−2; MM[SP]:=PC+3;
PC:=KJ **end**
else PC:+3。

9．SP:−2; MM[SP]:=PC+1; PC:=8×S。

10．SP:+2; PC:=MM[SP−2]。

11．**if** cnd **then begin** SP:+2; PC:=MM[SP−2]
end
else PC:+1。

8080 卡诺图

表 3.8 8080 卡诺图

	r_2 \ r_1	B	D	M[HL]	H	L	A	E	C
BC	B	00 NULL	02 M[BC] :=A	06	04	05	07 A:@L	03	01
DE	D	(shaded)	12 M[DE] :=A	16	14	15	17 AK:@L	13 ρ:+1	11 ρ:=KJ
SP	M[HL]	(shaded)	32 M[KJ] :=A	36	34	35	37 K:=1	33	31
HL	H	(shaded)	22 MM[KJ] :=HL	26 r:=J	24 r:+1	25 r:-1	27 DEC	23	21
HL	L	(shaded)	2A HL:= MM[KJ]	2E	2C	2D	2F A:#	2B	29
SP	A	(shaded)	3A A:= M[KJ]	3E	3C	3D	3F K:#	3B ρ:-1	39 HL:+ρ
DE	E	(shaded)	1A A:= M[DE]	1E	1C	1D	1F AK:@R	1B	19
BC	C	(shaded)	0A A:= M[BC]	0E	0C	0D	0F A:@R	0B	09

C	E	A	L	H	M[HL]	D	B	r_1 \ r_2
41	43	47	45	44	46	42	40	B
51	53	57	55	54	56	52	50	D
71	73	77	75	74	76 HALT	72	70	M[HL]
61	63	67	65	64	66	62	60	H
69	6B	6F	6D r_1:=r_2	6C	6E	6A	68	L
79	7B	7F	7D	7C	7E	7A	78	A
59	5B	5F	5D	5C	5E	5A	58	E
49	4B	4F	4D	4C	4E	4A	48	C

op \ r	B	D	M[HL]	H	L	A	E	C
:++	88	8A	8E	8C	8D	8F	8B	89
:--	98	9A	9E	9C	9D	9F	9B	99
[illegible]	B8	BA	BE	BC	BD	BF	BB	B9
:#	A8	AA	AE	AC	AD A op r	AF	AB	A9
:&	A0	A2	A6	A4	A5	A7	A3	A1
:U	B0	B2	B6	B4	B5	B7	B3	B1
:-	90	92	96	94	95	97	93	91
:+	80	82	86	84	85	87	83	81

C	E	A	L	H	M[HL]	D	B	cnd	ρ'
C9 RET	(shaded)	CF (1)	CD JS[KJ]	CC	CE	CA	C8	Z	
(shaded)	DB A:= G[J]	DF (3)	(shaded)	DC	DE	DA	D8	K	
F9 SP:=HL	FB I:=1	FF (7)	(shaded)	FC	FE	FA	F8	N	
E9 J[HL]	EB HL:=: DE	EF (5) JS s	(shaded)	EC JS cnd [KJ]	EE A op J	EA J cnd [KJ]	E8 R cnd	P	
E1	E3 HL:=: ST	E7 (4)	E5	E4	E6	E2	E0	NP	HL
F1 ρ':=ST	F3 I:=0	F7 (6)	F5 ST:=ρ'	F4	F6	F2	F0	NN	AF
D1	D3 G[J] :=A	D7 (2)	D5	D4	D6	D2	D0	NK	DE
C1	C3 J[KJ]	C7 (0)	C5	C4	C6	C2	C0	NZ	BC

(s)

8080 指令表(按操作码的数值排列)

表 3.9 8080 指令表

00			NULL	40		B:=B	80		A:+B	C0			RNZ
01	34	12	BC:=1234	41		B:=C	81		A:+C	C1			BC:=ST
02			M[BC]:=A	42		B:=D	82		A:+D	C2	34	12	JNZ1234
03			BC:+1	43		B:=E	83		A:+E	C3	34	12	J1234
04			B:+1	44		B:=H	84		A:+H	C4	34	12	JSNZ1234
05			B:-1	45		B:=L	85		A:+L	C5			ST:=BC
06	12		B:=12	46		B:=M	86		A:+M	C6	12		A:+12
07			A:@L	47		B:=A	87		A:+A	C7			JSS0
08				48		C:=B	88		A:++B	C8			RZ
09			HL:+BC	49		C:=C	89		A:++C	C9			RET
0A			A:=M[BC]	4A		C:=D	8A		A:++D	CA	34	12	JZ1234
0B			BC:-1	4B		C:=E	8B		A:++E	CB			
0C			C:+1	4C		C:=H	8C		A:++H	CC	34	12	JSZ1234
0D			C:-1	4D		C:=L	8D		A:++L	CD	34	12	JS1234
0E	12		C:=12	4E		C:=M	8E		A:++M	CE	12		A:++12
0F			A:@R	4F		C:=A	8F		A:++A	CF			JSS1
10				50		D:=B	90		A:-B	D0			RNK
11	34	12	DE:=1234	51		D:=C	91		A:-C	D1			DE:=ST
12			M[DE]:=A	52		D:=D	92		A:-D	D2	34	12	JNK1234
13			DE:+1	53		D:=E	93		A:-E	D3	12		G12:=A
14			D:+1	54		D:=H	94		A:-H	D4	34	12	JSNK1234
15			D:-1	55		D:=L	95		A:-L	D5			ST:=DE
16	12		D:=12	56		D:=M	96		A:-M	D6	12		A:-12
17			AK:@L	57		D:=A	97		A:-A	D7			JSS2
18				58		E:=B	98		A:--B	D8			RK
19			HL:+DE	59		E:=C	99		A:--C	D9			
1A			A:=M[DE]	5A		E:=D	9A		A:--D	DA	34	12	JK1234
1B			DE:-1	5B		E:=E	9B		A:--E	DB	12		A:=G12
1C			E:+1	5C		E:=H	9C		A:--H	DC	34	12	JSK1234
1D			E:-1	5D		E:=L	9D		A:--L	DD			
1E	12		E:=12	5E		E:=M	9E		A:--M	DE	12		A:--12
1F			AK:@R	5F		E:=A	9F		A:--A	DF			JSS3
20				60		H:=B	A0		A:&B	E0			RNP
21	34	12	HL:=1234	61		H:=C	A1		A:&C	E1			HL:=ST
22	34	12	MM1234:=HL	62		H:=D	A2		A:&D	E2	34	12	JNP1234
23			HL:+1	63		H:=E	A3		A:&E	E3			HL:=:ST
24			H:+1	64		H:=H	A4		A:&H	E4	34	12	JSNP1234
25			H:-1	65		H:=L	A5		A:&L	E5			ST:=HL
26	12		H:=12	66		H:=M	A6		A:&M	E6	12		A:&12
27			DEC	67		H:=A	A7		A:&A	E7			JSS4
28				68		L:=B	A8		A: #B	E8			RP
29			HL:+HL	69		L:=C	A9		A: #C	E9			J[HL]
2A	34	12	HL:=MM1234	6A		L:=D	AA		A: #D	EA	34	12	JP1234
2B			HL:-1	6B		L:=E	AB		A: #E	EB			HL:=:DE
2C			L:+1	6C		L:=H	AC		A: #H	EC	34	12	JSP1234
2D			L:-1	6D		L:=L	AD		A: #L	ED			
2E	12		L:=12	6E		L:=M	AE		A: #M	EE	12		A: #12
2F			A: #	6F		L:=A	AF		A: #A	EF			JSS5
30				70		M:=B	B0		A:UB	F0			RNN
31	34	12	SP:=1234	71		M:=C	B1		A:UC	F1			AF:=ST
32	34	12	M1234:=A	72		M:=D	B2		A:UD	F2	34	12	JNN1234
33			SP:+1	73		M:=E	B3		A:UE	F3			I:=0
34			M:+1	74		M:=H	B4		A:UH	F4	34	12	JSNN1234
35			M:-1	75		M:=L	B5		A:UL	F5			ST:=AF
36	12		M:=12	76		HALT	B6		A:UM	F6	12		A:U12
37			K:=1	77		M:=A	B7		A:UA	F7			JSS6
38				78		A:=B	B8		A-B	F8			RN
39			HL:+SP	79		A:=C	B9		A-C	F9			SP:=HL
3A	34	12	A:=M1234	7A		A:=D	BA		A-D	FA	34	12	JN1234
3B			SP:-1	7B		A:=E	BB		A-E	FB			I:=1
3C			A:+1	7C		A:=H	BC		A-H	FC	34	12	JSN1234
3D			A:-1	7D		A:=L	BD		A-L	FD			
3E	12		A:=12	7E		A:=M	BE		A-M	FE	12		A-12
3F			K: #	7F		A:=A	BF		A-A	FF			JSS7

8085 微处理机

除下述情况外，8080 的说明也适用于 8085。

条件标志：除 8080 已有的以外，还有：

(iii) 三位的中断屏蔽标志 IM≡IM7 ‖ IM6 ‖ IM5

(iv) "串行输出数据"T，表示 SOD（串行输出数据）端数值有效。

复位信号：其作用是：

I:=0; IM:=111; T:=0; PC:=0000

对中断请求信号的响应：完成现行指令后

```
if TRAP then begin I:=0; JS0024 end
        else if I then begin if RST 7.5∧¬IM7 then
          begin I:=0; JS003C end
                    else if RST6.5∧¬IM6 then
                      begin I:=0; JS0034 end
                    else if RST5.5∧¬IM5 then
                      begin I:=0; JS002C end
                    else if INTR then
                      begin I:=0; ST:=PC;
                            ⟨bus⟩ end
```

其中 JS003C 表示 SP:−2; MM[SP]:=PC; PC:=003C;

ST:=PC 表示 SP:−2; MM[SP]:=PC;

⟨bus⟩ 表示"执行中断设备放到数据总线上的指令"。

指令系统：与 8080 的指令系统相同，只是再增加下述两条指令：

00100000	20	A:=IM	(注 5)……	1
00110000	30	IM:=A	(注 6)……	1

代换表：与 8080 的相同

注解：与8080的相同，但增加下述两条：

5 其作用是：

$A_0:=IM5$; $A_1:=IM6$; $A_2:=IM7$; $A_3:=I$;

$A_4:=RST5.5$; $A_5:=RST6.5$; $A_6:=RST7.5$;

$A_7:=SID$。

6 其作用是：

$IM5:=IM5\wedge\overline{A}_3\vee A_0\wedge A_3$;

$IM6:=IM6\wedge\overline{A}_3\vee A_1\wedge A_3$;

$IM7:=(IM7\wedge\overline{A}_3\vee A_2\wedge A_3)\wedge\overline{A}_4$;

$T:=T\wedge\overline{A}_6\vee A_7\wedge A_6$。

卡诺图与指令表中应插入新增加的两条指令20(A:=IM)与30(IM:=A)。

§3.4 Zilog Z80

Z80微处理机

字长：8位　　地址：16位

单字长寄存器：A、B、C、D、E、F、H、L、Q、R

双字长寄存器：BC≡B‖C、DE≡D‖E、HL≡H‖L、AF≡A‖F、IX、IY、PC、SP、B′C′、D′E′、H′L′、A′F′

条件标志：(i) 条件标志寄存器中：

N≡F.7(负)

Z≡F.6(零)

H≡F.4(半进位)

P≡V≡F.2(偶校验或溢出)

S≡F.1(减法)　K≡F.0(进位)

(F.5与F.3的值不定)

(ii) 独立的标志：

IM(中断模式)

堆栈指示器：SP

变址寄存器：IX、IY

程序计数器：PC

累加器：A、HL、IX、IY

专用寄存器：Q(中断模式2所用的间接地址的高半——见下文)

R(7位——动态RAM再生地址计数器)

双字长寻址：$MM[x]\equiv M[x+1]\parallel M[x]$

专用地址(不属于M0000～MFFFF)：

00～FF供G00～GFF输入-输出转接口用

复位信号的作用：I:=0; IM:=0; Q:=00; R:=00; PC:=0000

对中断请求信号的响应：

INT：完成现行指令后，若I=1，则：

I:=0; SP:−2; MM[SP]:=PC; 然后：

若IM=0 执行中断设备放在数据总线上的指令(任意指令)。

若IM=1 PC:=0038。

若IM=2 PC:=MM[Q‖p]，其中p是中断设备放在数据总线上的数值为偶数的一个字节。

NMI：完成现行指令后无论I为何值都执行：

SP:−2; MM[SP]:=PC; PC:=0066。

在这个中断过程中，I=0；返回至被中断的程序(RNMI)时I恢复其原值。

Z80指令系统——总表

表3.10 Z80指令系统——总表

I_0 76543210	I或I_1 76543210	J 76543210	K 76543210	I_0	I或I_1	J	K	书写形式	操作说明	标志(注1) N	Z	H	P	S	K	PC :+	索引	分类
	00000000				00			NULL		•	•	•	•	•	•	1	2.3	空操作
	01 r_1 r_2				--			$r_1:=r_2$	$r_1:=r_2$							1	2.3	
	00 r 110	j_1 j_0			--	--		r:=j_1j_0	r:=J		•	•	•	•	•	2	2.3	单字长
	00110010	j_1 j_0	k_1 k_0		32	--	--	$Mk_1k_0j_1j_0$:=A	M[KJ]:=A	•	•	•	•	•	•	3	2.3	
	00111010	j_1 j_0	k_1 k_0		3A	--	--	A:=$Mk_1k_0j_1j_0$	A:=M[KJ]	•	•	•	•	•	•	3	2.3	
	00000010				02			M[BC]:=A	M[BC]:=A	•	•	•	•	•	•	1	2.3	
	00001010				0A			A:=M[BC]	A:=M[BC]	•	•	•	•	•	•	1	2.3	
	00010010				12			M[DE]:=A	M[DE]:=A	•	•	•	•	•	•	1	2.3	
	00011010				1A			A:=M[DE]	A:=M[DE]	•	•	•	•	•	•	1	2.3	
11x11101	00110110	j_1 j_0	k_1 k_0	-D	36	--	--	Mj_1j_0x:=k_1k_0	M[Ix+J]:=K	•	•	•	•	•	•	4	2.3	
11x11101	01 r' 110	j_1 j_0		-D	--	--		Mj_1j_0x:=r'	M[Ix+J]:=r'	•	•	•	•	•	•	2	2.3	
11x11101	01110 r'	j_1 j_0		-D	7-	--		r':=Mj_1j_0x	r':=M[Ix+J]	•	•	•	•	•	•	2	2.3	
	00 ρ 0001	j_1 j_0	k_1 k_0		-1	--	--	ρ:=$k_1k_0j_1j_0$	ρ:=KJ		•	•	•	•	•	3	2.3	
11x11101	00100001	j_1 j_0	k_1 k_0	-D	21	--	--	Ix:=$k_1k_0j_1j_0$	Ix:=KJ	•						4	2.3	
	00100010	j_1 j_0	k_1 k_0		22	--	--	$MMk_1k_0j_1j_0$:=HL	MM[KJ]:=HL		•	•	•	•	•	3	2.3	双字长
	00101010	j_1 j_0	k_1 k_0		2A	--	--	HL:=$MMk_1k_0j_1j_0$	HL:=MM[KJ]		•	•	•	•	•	3	2.3	
11101101	01 ρ "0011	j_1 j_0	k_1 k_0	ED	-3	--	--	$MMk_1k_0j_1j_0$:=ρ"	MM[KJ]:=ρ"		•	•	•	•	•	4	2.3	
11101101	01 ρ" 1011	j_1 j_0	k_1 k_0	ED	-B	--	--	ρ":=$MMk_1k_0j_1j_0$	ρ":=MM[KJ]		•	•	•	•	•	4	2.3	
11x11101	00100010	j_1 j_0	k_1 k_0	-D	22	--	--	$MMk_1k_0j_1j_0$:=Ix	MM[KJ]:=Ix		•	•	•	•	•	4	2.3	
11x11101	00101010	j_1 j_0	k_1 k_0	-D	2A	--	--	Ix:=$MMk_1k_0j_1j_0$	Ix:=MM[KJ]		•	•	•	•	•	4	2.3	
	11111001				F9			SP:=HL	SP:=HL		•	•	•	•	•	1	2.3	
11x11101	11111001			-D	F9			SP:=Ix	SP:=Ix		•	•	•	•	•	2	2.3	
11001011	1b s r			CB	--			r.s:=b	r.s:=b		•	•	•	•	•	2	2.10	赋值
11x11101	11001011	j_1 j_0	1b s 110	-D	CB	--	--	$Mj_1j_0x.s$:=b	M[Ix+J].s:=b		•	•	•	•	•	4	2.10	位
	00110111				37			K:=1		!	•	•	0	•	0	1	2.3.7	标志
	1111b011				F-			I:=b	(注7)		•	•	•	•	•	1	2.9	
11001011	01 s r				CB	--		Z:= #r.s	Z:=¬(r.s)	?	e	1	?	0		2	2.10	I/O
11x11101	11001011	j_1 j_0	01 s 110	-D	CB	--	--	Z:= #$Mj_1j_0x.s$	Z:=¬(M[Ix+J].s)	?	e	1	?	0		4	2.10	
	11011011	j_1 j_0			DB	--		A:=Gj_1j_0	A:=G[J]		•	•	•	•	•	2	2.8	
	11010011	j_1 j_0			D3	--		Gj_1j_0:=A	G[J]:=A		•	•	•	•	•	2	2.8	特殊
11101101	01 r' 000			ED	--			r':=G[C]	r':=G[C]	G_7	G	0	p	0		2	2.8	
11101101	01 r' 001			ED	--			G[C]:=r'	G[C]:=r'		•	•	•	•	•	2	2.8	
11101101	01000111			ED	47			Q:=A	Q:=A		•	•	•	•	•	2	2.9	
11101101	01010111			ED	57			A:=Q	A:=Q	Q_7	Q	0	1	0		2	2.9	
11101101	01001111			ED	4F			R:=A	R:=A		•	•	•	•	•	2	2.10	堆栈
11101101	01011111			ED	5F			A:=R	A:=R	R_7	R	0	1	0		2	2.10	
	11 ρ'0001				-1			ρ':=ST	ρ':=MM[SP]; SP:+2	•	•	•	•	•	•	1	2.5	
	11 ρ'0101				-5			ST:=ρ'	SP:−2; MM[SP]:=ρ'		•	•	•	•	•	1	2.5	交换
11x11101	11100001			-D	E1			Ix:=ST	Ix:=MM[SP]; SP:+2;		•	•	•	•	•	2	2.5	
11x11101	11100101			-D	E5			ST:=Ix	SP:−2; MM[SP]:=Ix		•	•	•	•	•	2	2.5	
	11100011				E3			HL:=:ST	HL:=:MM[SP]		•	•	•	•	•	1	2.3, 2.5	
11x11101	11100011			-D	E3			Ix:=:ST	Ix:=:MM[SP]		•	•	•	•	•	2	2.3, 2.5	
	11101011				EB			HL:=:DE	HL:=:DE		•	•	•	•	•	1	2.3	
	00001000				08			AF:=:XX	AF:=:A'F'		•	•		•	•	1	2.10	
	11011001				D9			BL:=:XX	(注8)		•	•		•	•	1	2.10	

(续表)

	00 r 10m						--		r:ml	r:ml	!	✓	✓	✓	v	m	·	1	2.4.9	加一/减一
11x11101	0011010m	j_1	j_0			−D	3−	--	Mj_1j_0x:ml	M[Ix+J]:ml		✓	✓	✓	v	m	·	3	2.4.9	
	00ρm011						--		ρ:ml	ρ:ml		·	·	·	·	·	·	1	2.4.9	
11x11101	0010m011					−D	2−		Ix:ml	Ix:ml		·	·	·	·	·	·	2	2.4.9	
	10 *op* r						--		A*opr*	A*opr*	!	✓	✓	①	②	③	④	1	2.4	算术－逻辑
	11 *op* 110	j_1	j_0				--	--	A*op*j_1j_0	A*op*J	!	✓	✓	①	②	③	④	2	2.4	
11x11101	10 *op* 110	j_1	j_0			−D	--	--	A*op*Mj_1j_0x	A*op*M[Ix+J]		✓	✓	①	②	③	④	3	2.4	
	00101111						2F		A: #	A:=¬A	!	·	·	1	·	1	·	1	2.4.6	
11101101	01000100					ED	44		A:−	A:=−A		✓	✓	✓	v	1	✓	2	2.4.4	
	00100111						27		DEC	(注2)		✓	✓	✓	p	·	✓	1	2.4.9	
	00000111						07		A:@L	(注3)	!	·	·	0	·	0	✓	1	2.4.8	单字长移位
	00001111						0F		A:@R		!	·	·	0	·	0	✓	1	2.4.8	
	00010111						17		AK:@L		!	·	·	0	·	0	✓	1	2.4.8	
	00011111						1F		AK:@R		!	·	·	0	·	0	✓	1	2.4.8	
11001011	00 *shf* r					CB	--		r*shf*	(注3)		✓	✓	0	p	0	✓	2	2.4.8	双字长移位
11x11101	11001011	j_1	j_0	00 *shf* 110		−D	CB	----	Mj_1j_0x*shf*			✓	✓	0	p	0	✓	4	2.4.8	
11101101	01101111					ED	6F		A3M:@4L	(注3)		✓	✓	0	p	0	·	2	2.4.8	
11101101	01100111					ED	67		A3M:@4R			✓	✓	0	p	0	·	2	2.4.8	
	00ρ1001						−9		HL:+ρ	HL:+ρ	!	·	·	?	·	0	✓	1	2.4.3.2	
11101101	01ρ1010					ED	−A		HL:++ρ	HL:++ρ		✓	✓	?	v	0	✓	2	2.4.3.2	
11101101	01ρ0010					ED	−2		HL:−−ρ	HL:−−ρ		✓	✓	?	v	1	✓	2	2.4.5.2	
11x11101	00ρ″1001					−D	−9		Ix:+ρ″	Ix:+ρ″		·	·	?	·	0	✓	2	2.4.3.2	
	00111111						3F		K: #		!	·	·	?	·	0	K	1	2.3.7	标志
	11000011	j_1	j_0	k_1	k_0		C3	-- --	$Jk_1k_0j_1j_0$	PC:=KJ		·	·	·	·	·	·	–	2.6	控制传递 转移
	11101001						E9		J[HL]	PC:=HL		·	·	·	·	·	·	–	2.6	
11x11101	11101001					−D	E9		J[Ix]	PC:=Ix		·	·	·	·	·	·	–	2.6	
	00011000	j_1	j_0				18	--	J+j_1j_0	PC:=PC+J+2		·	·	·	·	·	·	–	2.6	
	11*cnd*010	j_1	j_0	k_1	k_0		--	-- --	J*cnd*$k_1k_0j_1j_0$	(注 9)		·	·	·	·	·	·	–	2.6	
	00100000	j_1	j_0				20	--	JNZ+j_1j_0	(注 10)		·	·	·	·	·	·	–	2.6	
	00101000	j_1	j_0				28	--	JZ+j_1j_0	(注 11)		·	·	·	·	·	·	–	2.6	
	00110000	j_1	j_0				30	--	JNK+j_1j_0	(注 12)		·	·	·	·	·	·	–	2.6	
	00111000	j_1	j_0				38	--	JK+j_1j_0	(注 13)		·	·	·	·	·	·	–	2.6	
	00010000	j_1	j_0				10	--	B:−1,JNZ+j_1j_0	(注 14)		·	·	·	·	·	·	–	2.10	
	11001101	j_1	j_0	k_1	k_0		CD	-- --	$JSk_1k_0j_1j_0$	(注 15)		·	·	·	·	·	·	–	2.7	调用
	11 s 111						--		JSSs	(注 16)		·	·	·	·	·	·	–	2.7	
	11*cnd*100	j_1	j_0	k_1	k_0		--	-- --	JS*cnd*$k_1k_0j_1j_0$	(注 17)		·	·	·	·	·	·	–	2.7	
	11001001						C9		RET	(注 18)		·	·	·	·	·	·	–	2.7	返回
	11*cnd*000						--		R*cnd*	(注 19)		·	·	·	·	·	·	–	2.7	
11101101	01001101					ED	4D		RI	(注 4 和 20)		·	·	·	·	·	·	–	2.9	
11101101	01000101					ED	45		RNMI	(注 4 和 20)		·	·	·	·			–	2.9	
11101101	010*im*110					ED	--		IM:=*im*	IM:=*im*		·						2	2.9	特殊
11101101	101qd0*gp*					ED	--		*gpqd*	(注 21)		(注 6)						2	2.10	
	01110110						76		HALT	(注 5)		·	·	·	·	·	·	1	2.9	

Z80 代换表

表 3.11

r r_1 r_2	书写形式	说 明
000	B	B
001	C	C
010	D	D
011	E	E
100	H	H
101	L	L
110	M	M[HL]†
111	A	A

r	书写形式	说 明
000	B	B
001	C	C
010	D	D
011	E	E
100	H	H
101	L	L
110		(未定)
111	A	A

† 若 $r_1=r_2=110$，则不定(见 HALT)。

表 3.12

ρ	书写形式	说明
00	BC	BC
01	DE	DE
10	HL	HL
11	SP	SP

ρ′	书写形式	说明
00	BC	BC
01	DE	DE
10	HL	HL
11	AF	AF

ρ″	书写形式	说明
00	BC	BC
01	DE	DE
10	Ix	Ix
11	SP	SP

ρ‴	书写形式	说明
00	BC	BC
01	DE	DE
10		(未定)
11	SP	SP

x	书写形式	说明
0	X	X
1	Y	Y

b	书写形式	说明
0	0	0
1	1	1

m	书写形式	说明
0	+	+
1	−	−

im	书写形式	说明
00	0	0
01		(未定)
10	1	1
11	2	2

s	书写形式	说明†
000	0	0(0000)
001	1	1(0008)
010	2	2(0010)
011	3	3(0018)
100	4	4(0020)
101	5	5(0028)
110	6	6(0030)
111	7	7(0038)

op	书写形式	说明
000	:+	:+
001	:++	:++
010	:−	:−
011	:−−	:−−
100	:&	:∧
101	:≠	:∨
110	:U	:∨
111	−	−

shf	书写形式	说明
000	:@L	见 §2.4.8
001	:@R	
010	K:@L	
011	K:@R	
100	:*2	
101	:/2	
110		(未定)
111	:−>	见§2.4.8

† 有关 s 的表中括号内的值是十六进制值 8×s，即所调用的地址。

表 3 13

cnd	书写形式	说明
000	NZ	¬Z
001	Z	Z
010	NK	¬K
011	K	K
100	NP	¬P
101	P	P
110	NN	¬N
111	N	N

gp	书写形式	说明
00	C	复制
01	S	搜索
10	I	输入
11	O	输出

c	书写形式	说明
0	S	单步
1	B	字组

d	书写形式	说明
0	U	向前
1	D	向后

Z80 注解

1. 如表中所示，每个操作都要影响条件标志 N、Z、H、P(≡V)、S、K：

· 操作前后数据不变

* 除非是 AF:=ST(十六进制 F1)操作，否则不受影响。

A_7 标志取操作前该位的值

A'_7 标志取结果中该位的值

e 标志取".="右边表达式的值

? 操作后标志值不定

对 N 标志：标志取结果的最高位的值

对 Z 标志：若结果为零，则 Z:=1；否则 Z:=0 G、Q、R 若传送值为零，则 Z:=1；否则 Z:=0

对 H 标志：若操作产生第 3 位向第 4 位的进位，则 H:=1；否则 H:=0

对 P(V)标志：

P：若结果有偶数个"1"，则 P:=1(参阅 8080 注 1)；否则 P:=0

V：若操作产生溢出，则 P:=1 (参阅 §2.4.3～§2.4.7)；否则 P:=0

对 K 标志：若操作在最高位产生进位，则 K:=1；否则

K:=0

① 在 :&、:#、:U 中 H:=1。

② 在 :&、:#、:U 中 P:=p；否则 P=v。

③ 在 :-、:--、- 后 S=1；否则 S=0。

④ 在 :&、:#、:U 后 K=0。

还可参阅注 6。

注意：在 Z80 与 8080 的若干指令中，对标志 H、P 的处理是不同的，这在 Z80 总表中标志栏旁用"!"来表示。

2．十进制调整指令 27(十六进制码)写作 $A=2^4a_1+a_0$，相当于：

对 S=0：**if** $a_0>9\vee$H **then** A:+06；

if $a_1>9\vee$K **then** A:+60；

对 S=1：**if** H **then** A:−06；**if** K **then** A:−60；

参阅§2.4.9。

3．这几个"移位"与"循环移位"操作在§2.4.8中已介绍了。

标志：注意：指令 07、0F、17、1F 与 8080 中对应的指令不同之处是 Z80 清除 H，而 8080 不影响 H 值。就基本作用而言，指令 CB 07、CB 0F、CB 17、CB 1F 分别与指令 07、0F、17、1F 相同；但是前一组中的指令影响 N、Z 与 P 标志，而后一组中的指令则不影响这些标志。

4．从中断返回：执行这些指令中任意一条的作用都与 RET(C9)指令相同。但是，Zilog 公司的 Z80PIO 与 Z80SIO(见§1.4.1)与 Z80 处理机一样能识别出数据总线上的这些指令。

5．HALT 指令 76(十六进制码)：停止执行指令周期，直至接收到(硬件)信号 RESET、INT 或 NMI 时为止。

6．"复制"类指令 (ED A0、ED B0、ED A8、ED B8)：N、Z、K 不受影响。H:=0，S:=0。指令后若 BC=0000，则 P=

0；否则 P=1。

“搜索”类指令（ED A1、ED B1、ED A9、ED B9）：N、Z、H 标志根据（最后的）减法 A－M[HL] 的结果进行设置。但是 K 不受影响。S:=1。执行指令后若 BC=0000，则 P=0；否则 P=1。

“输入”与“输出”类指令（ED A2、ED B2、ED AA、ED BA、ED A3、ED B3、ED AB、ED BB）：对 N、H、P 的作用不定；K 不受影响；S:=1。指令后若 B=00，则 Z=1；否则 Z=0。

这些指令在§2.10 进行介绍。

7. I:=b（若 b=1，则执行时间延长一个指令周期）。

8. BC:=:B′C′；DE:=:D′E′；HL:=:H′L′。

9. **if** cnd **then** PC:=KJ **else** PC:+3。

10. **if** ¬Z **then** PC:=PC+J+2 **else** PC:+2。

11. **if**Z **then** PC:=PC+J+2 **else** PC:+2。

12. **if** ¬K **then** PC:=PC+J+2 **else** PC:+2。

13. **if** K **then** PC:=PC+J+2 **else** PC:+2。

14. B:−1；**if** ¬Z **then** PC:=PC+J+2 **else** PC:+2。

15. SP:−2；MM[SP]:=PC+3；PC:=KJ。

16. SP:−2；MM[SP]:=PC+1；PC:=8×S。

17. **if** cnd **then begin** SP:−2；MM[SP]:=PC+3；PC:=KJ；**end else** PC:+3。

18. SP:+2；PC:=MM[SP−2]。

19. **if** cnd **then begin** SP:+2；PC:=MM[SP−2] **end else** PC:+1。

20. SP:+2；PC:=MM[SP−2]。

21. *gp* q d（见§2.10）

Z80 卡诺图

表 3.14 Z80 卡诺图

rp	r \ r₂	B	D	M[HL]	H	L	A	E	C	C	E	A	L	H	M[HL]	D	B	r₁
BC	B	00 NULL	02 M[BC] :=A	06	04	05	07 'A:@L	03	01	41	43	47	45	44	46	42	40	B
DE	D	10 B:-1, JNZ+[J]	12 M[DE] :=A	16	14	15	17 AK:@L	13 ρ:+1	11 ρ:=K	51	53	57	55	54	56	52	50	D
SP	M[HL]	30 JNK +[J]	32 M[KJ] :=A	36	34	35	37 K:=1	33	31	71	73	77	75	74	76 HALT	72	70	M[HL]
HL	H	20 JNZ +[J]	22 MM[KJ] :=HL	26 r:=	24 r:+1	25 r:-1	27 DEC	23	21	61	63	67	65	64	66	62	60	H
HL	L	28 JZ+[J]	2A HL:= MM[KJ]	2E	2C	2D	2F A:#	2B	29	69	6B	6F	6D $r_1:=r_2$	6C	6E	6A	68	L
SP	A	38 JK+[J]	3A A:= M[KJ]	3E	3C	3D	3F K:#	3B ρ:-1	39 HL:+ρ	79	7B	7F	7D	7C	7E	7A	78	A
DE	E	18 J+[J]	1A A:= M[DE]	1E	1C	1D	1F AK:@R	1B	19	59	5B	5F	5D	5C	5E	5A	58	E
BC	C	08 AF:=: XX	0A A:= M[BC]	0E	0C	0D	0F A:@R	0B	09	49	4B	4F	4D	4C	4E	4A	48	C

op	B	D	M[HL]	H	L	A	E	C	C	E	A	L	H	M[HL]	D	B	cnd	ρ'
:++	88	8A	8E	8C	8D	8F	8B	89	C9 RET	CB (I₀)	CF (1)	CD JS[KJ]	CC	CE	CA	C8	Z	
:--	98	9A	9E	9C	9D	9F	9B	99	D9 BL:=: XX	DB A:= G[J]	DF (3)	DD (I₀)	DC	DE	DA	D8	K	
-	B8	BA	BE	BC	BD	BF	BB	B9	F9 SP:=HL	FB I:=1	FF (7)	FD (I₀)	FC	FE	FA	F8	N	
:#	A8	AA	AE	AC	AD A opr	AF	AB	A9	E9 J[HL]	EB HL:=: DE	EF (5)	ED (I₀)	EC JS cnd[KJ]	EE A op J	EA J cnd[KJ]	E8 R cnd	P	
:&	A0	A2	A6	A4	A5	A7	A3	A1	E1	E3 HL:=: ST	E7 (4) JSS₃	E5	E4	E6	E2	E0	NP	HL
:U	B0	B2	B6	B4	B5	B7	B3	B1	F1 ρ':=ST	F3 I:=0	F7 (6)	F5 ST:=ρ'	F4	F6	F2	F0	NN	AF
:-	90	92	96	94	95	97	93	91	D1	D3 G[J] :=A	D7 (2)	D5	D4	D6	D2	D0	NK	DE
:+	80	82	86	84	85	87	83	81	C1	C3 J[KJ]	C7 (0)	C5	C4	C6	C2	C0	NZ	BC

(s)

表 3.15 Z80 卡诺图(I_0=CB)

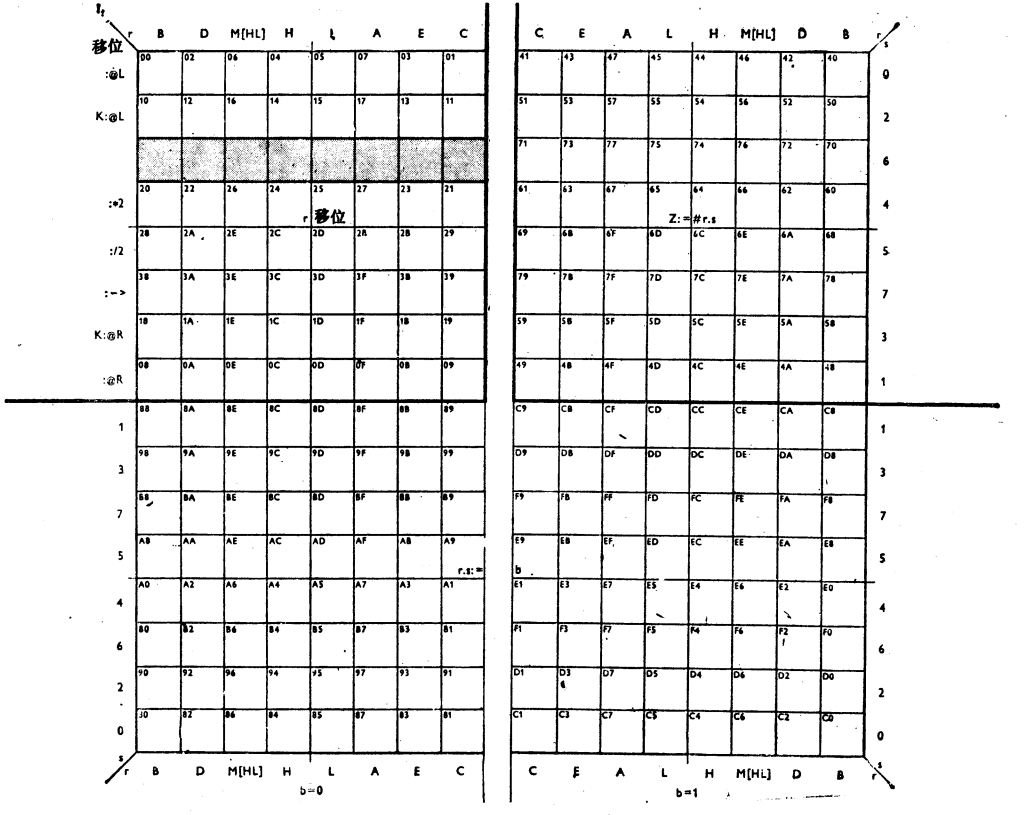


表 3.16　Z80 卡诺图(I_0=DD、I_0=FD)

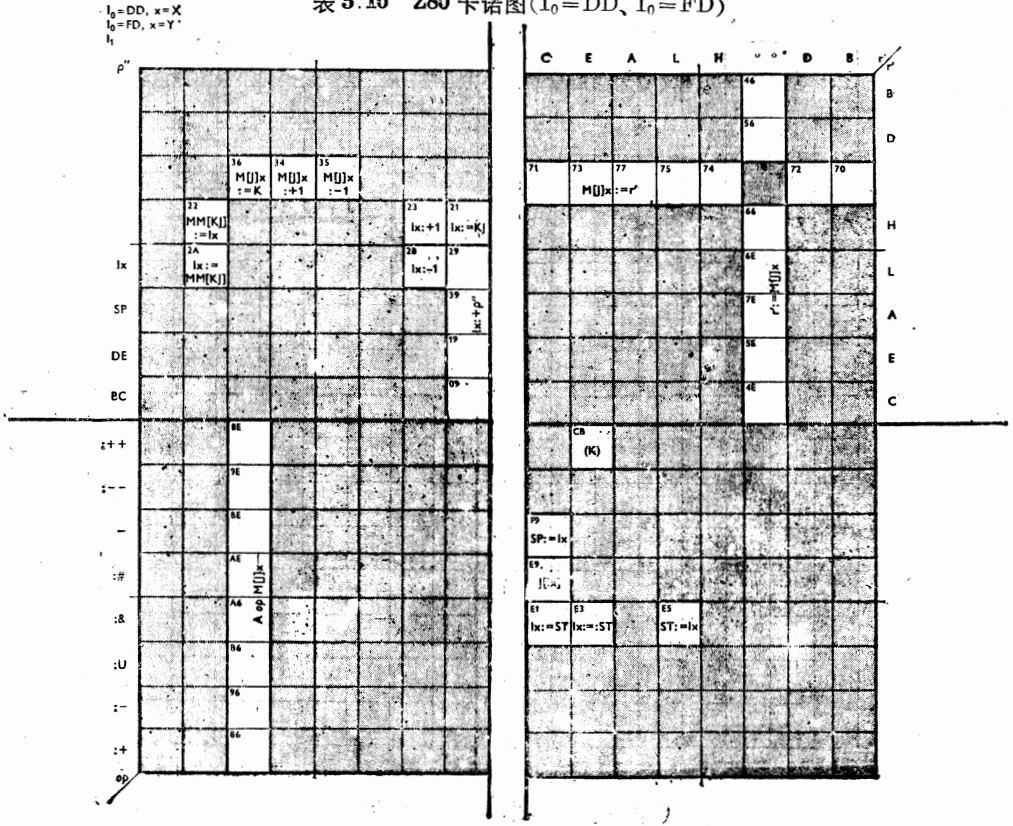


表 3.17　Z80 卡诺图(I_0=DD、I_0=FD 且 I_1=CB)

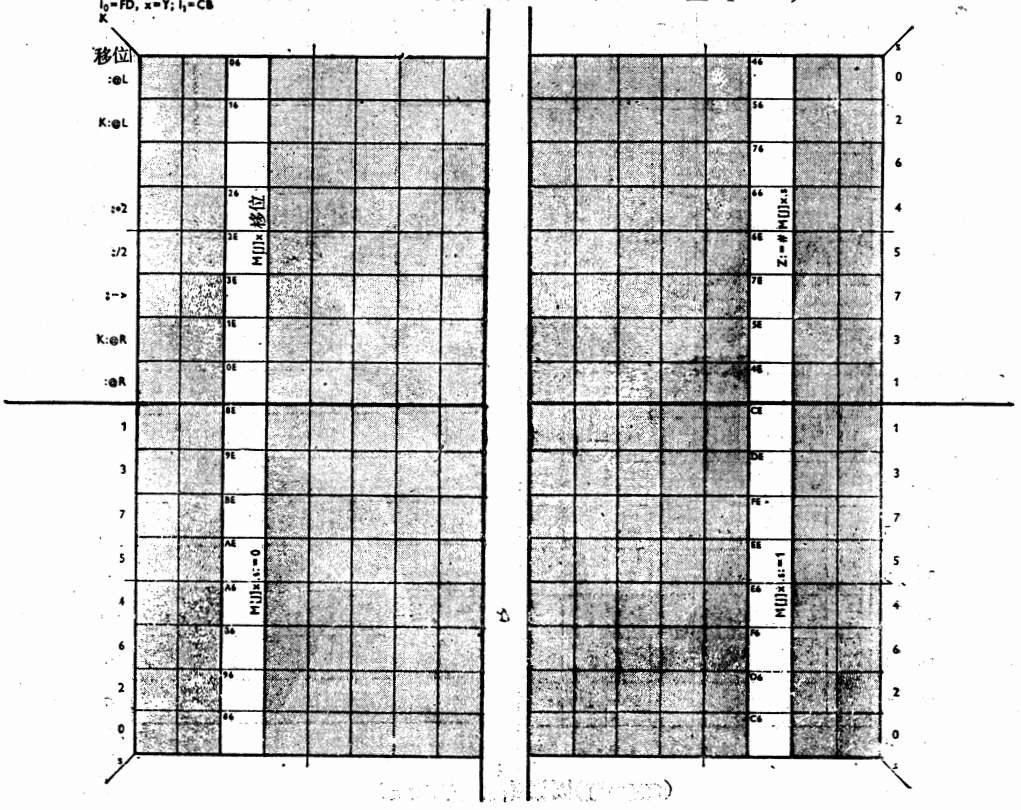
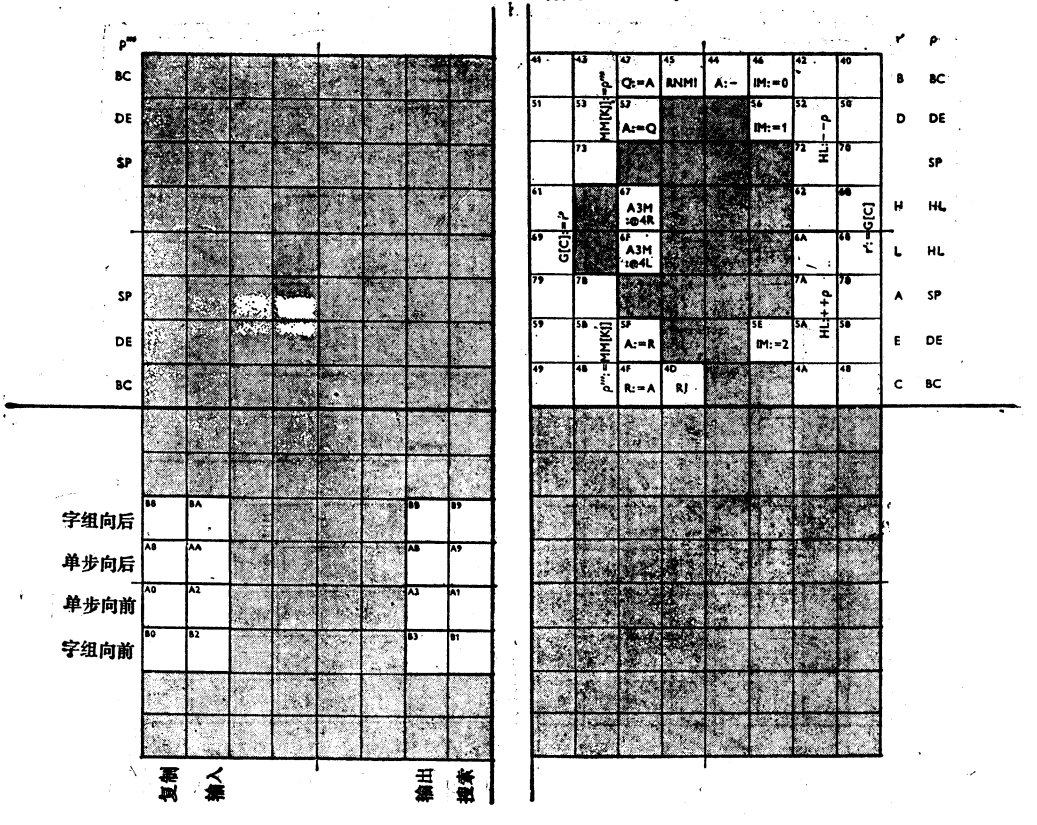


表 3.18　Z 卡诺图(I_0=ED)



Z80 指令表(按操作码的数值排列)

Z80 的指令系统与 8080 的相同,只是增加下述各条:

08		AF:=:XX
10	12	B:−1, JNZ+12
18	12	J+12
20	12	JNZ+12
28	12	JZ+12
30	12	JNK+12
38	12	JK+12
D9		BL:=:XX

CB 00～CB FF(包括这两条指令在内):

若第二个字节(I_1)表示成八进制(见§2.2),变成三个八进制数字 tsr, 则:

r 表示一个寄存器或 M[HL](见 r 的代换表),并且:

若 t=0 指令是移位指令 r shf, 其中 shf=s(见 shf 的代换表)。

若 t=1 指令是 Z:=#r.s, 其中 s 是一个位号(见 s 的代换表)。

若 t=2 指令是 r.s:=0, 其中 s 是一个位号(见 s 的代换表)。

若 t=3 指令是 r.s:=1, 其中 s 是一个位号(见 s 的代换表)。

(注: CB 30～CB 37 的操作尚未定义)。

§3.5 四个机种的指令汇总表

下面是一张 6800、8080、8085 与 Z80 的指令汇总表,采用本书中的指令书写形式按字母顺序排列。汇总时的顺序(扩充

表 3.19

DD	09			IX:+BC	DD	72	12		M12X:=D
DD	19			IX:+DE	DD	73	12		M12X:=E
DD	21	34	12	IX:=1234	DD	74	12		M12X:=H
DD	22	34	12	MM1234:=IX	DD	75	12		M12X:=L
DD	23			IX:+1	DD	77	12		M12X:=A
DD	29			IX:+IX	DD	7E	12		A:=M12X
DD	2A	34	12	IX:=MM1234	DD	86	12		A:+M12X
DD	2B			IX:-1	DD	8E	12		A:++M12X
DD	34	12		M12X:+1	DD	96	12		A:-M12X
DD	35	12		M12X:-1	DD	9E	12		A:--M12X
DD	36	12	34	M12X:=34	DD	A6	12		A:&M12X
DD	39			IX:+SP	DD	AE	12		A:#M12X
DD	46	12		B:=M12X	DD	B6	12		A:UM12X
DD	4E	12		C:=M12X	DD	BE	12		A-M12X
DD	56	12		D:=M12X	DD	CB	..	..	—见下面)
DD	5E	12		E:=M12X	DD	E1			IX:=ST
DD	66	12		H:=M12X	DD	E3			IX:=:ST
DD	6E	12		L:=M12X	DD	E5			ST:=IX
DD	70	12		M12X:=B	DD	E9			J[IX]
DD	71	12		M12X:=C	DD	F9			SP:=IX

DD	CB	12	06	M12X:@L	DD	CB	12	86	M12X.0:=0
DD	CB	12	0E	M12X:@R	DD	CB	12	8E	M12X.1:=0
DD	CB	12	16	M12XK:@L	DD	CB	12	96	M12X.2:=0
DD	CB	12	1E	M12XK:@R	DD	CB	12	9E	M12X.3:=0
DD	CB	12	26	M12X:*2	DD	CB	12	A6	M12X.4:=0
DD	CB	12	2E	M12X:/2	DD	CB	12	AE	M12X.5:=0
DD	CB	12	36		DD	CB	12	B6	M12X.6:=0
DD	CB	12	3E	M12X:->>	DD	CB	12	BE	M12X.7:=0
DD	CB	12	46	Z:=#M12X.0	DD	CB	12	C6	M12X.0:=1
DD	CB	12	4E	Z:=#M12X.1	DD	CB	12	CE	M12X.1:=1
DD	CB	12	56	Z:=#M12X.2	DD	CB	12	D6	M12X.2:=1
DD	CB	12	5E	Z:=#M12X.3	DD	CB	12	DE	M12X.3:=1
DD	CB	12	66	Z:=#M12X.4	DD	CB	12	E6	M12X.4:=1
DD	CB	12	6E	Z:=#M12X.5	DD	CB	12	EE	M12X.5:=1
DD	CB	12	76	Z:=#M12X.6	DD	CB	12	F6	M12X.6:=1
DD	CB	12	7E	Z:=#M12X.7	DD	CB	12	FE	M12X.7:=1

ED	40			B:=G[C]	ED	58			E:=G[C]
ED	41			G[C]:=B	ED	59			G[C]:=E
ED	42			HL:--BC	ED	5A			HL:++DE
ED	43	34	12	MM1234:=BC	ED	5B	34	12	DE:=MM1234
ED	44			A:-	ED	5E			IM:=2
ED	45			RNMI	ED	5F			A:=R
ED	46			IM:=0	ED	60			H:=G[C]
ED	47			Q:=A	ED	61			G[C]:=H
ED	48			C:=G[C]	ED	62			HL:--HL
ED	49			G[C]:=C	ED	67			A3M:@4R
ED	4A			HL:++BC	ED	68			L:=G[C]
ED	4B	34	12	BC:=MM1234	ED	69			G[C]:=L
ED	4D			RI	ED	6A			HL:++HL
ED	4F			R:=A	ED	6F			A3M:@4L
ED	50			D:=G[C]	ED	72			HL:--SP
ED	51			G[C]:=D	ED	73	34	12	MM1234:=SP
ED	52			HL:--DE	ED	78			A:=G[C]
ED	53	34	12	MM12341=DE	ED	79			G[C]:=A
ED	56			IM:=1	ED	7A			HL:++SP
ED	57			A:=Q	ED	7B	34	12	SP:=MM1234

ED	A0	复制单步向前	ED	B0	复制字组向前
ED	A1	搜索单步向前	ED	B1	搜索字组向前
ED	A2	输入单步向前	ED	B2	输入字组向前
ED	A3	输出单步向前	ED	B3	输出字组向前
ED	A8	复制单步向后	ED	B8	复制字组向后
ED	A9	搜索单步向后	ED	B9	搜索字组向后
ED	AA	输入单步向后	ED	BA	输入字组向后
ED	AB	输出单步向后	ED	BB	输出字组向后

FD…在上述指令表内 I_0=DD 的各条指令中用 FD 代替 DD，用 Y 代替 X。

的字母顺序)由下一章表 4.1 的 ASCII 码决定。

表中还给出了对应的机器码(一般用十六进制表示)。与本章前面的表格一样,用"12"与"1234"来代表单字节或双字节值。某些指令中的"7"表示一个典型的三位二进制值。在这种情况下,对应的机器码指令中的一个字节是用八进制来表示的,中间的八进制数字是"7"。可以用两个其它的八进制数字代替这些"7",这样就可用十六进制数表示这个八进制字节了。

注意:8080 的指令系统与 8085 的相同,只是 8085 增加了两条指令 A:=IM 与 IM:=A。

四种机器上每条指令的执行时间分别用四栏给出。所有的时间单位都是"时钟周期",各种系统的时钟周期可不同,其范围

表 3.20

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
A	4D			2				
A&12	85 12			2				
A&M1234	B5 12 34			4				
A&M12X	A5 12			5				
A&MZ12	95 12			3				
A−12	81 12	FE 12	FE 12	2	7	7	7	
A−A		BF	BF		4	4	4	
A−B	11	B8	B8	2	4	4	4	
A−C		B9	B9		4	4	4	2.4.7
A−D		BA	BA		4	4	4	
A−E		BB	BB		4	4	4	
A−H		BC	BC		4	4	4	
A−L		BD	BD		4	4	4	
A−M		BE	BE		7	7	7	
A−M1234	B1 12 34			4				
A−M12X	A1 12		DD BE 12	5			19	
A−M12Y			FD BE 12				19	
A−MZ12	91 12			3				
A.7:=0			CB 277				8	2.10
A.7:=1			CB 377				8	2.10
A3M:@4L			ED 6F				18	2.4.8
A3M:@4R			ED 67				18	2.4.8

表 3.21

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
A:#	43	2F	2F	2	4	4	4	
A:#12	88 12	EE 12	EE 12	2	7	7	7	
A:#A		AF	AF		4	4	4	
A:#B		A8	A8		4	4	4	
A:#C		A9	A9		4	4	4	
A:#D		AA	AA		4	4	4	
A:#E		AB	AB		4	4	4	
A:#H		AC	AC		4	4	4	
A:#L		AD	AD		4	4	4	
A:#M		AE	AE		7	7	7	
A:#M1234	B8 12 34			4				
A:#M12X	A8 12		DD AE 12	5			19	
A:#M12Y			FD AE 12				19	
A:#MZ12	98 12			3				2.4.6
A:&12	84 12	E6 12	E6 12	2	7	7	7	
A:&A		A7	A7		4	4	4	
A:&B		A0	A0		4	4	4	
A:&C		A1	A1		4	4	4	
A:&D		A2	A2		4	4	4	
A:&E		A3	A3		4	4	4	
A:&H		A4	A4		4	4	4	
A:&L		A5	A5		4	4	4	
A:&M		A6	A6		7	7	7	
A:&M1234	B4 12 34			4				
A:&M12X	A4 12		DD A6 12	5			19	
A:&M12Y			FD A6 12				19	
A:&MZ12	94 12			3				
A:*2	48		CB 27	2			8	2.4.8
A:++12	89 12	CE 12	CE 12	2	7	7	7	
A:++A		8F	8F		4	4	4	
A:++B		88	88		4	4	4	
A:++C		89	89		4	4	4	
A:++D		8A	8A		4	4	4	
A:++E		8B	8B		4	4	4	
A:++H		8C	8C		4	4	4	2.4.3.2
A:++L		8D	8D		4	4	4	
A:++M		8E	8E		7	7	7	
A:++M1234	B9 12 34			4				
A:++M12X	A9 12		DD 8E 12	5			19	
A:++M12Y			FD 8E 12				19	
A:++MZ12	99 12			3				

表 3.22

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
A:+1	4C	3C	3C	2	5	4	4	2.4.9
A:+12	8B 12	C6 12	C6 12	2	7	7	7	
A:+A		87	87		4	4	4	
A:+B	1B	80	80	2	4	4	4	
A:+C		81	81		4	4	4	
A:+D		82	82		4	4	4	
A:+E		83	83		4	4	4	
A:+H		84	84		4	4	4	2.4.3.1
A:+L		85	85		4	4	4	
A:+M		86	86		7	7	7	
A:+M1234	BB 12 34			4				
A:+M12X	AB 12		DD 86 12	5			19	
A:+M12Y			FD 86 12				19	
A:+MZ12	9B 12			3				
A:−	40		ED 44	2			8	2.4.4
A:−−12	82 12	DE 12	DE 12	2	7	7	7	
A:−−A		9F	9F		4	4	4	
A:−−B		98	98		4	4	4	
A:−−C		99	99		4	4	4	
A:−−D		9A	9A		4	4	4	2.4.5.2
A:−−E		9B	9B		4	4	4	
A:−−H		9C	9C		4	4	4	
A:−−L		9D	9D		4	4	4	
A:−−M		9E	9E		7	7	7	
A:−−M1234	B2 12 34			4				
A:−−M12X	A2 12		DD 9E 12	5			19	2.4.5.2
A:−−M12Y			FD 9E 12				19	
A:−−MZ12	92 12			3				
A:−1	4A	3D	3D	2	5	4	4	2.4.9
A:−12	80 12	D6 12	D6 12	2	7	7	7	2.4.5.1
A:−>	44		CB 3F	2			8	2.4.8
A:−A		97	97		4	4	4	
A:−B	10	90	90	2	4	4	4	
A:−C		91	91		4	4	4	
A:−D		92	92		4	4	4	
A:−E		93	93		4	4	4	
A:−H		94	94		4	4	4	2.4.5.1
A:−L		95	95		4	4	4	
A:−M		96	96		7	7	7	
A:−M1234	B0 12 34			4				
A:−M12X	A0 12		DD 96 12	5			19	

表 3.23

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
A:－M12Y			FD 96 12				19	2.4.5.1
A:－MZ12	90 12			3				2.4.5.1
A:/2	47		CB 2F	2			8	2.4.8
A:＝0	4F			2				
A:＝12	86 12	3E 12	3E 12	2	7	7	7	
A:＝A		7F	7F		5	4	4	
A:＝B	17	78	78	2	5	4	4	2.3.3
A:＝C	07	79	79	2	5	4	4	
A:＝D		7A	7A		5	4	4	
A:＝E		7B	7B		5	4	4	
A:＝G12		DB 12	DB 12		10	10	11	2.8
A:＝G[C]			ED 78				12	2.8
A:＝H		7C	7C		5	4	4	2.3.3
A:＝IM		20				4		2.9
A:＝L		7D	7D		5	4	4	
A:＝M		7E	7E		7	7	7	
A:＝M1234	B6 12 34	3A 34 12	3A 34 12	4	13	13	13	
A:＝M12X	A6 12		DD 7E 12	5			19	2.3.3
A:＝M12Y			FD 7E 12				19	
A:＝MZ12	96 12			3				
A:＝M[BC]		0A	0A		7	7	7	
A:＝M[DE]		1A	1A		7	7	7	
A:＝Q			ED 57				9	2.9
A:＝R			ED 5F				9	2.10
A:＝S	32			4				2.5
A:@L		07	07		4	4	4	2.4.8
A:@R		0F	0F		4	4	4	2.4.8
A:U12	8A 12	F6 12	F6 12	2	7	7	7	
A:UA		B7	B7		4	4	4	
A:UB		B0	B0		4	4	4	
A:UC		B1	B1		4	4	4	
A:UD		B2	B2		4	4	4	
A:UE		B3	B3		4	4	4	
A:UH		B4	B4		4	4	4	2.4.6
A:UL		B5	B5		4	4	4	
A:UM		B6	B6		7	7	7	
A:UM1234	BA 12 34			4				
A:UM12X	AA 12		DD B6 12	5			19	
A:UM12Y			FD B6 12				19	
A:UMZ12	9A 12			3				
AF:＝:XX			08				4	2.10

表 3.24

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	80	85	Z80	
AF:=ST		F1	F1		10	10	10	2.5
AK:@L	49	17	17	2	4	4	4	2.4.8
AK:@R	46	1F	1F	2	4	4	4	2.4.8
B	5D			2				
B&12	C5 12			2				
B&M1234	F5 12 34			4				
B&M12X	E5 12			5				
B&MZ12	D5 12			3				2.4.7
B−12	C1 12			2				
B−M1234	F1 12 34			4				
B−M12X	E1 12			5				
B−MZ12	D1 12			3				
B.7:=0			CB 270				8	2.10
B.7:=1			CB 370				8	2.10
B:≠	53			2				2.4.4
B:≠12	C8 12			2				
B:≠M1234	F8 12 34			4				
B:≠M12X	E8 12			5				
B:≠MZ12	D8 12			3				2.4.6
B:&12	C4 12			2				
B:&M1234	F4 12 34			4				
B:&M12X	E4 12			5				
B:&MZ12	D4 12			3				
B:•2	58		CB 20	2			8	2.4.8
B:++12	C9 12			2				
B:++M1234	F9 12 34			4				2.4.3.2
B:++M12X	E9 12			5				
B:++MZ12	D9 12			3				
B:+1	5C	04	04	2	5	4	4	2.4.9
B:+12	CB 12			2				2.4.3.1
B:+M1234	FB 12 34			4				
B:+M12X	EB 12			5				2.4.3.1
B:+MZ12	DB 12			3				
B:−	50			2				2.4.4
B:−−12	C2 12			2				
B:−−M1234	F2 12 34			4				2.4.5.2
B:−−M12X	E2 12			5				
B:−−MZ12	D2 12			3				
B:−1	5A	05	05	2	5	4	4	2.4.9
B:−1,JNZ+12			10 12				8/13	2.10
B:−12	C0 12			2				2.4.5.1

表 3.25

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
B:－M1234	F0 12 34			4				↑
B:－M12X	E0 12			5				2.4.5.1
B:－MZ12	D0 12			3				↓
B:－>	54		CB 38	2			8	2.4.8
B:/2	57		CB 28	2			8	2.4.8
B:＝0	5F			2				↑
B:＝12	C6 12	06 12	06 12	2	7	7	7	
B:＝A	16	47	47	2	5	4	4	
B:＝B		40	40		5	4	4	2.3
B:＝C		41	41		5	4	4	
B:＝D		42	42		5	4	4	
B:＝E		43	43		5	4	4	↓
B:＝G[C]			ED 40				12	2.8
B:＝H		44	44		5	4	4	↑
B:＝L		45	45		5	4	4	
B:＝M		46	46		7	7	7	2.3
B:＝M1234	F6 12 34			4				↓
B:＝M12X	E6 12		DD 46 12	5			19	
B:＝M12Y			FD 46 12				19	2.3
B:＝MZ12	D6 12			3				2.3
B:＝S	33			4				2.5
B:@L			CB 00				8	2.4.8
B:@R			CB 08				8	2.4.8
B:U12	CA 12			2				↑
B:UM1234	FA 12 34			4				2.4.6
B:UM12X	EA 12			5				
B:UMZ12	DA 12			3				↓
BC:+1		03	03		5	6	6	2.4.9
BC:－1		0B	0B		5	6	6	2.4.9
BC:＝1234		01 34 12	01 34 12		10	10	10	2.3
BC:＝MM1234			ED 4B 34 12				20	2.3
BC:＝ST		C1	C1		10	10	10	2.5
BK:@L	59		CB 10	2			8	2.4.8
BK:@R	56		CB 18	2			8	2.4.8
BL:＝:XX			D9				4	2.10
C.7:＝0			CB 271				8	2.10
C.7:＝1			CB 371				8	2.10
C:*2			CB 21				8	2.4.8
C:+1		0C	0C		5	4	4	2.4.9
C:－1		0D	0D		5	4	4	2.4.9
C:－>			CB 39				8	2.4.8

表 3.26

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
C:/2			CB 29				8	2.4.8
C:=12		0E 12	0E 12		7	7	7	
C:=A	06	4F	4F	2	5	4	4	
C:=B		48	48		5	4	4	2.3
C:=C		49	49		5	4	4	
C:=D		4A	4A		5	4	4	
C:=E		4B	4B		5	4	4	2.3
C:=G[C]			ED 48				12	2.8
C:=H		4C	4C		5	4	4	
C:=L		4D	4D		5	4	4	
C:=M		4E	4E		7	7	7	2.3
C:=M12X			DD 4E 12				19	
C:=M12Y			FD 4E 12				19	
C:@L			CB 01				8	2.4.8
C:@R			CB 09				8	2.4.8
CBD			ED B8				*	2.10
CBU			ED B0				*	2.10
CK:@L			CB 11				8	2.4.8
CK:@R			CB 19				8	2.4.8
CSD			ED A8				16	2.10
CSU			ED A0				16	2.10
D.7:=0			CB 272				8	2.10
D.7:=1			CB 372				8	2.10
D:*2			CB 22				8	2.4.8
D:+1		14	14		5	4	4	2.4.9
D:-1		15	15		5	4	4	2.4.9
D:->			CB 3A				8	2.4.8
D:/2			CB 2A				8	2.4.8
D:=12		16 12	16 12		7	7	7	
D:=A		57	57		5	4	4	
D:=B		50	50		5	4	4	2.3
D:=C		51	51		5	4	4	
D:=D		52	52		5	4	4	
D:=E		53	53		5	4	4	
D:=G[C]			ED 50				12	2.8
D:=H		54	54		5	4	4	
D:=L		55	55		5	4	4	
D:=M		56	56		7	7	7	2.3
D:=M12X			DD 56 12				19	
D:=M12Y			FD 56 12				19	
D:@L			CB 02				8	2.4.8

表 3.27

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
D:@R			CB 0A				8	2.4.8
DE:+1		13	13		5	6	6	2.4.9
DE:−1		1B	1B		5	6	6	2.4.9
DE:=1234		11 34 12	11 34 12		10	10	10	2.3
DE:=MM1234			ED 5B 34 12				20	2.3
DE:=ST		D1	D1		10	10	10	2.5
DEC	19	27	27	2	4	4	4	2.4.9
DK:@L			CB 12				8	2.4.8
DK:@R			CB 1A				8	2.4.8
E.7:=0			CB 273				8	2.10
E.7:=1			CB 373				8	2.10
E:*2			CB 23				8	2.4.8
E:+1		1C	1C		5	4	4	2.4.9
E:−1		1D	1D		5	4	4	2.4.9
E:−>			CB 3B				8	2.4.8
E:/2			CB 2B				8	2.4.8
E:=12		1E 12	1E 12		7	7	7	
E:=A		5F	5F		5	4	4	
E:=B		58	58		5	4	4	2.3
E:=C		59	59		5	4	4	
E:=D		5A	5A		5	4	4	
E:=E		5B	5B		5	4	4	
E:=G[C]			ED 58				12	2.8
E:=H		5C	5C		5	4	4	
E:=L		5D	5D		5	4	4	
E:=M		5E	5E		7	7	7	2.3
E:=M12X			DD 5E 12				19	
E:=M12Y			FD 5E 12				19	
E:@L			CB 03				8	
E:@R			CB 0B				8	2.4.8
EK:@L			CB 13				8	
EK:@R			CB 1B				8	
G12:=A		D3 12	D3 12		10	10	11	
G[C]:=A			ED 79				12	
G[C]:=B			ED 41				12	
G[C]:=C			ED 49				12	2.8
G[C]:=D			ED 51				12	
G[C]:=E			ED 59				12	
G[C]:=H			ED 61				12	
G[C]:=L			ED 69				12	
H.7:=0			CR 274				8	2.10

表 3.28

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
H.7:=1			CB 374				8	2.10
H:*2			CB 24				8	2.4.8
H:+1		24	24		5	4	4	2.4.9
H:−1		25	25		5	4	4	2.4.9
H:−>			CB 3C				8	2.4.8
H:/2			CB 2C				8	2.4.8
H:=12		26 12	26 12		7	7	7	
H:=A		67	67		5	4	4	
H:=B		60	60		5	4	4	2.3
H:=C		61	61		5	4	4	
H:=D		62	62		5	4	4	
H:=E		63	63		5	4	4	2.3
H:=G[C]			ED 60				12	2.8
H:=H		64	64		5	4	4	
H:=L		65	65		5	4	4	
H:=M		66	66		7	7	7	2.3
H:=M12X			DD 66 12				19	
H:=M12Y			FD 66 12				19	
H:@L			CB 04				8	2.4.8
H:@R			CB 0C				8	2.4.8
HALT		76	76		7	5	4	2.9
HK:@L			CB 14				8	2.4.8
HK:@R			CB 1C				8	2.4.8
HL:++BC			ED 4A				15	
HL:++DE			ED 5A				15	2.4.3.2
HL:++HL			ED 6A				15	
HL:++SP			ED 7A				15	
HL:+1		23	23		5	6	6	2.4.9
HL:+BC		09	09		10	10	11	
HL:+DE		19	19		10	10	11	2.4.3.2
HL:+HL		29	29		10	10	11	
HL:+SP		39	39		10	10	11	
HL:−−BC			ED 42				15	
HL:−−DE			ED 52				15	2.4.5.2
HL:−−HL			ED 62				15	
HL:−−SP			ED 72				15	
HL:−1		2B	2B		5	6	6	2.4.9
HL:=:DE		EB	EB		4	4	4	2.3.4
HL:=:ST		E3	E3		18	16	19	2.3.4,2.5
HL:=1234		21 34 12	21 34 12		10	10	10	2.3
HL:=MM1234		2A 34 12	2A 34 12		16	16	16	2.3

表 3.29

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
HL:=ST		E1	E1		10	10	10	2.5
I:=0	0E	F3	F3	2	4	4	4	2.9
I:=1	0F	FB	FB	2	4	4	4	2.9
IBD			ED BA				*	2.10
IBU			ED B2				*	2.10
IM:=0			ED 46				8	↑
IM:=1			ED 56				8	2.9
IM:=2			ED 5E				8	↓
IM:=A		30				4		
ISD			ED AA				16	2.10
ISU			ED A2				16	2.10
IX-1234	8C 12 34			3				↑
IX-MM1234	BC 12 34			5				2.4.7
IX-MM12X	AC 12			6				↓
IX-MMZ12	9C 12			4				
IX:+1	08		DD 23	4			10	2.4.9
IX:+BC			DD 09				15	↑
IX:+DE			DD 19				15	2.4.3.2
IX:+IX			DD 29				15	
IX:+SP			DD 39				15	↓
IX:-1	09		DD 2B	4			10	2.4.9
IX:=:ST			DD E3				23	2.3.4, 2.5
IX:=1234	CE 12 34		DD 21 34 12	3			14	↑
IX:=MM1234	FE 12 34		DD 2A 34 12	5			20	2.3
IX:=MM12X	EE 12			6				↓
IX:=MMZ12	DE 12			4				
IX:=SP+1	30			4				2.3, 2.5
IX:=ST			DD E1				14	2.5
IY:+1			FD 23				10	2.4.9
IY:+BC			FD 09				15	↑
IY:+DE			FD 19				15	2.4.3.2
IY:+IY			FD 29				15	
IY:+SP			FD 39				15	↓
IY:-1			FD 2B				10	2.4.9
IY:=:ST			FD E3				23	2.3.4, 2.5
IY:=1234			FD 21 34 12				14	2.3
IY:=MM1234			FD 2A 34 12				20	2.3
IY:=ST			FD E1				14	2.5
J+12	20 12		18 12	4			12	↑
J1234	7E 12 34	C3 34 12	C3 34 12	3	10	10	10	2.6
J12X	6E 12			4				↓

表 3.30

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80 80	80 85	Z 80	
JK+12	25 12		38 12	4			7/12	
JK1234		DA 34 12	DA 34 12		10	7/10	10	
JKZ+12	23 12			4				
JN+12	2B 12			4				
JN1234		FA 34 12	FA 34 12		10	7/10	10	
JNK+12	24 12		30 12	4			7/12	
JNK1234		D2 34 12	D2 34 12		10	7/10	10	
JNKZ+12	22 12			4				
JNN+12	2A 12			4				2.6
JNN1234		F2 34 12	F2 34 12		10	7/10	10	
JNP1234		E2 34 12	E2 34 12		10	7/10	10	
JNV+12	28 12			4				
JNY+12	2C 12			4				
JNYZ+12	2E 12			4				
JNZ+12	26 12		20 12	4			7/12	
JNZ1234		C2 34 12	C2 34 12		10	7/10	10	
JP1234		EA 34 12	EA 34 12		10	7/10	10	
JS+12	8D 12			8				
JS1234	BD 12 34	CD 34 12	CD 34 12	9	17	18	17	
JS12X	AD 12			8				
JSK1234		DC 34 12	DC 34 12		11/17	9/18	10/17	
JSN1234		FC 34 12	FC 34 12		11/17	9/18	10/17	
JSNK1234		D4 34 12	D4 34 12		11/17	9/18	10/17	
JSNN1234		F4 34 12	F4 34 12		11/17	9/18	10/17	2.7
JSNP1234		E4 34 12	E4 34 12		11/17	9/18	10/17	
JSNZ1234		C4 34 12	C4 34 12		11/17	9/18	10/17	
JSP1234		EC 34 12	EC 34 12		11/17	9/18	10/17	
JSS7		377	377		11	12	11	
JSZ1234		CC 34 12	CC 34 12		11/17	9/18	10/17	
JV+12	29 12			4				
JY+12	2D 12			4				
JYZ+12	2F 12			4				
JZ+12	27 12		28 12	4			7/12	2.6
JZ1234		CA 34 12	CA 34 12		10	7/10	10	
J[HL]		E9	E9		5	6	4	
J[IX]			DD E9				8	
J[IY]			FD E9				8	
K:≠		3F	3F		4	4	4	
K:=0	0C			2				2.3.7
K:=1	0D	37	37	2	4	4	4	
L.7:=0			CB 275				8	2.10

表 3.31

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
L.7:=1			CB 375				8	2.10
L:•2			CB 25				8	2.4.8
L:+1		2C	2C		5	4	4	2.4.9
L:−1		2D	2D		5	4	4	2.4.9
L:−>			CB 3D				8	2.4.8
L:/2			CB 2D				8	2.4.8
L:=12		2E 12	2E 12		7	7	7	
L:=A		6F	6F		5	4	4	
L:=B		68	68		5	4	4	2.3
L:=C		69	69		5	4	4	
L:=D		6A	6A		5	4	4	
L:=E		6B	6B		5	4	4	
L:=G[C]			ED 68				12	2.8
L:=H		6C	6C		5	4	4	
L:=L		6D	6D		5	4	4	
L:=M		6E	6E		7	7	7	2.3
L:=M12X			DD 6E 12				19	
L:=M12Y			FD 6E 12				19	
L:@L			CB 05				8	
L:@R			CB 0D				8	2.4.8
LK:@L			CB 15				8	
LK:@R			CB 1D				8	
M1234	7D 12 34			6				2.4.7
M1234:#	73 12 34			6				2.4.4
M1234:•2	78 12 34			6				2.4.8
M1234:+1	7C 12 34			6				2.4.9
M1234:−	70 12 34			6				2.4.4
M1234:−1	7A 12 34			6				2.4.9
M1234:−>	74 12 34			6				2.4.8
M1234:/2	77 12 34			6				2.4.8
M1234:=0	7F 12 34			6				
M1234:=A	B7 12 34	32 34 12	32 34 12	5	10	10	13	2.3
M1234:=B	F7 12 34			5				
M1234K:@L	79 12 34			6				2.4.8
M1234K:@R	76 12 34			6				2.4.8
M12X	6D 12			7				2.4.7
M12X.7:=0			DD CB 12 276				23	2.10
M12X.7:=1			DD CB 12 376				23	2.10
M12X:#	63 12			7				2.4.4
M12X:•2	68 12		DD CB 12 26	7			23	2.4.8
M12X:+1	6C 12		DD 34 12	7			23	2.4.9

表 3.32

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
M12X:−	60 12			7				2.4.4
M12X:−1	6A 12		DD 35 12	7			23	2.4.9
M12X:− >	64 12		DD CB 12 3E	7			23	2.4.8
M12X:/2	67 12		DD CB 12 2E	7			23	2.4.8
M12X:＝0	6F 12			7				
M12X:＝34			DD 36 12 34				19	
M12X:＝A	A7 12		DD 77 12	6			19	
M12X:＝B	E7 12		DD 70 12	6			19	
M12X:＝C			DD 71 12				19	2.3
M12X:＝D			DD 72 12				19	
M12X:＝E			DD 73 12				19	
M12X:＝H			DD 74 12				19	
M12X:＝L			DD 75 12				19	
M12X:@L			DD CB 12 06				23	
M12X:@R			DD CB 12 0E				23	2.4.8
M12XK:@L	69 12		DD CB 12 16				23	
M12XK:@R	66 12		DD CB 12 1E				23	
M12Y.7:＝0			FD CB 12 276				23	2.10
M12Y.7:＝1			FD CB 12 376				23	2.10
M12Y:•2			FD CB 12 26				23	2.4.8
M12Y:+1			FD 34 12				23	2.4.9
M12Y:−1			FD 35 12				23	2.4.9
M12Y:− >			FD CB 12 3E				23	2.4.8
M12Y:/2			FD CB 12 2E				23	2.4.8
M12Y:＝34			FD 36 12 34				19	
M12Y:＝A			FD 77 12				19	
M12Y:＝B			FD 70 12				19	
M12Y:＝C			FD 71 12				19	2.3
M12Y:＝D			FD 72 12				19	
M12Y:＝E			FD 73 12				19	
M12Y:＝H			FD 74 12				19	
M12Y:＝L			FD 75 12				19	
M12Y:@L			FD CB 12 06				23	
M12Y:@R			FD CB 12 0E				23	2.4.8
M12YK:@L			FD CB 12 16				23	
M12YK:@R			FD CB 12 1E				23	
M.7:＝0			CB 276				15	2.10
M.7:＝1			CB 376				15	2.10
M:•2			CB 26				15	2.10
M:+1		34	34		10	10	11	2.4.9
M:−1		35	35		10	10	11	2.4.9

表 3.33

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	6800	8080	8085	Z80	
M:−>			CB 3E				15	2.4.8
M:/2			CB 2E				15	2.4.8
M:=12		36 12	36 12		7	7	10	
M:=A		77	77		7	7	7	
M:=B		70	70		7	7	7	
M:=C		71	71		7	7	7	2.3
M:=D		72	72		7	7	7	
M:=E		73	73		7	7	7	
M:=H		74	74		7	7	7	
M:=L		75	75		7	7	7	
M:@L			CB 06				15	
M:@R			CB 0E				15	2.4.8
MK:@L			CB 16				15	
MK:@R			CB 1E				15	
MM1234:=BC			ED 43 34 12				20	
MM1234:=DE			ED 53 34 12				20	
MM1234:=HL		22 34 12	22 34 12		16	16	16	
MM1234:=IX	FF 12 34		DD 22 34 12	6			20	
MM1234:=IY			FD 22 34 12				20	
MM1234:=SP	BF 12 34		ED 73 34 12	6			20	
MM12X:=IX	EF 12			7				
MM12X:=SP	AF 12			7				
MMZ12:=IX	DF 12			5				2.3
MMZ12:=SP	9F 12			5				
MZ12:=A	97 12			4				
MZ12:=B	D7 12			4				
M[BC]:=A		02	02		7	7	7	
M[DE]:=A		12	12		7	7	7	
NULL	01	00	00	2	4	4	4	
OBD			ED BB				*	
OBU			ED B3				*	2.10
OSD			ED AB				16	
OSU			ED A3				16	
Q:=A			ED 47				9	2.9
R:=A			ED 4F				9	2.10
RET	39	C9	C9	5	10	10	10	2.7
RI	3B		ED 4D	10			14	2.9
RK		D8	D8		5/11	6/12	5/11	2.7
RN		F8	F8		5/11	6/12	5/11	2.7
RNK		D0	D0		5/11	6/12	5/11	2.7
RNMI			ED 45				14	2.9

表 3.34

指令	机器码			时钟周期				索引
	6800	8080/8085	Z80	68 00	80	80 85	Z 80	
RNN		F0	F0		5/11	6/12	5/11	
RNP		E0	E0		5/11	6/12	5/11	
RNZ		C0	C0		5/11	6/12	5/11	2.7
RP		E8	E8		5/11	6/12	5/11	
RZ		C8	C8		5/11	6/12	5/11	
S:=A	36			4				2.5
S:=B	37			4				2.5
SBD			ED B9				*	2.10
SBU			ED B1				*	2.10
SI	3F			12				2.7
SP:+1	31	33	33	4	5	6	6	2.4.9
SP:−1	34	3B	3B	4	5	6	6	2.4.9
SP:='1234	8E 12 34	31 34 12	31 34 12	3	10	10	10	
SP:=HL		F9	F9		5	6	6	
SP:=IX			DD F9				10	
SP:=IX−1	35			4				2.3, 2.5
SP:=IY			FD F9				10	
SP:=MM1234	BE 12 34		ED 7B 34 12	5			20	
SP:=MM12X	AE 12			6				
SP:=MMZ12	9E 12			4				
SSD			ED A9				16	2.10
SSU			ED A1				16	2.10
ST:=AF		F5	F5		11	12	11	
ST:=BC		C5	C5		11	12	11	
ST:=DE		D5	D5		11	12	11	2.5
ST:=HL		E5	E5		11	12	11	
ST:=IX			DD E5				15	
ST:=IY			FD E5				15	2.5
V:=0	0A			2				2.3.7
V:=1	0B			2				2.3.7
WI	3E			9				2.9
Z:=≠A.7			CB 177				8	
Z:=≠B.7			CB 170				8	
Z:=≠C.7			CB 171				8	
Z:=≠D.7			CB 172				8	
Z:=≠E.7			CB 173				8	2.10
Z:=≠H.7			CB 174				8	
Z:=≠L.7			CB 175				8	
Z:=≠M.7			CB 176				12	
Z:=≠M12X.7			DD CB 12 176				20	
Z:=≠M12Y.7			FD CB 12 176				20	

大致如下：

6800：1.0 μs～10 μs

8080/8085：0.32 μs～2.0 μs

Z80：0.25 μs～2.2 μs

在一些特殊情况下给出了两个时间，此时执行时间取决于条件。较长的时间对应于满足条件的情况，此时要执行整个操作。Z80“字组处理”指令（表格中用 * 表示）的执行时间在任何情况下都是 $21n-5$ 个时钟周期，其中 n 是执行的步数（$n \geqslant 1$）。

最后一栏给出了第二章讨论各有关类型指令的节号。

第四章　算术运算的程序设计

§4.1　可供实用的最小系统和软件

程序设计这种注重实践的工作，一直要到程序能在实际的机器上用实际的数据正确运行时设计才算结束。对程序进行测试和修改(调试)是程序员的一部分不可缺少的任务。所以，重要的是读者必须有机会使用适当的微处理机系统。

虽然有些初学者已能熟练地使用十六键的计算器键盘以及一组七段数字显示器，但是如果可能的话，初学者使用的系统应该有一个适当的控制台，最好配有 ASCII 码 (见 §4.2) 的键盘和显示器 (打印机或目视显示部件)。随机存取存贮器 (RAM) 要作为一个实用的暂存存贮器的话，最少需要 256 个字节的容量；如果有一个存有基本的监督程序的小容量可编程序只读存贮器(PROM)，则能省却许多麻烦；因此也几乎是必备的。

一开始最好不要用配备昂贵软件的大型系统从事程序设计。毫无疑问，与大型计算机的用户相比，微处理机的用户尤其必须了解硬件，而且首先需熟悉机器和机器码。这样，用户就能评估后阶段可供使用的软件。如果软件质量较好，就能提高用户的工作效率；而软件的质量较差，就会使用户遇到不必要的困难，造成意想不到的损失。

在几个初步的程序设计例子中，只作如下假定：

(i) 可用的 RAM 地址为 1000～10FF(256 个字节)。

(ii) 有一个存有简单的监督程序的 PROM(也许是 ROM)。可借助于系统的复位信号(RESET)进入这个监督程序(即开始

执行监督程序)。这个程序至少应该有下述功能:

(a) 读入一系列字节(以成对的十六进制数字进行输入),并将其存入RAM中连续的单元内;

(b) 读取任意的一个地址(四个十六进制数字),并转至该地址。例如,可以用这个操作启动执行RAM中的一个程序;

(c) 显示处理机寄存器的内容;

(d) 显示存贮单元的内容,以及(若为RAM单元的话)改变这些单元的内容。

(iii) PROM中有两个基本的子程序(是为监督程序配备的,但是其它程序也可使用):

(a) "输入一个字符"——用于从键盘读入一个字符,并将其"ASCII值"(见§4.2)存入寄存器A;

(b) "输出一个字符"——用于显示一个字符,该字符的ASCII值已存在寄存器A中(见§2.8)。

(iv) 控制台能读入和显示ASCII码字符。

若现有系统的有关细节与上述规格不同,则下述例子中的程序可能需要进行修改。例如,系统中RAM的地址可能不是1000,而是其它的地址;输入-输出子程序可能使用寄存器A之外的其它寄存器。如果没有ASCII码的控制台,就会降低装入程序、输入数据和输出结果的速度。如果不考虑输入输出的速度,则上述控制台还是切实可行的。

为了能获得妥善的解决办法,必须考虑到输入-输出子程序的一些细节。"输入一个字符"子程序存在PROM中,入口地址为0231。该子程序将一个字符读入寄存器A,不使用其它的寄存器。"输出一个字符"子程序存在PROM中,入口地址为0312。该子程序输出寄存器A中的字符,不使用其它的寄存器。

在下述例子中，调用“输入一个字符”子程序将被写作“A:=inchar”；调用“输出一个字符”子程序写作“outchar(A)”。

有一点是用惯磁芯存贮器的程序员应特别注意的。虽然微处理机系统用的程序最初是写在RAM中，并在RAM中进行研制的，但是其最后形式多半是编入PROM中去。由于PROM是“只读”型的存贮装置，因此不可能象在磁芯存贮器中那样对指令进行修改。“暂存区”必须在RAM中，所以在这些例子中必须妥善安排，使暂时存贮所需的单元与那些用于保存指令的单元分开。

§4.2 ASCII码

表4.1是七位的ASCII码（美国信息交换标准代码）。若将ASCII字符表示成八位的数值，则对于附加的一位常用下列两种约定：

（i）最高位为零；

（ii）最高位为奇偶位。

在现有的控制台设备上，这两种方法都可能使用。假定这里介绍的子程序能以适当的表示方法与控制台设备进行通信，而送入寄存器A以及从A中取出的字符字节都是最高位为零的字符字节。

在这些例子中只需使用20～5F之间的符号（字母、数字和其它打印字符）、0A（换行）、0D（回车）以及1B（换码）。

§4.3 读取十进制数字

调用了“A:=inchar”子程序以后，该子程序将执行到对键盘上的一个键进行操作为止。若按下的是“5”键，则寄存器A中的结果将是16进制35(00110101)。为了将这个结果转换成

表 4.1 七位的 ASCII 码

	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111
0 ·000	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1 ·001	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	VS
2 ·010	Spce	!	"	#	$	%	&	'	(	)	*	+	,	−	.	/
3 ·011	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4 ·100	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5 ·101	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	←
6 ·110	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7 ·111	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

5的二进制值(00000101)，必须减去十六进制的30(00110000)。因此，若按下的键是标有"0"～"9"的键中的一个，则下述指令串将给出00000000～00001001范围内的二进制值：

```
A:=inchar
A:-30
```

但是若按下的是其它的键，则寄存器A中就是一个无意义的值。因此，当程序希望输入一个十进制数字时有一个较好的办法，就是检查子程序输入的数值是否在30～39范围内。若在此范围内，则一切正常；若不在此范围内，则程序必须转入"出错例行程序"——其功能也许是打印(显示)一个问号，然后尝试第二次读入。

下述指令串将读取、检查一个十进制数字，并将其转换成二进制值：

```
A:=inchar
A-30
JN(至出错例行程序)
A-3A
JNN(至出错例行程序)
A:-30
```

这串指令将作为例1中程序的核心。

例1 编写一个从1000号单元开始的程序，该程序读取一串十进制数字并将其二进制值存入从1080开始的连续单元内。当读入一个非数字字符时该程序结束，并将这个字符的ASCII值存在寄存器A中。

首先请注意，因"出错"而转出"读取、检查和转换"部分时，寄存器A仍存有刚读入的字符的ASCII值。所以程序的最后部分不会有什么困难。为了将数值存入连续的存贮单元，必须

6800	8080, 8085, Z80
IX:=1080	HL:=1080
→A:=inchar	→A:=inchar
A−30	A−30
—JN	—JN
A−3A	A−3A
—JNN	—JNN
A:−30	A:−30
MOOX:=A	M:=A (即: M[HL]:=A)
IX:+1	HL:+1
—J	—J
→(继续)	→(继续)

采用变址寻址或使用计数寄存器的间接寻址。变址值或计数值必须预置成起始地址值1080。在读取、检查和转换了每个数字以后，必须将其存贮起来，并在读取下一个字符之前将变址寄存器或计数寄存器的内容加1。

自此以后，6800用的程序与8080、8085和Z80用的程序就开始不同了。6800可用变址寄存器IX；而其它微处理机则可用HL寄存器对，这是一个有"加1"指令的寻址寄存器。

在上述程序的说明中，用箭头表示转移指令转向何处。下面说明另一种约定。在这种约定中，转移指令所转向的那些指令的左边加一个标号。在每条转移指令中都用标号代替了目的单元的"绝对"地址。标号的形式和引用这些标号的方式必须是第八章介绍的汇编程序所能够接受的。目前，标号将被看作表示尚未确定存贮单元的指令的标记。除了一、二个例外情况外，在后面的例子中都将采用标号这种约定，而不画箭头。程序员已由此而分成两部分：一部分认为在这类说明中采用箭头有用，另一部分则认为箭头说明方法不足取。属于前一部分的读者将发现画上这些箭头比后一部分读者擦掉这些箭头容易。

	6800		8080, 8085, Z80	注释
	IX:=1080		HL:=1080	设置起始地址
01:	A:=inchar	01:	A:=inchar	读取 ASCII 字符
	A−30		A−30	若非数字，则转至出口
	JN(L02)		JN(L02)	
	A−3A		A−3A	
	JNN(L02)		JNN(L02)	
	A:−30		A:−30	从 ASCII 码转换成二进制
	MOOX:=A		M:=A	存入设定的地址
	IX:+1		HL:+1	地址加一
	J(L01)		J(L01)	返回，读取下一个字符
02:		02:		

现在再来讨论这个例子，下一步工作是用第三章中的表格把程序变换成机器码(十六进制)。此时必须知道输入子程序的起始地址(入口)。在 6800 中，由于采用了相对转移指令，所以必须计算出指令之间的字节数；在其它机器中，由于采用了直接(绝对)转移指令，所以必须知道每条指令存在何处。这两种情况下的全部结果如图 4.1 所示。

在 6800 的程序中，最巧妙的是算出放在 1014 号单元中的值。1013 和 1014 中的指令是一条向后转移指令，所以所需的数值是个负数。"正常"的顺序是 1015，但是程序要求转至 1003，即向后 12 (十六进制! 十进制中为 18) 个字节。由表 2.1 可得：−18(十进制)相当于 EE (十六进制)。这是一种计算办法，但是实际上 6800 的程序员可能会变得非常精于以十六进制计算向前和向后的字节数。

8080-8085-Z80 的程序相当简单，但是切勿忽视双字长数值(在 1001, 1002, 1004, 1005 等单元中)中两个字节的顺序。

在将这个程序放入 RAM 中进行测试之前，必须在程序结

```
 >1000,1016
1000  CE  IX:=1080;
1003  BD  JS0231;
1006  81  A-30;
1008  23  JN+0B;
100A  81  A-3A;
100C  3A  JNN+07;
100E  80  A:-30;
1010  A7  M00X:=A;
1012  08  IX:+1;
1013  20  J+EE;
1015  01  NULL;

MONITOR
>D1000,1015
1000 CE 10 80 BD 02 31 81 30 23 0B 8  3A 3A 07 80 30
1010 A7 00 08 20 EE 01
>
```

6800 文本

```
 G2800
1000  21  HL:=1080;
1003  CD  JS0231;
1006  FE  A-30;
1008  FA  JN1017;
100B  FE  A-3A;
100D  F2  JNN1017;
1010  D6  A:-30;
1012  77  M:=A;
1013  23  HL:+1;
1014  C3  J1003;
1017  00  NULL;

MONITOR
>D1000,1017
1000 21 80 10 CD 31 02 FE 30 FA 17 10 FE 3A F2 17 10
1010 D6 30 77 23 C3 03 10 00
>
```

8080 文本

图 4.1

尾处使其返回监督程序(因为这是我们讨论的第一个程序,所以在编写其它程序之前对其进行测试比较好些)。究竟如何处理,取决于监督程序;可能需要一个无条件转移或子程序调用操作,或者(在 6800 上)需要一个所谓的"软件中断"。许多 8080 的初学者都会写上一条 HALT(停机)指令。但是很快就会发现,这样做所产生的麻烦是:从表面上看起来停机与无限的重复循环是无法区别的。但是,现在只要进入监督程序(无论用何种方法)就肯定地表示到达了程序的末尾,并使机器进入处理监督程序的命令的正确状态。

可用下述步骤开始进行测试:

(i) 用监督程序打入用户程序。

(ii) 用监督程序开始执行 1000 号单元的程序。执行过程应在 1003 单元第一次调用 "A:=inchar" 子程序时停机。

(iii) 打入一些十进制数字,接着打入某个其它的字符。若一切顺利,则将重新进入监督程序并准备。

(iv) 显示寄存器 A 和 1080 以上单元的内容。

如果未出现预期的作用,或者未获得正确的结果,那么就必须找出问题的原因,并加以修正。

问题:若 6800 的程序中 1014 号单元的值误打成 EF,或者 8080、8085、Z80 的程序中 1015 号单元的内容误打成 04,则运行这样的程序时会出现什么现象?

若把例 1 中的程序编成为一个子程序,则其它程序在需要时就可调用这个子程序。作为一个子程序使用,如果使寻址寄存器在每次调用时能分别进行预置,则该子程序更为有用。所以每次调用时的起始地址将作为一个参数送入子程序。

代替图 4.1 程序的子程序如图 4.2 所示。值得注意的是,即使 6800 子程序的转移指令与原程序中的转移指令转向的是

```
>1000, 1013
1000  BD  JS0231;
1003  81  A-30;
1005  2B   JN+0B;
1007  81  A-3A;
1009  2A  JNN+07;
100B  80  A:-30;
100D  A7  M00X:=A;
100F  08  IX:+1;
1010  20  J+EE;
1012  39  RET;
MONITOR
>D1000, 1012
1000 BD 02 31 81 30 2B 0F
  81 3A 2A 07 80 30 A7 00 08
1010 20 EE 39
>
```

6800 文本

```
  G2800
1000  CD  JS0231;
1003  FE  A-30;
1005  F8  RN;
1006  FE  A-3A;
1008  F0  RNN;
1009  D6  A:-30;
100B  77  M:=A;
100C  23  HL:+1;
100D  C3  J1000;
MONITOR
>D1000, 100F
1000 CD 31 02 FE 30 F8 FE
  3A F0 D6 30 77 23 C3 00 10
>
```

8080 文本

图 4.2

不同的单元，但是由于转移地址是相对地址，所以这两条指令是一样的。8080-8085-Z80 程序中的条件转移指令已被条件返回指令(6800 无此指令)取代之。

练习

1．编写一个子程序，该子程序读取一系列字母并将其存入一系列连续的存贮单元内，这些单元的起始地址由一个参数给定。当读入一个非字母字符时，该子程序就结束执行，而将这个字符存在寄存器 A 中。

2．由一系列字母和数字混合组成的一组信息以一个句点(ASCII 码 2E)作为结尾。编写一个程序，用现有的子程序读入这组信息，并将其所有的字母存入从 M1080 向上的连续单元内，将其所有的数字存入从 M10C0 向上的连续单元内。

3．继续编写“练习 2”的程序，用“输出一个字符”子程序(入口为 0312)打印出存在 M1080 以上单元内的字母串。

§4.4 十进制整数的输入和输出

下述子程序将从键盘读入一个十进制数字，检查是否是数字，并将其转换成二进制。若按下的是非数字键，则将打印出一个问号(ASCII 码 3F)，并允许继续按键输入其它数字。

```
01: A:=inchar
    A-30
    JN(L02)
    A-3A
    JNN(L02)
    A:-30
    RET
02: A:=3F
    outchar(A)
    J(L01)
```

调用这个子程序的操作可写作“A:=decdig”。现在来考虑下述子程序。中间一列是每条指令的“注释”。由于程序中有两处调用了“A:=decdig”，所以该子程序是读入两个十进制数字。这两个数字可分别称为 t 和 u。指令 2～6 用寄存器 B 作为 $2t$ 的暂存器，将 t 乘以 10(若已知结果是单字长数，则这种办法也许是各种乘 10 的办法中最快的一种了。执行速度在其它程序中可能是一个需要认真考虑的问题，而在这个程序中速度则不太重要，因为两次读操作之间有相当长的一段空闲时间)。当读取 u时用寄存器 B 保存 $10t$，最后的结果是存在寄存器 A 中的 $10t+u$。换言之，该子程序读入一个两位的十进制整数，并将其

6800	注释	8080、8085、Z80
A:=decdig	1:A:=t	A:=decdig
A:*2	2:A:=$2t$	A:+A
B:=A	3:B:=$2t$	B:=A
A:*2	4:A:=$4t$	A:+A
A:*2	5:A:=$8t$	A:+A
A:+B	6:A:=$10t$	A:+B
B:=A	7:B:=$10t$	B:=A
A:=decdig	8:A:=u	A:=decdig
A:+B	9:A:=$10t+u$	A:+B
RET		RET

二进制值存入寄存器A。例如，若读入的是“93”，则寄存器A中就存入“01011101”(十六进制5D)。

该子程序可处理任何无符号的两位整数，即可以处理00～99范围内的任意十进制整数。但是带符号单字长数的整个数值范围是 −128～+127，其中包括带符号或不带符号，有一、二或三个有效数字的数。第八章列出了一个能读取这个范围内的任意数的8080子程序(存贮区为0E74～0EB9)。读者应该仔细地研究这个程序，并注意下述各点：

(i) 当符号为“+”时，符号可有可无。

(ii) 只有当读入跟在数的最后一位数字之后的字符时，该子程序才结束。这个符号不是一个数字，但是读入这个字符不应引起出错。这个字符应该送入一个寄存器，以便检查(0020的“输入-输出”子程序将其结果同时送入寄存器A和C)。

(iii) 必须对数值进行检查，以免其值超出范围。即使是以1开头的三位数也有可能超出范围。

满足这些要求的子程序与读取双字长带符号十进制整数(其范围是 −32768～+32767)的其它子程序(见第八章，存贮单元

0EBA～0F16)一起都应该成为任何正式使用的系统的基本软件的组成部分。

现在再回过头来讨论这一系列练习。考虑与输入相反的问题，即显示该范围内的二进制整数的十进制值(00～99)。为此，必须将该数除以10，要打印的两个数字就分别是商和余数。在这个程序中除法的速度是无关紧要的，所以采用重复减10的简单方法就行了。

```
        B:=00          置十位数字为0。
01:     A-0A           当A小于10时，则转出；A就是个位
                         数字
        JN(L02)
        A:-0A          从"个位数"中减去10
        B:+1           "十位数"加1
        J(L01)         重复测试
02:     B-0A           若产生的十位数字大于9，则出错
        JNN(出错)
        (继续)         十位数字在B中，个位数字在A中
```

由于6800与8080、8085和Z80的内部结构不同，所以将这一设想编入打印子程序时，要求的处理方法也各异(图4.3)。

在6800的程序中，值得注意的是寄存器A和B的作用互换了一下。因此十位数字(先打印)存在A中，在打印十位数字的过程中，个位数字保存在B中。还可以注意到，出错转出时是先经过无条件转移，然后才进入出错处理程序。因为实用时常常必须这样做，所以在例子中已经这样安排了。相对转移的范围只是向前127个字节，向后128个字节。而无条件转移(操作码为十六进制7E)可转向任何地址，因而出错处理程序可以距打印程序127或128个字节以上。

6800		8080, 8085, Z80
A		A−00
JN+1B	(至 L01)	JN(出错)
B:=A		B:=00
A:=0		02:A−0A
02:B−0A		JN(L01)
JN+05	(至 L03)	A:−0A
B:−0A		B:+1
A:+1		J(L02)
J+F7	(至 L02)	01:C:=A
03:A−0A		A:=B
JNN+0C	(至 L01)	A−0A
A:+30		JNN(出错)
outchar(A)		A:+30
B:+30		outchar(A)
A:=B		A:=C
outchar(A)		A:+30
RET		outchar(A)
01:J(failure)		RET

图 4.3

在 8080-8085-Z80 的程序中，由于具有要求的特性的单字长累加器只有一个（寄存器 A），所以另用寄存器 C 作为暂存器。

两位数字的读入和打印子程序的全编码文本如图 4.4 和图 4.5 所示。其入口是：

A:=decdig	在两种文本中都是 1000
读出	6800 中为 1015; 8080 中为 1018
打印	6800 中为 1023; 8080 中为 1026。

除了提供出错处理程序的地址这一点以外，这两个打印子程序都是完整的。在“A:=decdig”子程序（入口为 1000）中，

```
>1000, 1044
1000    BD      JS0231;
1003    81      A-30;
1005    2B      JN+07;
1007    81      A-3A;
1009    2A      JNN+03;
100B    80      A:-30;
100D    39      RET;
100E    86      A:=3F;
1010    BD      JS0312;
1013    20      J+EB;
1015    BD      JS1000;
1018    48      A:*2;
1019    16      B:=A;
101A    48      A:*2;
101B    48      A:*2;
101C    1B      A:+B;
101D    16      B:=A;
101E    BD      JS1000;
1021    1B      A:+B;
1022    39      RET;
1023    4D      A;
1024    2B      JN+1B;
1026    16      B:=A;
1027    4F      A:=0;
1028    C1      B-0A;
102A    2B      JN+05;
102C    C0      B:-0A;
102E    4C      A:+1;
102F    20      J+F7;
1031    81      A-0A;
1033    2A      JNN+0C;
1035    8B      A:+30;
1037    BD      JS0312;
103A    CB      B:+30;
103C    17      A:=B;
103D    BD      JS0312;
1040    39      RET;
1041    7E      J9999;

MONITOR
>D1000, 1043
BD 02 31 81 30 2B 07 81 3A 2A 03 80 30 39 86 3F
BD 03 12 20 EB BD 10 00 48 16 48 48 1B 16 BD 10
00 1B 39 4D 2B 1B 16 4F C1 0A 2B 05 C0 0A 4C 20
F7 81 0A 2A 0C 8B 30 BD 03 12 CB 30 17 BD 03 12
39 7E 99 99
>
```

图 4.4

```
G2800
1000    CD    JS0231;
1003    FE    A-30;
1005    FA    JN1010;
1008    FE    A-3A;
100A    F2    JNN1010;
100D    D6    A:-30;
100F    C9    RET;
1010    3E    A:=3F;
1012    CD    JS0312;
1015    C3    J1000;
1018    CD    JS1000;
101B    87    A:+A;
101C    47    B:=A;
101D    87    A:+A;
101E    87    A:+A;
101F    80    A:+B;
1020    47    B:=A;
1021    CD    JS1000;
1024    80    A:+B;
1025    C9    RET;
1026    FE    A-00;
1028    FA    JN00B4;
102B    06    B:=00;
102D    FE    A-0A;
102F    FA    JN1038;
1032    D6    A:-0A;
1034    04    B:+1;
1035    C3    J102D;
1038    4F    C:=A;
1039    78    A:=B;
103A    FE    A-0A;
103C    F2    JNN00B4;
103F    C6    A:+30;
1041    CD    JS0312;
1044    79    A:=C;
1045    C6    A:+30;
1047    CD    JS0312;
104A    C9    RET;
```

```
MONITOR
>D1000, 104A
1000 CD 31 02 FE 30FA 10 10 FE 3A F2 10 10 D6 30 C9
1010 3E 3F CD 12 03 C3 00 10 CD 00 10 87 47 87 87 80
1020 47 CD 00 10 80 C9 FE 00 FA B4 00 06 00 FE 0A FA
1030 38 10 D6 0A 04 C3 2D 10 4F 78 FE 0A F2 B4 00 C6
1040 30 CD 12 03 79 C6 30 CD 12 03 C9
>
```

图 4.5

"出错"只是表示发生了打印错误。对出错的处理只是简单地打出一个问号引起操作员注意，并允许重新打入正确的数字。"读入"子程序调用两次"A:=decdig"子程序，两次调用过程中出现的错误，只须用同样的办法进行处理就行了。

打印子程序中的错误则完全不同，因为这表示寄存器A中需要由该子程序打印的值是该子程序无法打印的值。一种简单易行的办法就是转入监督程序中的出错处理程序(如果有的话)。一个好的监督程序最后能将执行过程返回至程序的起点，甚至能返回至程序中计算开始出错的数据之处。在这样处理时，必须克服存在于这些子程序内而又不能忽略的一个问题。

在这两种打印子程序中，出错转出时都只是转移出该子程序。若发生这一转移，则所离开的子程序的返回地址仍然留在堆栈中。实际上，若打印子程序是被另一个子程序所调用的，而该子程序又被其它的子程序所调用(如此等等)，则上述返回地址"之下"还可能有其它的返回地址。究竟会发生什么情况取决于打印完错误报告(错误信息)后要做什么工作。更确切地说，就是程序将如何继续下去。若程序必须从头开始执行，则必须取消这些返回地址(例如可重新预置堆栈指示器)。但是，若知道只有一个返回地址存在堆栈中，则用"SP:+1;SP:+1"这对指令就可取消这个地址，程序就可从子程序之外的任意点开始继续执行下去。

许多程序员认为出错时从子程序转出是一种笨办法，因而不惜任何代价避免这样做。若在执行子程序时检测到一个错误状态，它们就使该子程序打印出一个错误报告(如果适当的话)，然后正常地返回调用它的程序，返回时将表示出错的某种标志值存在某处。调用程序或子程序显然必须对这个值(常常只有

一位——即一个软件"标志")进行测试,并决定如何继续执行下去。

下述程序将读入两个整数并将其相加。若结果在允许的范围内,则将其打印出来。若超出了允许的范围,则将打印两个问号。无论出现哪种情况,该程序都将重复执行,重复的遍数是任意的。每重复一遍,打印输出就换一行。

6800	8080, 8085, Z80
01: A:=0A	01: A:=0A
outchar(A)	outchar(A)
A:=0D	A:=0D
outchar(A)	outchar(A)
JS1015	JS1018
S:=A	D:=A
JS1015	JS1018
B:=S	A:+D
A:+B	JS1026
JS1023	J(L01)
J(L01)	(出错): SP:+1
(出错): SP:+1	SP:+1
SP:+1	A:=3F
A:=3F	outchar(A)
outchar(A)	outchar(A)
outchar(A)	J(L01)
J(L01)	

许多程序设计专家认为以这种(无穷循环)方式编写程序是不足取的,因为这样就无处使用他们的理论中的"复位"(RESET)信号。有了"复位"信号后,就可在任何阶段很容易地放弃执行这个程序,而不会产生任何问题。附带说一句,可以注意到在这个程序中监督程序本身也几乎肯定是一个无穷循环的

程序。在这两种机器的程序中，读取第二个数时第一个数将被存入一个暂存器。在 8080、8085 和 Z80 的程序中，可使用寄存器 D；但是在 6800 中没有空闲的寄存器可供使用，所以用堆栈作为暂存器(S:=A; …; B:=S)。当程序被翻译成机器码并与子程序一起存入 RAM 中时，因出错从打印子程序转出的地址将是上述程序中标明"(出错)"的指令的单元地址。

练习

下述每个程序都要求执行一遍就换一行；同一行上的两个连续的数之间应该用一个空格分开。

1．编写一个程序，该程序读入若干个"成对"的数 a 和 b；打印出各对数之和 $x=a+b$ 以及差 $y=a-b$。

2．编写一个程序，该程序读入每组由三个数 a、b、c 组成的若干组数，顺序打印每组的和数 $x=b+c$、$y=c+a$ 以及 $z=a+b$。

3．利用打印子程序中所用的除法技术编写一个程序，该程序读入若干个"成对"的数 a 和 b，打印出各"成对"数之商 q 及余数 r (注：$a=b\times q+r$，$0\leqslant r\leqslant b$。除非 $b=0$，否则 q 与 r 必定在允许的数值范围内，见 § 4.5.5)。

第八章给出了适用于整个单字长整数和双字长整数范围的两个输出子程序，两个程序一起存在 0D CA～0E 73 单元内，入口是 0E 68(输出寄存器 A 中的单字长带符号整数)与 0DCA(输出寄存器对 HL 中的双字长带符号整数)。

§ 4.5 二进制整数的算术运算

上一节已讨论了将整数读入单字长和双字长寄存器的程序，以及打印存在这类寄存器中的整数值的程序。这些数值的外部形式都是十进制的，而其内部形式则是二进制的。本节和

下一节将讨论二进制整数的算术运算。

§2.4.3和§2.4.5已讨论了无符号和带符号二进制整数的加法和减法。

4.5.1 改变数的长度

将一个单字长无符号整数转变成双字长数，只须加上一个全零的高字节即可。在将一个双字长无符号整数转变成单字长数时，必须先检查高字节，以保证要去除的高字节是全零字节。

6800	8080、8085、Z80
(单字长数在A中)	(单字长数在A中)
B:=A	L:=A
A:=0	H:=00
B	A−H
JNN+01	RNN
A:−1	H:−1
RET	RET
(双字长数在AB中)	(双字长数在HL中)

为将一个单字长带符号整数转变成双字长数，必须形成一个八位都与符号位(即给定数值的最高位)相同的高字节。这一工作值得用一个子程序来完成。若欲将一个双字长带符号整数转变成单字长数，必须先检查高字节中的八位是否与低字节的最高位一样。这一工作同样值得用一个子程序来完成。

扩充多字长整数的长度相当简单，对于无符号整数只须增加一个全零字节，而对于带符号整数则需增加一个八位都与原数的符号位一样的字节。与此相反，若多字长无符号数的最高字节为全0，或带符号数最高字节中的八位都与下一个字节的最高位一样，则多字长整数的长度就可缩短一个字节。

6800	8080、8085、Z80
(双字长数在 AB 中)	(双字长数在 HL 中)
B	A:=L
JNN+01	A:&L
A:+1	JNN(L01)
JNZ+02	H:+1
A:=B	01: A:=H
RET	A:&L
J(出错)	JNZ(出错)
	A:=L
	RET
(单字长数在 A 中)	(单字长数在 A 中)

4.5.2 比较

§2.4.7 介绍并讨论了"测试"和"比较"指令。

比较两个多字长整数（长度相等!）显然可以通过检查多字长减法的结果来实现。但是，如果能完成比较操作而不必存放减法的结果，就不需要存贮多字长的差值，当然非常有利。

假定 a、b 是两个 n 倍字长的整数，各个字节分别为 a_{n-1}、a_{n-2}、…、a_0 和 b_{n-1}、b_{n-2}、… b_0，因此有：

$$a=\sum_{i=0}^{n-1}a_i2^{8i} \quad 及 \quad b=\sum_{i=0}^{n-1}b_i2^{8i}$$

若 $a_{n-1}<b_{n-1}$ 则 $a<b$；若 $a_{n-1}>b_{n-1}$ 则 $a>b$

在上述情况下都只须比较最高字节。但是若 $a_{n-1}=b_{n-1}$，则必须对下一个字节 a_{n-2}、b_{n-2} 进行比较。若二者不等，则比较操作完成；但是若二者相等，则必须再比较下一对字节。如果必要的话这个过程必须进行下去，直至对 a_0、b_0 也进行了比较为止。只有当比较的各对字节都相等时才能确定 a 与 b 相等。

因此，比较过程可以表示成：

$i:=n-1$

```
01: ai-bi
    RNZ
    i:-1
    JNN(L01)
    RET
```

若转出时 $i<0$，则 $a=b$（各对字节都相等）。否则就有：

（i）若 a、b 是无符号整数，或 a、b 是带符号数且 $i<n-1$，则若 K=1 就是 $a<b$，若 K=0 就是 $a>b$；

（ii）若 a、b 是带符号整数且 $i=n-1$，则比较操作就是比较两个带符号单字长整数 a_{n-1}、b_{n-1}（见§2.4.7）。

特别简单的比较过程是下述 8080、8085 或 Z80 用的子程序，该程序将对分别给定于 HL 和 DE 中的无符号整数 a、b 进行比较，若 $a=b$ 则 Z=1；若 $a<b$ 则 Z=0、K=1；若 $a>b$ 则 Z=0、K=0。程序将不可避免地使用寄存器 A。

```
A:=H; A-D; RNZ; A:=L; A-E; RET。
```

4.5.3 单字长乘法

两个 8 位的整数 a、b 的乘积 $a\times b$ 的字长有 16 位。虽然某些微处理机有乘法指令，但 6800、8080、8085 和 Z80 都没有。产生乘积（甚至更高级的功能）的一种办法是利用外部硬件。例如，可采用乘法集成电路实现乘法。这类器件可装配在系统内，占用四个地址。若 a 和 b 被“写入”其中两个地址，则读取另一对地址就可获得其乘积。另一种办法就是用一个子程序来完成乘法①。

① 能完成其它算术运算且与微处理机系统兼容的集成电路，现在有很多种可供使用。其中有些能完成全部浮点操作（见§4.7）、计算几种三角函数或其它函数（见§6.3）。将这样的器件装入系统需采用§5.7提及的那些技术，每种类型的器件都需具体进行考虑（设计超出这些器件的运算功能的程序时，也可能用到这些器件）。

常用的乘法过程很象是算术里的“多位数乘法”。例如，在十进制中 $5\times7=35$。

在二进制中就是：

```
      00000101
    ×00000111
--------------
      00000101
     00000101·
    00000101··
   00000000···
  00000000····
 00000000·····
00000000······
00000000·······
--------------
0000000000100011
```

实际上这很简单：若乘数(00000111)中有一位是1，就在该位上将被乘数(00000101)加入部分乘积。说得更确切一点就是若 $b_i=1$($i=00\sim07$ 或 $00\sim17$)，则将 $a\times2^i$ 加入部分乘积。

下面是一个单字长乘法子程序。在该子程序的运算过程中，其乘积形成于 AB(6800)或 HL(8080、8085、Z80)中。该乘积是两个无符号单字长整数的无符号双字长乘积。

6800	8080, 8085, Z80
(假定 a、b 在 RAM 中)	(a、b 分别在寄存器 C、E 中)
A:=0	HL:=0000
B:=0	A:=C
IX:=0008	D:=H
02: A:*2	B:=08
B:*2	02: HL:+HL
A:++00	A:@L
M(a):*2	JNK(L01)
JNK(L01)	HL:+DE
B:+M(b)	01: B:−1
A:++00	JNZ(L02)
01: IX:−1	RET
JNZ(L02)	(结果 $a\times b$ 在 HL 中)
RET	
(结果 $a\times b$ 在 AB 中)	

但是，若二因数之一是带符号的，则该子程序就不会产生预期的结果。这是因为符号位具有特殊的意义，而在上述子程序中对每个因数中八位数字的处理是一样的。为了理解如何根据因数的符号对产生的乘积进行修正，可回顾一下带符号整数值的定义（见§2.4.2）。即：

$$a=-2^7a_7+\sum_{i=0}^{6}a_i2^i$$

和

$$b=-2^7b_7+\sum_{i=0}^{6}b_i2^i$$

该子程序产生的乘积是：

$$\begin{aligned}P&=\sum_{i=0}^{7}a_i2^i\times\sum_{i=0}^{7}b_i2^i\\&=(a_7 2^8+a)\times(b_7 2^8+b)\\&=a\times b+2^8(a_7\times b+b_7\times a)\end{aligned}$$

（由于乘积只有16位，所以2^{16}项会丢失。）因此为给出带符号的乘积而必须对P进行的符号修正就是加上$-2^8(a_7\times b+b_7\times a)$。换言之，就是：

（i）若a为负，则必须从产生的乘积的高字节中减去b；

（ii）若b为负，则必须从产生的乘积的高字节中减去a。

显然，若a和b都是负，则两种符号修正步骤都要进行。

如下所示，在多字长带符号数的乘法中，必须能将(i)两个无符号数，(ii)一个带符号数与一个无符号数，以及(iii)两个带符号数乘在一起。所以，将这两种符号修正过程编成子程序是很有用处的。

两个带符号单字长整数相乘形成带符号双字长乘积的过程如下：

无符号乘法

根据a的符号进行修正

6800	8080、8085、Z80
(要修正的乘积在 AB 中)	(要修正的乘积在 HL 中)
M(*a*)	A:=C
JNN(L01)	A−00
A:−M(*b*)	RNN
01: RET	A:=H
(根据 *a* 的符号进行修正)	A:−E
	H:=A
(要修正的乘积在 AB 中)	RET
M(*b*)	(C 中存有根据 *a* 的符号
JNN(L02)	进行修正后的乘积)
A:−M(*a*)	(要修正的乘积在 HL 中)
02: RET	A:=E
(根据 *b* 的符号进行修正)	A−00
	RNN
	A:=H
	A:−C
	H:=A
	RET
	(E 中存有根据 *b* 的符号
	进行修正后的乘积)

根据 *b* 的符号进行修正

返回

本节介绍的子程序是第八章给出的软件的一部分，占用的单元是 0F18～0F43。

4.5.4 多字长乘法

考虑两个三倍字长整数乘法的过程：

$$a=2^{16}a_2+2^8a_1+a_0,\quad b=2^{16}b_2+2^8b_1+b_0$$

可得：

$$a\times b=2^{32}a_2b_2+2^{24}(a_2b_1+a_1b_2)+2^{16}(a_2b_0+a_1b_1+a_0b_2)$$
$$+2^8(a_1b_0+a_0b_1)+a_0b_0$$

结果 $a\times b$ 需要占用六个字节。根据双字长的部分乘积形成最后的乘积的过程如下所示(该过程称为"Wallace 树", 某些读者可能很熟悉):

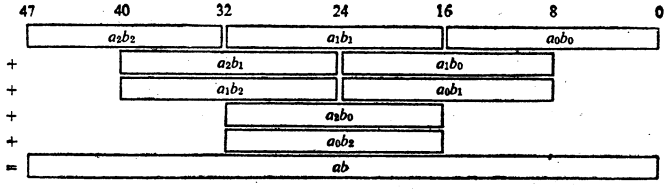


图 4.6

若 a 与 b 是两个无符号整数, 则经过这个过程就会产生其无符号乘积。若 a 与 b 是带符号整数,则形成带符号乘积的过程是: (i)若 a 为负, 则从乘积的高半段中减去 b; (ii)若 b 为负, 则从乘积的高半段中减去 a。因此,根据上图继续画下去的话, 有两种编制符号修正程序的办法。一种办法是如图所示, 先形成无符号乘积, 然后根据两个符号进行修正。另一种办法是记住四个部分积(a_1b_1、a_1b_0、a_0b_1、a_0b_0)是两个无符号数的乘积, 另外四个部分积(a_2b_1、a_2b_0、a_1b_2、a_0b_2)是由一个带符号数与一个无符号数形成的乘积, 而第九个部分积 (a_2b_2) 是两个带符号数的乘积。在形成后五个部分积时可顺序地逐个进行符号修正。但是也不能忘记: 这五个部分积的前四个中都可能产生进位,这些进位必须加入乘积的较高字节中去。

8080、8085 和 Z80 都有几个寄存器,且有双字长加法指令,

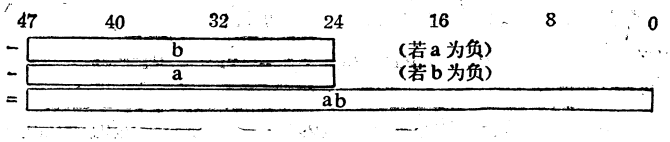


图 4.7

用于实现双字长或三字长乘法是很方便的。这一点留待第八章介绍另一种乘法子程序时再加以说明。以2400号单元为入口的子程序根据BC和DE中的两个16位无符号整数而求得其32位的乘积，存在DEHL中。该子程序采用§4.5.3介绍的方法，但是每步执行的都是十六位的加法。以2420号单元为入口的子程序利用了以2400号单元为入口的子程序计算两个带符号整数的带符号乘积。该子程序与入口为0F 3A的子程序不同，在开始进行无符号数的乘法之前就先将符号修正量准备在结果寄存器中。这样只需增加很少几条指令就可实现以2556号单元为入口的单符号修正子程序以及以2562号单元为入口的子程序。以250A单元为入口的子程序能根据BHL和CDE中的两个24位带符号整数产生48位带符号的乘积，存在BCDEHL中。该子程序采用的办法是将每个因数分割成8位带符号的"头"与16位无符号的"尾"。然后，顺序把"尾"的乘积(用以2400为入口的子程序)、第一个因数的"头"与第二个因数的"尾"的乘积(用以2556为入口的子程序)、两个"头"的乘积(用以0F 3A为入口的子程序)及第二个因数的"头"与第一个因数的"尾"的乘积(用以2562单元为入口的子程序)累加起来，实现带增量符号修正的Wallace树的作用。为了避免直接写出RAM单元的地址，在形成结果时该子程序采用堆栈指示器对这些部分积进行寻址。如果有其它堆栈指示器的话，就不必这样滥用堆栈(参阅§2.5和§2.7)。

这些子程序是供8080和8085使用的。由于Z80有其它的寄存器和指令可供使用，所以用于Z80时无疑可缩短一些。

上述子程序全都是以常规的多位数乘法为基础的。实现带符号乘法的另一种办法是使用"Booth算法"。这一过程将有规律地使用心算中常用的技巧。例如，在计算77×99时，可以先

将 77 乘以 100，得 7700；然后减去 77×1，得结果 7623。在这种特殊情况下，这比统统乘出来短得多。这种算法将乘数看作从最低位开始的一串二进制数字。只要这串数字为 0，就不执行任何操作。当这串数字一变成 1，则遇到第一个"1"时执行一次减法；而对于其后的"1"则不执行任何操作。当这串数字再变成"0"时，则遇到第一个"0"时执行一次加法；而对其后的"0"则不执行任何操作。所以，在计算 5(00000101)×7(00000111)时，该算法将 7 看作为三个"1"后跟有五个"0"的一串数字。对于第一个"1"，该算法将减去 5×2°，形成部分结果 −5(11111011)。对于第二和第三个"1"，则不执行任何操作。下一位 d_3 是"0"，所以要加 5×2^3(00101000)，得到最后的结果 00100011(35)。因此，这种办法用一次减法与一次加法(乘数中数字串改变了两次)代替了常规算法中的三次加法(乘数中有三个"1")。

但是，若将两个相乘的数交换一下，用 Booth 算法计算 7×5，就需要两次加法与两次减法 (0−7×2°+7×2^1−7×2^2+7×2^3=35)。而常规算法只需要执行两次加法(乘数中有两个"1")。在将两个 n 位数相乘形成 $2n$ 位的乘积时，每种算法都需要一个执行 n 遍的程序循环。若乘数有 m 个"1"，则采用常规算法，在 n 个步骤中的 m 步将执行加法。常规算法进行符号修正时还可能需要执行一或二次减法；而 Booth 算法能直接给出带符号乘积。但是，常规算法所需的指令数稍少一些。

4.5.5 单字长除法

整数 a 除以整数 b 的操作将产生两个整数值——商 q 及余数 r。

若 a、b 是无符号整数，则 q、r 由下述关系来定义：

$$a=q\times b+r \quad (0\leqslant r<b)$$

若除数 b 为 0，则除法无意义。

若 a、b 是无符号单字长整数且 $b\neq 0$，则 q 和 r 都是无符号单字长整数。

根据该定义就可得到一种简单的计算 q 和 r 的算法。

```
    q:=0
    r:=a
01: r-b
    RN
    r:-b
    q:+1
    J(L01)
```

实际上这就是§4.4中除以10所用的办法。在§4.4中就已认为这种办法是一种很慢的办法。用作十进制输出子程序时速度是不重要的，但是在其它情况下就可能太慢了(在 $a=255$，$b=1$ 的最坏情况下该循环要执行255遍)。

较快的一种方法是以"多位数除法"为基础的。首先将除数左移，直至再移一位就将大于被除数 a 为止。若左移了 s 位，则得 $2^s b$，且 $2^s b\leqslant a<2^{s+1}b$。从被除数中减去移位后的除数，被除数就变为 $a-2^s b$，同时得到商 q 的一位 $q_s(q_s=1)$。然后将除数右移，直至小于剩下的被除数为止。然后再执行一次减法，得到商中的另一位。当除数返回其原始位置时，就停止"右移并作减法"的过程。这一过程当然比前一种办法复杂得多，但是在大多数情况下也快得多。用程序(符号)来表示就是：

```
    p:=1
    s:=1
01: b-a
    JNN(L02)
    p:+1
```

```
    s:+1
    b:*2          ┐
    JNN(L01)      ├ *
    bK:@R         ┘
    p:-1
02: q:=0
03: q:*2
    a-b
    JN(L04)
    a:-b
    q:+1
04: a:*2
    p:-1
    JNZ(L03)
    RET
```

在返回原程序时，商为 q，余数为 $r=a\times 2^{-s}$。其中 s 为位置计数器 p 所达到的最大值。注明"*"的指令是防止除数侵入符号位用的（必须用 bK:@R 产生与 b:*2 相反的作用）。若不需要余数，则可去掉指令 $s:=1$、$s:+1$。

若 a 与 b 是带符号整数，则 q 与 r 的定义为：

$$a=q\times b+r$$

若 $b>0$，则 $0\leqslant r<b$

若 $b<0$，则 $b<r\leqslant 0$

根据上述定义可将简单而速度较慢的算法分成以下四种情况：

（i）$a\geqslant 0$, $b>0$

```
    q:=0
```

（ii）$a<0$, $b>0$

```
    q:=0
```

```
        r:=a                    r:=a
01:     r-b             02:     r-b
        RN                      RN
        r:-b                    r:÷b
        q:+1                    q:-1
        J(L01)                  J(L02)
(iii)   a≥0, b<0        (iv)    a<0, b<0
        q:=0                    q:=0
        r:=a                    r:=a
03:     b-r             04:     b-r
        RN                      RN
        r:+b                    r:-b
        q:-1                    q:+1
        J(L03)                  J(L04)
```

较快的过程也可按这四种情形进行划分。但是在许多场合没有必要这样地考虑余数，只须将$|a|$除以$|b|$得到$|q|$就够了。若a、b的符号不同，则$q=-|q|$。

4.5.5.1 单字长数除法形成的多字长商数

在上一节介绍的多位数除法过程中，若用p:+1代替第一条指令p:=1，且进入该过程时p是个正整数值（例如p_0），则结束时q的值就是：

$$q=\frac{a}{b}\times 2^{p_0}$$

但是这并非简单地将原来的q左移p_0位。相反，右边的p_0个数字也是有效数字——执行的除法实际上是$a\times 2^{p_0}$除以b。当然，由于a/b是单字长数，因此这个新的q需要$8+p_0$位。若p_0的范围为$0\leqslant p_0\leqslant 8$，则任意商数都处在双字长范围内。

现在讨论一下在特定的情况下如何使用 p_0。假设 $a=27$（十进制），$b=10$（十进制），则 $a/b=2.7$。

若 $p_0=0$，可得 $q=\dfrac{27\times 2^0}{10}=2$

$p_0=1 \quad q=\dfrac{27\times 2^1}{10}=5 \qquad (2\times 2.5) \qquad \left(2\dfrac{1}{2}\right)$

$p_0=2 \quad q=\dfrac{27\times 2^2}{10}=10 \qquad (4\times 2.5) \qquad \left(2\dfrac{2}{4}\right)$

$p_0=4 \quad q=\dfrac{27\times 2^4}{10}=43 \qquad (16\times 2.6875) \qquad \left(2\dfrac{11}{16}\right)$

$p_0=8 \quad q=\dfrac{27\times 2^8}{10}=691 \qquad (256\times 2.6992) \qquad \left(2\dfrac{179}{256}\right)$

p_0 的值越大，a 与 b 实际形成的商数的小数部分越接近真值。$p_0=8$ 的情形特别有趣。在该例中这种情形下的结果是整数691（十进制）。若分别观察结果的两个字节，则高字节的值为2（27/10 的整数部分）；低字节的值为179（十进制），表示2.7的小数部分的近似值为 179/256。

这是我们第一次遇到非整数的表示法。以后将会详细介绍这种表示法及其它一些表示法。增加 p_0 的范围显然能产生比双字长更长的商数。

第八章有一个供 8080/8085/Z80 用的单字长整数的无符号除法子程序，其入口为 0F 50。该子程序的 p_0 限于 8 以下，因此其结果为双字长数。此外还有一个带符号除法子程序，其入口为 0F 76。

4.5.6 多字长整数的除法

若上一节介绍的程序中每一步都编制得可对适当长度的数进行处理，则可用来执行双字长整数的除法。根据单字长整数的除法形成双字长整数除法的商数是行不通的。

第八章给出了双倍字长子程序，其入口地址分别为 0F 9B 和 0FCD，与前一节末尾所提及的 8080/8085/Z80 除法子程序相对应。

当除法的字长更长时，采用另一种办法可能更好些，即求出除数的倒数，并与被除数相乘。若处理的是非整数量，则可考虑采用倒数法。

§4.6 非整数量的二进制算术运算

4.6.1 非整数量的"定点"表示法

数 2.75 有一个整数部分 2 与一个小数部分 0.75。可以将该数看作为 2+75/100，或 275×10^{-2}。这两种表示法中都只含有整数。我们称 2.75 有两个十进制小数位；有三位有效数字。

同样的概念也适用于非整数量的二进制表示法。二进制数 10.11 有一个整数部分 10（十进制 2）与一个小数部分 0.11（3/4）。可以将其看作 10+011/100 或 1011/0100，这两种表示法也都只含有（二进制）整数。我们称 10.11 有两个二进制小数位；有四位有效数字。

因为在微处理机系统中一个字节（或一个字）只能容纳 8 位，两位之间无处放二进制小数点。但是程序员可认为在字节中的某处有一个二进制小数点，而编写算术运算程序时就好象有小数点一样。

因此，在知道 d_1 与 d_2 之间有一个二进制小数点，即该字节有两个二进制小数位时，$2\frac{3}{4}$(10.11) 就可表示成一个字节 00001011。

另一个数 $3\frac{7}{8}$(11.111) 可表示成一个字节 00011111，此时假想的小数点在 d_2 与 d_3 之间，或者说有三个二进制小数位。

4.6.2 乘法

若用带符号或无符号乘法子程序将上述两个字节乘起来，则可得16位的乘积：

0000000101010101

作为一个整数时其值为341(十进制)。但是，相乘的两个字节分别有两个和三个二进制小数位。所以，这个乘积有五个(2+3)二进制小数位，表示的是：

$$1010.10101 \quad \left(10\frac{21}{32}\right)$$

$2\frac{3}{4}\times 3\frac{7}{8}=10\frac{21}{32}$，所以结果是正确的。

若存贮的一个数 x 有 m 个二进制小数位，而存贮的另一个数 y 有 n 个二进制小数位，则二数相乘之积 xy 就有 $m+n$ 个二进制小数位。为说明这一点，可以用另一种方式来看待这种表示法。x 在机器中实际是由整数 $x\times 2^m$ 表示的，y 则由 $y\times 2^n$ 表示，由整数乘法给出的乘积是 $xy\times 2^{(m+n)}$，即把 xy 表示成有 $m+n$ 个二进制小数位。

在上述例子中，$2\frac{3}{4}$ 表示成 $2\frac{3}{4}\times 2^2=11$(00001011)，$3\frac{7}{8}$ 表示成 $3\frac{7}{8}\times 2^3=31$(00011111)；$11\times 31=341=10\frac{21}{32}\times 2^5$。

4.6.3 加法与减法

加法稍微复杂一些。先考虑上述二数(2.75与3.875)执行十进制加法的情形。相加之前必须先对准小数点(至少在心里)因此就有：

$$\begin{array}{r} 2.75 \\ 3.875 \\ \hline 6.625 \end{array} \quad \left(6\frac{5}{8}\right)$$

与此相仿，在执行二进制加法之前也必须对准二进制小数点：

$$\begin{array}{r} 10.11 \\ 11.111 \\ \hline 110.101 \end{array} \quad \left(6\frac{5}{8}\right)$$

换言之，每个加数都必须用同样位数的二进制小数位(b. p)来表示。

在相加之前，一个加数有两个二进制小数位，另一个加数有三个二进制小数位。若将有三个二进制小数位的数右移一位，使两个加数都是两个二进制小数位，则该数将损失最低位数字，这当然是不希望发生的。所以应将有两个二进制小数位的数左移一位，使两个数都有三个二进制小数位。然后执行整数加法：

$$\begin{array}{rr} 00001011\,(2\text{b. p}) & 00010110\,(3\text{b. p}) \\ & +00011111\,(3\text{b. p}) \\ \hline & 00110101\,(3\text{b. p}) \end{array}$$

由于相加的两个数均为三个二进制小数位，所以和数也是三个二进制小数位。因此结果就表示 $110.101\left(6\frac{5}{8}\right)$，是正确的。

由此可知，若一数 x 有 m 个二进制小数位，另一数 y 有 n 个二进制小数位，则：

(i) 若 $m=n$，直接相加可得和数 $x+y$，和数也有 $m(=n)$ 个二进制小数位；

(ii) 若 $m>n$，则 y 必须左移 $(m-n)$ 位，然后相加得 $x+y$，和数有 m 个二进制小数位；

(iii) 若 $m<n$，则 x 必须左移 $(n-m)$ 位，然后相加得 $x+y$，

和数有 n 个二进制小数位。

在第一种情况下，由于 $m=n$，x、y 分别表示成 $x\times 2^m$ 与 $y\times 2^m$。二者相加，可得 $x\times 2^m+y\times 2^m=(x+y)\times 2^m$，即 $x+y$ 有 m 个二进制小数位。

在第二种情况下，x 表示成 $x\times 2^m$；移位之前 y 表示成 $y\times 2^n$，移位后则表示成 $y\times 2^n\times 2^{m-n}=y\times 2^m$。

与(i)一样，$x\times 2^m$ 与 $y\times 2^m$ 相加可得 $(x+y)2^m$。

在第三种情况下，移位前 x 表示成 $x\times 2^m$，移位后 x 表示成 $x\times 2^m\times 2^{n-m}=x\times 2^n$；$y$ 表示成 $y\times 2^n$。$x\times 2^n$ 与 $y\times 2^n$ 相加可得 $(x+y)\times 2^n$，即 $x+y$ 有 n 个二进制小数位。

减法完全与加法类似。

但是，有一个困难不应忽视。在将任何数左移时，都有溢出的危险。在带符号数的算术运算中，最高有效位（正数最左边的1或负数最左边的0）决不允许移入符号位。

在许多情况下，程序员知道计算的结果不会导致发生上述情形；但是在大多数情况下却无法保证这一点。所以，一般说来，在每次左移之前必须先进行检查。若检查结果说明左移将出错，则进行加、减法的唯一可能办法就是将另一个数右移。

正如前面所述，右移将损失部分信息。若不得已这样做，则应保证损失的信息尽可能少（可参阅§4.6.4有关四舍五入的内容）。

加、减法本身也可能产生溢出（加法中二数符号相同，且接近于数值范围的界限，或减法中二数符号相反，且接近于数值范围的界限时都会产生这种情况）。但是，由于溢出的结果可被右移（应当小心！），使结果落在范围内（比通常预期的结果少一个二进制小数位），所以这种情形并不可怕。

4.6.4 除法

除法与乘法相似，不必进行移位就可完成。x(m 个二进制小数位)除以 y(n 个二进制小数位)就得 x/y[$(m-n)$ 个二进制小数位]。但是，若 m 与 n 相等或接近，则结果的二进制小数位就可能太少，即不能满足所要求的运算精度。因此，必须采用§4.5.5.1 中所介绍的"p_0"概念。

具有这种"p_0"功能的除法子程序能给出有 $(m-n+p_0)$ 个二进制小数位的商 x/y。例如：

取　　$x=2.75$ (2b. p)

　　$y=3.875$(3b. p)

若 $p_0=0$(或无 p_0 功能)，则商数将是零(−1b. p②)，这与真正的商数 0.7097 相去甚远。

但是若 $p_0=8$，则商数就是下式的整数部分：

$$\frac{(2.75\times 2^2)\times 2^8}{(3.875\times 2^3)}=90$$

该数在二进制中就是 0000000001011010。二进制小数位的位数为 2−3+8=7。因此，商数就是 0.1011010，即 90/128=0.703125，与真正的商数 0.7097 较接近了。

这个例子说明了在这类应用中应如何修改除法子程序，以给出较好的结果。若根据余数的数值四舍五入至商数，则结果就更好：

$$\frac{(2.75\times 2^2)\times 2^8}{3.875\times 2^3}=\frac{2816}{31}=90 \quad 余\ 26。$$

由于 26 大于除数 31 的一半，因此四舍五入使商数为 91。所以，改善后的结果就是：

0000000001011011(7b. p)

② 二进制小数位(b. p)为负数表示假想的小数点在现有最低位的右边。

即 0.1011011，等于 91/128＝0.7109，更接近于 0.7097。

读者可验证一下，若 $p_0=16$，并采用四舍五入的除法，则结果(15b.p)为 23255/32768＝0.7097（二进制结果与真正结果的误差在 0.01% 以内）。

§4.7 “浮点”二进制数

在程序中采用§4.6提出的概念时，主要需解决的问题是对各种移位操作加以控制，以保证达到所需的精度（二进制小数位位数），同时保持数值不超出范围。为此，程序员不仅必须知道数值的范围和每个输入量的精度，以及每个结果的范围和所需的精度，而且必须在设计程序之前，了解程序执行过程中产生的每个中间结果的范围和精度。

在许多小任务中，这个工作相当简单，增加这点麻烦是值得的。这样产生的程序可以既简短又有效。但是，在规模较大、较复杂的任务中，这个问题就变得很复杂、冗长且很费时。最糟糕的是，可能会导致程序设计出错，特别是产生极难判断与修正的隐蔽的错误。

当然，解决的办法是由机器来完成这一工作。若如前所述假想操作数有一个二进制小数点，则在编写对存贮器中的数值进行操作的程序时，必须考虑这个小数点的位置。但是，还有一种办法是为存贮器中的每个数配上一个整数（多占用一个字节），其值表示二进制小数点的位置。§4.6采用的技术是以下述假定为基础的，即存在特定单元或寄存器中并由特定指令进行处理的每一个数值都具有同样数量的二进制小数位。换言之，每个值都是定点变量。在本节介绍的方案中，每个数值都伴有一个整数，以决定其二进制小数点位置。上述数值可利用这些有关整数的子程序进行处理。这样表示的数称为浮点数。

在用“尾数”和“阶码”表示十进制数的所谓“科学记数法”中，这种概念已经是很熟识的了。阶码是一个十的整数幂，与尾数相乘后给出所表示的数值。因此，3×10^{10} 可代替 30000000000；1.727×10^{-7} 可代替 0.0000001727。在数值范围很大的情况下，采用这种记数法的优点是能避免写出前面或后面很多个零。在这种情况下，这种写法很简短，抄写时不易出错，也容易读。习惯上总是将这些“十进制浮点”数的尾数写成 1～10（或 −10～−1）之内的数，在列表时尾数采用同样个数的有效数字。

4.7.1 表示法与习惯

在计算机中，有若干种不同的表示浮点二进制数的习惯，但是这些习惯都是出于同样的考虑。

数值 x 由一对数 a、b 来表示，使 $x=a\times2^b$。尾数 a 是“规格化”的，即其最高有效数字（正数最左边的一个 1 或负数最左边的一个 0）总是处于其所占用的存贮空间的最左边一位上，阶码 b（总是整数）的值就是根据这一点决定的。从这方面来看，浮点数的表示方法很多（有些较主要，有些较次要），以至无法形成一个“通用”的浮点系统。所以，下面就以 8080 和 8085 中已采用的浮点表示法加以说明。

在这种表示法中，每个浮点数要用四个字节，尾数 a 用三个字节，阶码 b 用一个字节。

尾数表示成一个有 22 个二进制小数位的带符号三字长定点二进制数。除了 $a=0$ 的情形以外，尾数的最高位总是 d_{21} 位。由于在 8080 中，双字长数的高字节总是存在两个单元中地址较高的一个内，所以浮点数的阶码占据四个单元中地址最高的一个单元，尾数占用其次的三个单元。

例如，10 就被表示成：

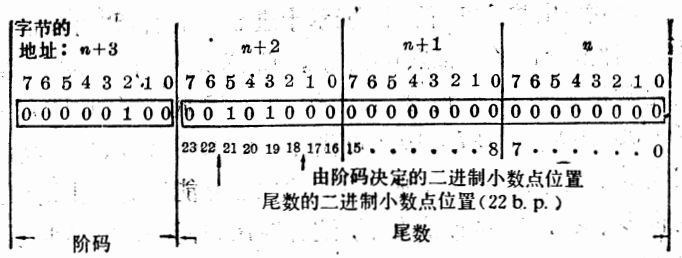


图 4.8

此时 $a=\frac{5}{8}$, $b=4\left(10=\frac{5}{8}\times 2^4\right)$。因此，一般情形是：

若 $x=0$; $a=0$, $b=0$

$x>0$; $\frac{1}{2}\leqslant a<1$, $-128\leqslant b<+127$

$x<0$; $-1\leqslant a<-\frac{1}{2}$, $-128\leqslant b\leqslant+127$

所以，在这种表示方案中，可表示下述范围内的任意 x 值：

$$x=0$$

或 $$2^{-129}<|x|<2^{+127};$$

即 $$1.47\times10^{-39}<|x|<1.70\times10^{+38}$$

其精度为 $\frac{1}{2^{22}}$(十进制有效数字在六位以上)。

变异的表示法

尾数较长或较短将影响精度，而不影响表达的数值范围。32 位尾数约相当于 9 位有效数字，16 位尾数相当于 4 位有效数字。阶码增加到 16 位将使数值范围从约 $10^{\pm38}$ 增大到约 $10^{\pm9864}$。$10^{\pm38}$ 的范围是非常有用的数值范围(虽然偶尔也不够用)；而 $10^{\pm9864}$ 的范围非常大，一般无此必要。有些大型计算机折衷采用十六进制的阶码，因此尾数 a 和阶码 b 不表示 $a\times2^b$ 而表示

$a\times16^b$。这样就将范围扩大到 $10^{\pm154}$ 左右，这个范围在几乎所有的正常情况下都够用。但是，这一改变产生了一个非常不理想的副作用。由于尾数用 16 乘或用 16 除时阶码才能改变 1，所以尾数的最高有效数字可能不是在"通常"的位置上，而在通常位置右边 1 位、2 位或 3 位处。在最后一种情况下，有效位数减少了三位，也即要减少一位十进制有效数字。在尾数为 24 位的系统中，十进制有效数字从 6 位下降到 5 位，精度下降得太过分，这是个严重的缺点。

允许尾数的最高有效数字位于符号位旁边就可多得到一位有效位。然而在执行加、减法时就有可能产生溢出(见下文)。溢出不难用溢出标志来处理，所以增加一点麻烦也是合算的。

表 4.2 列出了选定的一些值的浮点表示法。表中"∧"号指示二进制小数点的位置，是由阶码决定的。

4.7.2 规格化

将一个单字长、双字长或三字长的整数转换成浮点数是相当简单的。转换时，把该整数转换成一个三倍字长的整数，作为尾数；而阶码则规定成 22(十进制)。此时产生的是一个非规格化的浮点数，因此还必须进行规格化。规格化包括将尾数移位，使其最高有效数字处于正确的位置，并相应地调整阶码。规格化的过程如下：

(i) 若尾数为零，则阶码也为零。

(ii) 若尾数的最高位是 01… 或 10…，则尾数必须右移一位，阶码加 1。右移时必须进行四舍五入，可于移位前在尾数的最低位上加 1 来实现③。

(iii) 若尾数的最高位是 000… 或 111…，则尾数必须左移

③ 若加上这个 1 后有可能产生溢出(即要转换的整数是 $(2^{22}-1)$，则必须用某种办法避免这样的四舍五入。

表 4.2

十进制值	十六进制值				二进制值																										注
	阶码		尾数		阶码								尾数																		
	D	E	H	L	7	6	5	4	3	2	1	0	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0	
1	01	20	00	00	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10	04	28	00	00	0	0	0	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
100	07	32	00	00	0	0	0	0	0	1	1	1	0	0	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
1000	0A	3E	80	00	0	0	0	0	1	1	1	1	0	0	1	1	1	1	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	
10000	0E	27	10	00	0	0	0	0	0	1	1	1	0	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	
100000	11	30	D4	00	0	0	0	0	1	0	0	0	1	0	0	1	1	0	0	0	0	1	1	0	1	0	1	0	0	0	0	0	0	0	0	0	
1000000	14	3D	09	00	0	0	0	0	1	0	1	0	0	0	0	1	1	1	1	0	1	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	
10^7	18	26	25	A0	0	0	0	1	1	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	1	1	0	1	0	0	0	0	二进制小数点在尾数各位的右边
10^8	1B	2F	AF	08	0	0	0	1	1	0	1	1	0	0	0	1	0	1	1	1	1	1	0	1	0	1	1	1	1	0	0	0	0	1	0	0	
10^{16}	36	23	86	F2	0	0	1	1	0	1	1	0	0	0	1	0	0	0	1	1	1	1	0	0	0	0	1	1	0	1	1	1	1	0	0	1	0
10^{32}	6B	27	71	6B	0	1	1	0	1	0	1	1	0	0	1	0	0	1	1	1	0	1	1	1	0	0	0	1	0	1	1	0	1	0	1	1	
0·1	FD	33	33	33	1	1	1	1	1	1	0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1	
0·01	FA	28	F5	C3	1	1	1	1	1	0	1	0	0	0	1	0	1	0	0	0	1	1	1	1	0	1	0	1	1	1	0	0	0	0	1	1	
0·001	F7	20	C4	9C	1	1	1	1	0	1	1	1	0	0	0	0	0	0	0	1	1	0	0	0	1	0	0	1	0	0	0	1	1	1	0	0	
0·0001	F3	34	6D	C6	1	1	1	1	0	0	1	1	0	0	1	1	0	1	0	0	0	1	1	0	1	1	0	1	1	1	0	0	0	1	1	0	
10^{-5}	F0	29	F1	6B	1	1	1	1	0	0	0	0	0	0	1	0	1	0	0	1	1	1	1	1	0	0	0	1	0	1	1	0	1	0	1	1	二进制小数点在尾数各位的左边
10^{-6}	ED	21	8D	EF	1	1	1	0	1	1	0	1	0	0	0	1	0	0	0	1	1	0	0	0	1	1	0	1	1	1	1	0	1	1	1	1	
10^{-7}	E9	35	AF	E5	1	1	1	0	1	0	0	1	0	0	1	1	0	1	0	1	1	0	1	0	1	1	1	1	1	1	1	0	0	1	0	1	
10^{-8}	E6	2A	E6	3C	1	1	1	0	0	1	1	0	0	0	0	1	0	1	0	1	1	1	0	0	1	1	0	0	0	1	1	1	1	0	0	0	
10^{-16}	CB	39	A5	65	1	1	0	0	1	0	1	1	0	0	1	1	1	0	0	1	1	0	1	0	0	1	0	1	0	1	1	1	0	0	1	0	1
10^{-32}	26	33	EC	48	1	0	0	1	0	1	1	0	0	0	1	1	0	0	1	1	1	1	1	0	1	1	0	0	0	1	0	0	1	0	0	0	
π = 3.14159	02	32	43	F7	0	0	0	0	0	0	1	0	0	0	0	1	1	0	0	1	0	0	1	0	0	0	0	1	1	1	1	1	0	0	1	1	
$1/\pi$ = 0.31831	FF	28	BE	61	1	1	1	1	1	1	1	0	0	0	1	0	1	0	0	0	1	0	1	1	1	1	1	1	0	0	1	1	0	0	0	0	1
$\pi/180$ = 0.017453	FB	23	BE	8D	1	1	1	1	1	0	1	1	0	0	0	1	0	0	0	1	1	1	0	1	1	1	1	1	1	0	1	0	0	0	1	1	
$180/\pi$ = 57.2958	06	39	4B	B8	0	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	1	0	1	0	0	0	1	0	1	1	1	0	0	0	0	0	
$\log_{10}\pi$ = 0.49715	FF	3F	A2	9B	1	1	1	1	1	1	1	1	0	0	1	1	1	1	1	1	1	0	1	0	0	0	1	0	1	0	0	1	1	0	1	1	
$\ln\pi$ = 1.14473	01	24	A1	A1	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	1	0	1	0	0	0	0	0	1	1	0	1	0	0	0	1
e = 2.71828	02	2B	7E	15	0	0	0	0	0	0	0	1	0	0	0	1	0	1	0	1	1	0	1	1	1	1	1	1	1	0	0	0	0	1	0	1	0
$\log_{10}e$ = 0.43429	FF	37	96	F6	1	1	1	1	1	1	1	1	0	0	1	1	0	1	1	1	1	0	0	1	0	1	1	0	1	1	1	1	0	1	1	0	
$\log_2 e$ = 1.4427	01	2E	2A	8F	0	0	0	0	0	0	0	1	0	0	1	0	1	1	1	0	0	0	1	0	1	0	1	0	1	0	0	0	1	1	1	1	
γ = 0.57722	00	24	F1	1A	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	1	1	1	0	0	0	1	0	0	0	1	1	0	1	0
$\sqrt{2}$ = 1.41421	01	2D	41	3D	0	0	0	0	0	0	0	1	0	0	1	0	1	1	0	1	0	1	0	0	0	0	0	1	0	1	1	1	1	0	1	0	
$\ln 2$ = 0.69315	00	2C	5C	86	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	1	0	0	0	1	0	1	1	1	0	0	1	0	0	0	1	1	0
$\ln 10$ = 2.30259	02	24	D7	63	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	0	0	0	1	1	0	1	0	1	1	1	1	0	1	1	0	0	1

到最高位是 001… 或 110… 为止。并必须从阶码中减去所移的位数。

定点数转换成浮点数的过程与此相同，只是最后要从阶码中减去二进制小数位的位数。

在第八章给出的 8080/8085 程序中，上述第(ii)种情况由入口为 25CB 的子程序完成，第(i)、(iii)种情况由入口为 25A3 的子程序完成。两个程序都将阶码存在寄存器 B 中，尾数存在 AHL 中。AHL 的右移操作(2568—2576)比较困难，但在 6800 和 Z80 中却较简单。

4.7.3 辅助程序

由于浮点数的加、减、乘、除子程序处理的都是三字长数，所以在介绍这些子程序之前，必须先介绍几种对三字长数进行处理的操作。大部分这类操作本身也必须编写成子程序。

4.7.3.1 三字长数的移位

上文考虑规格化时已提及三字长数的算术移位。在 8080/8085 程序包中有下列子程序：

(i) 以 2568 为入口的子程序是一个 AHL 的截尾(丢失最低位)右移子程序。

(ii) 以 2577 为入口的子程序是 AHL 的四舍五入右移子程序(如 § 4.7.2 所述，移位前最低位加 1)。

(iii) 以 2583 为入口的子程序是一个 AHL 的截尾右移子程序，移位数等于寄存器 B 中的整数。

(iv) 以 258B 为入口的子程序是一个 AHL 的四舍五入右移子程序，移位数等于寄存器 B 中的整数(移位前将 $2^B \times d_0$ 加入 AHL，实现四舍五入)。

4.7.3.2 二进制小数位位数相同的三字长数加法

AHL:+CDE 就是 HL:+DE

A:++C

4.7.3.3 二进制小数位位数相同的三字长数减法

在 Z80 中，AHL:－CDE 就是 A:&A(清除 K)

HL:－－DE

A:－－C

但在 8080 和 8085 中所需要的指令较多，必须用一个有 12 个字节的子程序(入口为 25D8)来完成。

4.7.3.4 三字长数的乘法

BCDEHL:＝BHL×CDE(带符号)操作由以 250A 为入口的子程序完成。该子程序又要使用入口为 2400 的双字长乘法子程序与入口为 0F3A 的单字长乘法子程序。在研究入口为 250A 的子程序时，读者应该注意该子程序是如何用堆栈作为暂存器的。

将 48 位的乘积减少到三倍字长 (24 位) 是由入口为 25E4 的子程序完成的。乘法程序假定两个相乘的数都有 22 位二进制小数位，所以 48 位的乘积就有 44 位二进制小数位。将乘积左移两位(形成 46 个二进制小数位)，然后去掉后面 24 位，剩下一个有 22 位(46－24)二进制小数位的三字长数。然而，在最后一步中若去掉的 24 位的最高位为 1，则需在留下部分的最低位上加 1 以实现四舍五入。

顺序使用这两个子程序所产生的结果就是将两个有 22 位二进制小数位的数相乘形成有 22 位二进制小数位的乘积。

4.7.3.5 三字长数的除法

用单字长和双字长除法实现三字长除法不是一种实用的办法。直接进行三字长除法的子程序更现实一些，但这可能不是最好的办法。由于已经有了乘法子程序，为什么不用 x 乘以 y 的倒数形成商 x/y 呢？虽然倒数子程序与一般的除法采用的办

法相似，但是由于被除数是一个"1"后面跟一串"0"，所以不必再用寄存器保存被除数，因此倒数子程序要简单一些。但是这里仍然需要一些技巧，且此时无法用堆栈作为暂存存贮器。

入口为267A的倒数子程序可求得一个无符号（正的）浮点数的倒数，该浮点数的阶码由寄存器D给出，尾数存在EHL中。因此，若该子程序用于 $x=a\times 2^b$，则其结果是将 $(-b-s)$ 存在寄存器D中，将 $2^s/a$（22位二进制小数位）存在EHL中，s 是尾数规格化所需的移位次数。

以26D9为入口的子程序可求出带符号浮点数的倒数。该子程序先用无符号程序求出模的倒数，然后根据需要改变其符号。

4.7.4 浮点子程序概述

8080、8085和Z80有足够数量的内部寄存器，可以相当方便地处理长度为四个字节的数。在这里介绍的浮点"程序包"内，DEHL四个寄存器可实现浮点累加器的功能。D寄存阶码，EHL寄存尾数（最高字节在E中，最低字节在L中）。双字长寄存器BC用作寻址寄存器。

浮点子程序的作用也可用机器指令中所采用的符号系统来描述。例如，若浮点数占用的一组存贮单元是M1234、M1235、M1236和M1237，则可写作MMMM1234或更简单地写成$M^4$1234。阶码将存在M1237中，尾数的最高字节存在M1236中，中间字节存在M1235中，最低字节存在M1234中（该"字组地址"是1234）。

浮点程序包将包括下述子程序：

"存贮"：M^4[BC]:=DEHL

将浮点累加器中的浮点数存入一组单元，字组地址由BC给出（入口为2608）。

“取数”：DEHL:＝M⁴[BC]

与“存贮”子程序的功能相反(入口为 2619)。

“乘法”：DEHL:×M⁴[BC]

浮点累加器中的数与字组地址由 BC 给出的存贮单元中的数相乘，乘积保存在浮点累加器中。BC 中的值保持不变(入口为 264B)。

“倒数”：DEHL:/

浮点累加器中的数被其倒数所取代(入口为 26D9)。

“除法”：DEHL:/M⁴[BC]

浮点累加器中的数除以地址由 BC 给出的单元中的数，商保存在浮点累加器中。BC 中的值保持不变(入口为 26EC)。

“改变累加器中数的符号”：DEHL:－

改变浮点累加器中数的符号(入口为 265B)。

“改变存贮器中数的符号”：M⁴[DE]:－

改变地址由 DE 给出的单元内的数的符号(注意：地址寄存器不是 BC。“减法”子程序要用该子程序，见下文)(入口为 266A)。

“加法”：DEHL:＝M⁴[HL]＋M⁴[DE]

将地址由 HL 与 DE 给出的两组单元内的数之和存入浮点累加器中(入口为 26F7)。(这种作法显然是很笨的，若规定 DEHL:＋M⁴[BC]，则与其它子程序更一致，且易于使用。但这样规定主要是为了避免难以处理的堆栈操作。)

“减法”：DEHL:＝M⁴[HL]－M⁴[DE]

将地址由 HL 与 DE 给出的两组单元内的二数之差存入浮点累加器中(入口为 2731)。

"取数"与"存贮"子程序很简单，要用寄存器对 BC 作为变址寄存器。在这两个子程序的调用过程中，都用堆栈保存 AF 值(这称为该子程序对于 AF 是"透明的")。

"乘法"子程序一开始借助于堆栈将两个尾数分别存入 BHL 与 CDE，形成阶码之和存在寄存器 A 中。然后将阶码之和存入堆栈，并在 AHL 中形成尾数的乘积。再从堆栈中取出阶码之和存入寄存器 B，并对 BAHL 进行规格化。最后将 BA 中的数值送入 DE，因此在 DEHL 中得到浮点乘积。

"倒数"子程序已在§4.7.3.5 中简单作了介绍。

"除法"子程序开始时先将 DEHL(被除数) 入栈。然后将除数取入 DEHL，并用其倒数取代之。最后转入乘法子程序的第四条指令——将被除数乘以除数的倒数。

"加法"子程序相当复杂(有经验的程序员无疑会发现在本书所给出的文本中有修改的痕迹)。其第一部分检查两个尾数的最高字节。若二者之一为零，加法的结果就是另一个数。若二者均不为零，则必须执行加法 (从 26FE 开始的程序)。加法过程是先形成阶码之差 (存在寄存器 B 中)，将要右移的尾数放入 AHL (规格化的数不能左移)。然后 (若右移位数不为零的话)用条件子程序调用指令调用一个子程序进行移位。移位后执行尾数的加法，将适当的阶码存入寄存器 B，并完成全部的规格化过程，最后(结束时)结果存在 DEHL 中。

"减法"子程序开始时先就地改变减数的符号。然后调用加法子程序 (用 $a+(-b)$ 实现 $a-b$)，最后恢复改变了的符号。

4.7.5 浮点输入与输出

浮点读入子程序的功能是接受控制台键盘上以某种形式的十进制浮点表示法打入的数值，并将其转换成规格化的浮点二

进制数存入机器中。与此相反，浮点打印或显示子程序应以某种可接受的十进制浮点数的形式输出给定的规格化浮点二进制数。

十进制浮点数的尾数通常规格化成 $-10\sim-1$（负数）或 $+1\sim+10$（正数）范围内的数，或零。尾数的有效数字不应超过由计算过程（输出时）或原始信息的精度（输入时）所决定的位数。阶码只是一个一位或两位数字的带符号十进制整数。

4.7.5.1 浮点输出

入口为 2B80 的子程序打印出一个有六位有效数字的带符号尾数，并打印一个带符号的一位或两位数的阶码。尾数和阶码之间用字母 E 分开。被打印的数值应以规格化的浮点二进制数的形式存入 DEHL 中。

例如，数值 0.0000610351 将显示成：

$$+6.10551\text{E}-5$$

该子程序的作用如下：先打印尾数的符号，若该数为负，则其符号取反后存在浮点累加器 DEHL 中；若尾数为零，则跳过第二步（2B91～2BDE）；若尾数不为零，则与表示 10（打印十进制尾数的上限）的浮点常数进行比较，若该数大于或等于 10，则除以 10，并使十进制阶码加 1，然后重复这种比较与除 10 操作，直至余数小于 10 为止。这一步骤对原来就小于 10 的数无影响。然后再与 1（数值范围的下限）进行比较。若该数小于 1，则重复乘以 10，直至产生的数大于或等于 1 为止。经过这一步骤的处理就产生一个表示成 1～10 范围内的浮点二进制数的尾数 p，以及一个表示成二进制整数的阶码 q。这样，欲打印的数即为 $x=p\times10^q$。接着是打印尾数。表示 x 的浮点二进制数被简化为四倍字长的定点数，存入 DEHL 中，有 24 个二进制小数位。寄存器 D 存有尾数的整数部分。此时，还要加上一个“十

进制四舍五入”量 $\frac{1}{2}\times10^{-5}$ (十六进制 00.000054)。先打印寄存器 D 的值，然后调用“打印分数”子程序(入口地址为 07E6)，打印十进制小数点和五个十进制小数位。打印时逐次将余数乘以 10(如§4.4 所述)，并取其整数部分作为各次的打印值。最后打印“E”与十进制阶码值，在打印尾数时阶码存放在堆栈中。

这个子程序可能会使电传打字机产生“停顿”。若打印的数的阶码很大(正或负)，则重复进行乘、除法所需的时间可能大于打印机全速工作时，连续两次打印动作之间的间隔时间$\left(\frac{1}{30}\sim\frac{1}{10}\text{秒}\right)$。此时可以注意到在符号与尾数第一个数字之间有一个停顿(浮点乘、除法最长需时 4.5ms)。尾数 1.00000 偶尔会打印成 10.00000。这是由于“最后时刻”的进位使该数超出了正常的范围。这一数值虽然是非“标准”形式，但是打印的值仍然是正确的。

若对该子程序进行改进，例如避免产生“停顿”、预计到最后的进位以及由参数规定打印的有效数字个数等，甚至会使该子程序的长度更长。

用户必须注意，不要认为所打印的尾数中的数字都是有效数字，这些数字并未根据数据及产生这个结果的计算过程进行验证。

4.7.5.2 浮点输入

入口为 2745 的子程序从键盘读入一个数，并将其转换成浮点二进制数存入 DEHL 中。尾数和阶码都可以以标准或非标准的形式打入，但是必须先打入尾数，后打入阶码。例如，“1”可以用下述方式之一打入：

1E0

+1.00000E+00

+0.001E+3

1000E−3

因此，读入的第一个字符可能是符号，也可能是数字。若第一个字符是符号，则该子程序设置一个标记；若为数字，则将该标记置成表示“+”号，并将数字寄存起来。此后根据打印输入，在AHL中形成一个整数。若在任一阶段内（包括在第一个字符处）出现一个点“·”，则对其后出现的位数进行计数。这个整数被转换成浮点数并存入堆栈。然后用普通的“读单字长整数”子程序读入阶码，并从阶码中减去尾数中十进制小数点后的位数。从而得到应该与该浮点整数值相乘的10的幂。子程序中存有10^1、10^2、10^4、10^8、10^{16}、10^{32}的值。然后逐位检查阶码的各位，并根据检查结果使该浮点数乘以或除以10的幂。这是一种与10的幂相乘的简捷办法（见从27AD开始的程序）。

4.7.6 用浮点子程序编制简单的程序

若浮点子程序可供使用，则简单问题的编程工作将减少到只不过是写一系列子程序调用指令。

例 编写一个将华氏温度转换成摄氏温度的程序，转换公式为$C=(F-32)\times\frac{5}{9}$。操作员先打入常数32、5、9，机器就回车、换行。然后打入华氏温度，机器的响应则是打出对应的摄氏温度值，并回车、换行准备读入下一个温度值并进行处理。这个处理程序如下：

```
JS2745
BC:=1300      } 操作员打入“32E0”，机器读入该值
JS2608          并将其存入M⁴1300。
```

JS2745
BC: =1304 } 读入“5E0”并存入 M⁴1304。
JS2608

JS2745
BC: =1308 } 读入“9E0”并存入 M⁴1308。
JS2608

01: JS0200 “回车换行”(见第八章)

JS2745
BC: =130C } 操作员打入华氏温度值，该值被存入 M⁴130C。
JS2608

HL: =130C
DE: =1300 } 机器形成 F−32，并存入 DEHL 中。
JS2731

BC: =1304
JS264B } 机器形成(F−32)×5，并存入 DEHL 中。

BC: =1308
JS26EC } 机器形成 $(F-32)\times\frac{5}{9}=C$，并存入 DEHL 中。

JS2B80
J(L01) } 机器打印 C 的值(无条件转移，准备重复执行该程序)。

从下述附属的说明中可以看出该程序是如何输入计算机，以及如何使用的。监督程序、汇编程序和反汇编程序的具体用法见 § 6.1。

```
  G0C00                                  进入汇编程序
JS2745;          CD    45    27
BC: =1300;       01    00    13
JS2608;          CD    08    26
```

JS2745;	CD	45	27	
BC: =1304;	01	04	13	
JS2608;	CD	08	26	
JS2745;	CD	45	27	
BC: =1308;	01	08	13	当操作员打入每条指令时，机器以十六进制码作为响应
JS2608;	CD	08	26	
01: JS0200;	CD	00	02	
JS2745;	CD	45	27	
BC: =130C;	01	0C	13	
JS2608;	CD	08	26	
HL: =130C;	21	0C	13	
DE: =1300;	11	00	13	
JS2731;	CD	31	27	
BC: =1304;	01	04	13	
JS264B;	CD	4B	26	
BC: =1308;	01	08	13	
JS26EC;	CD	EC	26	
JS2B80;	CD	80	2B	
J(L01);	C3	1B	10	
FINISH;				文本输入结束
MONITOR				
>S1010 C2- 11-				表示程序的长度
>M1180, 11C1, 1000				程序传送至运行时的单元去
>X				

```
NZHPK   A    BC    DE    HL    SP    PC
01110   10   2020  0010  10C0  17F2  0CEF
                   1042  1000  准备执行反汇编程序
>G2800                         进入反汇编程序
1000    CD     JS2745;
1003    01     BC:=1300;
1006    CD     JS2608;
1009    CD     JS2745;
100C    01     BC:=1304;
100F    CD     JS2608;
1012    CD     JS2745;
1015    01     BC:=1308;      反汇编程序的输入,注意:在
                              103F中用绝对地址代替
                              了标号
1018    CD     JS2608;
101B    CD     JS0200;
101E    CD     JS2745;
1021    01     BC:=130C;
1024    CD     JS2608;
1027    21     HL:=130C;
102A    11     DE:=1300;
102D    CD     JS2731;
1030    01     BC:=1304;
1033    CD     JS264B;
1036    01     BC:=1308;
1039    CD     JS26EC;
103C    CD     JS2B80;
```

```
103F    C3      J101B;

MONITOR
>
  D1000, 1041                          十六进制打印输出
1000 CD 45 27 01 00 13 CD 08 26 CD 45 27 01 04 13 CD
1010 08 26 CD 45 27 01 08 13 CD 08 26 CD 00 02 CD 45
1020 27 01 0C 13 CD 08 26 21 0C 13 11 00 13 CD 31 27
1030 01 04 13 CD 4B 26 01 08 13 CD E0 26 CD 80 2B C3
1040 1B 10
>G1000                               进入应用程序
32E0  5E0  9E0                       打入常数
32E0   +0.00000E0
212E0  +1.00000E2
984E-1+3.68888E1                     打入华氏温度值后，机器以对应的
                                       摄氏温度值作为响应，并换一行。
-4E1   -4.00000E1
98.4E0 +3.68888E1
-40E0  -4.00000E1
-459.67E0-2.73150E2
68E0+2.00000E1
50E0+1.00000E1

MONITOR                              复位至监督程序
>
```

练习

1．编写将摄氏温度转换成华氏温度的反变换程序。

2．编写一个计算并联电阻的程序 $\left(\frac{1}{R}=\frac{1}{R1}+\frac{1}{R2}\right)$。该程序能读入 R1 与 R2，并打印出 R 值。

4.7.7 条件标志与浮点数

程序刚从计算一个新的浮点数并将其存入 DEHL 中的子程序转出时，条件标志的值不一定能给出有关这个数的任何有用信息。但是，可以非常简单地设置这些标志。

例如，若 DEHL 中内容为零，则下述指令串将设置"Z"标志；若 DEHL 为负，则将设置"N"标志。

```
A:=E
A:&A
```

因该值为规格化的值，故只需要检查尾数的最高字节。

但是读者应该记得，浮点算术运算与整数算术运算不同，其运算本身就是不精确的，所以计算值恰好等于零通常并没有什么意义。在大部分情况下，所需知道的只是计算值是否足够小，也即是否是在一定允差内的零值。说得更精确些就是，若 $|x|<E$，则认为在绝对误差为 E 的条件下 x 值为零。

所以，若 x 存在 DEHL 中，则可用下述程序进行判断：

```
    A:=E
    A:&A
    JNN(L01)
    JS265B          将|x|存入 M⁴(a)
01: BC:=a
    JS2608
    HL:=a
    DE:=b           形成|x|-E(假定 E 在 M⁴(b)中)
    JS2731
    A:=E
    A:&A
    JNN             x 不为零
```

(*x* 为零)

由于计数基本上是精确的整数值，所以显然不能采用浮点数进行计数。

§4.8 二-十进制(BCD)算术运算

说明"十进制调整"指令时(§2.4.9)已介绍了BCD算术运算，其基础是将一个字节解释成两位十进制数字。0～99范围内的无符号十进制数可以以两个"半字节"(四位字节)的形式存在一个字节内，每个半字节表示一个十进制数字。因此，0～9999范围内的无符号十进制整数可存在相邻的两个字节内。这一数值范围比无符号二进制整数的范围(单字长为0～255，双字长为0～65535)窄得多。

无符号BCD整数的输入与输出显然很简单：可直接使用§4.4末尾给出的那种子程序，唯一必须修改之处是在输入子程序中用"A:=decdig"代替"A:=one hex(一位十六进制数)"。

但是，即使采用了"十进制调整"指令，对BCD数进行算术运算仍然是很麻烦的。这一点在介绍了基本程序之后就会明白。BCD算术运算的实际用途是非常有限的。有些计算器用户知道计算器集成电路是采用BCD算术运算的，可能会对这种说法感到奇怪。但是这样说并没有矛盾。计算器通常需要将其操作过程中产生的每一个数都显示出来：输入的每一个数要显示；每个计算步骤的结果以及每个内部传送操作的结果也必须显示出来。所以简化计算器的BCD输入/输出操作是十分重要的。而在微处理机系统中，要显示或打印的结果很可能是由一长串操作产生的，因此宁可使输入和输出的十-二及二-十进制转换复杂一点，也要采用十分简单的二进制运算，因为这样做要合算得多。因而内部运算一般都采用二进制，偶尔需要进行少

量相当简单的计算时，在微处理机中可方便地用 BCD 算术运算结合简单的 BCD 输入-输出子程序加以实现。

在下述各段中，假定一个有 $2n$ 位十进制数字的十进制整数存在存贮器中一个有 n 个字节的存贮区内。各位数字分别用 a_{2n-1}、a_{2n-2}、…、a_1、a_0 来表示，而各个字节中分别存有 $a_{2n-1}a_{2n-2}$、$a_{2n-3}a_{2n-4}$、…、a_1a_0。最高位数字(10^{2n-1} 的系数)是 a_{2n-1}，最低位数字(个位)是 a_0。

4.8.1 无符号整数的加法

4.8.1.1 单字长加法

若 $a=a_1a_0$，$b=b_1b_0$，则这两个数都可存入一个字节。其和数 $c=a+b$ 处于 0～198(十进制)范围内，表示成 $c=c_2c_1c_0$。其中 c_1c_0 存放在累加器中，c_2(0 或 1)保存在进位标志中。对于本书所叙述的四种微处理机，都可写作：

$$\begin{aligned}&\mathrm{A}:=a_1a_0\\&\mathrm{A}:+b_1b_0\\&\mathrm{DEC}\\&(c_2)c_1c_0:=(\mathrm{K})\mathrm{A}\end{aligned}$$

最后一行表示 A 中存有结果的个位与十位数字，而进位标志 K 中存有百位数字。若 K=0，则结果就落在单字长范围内。

4.8.1.2 双字长加法

在双字长的情况下，高两位数字相加时必须考虑低两位数字相加产生的进位：

$$\begin{aligned}&\mathrm{A}:=a_1a_0\\&\mathrm{A}:+b_1b_0\\&\mathrm{DEC}\\&\left.\begin{aligned}&c_1c_0:=\mathrm{A}\\&\mathrm{A}:=a_3a_2\end{aligned}\right\}\text{这些指令不影响进位标志}\end{aligned}$$

A:++b_3b_2

DEC

$(c_4)c_3c_2$:=(K)A

4.8.1.3 多字长加法

8080、8085和Z80都有寻址寄存器，可用于对存在一组连续的单元内的长整数进行寻址。假设 $a=a_{2n-1}a_{2n-2}\cdots a_1a_0$ 存在 $p\sim p+n-1$（包括该单元）单元内，a_1a_0 存在 p 单元内，a_3a_2 存在 $p+1$ 单元内，$\cdots a_{2n-1}a_{2n-2}$ 存在 $p+n-1$ 单元内（这样安排为的是与这些机器中的实际寻址机构相一致）。同样，假设 b（长度与 a 相同）存在 $q\sim q+n-1$ 中，所需的结果 c 存在 $r\sim r+n-1$ 中。

最好用一个子程序来完成加法。执行该子程序前要求将 a、b、c 的基地址 p、q、r 存入寻址寄存器，而加法长度 n 则存入计数寄存器中。若将调用该子程序的过程写作“long BCD add”，则程序就应编成：

DE:=p

HL:=q

BC:=r

A:=n

long BCD add

而子程序本身可编写如下：

```
    A:&A          (或能清除K的其它适当的指令)
    ST:=AF        (保存字节计数器)
01: A:=M[DE]      (取a的下一个字节)
    A:++M         (带进位加入b的下一个字节)
    DEC           (对部分结果进行十进制调整)
    M[BC]:=A      (存贮结果的下一个字节)
```

```
DE: +1
HL: +1          (寻址寄存器加 1)
BC: +1
HL: = :ST(减计数)   } 这些指令都不影响进位标志
H: −1
HL: = :ST
JNZ(L01)        (重复执行 n 步)
HL: = ST        (清除堆栈)
RET             (K 存有结果中 10^2n 的系数，较低位数
                 字已被存贮起来)
```

在这个程序示例中采用这种笨拙的计数办法以及随后丢失 HL 中的地址是由于“AF: = ST”会破坏“带进位加”所需的进位位内容。A 不可能单独从堆栈中“弹出”来。最好用“SP: −1; SP: −1”来代替“HL: = ST”。

若已知结果在 $2n$ 个数字的数值范围内，则从子程序中转出时可不考虑 K；但是若稍有疑问，则程序应该检查 K，并根据其值执行适当的操作。

因为 6800 不能如此有效地使用寻址寄存器，所以与此子程序对应的 6800 子程序就要复杂得多。在编写该程序时，管理寻址操作的工作比算术运算的工作多得多，这将在叙述非数值操作的第五章中进行讨论。

4.8.2 带符号的 BCD 整数

在讨论减法问题之前，必须先决定在 BCD 数中如何表示负的整数。

与二进制算术运算相似，建议采用“10 的补码”表示法。在这种表示法中，一个字节表示的数值范围不是 00～99 之间的无符号整数，而是 −50～+49 之间的带符号整数。这样，就有可

能形成一种统一的加、减法实施方案；但是乘法用这种表示法时就会变得很复杂。另一种办法是用一个附加位（可能需占用一个半字节或一个字节）来表示100的“借位”。因此，将借位值写在括号内时：

$+23$ 将表示成 (0)23　　$(0\times100+23)$

-23 将表示成 (1)77　　$(-1\times100+77)$

用借位位以及一个字节可表示 $-100\sim+99$ 之内的任意带符号整数。

两个这样的数之和在 $-200\sim+198$ 范围内，而二者之差在 $-199\sim+199$ 范围内。

就字节中的数字而言，用这种方法表示的带符号整数的加法与无符号整数的加法相同，但是还必须对借位进行特殊的处理。字节加法经调整后产生的进位“1”表示 $+100$，借位的每个“1”表示 -100。因此可列成下表。若操作数符号相同，则可能产生溢出，溢出状态由字节加法产生的进位表示。

借位	进位	结果
0, 0	0	正，在范围内
0, 0	1	正，超出范围（溢出）
0, 1 或 1, 0	0	负，在范围内
0, 1 或 1, 0	1	正，在范围内
1, 1	0	负，超出范围（溢出）
1, 1	1	负，在范围内

除非 $x=-100$，否则都有可能求出 x 的负数；当 $x=-100$ 时，$-x$ 超出了单字长的范围。

假设 $-99\leqslant x\leqslant+99$。对于其正半部分 $(+1\leqslant x\leqslant+99)$，$x$ 的值由一个“0”借位与 BCD 数值 x 表示，$-x$ 值由借位位“1”和 BCD 数值 $100-x$ $(+1\leqslant100-x\leqslant+99)$ 表示。

对于负半部分($-99 \leqslant x \leqslant -1$)，$x$ 由借位位“1”与 BCD 值 $100+x$ 表示。$-x$ 的值由“0”借位位及 BCD 数值 $-x$ 表示($-x=100-(100+x)$ 且 $+1 \leqslant -x \leqslant +99$)。

因此在这种情况下，取负可通过将借位值取反，并形成此数 BCD 数值部分关于 100 的补码来实现。下述指令就可形成所需的补码：

A:=99

A:$-x$

A:+01

DEC

由于减法能直接以 BCD 方式给出 $99-x$，所以不需要进行调整。而与 01 相加后的调整步骤将照常执行。

现在还必须考虑 $x=0$ 的情形。此时借位位为“0”，BCD 数字是两个零。借位位取反后变成 1，而 BCD 数字取补将使 K=1，A=00。K=1 可解释为 +100，借位位为 1 表示 −100，显然二者将相互抵消，结果与原值一样($-x=0$)。

在 $x=-100$ 的特殊情况下，原值的借位位为“1”，两个 BCD 数字为零。借位位取反变为 0，BCD 数值取补使 K=1，A=00。结果超出了单字节范围(溢出)。结果的状态是借位位与 BCD 数字表示的数值都为零，而由进位标志表示的数值超出了范围(并代表进位值 +100)。

这样一来，带符号 BCD 值的减法可由取负再相加来实现($a-b \equiv a+(-b)$)。

在 6800、8080 和 8085 中将用下述指令串：

A:=99

A:$-b$

A:+01

```
        DEC
        A:+a
        DEC
```

但是在Z80中由于减法后也可执行十进制调整，所以可简化成：

```
        A:=a
        A:-b
        DEC
```

将这一过程引伸到多字长数的情形与多字长加法的情形大致相同。在上述两个指令串中，除了第一对字节的减法以外，都要用“A:--b”来代替指令“A:-b”。

4.8.3 BCD乘法

考虑下述子程序：

```
        A:=00
01:     B:-1
        RN
        A:+C
        DEC
        J(L01)
```

若进入该子程序时，00≤(B, C)≤09，则转出时00≤A≤81。A是原来存在B与C中的BCD数字之积。

现在再来考虑下述子程序：

```
        HL:=0000
02:     B:-1
        RN
        A:=L
        A:+C
        DEC
```

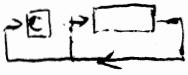
```
L:=A
A:=H
A:++00
DEC
HL:=A
J(L02)
```

若进入该子程序时 00≤B≤09, 00≤C≤99 (都有两个 BCD 数字),则转出时 0000≤HL≤0891。HL 就是 B 中的 BCD 数字与 C 中的无符号两位数 BCD 整数之积。

也可用第二个子程序来形成给定于 B、C 中的两个无符号两位的 BCD 整数之积。

将调用该子程序的过程写作 HL:=B_0×C(BCD), 则下述子程序将产生给定于 B、C 中的两个无符号两位的 BCD 整数的四位 BCD 乘积,存在 HL 中。

```
D:=B
A:=B
A:&F0
A:@R
A:@R
A:@R
A:@R
B:=A
HL:=B0×C(BCD)
HL:+HL
HL:+HL
HL:+HL
HL:+HL
```



```
A:=D
HL:=:DE
A:&0F
B:=A
HL:=B0×C(BCD)
A:=E
A:+L
DEC
L:=A
A:=D
A:++H
DEC
H:=A
RET
```

调用该子程序的过程可写作 HL:=B×C(BCD)。

显然，借助于§4.5.4提出的办法可用该子程序形成多位BCD整数的乘积。

这个子程序也可用于带符号数的乘法，所需的符号修正操作与二进制乘法中的相似（参阅§4.5.3）。在相乘的两个数中若有一个为负，则应从乘积的高半中减去另一个数。

上述无符号BCD乘法子程序需要50个字节，能将两个0～99范围内的整数相乘，形成0～9801范围内的乘积。

与此相比，第八章中入口为2400的无符号二进制乘法子程序DEHL:=BC×DE只占用32个字节，能将两个0～65535范围内的整数相乘，产生0～4294836225范围内的乘积。

如果读者仍然不认为BCD算术运算是麻烦的，则可编写一个BCD乘法子程序，使其操作数范围与这个二进制乘法子程序

相当。

4.8.4 BCD 小数

与定点二进制操作 (§ 4.6) 相似，在 BCD 算术运算中也可考虑十进制小数点的位置。在加、减法中，执行加、减操作之前必须先对准假想的十进制小数点的位置。若十进制小数位的位数相差偶数个，则准备操作必须保证每一步都正确地对相应的各对字节进行寻址。但是，若十进制小数位的位数相差奇数位，则必须完成将 BCD 数之一左移四位的棘手任务。这个任务在带"半字节移位"指令的 Z80 上也许是容易的，但是在其它机器上就很麻烦。

注意：计算总金额时最好不要以有两位十进制小数位的镑(元、先令等)为单位，而应以整数"新便士"(分、分尼等)为单位。这样，在简单的"借贷"计算中就可使用 BCD 算术运算。但是，只要涉及"百分比计算"形式的乘法(股份、营业税、价值附加税、折扣)，则无疑会发现二进制算术运算比 BCD 算术运算功能强得多，而且方便得多。截尾或四舍五入成整数值的作用与数制无关。将有 p 个二进制小数位的二进制量四舍五入成 d 个十进制小数位，可近似地通过在该数上加 $\frac{1}{2}10^{-d}2^{p}$ 来实现。

第五章　非数值操作的程序设计

§5.1　非 数 值 量

在有关指令与算术运算的章节里我们遇到过许多主要不是作为数值使用的量。也遇到过许多不属于算术范围的操作。例如，属于前者的有指令、地址、标志的状态和表示字符的 ASCII 码；属于后者的有代码转换、为进入一个子程序而准备地址以及区分数字与字母的代码(§4.3 练习 2)。

第八章给出的软件中相当大的一部分是"汇编程序"和"反汇编程序"，这两个程序的功能是在两种"语言"之间进行"翻译"，即在为便于人们理解而引入的记号与微处理机所需的二进制数字串之间进行"翻译"。这些程序虽然也使用了许多算术指令，但只是偶尔才涉及算术操作与数值。其计算工作只是反复地确定一种"语言"中的哪一串二进制数字(或符号)与另一种"语言"中的哪一串符号(或数字)相对应。

但是，不应过分地强调或误解"数值"计算与"非数值"计算之间的区别。这种区别只须记在程序员和用户的脑子里就行了；二者在机器里都是一串二进制数字。进入指令寄存器的二进制数字串就是指令；系统里其它的二进制数字串只表示它们约定的意义。通常认为"数据处理"或"信息处理"对于程序员和电子电路来说在某种程度上比单纯数值计算更高级，这种概念是错误的。

§5.2　复制成组的信息

§2.10 介绍的 Z80 "字组复制"指令能将成组的信息(即占

据地址是连续的存贮单元的一组字节)从存贮器的一个部分复制到另一个部分(当然必须是 RAM)去。

其作用(ED B0 是加, ED B8 是减)可写作:

repeat M[DE:±1]:=M[HL:±1],
BC:−1**until** BC:=0000;

若在其它处理机里需执行同样的操作, 则必须编写一段程序。即使在 Z80 上也应该知道若这个操作需要改变一下的话, 应如何用简单的指令编写这样的程序。

在 8080 与 8085 上, 可使用与 Z80 中相同的寄存器。直接翻译成 8080 或 8085 子程序就是:

```
        (ST:=AF)
01:     A:=M
        HL:±1
        M[DE]:=A
        DE:±1
        BC:−1
        A:=B
        A:UC
        JNZ(L01)
        (AF:=ST)
        RET
```

这里首先要注意的是, 8080 与 8085 没有直接的存贮器至存贮器复制指令可供使用, 必须用寄存器 A 作为"中间手段"。堆栈操作被编成任选指令, 以便使子程序对于 A 是"透明的"(即: 使子程序执行前后寄存器 A 的内容保持不变)。"加 1"与"减 1"指令不影响任何条件标志, 所以需用括号内的三条指令来测试计数器是否已减到零了, 这当然是个笨办法(若 B 与 C 中都没

有"1"了，则 A 将被清除)。

只有当字组长度可能超过 256 个字节时，才需使用双字长寄存器作为计数器。若不会超出这个范围，则使用单字长计数器就够了，且括号内的三条指令只要由 B:−1 来代替即可。

由于 6800 只有一个寻址寄存器 IX，所以很难将这个程序翻译成 6800 程序。但是，在满足下述限制的情况下，则子程序可能相当简单。这种限制就是源与目的地两个"起始地址"的间隔应小于 256 个字节，且要复制的信息不超过 256 个字节。此时，可采用下述程序：

```
        (S:=A)
01: A:=M00X
    M(j)X:=A    (j)表示一对十六进制数字
    IX:±1
    B:−1
    JNZ+F8      (转至 L01)
    (A:=S)
    RET
```

这段程序中假定第一个"目的"地址的值比第一个"源"地址的值大 j。若情况与此相反，则其中的 00 与(j)字节必须互换。在前一种情况下，进入该子程序前必须将第一个源地址存入 IX；在后一种情况下，则将第一个目的地址存入 IX。由于在这两种情况下都必须将(j)"置入"子程序体(这是一种很拙笨的编程技术，出现得比磁芯还早)，只有该子程序(或其中一部分)存在 RAM 中时才有可能采用这种技术，所以这个子程序是不够满意的。

需要作相当多的努力才能充分利用 Z80 的技术性能。由于实际工作中要用三个双字长计数器，所以必须占用 RAM 的六

个字节作为暂存空间。在进入子程序之前，必须将第一个源地址存入 MM(source)，将第一个目的地址存入 MM(destination)，并把要复制的字节数存入 MM(count)。

```
        (S:=A)
01: IX  =MM(source)
    A:=M00X
    IX:±1
    MM(source):=IX
    IX:=MM(destination)
    M00X:=A
    IX:±1
    MM(destination):=IX
    IX:=MM(count)
    IX:-1
    MM(count):=IX
    JNZ+E4  (至 L01)
    (A:=S)
    RET
```

是采用 Z80 的"字组复制"指令还是采用这些子程序之一，必须在仔细考虑了下述重要问题之后才能决定。

若源字组与目的字组是不交叉的(即没有公用的单元)，则就不会产生什么问题。但是若二者重迭，则在重迭区内的单元(作为信息源)被复制之前有可能(作为目的地)被写入其它信息。

图 5.1 之类的图形能清楚地说明这一点。在(I)情况下，若复制过程是递增进行的，则(a)与(b)之间的源信息在复制之前就会丢失；但是若递减进行，则整个操作将正确地完成。在第

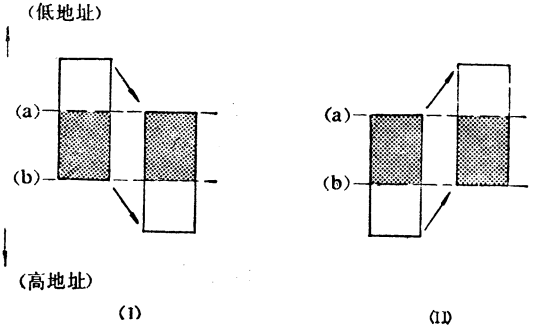


图 5.1

(II)种情况下,递增过程将产生正确的结果,而递减过程将出问题。

获得正确结果的一般规律(无论是否重迭)如下所述。若最低源地址高于最低目的地址,则采用递增法——向前。若最低源地址低于最低目的地址,则采用递减法——向后。在后一种情况下,复制过程应从最高源地址与最高目的地址开始进行。

进行地址比较时必须记住,地址是无符号双字长整数。写作"IX-*q*"的6800操作可用于测试地址是否相等,但是不能直接用于其它比较操作(见§2.4.7)。

监督程序的"字组复制"操作在第八章进行介绍;占用的单元是0180～01E5。

§5.3 检索表格

Z80具有"字组检索"指令ED B1(+)与ED B9(−),其作用是(见§2.10):

repeat A−M[HL:±1], BC:−1**until** BC=0000∨Z;

基本作用相同的8080～8085子程序是:

```
    ST:=DE
    D:=A
01: A-M
    HL:±1
    BC:-1
    JZ(L02)
    A:=B
    A:UC
    A:=D
    JNZ (L01)
    K:=1
02: DE:=ST
    RET
```

在转出该子程序时，Z=0。若 K=0，则已找到其值与 A 中的值相等的元素，其地址是 HL 中的值减一。若 K=1，则未找到所需的元素，且 BC=0000。

然而与 Z80 指令的作用稍有不同而更完善的子程序如下述(a)所示。此时，若转出时 K=0，则 HL 中存有所需元素的地址。如果不要求该子程序对于 D 是透明的，则可使用较短的子程序(b)；若字组长度不会超过 256 个字节，则可使用更短的子程序(c)。

```
    ST:=DE          D:=A        01: A-M
    D:=A        01: A-M             RZ
01: A-M             RZ              HL:±1
    JZ(L02)         HL:±1           B:-1
    HL:±1           BC:-1           JNZ(L01)
    BC:-1           A:=B            K:=1
```

```
A:=B          A:UC          RET
A:UC          A:=D
A:=D          JNZ(L01)
JNZ(L01)      K:=1
K:=1          RET
02: DE:=ST
RET
(a)           (b)           (c)
```

上述子程序(c)几乎可直接翻译成 6800 程序：

```
01: A-M00X
    JZ+05      (至 L02)
    IX:±1
    B:-1
    JNZ+F8     (至 L01)
    K:=1
02: RET
```

若字组长度超过 256 个字节，则须用下述程序：

```
    MM(address):=起始地址
    MM(count):=字组长度
01: IX:=MM(address)
    A-M00X
    JZ+(至 L02)
    IX:±1
    MM(address):=IX
    IX:=MM(count)
    IX:-1
    MM(count):=IX
```

JNZ+(至 L01)

K:=1

02: RET

但是，将字组分割成包含的字节数在256个或更少一些的段落，从而避免使用该子程序可能更可取些。

求出一个字组中数值最大或最小的元素，是经常需要进行的一种操作。例如，若一个字组中的元素是用ASCII码表示的字母，要求按字母顺序排列或打印这些字母，则在这个字组中检索到的最小元素就是新表格的第一个元素或要打印的第一个字母。按递减顺序排列各数的问题同样要从检索最大元素开始。

寻找一个字组中最小元素的过程可表示如下：

$m:=m0$;

for $i:=1$ **to** n **do**

if $a_i<m$ **then begin** $m:=a_i$; $j:=i$ **end**

其中 $m0$ 是大于字组中每一个元素的任意数；n 是字组中的元素数目；a_i 是字组中的第 i 个元素。这个过程将逐个检查各个元素；若发现一个元素小于 m 中的值，则用该值取代 m 值，并将其位置记录在 j 中。因此，结束时 m 就存有最小元素的值，其位置由 j 给出。

假设这些元素都是大写字母的ASCII码，则其范围在（十六进制）41(A)～5A(Z)之间。第一个元素的地址将存在IX(6800)或HL(8080、8085、Z80)中，元素个数存在寄存器B中。结果的值将存在寄存器A(6800)或C(8080、8085、Z80)中，这个最小元素的地址存在MM(address)或寄存器对DE中。若具有最小值的元素有一个以上，则转出时的地址是第一个被发现的具有最小值的元素的地址——即这些具有最小值的元素中地址值最低的那个地址。在6800程序中，将"JKZ"改成"JK"将

6800	8080, 8085, Z80
A:=5B	C:=5B
01: A−M00X	01: A:=M
JKZ+05 (至 L02)	A−C
A:=M00X	JNK(L02)
MM(address):=IX	C:=A
02: IX:+1	D:=H
B:−1	E:=L
JNZ+F3 (至 L01)	02: HL:+1
RET	B:−1 JNZ(L01) RET } 在 Z80 中是一条指令

给出这些元素中地址值最高的那个地址。若要求(或允许)这样做,则下述 8080、8085 和 Z80 子程序将缩短一个字节并可避免使用寄存器 C:

```
    A:=5B
01: A−M
    JK(L02)
    A:=M
    D:=H
    E:=L
02: HL:+1
    B:−1
    JNZ(L01)
    RET        (结果在 A 中,地址在 DE 中)
```

在这个程序中应该注意的是,m 的最初值也可以是 5A,即恰好是可能达到的最大值。因此,若字组中每个字母都是 Z,则转出时寄存器 A 中将存有 5A(Z),寄存器对 DE 中将存有字组中最

后一个元素的地址。

搜寻字组中的最大值的过程与此相仿，只是 m 的初值应该小于或等于(取决于所用的文本)元素的最小可能值。

在这个例子中，由于元素的值是无符号整数，所以用进位标志 K 作为判别标志。在元素是带符号数的情况下，则数值比较的方法以及随后对标志值的用法都必须考虑 §2.4.7 所提及的各点。

若要检查的元素不是存在连续的单元内，而是等间距地相隔不多几个字节，则对这些子程序进行简单的修改，在已有的加1指令后增加若干条加1指令(IX:+1 或 HL:+1)就可适用。

若元素具有多倍字长，则显然必须预置多倍字长的初始值，并执行多倍字长比较操作(见 §4.5.2)。

§5.4 排　　序

排序问题本身就是一个相当大的课题，已出版了几本有关的教科书①。这里只能简要地论及几种最简单和最有用的方法。

若已具备上节介绍的子程序，则直接使用这些子程序，或以其思路作为线索显然是值得的。

按递减的顺序打印出一组存在连续单元内的计算值就是一个相当简单的排序问题。解决这个问题的过程可表示如下：

repeat (找出组中最大值；
打印最大值；
从组中删除刚打印的值) **until** 全组数值均打完。

① Robert P. Rich, Internal Sorting Methods Illustrated with PL/I Programs. Englewood Cliffs NJ: pretice-Hall Inc., 1972。

若没有"删除"这一步，则该过程将无休止地重复打印该组元素的最大值。究竟用何种办法实现删除取决于：(a)表格是否仍需保存在存贮器里，供进一步计算用；(b)元素值的范围。

若打印后表格毋须保留，则只要用一个超过 m 的初始值的数值取代一个元素就可将其删除。这样可保证该元素不会再次被选中打印。若无超出 m 的初始值的数值可用，则办法之一是用这个字组中的最后一个元素值取代刚打印的元素值，并将字组长度减一。

若表格打印完后仍需保留，则删除方法主要有两种。若有足够的 RAM 可供使用，则可将表格复制到 RAM 的空单元中去，并根据复制的表格进行打印。另一种办法是打印一个元素后在其上作一个"标记"(而不是"删除")，并对检索例行程序进行修改，使其不再打印带标记的元素。若元素中有未用的位可容纳标记，就可采用标记法；打印完以后，可能还须将标记除去。

在任何情况下，每个步骤都必须执行 n 遍，每遍都是打印剩余部分的最大值。因此，整个过程将具有计数循环的形式，从 n 开始递减计数。这个计数循环向"求最大值"子程序提供起始地址及(剩余的)表格占用的字组长度，从而为执行下一步作准备。

若要求把按序排列的表格存在 RAM 的另一部分里，采用这些技术也很简单。唯一需修改之处是程序中不调用打印子程序，而用一串指令将有关的值填入一张表格中去，使其成为该表格的一个新元素。

但是较常提出的要求也许是在原有的空间内对表格进行排序，即不用任何"便笺式"暂时存贮器。有关排序的教科书中讨论了具有不同的速度、长度与风格的几种排序办法。下面讨论的方法并不是最快的，但肯定是最简单的办法之一。

这种算法称为"希氏排序"法(Shellsort)，是 J. Boothroyd

以 D. A. Shell② 的工作为基础研制出来的一种 ALGOL 60 算法。这一过程的基本部分的文本如下(不熟悉 ALGOL 60 的读者可跳过这一段，但是在读完了本节其余部分以后再回过来看这一段程序，理解起来就不会感到太难了)：

```
procedure Shellsort(a, n); value n; real array a;
  integer n;
        (注释: 按递增的顺序排列数组 a 中的 n 个元素)
  begin integer i, j, k, m; real w;
        m:=n;
        for m:=m÷2 while m≠0 do
          begin k:=n−m;
            for j:=1 step 1 until k do
              begin for i:=j step −m until 1 do
                begin if a[i+m]≥a[i] then go to L1;
                      w:=a[i]; a[i]:=a[i+m]; a[i+m]:=w
                end i;
              L1: end j;
          end m;
        end Shellsort;
```

注释：按递减顺序排列时只须用“≤”代替“≥”即可。

描述这个过程的 ALGOL 语句与本书所介绍并使用的表示法之间的“文字”对应关系如下：

② J. Boothroyd,“Algorithm 201”。Communications ACM, 6, 445 (1963); D. A. Shell,“ A high-speed sorting procedure”, Communications ACM, 2, 30 (1959)。

	begin integer i, j, k, m; **real** w;
$m:=n$	$m:=n$;
06: $m:/2$	**for** $m:=m\div 2$
JZ(L05)	**while** $m\neq 0$
$k:=n-m$	**do begin** $k:=n-m$;
$j:=1$	**for** $j:=1$ **step** 1
04: $k-j$	**until** k
JN(L03)	
$i:=j$	**do begin for** $i:=j$ **step** $-m$
02: $i-1$	**until** 1
JN(L01)	
$a_{i+m}-a_i$	**do begin if** $a_{i+m}\geqslant a_i$
JNN(L01)	**then go to** $L1$;
$a_{i+m}:=:a_i$	$w:=a_i$; $a_i:=a_{i+m}$; $a_{i+m}:=w$
$i:-m$	**end** i;
J(L02)	
01: $j:+1$	$L1$: **end** j;
J(L04)	
03: NULL	**end** m;
J(L06)	
05: RET	**end** Shellsort

实际供微处理机使用时，可将上述程序缩短成更符合“习惯”的文本：

步骤	
1	$m:=n$
2	06: $m:/2$
3	RZ

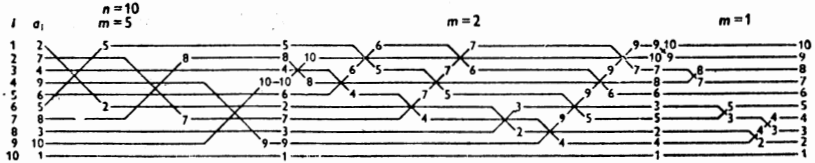
步骤	
4	$k:=n-m$
5	$j:=0$
6	01: $j:+1$
7	$k-j$
8	JN(L06)
9	$i:=j$
10	02: $i-1$
11	JN(L01)
12*	$a_{i+m}-a_i$
13	JNN(L01)
14	$a_{i+m}:=:a_i$
15	$i:-m$
16	J(L02)

* （若按递减顺序排列，则只须用 a_i-a_{i+m} 代替这条指令）

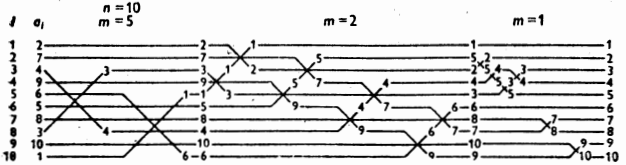
若给定 n 个元素 a_1, …, a_n, 上述过程将使其按递增的顺序排列，因此转出这段程序时就有 $a_1\leqslant a_2\leqslant a_3\leqslant\cdots\leqslant a_n$。若根据注释进行修改，就会按递减顺序排列，因此转出时就有 $a_1\geqslant a_2\geqslant a_3\geqslant\cdots\geqslant a_n$。

图 5.2 说明了这段程序对数值较小的整数组成的简短的表进行处理的全过程。处理工作的核心是将一对元素的值进行比较；若二者的排列顺序不正确，则相互交换位置。在第一阶段，选择进行比较的元素是相隔 $n/2$ 个单元的元素（这里"/"表示截尾整数除法，所以 10/2=5, 5/2=2 等等），对所有这种"元素对"都进行比较。在下一阶段中，间隔单元数减半（$m:/2$）。因此一个数值经过 m 数不同的几个步骤就可从其开始时所处的位置移到表格的顶部。这一工作是通过将其与每步末尾的元素进行比

(i) 将a={2,7,4,9,6,5,8,3,10,1 } 按递减顺序进行排列。



(ii) 将a={2,7,4,9,6,5,8,3,10,1 } 按递增顺序进行排列。



(iii) 将a={7,2,4,5,7,1,1,8,7 } 按递减顺序进行排列。

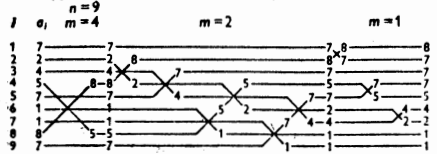


图5.2 “希氏排序”法的应用实例

较并交换，直至遇到与其相比排列顺序正确的元素为止。在最后阶段 $m=1$；经过这一阶段的处理后，程序结束且所有的元素将按要求的顺序进行排列。

读者将发现，用电话号码之类的数字在纸上亲自对这一过程的整个步骤进行处理练习是很有益处的。

在将任何这样的过程编成机器码之前，必须考虑到变量可能达到的数值范围。在 8080/8085 用的第一个文本中，采用了使子程序最简单的数值范围。

首先，假设要进行排序的数据项是无符号单字长整数。每一项占据一个字节，所以整个表格占用 RAM 中 n 个字节组成的一个区域。n 值必须是正值；从第 1～8 步可看出，m、k、j 也都是正的，且数值不会超过 n 值。在 12～14 步中，i 必须在 1～$n-m$ 范围内，即 i 为正值且小于 n。但是第 15 步会产生一些小问题，这一步可能使 i 变负(若出现这种情况，则在第 11 步时会发生转移)。因此考虑到 i 的全部取值，则 i 是范围为 $-(n/2)\sim+n$ 的带符号变量。若将 i 看作为带符号单字长量，则 n、m、k、j 都是正的，而且特别是 n 不应超过 128。

m、n、k、j、i 可分别安排在寄存器 B、C、D、E、H 中。这个程序中最棘手的部分将是第 12 步与第 14 步，在这两步中要计算地址。暂时不考虑这一点，则将其余部分转变成机器码不会有什么困难。

步骤		
1	$m:=n$	B:=C;
2	06: $m:/2$	06: A:=B; A:&FE; A:@R; B:=A;
3	RZ	RZ;
4	$k:=n-m$	A:=C; A:−B; D:=A;
5	$j:=0$	E:=00;

步骤		
6	01: j: +1	01: E: +1;
7	$k-j$	A: =D; A−E;
8	JN(L06)	JN(L06);
9	i: =j	H: =E;
10	02: i−1	02: H: −1;
11	JN(L01)	JN(L01); H: +1;
12	$a_{i+m}-a_i$	$a_{i+m}-a_i$;
13	JNN(L01)	JNK(L01);
14	a_{i+m}: = :a_i	a_{i+m}: = :a_i;
15	i: −m	A: =H; A: −B; H: =A;
16	J(L02)	J(L02);

应当注意的是，在机器码文本的第13行，用标志K(进位)而不用标志N(负)来表示两个无符号整数的比较结果(见§2.4.7)。其余部分需要仔细对待。在8080上最好的办法也许是在执行寻址操作时用堆栈保存信息。如果这样做的话，可将a_1的地址存入HL，将i存入A，将m存入B。因此，可写成：

12	HL: −1;	(设想的)a_0的地址
	C: =A;	i
	A: =B;	m
	B: =00;	
	HL: +BC;	a_i的地址存入HL
	D: =H	
	E: =L;	a_i的地址存入DE
	C: =A;	m
	HL: +BC;	a_{i+m}的地址存入HL
	HL: = :DE;	地址交换(见程序文本)

```
        A:=M[DE];       a_{i+m}
        A-M             a_{i+m}-a_i
13      JNK(L01);
        B:=A;
        A:=M;
        M[DE]:=A;
        M:=B;           a_{i+m}:=:a_i
```

最后必须将这一段指令填入上述程序。在执行这个操作时产生的麻烦是在进入该子程序时基地址(a_1 的地址)应在 HL 中，并必须保存起来以供每次执行上面给出的这段指令时使用。图 5.3 是一种可实现的子程序的全译码文本，其入口是 1000。

进入该子程序时 n 值应存入 C，a_1 的地址存入 HL，该子程序就将 a_1，…，a_n 的值按递增的顺序进行排列。若用空操作(NULL)代替交换指令 HL:=:DE，则修改后的子程序就将这些值按递减顺序排列。

这个子程序可以有许多变化形式，其中有一些可供读者作为练习，例如：

(i) 表格的长度

在 100F 与 1015 处采用条件转移指令就可使该子程序处理元素数多达 255 个的表格。100F 处可用"若有进位则转移"指令代替，另一个单元(1015)应该用什么指令代替呢？完全采用双字长操作当然就能对更长的表格进行排序。

(ii) 其它类型的表格

(a) 若表格中的元素是带符号单字长整数，则 $a_{i+m}-a_i$ 之差可能超出单字长范围，因此 102A 处的比较操作 A－M 就不适用了(见§2.4.7和§4.5.2)。

(b) 若表格中的元素是双字长数，则地址计算、元素值比较

```
G2800                                   G2800
1000    41    B:=C;                     1020    78    A:=B;
1001    78    A:=B;                     1021    06    B:=00;
1002    E6    A:&FE;                    1023    09    HL:+BC;
1004    0F    A:0R;                     1024    54    D:=H;
1005    47    B:=A;                     1025    5D    E:=L;
1006    C8    RZ;                       1026    4F    C:=A;
1007    79    A:=C;                     1027    09    HL:+BC;
1008    90    A:-B;                     1028    EB    HL:=:DE;
1009    57    D:=A;                     1029    1A    A:=M[DE];
100A    1E    E:=00;                    102A    BE    A-M;
100C    1C    E:+1;                     102B    D2    JNK1032;
100D    7A    A:=D;                     102E    47    B:=A;
100E    BB    A-E;                      102F    7E    A:=M;
100F    FA    JN1001;                   1030    12    M[DE]:=A;
1012    E5    ST:=HL;                   1031    70    M:=B;
1013    63    H:=E;                     1032    C1    BC:=ST;
1014    25    H:-1;                     1033    D1    DE:=ST;
1015    FA    JN103F;                   1034    E1    HL:=ST;
1018    24    H:+1;                     1035    E3    HL:=:ST;
1019    7C    A:=H;                     1036    D2    JNK103F;
101A    E3    HL:=:ST;                  1039    7C    A:=H;
101B    E5    ST:=HL;                   103A    90    A:-B;
101C    D5    ST:=DE;                   103B    67    H:=A;
101D    C5    ST:=BC;                   103C    C3    J1014;
101E    2B    HL:-1;                    103F    E1    HL:=ST;
101F    4F    C:=A;                     1040    C3    J100C;
MONITOR
>D1000, 1042
1000 41 78 E6 FE 0F 47 C8 79 90  57 1E 00  1C 7A BB FA
1010 01 10 E5 63  25 FA 3F 10 24  7C E3 E5  D5 C5 2B  4F
1020 78 06 00 09  54 5D 4F 09 EB 1A BE D2  32  10 47  7E
1030 12 70 C1 D1  E1 E3 D2 3F 10  7C 90 67  C3  14 10  E1
1040 C3 0C 10
>
```

图 5.3

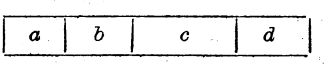
和交换操作都必须修改。

(c) "名称"是一种可能需要排序的重要的多字长量。"名称"是由数量不同的字母组成的，要求按字母顺序进行排列。较方便的办法是为每个名称保留一个小字组(例如每个字组有十六个字节)。因此，每个名称应该在字组内"向左调整"——即第一个字母应存入"最高字节"；而不用的字节排在右边(直至"最低字节")，并填入空格码(ASCII码20)。对按这种方式存贮的名称进行比较的比较操作就完全与多字长无符号整数的比较操作一样。实际情况是每个名称都可能是一个记录的一部分(见§5.6)，要求的就是对这些记录进行排序。若这些记录占用同样长度的存贮区域，则各个名称之间的间距就相等。因此能规则地对名称进行寻址；比较操作只涉及名称，但是交换操作必须对两个完整的记录进行交换。

§5.5 "组装式"信息——长度小于一个字节的量

几乎在每个应用问题中都有一些数值范围非常有限的量，保存这些信息所需的位数还不到一个字节。四位二进制数字组成的BCD数字就是"人为的"这样一种量，两个这样的量可以存在一个字节内，§4.8详细讨论了BCD数。有关逻辑条件的例子就更"自然"了，每个逻辑条件只需要一位。最重要的一些逻辑条件保存在条件标志内，并把几个条件标志放在一起形成一个标志寄存器。此外，一个特定的程序还可能涉及其它的条件或只有一位的变量，这些条件和量需作为普通字节的一部分进行存贮。这种情形我们至少已遇到过两种，一种是在读入一个输入量的数值时用一个标志表示该数的符号(§4.4)，另一种是用一个标志注出正在进行某种处理的表格中已处理过的项(§5.3)。

假设有四个量 a、b、c、d，分别有 1、2、3、2 位。如下所示，将它们“组装”成一个字节也许是值得的：



信息的“组装”与“拆散”可用下述指令串完成：

```
A:=a                    B:=A
2×A:+A(2×A:*2)          A:&03
A:+b                    d:=A
3×A:+A                  A:=B
A:+c                    A:&1C
2×A:+A                  2×A:@R(2×A:->)
A:+d                    c:=A
                        A:=B
                        A:&60
                        3×A:@L(5×A:->)
                        b:=A
                        A:=B
                        A:&80
                        A:@L(2×AK:@L)
                        a:=A
```

左边的一列指令将 a、b、c 与 d 组装成一个字节存在寄存器 A 中；右边的一列将 A 中的一个字节拆散成四个部分。指令左边的“2×”、“3×”、“5×”等记号表示该指令应该写 2、3 或 5 遍。括号中的指令是在本例中具有同样作用的另一种方法。

在监督程序(§ 8.2.3)中，X 指令的处理分支必须将标志寄存器的内容拆成五个一位的值，而且还必须能将五个给定的位

组装成一个字节以便送入标志寄存器(024D－0263，0312－031E；0296－02AB，031F－032C)。

反汇编程序(§8.4)将被处理的程序中的每一个操作码字节看作是一个"组装式"字节。从第三章中有关8080操作码的卡诺图中可清楚地看出，8080操作码主要可分成下述类别：

01……	寄存器间传送，$r_1:=r_2$(除01110110外)
10……	累加器操作，A—操作码—寄存器
11…110	累加器操作，A—操作码—J字节
11…0.1	条件转移与返回
11…100	条件调用
11…0.01	堆栈操作
11…111	特殊子程序调用
00…0.1	双字长寄存器操作
00…10.	单字长寄存器操作
00…110	单字长寄存器操作
00…010	存取操作
00…111	移位及其它操作
11…1.01	其它操作
11…011	其它操作
00…000	空操作或未定义的操作

反汇编程序可根据操作码"10010101"(95)的最左边两位识别出该操作是累加器操作。然后分离出"操作码字段"(接下来的三位，"010"表示":－")与"寄存器字段"(最后三位，"101"表示"L")。反汇编程序从一张简单的表格中查出各项内容，然后顺序打印出"A"、":－"、"L"，整个指令就是"A:－L"(参阅§8.8中28A3以下的程序)。

与此相反，若给出"A:－L"，则汇编程序必须将与"A"(累

加器操作)、":－"(减法)、"L"(寄存器)对应的元素组装在一起,以形成目的程序字节"10010101"(参阅§8.8中04A6以下的程序)。

§5.6 记录与文件

组装在一起的信息常常成为文件中每个记录的一部分。文件的概念来源于簿记学。在簿记学中,文件表示形式相同的一组记录,每个记录与一个特定的帐目有关。在计算机里,"文件"具有更广泛的意义,是指一组信息(称为"记录"),每组信息具有同样的"结构",即由数量相同的信息"项"构成或由以同样的方式相互关联的信息段构成。

从这个意义上来说,目视显示部件(v.d.u.)显示的一个图形就是一个简单的文件,图形的每一点对应于一个由一对座标值组成的记录,也许每个座标占用一个字节。在更复杂的文件中,记录可能是仪表的一组读数(例如时钟时间、各种温度、压力等等),所以这个文件就表示了物理过程的变化"历史"。

文件的"处理"一般意味着顺序对其每一个记录执行同样的操作("顺序处理")或对按某种计算出来的顺序选中的一组记录执行同样的操作("随机处理")。在上述图形的例子中,通过对这些记录执行同样的变换就可实现图形的变换、旋转或比例变换。在上述"物理过程"的例子中,可根据满足某种判据的记录作出人们所关心的事件的报告。

在现有的许多微型计算机应用中,都要进行文件处理,但是处理操作很少以上述形态出现。若所有的记录同时都存在RAM中(如§5.3和§5.4中的排序与检索问题那样),则构成一个文件的意义不大。"真正的"文件处理所涉及的是存在辅助存贮器(见§1.3.1)中的文件,一般是磁带(或盒式磁带)上的顺

序文件及磁盘上的随机存取文件。这些设备与有关的程序设计问题超出了本书的范围。

§5.7 特殊的外围设备

§2.7 扼要地讨论了微处理机系统与控制台设备之间的各种接口。许多外围设备可由非常简单的接口进行管理。例如，从逻辑上来说，由计算机程序控制的通断开关 S 只需如图 5.4 所示进行控制。

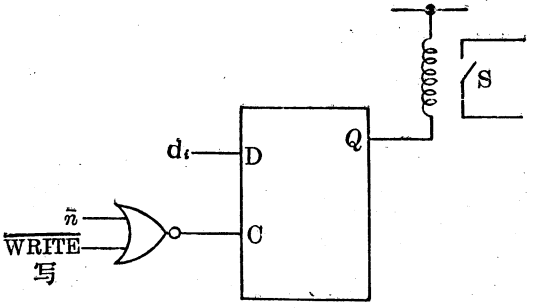


图 5.4

图中 Q=1 时开关 S 断开，Q=0 时闭合。C 端接收到一个脉冲时，Q 的值就是 D 端的值。若地址总线译码形成地址 "n" 时微处理机产生了一个"写"脉冲，则 C 端就接收到一个脉冲。此时，Q 将取得数据总线上 di 位的值。因此，下述指令将使开关 S 闭合：

A:=00　　(或 di=0 的其它任何值)

M(n):=A　　(若适用的话也可用 G(n):=A)

而下述指令将使开关 S 断开：

A:=FF　　(或 di=1 的其它任何值)

M(*n*(:=A　　(若适用的话也可用 G(*n*):=A)

接至Q端上的也可以是其它一些设备，例如由Q上的脉冲驱动的逻辑电路。下述指令串可在Q端产生一个脉冲：

```
A:=00
M(n):=A
A:=FF
M(n):=A      }(或上面提及的其它指令)
A:=00
M(n):=A
```

脉冲的宽度是第4与第6条指令中使触发器触发的两个相应的时刻之间的时间。这段时间与最后两条指令的执行时间之和一样。

第三章末尾的指令表列出了以时钟周期为单位的指令执行时间。在上述例子中，若采用接近于最高频率的时钟，则脉冲宽度将为5～7 μs。若这样的脉冲宽度可以接受，则不会产生什么问题。若要求脉冲宽度较窄，则可能需在Q上接一个具有适当的定时元件的单稳电路。若要求脉冲宽度较宽，当然也可使用单稳电路。但还可在最后两条输出指令之间插入延时指令，用程序产生要求的时间。一条"空指令"需时1～2 μs；其它指令需时更长。延时指令当然必须仔细地进行选择。借助于简单的计数循环，可获得高达1或1.5 ms的延时：

```
      A:=某个值
01:   A:-1
      JNZ (L01)
```

采用双字长寄存器可使延时范围增大到 $\frac{1}{4}$ 或 $\frac{1}{3}$ 秒。在任何情况下，定时精度当然都取决于实际的时钟频率。

注意：包含有时间要求很严格的段落的程序在新的系统上运行之前必须进行调整。虽然 8080 与 8085 的指令系统大体相同，但是在许多情况下两者的定时是不同的。在 Z80 与 8080 指令系统的相同部分中，定时也不同。

实际上，图 5.4 所示的接口不会由另外的分立逻辑元件构成，而是现成的输出转接口的一部分。例如，英特尔 8212 有八套这样的电路，最多可控制八个上述类型的输出。也可使用 §1.4.1 提及的并行接口转接器，可控制 16 或 24 条输出线。在所有这些情况下，程序设计方面需考虑的问题都是一样的，控制脉冲宽度也可采用同样的技术。

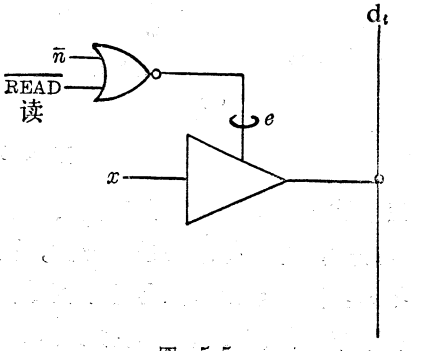


图 5.5

在只有一位的输入设备中，最简单的接口也许是图 5.5 所示的接口。一条“A:=M(n)”之类的指令就可启用三态缓冲器。这样，输入位 X 的值将被读入寄存器 A 的第 i 位。同样可用 8212 与并行接口提供这种硬件功能。

遗憾的是，并非所有的输入都能这样简单地进行处理。若系统本身对 X 端的信号的有效性有所限制（例如在一段时间内信号处于未定状态），则需要使用更复杂的接口。这个问题超出了本书的范围。

实践证明，上述输入-输出技术是适用于 PROM 编程器的（§8.1 和 §8.2.5）。图 5.6 是该电路接至 6820（6800 系统）以及接至 8255（8080 系统）上的接法（见 §1.4.1）。八条“数据”线

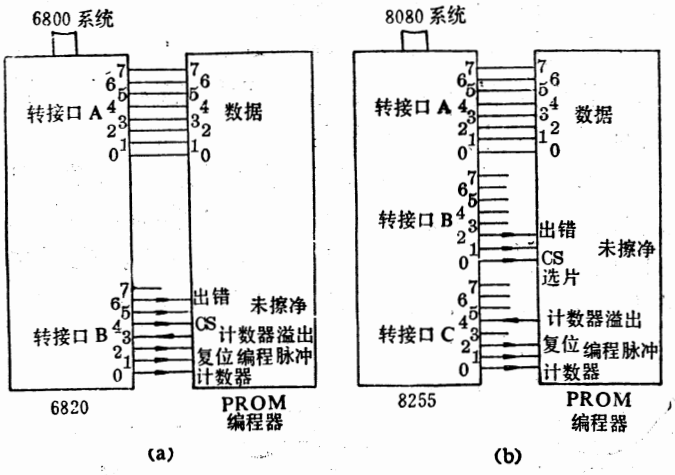


图 5.6

以上述工作方式在不同时间内或者作为输入，或者作为输出。对“程序”脉冲应仔细地进行测量，而“计数”脉冲宽度则几乎可以任取。其余输出都只是电平信号。可以在任何时刻相当简单地读取唯一的控制输入(计数器溢出)。

第六章　用高级语言的程序设计

§6.1　用机器码进行程序设计的基本工具

§4.1介绍了基本的监督程序的特性。借助于这样一个监督程序，就有可能对第四、五章的例子和练习进行处理。也可以继续用监督程序编写与测试实用的程序，但是大多数程序员会感到做这项工作既枯燥乏味又十分单调，而且速度太慢。为克服这些缺点，并尽可能使程序设计成为一项不太繁杂的工作，就需要一些有力而可靠的软件"工具"，能将烦琐的程序设计工作交付给机器去完成。

到目前为止，我们都是用"机器码"进行程序设计的，即解决每个问题用的程序都是用微处理机指令系统内的指令编写的。虽然在书写时能用相当易读的记号表示这些指令，但是将每条指令打入系统时必须将其表示成一对十六进制数字。如果说这种程序设计工作并不完全是用机器码进行的话，那唯一的含意就是已经使用了子程序。通过建立一组有用的子程序，就有可能在程序的任何地方用一条指令来引出一个子程序，而子程序可包含非常复杂的操作。

能直接改善程序员的工作状况的办法有以下三种：

(i) 将监督程序加以扩充，为程序的测试、修改与运行提供更多更好的功能；

(ii) 将子程序集编入PROM，并按研制"软件库"的要求进行安排。

(iii) 提供以某种书写形式将指令直接打入计算机，并由计

算机将其转换成机器码的手段。

具有某种功能的监督程序(见§6.1.1)是第八章给出的软件包的一部分。该监督程序及其子程序一起共占用了864(十进制)个字节(0000～035F);§8.2给出了有关的技术说明。第四、五章编写、介绍或提及的各个子程序以及正确使用这些子程序所需的信息列于§6.2中。有关的技术说明可参阅§8.2.2、§8.2.4、§8.5和§8.6,这些子程序带注释的文本列于§8.8。哪些子程序准备永久性地存入PROM,哪些准备存在盒式磁带、纸带之类的辅助存贮媒介上或留在纸上显然必须由每个用户(或每组用户)作出选择。

为实现第(iii)条功能,则需要一个程序,能读入以程序员所用的书写格式写成的任何指令,并产生相应的机器码字节。该程序还必须满足后面讨论的各项要求。在任何处理机上采用本书所用的表示法都需要一个约有1.5K字节的"转录程序"。为供实用,转录程序必须能读入一系列指令(也许在每读入一条指令后还立即将其对应的机器码打印出来),并将这一系列机器码字节存贮起来。所以在执行完转录操作后,存贮器中就存有一个随时准备运行的程序。为提高其实用价值,这个转录程序还应能提供其它的功能,幸而增加这些功能所需的存贮空间或工作量很小。

正如举例时那样,编写程序时程序中的"绝对"地址常常是不知道的。此外,在任何情况下,相对转移的范围都很难计算,而且还可能受到修改。所以,汇编程序必须能读取标号,并能处理涉及标号的操作。对这一工作如何进行管理可参阅§8.3。同时,汇编程序还可提供建立常数与定义暂存存贮区的手段。第八章给出的8080汇编程序(其技术说明见§8.3,文本见§8.8)能提供这些功能,约占用21/4K字节的存贮空间,其用户指南

将在§6.1.2中介绍。

与汇编程序的作用相反的程序——反汇编程序也非常有用。这样一个程序约需1K字节(Z80的多一些，8080的少一些)。该程序能取出存在存贮器中的程序(或程序段)，并打印出其"汇编语言"文本。本书中的说明就是用第八章给出的8080反汇编程序(其技术说明见§8.4，文本见§8.8 2800-2B7F)产生的。在实际工作中，使用反汇编程序要比使用汇编程序频繁得多。对程序进行修改或纠正后，可用反汇编程序产生易读而精确的文件。反汇编程序的用户指南见§6.1.3。

这些程序设计手段配上PROM编程器(及其软件，见§6.1.4)就向着满足第一段中所提出的要求的方向大大前进了一步。但是，即使有了这些手段，程序员还得完全在机器码这一级上编写程序，因为在汇编程序(或反汇编程序)中书写的指令与机器指令是一一对应的。§6.3将叙述用高级语言设计程序的问题。

6.1.1 监督程序

当系统接通电源或复位时，或者其它程序调用监督程序时，都会打印或显示"MONITOR"(监督程序)，然后在下一行的左边边沿打印提示符">"。完成下述D、I、M、S、X操作后也会打印这个提示符。该符号表示监督程序准备接受需立即执行的命令。每条监督程序命令只有一个字母(D、G、I、M、R、S、X)。如后文所规定的，在命令符的后面还可能跟有一个或多个地址。地址必须正确地打成四个紧密相连的十六进制数字；字节必须打成两个紧密相连的十六进制数字。在检测到任何键盘误操作后，监督程序将放弃执行其现行的命令，打印一个"？"，然后换行，打印一个提示符。在下述说明中，实际的数值只是举例而已，任何这样的值都可被长度相同的其它任何值所取代。在任

何情况下，逗号(“,”)都可由空格来代替。

D——“显示”命令

打入“D1234, 1269”后，监督程序就会把起始地址1234至结束地址1269(包括这两个地址)内的全部信息以十六进制数字的形式显示(或打印)出来，例如：

```
D 1234, 1269
1234 B9 19 5B 08 00 1F 6B 04 3B CB 24 99
1240 FE 2A 77 08 FB 48 DD 14 ED 09 FE 85 FF 4E E5 02
1250 FF A3 FB 42 E9 6C 7D 20 FF 92 FB CF F7 1A FF 06
1260 FB 04 37 03 4E 03 05 02 FB 03
>
```

在显示过程的任何时刻，都可通过按下“escape”(换码)键来中止显示过程，例如：

```
D 1234, 1269
1234 B9 19 5B 08 00 1F 6B 04 3B CB 24 99
1240 FE 2A 77 08 FB 48 DD 14 ED 09 FE 85 FF 4E E5 02
1250 FF A3 FB 42 E9 6C 7D 20
>
```

G——“执行”命令

打入“G1234”后监督程序的响应是换行，然后转移至1234，此时的标志与寄存器值是最近一次进入监督程序时所保护的值，但它可能受到“X”命令的修改(见“X”命令和“R”命令)。

I——“输入”命令

本命令用于将一组连续的字节值装入RAM中从起始地址开始的连续单元内。打入“I1234”引起的响应是换行，然后机器将读入任何数量的成对十六进制数字，并将其值存入连续的存贮单元中去。打印时，可随意地用空格、逗号及回车等字符改变

字节在打字纸上或屏幕上的布局，但是一个字节中的两个数字不可分开。这些分隔符不会被存入存贮器。按下“escape”键就可结束该操作。所有完整的成对数字将被存贮起来。若输入的一对数字中的第二个数字不是十六进制数字，则会引起出错响应。指定存入该奇数数字的单元不受影响。所有完整的字节仍将存贮起来。

M——“复制”命令

打入“M1234, 1269, 1475”会使1234～1269单元(包括这两个单元)内的信息被复制到1475～14AA单元(包括这两个单元)中(这些单元当然必须在RAM中)。若采用其它一些地址值，则目的字块可能与源字块部分重迭，但在完成该操作时，目的字块总能得到源字块的拷贝(见§5.2)。若第二个地址低于第一个地址，则出错。

注：“M”并不表示“传送”。虽然源字块的一部分可能被重写，但是除了这种情况以外源字块都保持不变。

R——“继续执行程序”命令

打入“R”后产生的响应是换行，并在最后一次调用监督程序的程序“断点”处继续执行下去(见下文有关“断点”的叙述)。说得更确切一点，若程序用“JSR1”(十六进制CF)调用了监督程序，且调用监督程序期间只执行了不影响PC内容的D、I、M、S或X命令(见下文“X”命令)，则使用“R”命令后就会转移至存有“CF”值的那个单元之后的第一个单元去，但是保存的标志与寄存器值可能已受到X命令的修改。

S——“代换”命令

这条命令用于检查存贮器中个别单元或一小组连续单元的内容，若有必要的话可对其内容进行修改。顺序打入“S”、一个地址及一个空格或逗号，机器的响应是显示存在该地址单元内

的值，后面跟一个连字符(减号“-”)。这时就可打入一个新值，或保持现有的数值不变(不作键盘操作)。若打入空格或逗号，机器将显示存在下一个地址单元中的数值，后面跟一个连字符。同样，这个值可以保持或用新值取代。若打入“回车”，则结束“代换”操作。

X——“检查寄存器”命令

打入“X”命令，机器将打印标志值与寄存器的内容，其打印顺序如下：

```
>X
NZHPK  A  BC   DE   HL   SP   PC
01010  27 0D3E F3B2 00C3 17F2 0021
```

若欲改变任何标志值，可打入一组(五位)新的数值，这样滑架就会移过两个空格，以对准“A”。若不改变标志值，则按一次空格键，滑架也会移动到对准“A”的位置。

这时可打入“A”的新值；再打入一个空格符就使滑架对准“BC”。其余的寄存器都作为双字长寄存器进行处理。在对应的位置上打入一个四位数的数值就可改变其中任何一个寄存器的内容。在任何时刻只要打入“回车”就会结束整个操作。

注：“PC”值是随后的“R”命令(如果有的话)要用的地址；若用X操作改变PC的内容后，则接着的R命令，与用“G”命令转入同一个新地址(PC值)的作用完全一样。

虽然任何时刻都能用RESET(复位)信号使控制返回至监督程序，但是只有当程序误入岐途需要恢复环境时才使用复位信号。程序员将会发现，动态地用“JSS1”(十六进制CF)结束程序更实用些。这样就能从适当的入口进入监督程序，从而保存所有的标志与寄存器值；此外，这样做对在程序的适当间隔处安置断点也很有用。每个断点处可安排一条“JSS1”(CF)指令。执

行该程序时，在每个断点处都会调用监督程序，此时可利用监督程序的功能研究所发生的情况并作出必要的修改。然后可用“R”命令继续执行用户程序。用“S”命令将每个“CF”改成“00”(空操作)就可方便地去掉断点。

6.1.2 汇编程序

为了正确地运行汇编程序，需使用监督程序及其子程序。如第八章所述，汇编程序能产生长达 1646 (十六进制)个字节的目的程序。

指令必须以第三章所列的 8080 指令的形式提供给汇编程序。在指令表中，单字节值都用“12”来代表。可用任何表示成两个十六进制数字的其它数值取代“12”这个值。该值将转换成所产生的机器码指令的第二个字节(J 字节)。

在指令表中双字长数值都用“1234”来代表，该值将成为所产生的机器码指令的第二和第三个字节(J 和 K 字节)。例如，“J1234”将汇编成“C3 34 12”。任何这样的值都可由形式如下的任意数值所取代(在这些数据形式中，“h”表示十六进制数字，每次出现“h”时都可选择任意的十六进制数值)：

(i) hhhh　四个十六进制数字

(ii) (L hh)　标号形式(注出两个十六进制数字)。汇编程序将这两个数字翻译成标号的地址“hh”(见下文)。

(iii) (Xh)　这个数据将被翻译成 Xh 字组的起始地址。

(iv) (Xh±hh)　比 Xh 字组的起始地址高(+)或低(−)“hh”个字节的地址。注意，这里 hh 是一个无符号整数，所以 ±hh 的范围就是 ±255(十进制)。

(v) (I±hh)　翻译成比(静态顺序中)下一条指令的地

址高或低“hh”个字节的地址。“hh”也是无符号整数。

任何指令前面都可加一个“hh:”形式的标号。在第八章程序包中，只能采用64（十进制）个标号（“00:”～“3F:”）。特别重要的是，子程序的第一条指令之前应该有标号。所以，若子程序以“07:A:=B”开始，则可用“JS(L07)”来调用这个子程序。

在源程序的开头可以对“X字组”进行说明，每个字组不超过255（十进制）个字节，字组数不超过16个。例如，一个程序的开头可以是：

X0:20;　X3:10;　XE:80;　01:A:=00;

在对应的目的程序中，X0字组的起始地址将为1000（第一个RAM地址）；X3字组的起始地址将为1020；XE字组的起始地址将为1030；标号“01:”的地址将是10B0。说明的形式是“Xh:hh”。在程序的第一条指令后不能再出现X说明语句。X字组有下列两个用途：(i)在RAM中保留暂存空间；(ii)将目标程序的第一条指令安排在规定的地址。

紧接着最后一条X说明（如果有的话），或紧接着任何无条件转移（非子程序转移）或无条件返回指令之后，可插入一个常数值字组。有一条写作“VALUES”的汇编程序命令可完成这一工作，该命令一般应加上标号，因此可有：

02:VALUES;
“ASCII-STRING”

汇编程序将引号内字符的ASCII值插入从标号地址开始的连续单元内。又如：

03:VALUES;
#01　23　45　67#

将插入由各对十六进制数字表示的数值。其中的数字必须是偶

数个；各对数字之间的空格、换行、回车字符将被滤除；一个字节的两个数字不得分开。

汇编程序由监督程序命令“G0C00”输入至机器内。然后可打入X说明(如果有的话)以及指令和常数字组。每项内容必须用“;”或“回车”作为结束。机器将打印(显示)一个分号以及相应的目的码(最多三个字节)，然后返回下一行的开头准备打印下一项内容。若提供的指令涉及尚未出现的标号，则机器就以“FD”(向前)作为第三个字节打印出来。否则如上所述，机器将打印第二、三个字节的真正数值。若命令打入得不正确，可按下“擦除”(删除)键消除之。机器将打印“？”，但不会在目标程序上增添新的字节。然后可重新打入所需命令。然而，若已输入的误打命令是以“;”或“回车”结束的，则机器将产生三个全零(00 00 00)的目的码字节，并仍打印出“？”。

程序的结尾(即其文本的结尾)必须用命令“FINISH”(结束)来表示。机器辨认出该命令时就将“FD”涉及的地址填入这些单元，并返回至监督程序。

此时，RAM就存有下述信息，这些信息可借助于监督程序的命令显示出来：

1180以上　　目的程序的指令与常数

1010～1011　紧接着目标程序最后一个字节的存贮单元地址

1020～103D　字组X1～XF的起始地址。X0从1000号单元开始。MM(1020+2h)中存有X(h−1)的起始地址。

103E～103F　要执行的程序的第一条指令的地址。

1040～10BF　标号的地址。运行时标号hh将存在MM(1040+2hh)规定的单元内。

在执行之前，目标程序的指令与常数必须复制到由103E规

定的单元中去。只须使用监督程序命令“M1180, x, y”就可完成这一工作，其中 x=MM1010−1, y=MM103E(注：若无 X 字组，则 y=1000)。

§4.7.6 有一个简单程序的汇编和执行情况的记录。

6.1.3 反汇编程序

反汇编程序由监督程序命令“G2800”调用。其输出形式如§8.8及若干实例所示。调用反汇编程序处理的有关内容的起始与结束地址分别由 HL 与 DE 给出，通常由监督程序的 X 命令进行预置。明确地说，若调用反汇编程序时 HL=x, DE=y, 则打印输出将从 x 单元开始(假定 $x<y$)，打印到 y 之前的最后一个 I 字节所形成的指令为止。反汇编程序最后以“JSS1”(CF)指令调用监督程序作为结束。

反汇编程序不会区分指令与其它形式的信息。例如，若用反汇编程序对数值字组进行处理，则结果是毫无意义的。

6.1.4 PROM 的编程与读出

只有采用如§5.7所示的特殊的 PROM 编程器接口时，才能使用入口为 0360 与 03C0 的程序。这种编程器是为 2708/8708 型 1K 字节的 PROM 设计的。若将一块空白的 PROM 插入编程器，并用“G0360”命令调用编程程序，则 1000～13FF 号 RAM 单元中的信息将被写入 PROM。该编程程序开始时会检查 PROM 是否擦干净了；最后还能把新编程的 PROM 逐个字节地读出来，将其内容与信息源的内容进行比较。若发现有不相符之处，则点亮报警灯。用“G03C0”调用 PROM 读入程序，将使插在 PROM 编程器上的 PROM 中的信息被复制到 1000～13FF 号 RAM 单元中去(若 PROM 未插，则 RAM 中将填入全 1 字节)。

这些程序与监督程序的功能相结合，能非常方便地对

PROM 进行编程与编辑。本书中的软件都是用这些手段研制的。

当然，若使用其它类型的编程器或其它型号的 PROM，则程序也必须作相应的改变。

§6.2 子 程 序

在使用某一子程序之前，必须知道以下情形：该子程序能完成什么功能[其结果如何，使用哪些寄存器和再调用哪些子程序(如果有的话)]；进入该子程序时需要那些信息(入口状态)；转出该子程序时遗留下何种状态(出口状态)；若调用该子程序所处理的数据无效，则结果如何(出错处理)；其它一些特点(注解)以及该子程序的入口地址。

下面分四节介绍与第八章的子程序有关的情形：输入子程序(§6.2.1)、输出子程序(§6.2.2)、定点算术运算子程序(§6.2.3)和浮点算术运算子程序(§6.2.4)。除此之外，第八章另外还有几个子程序，但是这几个子程序是为特殊应用编写的，其它地方不太可能使用。

6.2.1 输入子程序

A:=inchar(输入字符)

入口：　0038，由 JSS7(十六进制 FF)调用。

结果：　从键盘读入 ASCII 字符送入寄存器 A 与 C。

出口状态：　ASCII 值(无奇偶位，见§4.2)存在寄存器 A、C 中。

需使用：　寄存器 A、C。

In-out(输入-输出)

入口：　0020，由 JSS4(十六进制 E7)调用。

结果：　从键盘读入 ASCII 字符，并将其回送(见§6.2.2)给打印机(或显示器)。

出口状态：A、B、C中都存有(最后)输出字符的ASCII值。

需使用：寄存器A、B、C；入口为0010、0038的子程序。

In string(ESC)(输入字符串)

入口：03EC。

入口状态：把字符串的第一个字符所指定的地址存在HL中。

结果：读入与打印ASCII字符串，并将其写入从给定地址起始的存贮单元中。按下结束符号"escape"(1B)键将结束这个操作，其值也被存入存贮器。

出口状态：HL的内容是存有"escape"字符的单元地址。

需使用：寄存器A、B、C、H、L；入口为0018和0038的子程序。

A:=in byte(输入字节)

入口：0220

结果：读入一对十六进制数字，将其组成一个字节存在A中。

出口状态：结果在A中；第一个数字的十六进制值在D中。

需使用：寄存器A、B、C、D；具有下列入口的各子程序：0010、0020、0028、0038、01E6。

出错处理：若有一个数字不是0～9或A～F中的字符，则转至监督程序。

其它入口：若第一个数字的ASCII值已在A中，则可从0221进入该子程序。若第一个数字的十六进制值(用二进制表示)已在A中，则可从0225

进入该子程序。

HL:=in address(输入地址)

入口：　01F7

结果：　读入四个十六进制数字，将其组成一个双字长整数存入 HL。

出口状态：　结果存在 HL 中，第二个字节在 A 中。

需使用：　寄存器 A、B、C、D、H、L；具有下列入口的各子程序：0010、0020、0028、0038、0220、01E6。

A:=in decimal integer(输入十进制整数)

入口：　0E74

结果：　读入带符号十进制整数 n($-128 \leqslant n \leqslant +127$)，符号有无皆可，数字最多为三个，其二进制值将存在 A 中。

出口状态：　n 的值在 A 中；结束字符(ASCII 值)在 C 中。

出错处理：　若整数超出范围或读入不正确的字符，则转至监督程序。

需使用：　寄存器 A、B、C、D、E；入口为 0020、0028、01E6 的各子程序。

HL:=in decimal integer(输入十进制整数)

入口：　0EBA

结果：　读入带符号十进制整数 n($-32768 \leqslant n \leqslant +32767$)，符号有无皆可，数字最多为五个，并将其转换成二进制，存在 HL 中。

出口状态：　n 的值在 HL 中；结束字符(ASCII 值)在 C 中。

出错处理：　若整数超出范围或读入不正确的字符，则转至监督程序。

需使用：　全部寄存器；入口为 0020、0028、01E6 各子程序。

DEHL:=in f.p.number(输入浮点数)

入口：　2745

结果：　从键盘读入十进制浮点数，将其转换成二进制浮点数(§4.7.1)，存在 DEHL 中。

出口状态：　浮点数在 DEHL 中；结束字符丢失。

需使用：　全部寄存器；具有下列入口的各子程序 0020、0E74、25A3、264B、26EC、265B。

注解：(i)输入形式：尾数是一串数字，十进制小数点与符号有无皆可；指数是"E"与一个带符号单字长整数(符号有无皆可)，最后以非数字结尾。

(ii) 输入值 x 的范围：$2.9388\text{E}-38 \leqslant |x| \leqslant 3.4028\text{E}+38$。尾数不超过七位数字(对输入数值有效性的检查是彻底的)，若发现错误则转至监督程序。

(iii) 常数：浮点常数的存贮单元如下：10 在 27DF 中，100 在 27E3 中，10^4 在 27E7 中，10^8 在 27EB 中，10^{16} 在 27EF 中，10^{32} 在 27F3 中。

6.2.2 输出子程序

Out char(C) (输出字符)

入口：　0018，由 JSS3(DF)调用。

入口状态：　C 中存有字符的 ASCII 值(见§4.2)。

结果：　打印(或显示)相应的字符。

出口状态：　A、C 中都是给定的 ASCII 值。

需使用：　寄存器 A、C。

Echo (回送)

入口：　0010，由 JSS2(D7)调用。

入口状态：　C 中存有字符的 ASCII 值。

结果：　若 ASCII 值为 1B 则打印“$”(换码)；若为 0D(回车)则先打印回车，再打换行；否则打印与给定值对应的字符。

出口状态：　A、B、C 中都存有(最后打印的字符的 ASCII 值。

需使用：　寄存器 A、B、C；入口为 0018 的子程序。

CRLF (换行)

入口：　0200

结果：　输出“回车”，接着一个“换行”。

出口状态：　A=B=C=0A

需使用：　寄存器 A、B、C；入口分别为 0010、0018 的子程序。

Out string(B) (输出字符串)

入口：　005F

入口状态：　一串 ASCII 值存在连续的存贮单元内；字符个数存在 B 中；第一个字符的地址在 HL 中。

结果：　打印相应的一串字符。

出口状态：　B=00；HL 的增量等于打印的字符数；最后打印的字符的 ASCII 值存在 A、C 中。

需使用：　寄存器 A、B、C、H、L；入口为 0018 的子程序。

Out string(ESC) (输出字符串)

入口：　03F6

入口状态：　一串 ASCII 值存在连续的存贮单元内；最后一个必须是“1B”。第一个字符的地址在 HL 中。

结果：　打印相应的字符串，直至“换码”(1B)为止，但“换码”不打印。

需使用：　寄存器 A、B、C、H、L；入口为 0018 的子程序。

Out byte(A)（输出字节）

入口：　0204

入口状态：　要打印的字节在 A 中。

结果：　将 A 中的值打成两个十六进制数字。

需使用：　寄存器 A、B、C；入口分别为 0018 和 0030 的子程序。

Out decimal integer(A)（输出十进制整数）

入口：　0E68

入口状态：　A 中存有一个带符号二进制整数。

结果：　打印该整数的值(符号以及最多三位十进制数字)，在宽度为六个字符的字段内靠右排齐。

需使用：　全部寄存器；入口为 0018 和 005F 的子程序。

Out decimal fraction (HL)（输出十进制小数）

入口：　07E6

入口状态：　由无符号整数 $x\times 2^{16}$ 表示的小数 $x(0\leqslant x<1)$ 存在 HL 中。

结果：　打印十进制小数点与 x 的前五位十进制小数位的值。

出口状态：　B=00

需使用：　全部寄存器；入口为 0018、0231、0C1A、0CF1 的各子程序。

其它入口：　若要求打印的位数 $n(1\leqslant n\leqslant 16)$ 已在 B 中，则可从 07E8 进入该子程序。

Out signed fraction (HL, 15b.p.)(输出带符号小数，15 位二进制小数)

入口：　由带符号整数 $x\times2^{15}$ 表示的 x 值 ($|x|<1$) 存在 HL 中。

结果：　x 值的打印顺序是：符号、"0"、四个十进制数字。

需使用：　全部寄存器；入口为 0018、07EB 和 0F93 的各程序。

注解：　从 24E8 进入该子程序将使结果前面增加一个空格。

F·p·print(DEHL)（打印浮点数）

入口：　2B80

入口状态：　x 的浮点值(见 § 4.7.1)在 DEHL 中。

结果：　打印 x 的十进制浮点值。

需使用：　全部寄存器；具有下列入口的各子程序：0020、0E74、25A3、264B、26EC、265B。

注解：(i)输出形式：尾数 m(符号及五个十进制小数位)的值规格化成 $m=0$ 或 $1\leqslant|m|\leqslant10$ (因此 1.00000 与 10.00000 都可能出现)；阶码是"E"后跟一个整数，若为正值则不打印符号。

(ii) 二～十进制转换过程会使打印出来的尾数值产生的最大误差为 $\pm4\times10^{-5}$。

6.2.3 定点算术运算子程序

ASCII-Hex (ASCII-十六进制转换)

入口：　0028，由 JSS5(EF)调用。

入口状态：　ASCII 值(见 § 4.2)在 A 中；要求数值为 30～39 (表示 0～9) 或 41～46 (表示 A～F)。(见

下面的“Check hex”子程序)

结果: (若给定值符合要求)相应的十六进制(用二进制表示)值存在A中(范围是00～0F)。

需使用: 寄存器A。

Hex-ASCII (十六进制-ASCII转换)

入口: 0030, 由JSS6(F7)调用。

入口状态: 00～0F范围内的值在A内(见下面的“Check hex”子程序)

结果: 与十六进制数字对应的ASCII值存在A中。

需使用: 寄存器A。

Check hex (十六进制检验)

入口: 01E6

入口状态: 需检查的值在A中。

结果: 若A中的数值在00～0F范围内,则无操作。

出错处理: 否则转至监督程序。不改变任何寄存器的内容。

HL-DE (比较)

入口: (i)0231, (ii)0C1A

入口状态: 无符号双字长整数在HL、DE中。

结果: 根据HL-DE的值来设置N、Z标志。

出口状态: HL、DE内容不变。

需使用: (i)寄存器A、B、C; (ii)寄存器A。

注解: 见§4.5.2。

HL:-DE (双字长减法)

入口: 0CF1

入口状态: 双字长整数x、y分别在HL、DE中。

结果: 双字长值$(x-y)$在HL中。

出口状态：　结果在 HL 中，y 在 DE 中。

需使用：　寄存器 A、HL、(DE)。

AHL:−CDE（三字长减法）

入口：　25D8

入口状态：　三字长整数 x 在 AHL 中，y 在 CDE 中。

结果：　三字长值 $(x-y)$ 在 AHL 中。

出口状态：　结果在 AHL 中，y 在 CDE 中。

需使用：　寄存器 AHL、(CDE)。

HL:/2（双字长右移）

入口：　24CF

入口状态：　带符号数在 HL 中。

结果：　原数带符号四舍五入右移一位，存在 HL 中。

出口状态：　结果在 HL 中。

需使用：　寄存器 A、HL；入口为 0FF1 的子程序。

AHL:/2（三字长右移，见 § 4.7.3.1）

入口：　2568 与 2577。

AHL:/2^B（三字长右移，见 § 4.7.3.1）

入口：　2583 与 258B

HL:=C×E(unsigned)（无符号乘法）

入口：　0F20

入口状态：　无符号整数 x 在 C 中，y 在 E 中。

结果：　$x\times y$ 的无符号值在 HL 中。

出口状态：　结果在 HL 中，B=D=00，C=x，E=y。

需使用：　寄存器 A、B、(C)、D、(E)、H、L。

HL:=C×E(signed)（带符号乘法）

入口：　0F3A

入口状态：　带符号整数 x 在 C 中，y 在 E 中。

结果：　　带符号值 $x \times y$ 在 HL 中。

出口状态：　结果在 HL 中，B＝D＝00，C＝x，E＝y。

需使用：　　寄存器 A、B、(C)、D、(E)、H、L；入口为 0F18、0F20、0F32 的各子程序。

DE:＝D×E((unsigned)（无符号乘法）

入口：　　0F44

入口状态：　无符号整数 x 在 D 中，y 在 E 中。

结果：　　无符号值 $x \times y$ 在 DE 中。

出口状态：　结果在 DE 中；其它寄存器不变。

需使用：　　寄存器 D、E；入口为 0F20 的子程序。

DEHL:＝BC×DE(unsigned)（无符号双字长乘法）

入口：　　2400

入口状态：　16 位无符号整数 x 在 BC 中，y 在 DE 中。

结果：　　32 位无符号整数 $x \times y$ 在 DEHL 中。

出口状态：　结果在 DEHL 中，x 在 BC 中。

需使用：　　寄存器 A、(BC)、DE、HL。

DEHL:＝BC×DE(signed)（双字长带符号乘法）

入口：　　2420

入口状态：　16 位带符号整数 x 在 BC 中，y 在 DE 中。

结果：　　32 位带符号整数 $x \times y$ 在 DEHL 中。

出口状态：　结果在 DEHL 中。

需使用：　　全部寄存器；入口地址为 2400 的子程序。

HL:＝BC×DE(15b. p.)（小数乘法）

入口：　　2440

入口状态：　x(15b. p.)在 BC 中；y(15b.p.)在 DE 中；$-1 \leqslant x, y < +1$。

结果：　　$x \times y$(15b.p.)在 HL 中($x=y=-1$ 除外)。

出口状态：　结果在 HL 中。

需使用：　全部寄存器；入口为 2400 和 2420 的子程序。

BCDEHL:＝BHL×CDE(signed)　(带符号三字长乘法)

入口：　250A

入口状态：　24 位带符号整数 x 在 BHL 中，y 在 CDE 中。

结果：　48 位带符号整数 $x\times y$ 在 BCDEHL 中。

出口状态：　结果在 BCDEHL 中。

需使用：　全部寄存器；具有下列入口的各子程序 0F3A、2400、2556、2562。

HL:＝(B/D)×2^C(unsigned)

入口：　0F50

入口状态：　无符号整数 y 在 B 中，x 在 D 中，p 在 C 中 ($0\leqslant p\leqslant 14$ 十进制)。

结果：　$(y/x)\times 2^p$ 的无符号值(见下文)在 HL 中。

出口状态：　结果在 HL 中；C＝00。

出错处理：　若结果超出范围，则转至监督程序。

需使用：　寄存器 A、B、C、D、H、L。

注解：p 为特定值的情形：当 $p=0$ 时，结果为双字长整数，在 HL 中；当 $p=8$ 时，结果为单字长整数，在 H 中；当 $p=14$ 时，结果有 14 位二进制小数位，在 HL 中。$(y>63\wedge y>x)\wedge([y/2]\times 2\neq y)$ 时，结果为 $[(y/2)/x]\times 2^{p+1}$，否则结果将正确地被截尾。

HL:＝(B/D)×2^C(signed)

入口：　0F76

入口状态：　带符号整数 y 在 B 中，x 在 D 中，p 在 C 中 ($0\leqslant p\leqslant 14$)。

结果：　$(y/x)\times 2^p$ 的带符号值在 HL 中。

出口状态：　结果在 HL 中；C=00。

出错处理：　若结果超出范围，则转至监督程序。

需使用：　全部寄存器；子程序 0F50。

注解：与 0F50 的注解相同，但是 x、y 只读入 $|x|$、$|y|$。

HL:=(BC/DE)×2^4(unsigned)

入口：　0F9B

入口状态：　无符号双字长整数 y 在 BC 中，x 在 DE 中；无符号整数 p 在 A 中($0 \leqslant p \leqslant 14$)。

结果：　$(y/x)\times 2^p$ 的无符号值存在 HL 中。

出口状态：　结果在 HL 中。

出错处理：　若结果超出范围，则转至监督程序。

需使用：　全部寄存器；入口为 0231 和 0CF1 的子程序。

HL:=(BC/DE)×2^4(signed)

入口：　0FCD

入口状态：　带符号双字长整数 y 在 BC 中，x 在 DE 中；无符号整数 p 在 A 中($0 \leqslant p \leqslant 14$)。

结果：　$(y/x)\times 2^p$ 的带符号值在 HL 中。

出口状态：　结果在 HL 中。

出错处理：　若结果超出范围，则转至监督程序。

需使用：　全部寄存器；入口为 0231、0CF1、0F9B 的各子程序。

HL:=sqrt(BC)×2^7

入口：　0BE0

入口状态：　正整数 x 在 BC 中($0 \leqslant x \leqslant 16383$)。

结果：　正值 $2^7\times x$ 在 HL 中($0 \leqslant 2^7 x \leqslant 16383$)。

出口状态：　结果在 HL 中，x 在 BC 中。

需使用：　寄存器 A、(BC)、DE、HL；入口为 0231、

0CF1、0F9B 的子程序。

注解：见§6.3。

HL:=sqrt(HL)×2^8

入口：　0D6F

入口状态：　正整数 x 在 HL 中($0\leqslant x\leqslant 16383$)。

结果：　$2^8\times y$ 的值在 HL 中，其中 $y=x$ 且 $0\leqslant 2^8\times y\leqslant 32767$。

出口状态：　结果在 HL 中。

出错处理：　若 x 超出范围(变负)，则转至监督程序。

需使用：　寄存器 A、HL；具有下列入口的各子程序：0231、0BE0、0C1A、0CF1、0E9B。

注解：转出该子程序时，H 中存有 y 的整数部分。

Polynomial(15b.p.)(求多项式的值)

入口：　2452

入口状态：　自变量 x(15b.p.)在 BC 中($-1\leqslant x<+1$)，多项式的次数 n(整数)在 A 中；a_n 的单元地址在 HL 中；系数 a_n、…、a_0 存在连续的各对单元中，都有 15 位二进制小数位。

结果：　多项式的值 p(15b.p.)在 HL 中，其中：

$$p=\sum_{i=0}^{n}a_ix^i$$

出口状态：　结果在 HL 中，a_0 在 DE 中，A=00。

需使用：　全部寄存器；入口为 2440 的子程序。

注解：见§6.3。

HL:=sin(HL)15b.p.(正弦)

入口：　2470

入口状态：　自变量 x(15b.p.，弧度)在 HL 中，$|x|<1$。

结果: $\sin(x)$ (15b.p.)在 HL 中。

出口状态: 结果在 HL 中。

需使用: 全部寄存器; 入口为 2440 和 2452 的子程序。

注解: 要求 $|x| \leqslant \pi/4$。在这个范围内的误差小于 ±0.0004 (见§6.3、§8.5)。

HL: = cos(HL) 15b.p. (余弦)

入口: 2490

入口状态: 自变量 x(15b.p., 弧度)在 HL 中, $0 < |x| < 1$。

结果: $\cos(x)$ (15b.p.)在 HL 中。

出口状态: 结果在 HL 中。

需使用: 全部寄存器; 具有下列入口的各子程序、0CF1、0D6F、0FF1、2440、2470。

注解: 要求 $0 < |x| \leqslant \pi/4$。在这个范围内的误差小于 ±0.0004(见§6.3)。

HL: = tan (HL) 15b.p. (正切)

入口: 24D9

入口状态: 自变量 x(15b.p., 弧度)在 HL 中, $0 < |x| < \pi/4$。

结果: $\tan(x)$ (15b.p.)在 HL 中。

出口状态: 结果在 HL 中。

需使用: 全部寄存器; 具有下列入口的各子程序 0CF1、0D6F、0FCD、0FF1、2440、2470、2490。

注解: 误差小于 ±0.0007(见§6.3)。

Degrees to radians (度-弧度转换)

入口: (i) 24C1, (ii) 24C4

入口状态: (i) 度数(整数) d 在 A 中, $|d| \leqslant 57$(十进制); (ii) 度数(8b.p.) d 在 HL 中, $|d| < 57.3$。

结果：　对应的弧度(15b.p.)在 HL 中。

出口状态：　结果在 HL 中。

需使用：　全部寄存器；入口为 2400 和 2420 的子程序

Radians to degrees（弧度-度转换）

入口：　2500

入口状态：　弧度数 r(15b.p.)在 HL 中，$-1 \leqslant r < +1$。

结果：　对应的度数(8b.p.)在 HL 中。

出口状态：　结果在 HL 中；r(15b.p.)在 BC 中。

需使用：　全部寄存器；入口为 2400 和 2420 的子程序。

6.2.4　浮点算术运算子程序

Normalize up（向左规格化）

入口：　(i)25A1，(ii)25A3

入口状态：　(i)x在 AHL 中 $-2^{22} \leqslant x \leqslant 2^{22}$

(ii)p（整数）在 B 中；与(i)相同，x 在 AHL 中。

结果：　y在 AHL 中，q 在 B 中，满足 $y=x=0$ 且 $q=0$，或者 $2^{21} \leqslant y < 2^{22}$ 或 $-2^{22} \leqslant y < -2^{21}$ 且(i) $y=x \times 2^q$ 或(ii)$y \times 2^p = x \times 2^q$(见§4.7.2)。

出口状态：　结果 q 在 B 中，y 在 AHL 中。

需使用：　寄存器 A、B、H、L。

Normalize down（向右规格化）

入口：　25CB

入口状态：　x 在 AHL 中；p 在 B 中。

结果：　若 $-2^{22} \leqslant x < 2^{22}$，则 x、p 在 AHL、B 中不变；否则 AHL 中为 $x/2$（四舍五入），B 中为 $p+1$(见§4.7.2)。

出口状态：　结果在 B、AHL 中。

需使用：　寄存器 A、B、C、H、L；入口为 2577 的子程序。

在§4.7.4 中介绍了以下子程序：

Store（存贮）	$M^4[BC]:=DEHL$	入口 2608
Fetch（取数）	$DEHL:=M^4[BC]$	入口 2619
Multiply（乘法）	$DEHL:\times M^4[BC]$	入口 264B
Negate（改变累加器的符号）		
	DEHL:-	入口 265B
Negate（改变存贮器中数的符号）		
	$M^4[DE]$:-	入口 266A
Reciprocal（倒数）		
	DEHL:/	入口 26D9
Divide（除法）	$DEHL:/M^4[BC]$	入口 26EC
Add（加法）	$DEHL:=M^4[HL]+M^4[DE]$	入口 26F7
Subtract（减法）	$DEHL:=M^4[HL]-M^4[DE]$	入口 2731
Read（读入）	DEHL:＝输入浮点数	入口 2745
Print（打印）	浮点打印(DEHL)	入口 2B80

最后两个子程序的外部(十进制)格式已在§6.2.1 与§6.2.2 进行了介绍。

§6.3　从机器码向高级语言发展

借助于监督程序，可将十六进制(用二进制表示)信息输入至系统、从系统的一个部分复制到另一个部分去、修改其内容并加以显示；以及输入程序、中断或继续执行程序。有了汇编程序，就可将用易读的形式书写的程序直接送入机器；并可用反汇编程序将程序(或部分程序)以同样易读(即易为操作员理解)的形式显示出来。但是这些软件都还不能提高程序员的表达能

力，即仍然受到具体处理机的指令系统的限制。然而，指令系统中包含有关于子程序的指令，正是由于研制了驻留的子程序库，才有可能用比大多数机器指令更高级的语言来表示程序（至少能表示某些程序的一部分）。一个“驻留”的子程序（即存在PROM中固定单元中的子程序）能使指令系统增加一条功能很强的新指令，这种办法颇为有效。例如，借助于PROM中的浮点子程序，就可用8080指令“JS2731”（CD 31 27）表示“将两个给定的浮点数的浮点差送入浮点累加器”操作，这是一个需执行大量简单指令的操作。从§4.7.6温度转换程序的简单例子中已看到，若有正确的子程序可供使用，特别是当这些子程序已被设计成一个完整的子程序集时，程序可简化成一系列子程序的调用操作，也许中间还要加上一些存、取参数的指令。

认为浮点子程序很有用的读者肯定希望将这个子程序集加以扩充，使之包括如所周知的“高级程序设计语言”的“标准函数”（平方根、正弦、反正切、对数等等）。若希望任何新的子程序与现有的子程序集一致，则其规格是很明确的。例如，“余弦子程序”将用$\cos(x)$的值取代浮点累加器中的x值。用符号表示就是：

$$\mathrm{DEHL}:=\cos(\mathrm{DEHL})$$

或者简写成：

$$\mathrm{DEHL}:\cos$$

本书给出的定点子程序包括一个双字长平方根子程序（入口为0BE0）与一个15位二进制小数的多项式计算子程序（入口为2452）。这些子程序所用的算法也可用于浮点子程序。

平方根子程序采用的是牛顿迭代算法。若$f(x)=0$的根y的一个近似值是y_0（f是一个性能良好的函数），则

$$y_1=y_0-\frac{f(y_0)}{f'(y_0)}$$

就是比 y_0 更精确的近似值。

以求平方根的情况为例，若 y_0 是 $\sqrt{x}$ 的一个近似值，则 $y_1=\frac{1}{2}\left(y_0+\frac{x}{y_0}\right)$ 就是一个更好的近似值。表示成子程序的形成就是：

```
      y:=y0
01:   b:=y
      a:=x
      a:/y
      a:+y
      a:/2
      y:=a
      y-b
      JNZ(L01)
      RET
```

其中“JNZ”表示“若两次连续迭代的结果（y 与 b）仍可区分，则转回”（见§4.7.7）。在入口为 0BE0 的定点平方根子程序中，当 y 与 b 相等或仅最低位有差异时，则整个操作结束。最初预测的 y_0 值无关紧要；可采用在结果的上下限之间的任何值。为了在双字长范围内给出尽可能多的有效数字，该子程序计算的是 $2^7\times\sqrt{x}$（即 $\sqrt{x}$ 有 7 位二进制小数位）。“$a:/y$”这一步骤是通过使 A=0E（$p_0=14$，见§4.5.5.1）时调用“HL:=(BC/DE)×2^A”来实现的。结果的绝对误差在 $\pm4\times10^{-3}$ 内。实际上，若 $2^7\sqrt{x}$ 是一个整数，则结果是精确的。否则当 x 较小时，其相对误差在 ±0.16% 内；而当 x 较大时，相对误差在 ±0.003% 内。

将这种算法用于浮点自变量（产生浮点结果）可有几种途

径。最明显的办法也许是程序中每步都调用一个现有的浮点子程序。另一种办法是将自变量看作为包含有一个阶码与一个尾数。若阶码为奇数，则将阶码减1，并将尾数左移一位。然后将阶码除以2并求出尾数的平方根。尾数是一个有22个二进制小数位的三字长数，编写处理这种数的子程序时，在某些方面需要仔细一些。但是，这种子程序一旦编好以后，整个程序就比"全浮点"式程序快得多，也短得多。

多项式计算子程序(入口为2452)所用的方法称为嵌套乘法。三次多项式 $a_3x^3+a_2x^2+a_1x+a_0$ 包括六次乘法与三次加法。但是，若表示成 $[(a_3x+a_2)x+a_1]x+a_0$ 则可减少到三次乘法与三次加法。n 次的多项式

$$p=\sum_{i=0}^{i=n} a_i x^i$$

用嵌套乘法进行计算，只需要 n 次[而不是 $\frac{1}{2}n(n+1)$ 次]乘法：

```
        i:=n                    i:=n
        p:=a_n                  p:=0
01:     i-0             01:     p:+a_i
        RZ                      i-0
        i:-1            或      RZ
        p:×x                    i:-1
        p:+a_i                  p:×x
        J(L01)                  J(L01)
```

入口为2452的子程序使用这两种形式中的第二种。处理浮点多项式时用第一种形式可能更方便些：

```
ST:=HL          a_n 的地址
ST:=BC          x 的地址
```

```
        ST:=AF          次数 n 存入 A
        B:=H
        C:=L
        JS2619          取 a_n
01:     AF:=ST          i
        BC:=ST          x 的地址
        A-00
        JZ(L02)
        ST:=BC
        ST:=AF
        JS264B          p:×x
        BC:=17FC
        JS2608          存贮 p
        AF:=ST
        A:-1            i:-1
        BC:=ST
        HL:=ST
      4×HL:+1           (参阅§5.5)
        ST:=HL
        ST:=BC
        ST:=AF
        DE:=17FC
        JS26F7          p:+a_i
        J(L01)
02:     BC:=ST          (清除堆栈)
        RET
```

注解：M[17FC] 是监督程序保存程序计数器与堆栈指示器

值用的单元，普通的子程序使用该单元是很危险的。只有符合下述条件时，一个合格的驻留子程序才能使用 M⁴17FC：(i)该子程序工作时监督程序不需保存寄存器值；(ii)驻留子程序所用的 RAM(而不是堆栈)应保持在尽可能小的范围内。

入口为 2470 的定点"正弦"子程序使用定点多项式子程序计算截尾的台劳级数(见§8.5)。这种办法在自变量 $|x|\leqslant\pi/4$ 时是一种很好的近似计算法，结果有 15 位二进制小数位。但是，若自变量是任意范围内的浮点变量，则这种办法就不是最好的办法。用简便的契比雪夫级数、连分数或其它办法计算三角函数及其它函数的近似值的办法在许多有关数学计算的书上都有介绍。为微处理机编制子程序集的程序员必须考虑到应用时的特定环境；采用的大型计算机上所用的子程序技术不一定是好办法。

除了需要一组算术运算子程序以及计算三角函数和其它函数用的子程序以外，还需要一组执行其它操作的子程序(也许后者能代替前者)。例如，一个与研制本书的程序时所用的系统相似的系统，就增加了一个 PROM 存贮空间，用来保存一组图形显示器用的子程序。如果可能的话，这些子程序应设计得相互统一。例如，一个子程序的结果可留在约定之处，供另一个子程序作为数据用；而且就数学计算而言，这些子程序在二进制小数位、数的长度等方面是"相容"的。

无论这些实际的驻留子程序集如何，都会出现大量的(一、二百或更多)"JS2731"(CD 31 27)之类的子程序调用指令，这些调用指令在系统中具有专门的意义。虽然"JS2731"的意义与其机器码指令所表达的意义相同，即"转移至入口为 2731 的子程序去"，但是在根据现有的子程序库形成的高级语言中，这种表示法与"D6 12"表示"减法"同样不适合、不明确。所以考虑

为汇编程序引入一种新的书写形式(每个子程序一条)是值得一提的。开始时可采用§6.2表中所用的描述性标题。这些标题肯定能使§4.7.6中的温度转换程序更加易懂易读:

```
        DEHL:=输入浮点数
            BC:=1300
        M4[BC]:=DEHL
        DEHL:=输入浮点数
        BC:=1304
        M4[BC]:=DEHL
        DEHL:=输入浮点数
        BC:=1308
        M4[BC]:=DEHL
01:     换行
        DEHL:=输入浮点数
        BC:=130C
        M4[BC]:=DEHL
        HL:=130C
        DE:=1300
        DEHL:=M4[HL]-M4[DE]
        BC:=1304
        DEHL:×M4[BC]
        BC:=1308
        DEHL:/M4[BC]
        打印浮点数(DEHL)
        J(L01)
```

对该程序作一些整理,例如缩短浮点读入部分的文本、不用小写字母(实际计算机的实用程序中不会出现小写字母)等,这个程

序就可用一个不比原先的汇编程序大很多的汇编程序进行读入。但是，每条机器指令仍然要占用一行；而且由于需要读取参数，所以包含相当数量的重复操作。例如，前三行（除了地址不同以外）在程序中重复了三次以上。每一组（三行）表示式在高级语言里表示一个操作——读入一个值并将其存入规定的单元内。这个操作可以用本书所用的表示法来表示，例如表示成"M4(1300):=READ"。同时，汇编程序也需加以扩充，使其能读入这段程序并产生九个机器码字节"CD 45 27 01 00 13 CD 08 26"。这样，温度转换程序就简化成：

```
        M4(1300):=READ
        M4(1304):=READ
        M4(1308):=READ
01:     NEWLINE (换行)
        M4(130C):=READ
        DEHL:=M4(130C)-M4(1300)
        DEHL:×M4(1304)
        DEHL:/M4(1308)
        PRINT(DEHL) (打印)
        J(L01)
```

就一行可代表几条机器指令而言，这个程序文本是我们遇到的第一个高级语言文本。但是根据其中出现的寄存器名称，仍可识别出该文本是为 8080、还是为 8085，或是为 Z80 编写的。而且，该文本仍然采用绝对地址。下面这段文本是第一个脱离具体机种的程序文本，上一个文本的寄存器名称与地址已由符号名称所取代。其水平相当于五十年代广泛采用的"自动代真"技术，只需作很少的修改就可被自动代真编译程序所接受。

```
        W0:=READ
```

```
    W1:=READ
    W2:=READ
01: NEWLINE (换行)
    W3:=READ
    R:=W3-W0
    R:×W1
    R:/W2
    PRINT(R)
    J(L01)
```

虽然也许能看出这段程序用的是带累加器的计算机，但是整个文本中并没有迹象表明这是哪种计算机用的程序：是 8080 用的或别的微处理机用的程序，还是其它某种计算机用的程序？ 但是，肯定可以用一种扩充的汇编程序将其翻译成 § 4.7.6 所示的机器码程序或其它某种处理机用的类似程序。汇编程序扩充到这一步才能被称为编译程序或翻译程序。

获得脱离具体机种的文本后，还有可能继续加以改善，将常数写成中间操作数，而由编译程序管理存贮器中的内容：

```
01: NEWLINE (换行)
    W3:=READ
    R:=W3-32
    R:×5
    R:/9
    PRINT(R)
    J(L01)
```

由于对算术表达式进行翻译的技术早在 1951 年就已出现了，所以还可用更高级的表示方法表达这段程序：

```
01: NEWLINE (换行)
```

```
F:=READ
C:=(F-32)×5/9
PRINT(C)
J(L01)
```

为使这段程序能成为 BASIC、FORTRAN、ALGOL 或 PL/1 程序，只需在连接部分作些微小的修改。将程序的内部结构表达得明确些就可得：

```
repeat(new line; F:=read;
        C:=(F-32)×5/9; print(C))
```

这个程序除了没有结尾以外完全可以看作为高级语言程序。

至少可从两方面来看待用上述各种文本代替 §4.7.6 中的程序作为微处理机系统的源程序的各个阶段。其一是将每一阶段看作为用较高级的表达式代替前一级(较低级)的一类表达式；其二是将每一阶段看作为能提供比前一阶段更强的表达能力。可能出现的一种情况是每一阶段配置一个新的编译程序，最后一级的编译程序并不比第一级的编译程序大得多或复杂得多。另一种可能的情况是所有级别的表达式都使用同一个编译程序，这可能会使系统的软件比我们所预期的更庞大。

若采用第一种办法，则最后的结果可能是有二个编译程序：一个是低级(机器码级)的编译程序，处理由具体控制机器操作的指令组成的程序；另一个是高级的编译程序，处理用方便的高级语言表达的程序。换言之，最后就象在大型机上那样只有汇编语言与一种高级语言，两者的表达能力是相差很多的。

若编译程序的容量与速度仍能控制在适当范围内的话，则第二种办法为微处理机用户提供了一种新的途径，是非常值得一试的。这是一种供"多级语言"使用的编译程序，既能接受低级表达式，又能接受高级表达式；在这种语言中有可能用同样的

功能通过程序控制输出至外设去的标准宽度脉冲或对从仪表来的读数进行计算。

§6.4 将高级语言翻译成机器码

无论用哪个级别的语言表达程序(或程序的一部分)，其运行时都是执行一系列机器指令。

§5.4举了一个用高级语言文本产生机器码程序的处理过程的例子。ALGOL60型希氏排序程序被翻译成8080子程序的形式。在该例中，没有严格地运用预定的变换规则来介绍翻译过程，而是以若干非正式而直观的步骤进行介绍的。在各个阶段，都必须决定所涉及的各种量的类型与范围，从而也决定了产生的子程序的适用范围。无论用"人工"方法，(如§5.4的办法)进行翻译，还是由机器自动进行翻译(如下文所述)，都必须作出这些决定。该例是将 n 个元素 $a_1, \cdots, a_n$ 按数值大小的顺序进行排列，每个元素都是实数。n 的值没有限制，但应该是整数，从而可以对其进行计数(见§4.7.7)。若允许的 n 值范围不同，则产生的子程序也有所不同。对于实数的排序通常用浮点算术操作来实现，但是也许需要的是对更简单的类型的数值进行排序的子程序，这在高级语言中是看不出什么区别的。因此，排序子程序的第一个文本编写得只能处理无符号单字长整数(即具有同样数量的二进制小数位的定点数)，并提出了编制处理其它类型数值的排序子程序的可能性。最后应说明，下标在ALGOL语言中只表示这些元素是编号的；该子程序的每种文本都要求将这些元素存在间距相同的存贮单元内。

假设已有一个能直接读入ALGOL文本并产生相应的机器码子程序的编译程序或翻译程序。该编译程序可能产生许多不同的程序文本(§5.4介绍的只是其中的一种)，我们希望产生哪

种文本呢？特别是编译程序如何决定 n 的上下限、元素值的数值类型以及如何进行存贮呢？若除了 ALGOL 文本外没有其它任何信息，则据推测对计数器应采用双字长整数算术运算，而元素应采用浮点数并假定这些元素存在连续的单元内。这样，就可对 RAM 中的任意一组浮点数进行排序，这就是可预期的结论。若 ALGOL 文本规定的是整数数组 a，就应产生处理双字长整数元素的文本。因为 ALGOL 60 中只有实数与整数两种数，因此无法产生超出这一范围的文本。其它某些高级语言(ALGOL68、PASCAL、PL/1)允许采用其它类型的数值，就这一点来说可能比 ALGOL 60、BASIC 与 FORTRAN 更适于作为设计微处理机用的高级语言的基础。但是，若准备以这几种语言之一为基础设计微处理机高级语言的话，就会发现对这种语言的用途作出必要的限制所需的工作量与扩充其它语言所需的工作量差不多，ALGOL 68 与 PL/1 则尤其是这样。供其中几种语言的不同文本用的几个编译程序已编写了出来，最常用的语言是 BASIC。在必须进行数值计算的场合，能规定执行整数与浮点算术运算的编译程序实际上是非常有用的。有几种编译程序以及较浅的高级语言编译程序只能进行整数的算术运算，因此毫无实用价值。其中有些受其性质及所编译的语言的性质所限，无法接受 PROM 编程器所需的那类程序 (§ 8.8 0360～03FD)的表示式，这类程序需要仔细的定时并需详细规定对非标准外围设备的操作。其它的编译程序可以接受这类程序，但是先必须解决大量的"概念混乱"问题。编制这类程序的方法必须改成能适应所采用的语言的其它方法，产生勉强而难懂的操作表达式。而这些操作用机器码来表达是非常清楚的。

若一个程序员对其最喜欢用的高级语言有深入的了解，但是在他喜欢使用的微处理机上却没有这种语言的编译程序，则

他将发现，用§5.7介绍的方法把任何短的程序或处理过程，翻译并改编成机器码程序并不太困难。如果这个工作很麻烦，那末总有一定的回旋余地将个别的表达式翻译成目标程序(如何处理§5.7中希氏排序算法的嵌套循环结构?)。

在第7章中将讨论一种既可采用高级表达式又可采用低级表达式的语言的设计问题。

第七章　软件的组织、语言与结构

§7.1　计算机软件的发展

在1977年国际信息处理联合会会议上，大多数著名的权威都认为微处理机使程序设计技术倒退了25年。遗憾的是，这一论调并不完全对。

1952年，“软件”这个词尚未出现。新的计算机必须从零开始进行程序设计，无法求助于其它计算机及其软件，也不受它们的干扰。但是二十五年后，新问世的微处理机却诞生在一个计算机大家族中，其中许多计算机很庞大，但也有许多计算机受到了四分之一世纪任其发展的软件所拖累而踌躇不前。若新机器是一个微处理机系列的最新产品，则一出现就具有该系列近二、三年积累起来的软件。将所有这些软件（当然包括第八章的软件）与多年来许多人积累起来的软件进行比较也许是不公道的。但是实际上要吃透那怕一台大型机上现有的软件，并将其削减到合适的大小都是一件很艰巨的任务。如果对任何一个机器实现这一点，就能以此作为将来的微处理机软件的模型。然而，这个工作还没有人去做，所以我们应当注意，不要模仿因袭下来的错误作法，也不要仿效现有机器的浪费状况。1978年曾有过这样一则广告：介绍一个包含用FORTRAN语言编写的模拟程序的系统，而该系统需要一个很大的FORTRAN编译程序、一个24K字的暂存存贮器、一个磁盘机以及一个人机对话式终端。所有这一切都用作为研制微处理机用的初级程序的手段。如果这台大计算机不仅是为此目的而专用的，则可能还有一些社会

价值；但是，若用这样一大堆软件和硬件来实现原来用规模适度的微处理机系统就能很有效地完成的工作，则是很荒唐的。

六十年代是以大型、集中式计算机为主的时代，当时软件的增长非常迅速。这种计算机是个威严的庞然大物，俨然是位于宇宙的中心，由它决定处理什么信息，而且只有在其附近的外部设备才能与其进行联络。在这一阶段的初期，FORTRAN 编译程序约有 20000 条指令，而良好的 ALGOL 编译程序还少得多。一、二年后，据说出现了其编译程序长度超过 25 万条指令的语言。而且，计算机并不只满足于编译程序，此外还有汇编程序、编辑程序、装入程序、模拟程序、仿真程序、操作系统……，包含数十万甚至上百万条指令。这些软件并不便宜，某些段落的价值要用"人-世纪"来计算（罗马教皇礼拜堂的天花板也只值 4 人-年）。这些软件并非总是设计或编制得很好的。一个明显的例子是，某软件包连续地月月宣布其中某些部分作废，每次作废都表示修正了上千个错误。当然没有必要对追究这种情况的人作出解释。由于计算机要发展，所以软件就会积累起来，不仅规模会增大，而且功能与影响也会增强。

正是由于半导体工艺的发展，才打破了计算机对信息处理工作的"垄断"。小型计算机不仅满足了互不相关的分散的计算工作的需要，而且在信息处理方面比中央计算机更能满足要求，因为在中央计算机上有大量的用户要争夺机器的使用权。在"实时处理"、"过程控制"、"物理系统"（physical system）等场合都有这种情况，即计算机必须与其它部件进行通讯。

微处理机（一种超大规模集成电路）及其有关的存贮器和接口器件非常灵活、适用、轻便，而且功能也很强。这些器件的价格非常便宜，运行成本极低，开销甚微。它开辟了一个崭新的应用领域，在这种情况下，可就地独立地完成大量的计算工作，而

且绝对可靠。以前使用中央计算机的地方，现在可以使用数百、甚至数千台本地计算机，这些本地计算机的机型也许只有几种。显然，在这样的情况下所研制的软件将不能再竭力效仿二十五年来所积累的软件了。微处理机的软件必须力求经济，并避免无计划的增长。

练习

假设你有一台 X 型微型计算机系统，并获得了一些在与之相当的 Y 型微型计算机上运行的程序(例如总共有 x 个字节)。若希望使用这些程序，并已知你自己的时间的价值，则 x 值达到多大时就应该购买一台 Y 型机器？ x 值处于哪个范围时才值得使用在 Z 型小型或微型计算机上运行的 Y—X 交叉编译程序-模拟程序？若你购买了一台 Y 型机，则使用 Y 型机时 X型机将做什么？是否有关系？另一方面，若只限于使用 X 型标准化的系统，则又将如何？

微处理机的发展有可能产生许多新型的 CPU，以及与之对应的许多指令系统。虽然许多现有的和将来的处理机无疑将会消失，但是予先赋予任何一个处理机或指令系统将来会需要的性能也是很不合算的。这种见解尚未出现。

作为一个程序员，有可能遇到各种机器、各种机器码以及高级语言。虽然日常工作只限于使用一、二种处理机，但是，能阅读并理解为若干不同系统所编写的程序这一优越性也是显而易见的。程序设计的发展历史中最奇怪的现象之一是“语言惰性”。这种惰性仍然表现在两个主要的方面。其一，相当数量的程序员在学习程序设计时只使用一种语言，他们似乎会变得接受不了任何其它的语言或程序设计风格；其二，某些早期语言的非实质性的、表面的形式竟被保留了下来，虽经一番据理力争的过程，但仍然把它们作为这些语言后继版本的特点。为避免这一

现象对将来产生的阻碍作用，应该鼓励那些易于接受新概念的人运用第二种语言，并希望新的一代懂多种语言的程序员跳出老框框，更好地推动程序语言的合理发展。

§7.2 软件的规划与设计中的一些考虑

任何一种基本软件（不属于"较高级"的软件）的规格都只能满足少数潜在的用户的需要。而且也不存在某些典型的用户或典型系统。

第八章的软件及本章所考虑的要点都可看作是特殊的（甚至人为的）环境的产物，这种环境就是州立大学数理工学院的"计算机科学"系，而州立大学无疑与其它部分一样都感到经费奇缺。

在这个系中攻读"计算机科学与数学"的学生在三年内要学习某些数学、学习在计算发展史与当前实践中具有深远意义的一些知识、获得进行程序设计与基本的计算机设计的能力，并真正理解已供其使用的一些工具。而攻读数学、物理学、医学及其它学科的学生所接受的课程要短得多，主要是使他们熟悉一些普通的设计思想，并获得一些程序设计的经验。第三种学生是参加夜校和假期短期学习的成年学生，他们的目的是使自己能跟得上新的发展。参加有关微处理机及其应用课程的人员（工业上的设计工程师、武装部队成员、商业公司的董事、医学考古学家、教师、打字员、店主等）不仅影响了成人教育课程的内容，而且影响了大学课程的内容。他们的关心甚至常常比前两类学生和他们的雇员的关心更直接，也更有用。

微处理机对所有这些课程可能会产生很深远的影响。以往大学里要么干脆不讲计算机设计，要么泛泛谈一些计算机中极简单的逻辑电路设计，从不详细讨论程序设计实习用的具体计

算机的设计。由于过去一般使用的是中央计算机，这种计算机终日忙个不停，而且距离太远，所以无法用于说明计算机设计问题。然而现在有了微处理机及其有关部件，事情就好办了。这些部件的特点是：

(i) 从概念上说是基本逻辑设计教学中广泛采用的TTL和CMOS系列集成电路的发展，实际上也与TTL和CMOS系列集成电路兼容；

(ii) 学生可以了解和掌握这些部件的内部结构，并能评价其优劣，用这些部件可以组装成一台完整的计算机，借以说明过去在课堂上学不到的那些计算机设计原理和计算机总体结构。

(iii) 可构成一个有用的计算机，能够毫无困难地实现以前只能在中央计算机上进行的程序设计实习。

总之，微处理机能提供大学里计算机科学系教学所需的全部东西，只缺少有关大型系统的某些特点的一些内容(如果这些特点尚未过时或暂时不会过时，则可作为专题加以讨论)。尽管多个微型计算机系统的投资比普通花在中央计算机设施上的投资要少得多，但却能提供各式各样的系统，使学生广泛了解各种处理机的差异，在数量上更能让每个学生有充裕的时间进行程序设计实习，以及从事计算机设计实验。当然也会有足够的微型计算机供不需使用大型机及其软件的研究人员和毕业生们使用。

在其它部门(例如前面提到的成人学习班)，虽然许多方面的要求有所不同，但是有理由认为他们也需要数量较多的微型计算机。

凡是配备有多台计算机的部门，多半是把其中一些计算机用于研制程序，而其它的计算机则可能主要用于"生产性"工作，即用于运行成熟的程序。这些功能可以互换与不能互换的范围

是与软件设计有关的一个重要问题。

生产性计算机的设计方案是由该机的若干应用中的目的程序提出的要求共同决定的。能否(或是否应该)用同一台(或同类的)机器研制程序？还是必须用专用的"开发系统"(甚至也许是以另一种处理机为基础构成的开发系统)完成研制程序这一特别任务呢？由于研制工作的最后阶段必须在目的机器(生产性计算机)上测试目的程序的运行情况，所以目的系统显然必须成为开发系统的一部分。因此，上述问题就变成了究竟需要为研制工作的前一阶段增加多少装备。

就这件事发表意见的人们当中已出现了一种倾向，即夸大程序的编写与调试工作的困难，主张使用很大、很复杂并且很昂贵的系统，而且怂恿人们要求使用很小、很便宜的目的系统和目的程序。实际上，最好的开发系统基本上应该是目的机器再增添两个部分：(i)存有任何必要的开发软件(汇编程序、反汇编程序、高级解释程序或编译程序)的 PROM；(ii)存贮被研制程序的辅助存贮器(也许是§6.1.4介绍的 PROM 编程器或盒式磁带机)。实施一个系统时，必须充分强调其经济效果，犹如敲核桃不必使用大铁锤那样。在小钢琴上不会演奏的孩子在教堂的大琴上也不可能演奏得更好。开发系统比上述扩充的目的系统多不了什么东西；而开发软件应与其功能相适应，尽可能地简单。这里还应指出的是，虽然半导体存贮器越来越便宜，但是仍然比纸张要贵数百倍。因而没有必要把程序都存入半导体存贮器。

§7.3 指令表示法与语言

第二、三章说明了四种微处理机的指令系统及其指令的表示法。其中 8080 与 8085 两种机型的指令系统大体相同。Z80 除了包括 8080 的指令系统外，还有其它一些指令，诸如明显

地受到6800指令系统启发而形成的许多指令。而6800与8080的指令系统差异很大。

第三章与§2.10(末尾)列出了一些表格，这说明有可能用一致的、系统而易读的表示法来表示几乎所有指令系统的基本功用。例外的二、三种情况是那些结果依赖于外部信号("复位"或中断请求)的指令。在某些情况(特别值得指出的是Z80的"字组处理"与"单步"指令)下，这种表示法需要使用相当长的表达式。为了简明扼要，第三章的表格中使用了一小组字母来表示这些指令，在实用程序与设计好的软件中也采用这种表示法。

这种表示法的用途已在若干方面反映了出来，其中有一些列于§2.2中。此外，第四、五、六章介绍的这种表示法本身，就说明它是可以简单地，而且是首尾一贯地发展而成的一种表示法，它能表示高级程序步骤以及那些尚未落实到具体机型去的程序步骤。这样就可以同时为各种处理机研制功能相当的子程序。任何一种处理机用的子程序或程序都可用同一种方式来表达。

对于这种表示法的用途，以及将其扩充到其它处理机与其它软件(除低级汇编程序以外)去的可能性提出了以下一些问题：

问题1 其它微处理机的指令系统能否用这一种符号系统表达？

答：有些机器肯定适用(不敢说全部)。原来的四种处理机最初采用该系统时都需对该系统特别进行扩充，该系统用于新的处理机时可能还需进一步加以扩充。然而，重要的是这种扩充不应影响这种表示系统的核心部分，而且应保持系统尽可能简单，并且在概念上尽可能首尾一致。

问题2 对于具体的某种处理机而言，这种表示法能在何

种软件中起作用？

答：在与8080的汇编程序和反汇编程序类似的软件中肯定可以起作用。其它处理机用的类似程序有的已编好，有的正在编。§6.3提出了一种编制高级编译程序的可能性，下面将讨论一种具有高级表达式的"语言"设想。

问题3 能否实现指令系统之间的翻译程序？

答：也许不能。处理条件标志的细节问题使这一工作变得极端困难。在任何情况下，用编写该程序时准备使用的机种运行该程序，比通过翻译在其它某种机器上运行或进行模拟要容易得多，也便宜得多。

问题4 能否实现供几种机器的组合指令系统用的软件？

答：不能。一个程序只在一种机器上有意义。若多处理机系统中包含不同型号的处理机，则多处理机程序就是一组单处理机程序的集合。但是，某些高级表达式可以脱离具体的体型，所以能翻译成若干种机器码。

设计常规计算机用的高级程序设计语言的工作，常常被认为就是为一台抽象的计算机规定高级指令系统，而这台抽象计算机是由实际的计算机模拟而成的。虽然这种办法(更不必说设计与机器无关的高级算法语言的理论方法了)对阐明某些概念是有好处的，但是由于微处理机用的语言基本上必须保持能表达微处理机的每条指令的操作，所以这种办法用在微处理机上并不完全满意。

在§8.8的文本中的许多地方都对这种困难作了说明。03C0—03EB的PROM读入程序是一个最说明问题的实例(注：03D1—03EB的子程序还供入口为0360的PROM编程程序使用)。这是一个相当简单的实现微处理机与特殊外围设备之间的通信用的程序。该程序采用的是供接至并行接口转接器上

的其它设备所用的那种技术(见5.7)。这个程序是一种为模/数转换器、图形显示器、行式打印机、磁盘存贮器等设备研制的典型的基础软件。这类程序无法用BASIC或FORTRAN语言来表达,即使能用其它某种高级语言来编写,表达得恐怕也有些勉强(见§6.4)。讨论这个例子可有助于确定用那一级(高级或初级)语言来表达这类程序最为适宜。首先应该考虑程序的规格。这种外围设备及其有关的引出端如图7.1所示。该设备可容纳1024个字节的信息,并认为这些字节是连续编号的。无论何时只要在"计数"端接收到一个脉冲,该设备就在"数据输出"端依次送出下一个字节。若在"复位"端接收到一个脉冲,则下一个字节就是编号最低的字节。在传送完1024个字节后,该设备会产生"计数器溢出"信号。该设备接至8255型器件上,如图5.8(*b*)所示,所有的信号都与8255兼容。PROM读入程序的作用是将该设备的信息读出来,写入RAM的1000~13FF(十六进制)各单元中去。"复位"与"计数"脉冲的宽度是无关紧要的。如果这样安排的话,程序的功能就与机型有很大关系。但是,如果未提及接口的性质,那末程序设计应进行到何种程度呢?回答是不会太深。考虑到整个程序的功能,可以用§6.3(温度转换程序)的方法提升原低级程序的级别。然后将该程序(及其子程序)与PROM编程程序编在一起,使占用的存贮空间尽可能小。这样做的一个不良后果是,用任何简明的符号表示法表达该子程序的操作都相当困难。例如,入口为03E3的子程序使HL加1,向8255的C转接口的0号

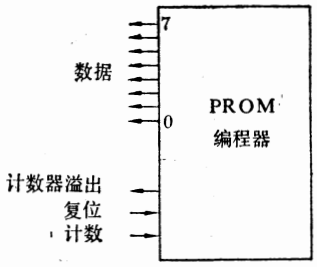


图 7.1

线输出一个正脉冲，并根据 C 转接口的 4 号线上的信号状态设置 Z 标志。这些操作几乎不可能用一条程序设计语言的语句来表达。若程序必须尽可能短，则§8.8 中的表达式可能就是最好的了。但是，若其它标准(例如结构清楚优雅)比紧凑更重要，则较有可能获得相当高级的表达式。若不使用子程序，则原来的程序可写成：

```
        A:=98
        GF7:=A      } 预置接口
        A:=04
        GF6:=A
        A:-A        } 向 PROM 编程器输出"复位脉冲"
        GF6:=A
        HL:=1000    —
01:     A:=GF4      } 从 PROM 读入下一个字节并将其存贮起
        M:=A        } 来
        HL:+1       —
        A:=01
        GF6:=A
        A:-A        } 向 PROM 编程器输出"计数"脉冲
        GF6:=A
        A:=GF6      } 读取并测试"计数器溢出"信号
        A:&10       }
        JZ(L01)     —
        JSS1        —
```

括起来的每组机器指令对应于一个完整而独立的接口操作。若软件的功能中没有提及 8255 接口的性质，则操作的注释只能达到上述程度。可以象§6.3 中那样为每组操作设计一种记号，

从而产生该程序的一种较高级的表达式。用这种表示法表示的程序显然是专供 8255 用的；若使用 6820 或 Z80PIO 的话，机器码程序就将不同，并需使用其它的专用记号。若将设备号为 F4～F7 的设备的 A 转接口写作“GF4.A”，则程序的一种可能形式就是：

```
        GF4.CONTROL:=98
        GF4.C.2:=PULSE
        HL:=1000
01:     M:=GF4.A
        HL:+1
        GF4.C.0:=PULSE
        Z:=#GF4.C.4
        JZ(L01)
        MONITOR
```

借助于标识符（符号名称，见§ 6.3）可将这段程序变成：

```
        PP.CONTROL:=98
        PP.RESET:=PULSE
        HL:=1000
01:     M:=PP.DATA
        HL:+1
        PP.COUNT:=PULSE
        Z:=#PP.OVERFLOW
        JZ(L01)
        MONITOR
```

若欲使程序的这个文本能为编译程序所接受，则编译程序中显然必须包含下述信息：PROM 编程器的“复位”端接至 I/O 设备号为 F4～F7 的 8255 的 C 转接口的第 2 位上，以及诸如此

类的一些信息。(在理论上可以这样安排,并通知机器操作员使其能正确地进行连接,但具体实现是很困难的。)若系统是固定的,则这种信息将隐含在编译程序中;增添任何新的外围设备都需对编译程序进行扩充,去掉任何外围设备也意味着应该缩短编译程序。使用能说明外围设备的编译程序,在某些情况下也可能是合理的,这种程序能处理一个程序中包含的有关决定目的系统中使用何种外围设备并安排在何处之类的信息。

可以象§6.3那样去掉标号,将程序的结构表示成:

```
PP.CONTROL:=98;
PP.RESET:=PULSE;
HL:=1000;
repeat(M[HL:+1]:=PP.DATA;
  PP.COUNT:=PULSE)
  until PP.OVERFLOW;
MONITOR
```

除了可以不用 HL 作为寻址寄存器以外,这个文本也许就是能获得的最高级的文本了。该程序文本是 8080 指令与高级程序结构的混合程序。该程序仍然是与具体机型有关的(特别是其第一行);除了第一行外,这个程序很容易理解。只有详细了解 8255 的性能才能理解"98"的含意。这就使人注意到另一个困难,这个困难一般与接口所用的元件有关。虽然 8255 是个相当简单的器件,但几乎可把该器件本身作为一个微处理机来看待;控制值"98"与所有其它的控制值不同,其意义无法用一、二个 ASCII 字符来表达。其它的并行接口转接器也有独特的问题,更不必说串行器件、专用接口以及新出现的各种元件了。目前尚不了解这些元件的全貌,所以无法确定能否用一种通用的方法来描述其操作的高级表达式。这些元件品种繁多,其设计技

巧也各不相同。要不了多久，每种新元件都可能象一种新的微处理机那样要求增添许多新型的表达式。

不与其他人交往的程序员当然可以完全自由地接受或拒绝向他提出的各种表示方法，并想出一些只有他自己能理解的缩写记号，构成其专用的符号系统，用于编写他自己的程序。但是，这种自由度是非常有限的。其原因是：可供他使用的软件很有限，所有的程序都必须按这种特殊规定编写或重新编写；而且他还有必要与完全不懂这种符号使用法的其他人交往。换言之，每个程序员迟早都将使用为用户所能理解的语言或由机器上已配备的程序设计语言来编写程序，甚至要同时用到上述两种语言。

没有任何软件的机器，其本身也有一种程序设计语言，这就是由指令系统组成的机器码语言。由于程序员可阅读指令的说明，并观察机器执行这些指令的情况，所以这种语言可以被程序员所理解。但是，由于这种语言采用二进制（或八进制、十六进制）表示法，也就无法向读者提示这部分程序的意义，所以用这种语言写的程序很难读。因此，通常在书写时用能提示指令意义的形式来表示指令。把用这种书写形式表示的指令转换成二进制码所用的程序一般还具有其它一些功能（例如§6.1.2介绍的功能），称为汇编程序。汇编程序能处理的语言称为汇编语言。由于用汇编语言编写的程序的每一个语句都只与一条机器指令对应，所以汇编语言是一种低级语言。

近年来已研制了许多高级语言。其特点通常是：一个高级语句（或程序步骤）会产生许多机器码指令操作。一种良好的、设计完善的高级语言对程序员有相当大的吸引力。最明显的是使用高级语言编写的程序非常接近于原始问题的描述语言，常常借用问题中变量的名称作为程序中的标识符。这种语言能接

受某种按一般规则构成的表达式(公式)。高级语言对子程序以及重复或迭代结构(见§7.4)的处理能力很强,因此原来的计算表示式可原封不动地用在程序中作为表达式。用良好的高级语言编写的程序即使写几遍也比用低级语言编程序快,而且错误少得多。高级程序甚至已达到与具体机型无关的程度,所以有可能用于已配备有这种语言的任何机器。使用高级语言所花的代价是要有这种语言的编译程序以及便于用这种语言进行程序研制的其它软件。

供某些微处理机使用的某些高级语言的编译程序可以购得。购买其中最好的编译程序是值得的。但是,有些编译程序所要求的存贮容量似乎太大了一点;也还有一些编译程序如上述所指出的那样,是华而不实的。

肯定有许多用户一时会认为某种编译程序已能满足其所有的程序设计方面的要求。同样,也有一些人认为满足不了他们的要求。这些用户可能已经发现了低级汇编语言的限制。他们的要求实际上是希望能用不同级别的语言来表示程序。有些程序很自然地可以用高级语言来编写;还有些程序由于直接与接口有关,可能需要使用低级表达式;第三类程序可能要求有些段落用高级语言编写,有些段落用低级语言编写,段落之间要用多种级别的语言编写,因此希望有一种"多级"程序设计语言,能接受任何程序。这些程序可以用机器码来表达,但是程序中的各个不同部分可以用最适宜的语言来写。低级语言可用来编写所有的程序,但非常麻烦;高级语言能提供方便并提高程序员的效率,但并不是任何高级语言都可表达系统要求执行的每件工作,并且很少能以最合适的方式来表达。

如果要满足上述要求,问题就成了逐步地对低级语言加以扩充(如§6.1.2举例说明的那样),使其能提高到§6.3和§6.4

及本章所达到的程度。与此同时还必须为不同发展阶段的语言配上编译程序或解释程序，使各级语言可供实用。并提供辅助软件，为程序员编写与修改程序的任务提供方便。

在较早期的计算机上这个问题不象现在这样突出。这也许是由于早期倾向于把为大型机服务的程序员分成“系统程序员”和“应用程序员”。系统程序员集中考虑使计算机尽可能通用的问题；应用程序员主要是用机器及其各种设备解决一系列具体的问题。这种区分的结果似乎扩大了低级语言程序设计与高级语言程序设计之间的距离。

拟定整个多级语言的范式和功能决不是轻而易举的事。某些阶段做起来困难不大，例如为调用驻留子程序操作提供记号（例如“HL:=C×E”）。但是有些阶段（例如引入各种类型的符号名称和常数）就需要非常仔细。若有可能出现 PROM 读入程序的最后一个文本那样的程序，则也可能会碰到问题。

下一节将扼要地介绍一个可行的研制计划的几个阶段，以及能使程序表达得较优雅，而所需的结构方面的特点。

§7.4 程序设计语言的结构与风格

微处理机用的多级语言范式必须符合某些标准。首先，机器码中的每一条指令必须能表示成多级语言中的一个语句。第二，在最高一级上的表达能力必须与目前能在微处理机上实现的高级语言的表达能力相仿。第三，必须可以实现从低级到高级的不同级别的语言。能在某一级别上运行的程序也必须能在任何较高的级别上运行。在任何级别上，与一条机器指令对应的语句产生的目的码必须仅仅表示那条指令（实现较高级的语句时规定不会这样严格）。第四，这个范式不应使任何级别的编译程序所占用的存贮容量达到不合理的程度。

这个语言的基础显然必须包括在表示机器指令的记号表(如第三章所列出的指令表)中。此外,在最低级别上也必须能在指令前加标号,并能用对标号的访问代替对地址的访问。

§6.3提出的第一个扩充步骤是采用能表达某种意义的子程序调用操作。这就要求对准备编制的子程序库进行仔细的考虑;每个子程序库对应着特定的一组操作(例如浮点算术运算),这一组子程序应作为一个整体来看待,并考虑到相互间的兼容性。每个子程序调用操作的书写形式在风格上应该与其它子程序调用操作的书写形式以及指令的书写形式相适应;而且如果可能的话,其格式应该与现有的书写形式一致。注意到这些问题有助于简化这一阶段的工作以及减小编译程序长度。

§6.3提出的第二步是允许每个表达式代表几条机器指令(子程序调用和参数处理指令),这一步不会产生新问题;但是相关性、风格与结构仍然很重要。这些"宏指令"无疑会蕴含地使用寄存器。程序员必须了解这些寄存器在这些"宏指令"中的使用情况。犹如在一个普通的子程序调用过程中,要知道哪些寄存器会受影响那样。

无论如何也不能说这两个扩充步骤是将要作出的其它扩充步骤所必须的。有些读者可能觉得他们需要这些步骤,有些则认为不需要。在任何情况下,此时的语言作为程序设计语言,从概念上来说都是非常简单的。其基本的语句形式也许只有四、五种。其中之一是"a op b"(a"操作"b),其中第三个成分(或者第二和第三个成分)可以不出现;另一种形式是"J mode cond addr"(J模式、条件、地址);还有一种形式是供"其余"操作用的,即"标识符"形式。语句的组成成分包括作为标识符用的寄存器名称、标志、中断屏蔽字、存贮器及一组输入-输出转接口,以及作为数值用的二进制的一位、两个十六进制数字、四个十六进制

数字和访问标号的各种量。其语法应该能简单地模仿 ALGOL 60 报告或其派生语种之一进行定义。其语意必须比第三章中一节的内容稍多一些，增加的部分是子程序的部分。就我们现在的目的而言，只要能决定哪些指令要与该语言中的某条语句对应就够了；没有必要定义机器码的语意。§ 6.3 所提出的其它扩充步骤引入了范围有限的标识符和非十六进制形式的常数值，这些步骤必须在决定了某些结构问题之后再考虑。

其中主要的问题与这种语言的使用方法有关。本书中假设程序（或程序中的模块）是脱离计算机而编写的，然后在计算机上作为一个整体进行测试。有些程序员喜欢在控制台设备的键盘上以交互式工作方式边编写程序边进行测试，某些类型的程序采用这种办法较为有利。APL 与 BASIC 之类的语言似乎较适合于这类用途，而 ALGOL 60 及其派生语种的模块结构则不适于这类工作方式。其它的问题都一样，选择交互式程序设计还是分离式程序设计方式取决于程序员的爱好，但事实说明后一种方式所产生的程序在风格与结构方面来得好些。除非准备采用交互式程序设计，否则进一步研制微处理机语言最好是以 ALGOL 60 的模块结构为基础。

程序员可说明（规定）一个标识符在模块中代替给定的某种类型的变量，特别方便的是按具体的微处理机中寄存器或标志的名称对标识符进行命名，或者以与这些寄存器或标志的用途相适配的形式对标识符进行命名。此外，可以说明一个模块中（或模块中的某种数值）的数字是十进制或者十六进制等。整个机构应该与具体机型无关，但是程序中任何影响寄存器或标志名称的说明将使该程序只能用于特定的一组机器（甚至只能用于一种机器）。这样地进行扩充以后，就为接受用户的变量名称以及使用方便的（十进制）表示法所表示的数值创造了条件。但

是，为不致于给程序员造成不必要的麻烦，要求在定义扩充部分的语意时非常仔细。如果如上所述提供了说明，那末还需对合并表达式或公式采取一些步骤。其中最重要的问题是保证单义性，不应模棱两可（即“二义性”）。必须竭力避免通常的“自上而下”语言定义中所出现的那种松散结构。

无论是否运用 ALGOL 60 的模块概念，在程序设计中定义“构件”时采用这种概念肯定能使结构清晰。一条语句既可表示一条机器指令，也可表示一条上述的那种“宏”指令，或者象 ALGOL 60 中那样表示一个适当的模块。因此，定义条件（或选择）语句，甚至定义多种选择（“case”）语句就成了很简单的事。引入计数和迭代结构也不困难。后面两种结构就是“for”和“repeat”（“while”或“until”）语句，本书中许多地方（例如§ 2.10）已非正式地引入了这些结构。

与模块概念有关的最后一个扩充部分也许是将子程序变换成带参数的“过程”。这个过程可以进行得很深；如果不先考虑上述任何建议对编译程序以及目的程序造成的后果，就将其规定在语言的范式中，那是很笨拙的。

有充分的理由认为在各种扩充步骤之中最迫切需要的是增添构件。当然，没有构件也可编写结构良好的程序①。但是，除非采用规则的结构，而不滥用无条件转移与条件转移，否则程序容易变得曲折隐晦、无法阅读并充满错误。但是，由于每种处理机的硬件都有自己的特色（特权编址就是一例），所以无法完全排除转移指令。虽然在为结构性较强的问题编制程序时应该尽量避免使用“go to”（“转向”）语句，但有时仍然必须使用 go to 语句。也有排序子程序之类相当混乱的情况（见图 5.5）。排序子

①：注 O. J. Dahl, E. W. Dijkstra 和 A. R. Hoare: “Structured Programming”。

程序的基本过程有三个“嵌套”循环，即一个套着一个的三重循环。最后产生的子程序的形式是三个联锁的循环，这种形式是无法单独用常规的构件来表示的。这个子程序的结构是否良好？或由于过分地要求节省存贮单元而不用这种结构是否值得？与这种算法已有的各种结构相比，这种算法的最佳结构是哪种结构？是否通得过正确性检验？你的回答是否取决于你是不是仍然喜欢使用程序设计计算机？

有一点是很清楚的，设计微处理机用的语言所涉及的问题远未完全解决。虽然有许多用户一时将满足于已有的“高级”编译程序，而不研究该程序是如何工作的；但是有些人发现，寻找比目前更适宜的手段表达微处理机程序时将遇到困难。

虽然近25年来的经验并不容人们乐观，但是设想一个读者正在继续为在2003年出版“微微处理机程序设计与软件研制”一书而努力，并能写道：“虽然新的微微处理机(picoprocessor)刚一问世就使程序设计落后了25年，但是这25年来取得的进步使我们有可能在几个月内就为微微处理机配上最好的软件”并不完全是幻想。

第八章　小型8080系统用的第一个软件

§8.1 历　　史

本章的软件(即程序与子程序的集合)是随同相应的系统一道生成的。除了8080系统以外没有使用其它的计算机。该软件是从1976年12月30日开始编写的；1977年3月完成了前4K程序(监督程序、汇编程序和基本算术运算程序)的编写工作，6月完成了其余2K程序(反汇编程序和浮点算术运算程序)的编写工作。此后，监督程序与反汇编程序均有一处需要改正。这两个错误都非常重要，所以后面作了简单的说明。花在这个软件上的全部工作量不超过三个人/月。主要的工作期是圣诞节与复活节假期，两个假期之间隔着一个春季学期，被学生与同事们分散了许多注意力。作这样一个说明是为了给那些对编写程序有畏难情绪的人鼓气，也希望读者对说明这个软件时将指出的一些较粗糙之处给于谅解。

最初编写该软件包时，使用的是未加修改的英特尔公司SDK(系统设计套件)单板机，以及配有容量为1K字节的ROM的MCS-80监督程序、3K字节的空白PROM(三块2708)与1K字节的RAM(八块2111)。并使用一台KSR-33型电传打字机，接至SDK中的8251接口上(参阅§1.4.1与§2.8)。一个月后由K. R. Brooks① 设计的PROM编程器投入使用，通过

① K. R. Brooks: "Microprocessor-controlled PROM programming" New Electronics, 2, 23(1976)。

SDK 中的 8255 与系统相接。二月份在 SDK 板上的绕接区(用户区)里增加了 1K 字节的 RAM 组件,并用存有下面将介绍的监督程序以及 PROM 编程程序的 PROM 取代了原来的监督程序ROM。三月份安装了一小块扩充板,使系统具有 6K的 PROM(六块 2708)。

该软件研制的进程如下:

(i) 编入 PROM 的第一个程序就是 PROM 编程程序(PPP)本身。借助于这个程序以及英特尔公司的监督程序逐段地对存在 RAM 中的汇编程序主体进行了测试,并将其写入 PROM。汇编程序的最后一部分被写入存有"PROM 编程程序"的 PROM 的多余空间里。在这一阶段,RAM 容量只有 1K (1000～13FF) 须用作监督程序的暂存存贮区。因而在主汇编程序的每块 PROM 中最后 30 个字节都存有零碎的子程序,这些内容是后来补进去的。

(ii) 在编写新的监督程序时,汇编程序被搁置起来。这个监督程序的功能比英特尔公司 ROM 监督程序稍强一些,并节省约 15% 的存贮空间。节省下来的存贮空间(0360～03FF)足以容纳一个 PROM 编程程序、一个 PROM 读出程序以及字符串输入与输出用的子程序。由于只能在没有监督程序可用的情况下对新的监督程序进行测试与修改,所以这项工作是这个软件包研制过程中最困难的环节。

(iii) 在新的监督程序与 PROM 编程程序投入使用以后,再借助于汇编程序,研制了后来被填入四块 PROM 内的各种子程序。直至此时才增添了另外的 1K 字节的 RAM,并可使用监督程序与汇编程序 PROM 底部的多余空间。

(iv) 如前所述,增加了一块 PROM 扩充板后才提供了新添的 2K PROM 空间。然后编写了反汇编程序并编入这两块

PROM 之一；最后设计的是浮点程序包，并安排在多余的存贮空间内。

(v) 还有其它一些程序(例如反波兰解释程序以及图形监督程序)也存在 PROM 中。借助于 PROM 读入程序就可将 PROM 编程器当作一种非常快速与方便的输入设备，使这些程序用几秒钟时间就能输入 RAM 中。

这些 PROM 软件已复制了几套正在使用。但是，在最新的 8080 与 8085 系统中用的是 2716 型 2 K PROM，而不是 2708 型 1 K PROM。

下面将逐段介绍这些软件。每个程序或子程序都有一简略的说明。这个说明之后就是由反汇编程序打印输出的整个软件包文本，并对文本作了详细的注释。本章末还列出了 6 K PROM 的无注释的十六进制码文本。

该系统的地址与设备号分配如下：

PROM	0000－0 FFF, 2400－2 BFF
RAM	1000－17 FF
8251	设备号 FA、FB (供电传打字机或目视显示部件用)
8255	设备号 F4、F5、F6、F7 (供 PROM 编程器用)

§8.2 监督程序 PROM (0000～03 FF)

在这块 PROM 占用的存贮区域内有几个地址需特别加以考虑：

(i) “复位”(“RESET”)地址(0000)。当开启系统电源或按下复位键后，程序计数器复位至 0000，因此执行该单元内的指令。所以，把为建立其它程序提供手段的监督程序的起始地址安排在 0000 单元。

(ii) "专用(再启动)子程序"地址 0000、0008、0010、0018、0020、0028、0030、0038。第一条指令位于这些单元之中的子程序,可以用一条单字节的指令来调用(在其它任何单元内的子程序需用三字节指令调用)。所以,最常用的子程序应写入这些单元内。但是,其中有一个地址是 0000("复位"地址),其用途已在上一条说明了。另一个地址 0008 由 JSS1(十六进制 CF)调用,供其它程序调用监督程序本身时使用,所以该地址已被用作监督程序的"正常"入口。剩下的还有其它六个地址,间隔 8 个字节。

8.2.1 监督程序——主体部分

复位信号不仅加到 CPU 上,而且会作用于外围设备接口,特别是加到计算机与控制台设备之间的 8251 上。从复位入口进入监督程序时,前四条指令 (0000～0007) 用于预置 8251 接口。这些指令执行的操作之一是规定要传送的信息(一个字符)的格式;规定的具体内容由制造厂有关 8251 的说明给出。从程序设计的观点来看,应该注意: (i) 复位后必须对 8251 进行预置; (ii) 若在使用接口之后与发出另一个复位信号之前再次执行预置指令,则其作用并不是重新进行预置。由于不能(或不希望)用软件产生复位信号,所以不应该通过软件从 00 重新进入监督程序。供通过软件进入监督程序的入口必须避免出现 8251 预置指令。这就是必须通过调用专用子程序 SS1(JSS1, 十六进制 CF)从 0008 重新进入监督程序的原因。

从 0008 开始,监督程序必须仔细地把执行用户程序出口处的 JSS1 指令时所有寄存器的内容存在"安全区"内。由于专用子程序占用了 0010～0066 这一块存贮区,所以 000D 处有一条无条件转移指令,转至 0067 号单元。0008～000C 与 0067～007D 内的指令将寄存器的信息存入 M17F4～M17FF (系统

RAM 的“高端”)。处理 SP 与 AF 的内容会遇到一些小问题,所以需要相当巧妙地进行处理。PC 值已被子程序调用指令 JSS1 存入堆栈,作为返回地址用。注意,复位和接口预置过程与这段指令是重复的,但是在复位和接口预置过程后执行这段指令没有害处。

007E～0086 内的指令使电传打字机打印(或使目视显示部件显示)信息“MONITOR”。打印这个信息是表示已通过复位或软件调用操作进入了监督程序。

0087～00B9 内的指令构成了监督程序的主循环。这段程序首先将堆栈指示器置成 17F4,所以监督程序使用的堆栈在保存寄存器内容的安全区之上。然后换一行打印(或显示)提示符号“>”,监督程序准备接受需执行的命令(0090)。监督程序的命令可以是 D、G、I、M、R、S、X 七个字母之一。若打入其中一个字母,就执行相应的监督程序分支部分。D、I、M、S、X 各分支以转至 0087 的操作结尾。0087 是主循环的第一条指令。G、R 分支的结束状态是进入其它某个(规定的)程序,在该程序中可在适当的时间再次从 0008 进入监督程序。若按下的是除这七个命令字母之外的其它键,则监督程序将打印一个问号“?”并返回循环的入口 0087 等待下一步的操作。

8.2.2 专用子程序

在讨论监督程序的各个分支之前必须先介绍一下专用子程序,因为监督程序要用到它们。

以 0038 为入口的专用子程序 SS7(“A:=inchar”),该子程序用于从控制台设备读出一个字符。在调用该子程序的过程中,该子程序不断地循环(0038～003B),直至操作员按下键盘上的一个键为止。按键操作后,转接口 FB 的第 1 位给出一个“1”,转接口 FA 将相应的字符送入寄存器 A。任何奇偶位(A_7)

都将被去除，输入字符将存入寄存器 A 与 C 中。

系统的硬件不会将输入字符“返回”或“回送”给电传打字机的打印部件或目视显示部件的显示屏。因此调用 SS7 只读出一个字符，但并不显示。

基本的输出子程序是专用子程序 SS3(“outchar(C)”)，其入口为 0018。由于 0018～0020 之间存贮容量太小，所以该子程序一开始就转至 0045 号单元。该子程序在 0045 进入一个循环，直至转接口 FB 的第 0 位为“1”，即一直循环到打印机或目视显示部件准备好接受下一个字符时为止。一当第 0 位改变成“1”时，字符就被输至转接口 FA。

入口为 0010 的子程序是 SS2(“echo”)，该子程序也许是多余的。排入这个子程序是为了便于处理某些控制字符。一般情况下该子程序只将给定于寄存器 C 中的字符输出至外设；但是若该字符是十六进制 1B(“换码”)，则该子程序输出十六进制 24(“$”)；若该字符是十六进制 0D(“回车”)，则连续输出两个字符 0D(“回车”)与 0A(“换行”)。

入口为 0020 的子程序是 SS4(“in-out”)，该子程序实质上是先调用“A:＝inchar”，再调用“echo”。因此调用“in-out”时除将一个字符读入计算机以外。还将其打在纸上或在显示屏上显示出来以供验证。利用该子程序可通过一次键盘操作达到换行的目的，并能显示出操作“换码”键的地方（见下文 I、D 部分）。

有些系统具有自动的“返回”或“回送”功能。但是，有些情况并不希望有这个功能。所以没有自动回送功能的方案更可取些，因为在这种情况下可通过程序控制产生回送。

其余两个专用子程序 SS5 与 SS6 非常简单。“ASCII-Hex”子程序将给定在 A 中的 ASCII 值（其值在 30～39、41～4F 之

内)转换成对应的十六进制值(00～0F)。"Hex-ASCII"子程序的作用与此相反。

监督程序将调用全部专用子程序；这些子程序当然也可供其它程序使用。

8.2.3 监督程序——分支部分

本节先讨论 I、S、D、m 分支，然后再讨论分别从 00C4、00CA 地址开始的 G、R 分支。

I("输入", 入口 00E3)

该分支开始时用入口为 01F7 的子程序读出起始地址。然后重复执行下述循环操作读入一对十六进制数字、将其合成一个字节并存入该地址内，然后将地址加一。字节之间打入的空格、逗号与回车符均不预理睬(因此可用这些字符来控制打印记录的格式)，打入"换码"字符时使操作结束。

S("代换", 入口 0106)

该分支开始时也是用入口为 01F7 的子程序读出起始地址。若打入空格或逗号(在 01D9 存有 JSS4)，机器将打印规定地址单元的内容，然后打一个连字符(减号"-")；若打入任何其它字符，则结束操作。在打出"-"以后，用户可打入一对十六进制数字，这对数字将合成为一个字节并取代刚才显示的值。或者在打入这对十六进制数字后，用户可打入(i)一个空格或逗号，此时机器将显示下一个较高地址单元的内容，并可继续如上所述采用该值或写入新值；(ii) 另一个字符，此时该操作结束。

D("显示"或"打印", 入口 0133)

这个分支在监督程序中用来以十六进制打印存贮器内容(如本章末尾的打印输出表所示)。正如读者看到的那样，输出清单中的一行由存放在连续单元中的十六个字节组成，中间用空格分隔开，每行开头都打印该行起始单元的地址。

这个分支开始时读入两个地址(两个地址之间用空格或逗号隔开)。若(i)打入的地址不正确,(ii)第一个地址后不是空格或逗号而是其它字符,(iii)第二个地址低于第一个地址,则操作无效。若操作正确,则从 014D 进入一个循环,其操作结果是显示存贮器中从第一个地址到第二个地址(包括这两个地址)的所有单元的内容。该循环先打印出下一个单元的地址,然后进入一个内部循环(0158～017B),打印连续单元内的值,直至打印的单元地址是十六的倍数为止。此时,内部循环返回至外部循环。这个操作一般继续执行到打印完结束地址中的内容为止,但是还有一组指令(015F～016E)用于在任意时刻中止显示操作。每通过这段指令时就检查键盘接口(见上文 SS7)。若 (i) FB 转接口的第 1 位为 1,以及 (ii) FA 转接口的值为 1B(即按下的是"换码"键),则显示将立即停止,并从 0087 开始执行监督程序的主循环(016C)。

M("**复制**", **入口 0180**)

该分支执行的是 § 5.2 讨论的那种"字组复制"操作。该分支从 0180 开始,读入三个地址,各地址之间由空格或逗号分开。这三个地址分别是源起始地址、源结束地址与目的起始地址(此程序中用源结束地址代替 § 5.2 例子中所需的复制的字节数)。若第二个地址小于第一个地址,则由一个错误出口转至 00B4*。若不出错,则程序将两个起始地址进行比较,并根据比较结果决定复制过程是向前(01B2)还是向后(01C4)。在后一种情况下必须执行一些简单的计算(01C4～01D6)以建立新的起始与结束条件。

其余的监督程序操作 G(执行)、R (继续执行程序)与 X(检查寄存器)是相互关联的,也许比上述各个分支更难于理解。

* 此处原文误为 0034——译者。

任何程序(例如汇编程序或 RAM 中待测试的用户程序)都应该象是调用 0008 处的子程序那样，以 JSS1(CF)调用监督程序作为结尾。正如上文§8.2.1 所解释的，这样就会在调用时把寄存器(AF、BC、DE、HL、SP、PC)的内容存入"安全"单元 M17F4～M17FF。这些单元可通过 G、R、X 操作来存取。

R("继续执行程序",00CA)

将保存在存贮器中的寄存器值恢复到寄存器中去的过程如下。首先将堆栈指示器置成 17F4,从而可将 DE、BC、AF 的值弹出堆栈。然后将 MM17FE 中的值经由 HL 恢复到堆栈指示器中去。再将程序计数器的值 (MM17FC) 存入堆栈，并恢复 HL 的值(MM17FA)。由最后的"RET"(00E0)指令完成恢复程序计数器的值的工作。

最后,返回到调用监督程序的程序,从监督程序调用指令后面的一条指令继续执行下去，且寄存器中的值保持调用时的状态(监督程序的性质就好象是一个完全透明的子程序)。

G("执行", 入口 0DC4)

"执行"操作是 R 分支的变形，用于从规定点进入某个程序,特别是可以从较适宜的入口进入某个程序。该分支在执行 R 分支之前先读入一个入口地址。该地址值立即被写入 MM17FC,改变其中所存的程序计数器值。

虽然精确地说，一条完整的"执行"指令(例如"G1234") 的作用是"寄存器保持其在最后一次调用 0008 处的子程序之前的数值，转移至 1234"，但是常常可以将该指令的作用看作是"进入从 1234 开始的程序"。

监督程序在 G 与 R 命令中严格保证的"透明性"会受到最后介绍的监督程序操作-X 操作的破坏。X 操作的作用是显示所保存的寄存器值,并可通过键盘改变其中的某些内容。

X("检查寄存器",入口 **0215**)

这是监督程序中最复杂的一个操作。该分支本身需要使用几个子程序。其操作过程是先在第一行上打印出标志与寄存器的名称，然后就在各个名称下打印存贮器中保存的与之对应的各项数值。标志以二进制值(0、1)打印；其余的数值用十六进制打印。然后用键盘上的空格键作为"标志"("tab")键，使滑架(或光标)对准需修改的具体数值。然后可在原有值下打入新值，从而代替原来的值。任何时刻只要打入"回车"，就会使控制返回至监督程序的主循环(0087)去。

X 分支占用存贮器 0215～021A 以及 0245～0311 单元，其子程序存在 0312～0359 中。在打印所保存的值时，打印标志值的部分(024D～025F)将五次调用 0312～031E 中的子程序。每次调用时，该子程序找到一个标志值并将其打印出来。打印其它寄存器的部分(0260～0287)更简单一些。其余部分(0288～02E9)则稍微复杂一些。在滑架(或光标)对准 N 标志值时，可打入五个新的二进制值取代原来的标志值，或者再按一次空格键使滑架(或光标)对准 A 寄存器的值，或者可使用回车键结束整个操作。可对 A、BC、DE、HL、SP 与 PC 值重复进行这种选择过程。通过连续调用 031F～032C 中的子程序，可将新的标志值读入机器并组合成一个新的标志寄存器值存入 M17F8。每个寄存器对或双字长寄存器都调用 033C～0359 中的子程序进行处理。每次调用该子程序时都可以象在标志值的情况下那样对三种选择进行处理。值得注意的是：0348 的条件转移指令，若执行该指令，则堆栈中尚留有一个未使用过的返回地址。但是，堆栈指示器立刻就会复位，所以这一点没什么关系。

8.2.4 其它子程序

在监督程序 PROM 中，还有一些子程序，供监督程序及其

它软件调用，现将这些子程序入口及其功能简单叙述如下：

01E6: “十六进制检查”，可用于“ASCII-hex”之后或“Hex-ASCII”之前。若寄存器 A 中的值在 00～0F 范围内（包括 00 和 0F 在内），则该子程序无影响；若 A 中的值超出该范围，则从一个出错出口转至监督程序（00B4）。

0220: “A:=in byte”，读入并检查两个十六进制数字。若一切正常，则将其装配成一个字节存入寄存器 A。

0204: “outbyte(A)”，将寄存器 A 中的字节拆成两个十六进制数字，然后打印出来。

0200: “CRLF”（换行），只是在 C 中存入 0D（回车）并调用“echo”（回送）子程序。

01F7: “HL:=in address”，只是两次调用“A:=in byte”，分别将输入值存入 H 与 L。

005F: “Outstring(B)”：在 B 中规定一个 n，在 HL 中规定一个地址 a，则该子程序将打印 n 个字符，字符的 ASCII 值存在 M(a)，M($a+1$)，…，M($a+n-1$)中。

03F6: “Outstring(ESC)”：在 HL 中规定一个地址 a，该子程序将打印一串字符，字符的 ASCII 值在从 M(a)开始的一些单元中，打印过程一直进行到遇到 1B（换码）值为止。此时该子程序不打印 1B 而返回原程序。

03EC: “Instring(ESC)”：在 HL 中规定一个地址 a，该子程序读入一串字符，并将其 ASCII 值存入 M(a)开始的单元中。当读入 1B（换码）时就将 1B 也存贮起来，可供以后“Outstring(ESC)”子程序使用，并返回原程序。

0231: “HL−DE”，根据给定的 HL 与 DE 中的两个无符号双字长整数（一般是地址）之差 HL−DE 设置条件标志 N、Z。

返回主程序时若 HL－DE＝0(HL＝DE)则

N＝0, Z－1;

若 HL－DE＞0(HL＞DE)则

N＝0, Z＝0;

若 HL－DE＜0(HL＜DE)则

N＝1, Z＝0。

最后,在 01F4 有一条转至 0010("echo"子程序的第一条指令)的无条件转移指令,在 021B 有一条转至 0038("A:＝inchar"子程序的第一条指令) 的无条件转移指令。 提供这两条指令是为了使某些原来用英特尔公司 SDK 监督程序(包括汇编程序)编写的程序尽量少改动就可借助于这个新的监督程序投入运行。

8.2.5 PROM 编程器用的程序(0360～03EB)

这里采用的技术已在§5.7 中介绍了。 这两个程序(0360 的"PROM 编程程序"与 03C0 的"PROM 读出程序")是配合某种型号的 PROM 编程器对特定型号的 PROM(如 2708 或 8708 等)编程用的,编程器必须以规定的方式与设备号为 F4～F7 的 8255 接口器件相连。 PROM 编程器的接口特点及其接法可见§5.7。

"PROM 编程程序"(入口为 0360)的工作分成三个阶段。第一个阶段是每次一个字节地读插在编程器上认为是空白的 PROM。只要读入的字节是十六进制 FF (全 1, 即"擦净"状态),读入过程就继续下去。当读入一个非 FF 字节时程序停机(03BA),并点亮编程器上"未擦净"报警灯。在成功地检查完整块 PROM 后,接口被安排成写入状态(0371～0376), 并开始编程操作。 对于 8708 与 2708 型器件而言, 编程过程采用内外循环结合的方法。对 PROM 的每 1024 个字节执行一次内循环

(0384～039A)，而外循环(037D～039E)重复200遍。这是由PROM的物理性质决定的。内循环中是一个计时循环(见§5.7)，该计时循环测量编程脉冲的宽度。这个过程的最后部分是检验，此时要求将接口安排成读入状态(039F～03A2)。新编程的PROM中的每个字节被读回寄存器A，与其原始值(仍在RAM的1000～13FF单元中)进行比较。若发现任何不符之处，则点亮编程器上的"出错"灯(03BB～03BE)，并调用监督程序(03BF)；然后可用监督程序的X操作命令来检查不相符的单元(HL)。

"PROM读入程序"(入口03C0)主要是一个简单的循环，逐个字节地读入插在PROM编程器上的PROM中的信息，并将其存入RAM中1000～13FF单元(与PROM编程程序使用的单元相同)。

§8.3 汇编程序(0400～07E2、0800～0BDF、0C00～0D6E、0D85～0DC9)

汇编程序未使用07E3～07FF与0BE0～0BFF的原因可参阅§8.1。

汇编程序所用的RAM单元具体情况如下：

1000～100F 存贮正在处理的源程序的字符(ASCII值)。

1010～1011 目的程序计数器(即存贮下一个汇编成的字节的单元地址)，预置成1180。

1012～1013 转移表计数器(存贮转移表下一项内容的单元地址)。

1014 标记(预置为00；由被编译的第一条指令置成01)。

1015～101F　未用

1020～103D　X表(X0总是从1000开始；MM(1020+2(h−1))是Xh的第一个元素运行时的地址(1≤h≤F)。

103E～103F　被编译的程序第一条指令运行时的地址。

1040～10BF　标号表：标号L运行时的地址在MM(1040+2L)中(0≤L≤3F)。

10C0～117F　转移表。该表中每一项元素由三个字节组成。这三个字节是一个标号与一个地址，该地址中有一条涉及标号的指令。汇编时若发现一个操作涉及的标号尚未出现过，则在该表中增加一项。在完成汇编(FINISH)时，借助于这张表以及标号表提供遗漏的地址。转移表最多可容纳64(十进制)项内容。

1180以上　汇编成的目的程序的指令与常数。这个部分的长度受堆栈范围限制。

汇编程序的入口是0C00，通常是通过监督程序指令G0C00进入汇编程序的。

汇编程序的第一部分是：

0C00～0C19：预置：X表、标号表与转移表被清除(置成"0")，标记(1014)被清除并转移至0400。

0400～0457：读指令例行程序：滤除任何"空格"或"换行"字符，读入形成一条源程序的字符并将其存入M1000以上的单元内。在这个过程中只要操作员在键盘上按下"删除"键，就会使本程序打印一个"？"并重新从0406开始执行。源指令以";"或"回车"结束。遇到这两个字符后该例行程序打印一个";"，然后打出足够的空格使滑架(或光标)对准距左边

沿有二十个字符的位置。

这时由汇编程序处理刚读入的指令(0C20 以上)。在产生目的程序字节(最多三个字节)后，汇编程序将按适当情况转到输出例行程序的若干入口点之一继续执行下去。

0460～0485　输出例行程序：本程序将利用 "out object byte"(输出目的字节)子程序(0458～045B、0486～0499)。该子程序将存在 A 中的字节存入目的程序的末尾(1180 以上)，并打印其值与一个空格。

输出例行程序有几个入口：

047A——输出存在寄存器 A 中的一个字节。

0479——输出一个字节，该字节由 A 与 C 中的值相加而成。

0475——输出存放在寄存器 B 中的一个字节。

0472——顺序输出 A、B 中的两个字节。

0467——顺序输出 A、B、E 中的三个字节。

0463——(出错入口)输出三个全零字节(00 00 00)。

0460——若标志 Z 被置位(Z=1)，则输出寄存器 B 中的一个字节。

若 Z 复位(Z=0)，则输出三个全零字节(000000)。

047D——不输出任何字节(处理的是 X 或 VALUES 命令)。

在输出所需个数的字节后，该例行程序检查刚处理的指令的最后一个字符后是否立即出现分号";"。若未出现，则打印"？"。在0436，该程序打印"换行"，并准备从 0406 开始读出另一条指令。

0C20 以上：处理一条指令

这个部分是汇编程序的主要部分，是一个树形结构，每个分支处理以相同字母开头的全部指令。这些分支的起始单元如下：

A	04A6	F	0CC1	K	07C7	S	08C7
B	0585	G	0633	L	0DA3	V	0D05
C	060A	H	0654	M	0DAF	X	0C2E
D	0616	I	0725	N	09A1	labels	0C93
E	0976	J	074B	R	0895		

每个分支将顺序检查指令中的各个字符，最后通过转入输出例行程序适当的入口点而离开该分支。下面将对各分支可能调用的子程序作出说明。

有几个分支需要特别进行注释。

只有当标记为零时才能进入X分支。处理一条X命令(暂存存贮模块的说明)将在X表中增添一项内容。不输出任何字节。遇到不以X开头的第一条指令时标记被置成01(非零)，并在处理该指令之前计算好所有已说明的X字组的起始地址(0C59)。此后，将拒绝接受任何以X开头的指令。第一个目标字节运行时的地址就是紧接着最后一个X字组的单元地址。

V分支("VALUES")不产生任何指令字节，但是可将一串ASCII字符或十六进制值编入目的程序。

标号分支将把一项内容存入标号表。在检查了标号后是否紧跟有冒号(":")以后，标号分支转移至04A0开始处理加有该标号的指令。

F分支不产生任何目的字节。该分支扫描转移表并完成目的程序中所有的向前引用，最后以调用监督程序为结束(0CEE)。

汇编程序用的子程序

"ASCII-Hex(assembler)"(汇编程序用ASCII-Hex转换子程序，入口0B00)

若将一个ASCII值放入寄存器A中，该子程序将在A与B中产生对应的十六进制数字；但是若这个ASCII值不是十六进制数字的一个值，则该子程序将FF存入B。

"operator"(操作符，入口0B20)

在进入该子程序之前，HL指向源指令中下一个字符；接着的一、二或三个字符将作为操作符用。返回时若已发现一个操作符，则C中的结果以及HL的增量如下所示：

若为		则		
	":+"		C:=00	HL:+1
	":++"		C:=01	HL:+2
	":-"		C:=02	HL:+1
	":--"		C:=03	HL:+2
	":&"		C:=04	HL:+1
	":#"		C:=05	HL:+1
	":U"		C:=06	HL:+1
	"_"		C:=07	HL:+0
	":="		C:=08	HL:+1

若未发现任何操作符，则C:=FF。在这种情况下HL加一，表示发现了冒号":"，但是下一个字符不是+、-、&、#、U、=等字符之一。

"Construct J"(构成J字节，入口为0B60)

源指令的下面两个字符(应该是十六进制数字)的ASCII码被转换成十六进制数字，并组成一个单字节值存在寄存器A中。若两个字符之一不是十六进制数字字符，则C中存入出错值FF。

"Single-length register or J"(单字长寄存器或J，入口为0B1B)

调用该子程序时接下来的一、二个字符应该是(i)单字长寄

存器的名称或M，(ii) 加一或减一，(iii) 构成一个J字节值的两个十六进制数字。转出该子程序时结果是：

(i) 若名称为"B"，则C=00；若为"C"，则C=01；若为"D"，则C=02；若为"E"，则C=03；若为"H"，则C=04；若为"L"则C=05；若为"M"，则C=06；若为"A"，则C=07。

(ii) 若为"1"，则C=09。

(iii) 若能形成J字节，则C=08，字节的值存放在A中，HL加一。否则C=FF(出错值)。

"Condition"(条件，入口为0BC0)

源指令中接下来的一、二个字符应该是条件标志的名称。转出该子程序时其结果为：

若为NZ 则C=00	若为NP 则C=04
若为Z 则C=01	若为P 则C=05
若为NK 则C=02	若为NN 则C=06
若为K 则C=03	若为N 则C=07

否则C=FF(出错值)。

"Form JK"(形成J、K字节，入口为0A00)

要求源指令中接下来的四个字符是十六进制数字。若是十六进制数字，则对前两个字符进行转换并组合成K字节值存入寄存器E中，后两个字符形成J字节值存在寄存器A中。

"Construct JK"(构成J、K字节，入口为0B80)

这是一个大而复杂的子程序，处理在汇编程序中可构成地址的各种数据。结果一般是绝对地址的低字节(J字节)存放在A中，高字节(K字节)存放在E中。

若为直接地址(四个十六进制数字"hhhh")，A与E中就是"Form JK"子程序产生的值。

若为标号形式的地址"(Lhh)"，则从标号表中查出标号的

地址。若有此标号地址，则由 A、E 给出。若无此地址，则在转移表中增加一项；E 中给出“FD”(向前引用)值。

若为 X 形式的地址(“Xh”或“Xh+hh”或“Xh−hh”)，则由 X 表给出有关的 X 字组的地址，若有必要可加上或减去位移量。

若为 I 形式的地址(“I+hh”或“I−hh”)，则 A、E 中的地址是由 MM1010、MM103E 与给定的位移量经计算而得的地址(见本节开头有关 RAM 用途的说明)。

§8.4 反汇编程序(2800～2B7F)

反汇编程序用于打印输出本章后面所示的那种软件文本。对于每条指令，先打印存贮单元地址，然后打印操作码(I 字节)，最后用本书所用的书写格式打印出指令。进入反汇编程序时应将起始地址存放在 HL 中，结束地址存在 DE 中。然后反汇编程序将从起始地址开始处理存贮器中的内容，一直处理至指令的 I 字节的地址等于或大于结束地址为止。

反汇编程序认为它所处理的文本只由连续的指令组成。若反汇编程序处理到存有其它信息(例如 ASCII 文本或浮点数)的存贮区，则其打印输出是毫无意义的。

反汇编程序的结构是简单的计数循环。每次通过这个循环时，在打印完地址与 I 字节后要用 I 字节的最高两位选择执行四个主分支之一。每个分支与第三章给出的卡诺图中的一个象限相对应。反汇编程序每个分支内的结构都根据卡诺图的结构进行安排。由于图的左下角的结构是完全规则的，所以处理 I=10…… 的分支(28A3)非常简单。其左上角被分成几个部分，每部分包括若干单元。因此处理 I=00…… 的分支(2960)中包括一些连续的段，一段对应于一个 I 字节值。

指令的书写形式可包括三部分，在有些情况下其中一个或两个部分可略去。这三个部分是“首字符串”、“J 或 KJ 值”以及“尾字符串”。例如，指令“MM2345:=HL”三个部分都有，这三部分分别为“MM”、“2345”以及“:=HL”；指令“C:=25”只有前两个部分，分别是“C:=”与“25”；指令“K:=1”将作为只有一个尾字符串处理。

查阅带注释的文本可以很容易地发现打印某条指令的方法。许多分支结束时都转移至 2828～284F 范围内的某个地址去，以便打印尾字符串。读者可发现其中使用了许多技巧。将数据相继地存入寄存器对 DE 是一种节省存贮空间的技巧，反汇编程序在 283A～284C 中采用了这种技巧，在到达 284C 时，寄存器 B 中存有一个有限范围内的某个数。若转入 2838，则将把 02 存入 B，继而将一个无用的值存入空着的寄存器对 DE。转移到 2841（这些无用值之一的第二个字节）将把 05 存入寄存器 B，而把较简单的无用值存入 DE。只有当有必要节省存贮空间时才有理由使用这种技巧（现在就有这种必要），且只能使用到这里指出的与所解释的程度。

反汇编程序的子程序有“Find name of register”（找寄存器名称，入口为 2860）、“Print value of KJ”（打印 KJ 值，入口 286F）、“Find name of register pair”（找寄存器对名称，入口为 2940）、“Ordinary register pair”（普通寄存器对，入口为 2936）、“Stacking register pair”（找存入堆栈的寄存器对名称，入口为 2955）、“print name of Condition”（打印条件名称，入口为 2A56）。这些子程序都很简单明了。

“Find and print symbols for *op*”（找出并打印操作符，入口为 2900）子程序被编成一对相互嵌套的子程序（在 2905 处调用内部的一个子程序），以便利用 8080 的单字节条件返回指令。

若将其编成单级子程序，则在存有条件返回或无条件返回指令之处(最后一条除外)都应使用一条三字节的转移指令。

§8.5 整数与定点算术运算的子程序

本节介绍的许多子程序已在第四章介绍了。这里将列出这些子程序，并给出第四章中与这些子程序有关的节号，以及§8.8中写不下的其它注释。

07E6/8　"out decimal fraction(HL)"(输出十进制小数)

HL中给出 $X=f\times 2^{16}$ $(0\leqslant X\leqslant 65535)$，该子程序以5个(07E6)或B个(07E8)十进制小数位打印出f的近似值。

0BE0　"HL:=sqrt(BC)×2⁷"

本子程序的算法是牛顿算法(见§6.3)。双字长除法的 p_0 值(见§4.5.5.1)是14(十进制)。当连续两次的近似值相等(0BF3)或只差最低位数字(0BF5～0BFA)时迭代过程结束。

0F18～0F4F　单字长乘法(见§4.5.3)。

0F50～0F9A　单字长除法(见§4.5.5)。

0F9B～0FF0　双字长除法(见§4.5.5)。

0FF1/2　"HL:/2"

由0FF1进入本子程序时在最低位加一，完成四舍五入(见§4.6.4)。

2400～243E　双字长乘法，所用的方法与§4.5.3介绍的方法相同，但是尽可能使用双字长寄存器及双字长操作。(见§4.5.4)。

2440　乘法（15 位二进制小数位）——使用双字长整数乘法（入口为 2420）给出 $(x\times 2^{15})\times(y\times 2^{15})=xy\times 2^{30}$，然后左移一位形成 $xy\times 2^{31}$；再四舍五入成双字长数，给出所需的结果 $(xy\times 2^{31})\times 2^{-6}=xy\times 2^{15}$。

2452　多项式（15 位二进制小数位，见 § 6.3）。

2470　"HL: = sin (HL) 15b. p"

若在 HL 中给出 $x\times 2^{15}$，则本子程序可对下式进行计算：

$$\left(2\times x\times\left(\left(\left(-\frac{x^2}{2\times 7!}+\frac{1}{2\times 5!}\right)\times x^2-\frac{1}{2\times 3!}\right)\times x^2+\frac{1}{2}\right)\times 2^{15}\right)$$

注：$-2^{14}\div 7!=-3$(FFFD)

$+2^{14}\div 5!=+137$(0089)

$-2^{14}\div 3!=-2731$(F555)

$+2^{14}=+16384$(4000)

2490　"HL: = cos (HL) 15b. p"

$\cos(x)=(1-\sin^2(x))^{\frac{1}{2}}$

250A—2567　三倍字长带符号整数乘法（见 § 4.5.4 与 § 4.7.3.4）。

2568—25A0　三倍字长算术右移（见 § 4.7.3.1）。

25E4　四舍五入（44b. p 至 22b. p，见 § 4.7.3.4）。

§ 8.6　浮点子程序

250A—25A1 与 25D7—2602 内的三倍字长子程序是准备供浮点子程序集作为辅助子程序用的（见 § 4.7.3）。规格化

(25A1—25D7)已在§4.7.2介绍了。

§4.7.4介绍了浮点算术运算子程序(2608—2629、263E—2744、24A7—24BF)，§4.7.5介绍了浮点输入与输出子程序(2745—27FF、2B80—2BFF、262A—263D)。

§8.7 错误与缺点

在§8.1中曾提到监督程序与反汇编程序中原来各有一个错误，用了几个月才查出。

监督程序中的那个错误实质上是遗漏了0081单元中的指令"SP:=17F4"。这条指令的目的是保证将调用"out string"(输出字符串)子程序及其内部调用"out char"(输出字符)子程序时的返回地址都写入RAM，而不会写入PROM某处或无处可写。在补上这条指令之前，监督程序在大部分时间内工作都是正确的。非常凑巧，几乎每次开机时堆栈指示器中的"随机"值总是指向RAM中的地址。偶尔当堆栈指示器中的"随机"值不指向RAM时，就不能正确地打印"MONITOR"这个信息，因此就责怪电源馈线上有浪涌或电路板上绕接点接触不良。后来在另一个系统中安装了同样的监督程序PROM。但是在该系统中只是偶然能够工作。原始的与复制的监督程序PROM几乎都无法在第二个系统中工作；而在第一个系统中却几乎总是能工作。程序中的这个错误一经查出，立即就知道两个机器原来都没有问题。

反汇编程序中的错误是遗漏了另一条指令——K:=1(十六进制37)。大概在前八个月中使用反汇编程序的任何人都没有准备设置进位标志，所以这个错误没有暴露出来。

任何有经验的程序员只要仔细地寻找一下就很容易发现该软件文本有修正与增删的痕迹。

虽然这个软件在规划时只有一个梗概，但是最后的细节就是在这个基础上生成的。现在正在对这个软件进行修改和编辑，并考虑到了高级语言程序设计的需要(见§6.3)。

在该软件中有几处是可能节省监督程序PROM的指令空间的。例如，入口为0231的子程序有22个字节(现在看起来似乎相当好)，但可以用§4.5.2给出的只有6个字节的子程序取而代之。节省的空间足以将反汇编程序编成为监督程序的一个分支，而且还可编入对2708/8708以外的其它类型的PROM进行编程与读入的程序，这一切都编在同一块1024字节的PROM内。

这个软件中最早编写的部分是汇编程序，这也是最需要改进的部分。肯定可以用小于2K字节的存贮空间提供现有的功能，也许还可增加少量的功能(例如采用相对于标号的地址以及采用十进制和浮点值)。更重要的是可对汇编程序进行改编，使其成为§6.3所讨论的高级程序设计语言的基础。首先需增加一块2K字节的存贮空间。

然后将算术子程序与其它子程序整齐地编入2K字节的空间内，使第二个版本总共有8K字节(8×2708或4×2716)。"15b. p."型的子程序可以删去，用浮点多项式和三角函数子程序取而代之。修改时当然应该对现有子程序的明显的缺点(例如对输入的十进制整数的检查不完善、除法以后四舍五入的处理以及§4.7.5提及的浮点输出问题)加以补救。

§8.8 软件的带注释的文本

下面是由反汇编程序产生的整个软件的打印输出，总共有6K字节(0000—0FFF以及2400—2BFF)。

指令的格式是：I字节的存贮单元，I字节的数值，指令的书

写形式。如果有 J 和 K 字节的话，将在书写形式中表现出来。

ASCII 文本部分(即一组以 ASCII 码存贮的字符) 的格式是：第一个字符的存贮单元，引号，文本，引号。

数值部分(二进制常数)的格式是：第一个字节的存贮单元，符号“#”，十六进制值(每个字节两个十六进制数字)，符号“#”。

为便于打印，将文本分成了长度适中的若干段，均按地址值顺序排列，但下述几条指令例外：

0010—0066 跟在 00C3 后

035A—035F 跟在 0244 后

01E6—0230 跟在 035F 后

07E6—07EF 跟在 0E71 后

24A7—24C0 跟在 2744 后

我们希望文本的注释能说明些问题，读者可以结合 §8.2~§8.6 各节中的一般介绍进行阅读。

```
  G2800
0000   3E   A:=CF;              (SS0)监督程序的“RESET”(复位)入口
0002   D3   GFB:=A;
0004   3E   A:=27;              预置 TTY/VDU 接口
0006   D3   GFB:=A;
0008   22   MM17FA:=HL;         (SS1)监督程序的正常入口
000B   F5   ST:=AF;
000C   E1   HL:=ST;
000D   C3   J0067;

MONITOR
>
  G2800
0067   22   MM17F8:=HL;
```

```
006A  F1  HL:=ST;
006B  22  MM17FC:=HL;
006E  21  HL:=0000;
0071  39  HL:+SP;
0072  22  MM17FE:=HL;
0075  60  H:=B;
0076  69  L:=C;
0077  22  MM17F6:=HL;
007A  EB  HL:=:DE;
007B  22  MM17F4:=HL
007E  21  HL:=00BA;
0081  31  SP:=17F4;
0084  CD  JS03F6;
0087  31  SP:=17F4;
008A  CD  JS0200;
008D  0E  C:=3E;
008F  DF  JSS3;
0090  E7  JSS4;
0091  FE  A-44;
0093  CA  JZ0133;
0096  FE  A-47;
0098  CA  JZ00C4;
009B  FE  A-49;
009D  CA  JZ00E3;
00A0  FE  A-4D;
00A2  CA  JZ0180;
00A5  FE  A-52;
00A7  CA  JZ00CA;
00AA  FE  A-53;
00AC  CA  JZ0106;
00AF  FE  A-58;
```

保存寄存器内容：

```
MM17F4:=DE
MM17F6:=BC
MM17F8:=AF
MM17FA:=HL
MM17FC:=PC
MM17FE:=SP
```

打印信息“MONITOR”

执行正常的监督程序命令后重入监督程序；预置堆栈指示器；换一行打印提示符号“>”

读入下一条监督程序命令

若为“D”，转至 0133

若为“G”，转至 00C4

若为“I”，转至 00E3

若为“M”，转至 0180

若为“R”，转至 00CA

若为“S”，转至 0106

```
00B1   CA   JZ0215;          若为"X",转至 0215
00B4   0E   C:=3F;
00B6   DF   JSS3;            出错:打印"?"
00B7   C3   J0087;           循环,执行下一条监督程序命令

MONITOR
>
00BA "MONITOR"
  G2800
0010   41   B:=C;            (SS2)"回送"子程序入口
0011   3E   A:=1B;
0013   B8   A-B;
0014   C3   J0050;
0017   00   NULL;
0018   C3   J0045;           (SS3)"输出字符(C)"子程序入口
001B   0E   C:=0A;
001D   DF   JSS3;
001E   48   C:=B;
001F   C9   RET;
0020   FE   JSS7;           (SS4)"输入-输出"子程序入口
0021   4F   C:=A;
0022   D7   JSS2;
0023   79   A:=C;
0024   C9   RET;
0025   00   NULL;
0026   00   NULL;
0027   00   NULL;
0028   D6   A:-30;          (SS5)"ASCII-十六进制转换"子程序
                            入口
002A   FE   A-0A;
002C   F8   RN;
```

```
002D  D6  A:-07;
002F  C9  RET;
0030  C6  A:+30;          (SS6)"十六进制-ASCII 转换"子程序
                          入口
0032  FE  A-3A;
0034  F8  RN;
0035  C6  A:+07;
0037  C9  RET;
0038  DB  A:=GFB;         (SS7)"A:=输入字符"子程序入口
003A  E6  A:&02;
003C  CA  JZ0038;
003F  DB  A:=GFA;
0041  E6  A:&7F;
0043  4F  C:=A;
0044  C9  RET;
0045  DE  A:=GFB;         SS3 的继续
0047  E6  A:&01;
0049  CA  JZ0045;
004C  79  A:=C;
004D  D3  GFA:=A;
004F  C9  RET;
0050  C2  JNZ0055;       SS2 的继续
0053  0E  C:=24;
0055  DF  JSS3;
0056  3E  A:=0D;
0058  B8  A-B;
0059  CA  JZ001B;
005C  48  C:=B;
005D  C9  RET;
005E  00  NULL;
005F  4E  C:=M;           "输出字符串(B)"子程序入口
```

```
0060   DF   JSS3;
0061   23   HL:+1;
0062   05   B:-1;
0063   C2   JNZ005F;
0066   C9   RET;

MONITOR
>
  G2800
00C4   CD   JS01F7;          "G"MM17FC:=输入地址
00C7   22   MM17FC:=HL;
00CA   F3   I:=0;            "R"
00CB   CD JS0200;            换行
00CE   31   SP:=17F4;        ┐
00D1   D1   DE:=ST;          │
00D2   C1   BC:=ST;          │
00D3   F1   AF:=ST;          │
00D4   2A   HL:=MM17FE;      ├ 恢复寄存器内容 ┬ DE:=MM17F4
00D7   F9   SP:=HL;          │                ├ BC:=MM17F6
00D8   2A   HL:=MM17FC;      │                ├ AF:=MM17F8
00DB   E5   ST:=HL;          │                ├ SP:=MM17FE
00DC   2A   HL:=MM17FA;      ┘                ├ HL:=MM17FA
00DF   FB   I:=1;                             └ PC:=MM17FC
00ED   C9   RET;
00E1   00   NULL;
00E2   00   NULL;
00E3   CD   JS01F7;          "I"HL:=输入地址
00E6   CD   JS0200;          换行
00E9   E7   JSS4;            "输入-输出"
00EA   FE   A-20;
00EC   CA   JZ00E9;
```

```
00EF  FE  A-2C;
00F1  CA  JZ00E9;        滤除空格,逗号,回车符
00F4  FE  A-0D;
00F6  CA  JZ00E9;
00F9  FE  A-1B;
00FB  CA  JZ0087;        至"换码"结束
00FE  CD  JS0221;        读第二个字符并形成字节
0101  77  M:=A;          存贮并使寻址寄存器加一
0102  23  HL:+1;
0103  C3  J00E9;         循环,准备下一次输入
0106  CD  JS01F7;        "S"HL:=输入地址
0109  E7  JSS4;          "输入-输出"
010A  FE  A-20;
010C  CA  JZ0114;
010F  FE  A-2C;
0111  C2  JNZ0087;       若非空格或逗号,则结束
0114  7E  A:=M;          显示被寻址的单元的内容
0115  CD  JS0204;
0118  0E  C:=2D;         显示"-"
011A  DF  JSS3;
011B  E7  JSS4;          "输入-输出"
011C  47  B:=A;
011D  EF  JSS5;          "ASCII-十六进制转换"
011E  FE  A-00;          若非十六进制码,转至012E
0120  FA  JN012E;
0123  FE  A-10;
0125  F2  JNN012E;
0128  CD  JS0225;        否则完成整个字节
012B  77  M:=A;          存贮新值
012C  E7  JSS4;          "输入-输出"
012D  48  B:=A;
```

```
012E  78  A:=B;
012F  23  HL:+1;          寻址寄存器加一
0130  C3  J010A;          循环(继续或结束)

MONITOR
>
 G2800
0133  CD  JS01F7;         "D"
0136  E5  ST:=HL;         将起始地址读入堆栈
0137  E7  JSS4;           读入并打印分隔符
0138  FE  A-20;
013A  CA  JZ0142;
013D  FE  A-2C;
013F  C2  JNZ00B4;        若分隔符不是空格或逗号,则出错
0142  CD  JS01F7;         读入结束地址
0145  D1  DE:=ST;
0146  CD  JS0231;         若小于起始地址,则出错
0149  FA  JN00B4;
014C  EB  HL:=:DE;        HL 中是当前地址,DE 中是结束地址
014D  CD  JS0200;         换行
0150  7C  A:=H;
0151  CD  JS0204;         打印当前地址
0154  7D  A:=L;
0155  CD  JS0204;
0158  0E  C:=20;          打印空格
015A  DE  JSS3;
015B  7E  A:=M;           打印当前字节
015C  CD  JS0204;
015F  DB  A:=GFB;
0161  E6  A:&02;          有来自键盘的输入?
0163  CA  JZ016F;             若没有,跳至 016F
```

```
0166   DB   A:=GFA;          若有,则读入该字符(见 SS7)
0168   E6   A:&7F;
016A   FE   A-1B;
016C   CA   JZ0087;         若刚读入的是"换码",则结束
016F   CD   JS0231;         若当前地址等于结束地址,则结束
0172   CA   JZ0087;
0175   23   HL:+1;          寻址寄存器加一
0176   7D   A:=L;
0177   E6   A:&0F;          新的当前地址可被 16 除尽?
0179   C2   JNZ0158;        若不能除尽,则下一个字节在同一
                              行上
017C   C3   J014D;          若能除尽,则换行
017F   00   NULL;

MONITOR
>
  G2800
0180   CD   JS01F7;         "M"
0183   E5   ST:=HL;         源起始地址入栈
0184   E7   JSS4;           读入并打印分隔符
0185   FE   A-20;
0187   CA   JZ018F;
018A   FE   A-2C;
018C   C2   JNZ00B4;        若非空格或逗号,则出错
018F   CD   JS01F7;         读源结束地址
0192   D1   DE:=ST;
0193   CD   JS0231;
0196   FA   JN00B4;         若源地址不协调,则出错
0199   E5   ST:=HL;
019A   D5   ST:=DE;
019B   E7   JSS4;           读入并打印分隔符
```

```
019C  FE  A-20;
019E  CA  JZ01A6;
01A1  FE  A-2C;
01A3  C2  JNZ00B4;          若非空格或逗号,则出错
01A6  CD  JS01F7;           目的地起始地址入栈
01A9  E5  ST:=HL;
01AA  D1  DE:=ST;
01AB  E1  HL:=ST;
01AC  CD  JS0231;           比较起始地址
01AF  FA  JN01C4;           转至 01C4,"向后"进行复制
01B2  42  B:=D;
01B3  4B  C:=E;             BC 中是目的地址
01B4  D1  DE:=ST;           DE 中是源结束地址
01B5  7E  A:=M;
01B6  02  M[BC]:=A;
01B7  C5  ST:=BC;
01B8  CD  JS0231;           复制完最后一个源地址以后结束复制
                            循环(§ 5.2)
01BB  C1  BC:=ST;
01BC  CA  JZ0087;
01BF  03  BC:+1;
01C0  23  HL:+1;
01C1  C3  J01B5;

MONITOR
>
  G2800
01C4  EB  HL:=:DE;          "向后"复制(见§ 5.2)
01C5  44  B:=H;
01C6  4D  C:=L;
01C7  7A  A:=D;
```

```
01C8  2F   A:#;
01C9  67   H:=A;          01C5～01CE(包括这两个单元)中的指
                          令等效于"HL:-DE"
01CA  7B   A:=E;
01CB  2F   A:#;
01CC  6F   L:=A;
01CD  23   HL:+1;
01CE  09   HL:+BC;
01CF  EB   HL:=:DE;
01D0  E3   HL:=:ST;
01D1  FB   HL:=:DE;
01D2  19   HL:+DE;
01D3  44   B:=H;
01D4  4D   C:=L;
01D5  E1   HL:=ST;
01D6  EB   HL:=:DE;
01D7  7E   A:=M;
01D8  02   M[BC]:=A;      01D3～01F3 中的指令与0102～01C1
                          中的类似
01D9  C5   ST:=BC;
01DA  CD   JS0231;
01DD  C1   BC:=ST;
01DE  CA   JZ0087;
01E1  0B   BC:-1;
01E2  2B   HL:-1;
01E3  C3   J01D7;

MONITOR
>
  G2800
01E6  FE   A-00;          "十六进制检查"子程序入口
```

```
01E8  FA  JN01F1;
01EB  FE  A-10;
01ED  F2  JNN01F1;
01F0  C9  RET;
01F1  C3  J00B4;
01F4  C3  J0010;          "回送"子程序(SS2)的另一个入口
01F7  CD  JS0220;         "HL:=输入地址"子程序入口
01FA  67  H:=A;
01FB  CD  JS0220;
01FE  6F  L:=A;
01FF  C9  RET;
0200  0E  C:=0D;          "换行"(CRLF)子程序入口
0202  D7  JSS2;
0203  C9  RET;
0204  47  B:=A;           "输出字节(A)"子程序入口
0205  0F  A:@R;
0206  0F  A:@R;
0207  0F  A:@R;
0208  0F  A:@R;
0209  E6  A:&0F;
020B  F7  JSS6;
020C  4F  C:=A;
020D  DF  JSS3;
020E  78  A:=B;
020F  E6  A:&0F;
0211  F7  JSS6;
0212  4F  C:=A;
0213  DF  JSS3;
0214  C9  RET;
0215  CD  JS0200;         "X"换行
0218  C3  J0245;          转至0245继续执行
```

```
021B   C3   J0038;          "A:=输入字符"子程序(SS7)的另一
                             个入口
021E   00   NULL;
021F   00   NULL;
0220   E7   JSS4;           "A:=输入字节"子程序入口
0221   EF   JSS5;
0222   CD   JS01E6;
0225   87   A:+A;
0226   87   A:+A;
0227   87   A:+A;
0228   87   A:+A;
0229   57   D:=A;
022A   E7   JSS4;
022B   EF   JSS5;
022C   CD   JS01E6;
022F   82   A:+D;
0230   C9   RET;

MONITOR
>
  G2800
0231   44   B:=H;           "HL-DE"子程序入口
0232   4A   C:=D;
0233   78   A:=B;
0234   B9   A-C;
0235   C2   JNZ023D;
0238   45   B:=L;
0239   4B   C:=E;
023A   78   A:=B;
023B   B9   A-C;
023C   C8   RZ;
```

```
023D   A9   A:#C;
023E   FA   JN035A;
0241   78   A:=B;
0242   91   A:-C;
0243   C9   RET;
0244   00   NULL;

MONITOR
>
  G2800
035A   79   A:=C;              "HL-DE"子程序的继续
035B   F6   A:U7F;
035D   C9   RET;
035E   00   NULL;
035F   00   NULL;

MONITOR
>
  G2800
0245   06   B:=28;             "X"分支的继续
0247   21   HL:=02EA;
024A   CD   JS005F;            打印标题行
024D   3A   A:=M17F8;
0250   A7   A:&A;
0251   CD   JS0313;
0254   CD   JS0313;
0257   CD   JS0312;            打印标志值
025A   CD   JS0312;
025D   CD   JS0312;
0260   0E   C:=20;
0262   DF   JSS3;
```

```
0263  DF  JSS3;
0264  3A  A:=M17F9;
0267  CD  JS0204;          打印 A(M17F9)的值
026A  21  HL:=17F6;
026D  CD  JS032D;              BC(MM17F6)
0270  21  HL:=17F4;
0273  CD  JS032D;              DE(MM17F4)
0276  21  HL:=17FA;
0279  CD  JS032D;              HL(MM17FA)
027C  21  HL:=17FE;
027F  CD  JS032D;              SP(MM17FE)
0282  21  HL:=17FC;
0285  CD  JS032D;              PC(MM17FC)
0288  CD  JS0200;          换行
028B  E7  JSS4;            读取并打印输入字符
028C  FE  A-20;
028E  CA  JZ02B1;          若为空格,则转至 02B1
0291  FE  A-0D;
0293  CA  JZ0087;          若为回车,则结束操作
0296  CD  JS0324;
0299  CD  JS0322;
029C  CD  JS031F;
029F  CD  JS031F;          若非空格与回车,则将新值存入"N"标
02A2  82  A:+D;              志,读取并设置其它标志的新值
02A3  82  A:+D;
02A4  87  A:+A;
02A5  3C  A:+1;
02A6  CD  JS0322;
02A9  82  A:+D;
02AA  82  A:+D;
02AB  32  M17F8:=A;        F 的新值在 M17F8 中
```

```
02AE  C3   J02B5;

MONITOR
>
  G2800
02B1  DF   JSS3                    若标志值不变,则打印空格
02B2  DF   JSS3
02B3  DF   JSS3
02B4  DF   JSS3
02B5  0E   C:=20;                  打印空格以对准 A 值
02B7  DF   JSS3;
02B8  DF   JSS3;
02B9  E7   JSS4;                   读取并打印输入字符
02BA  FE   A-20;
02BC  CA   JZ02C8;                 若为空格,转至 02C8
02BF  CD   JS0221;                 若非空格,完成字节读取操作,并存入
02C2  32   M17F9:=A;                 寄存器 A 用的存贮单元(M17F9)
02C5  C3   J02C9;
02C8  DF   JSS3;                   若 A 值不变,打印空格
02C9  21   HL:=17F6;
02CC  CD   JS033C;
02CF  21   HL:=17F4;
02D2  CD   JS033C;
02D5  21   HL:=17FA;               BC、DE、HL、SP、PC 的任何改变都用
02D8  CD   JS033C;                   033C 的子程序处理
02DB  21   HL:=17FE;
02DE  CD   JS033C;
02E1  21   HL:=17FC;
02E4  CD   JS033C;
02E7  C3   J0087;                  结束
```

```
MONITOR
>
02EA  "NZHPK A BC DE HL SP PC"
  G2800
0312   87   A:+A;              "位输出(A6)"子程序入口
0313   47   B:=A;              "位输出(A7)"子程序入口
0314   3E   A:=30;
0316   F2   JNN031A;
0319   3C   A:+1;
031A   4F   C:=A;
031B   DF   JSS3;
031C   78   A:=B;
031D   87   A:+A;
031E   C9   RET;
031F   82   A:+D;              "位输入"子程序入口1
0320   82   A:+D;
0321   87   A:+A;
0322   57   D:=A;              "位输入"子程序入口2
0323   E7   JSS4;
0324   D6   A:-30;            "位输入"子程序入口3
0326   C8   RZ;
0327   FE   A-01;
0329   C8   RZ;
032A   C3   J00B4;
032D   0E   C:=20;  ┐          "寄存器输出"子程序入口
032F   DF   JSS3;   │
0330   DF   JSS3;   │
0331   23   HL:+1;  │
0332   7E   A:=M;   ├          输出两个空格，然后输出 MM(HL)的
0333   CD   JS0204; │            值(四位十六进制数字的整数)
0336   2B   HL:-1; ┘
```

```
0337   7E   A:=M;
0338   CD   JS0204;
033B   C9   RET;
033C   0E   C:=20;          "寄存器输入"子程序入口
033E   DF   JSS3;           输出两个空格
033F   DF   JSS3;
0340   E7   JSS4;           读取并打印输入字符
0341   FE   A-20;
0343   CA   JZ0356;         若为空格,则转至0356
0346   FE   A-0D;
0348   CA   JZ0087;         若为回车,则结束操作
034B   CD   JS0221;
034E   23   HL:+1;
034F   77   M:=A;           否则，完成四个十六进制数字的整数
0350   CD   JS0220;           值的读入操作，并将其值存入MM
0353   2B   HL:-1;            (HL)
0354   77   M:=A;
0355   C9   RET;
0356   DF   JSS3;
0357   DF   JSS3;
0358   DF   JSS3;
0359   C9   RET;           打印空格以对准位置
035A   79   A:=C;
035B   F6   A:U7F;
035D   C9   RET;
035E   00   NULL;
035F   00   NULL;

MONITOR
>
  G2800
```

```
0360  CD  JS03D1;     "PROM 编程"程序入口,预置
0363  DB  A:=GF4;     从 PROM 读入字节
0365  3C  A:+1;
0366  C2  JNZ03B6;    若非 FF 值，则转至出错处理例行程
                       序
0369  CD  JS03E4;     测试
036C  CA  JZ0363;     循环，直至检查完所有的PROM 单元
036F  3E  A:=88;
0371  D3  GF7:=A;     为写入操作准备接口
0373  3E  A:=01;
0375  D3  GF5:=A;     设置 PROM 的"写入"位
0377  21  HL:=1000;   RAM 基地址
037A  E5  ST:=HL;
037B  06  B:=C8;      重复次数计算器(十六进制 C8 二十进
                       制 200)
037D  3E  A:=04;
037F  CD  JS03DD;     "复位"脉冲发至编程器中的计数器去
0382  E1  HL:=ST;
0383  E5  ST:=HL;
0384  7E  A:=M;       从 RAM 读入下一个字节
0385  D3  GF4:=A;     字节发至 PROM 数据端
0387  00  NULL;
0388  00  NULL;       建立时间
0389  3E  A:=02;
038B  D3  GF6:=A;     编程脉冲的上升沿
038D  3E  A:=35;
038F  3D  A:-1;       脉宽计时
0390  C2  JNZ038F;
0393  D3  GF6:=A;     编程脉冲的下降沿
0395  CD  JS03E3;     地址加一
0398  CA  JZ0384;     循环，准备对下一字节进行操作，直至
```

```
                                    全部完成
039B  05    B:-1;
039C  C2    JNZ037D;                重复,直至完成200遍
039F  3E    A:=98;                  为读入操作准备接口
03A1  D3    GF7:=A;
03A3  E1    HL:=ST;
03A4  DB    A:=GF4;                 从PROM读入下一个字节
03A6  BE    A-M;                    与RAM中的原始值比较
03A7  C2    JNZ03BB;                若不同,转至出错处理例行程序
03AA  CD    JS03E3;                 测试地址加一
03AD  C2    JNZ03A4;                循环,直至全部检查完
03B0  76    HALT;                   操作完成
03B1  00    NULL;
03B2  00    NULL;
03B3  00    NULL;                   (未用)
03B4  00    NULL;
03B5  00    NULL;
03B6  3E    A:=02;                  出错:显示未擦净信号
03B8  D3    GF5:=A;
03BA  76    HALT;                   出错:显示"出错"信号
03BB  3E    A:=04;
03BD  D3    GF5:=A;
03BF  CF    JSS1;

MONITOR
>
  G2800
03C0  00    NULL;                   "PROM读入"程序入口
03C1  CD    JS03D1;                 预置接口
03C4  21    HL:=1000;               设置RAM基地址
03C7  DB    A:=GF4;                 从PROM读入字节
```

```
03C9  77  M:=A;          存入RAM
03CA  CD  JS03E3;        地址加一
03CD  CA  JZ03C7;        循环,直至全部完成
03D0  CF  JSS1;          调用监督程序
03D1  3E  A:=98;         "预置接口"子程序入口
03D3  D3  GF7:=A;
03D5  3E  A:=04;
03D7  CD  JS03DD;
03DA  C9  RET;
03DB  3E  A:=01;         "A0脉冲发至接口"入口
03DD  D3  GF6:=A;        "脉冲发至接口"入口
03DF  97  A:-A;
03E0  D3  GF6:=A;
03E2  C9  RET;
03E3  23  HL:+1;         "测试地址加一"入口
03E4  CD  JS03DB;        "测试"入口
03E7  DB  A:=GF6;
03E9  E6  A:&10;
03EB  C9  RET;
03EC  FF  JSS7;          "输入字符串(ESC)"子程序入口
03ED  77  M:=A;
03EE  FE  A-1B;
03F0  C8  RZ;
03F1  DF  JSS3;
03F2  23  HL:+1;
03F3  C3  J03EC;
03F6  4E  C:=M;          "输出字符串(ESC)"子程序入口
03F7  79  A:=C;
03F8  FE  A-1B;
03FA  C8  RZ;
03FB  DF  JSS3;
```

```
03FC  23   HL:+1;
03FD  C3   J03F6;

MONITOR
>
  G2800
0400  21   HL:=1180;        (读指令例行程序)
0403  22   MM1010:=HL;      预置目的程序计数器
0406  21   HL:=1000;        预置源字符地址
0409  16   D:=0E;           最大指令长度为 14
040B  CD   JS021B;          读字符(不打印)
040E  FE   A-20;
0410  CA   JZ040B;
0413  FE   A-0A;
0415  CA   JZ040B;          滤除空格或换行
0418  FE   A-7F;
041A  CA   JZ0431;          若读入"删除",则放弃该指令
041D  FE   A-3B;
041F  CA   JZ043E;
0422  FE   A-0D;
0424  CA   JZ043E;          指令以";"或回车作为结束
0427  77   M:=A;            存贮源字符
0428  23   HL:+1;           地址加一
0429  4F   C:=A;
042A  CD   JS01F4;          回送收到的字符
042D  15   D:-1;            减计数,若读入字符数 <14 则循环;否
                              则出错
042E  C2   JNZ040B;
0431  0E   C:=3F;
0433  CD   JS01F4;            打印"?"
0436  0E   C:=0D;
```

```
0438   CD   JS01F4;          换行
043B   C3   J0406;           循环,准备读下一条指令
043E   36   M:=FF;           存储十六进制 FF 作为结束符
0440   0F   C:=3B;
0442   CD   JS01F4;          输出":"
0445   7A   A:=D;
0446   C6   A:+05;
0448   57   D:=A;
0449   0E   C:=20;           输出足够的空格,以使字车对准第二十
                              个位置
044B   CD   JS01F4;
044E   15   D:-1;
044F   C2   JNZ0449;
0452   21   HL:=1000;        源字符地址复位
0455   C3   J0C20;           转至 0C20,处理指令
0458   C5   ST:=BC;          "输出目的字节"子程序入口
0459   C3   J0486;           转至 0486,继续执行
045C   00   NULL;
045D   00   NULL;
045E   00   NULL;
045F   00   NULL;

MONITOR
>
  G 2800                    (输出例行程序)
0460   CA   JZ0475;          若 Z=1,则转至 0475
0463   3E   A:=00;           准备三个全 0 字节
0465   47   B:=A;
0466   5F   E:=A;
0467   CD   JS0458;          输出目的字节
046A   78   A:=B;
```

```
046B  CD  JS0458;          输出目的字节
046E  78  A:=E;
046F  C3  J047A;           转至047A
0472  CD  JS0458;          输出目的字节
0475  78  A:=B;
0476  C3  J047A;           转至047A
0479  81  A:+C;
047A  CD  JS0458;          输出目的字节
047D  23  HL:+1;
047E  7F  A:=M;            取下一个源符号
047F  3C  A:+1;
0480  C2  JNZ0431;         若非结束符(FF),则出错
0483  C3  J0436;           重入读指令例行程序的主循环
0486  E5  ST:=HL;          "输出目的字节"子程序的继续
0487  2A  HL:=MM1010;      取目的程序计数器值
048A  77  M:=A;            存贮新的目的字节
048B  23  HL:+1;
048C  22  MM1010:=HL;      存贮加一后的目的程序计数器值
048F  E1  HL:=ST;
0490  CD  JS0204;          打印目的字节
0493  0F  C:=20;
0495  CD  JS01F4;          打印空格
0498  C1  BC:=ST;
0499  C9  RET;
049A  00  NULL;
049B  3E  A:=3C;           ("A:+1")
049D  C3  J047A;           输出一个字节(3C)

MONITOR
>
 G 2800
```

```
04A0  7E  A:=M;          (第一个字符,非"X")
04A1  FE  A-41;
04A3  C2  JNZ0580;       若非"A",则转至0580
04A6  23  HL:+1;         ("A")
04A7  7E  A:=M;          下一个字符
04A8  FE  A-3A;
04AA  CA  JZ09D1;        若为":",则转至09D1
04AD  C3  J09B8;         若非":",则至09B8继续执行

MONITOR
>
  G 2800
04B0  79  A:=C;          ("A_op")
04B1  FE  A-FF;
04B3  CA  JZ0463;        未发现操作符,出错
04B6  FE  A-08;
04B8  CA  JZ04F1;        若为"A:="转至04F1
04BB  59  E:=C;
04BC  CD  JS0B1B;        单字长寄存器或J值
04BF  47  B:=A;
04C0  79  A:=C;
04C1  FE  A-FF;
04C3  CA  JZ0463;        若无结束符,则出错
04C6  FE  A-08;
04C8  CA  JZ04E8;        若为"A_opJ"转至04E8
04CB  FE  A-09;
04CD  CA  JZ04D9;        若为"A_opI"转至04D9
04D0  7B  A:=E;          ("A_opP")
04D1  87  A:+A;          形成I
04D2  87  A:+A;
04D3  87  A:+A;
```

```
04D4  C6  A:+80;
04D6  C3  J0479;          输出是一个字节
04D9  7B  A:=E;           ("A_opi")
04DA  FE  A-00;
04DC  CA  JZ049B;         若为"A:+1"转至049B
04DF  FE  A-02;
04E1  06  B:=3D;
04E3  C3  J0460;          输出一个字节(3D),表示"A:-1"
04E6  00  NULL;
04E7  00  NULL;
04E8  7B  A:=E;           ("A_opJ")
04E9  87  A:+A;           形成I
04EA  87  A:+A;
04EB  87  A:+A;
04EC  C6  A:+C6;
04EE  C3  J0472;          输出两个字节
04F1  CD  JS0B1B;         ("A:=")单字长寄存器或J值
04F4  47  B:=A;
04F5  FE  A-47;
04F7  CA  JZ0528;         若为"A:=G"转至0528
04FA  79  A:=C;
04FB  FE  A-08;
04FD  C2  JNZ0505;        若非J值,转至0505
0500  3E  A:=3E;          ("A:=J")
0502  C3  J0472;          输出两个字节(3EJ)

MONITOR
>
  G 2800
0505  FE  A-FF;           ("A:=1")
0507  CA  JZ050F;         若无寄存器,则转至050F
```

```
050A  3E  A:=78;
050C  C3  J0479;          输出一个字节
050F  23  HL:+1;
0510  7E  A:=M;           下一个字符
0511  FE  A-FF;
0513  C2  JNZ051A;        若非结束符,转至051A
0516  2B  HL:-1;          ("A:=M")
0517  C3  J050A;          转回,输出一个字节
051A  FE  A-5B;
051C  CA  JZ0532;         若为"[",转至0532
051F  CD  US0B80;         构成JK
0522  47  B:=A;           ("A:=MKKjj")
0523  3E  A:=3A;
0525  C3  J0467;          输出三个字节(3A J K)
0528  23  HL:+1;          ("A:=G")
0529  CD  JS0B60;         构成J
052C  47  B:=A;           ("A:=Gjj")
052D  3E  A:=DB;
052F  C3  J0472;          输出两个字节(DBJ)
0532  23  HL:+1;          ("A:=M[")
0533  7E  A:=M;           下一个字符
0534  FE  A-42;
0536  CA  JZ054E;         若为"B", 转至054E
0539  FE  A-44;           ("A:=M[D"?)
053B  C2  JNZ0463;        若非"D",则出错
053E  23  HL:+1;
053F  7E  A:=M;           下一个字符
0540  FE  A-45;
0542  C2  JNZ0463;        若非"E", 则出错
0545  06  B:=1A;          I的值
0547  23  HL:+1;
```

```
0548   7E   A:=M;
0549   FE   A-5D;
054B   C3   J0460;          ("A:=M[DE]")输出一个字节(IA)
054E   23   HL:+1;          ("A:=M[B")
054F   7E   A:=M;           下一个字符
0550   FE   A-43;
0552   C2   JNZ0463;        若非"C",则出错
0555   06   B:=0A;          I的值
0557   C3   J0547;          检查"]"并输出0A,表示"A:=M[BC]"

MONITOR
>
  G2800
055A   23   HL:+1;          ("AF")
055B   7E   A:=M;           第三个字符
055C   FE   A-3A;
055E   C2   JNZ0463;        若非":",则出错
0561   23   HL:+1;
0562   7E   A:=M;           第四个字符
0563   FE   A-3D;
0565   C2   JNZ0463;        若非"=",则出错
0568   23   HL:+1;
0569   7E   A:=M;           第五个字符
056A   FE   A-53;
056C   C2   JNZ0463;        若非"S",则出错
056F   23   HL:+1;
0570   7E   A:=M;           第六个字符
0571   FE   A-54;
0573   06   B:=F1;
0575   C3   J0460;          ("AF:=ST")输出一个字节(F1)
0578   FE   A-43;           ("DE")
```

```
057A  06  B:=27;
057C  C3  J0460;          ("DEC")输出一个字节(27)
057F  00  NULL;

MONITOR
>
  G2800
0580  FE  A-42;           (第一个字符)
0582  C2  JNZ0605;        若非 B 转至 0605
0585  23  HL:+1;          ("B")
0586  7E  A:=M;           第二个字符
0587  FE  A-43;
0589  CA  JZ05CE;         若为"C", 则转至 05CE
058C  1E  E:=40;          (r 或 r1 将存入 B)
058E  FE  A-3A;
0590  C2  JNZ0463;        若非":",则出错
0593  23  HL:+1;          ("r:")
0594  7E  A:=M;           第三个字符
0595  FE  A-2B;
0597  CA  JZ05BD;         若为"+",则转至 05BD
059A  FE  A-2D;
059C  CA  JZ05C3;         若为"-",则转至 05C3
059F  FE  A-3D;
05A1  C2  JNZ0463;        若非"=",则出错
05A4  CD  JS0B1B;         ("r:=")单字长寄存器或 J
05A7  47  B:=A;
05A8  79  A:=C;
05A9  FE  A-08;
05AB  C2  JNZ05B4;        若非 J,则转至 05B4
05AE  7B  A:=E;           ("r:=jj")
05AF  C6  A:+C6;
```

```
05B1   C3   J0472;          输出两个字节
05B4   E6   A:&F8;
05B6   C2   JNZ0463;        若非 r2,则出错
05B9   7B   A:=E;           ("r1:=r2")输出一个字节
05BA   C3   J0479;
05BD   7B   A:=E;           ("r:+")
05BE   C6   A:+C4;
05C0   C3   J05C6;
05C3   7B   A:=E;           ("r:-")
05C4   C6   A:+C5;
05C6   47   B:=A;
05C7   23   HL:+1;
05C8   7E   A:=M;           下一个字符
05C9   FE   A-31;
05CB   C3   J0460;          ("r:+1"或"r:-1")输出一个字节

MONITOR
>
  G 2800
05CE   23   HL:+1;          ("BC")
05CF   7E   A:=M;           第三个字符
05D0   FE   A-3A;
05D2   C2   JNZ0463;        若非":",则出错
05D5   0E   C:=00;          基数 I=00 表示 BC
05D7   23   HL:+1;
05D8   7E   A:=M;           第四个字符
05D9   FE   A-2B;
05DB   CA   JZ0960;         若为"+",转至 0960
05DE   FE   A-2D;
05E0   CA   JZ0966;         若为"-",转至 0966
05E3   FE   A-3D;
```

```
05E5  C2  JNZ0463;          若非"=",则出错
05E8  23  HL:+1;
05E9  7E  A:=M;             第五个字符
05EA  FE  A-53;
05EC  CA  JZ05F8;           若为"S",转至05F8
05EF  0C  C:+1;             修改I,以供"打入立即数"
05F0  C3  J0764;            转至0764继续执行
05F3  79  A:=C;             (未用)
05F4  3C  A:+1;
05F5  C3  J0467;
05F8  23  HL:+1;            ("BC:=S")
05F9  7E  A:=M;             第六个字符
05FA  FE  A-54;
05FC  C2  JNZ0463;          若非"T",则出错
05FF  79  A:=C;             ("出栈"指令)
0600  C6  A:+C1;            结果C1、D1、E1或F1
0602  C3  J047A;            输出一个字节

MONITOR
>
 G 2800
0605  FE  A-43;             (第一个字符)
0607  C2  JNZ0611;          若非"C",转至0611
060A  23  HL:+1;            ("C")
060B  7E  A:=M;             第二个字符
060C  1E  E:=48;            (r或r1将存入C)
060E  C3  J058E;            转至0586继续执行
0611  FE  A-44;             (第一个字符)
0613  C2  JNZ062E;          若非"D"转至062E
0616  23  HL:+1;            ("D")
0617  7E  A:=M;             第二个字符
```

```
0618  FE  A-45;
061A  CA  JZ0622;        若为"E",则转至0622
061D  1E  E:=50;         (r或r1将存入D)
061F  C3  J058E;         转至058E继续执行
0622  23  HL:+1;         ("DE")
0623  7E  A:=M;          第三个字符
0624  FE  A-3A;
0626  C2  JNZ0578;       若非":",则转至0578
0629  0E  C:=10;         ("DE")基数I=10,表示0E
062B  C3  J05D7;         转至05D7继续执行(继续读入,用DE
                           代替BC)
062E  FE  A-47;          (第一个字符)
0630  C2  JNZ064F;       若非G转至064F
0633  23  HL:+1;         ("G")
0634  CD  JS0B60;        构成J
0637  47  B:=A;
0638  23  HL:+1;
0639  7E  A:=M;          下一个字符
063A  FE  A-3A;
063C  C2  JNZ0463;       若非":",则出错
063F  23  HL:+1;
0640  7E  A:=M;          下一个字符
0641  FE  A-3D;
0643  C2  JNZ0463;       若非"=",则出错
0646  23  HL:+1;
0647  7E  A:=M;          下一个字符
0648  FE  A-41;
064A  3E  A:=D3;
064C  CA  JZ0472;        ("Gjj:=A")输出两个字符
                           (D3J)
064F  FE  A-48;          (第一个字符)
```

```
0651   C2   JNZ0720;          若非 H, 转至 0720
0654   23   HL:+1;           ("H")
0655   7E   A:=M;            第二个字符
0656   FE   A-3A;
0658   C2   JNZ0660;         若非":"转至 0660
065B   1E   E:=60;           (r 或 r1 将存入 H)
065D   C3   J0593;           转至 0593 继续执行

MONITOR
>
  G 2800
0660   FE   A-4C;           ("H")(第二个字符)
0662   C2   JNZ0987;         若非"L", 转至 0987
0665   23   HL:+1;           ("HL")
0666   7E   A:=M;            第三个字符
0667   FE   A-3A;
0669   C2   JNZ0463;         若非":", 则出错
066C   23   HL:+1;
066D   7E   A:=M;            第四个字符
066E   FE   A-2D;
0670   CA   JZ06BC;          若为"-", 转至 06BC
0673   FE   A-2B;
0675   C2   JNZ06C5;         若非"+", 转至 06C5
0678   23   HL:+1;           ("HL:+")
0679   7E   A:=M;            第五个字符
067A   FE   A-31;
067C   C2   JNZ0684;         若非"1", 转至 0684
067F   3E   A:=23;
0681   C3   J047A;           ("HL:+1")输出一个字节(23)
0684   FE   A-42;           ("HL:+")(第五个字符)
0686   C2   JNZ0692;         若非"B"转至 0692
```

```
0689  23  HL:+1;          ("HL:+B")
068A  7E  A:=M;           第六个字符
068B  FE  A-43;
068D  06  B:=09;
068F  C3  J0460;          输出一个字节(09)表示"HL:+BC"
0692  FE  A-44;           ("HL:+")(第五个字符)
0694  C2  JNZ06A0;        若非"D",转至06A0
0697  23  HL:+1;          ("HL:+D")
0698  7E  A:=M;           第六个字符
0699  FE  A-45;
069B  06  B:=19;
069D  C3  J0460;          输出一个字节(19)表示"HL:+DE"
06A0  FE  A-48;           ("HL:+")(第五个字符)
06A2  C2  JNZ06AE;        若非"H",则转至06AE
06A5  23  HL:+1;          ("HL:+H")
06A6  7E  A:=M;           第六个字符
06A7  FE  A-4C;
06A9  06  B:=29;
06AB  C3  J0460;          输出一个字节(29)表示"HL:+HL"
06AE  FE  A-53;           ("HL:+")(第五个字符)
06B0  C2  JNZ0463;        若非"S",则出错
06B3  23  HL:+1;          ("HL:+S")
06B4  7E  A:=M;           第六个字符
06B5  FE  A-50;
06B7  06  B:=39;
06B9  C3  J0460;          输出一个字节(39)表示"HL:+SP"
06BC  23  HL:+1;          ("HL:-")
06BD  7E  A:=M;           第五个字符
06BE  FE  A-31;
06C0  06  B:=2B;
06C2  C3  J0460;          输出一个字节(2B)表示"HL:-1"
```

```
MONITOR
>
  G 2800
06C5   FE   A--3D;          ("HL:")(第四个字符)
06C7   C2   JNZ0463;        若非"=",则出错
06CA   23   HL:+1;          ("HL:=")
06CB   7E   A:=M;           第五个字符
06CC   FE   A-3A;
06CE   C2   JNZ06EF;        若非":", 转至 06EF
06D1   23   HL:+1;          ("HL:=:")
06D2   7E   A:=M;           第六个字符
06D3   FE   A-53;
06D5   C2   JNZ06E1;        若非"S",转至 06E1
06D8   23   HL:+1;          ("HL:=:S")
06D9   7E   A:=M;           第七个字符
06DA   FE   A-54;
06DC   06   B:=E3;
06DE   C3   J0460;          输出一个字节(E3)表示"HL:=:ST"
06E1   FE   A-44;           ("HL:=:")(第六个字符)
06E3   C2   JNZ0463;        若非"D",则出错
06E6   23   HL:+1;          ("HL:=:D")
06E7   7E   A:=M;           第七个字符
06E8   FE   A-45;
06EA   06   B:=EB;
06EC   C3   J0460;          输出一个字节(EB)表示"HL:=:DE"
06EF   FE   A-53;           ("HL:=")(第五个字符)
06F1   C2   JNZ0700;        若非"S",转至 0700
06F4   23   HL:+1;          ("HL:=S")
06F5   7E   A:=M;           第六个字符
06F6   FE   A-54;
06F8   06   B:=E1;
```

```
06FA  C3  J0460;        输出一个字节(E1)表示"HL:=ST"
06FD  00  NULL;         (未用)
06FE  00  NULL;
06FF  00  NULL;
0700  FE  A-4D;         ("HL:=")(第五个字符)
0702  C2  JNZ0716;      若非"M", 转至 0716
0705  23  HL:+1;        ("HL:=M")
0706  7E  A:=M;         第六个字符
0707  FE  A-4D;
0709  C2  JNZ0463;      若非"M",则出错
070C  23  HL:+1;        ("HL:=MM")
070D  CD  JS0B80;       构成 JK
0710  47  B:=A;
0711  3E  A:=2A;
0713  C3  J0467;        输出三个字节(2AJK)表示"HL:=
                          MMKKjj"
0716  CD  JS0B80;       ("HL:=")构成 JK
0719  47  B:=A;
071A  3E  A:=21;
071C  C3  J0467;        输出三个字节(21JK)表示"HL:=
                          KKjj"
071F  00  NULL;

MONITOR
>
  G 2800
0720  FE  A-49;         (第一个字符)
0722  C2  JNZ0746;      若非"I",转至 0746
0725  23  HL:+1;        ("I")
0726  7E  A:=M;         第二个字符
0727  FE  A-3A;
```

```
0729  C2  JNZ0463;        若非":",则出错
072C  23  HL:+1;
072D  7E  A:=M;          第三个字符
072E  FE  A-3D;
0730  C2  JNZ0463;        若非"=",则出错
0733  23  HL:+1;
0734  7E  A:=M;          第四个字符
0735  FE  A-30;
0737  C2  JNZ073F;       若非"0",转至073F
073A  3E  A:=F3;         ("I:=0")
073C  C3  J047A;         输出一个字节(F3)
073F  FE  A-31;          ("I:=")
0741  06  B:=FB;
0743  C3  J0460;         输出一个字节(FB)表示"I:=1"
0746  FE  A-4A;          (第一个字符)
0748  C2  JNZ07C2;       若非"J",转至07C2
074B  23  HL:+1;         ("J")
074C  7E  A:=M;          第二个字符
074D  FE  A-53;
074F  C2  JNZ078B;       若非"S",转至078B
0752  23  HL:+1;         ("JS")
0753  7E  A:=M;          第三个字符
0754  CD  JS0BC0;        条件
0757  79  A:=C;
0758  FE  A-FF;
075A  CA  JZ076C;        若无条件,则转至076C
075D  87  A:+A;          ("JScnd")
075E  87  A:+A;
075F  87  A:+A;          形成I字节
0760  C6  A:+C4;
0762  4F  C:=A;
```

```
0763  23  HL:+1;
0764  C5  ST:=BC;
0765  CD  JS0B80;          构成JK
0768  C1  BC:=ST;
0769  C3  J07DE;           转至07DE继续执行
076C  7E  A:=M;            ("JS")(第三个字符)
076D  FE  A-53;
076F  C2  JNZ0783;         若非"S",转至0783
0772  23  HL:+1;           ("JSS")
0773  7E  A:=M;            第四个字符
0774  D6  A:-30;
0776  FE  A-08;
0778  F2  JNN0463;         若不是八进制数字,则出错
077B  87  A:+A;            形成I字节
077C  87  A:+A;
077D  87  A:+A;
077E  C6  A:+C7;
0780  C3  J047A;           输出一个字节代表"JSSs"
0783  0E  C:=CD;           ("JS")(第三个字符)
0785  C3  J0764;           用"CD"作为I字节转至0764继续执行
0788  00  NULL;
0789  00  NULL;
078A  00  NULL;

MONITOR
>
  G 2800
078B  CD  JS0BC0;          ("J")条件
078E  79  A:=C;
078F  FE  A-FF;
0791  CA  JZ079C;          若无条件,转至079C
```

```
0794   87   A:+A;          ("Jend")
0795   87   A:+A;
0796   87   A:+A;
0797   C6   A:+C2;         形成I字节
0799   C3   J0762;         转至0762继续执行
079C   7E   A:=M;          ("J")(第二个字符)
079D   FE   A-5B;
079F   CA   JZ07AB;        若为"[", 转至07AB
07A2   CD   JS0B80;        ("J")构成JK
07A5   47   B:=A;
07A6   3E   A:=C3;
07A8   C3   J0467;         输出三个字节(C3JK)表示"JKKjj"
07AB   23   HL:+1;         ("J[")
07AC   7E   A:=M;          第三个字符
07AD   FE   A-48;
07AF   C2   JNZ0463;       若非"H", 则出错
07B2   23   HL:+1;
07B3   7E   A:=M;          第四个字符
07B4   FE   A-4C;
07B6   C2   JNZ0463;       若非"L", 则出错
07B9   23   HL:+1;
07BA   7E   A:=M;          第五个字符
07BB   FE   A-5D;
07BD   06   B:=E9;
07BF   C3   J0460;         输出一个字节(E9)表示"J[HL]"
07C2   FE   A-4B;          (第一个字符)
07C4   C2   JNZ0D9E;       若非"K",转至0D9E
07C7   23   HL:+1;         ("K")
07C8   7E   A:=M;          第二个字符
07C9   FE   A-3A;
07CB   C2   JNZ0463;       若非":",则出错
```

```
07CE  23  HL:+1;
07CF  7E  A:=M;          第三个字符
07D0  FE  A-3D;
07D2  C2  JNZ0980;       若非"=", 则转至 0980
07D5  23  HL:+1;         ("K:=")
07D6  7E  A:=M;
07D7  FE  A-31;
07D9  06  B:=37;
07DB  C3  J0460;         输出一个字节(37)表示"K:=1"
07DE  47  B:=A;          (由 0769 转至此)
07DF  79  A:=C;
07E0  C3  J0467;         输出三个字节
07E3  00  NULL;
07E4  00  NULL;
07E5  00  NULL;

MONITOR
>
 G 2800
0800  3E  A:=70;         (未用)
0802  C3  J0593;
0805  FE  A-5B;          ("M")(第二个字符)
0807  C2  JNZ0847;       若非"[", 转至 0847
080A  23  HL:+1;         ("M[")
080B  7E  A:=M;          第三个字符
080C  FE  A-42;
080E  C2  JNZ081D;       若非"B", 转至 081D
0811  23  HL:+1;         ("M[B")
0812  7E  A:=M;          第四个字符
0813  FE  A-43;
0815  C2  JNZ0463;       若非"C", 则出错
```

```
6813  06  B:=02;          置I=02
081A  C3  J082B;          转至082B继续执行
081D  FE  A-44;           ("M[")(第三个字符)
081F  C2  JNZ0463;        若非"D",则出错
0822  23  HL:+1;          ("M[D")
0823  7E  A:=M;           第四个字符
0824  FE  A-45;
0826  C2  JNZ0463;        若非"E",则出错
0829  06  B:=12;          置I=12
082B  23  HL:+1;
082C  7E  A:=M;           第五个字符
082D  FE  A-5D;
082F  C2  JNZ0463;        若非"]",则出错
0832  23  HL:+1;
0833  7E  A:=M;           第六个字符
0834  FE  A-3A;
0836  C2  JNZ0463;        若非":",则出错
0839  23  HL:+1;
083A  7E  A:=M;           第七个字符
083B  FE  A-3D;
083D  C2  JNZ0463;        若非"=",则出错
0840  23  HL:+1;
0841  7E  A:=M;           第八个字符
0842  FE  A-41;
0844  C3  J0460;          若为A,输出一个字节(02或12)

MONITOR
>
  G 2800
0847  FE  A-4D;           ("M")(第二个字符)
0849  C2  JNZ0872;        若非"M",转至0872
```

```
084C  23  HL:+1;          ("MM")
084D  CD  JS0B80;         构成JK
0850  47  B:=A;
0851  23  HL:+1;
0852  7E  A:=M;           下一个字符
0853  FE  A-3A;
0855  C2  JNZ0463;        若非":",则出错
0858  23  HL:+1;
0859  7E  A:=M;           下一个字符
085A  FE  A-3D;
085C  C2  JNZ0463;        若非"=",则出错
085F  23  HL:+1;
0860  7E  A:=M;           下一个字符
0861  FE  A-48;
0863  C2  JNZ0463;        若非"H",则出错
0866  23  HL:+1;
0867  7E  A:=M;           下一个字符
0868  FE  A-4C;
086A  C2  JNZ0463;        若非"L",则出错
086D  3E  A:=22;
086F  C3  J0467;          输出三个字节(22JK)表示"MMKKjj:
                            =HL"
0872  CD  JS0B80;         ("M")构成JK
0875  47  B:=A;
0876  23  HL:+1;
0877  7E  A:=M;           下一个字符
0878  FE  A-3A;
087A  C2  JNZ0463;        若非":",则出错
087D  23  HL:+1;
087E  7E  A:=M;           下一个字符
087F  FE  A-3D;
```

```
0881   C2   JNZ0463;        若非"=",则出错
0884   23   HL:+1;
0885   7E   A:=M;          下一个字符
0886   FE   A-41;
0888   C2   JNZ0463;        若非"A",则出错
088B   3E   A:=32;
088D   C3   J0467;          输出三个字节(32JK)表示"MKKjj:
                              =A"

MONITOR
>
  G 2800
0890   FE   A-52;          (第一个字符)
0892   C2   JNZ08C2;       若非"R", 转至 08C2
0895   23   HL:+1;         ("R")
0896   7E   A:=M;          第二个字符
0897   FE   A-FF;
0899   C2   JNZ08A2;       若非结束符,转至 08A2
089C   3E   A:=C9;         ("R")
089E   2B   HL:-1;
089F   C3   J047A;          输出一个字节(C9)表示"R"
08A2   CD   JS0BC0;         ("R")条件
08A5   79   A:=C;
08A6   FE   A-FF;
08A8   CA   JZ08B3;         若无条件,转至 08B3
08AB   87   A:+A;           ("Rend")
08AC   87   A:+A;
08AD   87   A:+A;
08AE   C6   A:+C0;          形成 I 字节
08B0   C3   J047A;          输出一个字节
08B3   7E   A:=M;           ("R")(第二个字符)
```

```
08B4   FE   A-45;
08B6   C2   JNZ0463;          若非"E",则出错
08B9   23   HL:+1;
08BA   7E   A:=M;             第三个字符
08BB   FE   A-54;
08BD   06   B:=C9;
08BF   C3   J0460;            输出一个字节(C9)表示"RET"

MONITOR
>
 G 2800
08C2   FE   A-53;             (第一个字符)
08C4   C2   JNZ0971;          若非"S",转至 0971
08C7   23   HL:+1;            ("S")
08C8   7E   A:=M;             第二个字符
08C9   FE   A-50;
08CB   C2   JNZ090D;          若非"P", 转至 090D
08CE   23   HL:+1;            ("SP")
08CF   7E   A:=M;             第三个字符
08D0   FE   A-3A;
08D2   C2   JNZ0463;          若非":",则出错
08D5   23   HL:+1;
08D6   7E   A:=M;             第四个字符
08D7   FE   A-3D;
08D9   C2   JNZ08F5;          若非"=",转至 08F5
08DC   23   HL:+1;            ("SP:=")
08DD   7E   A:=M;             第五个字符
08DE   FE   A-48;
08E0   C2   JNZ08EC;          若非"H",转至 08EC
08E3   23   HL:+1;            ("SP:=H")
08E4   7E   A:=M;             第六个字符
```

```
08E5  FE  A-4C;
08E7  06  B:=F9;
08E9  C3  J0460;          输出一个字节(F9)表示"SP:=HL"
08EC  CD  JS0B80;         ("SP:=")构成 JK
08EF  47  B:=A;
08F0  3E  A:=31;
08F2  C3  J0467;          输出三个字节(31JK)表示"SP:=
                            KKjj"
08F5  FE  A-2B;           ("SP:")(第四个字符)
08F7  C2  JNZ08FF;        若非"+",转至08FF
08FA  06  B:=33;          ("SP:+")
08FC  C3  J0906;          I=33,转至0906继续执行
08FF  FE  A-2D;           ("SP:")(第四个字符)
0901  C2  JNZ0463;        若非"-",则出错
0904  06  B:=3B;          置I=3B
0906  23  HL:+1;
0907  7E  A:=M;           第五个字符
0908  FE  A-31;
090A  C3  J0460;          输出一个字节(33或3B)表示"SP:±1"

MONITOR
>
  G 2800
090D  FE  A-54;           ("S")(第二个字符)
090F  C2  JNZ0463;        若非"T",则出错
0912  23  HL:+1;          ("ST")
0913  7E  A:=M;           第三个字符
0914  FE  A-3A;
0916  C2  JNZ0463;        若非":",则出错
0919  23  HL:+1;
091A  7E  A:=M;           第四个字符
```

```
091B  FE  A-3D;
091D  C2  JNZ0463;          若非"=",则出错
0920  23  HL:+1;
0921  7E  A:=M;            第五个字符
0922  FE  A-42;
0924  C2  JNZ0930;          若非"B",转至 0930
0927  23  HL:+1;            ("ST:=B")
0928  7E  A:=M;            第六个字符
0929  FE  A-43;
092B  06  B:=C5;
092D  C3  J0460;            输出一个字节(C5)表示"ST:=BC"
0930  FE  A-44;            ("ST:=")(第五个字符)
0932  C2  JNZ093E;          若非"D", 转至 093E
0935  23  HL:+1;            ("ST:=D")
0936  7E  A:=M;            第六个字符
0937  FE  A-45;
0939  06  B:=D5;
093B  C3  J0460;            输出一个字节(D5)表示"ST:=DE"
093E  FE  A-48;            ("ST:=")(第五个字符)
0940  C2  JNZ094C;          若非"H",转至 094C
0943  23  HL:+1;            ("ST:=H")
0944  7E  A:=M;            第六个字符
0945  FE  A-4C;
0947  06  B:=E5;
0949  C3  J0460;            输出一个字节(E5)表示"ST:=HL"
094C  FE  A-41;            ("ST:=")(第五个字符)
094E  C2  JNZ0463;          若非"A",则出错
0951  23  HL:+1;            ("ST=A")
0952  7E  A:=M;            第六个字符
0953  FE  A-46;
0955  06  B:=F5;
```

```
0957   C3   J0460;          输出一个字节(F5)表示"ST:=AF"

MONITOR
>
  G 2800
095A   D1   DE:=ST;         (由 0A82 转至此)
095B   C3   J0ABB;          转至 0ABB 继续执行
095E   00   NULL;
095F   00   NULL;
0960   79   A:=C;           ("BC:+")(或其它寄存器对)
0961   C6   A:+03;          修改 I(增量)
0963   C3   J0969;
0966   79   A:=C;           ("BC:-")(或其它寄存器对)
0967   C6   A:+0B;          修改 I(减量)
0969   47   B:=A;
096A   23   HL:+1;
096B   7E   A:=M;           第五个字符
096C   FE   A-31;
096E   C3   J0460;          若为 1, 则输出一个字节

MONITOR
>
  G 2800
0971   FE   A-45;          (第一个字符
0973   C2   JNZ099C;       若非"E", 转至 099C
0976   23   HL:+1;         ("E")
0977   7E   A:=M;          第二个字符
0978   1E   E:=58;         (r 或 r1 将存入 E)
097A   C3   J058E;          转至 058E 继续执行
097D   00   NULL;
```

```
097E  00  NULL;
097F  00  NULL;
0980  FE  A-23;          ("K:")
0982  06  B:=3F;
0984  C3  J0460;         输出一个字节(3F)表示"K:#"
0987  FE  A-41;          ("H")(第二个字符)
0989  C2  JNZ0463;       若非"A",则出错
098C  23   HL:+1;        ("HA")
098D  7E  A:=M;          第三个字符
098E  FE  A-4C;
0990  C2  JNZ0463;       若非"L",则出错
0993  23  HL:+1;         ("HAL")
0994  7E  A:=M;          第四个字符
0995  FE  A-54;
0997  06  B:=76;
0999  C3  J0460;         输出一个字节(76)表示"HALT"
099C  FE  A-4E;          (第一个字符)
099E  C2  JNZ0CF8;       若非"N",转至 0CF8
09A1  23  HL:+1;         ("N")
09A2  7E  A:=M;          第二个字符
09A3  FE  A-55;
09A5  C2  JNZ0463;       若非"U",则出错
09A8  23  HL:+1;         ("NU")
09A9  7E  A:=M;          第三个字符
09AA  FE  A-4C;
09AC  C2  JNZ0463;       若非"L",则出错
09AF  23  HL:+1;         ("NUL")
09B0  7E  A:=M;          第四个字符
09B1  FE  A-4C;
09B3  06  B:=00;
09B5  C3  J0460;         输出一个字节(00)表示"NULL"
```

```
MONITOR
>
  G 2800
09B8   FE   A-46;          (第二个字符)
09BA   CA   JZ055A;        若为"F", 转至 055A
09BD   FE   A-4B;
09BF   C2   JNZ09E5;       若非"K",转至 09E5
09C2   23   HL:+1;         ("AK")
09C3   7E   A:=M;          下一个字符
09C4   FE   A-3A;
09C6   C2   JNZ0463;       若非":",则出错
09C9   23   HL:+1;         ("AK:")
09CA   7E   A:=M;          下一个字符
09CB   01   BC:=171F;      目的字节可能是 17, 1F
09CE   C3   J09EE;         转至 09EE 继续执行
09D1   23   HL:+1;         ("A:")
09D2   7E   A:=M;          下一个字符
09D3   FE   A-23;
09D5   C2   JNZ09EB;       若非"#",转至 09EB
09D8   23   HL:+1;         ("A:#")
09D9   7E   A:=M;          下一个字符
09DA   2B   HL:-1;
09DB   3C   A:+1;
09DC   C2   JNZ09E4;       若非结束符,转至 09E4
09DF   3E   A:=2F;         ("A:#")
09E1   C3   J047A;         输出一个字节(2F)
09E4   2B   HL:-1;         ("A:#")
09E5   CD   JS0B20;        操作符
09E8   C3   J04B0;         转至04B0继续执行
09EB   01   BC:=070F;      ("A:")目的字节可能是 07、0F
09EE   FE   A-40;          第三个字符
```

```
09F0  C2  JNZ09E4;        若非"@",转至 09E4
09F3  23  HL:+1;
09F4  7E  A:=M;           下一个字符
09F5  FE  A-4C;
09F7  CA  JZ0475;         ("A:@L")输出一个字节(07)
09FA  FE  A-52;
09FC  41  B:=C;
09FD  C3  J0460;          ("A:@R")输出一个字节(0F)

MONITOR
>
  G 2800
0A00  CD  JS0B60;         "形成 JK"子程序入口
0A03  5F  E:=A;           K 的值在 E 中
0A04  23  HL:+1;
0A05  CD  JS0B60;         构成 J
0A08  C9  RET;            J 的值在 A 中

MONITOR
>
  G 2800
0A09  FE  A-00;           (由 0B86 转至此)
0A0B  FA  JN0A12;         若非十六进制数字,转至 0A12
0A0E  CD  JS0A00;         形成 JK
0A11  C9  RET;            J 在 A 中,K 在 E 中,返回
0A12  79  A:=C;           (非十六进制数字)
0A13  FE  A-28;
0A15  C2  JNZ0463;        若非"(",则出错
0A18  23  HL:+1;          ("(")
0A19  7E  A:=M;           下一个字符
0A1A  FE  A-4C;
```

```
0A1C  C2  JNZ0A56;          若非"L",转至0A56
0A1F  23  HL:+1;            ("(L")
0A20  CD  JS0B60;           构成J
0A23  E5  ST:=HL;
0A24  26  H:=00;            从标号表中找到相应的元素
0A26  6F  L:=A;
0A27  4F  C:=A;
0A28  29  HL:+HL;
0A29  EB  HL:=:DE;
0A2A  21  HL:=1040;         (标号表基地址)
0A2D  19  HL:+DE;
0A2E  5E  E:=M;
0A2F  23  HL:+1;
0A30  56  D:=M;             该元素的地址在DE中
0A31  7B  A:=E;
0A32  FE  A-00;
0A34  C2  JNZ0A3D;
0A37  7A  A:=D;
0A38  FE  A-00;
0A3A  CA  JZ0A40;           若元素为零,转至0A40
0A3D  C3  J0AB3;            (找到元素)转至0AB3继续执行
0A40  2A  HL:=MM1010;       (提前引用)
0A43  23  HL:+1;
0A44  EB  HL:=:DE;          DE中是目的程序中存有当前这个转
                              移地址的单元地址
0A45  2A  HL:=MM1012;       转移表计数器
0A48  71  M:=C;             将元素存入转移表(见注)
0A49  23  HL:+1;
0A4A  73  M:=E;
0A4B  23  HL:+1;
0A4C  72  M:=D;
```

```
0A4D  23   HL:+1;
0A4E  22   MM1012:=HL;     增加后的转移表计数器
0A51  16   D:=FD;         用"FD"表示提前引用
0A53  C3   J0AB3;          转至0AB3 继续执行

MONITOR
>
  G 2800
0A56  FE   A-58;           ("(")
0A58  C2   JNZ0AC0;        若非"X",转至0AC0
0A5B  23   HL:+1;          ("(X")
0A5C  7E   A:=M;           下一个字符
0A5D  4F   C:=A;
0A5E  CD   JS0B00;         ASCII-hex 转换
0A61  78   A:=B;
0A62  FE   A-00;
0A64  FA   JN0463;         若非十六进制数字,则出错
0A67  E5   ST:=HL;
0A68  CA   JZ0A85;         若为"0",转至0A85
0A6B  21   HL:=101E;       X表的基地址
0A6E  5F   E:=A;
0A6F  16   D:=00;
0A71  19   HL:+DE;
0A72  19   HL:+DE;         X表入口的地址
0A73  4E   C:=M;
0A74  23   HL:+1;
0A75  46   B:=M;           X字组的起始地址
0A76  60   H:=B;
0A77  69   L:=C;
0A78  EB   HL:=:DE;
0A79  E1   HL:=ST;
```

```
0A7A  D5  ST:=DE;
0A7B  23  HL:+1;
0A7C  7E  A:=M;          下一个字符
0A7D  FE  A-29;
0A7F  C2  JNZ0A8B;       若非")",转至0A8B
0A82  C3  J095A;         ("(Xh)")转至095A 继续执行
0A85  21  HL:=1000;      ("(X0")X0 的起始地址
0A88  C3  J0A78;         转至0A78 继续执行
0A8B  FE  A-2B;          ("(Xh")(下一个字符)
0A8D  C2  JNZ0A95;       若非"+",转至0A95
0A90  0E  C:=00;         ("(Xh+")
0A92  C3  J0A9C;         C=00,转至0A9C继续执行

MONITOR
>
 G 2800
0A95  FE  A-2D;          ("(Xh")(下一个字符)
0A97  C2  JNZ0463;       若非"-",则出错
0A9A  0E  C:=01;         C=01(符号标记)
0A9C  23  HL:+1;
0A9D  CD  JS0B60;        构成J(位移量)
0AA0  E3  HL:=:ST;       X 字组的起始地址在 HL 中
0AA1  5F  E:=A;
0AA2  16  D:=00;         位移量在 DE 中
0AA4  79  A:=C;          位移的符号
0AA5  FE  A-00;
0AA7  CA  JZ0AB1;        若位移量为正,跳至0AB1
0AAA  7A  A:=D;          DE:-
0AAB  2F  A:#;
0AAC  57  D:=A;
0AAD  7B  A:=E;
```

```
0AAE  2F   A:#;
0AAF  5F   E:=A;
0AB0  13   DE:+1;
0AB1  19   HL:+DE;          位移量加至起始地址
0AB2  EB   HL:=:DE;         计算出来的JK值
0AB3  E1   HL:=ST;
0AB4  23   HL:+1;
0AB5  7E   A:=M;            下一个字符
0AB6  FE   A-29;
0AB8  C2   JNZ0463;         若非")",则出错
0ABB  7B   A:=E;
0ABC  5A   E:=D;
0ABD  C9   RET;             J在A中,K在E中,返回
0ABE  00   NULL;
0ABF  00   NULL;

MONITOR
>
  G 2800
0AC0  FE   A-49;            ("(")
0AC2  C2   JNZ0463;         若非"I",则出错
0AC5  23   HL:+1;           ("(I")
0AC6  7E   A:=M;            下一个字符
0AC7  FE   A-2B;
0AC9  C2   JNZ0AD1;         若非"+",转至0AD1
0ACC  0E   C:=00;           置C=00作为符号标记
0ACE  C3   J0AD8;           转至0AD8继续执行
0AD1  FE   A-2D;
0AD3  C2   JNZ0463;         若非"-",则出错
0AD6  0E   C:=01;           置C=01作为符号标记
0AD8  23   HL:+1;
```

```
0AD9  CD  JS0B60;          构成 J
0ADC  E5  ST:=HL;
0ADD  5F  E:=A;
0ADE  16  D:=00;           位置量在 DE 中
0AE0  79  A:=C;
0AE1  FE  A-00;
0AE3  CA  JZ0AED;          若位移量为正,跳至 0AED
0AE6  7A  A:=D;            "DE:-"
0AE7  2F  A:#;
0AE8  57  D:=A;
0AE9  7B  A:=E;
0AEA  2F  A:#;
0AEB  5F  E:=A;
0AEC  13  DE:+1;
0AED  2A  HL:=MM103E;      目的程序第一条指令运行时的单元地
                           址
0AF0  19  HL:+DE;
0AF1  EB  HL:=:DE;
0AF2  2A  HL:=MM1010;      目的程序计数器
0AF5  19  HL:+DE;
0AF6  23  HL:+1;           加三(当前指令的长度)
0AF7  23  HL:+1;
0AF8  23  HL:+1;
0AF9  11  DE:=EE80;        减 1180 作为编译时的基地址
0AFC  19  HL:+DE;
0AFD  C3  J0AB2;           转至 0AB2 继续执行

MONITOR
>
  G 2800
0B00  D6  A:-30;           汇编程序用"ASCII-hex"转换子程序
```

的入口

```
0B02   FA   JN0B14;
0B05   FE   A-0A;
0B07   FA   JN0B19;
0B0A   D6   A:-11;
0B0C   FA   JN0B14;
0B0F   FE   A-06;
0B11   FA   JN0B17;
0B14   06   B:=FF;
0B16   C9   RET;          无十六进制数字,将FF存入B
0B17   C6   A:+0A;
0B19   47   B:=A;
0B1A   C9   RET;          十六进制数字,其值在A与B中
0B1B   23   HL:+1;        "单字长寄存器或J"子程序入口
0B1C   23   HL:+1;
0B1D   C3   J0B89;        转至0B89继续执行

MONITOR
>
  G 2800
0B20   0E   C:=00;        "操作符"子程序入口
0B22   7E   A:=M;         下一个字符
0B23   FE   A-3A;
0B25   CA   JZ0B2F;       若为":",转至0B2F
0B28   FE   A-2D;
0B2A   CA   JZ0B52;       若为"-",转至0B52
0B2D   0D   C:-1;
0B2E   C9   RET;          出错,C=FF并返回
0B2F   23   HL:+1;        (":")
0B30   7E   A:=M;         下一个字符
0B31   FE   A-2B;
```

```
0B33  CA  JZ0B55;        若为"+",转至 0B55
0B36  0C  C:+1;          (":")
0B37  0C  C:+1;
0B38  FE  A-2D;
0B3A  CA  JZ0B55;        若为"-",转至 0B55
0B3D  0C  C:+1;
0B3E  0C  C:+1;
0B3F  FE  A-26;
0B41  CA  RZ;            C=04(表示":&")并返回
0B42  0C  C:+1;
0B43  FE  A-23;
0B45  C8  RZ;            C=05(表示":#")并返回
0B46  0C  C:+1;
0B47  FE  A-55;
0B49  C8  RZ;            C=06(表示":U")并返回
0B4A  0C  C:+1;
0B4B  0C  C:+1;
0B4C  FE  A-3D;
0B4E  C8  RZ;            C=08(表示":=")并返回
0B4F  0E  C:=FF;
0B51  C9  RET;           出错,C=FF 并返回
0B52  0E  C:=07;         ("-")
0B54  C9  RET;           C=07(表示"-")并返回
0B55  47  B:=A;          (":+")或(":-")
0B56  23  HL:+1;
0B57  7E  A:=M;          下一个字符
0B58  B8  A-B:
0B59  C2  JNZ0B5E;       若非"+"或"-",转至 0B5E
0B5C  0C  C:+1;          (":++")或(":--")
0B5D  C9  RET;           若为":++", C=01 并返回;
                           若为":--",C=03 并返回
```

```
0B5E  2B   HL:-1;          (":+")或(":-")
0B5F  C9   RET;            若为":+", C=00 并返回;
                           若为":-", C=02 并返回

MONITOR
>
  G 2800
0B60  7E   A:=M;           "构成 J" 子程序入口
0B61  CD   JS0B00;         ASCII-hex 转换
0B64  78   A:=B;
0B65  3C   A:+1;
0B66  CA   JZ0B7B;         若无十六进制数字,转至0B7B
0B69  50   D:=B;
0B6A  23   HL:+1;
0B6B  7E   A:=M;           下一个字符
0B6C  CD   JS0B00;         ASCII-hex 转换
0B6F  78   A:=B;
0B70  3C   A:+1;
0B71  CA   JZ0B7B;         若无十六进制数字,转至0B7B
0B74  7A   A:=D;           高位数字左移
0B75  87   A:+A;
0B76  87   A:+A;
0B77  87   A:+A;
0B78  87   A:+A;
0B79  80   A:+B;           加入低位数字
0B7A  C9   RET;            构成 J 值后返回
0B7B  0E   C:=FF;          (非十六进制数字)
0B7D  7E   A:=M;
0B7E  C9   RET;            字符存在 A 中且 C=FF,返回
0B7F  00   NULL;
0B80  7E   A:=M;           "构成 JK"子程序入口
```

```
0B81   4F   C:=A;
0B82   CD   JS0B00;          ASCII-hex 转换
0B85   78   A:=B;
0B86   C3   J0A09;           转至 0A09 继续执行

MONITOR
>
  G 2800
0B89   7E   A:=M;            (由 0B1D 转至此)
0B8A   FE   A-FF;
0B8C   2B   HL:-1;
0B8D   C2   JNZ0BB6;         若留下的字符超过一个,转至0BB6
0B90   7E   A:=M;
0B91   0E   C:=00;
0B93   D6   A:-42;
0B95   C8   RZ;              C=00(表示"B"),返回
0B96   0C   C:+1;
0B97   3D   A:-1;
0B98   C8   RZ;              C=01(表示"C"),返回
0B99   0C   C:+1;
0B9A   3D   A:-1;
0B9B   C8   RZ;              C=02(表示"D"),返回
0B9C   0C   C:+1;
0B9D   3D   A:-1;
0B9E   C8   RZ;              C=03(表示"E"),返回
0B9F   0C   C:+1;
0BA0   D6   A:-03;
0BA2   C8   RZ;              C=04(表示"H"),返回
0BA3   0C   C:+1;
0BA4   D6   A:-04;
0BA6   C8   RZ;              C=05(表示"L"),返回
```

```
0BA7  0C   C:+1;
0BA8  3D   A:-1;
0BA9  C8   RZ;              C=06(表示"M"),返回
0BAA  0C   C:+1;
0BAB  C6   A:+0C;
0BAD  C8   RZ;              C=07(表示"A"),返回
0BAE  0C   C:+1;
0BAF  0C   C:+1;
0BB0  C6   A:+10;
0BB2  C8   RZ;              C=09(表示"I"),返回
0BB3  0E   C:=FF;
0BB5  C9   RET;             出错,C=FF 并返回
0BB6  0E   C:=08;
0BB8  CD   JS0B60;          构成 J 字节
0BBB  C9   RET;             C=08, J 字节在 A 中,返回
0BBC  00   NULL;
0BBD  00   NULL;
0BBE  00   NULL;
0BBF  00   NULL;

MONITOR
>
  G 2800
0BC0  FE   A-4E;            "条件"子程序入口
0BC2  C2   JNZ0D85;         若非"N",转至 0D85
0BC5  0E   C:=07;
0BC7  23   HL:+1;
0BC8  7E   A:=M;            下一个字符
0BC9  FE   A-47;
0BCB  F2   JNN0BD0;         若为"F"之后的字母,转至 0BD0
0BCE  2B   HL:-1;
```

```
0BCF  C9   RET;              C=07(表示"N"),返回
0BD0  0D   C:-1;
0BD1  FE   A-4E;
0BD3  C8   RZ;               C=06(表示"NN"),返回
0BD4  0D   C:-1;
0BD5  0D   C:-1;
0BD6  FE   A-50;
0BD8  C8   RZ;               C=04(表示"NP"),返回
0BD9  0D   C:-1;
0BDA  0D   C:-1;
0BDB  FE   A-4B;
0BDD  C3   J0D97;            转至0D97继续执行

MONITOR
>
  G 2800
0BE0  26   H:=30;            "HL:=sqrt(BC)×2⁷"子程序入口
0BE2  EB   HL:=:DE;
0BE3  3E   A:=0E;
0BE5  C5   ST:=BC;
0BE6  D5   ST:=DE;
0BE7  CD   JS0F93;           HL:=(BC/DE)×2^
0BEA  D1   DE:=ST;
0BEB  C1   BC:=ST;
0BEC  19   HL:+DE;
0BED  CD   JS0FF1;           HL:/2(无符号,四舍五入)
0BF0  CD   JS0C1A;           HL-DE
0BF3  C8   RZ;
0BF4  E5   ST:=HL;
0BF5  CD   JS0CF1;           HL:-DE
0BF8  23   HL:+1;
```

```
0BF9  7C   A:=H;
0BFA  B5   A:UL;
0BFB  E1   HL:=ST;
0BFC  C2   JNZ0BE2;
0BFF  C9   RET;

MONITOR
>
  G 2800
0C00  06   B:=00;          "汇编程序"入口
0C02  21   HL:=1020;
0C05  70   M:=B;
0C06  23   HL:+1;          清除 X 表与标号表
0C07  7D   A:=L;
0C08  FE   A-C0;
0C0A  C2   JNZ0C05;
0C0D  78   A:=B;
0C0E  32   M1014:=A;       预置标记与转移表计数器
0C11  21   HL:=10C0;
0C14  22   MM1012:=HL;
0C17  C3   J0400;          转至指令读入程序
0C1A  C5   ST:=BC;         "HL-DE(对BC是透明的)"子程序入
                             口
0C1B  CD   JS0231;
0C1E  C1   BC:=ST;
0C1F  C9   RET;

MONITOR
>
  G 2800
0C20  3A   A:=M1014;       .(处理指令)
```

```
0C23  FE  A－00;
0C25  C2  JNZ0C8A;          若标记已设置,则转至 0C8A
0C28  7E  A:＝M;            源指令的第一个字符
0C29  FE  A－58;
0C2B  C2  JNZ0C59;          若非“X”,转至 0C59
0C2E  23  HL:＋1;           (“X”)
0C2F  7E  A:＝M;            下一个字符
0C30  CD  JS0B00;           “ASCII-hex”转换
0C33  78  A:＝B;
0C34  FE  A－00;
0C36  FA  JN0463;           若非十六进制数,则出错
0C39  E5  ST:＝HL;
0C3A  26  H:＝00;
0C3C  6F  L:＝A;
0C3D  29  HL:＋HL;
0C3E  EB  HL:＝:DE;
0C3F  21  HL:＝1020;
0C42  19  HL:＋DE;          X 表元素的地址
0C43  E3  HL:＝:ST;
0C44  23  HL:＋1;
0C45  7E  A:＝M;            下一个字符
0C46  FE  A－3A;
0C48  C2  JNZ0463;          若非“:”,则出错
0C4B  23  HL:＋1;
0C4C  CD  JS0B60;           构成 J
0C4F  E3  HL:＝:ST;
0C50  77  M:＝A;
0C51  23  HL:＋1;
0C52  3E  A:＝00;           } 将 J 值存入 X 表(高字节为零)
0C54  77  M:＝A;
0C55  E1  HL:＝ST;
```

```
0C56   C3   J047D;              转至输出例行程序(无输出字节)

MONITOR
>
  G 2800
0C59   3E   A:=01;               (非"X")
0C5B   32   M1014:=A;            设置标记(没有其它的 X 字组了)
0C5E   E5   ST:=HL;
0C5F   2A   HL:=MM1020;          X0 字组的长度
0C62   EB   HL:=:DE;
0C63   21   HL:=1000;            目的程序(X0)用的 RAM 基地址
0C66   19   HL:+DE;
0C67   22   MM1020:=HL;          X1 的地址
0C6A   21   HL:=1020;            X 表的地址
0C6D   4E   C:=M;
0C6E   23   HL:+1;               取 X 表的元素(X 字组的地址)
0C6F   46   B:=M;
0C70   23   HL:+1;
0C71   5E   E:=M;
0C72   23   HL:+1;               取 X 表的元素(X 字组的长度)
0C73   56   D:=M;
0C74   EB   HL:=:DE;
0C75   09   HL:+BC;
0C76   EB   HL:=:DE;
0C77   72   M:=D;
0C78   2B   HL:-1;               存 X 表的元素(下一个 X 字组的地址)
0C79   73   M:=E;
0C7A   7C   A:=H;
0C7B   FE   A-10;
0C7D   C2   JNZ0C6D;
0C80   7D   A:=L;
```

```
0C81  FE  A-3E;
0C83  C2  JNZ0C6D;          重复执行,直至处理完十六个元素
0C86  E1  HL:=ST;
0C87  C3  J04A0;            转至 04A0 继续执行
0C8A  7E  A:=M;             (标记已置)第一个字符
0C8B  FE  A-58;
0C8D  CA  JZ0463;           若为"X",则出错
0C90  C3  J04A1;            转至 04A1 继续执行

MONITOR
>
 G 2800
0C93  CD  JS0B60;           (标号)构成 J
0C96  FE  A-40;
0C98  F2  JNN0463;          若超出范围,则出错
0C9B  E5  ST:=HL;
0C9C  2A  HL:=MM103E;       目的程序第一条指令运行时的单元地
                            址
0C9F  EB  HL:=:DE;
0CA0  2A  HL:=MM1010;       目的程序计数器
0CA3  19  HL:+DE;
0CA4  EB  HL:=:DE;
0CA5  21  HL:=EE80;         减 1180 作为编译时的基地址
0CA8  19  HL:+DE;
0CA9  EB  HL:=:DE;          标号运行时的单元地址在 DE 中
0CAA  06  B:=00;
0CAC  4F  C:=A;             标号
0CAD  21  HL:=1040;         标号表的基地址
0CB0  09  HL:+BC;
0CB1  09  HL:+BC;
0CB2  73  M:=E;
```

```
0CB3  23   HL:+1;
0CB4  72   M:=D;                标号存入标号表
0CB5  E1   HL:=ST;
0CB6  23   HL:+1;
0CB7  7E   A:=M;                下一个字符
0CB8  FE   A-3A;
0CBA  C2   JNZ0463;             若非":",则出错
0CBD  23   HL:+1;
0CBE  C3   J04A0;               转至04A0继续执行
0CC1  21   HL:=10C0;            ("F")转移表的基地址
0CC4  3A   A:=M1012;            转移表计数器(低半)
0CC7  BD   A-L;
0CC8  C2   JNZ0CD2;
0CCB  3A   A:=M1013;            转移表计数器(高半)
0CCE  BC   A-H;
0CCF  CA   JZ0CEE;              若全部完成,转至0CEE
0CD2  7E   A:=M;
0CD3  E5   ST:=HL;
0CD4  26   H:=00;
0CD6  6F   L:=A;
0CD7  29   HL:+HL;
0CD8  EB   HL:=:DE;
0CD9  21   HL:=1040;            标号表的基地址
0CDC  19   HL:+DE;
0CDD  5E   E:=M;
0CDE  23   HL:+1;
0CDF  56   D:=M;                标号的绝对地址在DE中
0CE0  E1   HL:=ST;
0CE1  23   HL:+1;
0CE2  4E   C:=M;
0CE3  23   HL:+1;
```

```
0CE4   46   B:=M;          要填入J字节的单元地址在BC中
0CE5   7B   A:=E;
0CE6   02   M[BC]:=A;      填入J字节
0CE7   03   BC:+1;
0CE8   7A   A:=D;
0CE9   02   M[BC]:=A;      填入K字节
0CEA   23   HL:+1;
0CEB   C3   J0CC4;         循环至0CC4
0CEE   CF   JSS1;          汇编完成,调用监督程序
0CEF   00   NULL;
0CF0   00   NULL;

MONITOR
>
  G 2800
0CF1   7D   A:=L;          "HL:-DE"子程序入口
0CF2   93   A:-E;
0CF3   6F   L:=A;
0CF4   7C   A:=H;
0CF5   9A   A:--D;
0CF6   67   H:=A;
0CF7   C9   RET;

MONITOR
>
  G 2800
0CF8   FE   A-46;          (第一个字符)
0CFA   C2   JNZ0D00;       若非"F",转至0D00
0CFD   C3   J0CC1;         若为"F",转至0CC1
0D00   FE   A-56;          (第一个字符)
0D02   C2   JNZ0C93;       若非"V",转至0C93
```

```
0D05  CD  JS0200;     ("V"),换行(控制台)
0D08  CD  JS0D66;     输入-输出
0D0B  FE  A-22;
0D0D  C2  JNZ 0D1E;   若非引号,转至0D1E
0D10  CD  JS0D66;     (" ")输入-输出
0D13  FE  A-22;
0D15  CA  JZ0436;     若为反引号,转至0436
0D18  CD  JS0D59;     存贮字节
0D1B  C3  J0D10;      循环至0D10
0D1E  FE  A-23;
0D20  C2  JNZ0463;    若非"#",则出错
0D23  CD  JS0D66;     输入-输出
0D26  CD  JS0B00;     "ASCII-hex"转换
0D29  78  A:=B;
0D2A  FE  A-00;
0D2C  FA  JN0463;     若非十六进制数,则出错
0D2F  87  A:+A;
0D30  87  A:+A;
0D31  87  A:+A;
0D32  87  A:+A;
0D33  57  D:=A;       高半
0D34  CD  JS0D66;     输入-输出
0D37  CD  JS0B00;     "ASCII-hex"转换
0D3A  78  A:=B;
0D3B  FE  A-00;
0D3D  FA  JN0463;     若非十六进制数,则出错
0D40  82  A:+D;       低半
0D41  CD  JS0D59;     存贮字节
0D44  CD  JS0D66;     输入-输出
0D47  FE  A-20;
0D49  CA  JZ0D44;
```

```
0D4C  FE  A-0D;
0D4E  CA  JZ0D44;          滤除空格或回车
0D51  FE  A-23;
0D53  C2  JNZ0D26;         若非"#",则循环
0D56  C3  J0436;           转至0436(主循环内)

MONITOR
>
  G 2800
0D59  C5  ST:=BC;          "存贮字节"子程序入口
0D5A  E5  ST:=HL;
0D5B  2A  HL:=MM1010;      目的程序计数器
0D5E  77  M:=A;
0D5F  23  HL:+1;
0D60  22  MM1010:=HL;
0D63  E1  HL:=ST;
0D64  C1  BC:=ST;
0D65  C9  RET;
0D66  CD  JS021B;          "输入-输出"子程序入口
0D69  4F  C:=A;
0D6A  CD  JS01F4;
0D6D  79  A:=C;
0D6E  C9  RET;
0D6F  C5  ST:=BC;          "HL:=sqrt(HL)×2⁸"子程序入口
0D70  D5  ST:=DE;
0D71  44  B:=H;
0D72  4D  C:=L;
0D73  78  A:=B;
0D74  E6  A:&C0;
0D76  C2  JNZ00B4;         若超出范围,则出错
0D79  CD  JS0BE9;          HL:=sqrt(BC),7b.p.
```

```
0D7C  29   HL:+HL;          8b.p
0D7D  D1   DE:=ST;
0D7E  C1   BC:=ST;
0D7F  C9   RET;
0D80  00   NULL;
0D81  00   NULL;
0D82  00   NULL;
0D83  00   NULL;
0D84  00   NULL;

MONITOR
>
  G 2800
0D85  0E   C:=05;            (由 0BC2 转至此)
0D87  FE   A-50;
0D89  C8   RZ;               C=05(表示"P"), 返回
0D8A  0D   C:-1;
0D8B  0D   C:-1;
0D8C  FE   A-4B;
0D8E  C8   RZ;               C=03(表示"K"),返回
0D8F  0D   C:-1;
0D90  0D   C:-1;
0D91  FE   A-5A;
0D93  C8   RZ;               C=01(表示"Z"),返回
0D94  0D   C:-1;
0D95  0D   C:-1;
0D96  C9   RET;              出错,C=FF 并返回
0D97  C8   RZ;               C=02(表示"NK"),返回
0D98  0D   C:-1;
0D99  0D   C:-1;
0D9A  C3   J0DC5;            转至 0DC5 继续执行
```

```
0D9D  00   NULL;
0D9E  FE   A-4C;          (第一个字符)
0DA0  C2   JNZ0DAA;       若非"L",转至0DAA
0DA3  23   HL:+1;         ("L")
0DA4  7E   A:=M;          第二个字符
0DA5  1E   E:=68;         (r 或 r1 将存入 L)
0DA7  C3   J058E;         转至058E继续执行
0DAA  FE   A-4D;          (第一个字符)
0DAC  C2   JNZ0890;       若非"M"转至0890
0DAF  23   HL:+1;         ("M")
0DB0  7E   A:=M;          第二个字符
0DB1  FE   A-3A;
0DB3  C2   JNZ0805;       若非":",转至0805
0DB6  23   HL:+1;         ("M:")
0DB7  23   HL:+1;
0DB8  7E   A:=M;          第四个字符
0DB9  FE   A-4D;
0DBB  CA   JZ0463;        若为"M",则出错
0DBE  2B   HL:-1;
0DBF  2B   HL:-1;
0DC0  1E   E:=70;         (r 或 r1 将存入 M)
0DC2  C3   J0593;         转至 0593 继续执行
0DC5  FE   A-5A;          (由 0D9A 转至此)
0DC7  C8   RZ;            C=00(表示"NZ")返回
0DC8  0D   C:-1;
0DC9  C9   RET;           出错, C=FF 并返回

MONITOR
>
  G 2800
0DCA  16   D:=2B;         "输出十进制整数(HL)"子程序入口
```

```
0DCC  7C  A:=H;
0DCD  FE  A-80;
0DCF  C2  JNZ0DD9;
0DD2  7D  A:=L;
0DD3  FE  A-00;
0DD5  CA  JZ0E4E;        若 HL=8000，转至 0E4E
0DD8  7C  A:=H;
0DD9  FE  A-00;
0DDB  F2  JNN0DE6;       除非数值为负，否则转至 0DE6
0DDE  14  D:+1;          (负)
0DDF  14  D:+1;          将"+"改为"-"
0DE0  2F  A:#;
0DE1  67  H:=A;
0DE2  7D  A:=L;
0DE3  2F  A:#;
0DE4  6F  L:=A;
0DE5  23  HL:+1;         取负操作
0DE6  01  BC:=D8F0;
0DE9  CD  JS0E5D;        除以 10000
0DEC  01  BC:=2710;
0DEF  09  HL:+BC;
0DF0  7B  A:=E;          ASCII 形式的 d4
0DF1  FE  A-30;
0DF3  C2  JNZ0DF9;       若商不是零，跳至 0DF9
0DF6  5A  E:=D;          移动符号，去除无意义的零
0DF7  16  D:=20;
0DF9  D5  ST:=DE;
0DFA  01  BC:=EC18;
0DFD  CD  JS0E5D;        除以 1000
0E00  01  BC:=03E8;
0E03  09  HL:+BC;
```

```
0E04  53  D:=E;                ASCII 形式的 d3
0E05  01  BC:=FF9C;
0E08  CD  JS0E5D;              除以 100
0E0B  01  BC:=0064;
0E0E  09  HL:+BC;              ASCII 形式的 d2
0E0F  26  H:=30;
0E11  7D  A:=L;
0E12  FE  A-0A;
0E14  FA  JN0E1D;              除以 10 (单字长)
0E17  D6  A:-0A;
0E19  24  H:+1;                ASCII 形式的 d1 在 H 中
0E1A  C3  J0E12;

MONITOR
>
  G 2800
0E1D  C6  A:+30;
0E1F  6F  L:=A;                ASCII 形式的 d0 在 L 中
0E20  C1  BC:=ST;
0E21  78  A:=B;
0E22  FE  A-20;
0E24  C2  JNZ0E3F;             去掉无意义的零,将符号移至紧靠最高
                               有效数字
0E27  7A  A:=D;
0E28  FE  A-30;
0E2A  C2  JNZ0E3F;
0E2D  51  D:=C;
0E2E  48  C:=B;
0E2F  7B  A:=E;
0E30  FE  A-30;
0E32  C2  JNZ0E3F;
```

```
0E35  5A   E:=D;
0E36  51   D:=C;
0E37  7C   A:=H;
0E38  FE   A-30;
0E3A  C2   JNZ0E3F;          打印形成十进制值的各位字符,在六位
                             的字段内靠右对齐
0E3D  63   H:=E;
0E3E  5A   E:=D;
0E3F  79   A:=C;
0E40  48   C:=B;
0E41  47   B:=A;
0E42  DF   JSS3;
0E43  48   C:=B;
0E44  DF   JSS3;
0E45  4A   C:=D;
0E46  DF   JSS3;
0E47  4B   C:=E;
0E48  DF   JSS3;
0E49  4C   C:=H;
0E4A  DF   JSS3;
0E4B  4D   C:=L;
0E4C  DF   JSS3;
0E4D  C9   RET;
0E4E  21   HL:=0E57;         (无法取负的)负极值的特殊处理
0E51  06   B:=06;
0E53  CD   JS005F;
0E56  C9   RET;

MONITOR
>
0E57 "-32768";
```

```
 G 2800
0E5D  1E   E:=30;          十进制输出用的"除法"子程序入口
0E5F  09   HL:+BC;
0E60  7C   A:=H;
0E61  FE   A-00;
0E63  F8   RN;
0E64  1C   E:+1;
0E65  C3   J0E5F;
0E68  26   H:=00;          "输出十进制整数(A)"子程序入口
0E6A  6F   L:=A;           将数值改成双字长,存在 HL 中,并转
                             至 0DCA
0E6B  FE   A-00;
0E6D  F2   JNN0DCA;
0E70  25   H:-1;
0E71  C3   J0DCA;

MONITOR
>
 G 2800
07E6  06   B:=05;          "输出十进制分数(HL),5d.p"子程序
                             入口
07E8  0E   C:=2E;          "输出十进制分数(HL),Bd.p"子程序
                             入口
07EA  DF   JSS3;           打印十进制小数点
07EB  97   A:-A;
07EC  29   HL:+HL;
07ED  8F   A:++A;          HL乘10,形成结果的整数部分存在A
                             中,结果的分数部分存在 HL 中
07EE  4F   C:=A;
07EF  54   D:=H;
07F0  5D   E:=L;
```

```
07F1  29   HL:+HL;
07F2  8F   A:++A;
07F3  29   HL:+HL;
07F4  8F   A:++A;
07F5  19   HL:+DE;
07F6  89   A:++C;          下一位数字在 A 中
07F7  C6   A:+30;          数字的 ASCII 形式
07F9  4F   C:=A;
07FA  DF   JSS3;           打印数字
07FB  05   B:-1;
07FC  C2   JNZ07EB;        重复执行,直至打印完规定的位数
07FF  C9   RET;

MONITOR
>
  G 2800
0E74  11   DE:=0000;       "A:=输入十进制整数"子程序入口
0E77  E7   JSS4;           读入第一个字符
0E78  FE   A-2B;
0E7A  CA   JZ0E83;         若为"+",跳至 0E83
0E7D  FE   A-2D;
0E7F  C2   JNZ0E84;        若非符号,跳至 0E84
0E82  14   D:+1;           若为"-",则设置负标记
0E83  E7   JSS4;           读入符号后的字符
0E84  EF   JSS5;           "ASCII-hex"转换
0E85  CD   JS01E6;         十六进制检查
0E88  5F   E:=A;
0E89  E7   JSS4;           读下一个字符
0E8A  FE   A-30;
0E8C  FA   JN0EB4;
0E8F  FE   A-3A;
```

```
0E91  F2  JNN0EB4;          若非十进制数字,则转至 OEB4
0E94  D6  A:-30;            数字的二进制值
0E96  47  B:=A;
0E97  7B  A:=E;
0E98  FE  A-0D;
0E9A  F2  JNN00B4;          若先前的值为 13 以上,则出错
0E9D  87  A:+A;
0E9E  4F  C:=A;
0E9F  87  A:+A;             先前的值乘以 10, 并加至新的数字上
0EA0  87  A:+A;
0EA1  81  A:+C;
0EA2  80  A:+B;
0EA3  5F  E:=A;
0EA4  FE  A-00;
0EA6  F2  JNN0E89;          若在范围内,则读入下一位数字
0EA9  87  A:+A;
0EAA  C2  JNZ00B4;  }若为 +128 与 ±129, 则出错, 但可接
0EAD  15  D:-1;     }  受 -128
0EAE  C2  JNZ00B4;
0EB1  E7  JSS4;     }读取结束字符(A:80)
0EB2  7B  A:=E;     }
0EB3  C9  RET;
0EB4  7B  A:=E;
0EB5  15  D:-1;
0EB6  F8  RN;               已读入正值,返回
0EB7  2F  A:#;              A 取补
0EB8  3C  A:+1;
0EB9  C9  RET;              已读入负值,返回

MONITOR
>
  G 2800
```

```
0EBA  21  HL:=0000;          "HL:=输入十进制整数"子程序入口
0EBD  54  D:=H;
0EBE  E7  JSS4;              读第一个字符
0EBF  FE  A-2B;
0EC1  CA  JZ0ECA;            若为"+",跳至0ECA
0EC4  FE  A-2D;
0EC6  C2  JNZ0ECB;           若非符号,跳至0ECB
0EC9  14  D:+1;              若为"-",则设置负标记
0ECA  E7  JSS4;              读取符号后的字符
0ECB  EF  JSS5;              "ASCII-hex"转换
0ECC  CD  JS01E6;            十六进制检查
0ECF  6F  L:=A;
0ED0  E7  JSS4;              读入下一个字符
0ED1  FE  A-30;
0ED3  FA  JN0F0D;
0ED6  FE  A-3A;
0ED8  F2  JNN0F0D;           若非十进制数字,转至0F0D
0EDB  D6  A:-30;             数字的二进制值
0EDD  5F  E:=A;
0EDE  01  BC:=F333;          -3277(十进制)
0EE1  09  HL:+BC;
0EE2  7C  A:=H;
0EE3  FE  A-00;
0EE5  F2  JNN00B4;           若先前的值为3277以上,则出错
0EE8  01  BC:=0CCD;          +3277(十进制)
0EEB  09  HL:+BC;
0EEC  29  HL:+HL;
0EED  44  B:=H;
0EEE  4D  C:=L;
0EEF  29  HL:+HL;  }
0EF0  29  HL:+HL;  }         先前的值乘以10并加至新数字上
```

```
0EF1  09   HL:+BC;
0EF2  06   B:=00;
0EF4  4B   C:=E;
0EF5  09   HL:+BC;
0EF6  7C   A:+H;
0EF7  FE   A-00;
0EF9  F2   JNN0ED0;          若在范围内,则读入下一位数字
0EFC  FE   A-80;
0EFE  C2   JNZ00B4;
0F01  7D   A:=L;             若为 ±32768, ±32769 则出错, 但可
0F02  FE   A-00;               接受 -32768
0F04  C2   JNZ00B4;
0F07  15   D:-1;
0F08  C2   JNZ00B4;
0F0B  F7   JSS4;             读取结束字符
0F0C  C9   RET;              (HL=8000)
0F0D  15   D:-1;
0F0E  F8   RN;               已读入正值,返回
0F0F  7C   A:=H;
0F10  2F   A:#;
0F11  67   H:=A;
0F12  7D   A:=L;             HL 取补
0F13  2F   A:#;
0F14  6F   L:=A;
0F15  23   HL:+1;
0F16  C9   RET;              已读入负值,返回
0F17  00   NULL;

MONITOR
>
  G 2800
```

```
0F18  79   A:=C;              "符号修正(C)"子程序入口
0F19  FE   A-00;
0F1B  F0   RNN;
0F1C  7C   A:=H;
0F1D  93   A:-E;
0F1E  67   H:=A;
0F1F  C9   RET;
0F20  21   HL:=0000;         "HL:=C×E(无符号)"子程序入口
0F23  79   A:=C;
0F24  54   D:=H;
0F25  06   B:=08;
0F27  29   HL:+HL;
0F28  07   A:@L;
0F29  D2   JNK0F2D;
0F2C  19   HL:+DE;
0F2D  05   B:-1;
0F2E  C2   JNZ0F27;
0F31  C9   RET;
0F32  7B   A:=E;             "符号修正(E)"子程序入口
0F33  FE   A-00;
0F35  F0   RNN;
0F36  7C   A:=H;
0F37  91   A:-C;
0F38  67   H:=A;
0F39  C9   RET;
0F3A  CD   JS0F20;           "HL:=C×E(带符号)"子程序入口
0F3D  CD   JS0F18;
0F40  CD   JS0F32;
0F43  C9   RET;
0F44  F5   ST:=AF;           "DE:=D×E(无符号)"子程序入口
0F45  C5   ST:=BC;
```

```
0F46   E5   ST:=HL;
0F47   4A   C:=D;
0F48   CD   JS0F20;
0F4B   EB   HL:=:DE;
0F4C   E1   HL:=ST;
0F4D   C1   BC:=ST;
0F4E   F1   AF:=ST;
0F4F   C9   RET;

MONITOR
>
  G 2800
0F50   0C   C:+1;          "HL:=(B/D)×2^C(无符号)"子程序
                           入口
0F51   7A   A:=D;
0F52   B8   A-B;
0F53   F2   JNN0F5D;
0F56   0C   C:+1;
0F57   87   A:+A;
0F58   F2   JNN0F52;
0F5B   1F   AK:@R;
0F5C   0D   C:-1;
0F5D   57   D:=A;
0F5E   21   HL:=0000;      (见§4.5.5)
0F61   29   HL:+HL;
0F62   7C   A:=H;
0F63   FE   A-00;
0F65   FA   JN00B4;
0F68   78   A:=B;
0F69   BA   A-D;
0F6A   FA   JN0F6F;
```

```
0F6D  92  A:-D;
0F6E  23  HL:+1;
0F6F  87  A:+A;
0F70  47  B:=A;
0F71  0D  C:-1;
0F72  C2  JNZ0F61;
0F75  C9  RET;
0F76  1E  E:=00;          "HL:=(BC)×2^C(带符号)"子程序入口

0F78  78  A:=B;
0F79  FE  A-00;
0F7B  F2  JNN0F82;        被除数的绝对值
0F7E  2F  A:#;
0F7F  3C  A:+1;
0F80  47  B:=A;
0F81  1C  E:+1;
0F82  7A  A:=D;
0F83  FE  A-00;
0F85  F2  JNN0F8C;        除数的绝对值
0F88  2F  A:#;
0F89  3C  A:+1;
0F8A  57  D:=A;
0F8B  1D  E:-1;
0F8C  CD  JS0F50;         无符号除法
0F8F  7B  A:=E;
0F90  FE  A-00;           符号检验
0F92  C8  RZ;             已得到正的商,返回
0F93  7C  A:=H;
0F94  2F  A:#;
0F95  67  H:=A;
0F96  7D  A:=L;           改变HL的符号
```

```
0F97  2F   A:#;
0F98  6F   L:=A;
0F99  23   HL:+1;
0F9A  C9   RET;              已得到负的商,返回

MONITOR
>
  G 2800
0F9B  EB   HL:=:DE;          "HL:=(BC/DE)×2^4(无符号)"子程
                               序入口
0F9C  C5   ST:=BC;
0F9D  D1   DE:=ST;
0F9E  47   B:=A;
0F9F  04   B:+1;
0FA0  CD   JS0C1A;           HL-DE
0FA3  F2   JNN0FAF;
0FA6  04   B:+1;
0FA7  29   HL:+HL;
0FA8  D2   JNK0FA0;
0FAB  CD   JS0FF2;           HL:/2(无符号,截尾)
0FAE  05   B:-1;
0FAF  EB   HL:=:DE;
0FB0  E5   ST:=HL;
0FB1  21   HL:=0000;         (见§4.5.6)
0FB4  29   HL:+HL;
0FB5  DA   JK00B4;
0FB8  E3   HL:=:ST;
0FB9  CD   JS0C1A;           HL-DE
0FBC  FA   JN0FC5;
0FBF  CD   JS0CF1;           HL:-DE
0FC2  E3   HL:=:ST;
```

```
0FC3  23  HL:+1;
0FC4  E3  HL:=:ST;
0FC5  29  HL:+HL;
0FC6  E3  HL:=:ST;
0FC7  05  B:-1;
0FC8  C2  JNZ0FB4;
0FCB  C1  BC:=ST;
0FCC  C9  RET;
0FCD  F5  ST:=AF;          "HL:=(BC/DE)×2^(带符号)"子程
                           序入口
0FCE  26  H:=00;
0FD0  78  A:=B;
0FD1  A7  A:&A;
0FD2  F2  JNN0FDC;         被除数的绝对值
0FD5  2F  A:#;
0FD6  47  B:=A;
0FD7  79  A:=C;
0FD8  2F  A:#;
0FD9  4F  C:=A;
0FDA  03  BC:+1;
0FDB  24  H:+1;
0FDC  7A  A:=D;
0FDD  A7  A:&A;
0FDE  F2  JNN0FE8;         除数的绝对值
0FE1  2F  A:#;
0FE2  57  D:=A;
0FE3  7B  A:=E;
0FE4  2F  A:#;
0FE5  5F  E:=A;
0FE6  13  DE:+1;
0FE7  25  H:-1;
```

```
0FE8  F1   AF:=ST;
0FE9  E5   ST:=HL;
0FEA  CD   JS0F9B;          无符号除法
0FED  F1   AF:=ST;
0FEE  C3   J0F90;           转至0F90进行符号修正

MONITOR
>
  G 2800
0FF1  23   HL:+1;           "HL:/2(无符号,四舍五入)"子程序入
                              口
0FF2  7D   A:=L;            "HL:/2(无符号,截尾)"子程序入口
0FF3  E6   A:&FE;
0FF5  0F   A:@R;
0FF6  6F   L:=A;
0FF7  7C   A:=H;
0FF8  1F   AK:@R;
0FF9  67   H:=A;
0FFA  D0   RNK;
0FFB  7D   A:=L;
0FFC  EE   A:#80;
0FFE  6F   L:=A;
0FFF  C9   RET;

MONITOR
>
  G 2800
2400  3E   A:=10;           "DEHL:=BC×DE(无符号)"子程序
                              入口
2402  F5   ST:=AF;
2403  97   A:-A;
```

```
2404  67  H:=A;
2405  6F  L:=A;
2406  EB  HL:=:DE;
2407  29  HL:+HL;
2408  EB  HL:=:DE;
2409  1F  AK:@R;
240A  A7  A:&A;
240B  29  HL:+HL;
240C  D2  JNK2410;          (见§4.5.3)
240F  13  DE:+1;
2410  F2  JNN2418;
2413  09  HL:+BC;
2414  D2  JNK2418;
2417  13  DE:+1;
2418  E3  HL:=:ST;
2419  25  H:-1;
241A  E3  HL:=:ST;
241B  C2  JNZ2406;
241E  F1  AF:=ST;
241F  C9  RET;
2420  21  HL:=0000;        "DEHL:=BC×DE(带符号)"子程序
                          入口
2423  78  A:=B;
2424  A7  A:&A;
2425  F2  JNN2429;
2428  19  HL:+DE;          对 BC 进行符号修正
2429  7A  A:=D;
242A  A7  A:&A;
242B  F2  JNN242F;
242E  09  HL:+BC;          对 DE 进行符号修正
242F  7C  A:=H;
```

```
2430   2F   A:#;
2431   67   H:=A;
2432   7D   A:=L;
2433   2F   A:#;
2434   6F   L:=A;
2435   23   HL:+1;          总的符号修正值取补
2436   E5   ST:=HL;
2437   CD   JS2400;        DEHL:=BC×DE(无符号)
243A   EB   HL:=:DE;
243B   C1   BC:=ST;
243C   09   HL:+BC;        符号修正
243D   EB   HL:=:DE;
243E   C9   RET;
243F   00   NULL;
2440   CD   JS2420;        "HL:=BC×DE(全部是15b. p.)"子
                           程序入口
2443   EB   HL:=:DE;
2444   29   HL:+HL;
2445   EB   HL:=:DE;
2446   29   HL:+HL;
2447   D2   JNK244B;
244A   13   DE:+1;         DEHL 左移一位
244B   29   HL:+HL;
244C   D2   JNK2450;
244F   13   DE:+1;
2450   EB   HL:=:DE;       结果四舍五入成双字长
2451   C9   RET;

MONITOR
>
 G 2800
```

```
2452  11  DE:=0000;          "多项式(15b.p.)"P:=0 子程序入口
2455  D5  ST:=DE;
2456  5E  E:=M;
2457  23  HL:+1;
2458  56  D:=M;              下一个字数 a_i 送入 DE
2459  23  HL:+1;
245A  E3  HL:=:ST;
245B  19  HL:+DE;           P:+a_i
245C  A7  A:&A;
245D  CA  JZ246D;           若 i=0,则结束
2460  3D  A:-1;             i:-1
2461  EB  HL:=:DE;
2462  F5  ST:=AF;
2463  C5  ST:=BC;
2464  CD  JS2440;           HL:=BC×DE(全部是 15b.p.)P:××
2467  C1  BC:=ST;
2468  F1  AF:=ST;
2469  E3  HL:=:ST;
246A  C3  J2456;            循环,计算下一个系数
246D  D1  DE:=ST;
246E  C9  RET;
246F  00  NULL;
2470  E5  ST:=HL;           "HL:=sin(HL)15b.p."子程序入口
2471  EB  HL:=:DE;
2472  42  B:=D;
2473  4B  C:=E;
2474  CD  JS2440;           HL:=BC×DE(15b.p.)
2477  44  B:=H;
2478  4D  C:=L;             自变量的值
2479  3E  A:=03;            多项式的次数
247B  21  HL:=2488;         系数的存储单元
```

```
247E  CD  JS2452;          求多项式的值(15b.p.)
2481  EB  HL:=:DE;
2482  C1  BC:=ST;
2483  CD  JS2440;          HL:=BC×DE(15b.p.)
2486  29  HL:+HL;
2487  C9  RET;

MONITOR
>
2488  # FD FF 89 00 55 F5 00 40 #
  G 2800
2490  CD  JS2470;          "HL:=cos(HL)15b.p."子程序的入口
2493  EB  HL:=:DE;         正弦值在 DE 中
2494  42  B:=D;
2495  4B  C:=E:
2496  CD  JS2440;          HL:=BC×DE(15b.p.)
2499  CD  JS24CF;          HL:/2(带符号,四舍五入)
249C  EB  HL:=DE;
249D  21  HL:=4000;
24A0  CD  JS0CF1;          HL:-DE
24A3  CD  JS0D6F;          HL:=sqrt(HL)×2^8
24A6  C9  RET;

MONITOR
>
  G 2800
24C1  67  H:=A;            "HL(弧度, 15b.p.):=A(度, 0b.p.)"
                             子程序入口
24C2  2E  L:=00;
24C4  29  HL:+HL;          "HL(弧度,15b.p.):=HL(度,8b.p.)"
                             子程序入口
```

```
24C5  44  B:=H;
24C6  4D  C:=L;
24C7  11  DE:=477D;          (π/180, 20b.p.)
24CA  CD  JS2440;            HL:=BC×DE(15b.p.)
24CD  29  HL:+HL;
24CE  C9  RET;
24CF  CD  JS0FF1;  ┐         "HL:/2 (带符号，四舍五入)"子程序
24D2  7C  A:=H;    │           入口
24D3  E6  A:&40;   │
24D5  87  A:+A;    ├         作为无符号数右移并复制符号位
24D6  84  A:+H;    │
24D7  67  H:=A;    │
24D8  C9  RET;     ┘
24D9  CD  JS2470;            "HL:=tan(HL)15b.p."子程序入口
24DC  E5  ST:=HL;            正弦值存入堆栈
24DD  CD  JS2493;            求余弦(从较后的入口进入)
24E0  C1  BC:=ST;
24E1  EB  HL:=:DE;
24E2  3E  A:=0F;
24E4  CD  JS0FCD;            tan=sin/cos
24E7  C9  RET;
24E8  0E  C:=20;             "输出带符号分数(HL15b.p.)"子程序
                               入口
24EA  DF  JSS3;
24EB  0E  C:=2B;
24ED  29  HL:+HL;            16b.p.
24EE  D2  JNK24F6;
24F1  CD  JS0F93;            HL:-
24F4  0E  C:=2D;
24F6  DF  JSS3;
24F7  0E  C:=30;
```

```
24F9  DF  JSS3;          打印"+0"或"-0"
24FA  06  B:=04;         四个十进制小数位
24FC  CD  JS07E8;        输出十进制分数
24FF  C9  RET;

MONITOR
>
  G 2800
2500  44  B:=H;          "HL(度,8b.p.):=HL(弧度,15b.p.)"
                           子程序入口
2501  4D  C:=L;
2502  11  DE:=7297;      (180/π, 9b.p.)
2505  CD  JS2420;        DEHL:=BC×DE(带符号)
2508  EB  HL:=:DE;
2509  C9  RET;

MONITOR
>
  G 2800
250A  E5  ST:=HL;        "BCDEHL:=BHL×CDE(带符号)"
                           子程序入口
250B  D5  ST:=DE;
250C  C5  ST:=BC;
250D  44  B:=H;
250E  4D  C:=L;
250F  CD  JS2400;        DEHL:=BC×DE(无符号)
2512  C1  BC:=ST;
2513  E3  HL:=:ST;
2514  D5  ST:=DE;
2515  C5  ST:=BC;
2516  EB  HL:=:DE;
```

```
2517  48  C:=B;
2518  79  A:=C;
2519  17  AK:@L;
251A  9F  A:--A;
251B  47  B:=A;
251C  CD  JS2556;          DEHL:=BC(带符号)×DE(无符号)
251F  E3  HL:=:ST;
2520  D5  ST:=DE;
2521  EB  HL:=:DE;
2522  4A  C:=D;
2523  CD  JS0F3A;          HL:=C×E(带符号)
2526  E5  ST:=HL;
2527  7B  A:=E;
2528  17  AK:@L;
2529  9F  A:--A;
252A  57  D:=A;
252B  21  HL:=000A;
252E  39  HL:+SP;
252F  F9  SP:=HL;
2530  C1  BC:=ST;
2531  21  HL:=FFF4;
2534  39  HL:+SP;
2535  F9  SP:=HL;
2536  CD  JS2562;          DEHL:=BC(无符号)×DE(带符号)
2539  C1  BC:=ST;
253A  EB  HL:=:DE;
253B  09  HL:+BC;
253C  44  B:=H;
253D  4D  C:=L;
253E  E1  HL:=ST;
253F  09  HL:+BC;
```

```
2540   44   B:=H;
2541   4D   C:=L;
2542   E1   HL:=ST;
2543   19   HL:+DE;
2544   D2   JNK2548;
2547   03   BC:+1;
2548   54   D:=H;
2549   5D   E:=L;
254A   E1   HL:=ST;
254B   19   HL:+DE;
254C   D2   JNK2550;
254F   03   BC:+1;
2550   54   D:=H;
2551   5D   E:=L;
2552   E1   HL:=ST;
2553   33   SP:+1;
2554   33   SP:+1;
2555   C9   RET;

MONITOR
>
  G 2800
2556   21   HL:=0000;      "DEHL:=BC(带符号)×DE(无符号)"
                           子程序入口
2559   78   A:=B;
255A   A7   A:&A;
255B   F2   JNN242F;  }
255E   19   HL:+DE;   }   若有必要，则对 BC 进行符号修正，并
255F   C3   J242F;    }   转至 242F
2562   21   HL:=0000;      "DEHL:=BC(无符号)×DE(带符号)"
                           子程序入口
```

```
2565  C3  J2429;          避免对 BC 进行符号修正
2568  F5  ST:=AF;         "AHL:/2(截尾)"子程序入口
2569  CD  JS0FF2;         HL:/2(无符号,截尾)
256C  F1  AF:=ST;
256D  07  A:@L;
256E  1F  AK:@R;
256F  1F  AK:@R;
2570  F5  ST:=AF;
2571  7C  A:=H;
2572  17  AK:@L;
2573  0F  A:@R;
2574  67  H:=A;
2575  F1  AF:=ST;
2576  C9  RET;
2577  D5  ST:=DE;         "AHL:/2(四舍五入)"子程序入口
2578  11  DE:=0001;
257B  19  HL:+DE;
257C  CE  A:++00;
257E  CD  JS2568;        AHL:/2(截尾)
2581  D1  DE:=ST;
2582  C9  RET;
2583  CD  JS2568;        "AHL:/2^B(截尾)"子程序入口
2586  05  B:-1;
2587  C2  JNZ2583;
258A  C9  RET;
258B  05  B:-1;         "AHL:/2^B(四舍五入)"子程序入口
258C  CA  JZ2595;
258F  CD  JS2568;        AHL:/2(截尾)
2592  C3  J258B;
2595  D5  ST:=DE;
2596  11  DE:=0001;
```

```
2599   19   HL:+DE;
259A   CE   A:++00;
259C   CD   JS2568;              AHL:/2(截尾)
259F   D1   DE:=ST;
25A0   C9   RET;

MONITOR
>
  G 2800
25A1   06   B:=00;               "左规格化(AHL)"子程序入口
25A3   F5   ST:=AF;              "左规格化"入口
25A4   A7   A:&A;
25A5   C2   JNZ25B2;
25A8   7C   A:=H;
25A9   A7   A:&A;
25AA   C2   JNZ25B2;
25AD   7D   A:=L;
25AE   A7   A:&A;
25AF   CA   JZ25C7;
25B2   F1   AF:=ST;
25B3   F5   ST:=AF;
25B4   E6   A:&E0;
25B6   FE   A-20;
25B8   CA   JZ25C9;
25BB   FE   A-C0;
25BD   CA   JZ25C9;
25C0   F1   AF:=ST;
25C1   29   HL:+HL;
25C2   8F   A:++A;
25C3   05   B:-1;
25C4   C3   J25A3;
```

```
25C7  06  B:=00;
25C9  F1  AF:=ST;
25CA  C9  RET;
25CB  4F  C:=A;          "右规格化"子程序入口
25CC  07  A:@L;
25CD  17  AK:@L;
25CE  CE  A:++00;
25D0  1F  AK:@R;
25D1  79  A:=C;
25D2  D0  RNK;
25D3  04  B:+1;
25D4  CD  JS2577;        AHL:/2(四舍五入)
25D7  C9  RET;
25D8  C5  ST:=BC;        "AHL:-CDE"子程序入口
25D9  47  B:=A;
25DA  7D  A:=L;
25DB  93  A:-F;
25DC  6F  L:=A;
25DD  7C  A:=H;
25DE  9A  A:--D;
25DF  67  H:=A;
25E0  78  A:=B;
25E1  99  A:--C;
25E2  C1  BC:=ST;
25E3  C9  RET;

MONITOR
>
  G 2800
25E4  EB  HL:=:DE;       "AHL(23b.p.):=BCDEHL(44b.p.)
                         四舍五入"子程序入口
```

```
25E5  50   D:=B;
25E6  59   E:=C;
25E7  01   BC:=0020;
25EA  09   HL:+BC;
25EB  D2   JNK 25EF;
25EE  13   DE:+1;
25EF  EB   HL:=:DE;
25F0  29   HL:+HL;
25F1  EB   HL:=:DE;
25F2  29   HL:+HL;
25F3  D2   JNK25F7;
25F6  13   DE:+1;
25F7  EB   HL:=:DE;
25F8  29   HL:+HL;
25F9  EB   HL:=:DE;
25FA  29   HL:+HL;
25FB  D2   JNK25FF;
25FE  13   DE:+1;
25FF  6C   L:=H;
2600  63   H:=E;
2601  7A   A:=D;
2602  C9   RET;
2603  00   NULL;
2604  00   NULL;
2605  00   NULL;
2606  00   NULL;
2607  00   NULL;

MONITOR
>
  G 2800
```

```
2608  F5  ST:=AF;          "M4[BC]:=DEHL"(浮点存数)子程
                           序入口
2609  7D  A:=L;
260A  02  M[BC]:=A;
260B  03  BC:+1;
260C  7C  A:=H;
260D  02  M[BC]:=A;
260E  03  BC:+1;
260F  7B  A:=E;
2610  02  M[BC]:=A;
2611  03  BC:+1;
2612  7A  A:=D;
2613  02  M[BC]:=A;
2614  0B  B:-1;
2615  0B  BC:-1;
2616  0B  BC:-1;
2617  F1  AF:=ST;
2618  C9  RET;
2619  F5  ST:=AF;          "DEHL:=M4[BC]"(浮点取数)子程
                           序入口
261A  0A  A:=M[BC];
261B  6F  L:=A;
261C  03  BC:+1;
261D  0A  A:=M[BC];
261E  67  H:=A;
261F  03  BC:+1;
2620  0A  A:=M[BC];
2621  5F  E:=A;
2622  03  BC:+1;
2623  0A  A:=M[BC];
2624  57  D:=A;
```

```
2625   0B   BC:-1;
2626   0B   BC:-1;
2627   0B   BC:-1;
2628   F1   AF:=ST;
2629   C9   RET;

MONITOR
>
  G 2800
262A   2F   A:#;          (浮点打印子程序的继续)
262B   3C   A:+1;
262C   0E   A:=30;
262E   D6   A:-0A;
2630   FA   JN2637;
2633   0C   C:+1;
2634   C3   J262E;
2637   C6   A:+3A;
2639   47   B:=A;
263A   79   A:=C;
263B   C3   J27F7;
263E   CD   JS250A;       (浮点乘法子程序的继续)
2641   CD   JS25E4;       尾数之乘积截成 22b. p.
2644   C1   BC:=ST;
2645   CD   JS25A3;       左规格化
2648   50   D:=B;
2649   5F   E:=A;
264A   C9   RET;

MONITOR
>
  G 2800
```

```
264B  D5  ST:=DE;          "DEHL:×M4[BC]"(浮点乘法)子程
                             序入口
264C  E5  ST:=HL;
264D  CD  JS2619;          取浮点数
2650  7A  A:=D;
2651  43  B:=E;
2652  D1  DE:=ST;
2653  E3  HL:=:ST;
2654  84  A:+H;           阶码的和
2655  4D  C:=L;
2656  E1  HL:=ST;        尾数在 BHL, CDE 中
2657  F5  ST:=AF;
2658  C3  J263E;         转至 263E 继续执行
265B  EB  HL:=:DE;       "DEHL:-"(浮点取补)子程序入口
265C  44  B:=H;
265D  4D  C:=L;
265E  97  A:-A;
265F  67  H:=A;          清除 AHL
2660  6F  L:=A;
2661  CD  JS25D8;        AHL:-CDE
2664  CD  JS25CB;        右规格化
2667  5F  E:=A;
2668  50  D:=B;
2669  C9  RET;
266A  D5  ST:=DE;        "M4[DF]:-"子程序入口
266B  42  B:=D;
266C  4B  C:=E;
266D  CD  JS2619;        取浮点数
2670  CD  JS265B;        浮点数取补
2673  C1  BC:=ST;
2674  C5  ST:=BC;
```

```
2675   CD   JS2608;          存浮点数
2678   D1   DE:=ST;
2679   C9   RET;

MONITOR
>
  G 2800
267A   7A   A:=D;            "DEHL:=1/DEHL"(浮点，无符号)
                             子程序入口
267B   2F   A:#;
267C   3C   A:+1;
267D   3C   A:+1;
267E   F5   ST:=AF;
267F   EB   HL:=:DE;
2680   4D   C:=L;
2681   21   HL:=0000;
2684   3E   A:=20;
2686   06   B:=17;
2688   F5   ST:=AF;
2689   E5   ST:=HL;
268A   7C   A:=H;
268B   29   HL:+HL;
268C   8F   A:++A;
268D   FA   JN00B4;
2690   E3   HL:=:ST;
2691   F5   ST:=AF;
2692   33   SP:+1;
2693   33   SP:+1;
2694   33   SP:+1;
2695   33   SP:+1;
2696   F1   AF:=ST;
```

```
2697   3B   SP:-1;
2698   3B   SP:-1;
2699   3B   SP:-1;
269A   3B   SP:-1;
269B   3B   SP:-1;
269C   3B   SP:-1;
269D   CD   JS25D8;          AHL:-CDE
26A0   F2   JNN26B7;
26A3   19   HL:+DE;
26A4   89   A:++C;
26A5   29   HL:+HL;
26A6   8F   A:++A;
26A7   33   SP:+1;
26A8   33   SP:+1;
26A9   33   SP:+1;
26AA   33   SP:+1;
26AB   33   SP:+1;
26AC   33   SP:+1;
26AD   F5   ST:=AF;
26AE   3B   SP:-1;
26AF   3B   SP:-1;
26B0   3B   SP:-1;
26B1   3B   SP:-1;
26B2   F1   AF:=ST;
26B3   E3   HL:=:ST;
26B4   C3   J26CE;

MONITOR
>
  G 2800
26B7   29   HL:+HL;          (DEHL:=1/DEHL子程序的继续)
```

```
26B8  8F  A:++A;
26B9  33  SP:+1;
26BA  33  SP:+1;
26BB  33  SP:+1;
26BC  33  SP:+1;
26BD  33  SP:+1;
26BE  33  SP:+1;
26BF  F5  ST:=AF;
26C0  3B  SP:-1;
26C1  3B  SP:-1;
26C2  3B  SP:-1;
26C3  3B  SP:-1;
26C4  F1  AF:=ST;
26C5  E3  HL:=:ST;
26C6  D5  ST:=DE;
26C7  11  DE:=0001;
26CA  19  HL:+DE;
26CB  CE  A:++00;
26CD  D1  DE:=ST;
26CE  05  B:-1;          步数计数器
26CF  C2  JNZ268B;       重复移位与减法,直至完成 23 遍
26D2  D1  DE:=ST;
26D3  D1  DE:=ST;
26D4  C1  BC:=ST;
26D5  C3  J2664;         转至 2664, 右规格化
26D8  00  NULL;
26D9  7B  A:=E;          "DEHL:=1/DEHL (带符号)" (浮点
                           倒数)子程序入口
26DA  A7  A:&A;
26DB  FA  JN26E2;        根据尾数的符号转移(§ 4.7.7)
26DE  CD  JS267A;        求无符号的倒数
```

```
26E1  C9  RET;              已得正的结果,返回
26E2  CD  JS265B;           浮点数取补
26E5  CD  JS267A;           求无符号的倒数
26E8  CD  JS265B;           浮点数取补
26EB  C9  RET;              已得负的结果,返回
26EC  D5  ST:=DE;           "DEHL:/M4[BC]"(浮点除法)子程
                              序入口
26ED  E5  ST:=HL;
26EE  CD  JS2619;           取浮点数
26F1  CD  JS26D9;           浮点倒数
26F4  C3  J2650;            转至2650,执行乘法

MONITOR
>
  G 2800
26F7  E5  ST:=HL;           "DEHL:=M4[HL]+M4[DE]"(浮点
                              加法)子程序入口
26F8  D5  ST:=DE;
26F9  23  HL:+1;
26FA  23  HL:+1;
26FB  C3  J24A7;            转至24A7继续执行
26FE  13  DE:+1;            (浮点加法的继续)
26FF  1A  A:=M[DE];
2700  96  A:-M;             阶码的差
2701  FA  JN270C;
2704  47  B:=A;             若为正,尾数互换
2705  D1  DE:=ST;
2706  E1  HL:=ST;
2707  D5  ST:=DE;
2708  EB  HL:=:DE;
2709  C3  J2710;
```

```
270C  2F  A:#;                  若为负,则改变符号
270D  3C  A:+1;
270E  47  B:=A;
270F  D1  DE:=ST;
2710  1A  A:=M[DE];             取要移位的尾数
2711  CF  L:=A;
2712  13  DE:+1;
2713  1A  A:=M[DE];
2714  67  H:=A;
2715  13  DE:+1;
2716  1A  A:=M[DE];
2717  C4  JSN258B;              若B>0,则 AHL:/2^B(四舍五入)
271A  E3  HL:=:ST;
271B  5E  E:=M;                 取另一个尾数
271C  23  HL:+1;
271D  56  D:=M;
271E  23  HL:+1;
271F  4E  C:=M;
2720  23  HL:+1;
2721  E3  HL:=:ST;
2722  19  HL:+DE;               尾数相加(二进制小数位已相等)
2723  89  A:++C;
2724  D1  DE:=ST;
2725  EB  HL:=:DE;
2726  46  B:=M;                 未规格化的阶码
2727  EB  HL:=:DE;
2728  CD  JS25CB;               右规格化
272B  CD  JS25A3;               左规格化
272E  5F  E:=A;
272F  50  D:=B;
2730  C9  RET;
```

```
2731   D5   ST:=DE;          "DEHL:=M4[HL]-M4[DE]"(浮点
                              减法)子程序入口
2732   E5   ST:=HL;
2733   CD   JS266A;          M4[DE]:-
2736   E1   HL:=ST;
2737   CD   JS26F7;          浮点加法
273A   C1   BC:=ST;
273B   E5   ST:=HL;
273C   D5   ST:=DE;
273D   50   D:=B;
273E   59   E:=C;
273F   CD   JS266A;          M4[DE]:-
2742   D1   DE:=ST;
2743   E1   HL:=ST;
2744   C9   RET;

MONITOR
>
  G 2800
24A7   7E   A:=M;            (浮点加法的继续)
24A8   A7   A:&A;
24A9   CA   JZ24BA;          若 M4[HL]=0E0,则转至 24BA
24AC   23   HL:+1;
24AD   13   DE:+1;
24AE   13   DE:+1;
24AF   1A   A:=M[DE];
24B0   A7   A:=&A;
24B1   C2   JNZ26FE;         除 M4[DE]=0E0 外均转至 26FE
24B4   F1   AF:=ST;          (M4[DE]=0E0)
24B5   C1   BC:=ST;
24B6   CD   JS2619;          取浮点数
```

```
24B9   C9   RET;              结果是 M4[HL]
24BA   C1   BC:=ST;           (M4[HL]=0E0)
24BB   F1   AF:=ST;
24BC   CD   JS2619;           取浮点数
24BF   C9   RET;              结果是 M4[DE]
24C0   00   NULL;

MONITOR
>
  G 2800
2745   E7   JSS4;             "DEHL:=输入浮点数"(浮点读入)子
                                程序入口
2746   01   BC:=0000;
2749   FE   A-2B;
274B   CA   JZ2753;
274E   FE   A-2D;
2750   CA   JZ2757;
2753   4F   C:=A;
2754   C3   J2758;
2757   05   B:-1;
2758   C5   ST:=BC;           符号标记入栈
2759   97   A:-A;             清除 AHL 并
275A   67   H:=A;
275B   6C   L:=H;
275C   E5   ST:=HL;           将其存入堆栈作为尾数的起始值
275D   F5   ST:=AF;
275E   79   A:=C;
275F   A7   A:&A;
2760   C2   JNZ2764;
2763   E7   JSS4;             读下一个字符
2764   D6   A:-30;
```

```
2766   FA   JN278C;
2769   FE   A-0A;          尾数结束时转至 278C
276B   F2   JNN278C;
276E   4F   C:=A;          (读入十进制数字)
276F   F1   AF:=ST;
2770   E3   HL:=:ST;       尾数的先前值在 AHL 中
2771   29   HL:+HL;
2772   8F   A:++A;
2773   47   B:=A;
2774   54   D:=H;
2775   5D   E:=L;
2776   29   HL:+HL;
2777   8F   A:++A;
2778   29   HL:+HL;
2779   8F   A:++A;
277A   19   HL:+DE;
277B   88   A:++B;         先前值乘以 10
277C   16   D:=00;
277E   59   E:=C;
277F   19   HL:+DE;
2780   8A   A:++D;         新数字加入先前值
2781   E3   HL:=:ST;
2782   F5   ST:=AF;        新的尾数值存入堆栈
2783   7C   A:=H;
2784   A7   A:&A;          十进制小数点标记
2785   CA   JZ2763;        若无标记,转回读入下一个字符
2788   2D   L:-1;          若有标记,从十进制阶码中取出"1"
2789   C3   J2763;         循环,取下一个字符

MONITOR
>
```

G 2800

```
278C   FE   A-FE;          (浮点读入子程序的继续)
278E   C2   JNZ2795;       若结束符非十进制小数点，转至2795
2791   24   H:+1;  }
2792   C3   J2763;  }      设置十进制小数点标记并继续读入尾数
2795   45   B:=L;          尾数中十进制小数位的负计数值
2796   F1   AF:=ST;
2797   E1   HL:=ST;
2798   C5   ST:=BC;
2799   06   B:=16;
279B   CD   JS25A3;        规格化
279E   50   D:=B;
279F   5F   E:=A;
27A0   D5   ST:=DE;
27A1   E5   ST:=HL;        以浮点整数形式将尾数存入堆栈
27A2   CD   JS0E74;        A;=输入十进制整数(阶码)
27A5   E1   HL:=ST;
27A6   D1   DE:=ST;
27A7   C1   BC:=ST;
27A8   80   A:+B;          阶码小于尾数中十进制小数位的位数
27A9   01   BC:=27DF;      10的幂所在的存贮单元
27AC   E5   ST:=HL;
27AD   26   H:=00;
27AF   A7   A:&A;
27B0   F2   JNN27B6;
27B3   2F   A:#;
27B4   3C   A:+1;
27B5   25   H:-1;          阶码的符号与模分开
27B6   0F   A:@R;          模的下一位存入K
27B7   D2   JNK27D0;       若该位为零，跳至27D
27BA   6F   L:=A;
```

```
27BB  7C  A:=H;
27BC  E3  HL:=:ST;
27BD  A7  A:&A;
27BE  C5  ST:=BC;
27BF  F4  JSNN264B;          若阶码为正,执行乘法
27C2  C1  BC:=ST;
27C3  E3  HL:=:ST;
27C4  7C  A:=H;
27C5  E3  HL:=:ST;
27C6  A7  A:&A;
27C7  C5  ST:=BC;
27C8  FC  JSN26EC;           若阶码为负,执行除法
27CB  C1  BC:=ST;
27CC  E3  HL:=:ST;
27CD  7D  A:=L;
27CE  E6  A:&7F;
27D0  03  BC:+1;             BC:+4,处理高一次的10的幂
27D1  03  BC:+1;
27D2  03  BC:+1;
27D3  03  BC:+1;
27D4  A7  A:&A;
27D5  C2  JNZ27B6;           若阶码中仍有非零位,则循环
27D8  E1  HL:=ST;
27D9  F1  AF:=ST;
27DA  A7  A:&A;              尾数的符号
27DB  FC  JSN265B;           若为负,则浮点数取补
27DE  C9  RET;

MONITOR
>
27DF  #00  00  23  04#   10^1的浮点值
27E3  #00  00  32  07#   10^2
```

```
27E7   #00   10   27   0E#   10^4
27EB   #08   AF   2F   1B#   10^8
27EF   #F2   86   23   36#   10^16
27E3   #6B   71   27   6B#   10^32
  G 2800
27F7   FE   A-30;                (浮点打印子程序的继续)
27F9   CA   JZ27FD;
27FC   DF   JSS3;                打印阶码非零的十位数字
27FD   48   C:=B;
27FE   DF   JSS3;                打印阶码的个位数字
27FF   C9   RET;

MONITOR
>
  G 2800
2800   D5   ST:=DE;              "反汇编程序"入口
2801   E5   ST:=HL;              将上、下限地址存入堆栈
2802   7C   A:=H;
2803   CD   JS0204;              打印当前地址
2806   7D   A:=L;
2807   CD   JS0204;
280A   0E   C:=20;
280C   DF   JSS3;                两个空格
280D   DF   JSS3;
280E   7E   A:=M;
280F   CD   JS0204;              打印I字节
2812   0E   C:=20;
2814   DF   JSS3;                两个空格
2815   DF   JSS3;
2816   7E   A:=M;                恢复I
2817   87   A:+A;
```

```
2818  DA  JK2821;
281B  FA  JN2830;          若I=01……，转至2830
281E  C3  J2960;           若I=00……，转至2960
2821  FA  JN2A70;          若I=11……，转至2A70
2824  C3  J28A3;           若I=10……，转至28A3
2827  00  NULL;
2828  06  B:=06;
282A  11  DE:=0406;  282B  06  B:=04
282D  CD  JS005F;          打印前导字符串
2830  E1  HL:=ST;
2831  CD  JS286F;          打印 KJ 值
2834  E5  ST:=HL;
2835  C3  J284F;           转至284F
2838  06  B:=02;
283A  11  DE:=0306;  283B  06  B:=03
283D  11  DE:=0406;  283E  06  B:=04
2840  11  DE:=0506;  2841  06  B:=05
2843  11  DE:=0606;  2844  06  B:=06
2846  11  DE:=0706;  2847  06  B:=07
2849  11  DE:=0806;  284A  06  B:=08
284C  CD  JS005F;          打印尾字符串
284F  0F  C:=3B;           打印分号
2851  DF  JSS3;
2852  CD  JS0200;          换引
2855  E1  HL:=ST;
2856  23  HL:+1;           寻址寄存器加一
2857  D1  DE:=ST;
2858  D5  ST:=DF;
2859  CD  JS0231;          比较地址，并循环，直至结束
285C  FA  JN2801;
285F  CF  JSS1;            调用监督程序
```

```
MONITOR
>
  G 2800
2860   11   DE:=2867;          "找寄存器名称"子程序入口
2863   83   A:+E;
2864   5F   E:=A;
2865   1A   A:=M[DE];
2866   C9   RET;

MONITOR
>
2867 "BCDEHLMA";
  G 2800
286F   23   HL:+1;             "打印 KJ 值"子程序入口
2870   5E   E:=M;
2871   23   HL:+1;
2872   7E   A:=M;
2873   CD   JS0204;
2876   7B   A:=E;
2877   CD   JS0204;
287A   C9   RET;
287B   00   NULL;

MONITOR
>
287C "HALT"
  G 2800
2880   47   B:=A;              (I=01……)
2881   FE   A-EC;              (EC=2×76)
2883   CA   JZ28FA;            若为"HALT", 转至 28FA
2886   E6   A:&70;             否则指令就是"r1:=r2"
```

```
2888  0F  A:@R;
2889  0F  A:@R;
288A  0F  A:@R;
288B  0F  A:@R;
288C  CD  JS2860;          找 r1 的名称
288F  4F  C:=A;
2890  DF  JSS3;           打印 r1 的名称
2891  0E  C:=3A;
2893  DF  JSS3;           打印":="
2894  0E  C:=3D;
2896  DF  JSS3;
2897  78  A:=B;
2898  E6  A:&0E;
289A  0F  A:@R;
289B  CD  JS2860;          找 r2 的名称
289E  4F  C:=A;
289F  DF  JSS3;           打印 r2 的名称
28A0  C3  J284F;          转回至主循环
28A3  47  B:=A;           (I=10……)指令是"A_opr"
28A4  0E  C:=41;
28A6  DF  JSS3;           打印"A"
28A7  CD  JS2900;         找出操作符并打印
28AA  78  A:=B;
28AB  C3  J2898;          转至 2898,象处理 r2 那样处理 r
28AE  00  NULL;
28AF  00  NULL;
28B0  "NULLA:=M[BC]:=A:"
28C0  "=M[DE]:=A:$DECA:"
28D0  "@LA:@RAK:@LAK:@R"
28E0  "K:#]:=HL:=MM[:+1"
28F0  ":-1HL:+"
```

```
G 2800
28F7  00  NULL;
28F8  00  NULL;
28F9  00  NULL;
28FA  21  HL:=287C;   } "HALT"的存贮单元,将"HALT"作为
28FD  C3  J283E;      }   尾字符串打印出来
2900  CD  JS2905;      "找出操作符并打印出来"子程序入口
2903  DF  JSS3;
2904  C9  RET;
2905  78  A:=B;        "找操作符"的主干部分入口
2906  E6  A:&70;
2908  FE  A-70;
290A  0E  C:=2D;
290C  C8  RZ;         找到"-",返回
290D  0E  C:=3A;
290F  DF  JSS3;       打印":"
2910  78  A:=B;
2911  F6  A:&70;
2913  0F  A:@R;       用掩码取出"操作符"并右移
2914  0F  A:@R;
2915  0F  A:@R;
2916  0F  A:@R;
2917  0E  C:=2B;
2919  C8  RZ;         找到"+",返回
291A  3D  A:-1;
291B  C2  JNZ2920;    若非"++",转至2920
291E  DF  JSS3;   }   若为"++",则打印"+",并带着"+"
291F  C9  RET;    }     码返回
2920  0E  C:=2D;
2922  3D  A:-1;
2923  C8  RZ;         找到"-",返回
```

```
2924   3D   A:-1;
2925   C2   JNZ292A;          若非"--",转至292A
2928   DF   JSS3;             若为"--",打印"-",并带着"-"码
                              返回
2929   C9   RET;
292A   0E   C:=26;
292C   3D   A:-1;
292D   C8   RZ;               找到"&",返回
292E   0E   C:=23;
2930   3D   A:-1;
2931   C8   RZ;               找到"#",返回
2932   0E   C:=55;
2934   C9   RET;              找到"U",返回
2935   00   NULL;

MONITOR
>
  G 2800
2936   11   DE:=5053;         "普通寄存器对"子程序入口
2939   CD   JS2940;          找寄存器对的名称
293C   DF   JSS3;
293D   48   C:=B;             打印寄存器对的名称
293E   DF   JSS3;
293F   C9   RET;
2940   01   BC:=4342;         "找寄存器对的名称"子程序入口
2943   E6   A:&30;
2945   C8   RZ;               找到"BC",返回
2946   01   BC:=4544;
2949   FE   A-10;
294B   C8   RZ;               找到"DE",返回
294C   01   BC:=4C43;
```

```
294F   FE   A-20;
2951   C8   RZ;                  找到"HL",返回
2952   42   B:=D;
2953   4B   C:=E;
2954   C9   RET;                 找到"SP"或"AF",返回
2955   78   A:=B;                "找堆栈操作的寄存器对名称"子程序
                                   入口
2956   11   DE:=4641;            用"AF"代替"SP"
2959   C3   J2939;

MONITOR
>
295C "ZKPN"
  G 2800
2960   0F   A:@R;                (I=00……)
2961   47   B:=A;
2962   5F   E:=A;
2963   21   HL:=28B0;
2966   A7   A:&A;
2967   CA   JZ283E;              若为十六进制 00,打印"NULL"
296A   2E   L:=B7;
296C   FE   A-02;
296E   CA   JZ284A;              若号 02, 打印"A:=M[BC]"
2971   2E   L:=B4;
2973   FE   A-0A;
2975   CA   JZ284A;              若号 0A,打印"M[BC]:=A"
2978   2E   L:=C1;
297A   FE   A-12;              若号 12,打印"A:=M[DE]"
297C   CA   JZ284A;
297F   2E   L:=BE;
2981   FE   A-1A;
```

```
2983  CA  JZ284A;          若号 IA,打印"M[DE]:=A"
2986  2E  L:=C8;
2988  FE  A-2F;
298A  CA  JZ283B;          若号 2F, 打印"A:#"
298D  2E  L:=CB;
298F  FE  A-27;
2991  CA  JZ283B;          若号 27, 打印"DEC"
2994  2E  L:=CE;
2996  FE  A-07;
2998  CA  JZ283E;          若号 07,打印"A:@L"
299B  2E  L:=D2;
299D  FE  A-0F;
299F  CA  JZ283E;          若号 0F,打印"A:@R"
29A2  2E  L:=D6;
29A4  FE  A-17;
29A6  CA  JZ2841;          若号 17,打印"AK:@L"
29A9  2E  L:=DB;
29AB  FE  A-1F;
29AD  CA  JZ2841;          若号 1F,打印"AK:@R"
29B0  2E  L:=E0;
29B2  FE  A-3F;
29B4  CA  JZ283B;          若号 3F,打印"K:#"
29B7  FE  A-32;
29B9  C2  JNZ29CA;         若非 32,转至 29CA
29BC  0F  C:=4D;           (I=32),打印"M"
29BE  DF  JSS3;
29BF  E1  HL:=ST;
29C0  CD  JS286F;          打印 KJ 值
29C3  E5  ST:=HL;
29C4  21  HL:=28BC;  }     ":=A"的存储单元, 将":=A"作为尾
29C7  C3  J283B;     }       字符串打印出来
```

```
MONITOR
>
  G 2800
29CA  2E   L:=B4;          (I=00……的继续)
29CC  FE   A-3A;
29CE  CA   JZ282B;         若为 3A,前导字符串为"A:=M"
29D1  FE   A-22;
29D3  C2   JNZ29E8;        若非 22,转至 29E8
29D6  2E   L:=EA;          (I=22)
29D8  06   B:=02;
29DA  CD   JS005F;         打印"MM"
29DD  E1   HL:=ST;
29DE  CD   JS286F;         打印 KJ 值
29E1  E5   ST:=HL;
29E2  21   HL:=28E4;  }    ":=HL" 的存储单元,将 ":=HL" 作
29E5  C3   J283E;     }      为尾字符串打印出来
29E8  FE   A-2A;
29EA  C2   JNZ2B72;        若非 2A,转至 2B72
29ED  2E   L:=E6;
29EF  06   B:=06;
29F1  C3   J282D;          若为 2A,前导字符串为"HL:=MM"
29F4  E6   A:&04;
29F6  CA   JZ2A21;         若为寄存器对指令,转至 2A21
29F9  78   A:=B;           (I=00…1…)
29FA  E6   A:&38;
29FC  0F   A:@R;           用掩码取出"1"并右移
29FD  0F   A:@R;
29FE  0F   A:@R;
29FF  CD   JS2860;         找寄存器的名称
2A02  4F   C:=A;
2A03  DF   JSS3;           打印寄存器的名称
```

```
2A04  78   A:=B;
2A05  0F   A:@R;
2A06  2E   L:=F0;
2A08  DA   JK283B;        若为I=00…101,尾字符串就是":-1"
2A0B  0F   A:@R;
2A0C  2E   L:=ED;
2A0E  D2   JNK283B;       若为I=00…100,尾字符串就是":+1"
2A11  0E   C:=3A;         (I=00…110)
2A13  DF   JSS3;          打印":="
2A14  0E   C:=3D;
2A16  DF   JSS3;
2A17  E1   HL:=ST;        取J字节并打印
2A18  23   HL:+1;
2A19  7E   A:=M;
2A1A  E5   ST:=HL;
2A1B  CD   JS0204;
2A1E  C3   J284F;         转回主循环

MONITOR
>
  G 2800
2A21  68   L:=B;          (I=00…0.1)
2A22  78   A:=B;
2A23  E6   A:&0F;
2A25  FE   A-09;
2A27  C2   JNZ2A38;       若非双字长加法,转至2A38
2A2A  2E   L:=F3;         (I=00…1001)
2A2C  06   B:=04;
2A2E  CD   JS005F;        打印"HL:+"
2A31  7B   A:=E;
2A32  CD   JS2936;        普通寄存器对
```

```
2A35  C3  J284F;          转回主循环
2A38  7D  A:=L;
2A39  CD  JS2936;         普通寄存器对
2A3C  7D  A:=L;
2A3D  E6  A:&0F;
2A3F  2E  L:=ED;
2A41  FE  A-03;           若为I=00…0011，尾字符串就是
2A43  CA  JZ283B;           ":+1"
2A46  2E  L:=F0;
2A48  FE  A-0B;           若为I=00…1011，尾字符串就是
2A4A  CA  JZ283B;           ":-1"
2A4D  0E  C:=3A;
2A4F  DF  JSS3;
2A50  0E  C:=3D;          打印":="
2A52  DF  JSS3;
2A53  C3  J2830;          转回，打印KJ值
2A56  78  A:=B;           "打印条件名称"子程序入口
2A57  E6  A:&08;
2A59  C2  JNZ2A5F;
2A5C  0E  C:=4E;
2A5E  DF  JSS3;           打印"N"表示"N…"条件
2A5F  78  A:=B;
2A60  E6  A:&30;
2A62  0F  A:@R;
2A63  0F  A:@R;
2A64  0F  A:@R;
2A65  0F  A:@R;
2A66  11  DE:=295C;       "ZKPN"的单元
2A69  83  A:+E;
2A6A  5F  E:=A;
2A6B  1A  A:=M[DE];
```

```
2A6C  4F   C:=A;
2A6D  DF   JSS3;                打印"Z", "K", "P"或"N"
2A6E  C9   RET;
2A6F  00   NULL;

MONITOR
>
  G 2800
2A70  1F   AK:@R;               (I=11……)
2A71  47   B:=A;
2A72  21   HL:=2B40;
2A75  FE   A-F9;
2A77  CA   JZ2844;              若为十六进制 F9, 则打印"SP:=HL"
2A7A  2E   L:=44;
2A7C  FE   A-EB;
2A7E  CA   JZ2847;              若为 EB,则打印"HL:=:DE"
2A81  2E   L:=4B;
2A83  FE   A-E3;
2A85  CA   JZ2847;              若为 E3,则打印"HL:=:ST"
2A88  2E   L:=56;
2A8A  FE   A-FB;
2A8C  CA   JZ283E;              若为 FB,则打印"I:=1"
2A8F  2E   L:=5A;
2A91  FE   A-F3;
2A93  CA   JZ283E;              若为 F3,则打印"I:=0"
2A96  2E   L:=5E;
2A98  FE   A-E9;
2A9A  CA   JZ2841;              若为 F9,则打印"J[HL]"
2A9D  2E   L:=63;
2A9F  FE   A-C9;
2AA1  CA   JZ283B;              若为 C9,则打印"RET"
```

```
2AA4  2E   L:=68;
2AA6  FE   A-DB;
2AA8  C2   JNZ2AB3;        若非 DB,转至 2AB3
2AAB  06   B:=04;          (I=十六进制 DB)
2AAD  CD   JS005F;         打印"A:=G"
2AB0  C3   J2A17;          转至 2A17,打印 J 字节
2AB3  FE   A-D3;
2AB5  C2   JNZ2AC8;        若非 D3,转至 2AC8
2AB8  0F   C:=47;          (I=十六进制 D3)
2ABA  DF   JSS3;           打印"G"
2ABB  E1   HL:=ST;
2ABC  23   HL:+1;          寻址寄存器加一
2ABD  7E   A:=M;           取 J 字节
2ABE  E5   ST:=HL;
2ABF  CD   JS0204;         打印 J 值
2AC2  21   HL:=2B66;       ":=A"的存贮单元, 将":=A"作为尾
2AC5  C3   J283B;            字符串打印出来
2AC8  0E   C:=4A;
2ACA  FE   A-C3;
2ACC  CA   JZ2AD7;         若为 C3,则打印"J"并且
2ACF  FE   A-CD;
2AD1  C2   JNZ2ADB;
2AD4  DF   JSS3;           若为 CD,打印"DS",然后
2AD5  0E   C:=53;
2AD7  DF   JSS3;
2AD8  C3   J2830;          转至 2830,打印 KJ 值

MONITOR
>
  G 2800
2ADB  E6   A:&07;
```

```
2ADD  C2   JNZ2AE9;
2AE0  0E   C:=52;          (条件返回)
2AE2  DF   JSS3;           打印"R"
2AE3  CD   JS2A56;         打印条件名称
2AE6  C3   J284F;          转回至循环
2AE9  3D   A:-1;
2AEA  C2   JNZ2AF6;
2AED  CD   JS2955;         ("出栈"操作)找堆栈操作的寄存器对
                             名称
2AF0  21   HL:=2B52;       ":=ST"的存贮单元，将":=ST"作为
2AF3  C3   J283E;            尾字符串打印出去
2AF6  3D   A:-1;
2AF7  C2   JNZ2B03;
2AFA  0E   C:=4A;          (条件转移)
2AFC  DF   JSS3;           打印"J"
2AFD  CD   JS2A56;         打印条件标志
2B00  C3   J2830;          转至2830，打印KJ值
2B03  3D   A:-1;
2B04  3D   A:-1;
2B05  C2   JNZ2B10;
2B08  0E   C:=4A;          (条件子程序调用)
2B0A  DF   JSS3;
2B0B  0E   C:=53;          打印"JS"
2B0D  C3   J2AFC;          其后的过程与条件转移指令一样
2B10  3D   A:-1;
2B11  C2   JNZ2B23;
2B14  2E   L:=50;          ("入栈"操作)
2B16  C5   ST:=BC;
2B17  06   B:=04;
2B19  CD   JS005F;         打印"ST:="
2B1C  C1   BC:=ST;
```

```
2B1D  CD  JS2955;          找堆栈操作的寄存器对名称
2B20  C3  J284F;           转回主循环
2B23  3D  A:-1;
2B24  C2  JNZ2B33;
2B27  0E  C:=41;           (A_opJ 型指令)
2B29  DF  JSS3;            打印"A"
2B2A  78  A:=B;
2B2B  87  A:+A;
2B2C  47  B:=A;
2B2D  CD  JS2900;          找出操作并打印
2B30  C3  J2A17;           转至 2A17,处理 J 字节
2B33  0E  C:=4A;           (专用子程序调用)
2B35  DF  JSS3;            打印"J"
2B36  0E  C:=53;
2B38  DF  JSS3;            打印"SS"
2B39  DF  JSS3;
2B3A  78  A:=B;
2B3B  0F  A:@R;
2B3C  0F  A:@R;            取出专用子程序编号
2B3D  C3  J2B6C;
```

```
MONITOR
>
2B40  "SP:=HL:=:DEHL:=:"
2B50  "ST:=STI:=1I:=0J["
2B60  "HLJRET:=A:=G"
  G 2800
2B6C  0F  A:@R;            (由 2B3D 转至此)
2B6D  E6  A:&37;           将编号变成 ASCII 形式
2B6F  C3  J289E;           转至 289E,打印并重入主循环
2B72  FE  A-37;            (由 29EA 转至此)
```

```
2B74   C2   JNZ29F4;          若非 37,转至 29F4
2B77   0E   C:=4B;           (I=十六进制 37)
2B79   DF   JSS3;            打印"K"
2B7A   21   HL:=2B57;
2B7D   C3   J283B;           尾字符串是":=1"

MONITOR
>
  G 2800
2B80   7B   A:=E;            "浮点打印(DEHL)"子程序入口
2B81   0F   C:=2B;
2B83   A7   A:&A;            尾数的符号
2B84   F2   JNN2B8C;
2B87   CD   JS265B;          浮点数取外
2B8A   0F   C:=2D;
2B8C   DF   JSS3;            打印符号
2B8D   97   A:-A;            将十进制阶码置成 0
2B8E   F5   ST:=AF;
2B8F   7B   A:=E;
2B90   A7   A:&A;
2B91   CA   JZ2BDE;          若尾数为零,转至 2BDE
2B94   7B   A:=E;
2B95   D6   A:-2B;
2B97   7A   A:=D;
2B98   DE   A:--04;
2B9A   FA   JN2BA9;          若数值小于 10,转至 2BA9
2B9D   01   BC:=27DF;        "10"的存贮单元
2BA0   CD   JS26EC;          浮点除法
2BA3   F1   AF:=ST;
2BA4   3C   A:+1;            十进制阶码加一
2BA5   F5   ST:=AF;
```

```
2BA6  C3   J2B94;          循环至 2B94
2BA9  7B   A:=E;
2BAA  D6   A:-20;
2BAC  7A   A:=D;
2BAD  DE   A:--01;
2BAF  F2   JNN2BBE;        若数值不小于1,转至 2BBE
2BB2  01   BC:=27DF;       "10"的存贮单元
2BB5  CD   JS264B;         浮点乘法
2BB8  F1   AF:=ST;
2BB9  3D   A:-1;           从十进制阶码中减一
2BBA  F5   ST:=AF;
2BBB  C3   J2BA9;          循环至 2BA9

MONITOR
>
  G 2800
2BBE  7A   A:=D;           (浮点打印子程序的继续)
2BBF  3C   A:+1;
2BC0  16   D:=00;
2BC2  EB   HL:=:DE;        DEHL 左移
2BC3  29   HL:+HL;
2BC4  EB   HL:=:DE;
2BC5  29   HL:+HL;
2BC6  D2   JNK2BCA;
2BC9  13   DE:+1;
2BCA  3D   A:-1;           A 减计数
2BCB  F2   JNN2BC2;        循环至2BC2
2BCE  01   BC:=0054;       十进制四舍五入(十六进制 00.000054
                           =1/2×10^-5)
2BD1  09   HL:+BC;
```

```
2BD2  D2   JNK2BD6;
2BD5  13   DE:+1;
2BD6  4D   C:=L;
2BD7  09   HL:+BC;
2BD8  D2   JNK2BDC;
2BDB  13   DE:+1;
2BDC  6C   L:=H;
2BDD  63   H:=E;          HL 中是尾数的分数部分
2BDE  7A   A:=D;          A 是尾数的整数部分
2BDF  FE   A-0A;
2BE1  CA   JZ2BE9;
2BE4  F7   JSS6;          "Hex.-ASCII"转换
2BE5  4F   C:=A;
2BE6  C3   J2BED;
2BE9  0F   C:=31;         (由于四舍五入的结果，尾数的整数部
                            分可以是 10)
2BEB  DF   JSS3;
2BEC  0D   C:-1;
2BED  DF   JSS3;          打印尾数的整数部分
2BEE  CD   JS07E6;        打印尾数的分数部分
2BF1  0E   C:=45;
2BF3  DF   JSS3;          打印"E"
2BF4  F1   AF:=ST;
2BF5  F2   JNN262C;       若阶码为正，跳至 262C
2BF8  F5   ST:=AF;
2BF9  0E   C:=2D;   ┐
2BFB  DF   JSS3;    ├     打印"-"
2BFC  F1   AF:=ST;  ┘
2BFD  C3   J262A;         转至 262A 继续执行

MONITOR
>
```

§8.9 软件的十六进制文本

表 8.1

```
D0000,01FF
0000 3E CF D3 FB 3E 27 D3 FB 22 FA 17 F5 E1 C3 67 00
0010 41 3E 1B B8 C3 50 00 00 C3 45 00 0E 0A DF 48 C9
0020 FF 4F D7 79 C9 00 00 00 D6 30 FE 0A F8 D6 07 C9
0030 C6 30 FE 3A F8 C6 07 C9 DB FB E6 02 CA 38 00 DB
0040 FA E6 7F 4F C9 DB FB E6 01 CA 45 00 79 D3 FA C9
0050 C2 55 00 0E 24 DF 3F 0D B8 CA 1B 00 48 C9 00 4E
0060 DF 23 05 C2 5F 00 C9 22 F8 17 E1 22 FC 17 21 00
0070 00 39 22 FE 17 60 69 22 F6 17 EB 22 F4 17 21 BA
0080 00 31 F4 17 CD F6 03 31 F4 17 CD 00 02 GE 3E DF
0090 E7 FE 44 CA 33 01 FE 47 CA C4 00 FE 49 CA E3 00
00A0 FE 4D CA 80 01 FE 52 CA CA 00 FE 53 CA 06 01 FE
00B0 58 CA 15 02 0E 3F DF C3 87 00 0D 0A 4D 4F 4E 49
00C0 54 4F 52 1B CD F7 01 22 FC 17 F3 CD 00 02 31 F4
00D0 17 D1 C1 F1 2A FE 17 F9 2A FC 17 E5 2A FA 17 FB
00E0 C9 00 00 CD F7 01 CD 00 02 E7 FE 20 CA E9 00 FE
00F0 2C CA E9 00 FE 0D CA E9 00 FE 1B CA 87 00 CD 21
0100 02 77 23 C3 E9 00 CD F7 01 E7 FE 20 CA 14 01 FE
0110 2C C2 87 00 7E CD 04 02 0E 2D DF E7 47 EF FE 00
0120 FA 2E 01 FE 10 F2 2E 01 CD 25 02 77 E7 47 76 23
0130 C3 0A 01 CD F7 01 E5 E7 FE 20 CA 42 01 FE 2C C2
0140 B4 00 CD F7 01 D1 CD 31 02 FA B4 00 EB CD 00 02
0150 7C CD 04 02 7D CD 04 02 0E 20 DF 7E CD 04 02 DB
0160 FB E6 02 CA 6F 01 DB FA E6 7F FE 1B CA 87 00 CD
0170 31 02 CA 87 00 23 7D E6 0F C2 58 01 C3 4D C1 09
0180 CD F7 01 E5 E7 FE 20 CA 8F 01 FE 2C C2 B4 00 CD
0190 F7 01 D1 CD 31 02 FA B4 00 E5 D5 E7 FE 20 CA A6
01A0 01 FE 2C C2 B4 00 CD F7 01 E5 D1 E1 CD 31 02 FA
01B0 C4 01 42 4B D1 7E 02 C5 CD 31 02 C1 CA 87 00 03
01C0 23 C3 B5 01 EB 44 4D 7A 2F 67 7B 2F 6F 23 09 EB
01D0 E3 EB 19 44 4D E1 EB 7E 02 C5 CD 31 02 C1 CA 87
01E0 00 0B 2B C3 D7 01 FE 00 FA F1 01 FE 10 F2 F1 01
01F0 C9 C3 B4 00 C3 10 00 CD 20 02 67 CD 20 02 6F C9
►

D0200,03FF
0200 0E 0D D7 C9 47 OF OF OF OF E6 OF F7 4F DF 78 E6
0210 OF F7 4F DF C9 CD 00 02 C3 45 02 C3 38 00 00 00
0220 E7 EF CD E6 01 67 87 87 57 E7 EF CD E6 01 82
0230 C9 44 4A 78 B9 C2 3D 02 45 4B 78 B9 C8 A9 FA 5A
0240 03 78 91 C9 00 06 28 21 EA 02 CD 5F 00 3A F8 17
0250 A7 CD 13 03 CD 13 03 CD 12 03 CD 12 03 CD 12 03
0260 0E 20 DF DF 3A F9 17 CD 04 02 21 F6 17 CD 2D 03
0270 21 F4 17 CD 2D 03 21 FA 17 CD 2D 03 21 FE 17 CD
0280 2D 03 21 FC 17 CD 2D 03 CD 00 02 E7 FE 20 CA B1
0290 02 FE 0D CA 87 00 CD 24 03 CD 22 03 CD 1F 03 CD
02A0 1F 03 82 82 87 3C CD 22 03 82 82 32 F8 17 C3 B5
02B0 02 DF DF DF DF 0E 20 DF DF E7 FE 20 CA C8 02 CD
02C0 21 02 32 F9 17 C3 C9 02 DF 21 F6 17 CD 3C C3 21
02D0 F4 17 CD 3C 03 21 FA 17 CD 3C 03 21 FE 17 CD 3C
02E0 03 21 FC 17 CD 3C 03 C3 87 00 4E 5A 48 50 4B 20
02F0 20 41 20 20 20 20 42 43 20 20 20 20 44 45 20 20
0300 20 20 48 4C 20 20 20 20 53 50 20 20 20 20 50 43
0310 0D 0A 87 47 3E 30 F2 1A 03 3C 4F DF 78 87 C9 82
0320 82 87 57 E7 D6 30 C8 FE 01 C8 C3 B4 00 0E 20 DF
0330 DF 23 7E CD 04 02 2B 7E CD 04 02 C9 0E 20 DF DF
0340 E7 FE 20 CA 56 03 FE 0D CA 87 00 CD 21 02 23 77
0350 CD 20 02 2B 77 C9 DF DF DF C9 79 F6 7F C9 00 00
0360 CD D1 03 DB F4 3C C2 B6 03 CD E4 03 CA 63 03 3E
0370 88 D3 F7 3E 01 D3 F5 21 00 10 E5 06 C8 3E 04 CD
0380 DD 03 E1 F5 7E D3 F4 00 00 3E 02 D3 F6 3E 35 3D
0390 C2 8F 03 D3 F6 CD E3 03 CA 84 03 05 C2 7D 03 3E
03A0 98 D3 F7 E1 DB F4 BE C2 BB 03 CD E3 03 C2 A4 03
03B0 76 00 00 00 00 00 3E 02 D3 F5 76 3E 04 D3 F5 CF
03C0 00 CD D1 03 21 00 10 DB F4 77 CD E3 03 CA C7 03
03D0 CF 3E 98 D3 F7 3E 04 CD DD 03 C9 3E 01 D3 F6 97
03E0 D3 F6 C9 23 CD DB 03 DB F6 E6 10 C9 FF 77 FE 1B
03F0 C8 DF 23 C3 EC 03 4E 79 FE 1B C8 DF 23 C3 F6 03
►
```

表 8.2

```
D0400,05FF
0400 21 80 11 22 10 10 21 00 10 16 0E CD 1B 02 FE 20
0410 CA 0B 04 FE 0A CA 0B 04 FE 7F CA 31 04 FE 3B CA
0420 3E 04 FE 0D CA 3E 04 77 23 4F CD F4 01 15 C2 0B
0430 04 0E 3F CD F4 01 0E 0D CD F4 01 C3 06 04 36 FF
0440 0E 3B CD F4 01 7A C6 05 57 0E 20 CD F4 01 15 C2
0450 49 04 21 00 10 C3 20 0C C5 C3 86 04 00 00 00 00
0460 CA 75 04 3E 00 47 5F CD 58 04 78 CD 58 04 7B C3
0470 7A 04 CD 58 04 78 C3 7A 04 81 CD 58 04 23 7E 3C
0480 C2 31 04 C3 36 04 E5 2A 10 10 77 23 22 10 10 E1
0490 CD 04 02 0E 20 CD F4 01 C1 C9 00 3E 3C C3 7A 04
04A0 7E FE 41 C2 80 05 23 7E FE 3A CA D1 09 C3 B8 09
04B0 79 FE FF CA 63 04 FE 08 CA F1 04 59 CD 1B 0B 47
04C0 79 FE FF CA 63 04 FE 08 CA E8 04 FE 09 CA D9 04
04D0 7B 87 87 87 C6 80 C3 79 04 7B FE 00 CA 9B 04 FE
04E0 02 06 3D C3 60 04 00 00 7B 87 87 87 C6 C6 C3 72
04F0 04 CD 1B 0B 47 FE 47 CA 28 05 79 FE 08 C2 05 05
0500 3E 3E C3 72 04 FE FF CA 0F 05 3E 78 C3 79 04 23
0510 7E FE FF C2 1A 05 2B C3 0A 05 FE 5B CA 32 05 CD
0520 80 0B 47 3E 3A C3 67 04 23 CD 60 0B 47 3E DB C3
0530 72 04 23 7E FE 42 CA 4E 05 FE 44 C2 63 04 23 7E
0540 FE 45 C2 63 04 06 1A 23 7E FE 5D C3 60 04 23 7E
0550 FE 43 C2 63 04 06 0A C3 47 05 23 7E FE 3A C2 63
0560 04 23 7E FE 3D C2 63 04 23 7E FE 53 C2 63 04 23
0570 7E FE 54 06 F1 C3 60 04 FE 43 06 27 C3 60 04 00
0580 FE 42 C2 05 06 23 7E FE 43 CA CE 05 1E 40 FE 3A
0590 C2 63 04 23 7E FE 2B CA BD 05 FE 2D CA C3 05 FE
05A0 3D C2 63 04 CD 1B 0B 47 79 FE 08 C2 B4 05 7B C6
05B0 C6 C3 72 04 E6 F8 C2 63 04 7B C3 79 04 7B C6 C4
05C0 C3 C6 05 7B C6 C5 47 23 7E FE 31 C3 60 04 23 7E
05D0 FE 3A C2 63 04 0E 00 23 7E FE 2B CA 60 09 FE 2D
05E0 CA 66 09 FE 3D C2 63 04 23 7E FE 53 CA F8 05 0C
05F0 C3 64 07 79 3C C3 67 04 23 7E FE 54 C2 63 04 79
>

D0600,07FF
0600 C6 C1 C3 7A 04 FE 43 C2 11 06 23 7E 1E 48 C3 8E
0610 05 FE 44 C2 2E 06 23 7E FE 45 CA 22 06 1E 50 C3
0620 8E 05 23 7E FE 3A C2 78 05 0E 10 C3 D7 05 FE 47
0630 C2 4F 06 23 CD 60 0B 47 23 7E FE 3A C2 63 04 23
0640 7E FE 3D C2 63 04 23 7E FE 41 3E D3 CA 72 04 FE
0650 48 C2 20 07 23 7E FE 3A C2 60 06 1E 60 C3 93 05
0660 FE 4C C2 87 09 23 7E FE 3A C2 63 04 23 7E FE 2D
0670 CA BC 06 FE 2B C2 C5 06 23 7E FE 31 C2 84 06 3E
0680 23 C3 7A 04 FE 42 C2 92 06 23 7E FE 43 06 09 C3
0690 60 04 FE 44 C2 A0 06 23 7E FE 45 06 19 C3 60 04
06A0 FE 48 C2 AF 06 23 7E FE 4C 06 29 C3 60 04 FE 53
06B0 C2 63 04 23 7E FE 50 06 39 C3 60 04 23 7E FE 31
06C0 06 2B C3 60 04 FE 3D C2 63 04 23 7E FE 3A C2 EF
06D0 06 23 7E FE 53 C2 E1 06 23 7E FE 54 06 E3 C3 60
06E0 04 FE 44 C2 63 04 23 7E FE 45 06 EB C3 60 04 FE
06F0 53 C2 00 07 23 7E FE 54 06 E1 C3 60 04 00 00 00
0700 FE 4D C2 16 07 23 7E FE 4D C2 63 04 23 CD 80 0B
0710 47 3E 2A C3 67 04 CD 80 0B 47 3E 21 C3 67 04 00
0720 FE 49 C2 46 07 23 7E FE 3A C2 63 04 23 7E FE 3D
0730 C2 63 04 23 7E FE 30 C2 3F 07 3E F3 C3 7A 04 FE
0740 31 06 FB C3 60 04 FE 4A C2 C2 07 23 7E FE 53 C2
0750 8B 07 23 7E CD C0 0B 79 FE FF CA 6C 07 87 87 87
0760 C6 C4 4F 23 C5 CD 80 0B C1 C3 DE 07 7E FE 53 C2
0770 83 07 23 7E D6 30 FE 08 F2 63 04 87 87 87 C6 C7
0780 C3 7A 04 0E CD C3 64 07 00 00 00 CD C0 0B 79 FE
0790 FF CA 9C 07 87 87 87 C6 C2 C3 62 07 7E FE 5B CA
07A0 AB 07 CD 80 0B 47 3E C3 C3 67 04 23 7E FE 48 C2
07B0 63 04 23 7E FE 4C C2 63 04 23 7E FE 5D 06 E9 C3
07C0 60 04 FE 4B C2 9E 0D 23 7E FE 3A C2 63 04 23 7E
07D0 FE 3D C2 80 09 23 7E FE 31 06 37 C3 60 04 47 79
07E0 C3 67 04 00 00 00 00 06 05 DE 2E DF 97 29 8F 4F 54
07F0 5D 29 8F 29 8F 19 89 C6 30 4F DF 05 C2 EB D7 C9
>
```

表 8.3

```
D0800,09FF
0800 3E 70 C3 93 05 FE 5B C2 47 08 23 7E FE 42 C2 1D
0810 08 23 7E FE 43 C2 63 04 06 02 C3 2B 08 FE 44 C2
0820 63 04 23 7E FE 45 C2 63 04 06 12 23 7E FE 5D C2
0830 63 04 23 7E FE 3A C2 63 04 23 7E FE 3D C2 63 04
0840 23 7E FE 41 C3 60 04 FE 4D C2 72 08 23 CD 80 0B
0850 47 23 7E FE 3A C2 63 04 23 7E FE 3D C2 63 04 23
0860 7E FE 48 C2 63 04 23 7E FE 4C C2 63 04 3E 22 C3
0870 67 04 CD 80 0B 47 23 7E FE 3A C2 63 04 23 7E FE
0880 3D C2 63 04 23 7E FE 41 C2 63 04 3E 32 C3 67 04
0890 FE 52 C2 C2 08 23 7E FE FF C2 A2 08 3E C9 2B C3
08A0 7A 04 CD C0 0B 79 FE FF CA B3 08 87 87 87 C6 C0
08B0 C3 7A 04 7E FE 45 C2 63 04 23 7E FE 54 06 C9 C3
08C0 60 04 FE 53 C2 71 09 23 7E FE 50 C2 0D 09 23 7E
08D0 FE 3A C2 63 04 23 7E FE 3D C2 F5 08 23 7E FE 48
08E0 C2 EC 08 23 7E FE 4C 06 F9 C3 60 04 CD 80 0B 47
08F0 3E 31 C3 67 04 FE 2B C2 FF 08 06 33 C3 06 09 FE
0900 2D C2 63 04 06 3B 23 7E FE 31 C3 60 04 FE 54 C2
0910 63 04 23 7E FE 3A C2 63 04 23 7E FE 3D C2 63 04
0920 23 7E FE 42 C2 30 09 23 7E FE 43 06 C5 C3 60 04
0930 FE 44 C2 3E 09 23 7E FE 45 06 D5 C3 60 04 FE 48
0940 C2 4C 09 23 7E FE 4C 06 E5 C3 60 04 FE 41 C2 63
0950 04 23 7E FE 46 06 F5 C3 60 04 D1 C3 BB 0A 00 00
0960 79 C6 03 C3 69 09 79 C6 0B 47 23 7E FE 31 C3 60
0970 04 FE 45 C2 9C 09 23 7E 1E 58 C3 8E 05 00 00 00
0980 FE 23 06 3F C3 60 04 FE 41 C2 63 04 23 7E FE 4C
0990 C2 63 04 23 7E FE 54 06 76 C3 60 04 FE 4E C2 F8
09A0 0C 23 7E FE 55 C2 63 04 23 7E FE 4C C2 63 04 23
09B0 7E FE 4C 06 00 C3 60 04 FE 46 CA 5A 05 FE 4B C2
09C0 E5 09 23 7E FE 3A C2 63 04 23 7E 01 1F 17 C3 EE
09D0 09 23 7E FE 23 C2 EB 09 23 7E 2B 3C C2 E4 09 3E
09E0 2F C3 7A 04 2B CD 20 0B C3 B0 04 01 0F 07 FE 40
09F0 C2 E4 09 23 7E FE 4C CA 75 04 FE 52 41 C3 60 04
>

D0A00,0BFF
0A00 CD 60 0B 5F 23 CD 60 0B C9 FE 00 FA 12 0A CD 00
0A10 0A C9 79 FE 28 C2 63 04 23 7E FE 4C C2 56 0A 23
0A20 CD 60 0B E5 26 00 6F 4F 29 EB 21 40 10 19 5E 23
0A30 56 7B FE 00 C2 3D 0A 7A FE 00 CA 40 0A C3 B3 0A
0A40 2A 10 10 23 EB 2A 12 10 71 23 73 23 72 23 22 12
0A50 10 16 FD C3 B3 0A FE 58 C2 C0 0A 23 7E 4F CD 00
0A60 0B 78 FE 00 FA 63 04 E5 CA 85 0A 21 1E 10 5F 16
0A70 00 19 19 4E 23 46 60 69 EB E1 D5 23 7E FE 29 C2
0A80 8B 0A C3 5A 09 21 00 10 C3 78 0A FE 2B C2 95 0A
0A90 0E 00 C3 9C 0A FE 2D C2 63 04 0E 01 23 CD 60 0B
0AA0 E3 5F 16 00 79 FE 00 CA B1 0A 7A 2F 57 7B 2F 5F
0AB0 13 19 EB E1 23 7E FE 29 C2 63 04 7B 5A C9 00 00
0AC0 FE 49 C2 63 04 23 7E FE 2B C2 D1 0A 0E 00 C3 D8
0AD0 0A FE 2D C2 63 04 0E 01 23 CD 60 0B E5 5F 16 00
0AE0 79 FE 00 CA ED 0A 7A 2F 57 7B 2F 5F 13 2A 3E 10
0AF0 19 EB 2A 10 10 19 23 23 23 11 80 EE 19 C3 B2 0A
0B00 D6 30 FA 14 0B FE 0A FA 19 0B D6 11 FA 14 0B FE
0B10 06 FA 17 0B 06 FF C9 C6 0A 47 C9 23 23 C3 89 0B
0B20 0E 00 7E FE 3A CA 2F 0B FE 2D CA 52 0B 0D C9 23
0B30 7E FE 2B CA 55 0B 0C 0C FE 2D CA 55 0B 0C 0C FE
0B40 26 C8 0C FE 23 C8 0C FE 55 C8 0C 0C FE 3D C8 0E
0B50 FF C9 0E 07 C9 47 23 7E B8 C2 5E 0B 0C C9 2B C9
0B60 7E CD 00 0B 78 3C CA 7B 0B 50 23 7E CD 00 0B 78
0B70 3C CA 7B 0B 7A 87 87 87 80 C9 0E FF 7E C9 00
0B80 7E 4F CD 00 0B 78 C3 09 0A 7E FE FF 2B C2 B6 0B
0B90 7E 0E 00 D6 42 C8 0C 3D C8 0C 3D C8 0C 3D C8 0C
0BA0 D6 03 C8 0C D6 04 C8 0C 3D C8 0C C6 0C C8 0C 0C
0BB0 C6 10 C8 0E FF C9 0E 08 CD 60 0B C9 00 00 00 00
0BC0 FE 4E C2 B5 0D 0E 07 23 7E FE 47 F2 D0 0B 2B C9
0BD0 0D FE 4E C8 0D 0D FF 50 C8 0D 0D FE 4B C3 97 0D
0BF0 26 30 EB 3E 0E C5 D5 CD 9B 0F D1 C1 19 CD F1 0F
0BF0 CD 1A 0C C8 E5 CD F1 0C 23 7C B5 E1 C2 E2 0B C9
>
```

表 8.4

```
D0C00,0DFF
0C00 06 00 21 20 10 70 23 7D FE C0 C2 05 0C 78 32 14
0C10 10 21 C0 10 22 12 10 C3 00 04 C5 CD 31 02 C1 C9
0C20 3A 14 10 FE 00 C2 8A 0C 7E FE 58 C2 59 0C 23 7E
0C30 CD 00 0B 78 FE 00 FA 63 04 E5 26 00 6F 29 EB 21
0C40 20 10 19 E3 23 7E FE 3A C2 63 04 23 CD 60 0B E3
0C50 77 23 3E 00 77 E1 C3 7D 04 3E 01 32 14 10 E5 2A
0C60 20 10 EB 21 00 10 19 22 20 10 21 20 10 4E 23 46
0C70 23 5E 23 56 EB 09 EB 72 2B 73 7C FE 10 C2 6D 0C
0C80 7D FE 3E C2 6D 0C E1 C3 A0 04 7E FE 58 CA 63 04
0C90 C3 A1 04 CD 60 0B FE 40 F2 63 04 E5 2A 3E 10 ED
0CA0 2A 10 10 19 EB 21 80 EE 19 EB 06 00 4F 21 40 10
0CB0 09 09 73 23 72 E1 23 7E FE 3A C2 63 04 23 C3 A0
0CC0 04 21 C0 10 3A 12 10 BE C2 D2 0C 3A 13 10 BC CA
0CD0 EE 0C 7E E5 26 00 6F 29 EB 21 40 10 19 5E 23 56
0CE0 E1 23 4E 23 46 7B 02 03 7A 02 23 C3 C4 0C CF 00
0CF0 00 7D 93 6F 7C 9A 67 C9 FE 46 C2 00 0D C3 C1 0C
0D00 FE 56 C2 93 0C CD 00 02 CD 66 0D FE 22 C2 1E 0D
0D10 CD 66 0D FE 22 CA 36 04 CD 59 0D C3 10 0D FE 23
0D20 C2 63 04 CD 66 0D CD 00 0B 78 FE 00 FA 63 04 87
0D30 87 87 87 57 CD 66 0D CD 00 0B 78 FE 00 FA 63 04
0D40 32 CD 59 0D CD 66 0D FE 20 CA 44 0D FE 0D CA 44
0D50 0D FE 23 C2 26 0D C3 36 04 C5 E5 2A 10 10 77 23
0D60 22 10 10 E1 C1 C9 CD 1B 02 4F CD F4 01 79 C9 C5
0D70 D5 44 4D 78 E6 C0 C2 B4 00 CD E0 0B 29 D1 C1 C9
0D80 00 00 00 00 00 00 0E 05 FE 50 C8 0D 0D FE 4B C8 0D
0D90 0D FE 5A C8 0D 0D C9 C8 0D 0D C3 C5 0D 00 FE 4C
0DA0 C2 AA 0D 23 7E 1E 68 C3 8E 05 FE 4D C2 90 08 23
0DB0 7E FE 3A C2 05 08 23 23 7E FE 4D CA 63 04 2B 2B
0DC0 1E 70 C3 93 05 FE 5A C8 0D C9 16 2B 7C FE 80 C2
0DD0 D9 0D 7D FE 00 CA 4E 0E 7C FE 00 F2 E6 0D 14 14
0DE0 2F 67 7D 2F 6F 23 01 F0 D8 CD 5D 0E 01 10 27 09
0DF0 7B FE 30 C2 F9 0D 5A 16 20 D5 01 18 FC CD 5D 0E
>

D0E00,0FFF
0E00 01 E8 03 09 53 01 9C FF CD 5D 0E 01 64 00 09 26
0E10 30 7D FE 0A FA 1D 0E D6 0A 24 C3 12 0E C6 30 6F
0E20 C1 78 FE 20 C2 3F 0E 7A FE 30 C2 3F 0E 51 48 7B
0E30 FE 30 C2 3F 0E 5A 51 7C FE 30 C2 3F 0E 63 5A 79
0E40 48 47 DF 48 DF 4A DF 4B DF 4C DF 4D DF C9 21 57
0E50 0E 06 06 CD 5F 00 C9 2D 33 32 37 36 38 1E 30 09
0E60 7C FE 00 F8 1C C3 5F 0E 26 00 6F FE 00 F2 CA 0D
0E70 25 C3 CA 0D 11 00 00 E7 FE 2B CA 83 0E FE 2D C2
0E80 84 0E 14 E7 EF CD E6 01 5F E7 FE 30 FA B4 0E FE
0E90 3A F2 B4 0E D6 30 47 7B FE 0D F2 B4 00 87 4F 87
0EA0 87 81 80 5F FE 00 F2 89 0E 87 C2 B4 00 15 C2 B4
0EB0 00 E7 7B C9 7B 15 F8 2F 3C C9 21 00 00 54 E7 FE
0EC0 2B CA CA 0E FE 2D C2 CB 0E 14 E7 EF CD E6 01 6F
0ED0 E7 FE 30 FA 0D 0F FE 3A F2 0D 0F D6 30 5F 01 33
0EE0 F3 09 7C FE 00 F2 B4 00 01 CD 0C 09 29 44 4D 29
0EF0 29 09 06 00 4B 09 7C FE 00 F2 D0 0E FE 80 C2 B4
0F00 00 7D FE 00 C2 B4 00 15 C2 B4 00 E7 C9 15 F8 7C
0F10 2F 67 7D 2F 6F 23 C9 00 79 FE 00 F0 7C 93 67 C9
0F20 21 00 00 79 54 06 08 29 07 D2 2D 0F 19 05 C2 27
0F30 0F C9 7B FE 00 F0 7C 91 67 C9 CD 20 0F CD 18 0F
0F40 CD 32 0F C9 F5 C5 E5 4A CD 20 0F EB E1 C1 F1 C9
0F50 0C 7A B8 F2 5D 0F 0C 87 F2 52 0F 1F 0D 57 21 00
0F60 00 29 7C FE 00 FA B4 00 78 BA FA 6F 0F 92 23 87
0F70 47 0D C2 61 0F C9 1E 00 78 FE 00 F2 82 0F 2F 3C
0F80 47 1C 7A FE 00 F2 8C 0F 2F 3C 57 1D CD 50 0F 7B
0F90 FE 00 C8 7C 2F 67 7D 2F 6F 23 C9 EB C5 D1 47 04
0FA0 CD 1A 0C F2 AF 0F 04 29 D2 A0 0F CD F2 0F 05 EB
0FB0 E5 21 00 00 29 DA B4 00 E3 CD 1A 0C FA C5 0F CD
0FC0 F1 0C E3 23 E3 29 F3 05 C2 B4 0F C1 C9 F5 26 00
0FD0 78 A7 F2 DC 0F 2F 47 79 2F 4F 03 24 7A A7 F2 E8
0FE0 0F 2F 57 7B 2F 5F 13 25 F1 E5 CD 9B 0F F1 C3 90
0FF0 0F 23 7D E6 FE 0F 6F 7C 1F 67 D0 7D EE 80 6F C9
>
```

表 8.5

```
D2400,25FF
2400 3E 10 F5 97 67 6F EB 29 EB 1F A7 29 D2 10 24 13
2410 F2 18 24 09 D2 18 24 13 E3 25 E3 C2 06 24 F1 C9
2420 21 00 00 78 A7 F2 29 24 19 7A A7 F2 2F 24 09 7C
2430 2F 67 7D 2F 6F 2C E5 CD 00 24 EB C1 09 EB C9 00
2440 CD 20 24 EB 29 EB 29 D2 4B 24 13 29 D2 50 24 13
2450 EB C9 11 00 00 D5 5E 23 56 23 E3 19 A7 CA 6D 24
2460 3D EB F5 C5 CD 40 24 C1 F1 E3 C3 56 24 D1 C9 00
2470 E5 EB 42 4B CD 40 24 44 4D 3E 03 21 88 24 CD 52
2480 24 EB C1 CD 40 24 29 C9 FD FF 89 00 55 F5 00 40
2490 CD 70 24 EB 42 43 CD 40 24 CD CF 24 EB 21 00 40
24A0 CD F1 0C CD 6F 0D C9 7E A7 CA BA 24 23 13 13 1A
24B0 A7 C2 FE 26 F1 C1 CD 19 26 C9 C1 F1 CD 19 26 C9
24C0 00 67 2E 00 29 44 4D 11 7D 47 CD 40 24 29 C9 CD
24D0 F1 0F 7C E6 40 87 84 67 C9 CD 70 24 E5 CD 93 24
24E0 C1 EB 3E 0F CD CD CF C9 0E 20 DF 0E 2B 29 D2 F6
24F0 24 CD 93 0F 0E 2D DF 0E 30 DF 06 04 CD E8 07 C9
2500 44 4D 11 97 72 CD 20 24 EB C9 E5 D5 C5 44 4D CD
2510 00 24 C1 E3 D5 C5 EB 48 79 17 9F 47 CD 56 25 E3
2520 D5 EB 4A CD 3A 0F E5 7B 17 9F 57 21 0A 00 39 F9
2530 C1 21 F4 FF 39 F9 CD 62 25 C1 EB 09 44 4D E1 09
2540 44 4D E1 19 D2 48 25 03 54 5D E1 19 D2 50 25 03
2550 54 5D E1 33 33 C9 21 00 00 78 A7 F2 2F 24 19 C3
2560 2F 24 21 00 00 C3 29 24 F5 CD F2 0F F1 07 1F 1F
2570 F5 7C 17 0F 67 F1 C9 D5 11 01 00 19 CE 00 CD 68
2580 25 D1 C9 CD 68 25 05 C2 83 25 C9 05 CA 95 25 CD
2590 68 25 C3 8B 25 D5 11 01 00 19 CE 00 CD 68 25 D1
25A0 C9 06 00 F5 A7 C2 B2 25 7C A7 C2 B2 25 7D A7 CA
25B0 C7 25 F1 F5 E6 E0 FE 20 CA C9 25 FE C0 CA C9 25
25C0 F1 29 8F D5 C3 A3 25 06 00 F1 C9 4F 07 17 CE 00
25D0 1F 79 D0 04 CD 77 25 C9 C5 47 7D 93 6F 7C 9A 67
25E0 78 99 C1 C9 EB 50 59 01 20 00 09 D2 EF 25 13 EB
25F0 29 EB 29 D2 F7 25 13 EB 29 EB 29 D2 FF 25 13 6C
>

D2600,27FF
2600 63 7A C9 00 00 00 00 00 F5 7D 02 03 7C 02 03 7B
2610 02 03 7A 02 0B 0B 0B F1 C9 F5 0A 6F 03 0A 67 03
2620 0A 5F 03 0A 57 0B 0B 0B F1 C9 2F 3C 0E 30 D6 0A
2630 FA 37 26 0C C3 2E 26 C6 3A 47 79 C3 F7 27 CD 0A
2640 25 CD E4 25 C1 CD A3 25 50 5F C9 D5 E5 CD 19 26
2650 7A 43 D1 E3 84 4D E1 F5 C3 3E 26 EB 44 4D 97 67
2660 6F CD D8 25 CD CB 25 5F 50 C9 D5 42 4B CD 19 26
2670 CD 5B 26 C1 C5 CD 08 26 D1 C9 7A 2F 3C 3C F5 EB
2680 4D 21 00 00 3E 20 06 17 F5 E5 7C 29 8F FA B4 00
2690 E3 F5 33 33 33 33 F1 3B 3B 3B 3B 3B CD D8 25
26A0 F2 B7 26 19 89 29 8F 33 33 33 33 33 F5 3B 3B
26B0 3B 3B F1 E3 C3 CE 26 29 8F 33 33 33 33 33 33 F5
26C0 3B 3B 3B 3B F1 E3 D5 11 01 00 19 CE 00 D1 C5 C2
26D0 8B 26 D1 D1 C1 C3 64 26 00 7B A7 FA E2 25 CD 7A
26E0 26 C9 CD 5B 26 CD 7A 26 CD 5B 26 C9 D5 E5 CD 19
26F0 26 CD D9 26 C3 50 26 E5 D5 23 23 C3 A7 24 13 1A
2700 96 FA 0C 27 47 D1 E1 D5 EB C3 10 27 2F 3C 47 D1
2710 1A 6F 13 1A 67 13 1A C4 8B 25 E3 5E 23 56 23 4E
2720 23 E3 19 89 D1 EB 46 EB CD CB 25 CD A3 25 5F 50
2730 C9 D5 E5 CD 6A 26 E1 CD F7 26 C1 E5 D5 50 59 CD
2740 6A 26 D1 E1 C9 E7 01 C0 00 FE 25 CA 58 27 FE 2D
2750 CA 57 27 4F C3 58 27 05 C5 97 67 6C L5 F5 79 A7
2760 C2 64 27 E7 D6 30 FA 8C 27 FE 0A F2 8C 27 4F F1
2770 E3 29 8F 47 54 5D 29 8F 29 8F 19 88 16 00 59 19
2780 8A E3 F5 7C A7 CA 63 27 2D C3 63 27 FE FE C2 95
2790 27 24 C3 63 27 45 F1 E1 C5 06 16 CD A3 25 50 5F
27A0 D5 E5 CD 74 0E E1 D1 C1 80 01 DF 27 E5 26 00 A7
27B0 F2 B6 27 2F 3C 25 0F D2 D0 27 6F 7C E3 A7 C5 F4
27C0 4B 26 C1 E3 7C E3 A7 C5 FC EC 26 C1 E3 7D E6 7F
27D0 03 03 03 03 A7 C2 B6 27 E1 F1 A7 FC 5B 26 C9 00
27E0 00 28 04 00 00 32 07 00 10 27 0E 08 AF 2F 1B F2
27F0 86 23 36 6B 71 27 6B FE 30 CA FD 27 DF 48 DF C9
>
```

表 8.6

```
D2800,29FF
2800 D5 E5 7C CD 04 02 7D CD 04 02 0E 20 DF DF 7E CD
2810 04 02 0E 20 DF DF 7E 87 DA 21 28 FA 80 28 C3 60
2820 29 FA 70 2A C3 A3 28 00 06 06 11 06 04 CD 5F 00
2830 E1 CD 6F 28 E5 C3 4F 28 06 02 11 06 03 11 06 04
2840 11 06 05 11 06 06 11 06 07 11 06 08 CD 5F 00 0E
2850 3B DF CD 00 02 E1 23 D1 D5 CD 31 02 FA 01 28 CF
2860 11 67 28 83 5F 1A C9 42 43 44 45 48 4C 4D 41 23
2870 5E 23 7E CD 04 02 7B CD 04 02 C9 00 48 41 4C 54
2880 47 FE EC CA FA 28 E6 70 0F 0F 0F 0F CD 60 28 4F
2890 DF 0E 3A DF 0E 3D DF 78 E6 0E 0F CD 60 28 4F DF
28A0 C3 4F 28 47 0E 41 DF CD 00 29 78 C3 98 28 00 00
28B0 4E 55 4C 4C 41 3A 3D 4D 5B 42 43 5D 3A 3D 41 3A
28C0 3D 4D 5B 44 45 5D 3A 3D 41 3A 23 44 45 43 41 3A
28D0 40 4C 41 3A 40 52 41 4B 3A 40 4C 41 4B 3A 40 52
28E0 4B 3A 23 5D 3A 3D 48 4C 3A 3D 4D 4D 5B 3A 2B 31
28F0 3A 2D 31 48 4C 3A 2B 00 00 00 21 7C 28 C3 3E 28
2900 CD 05 29 DF C9 78 E6 70 FE 70 0E 2D C8 0E 3A DF
2910 78 E6 70 0F 0F 0F 0F 0E 2B C8 3D C2 20 29 DF C9
2920 0E 2D 3D C8 3D C2 2A 29 DF C9 0E 26 3B C8 0E 23
2930 3D C8 0E 55 C9 00 11 53 50 CD 40 29 DF 48 DF C9
2940 01 42 43 E6 30 C8 01 44 45 FE 10 C8 01 48 4C FE
2950 20 C8 42 4B C9 78 11 41 46 C3 39 29 5A 4B 50 4E
2960 0F 47 5F 21 B0 28 A7 CA 3E 28 2E B7 FE 02 CA 4A
2970 28 2E B4 FE 0A CA 4A 28 2E C1 FE 12 CA 4A 28 2E
2980 BE FE 1A CA 4A 28 2E C8 FE 2F CA 3B 28 2E CB FE
2990 27 CA 3B 28 2E CE FE 07 CA 3E 28 2E D2 FE 0F CA
29A0 3E 28 2E D6 FE 17 CA 41 28 2E DB FE 1F CA 41 28
29B0 2E E0 FE 3F CA 3B 28 FE 32 C2 CA 29 0E 4D DF E1
29C0 CD 6F 28 E5 21 BC 28 C3 3B 28 2E B4 FE 3A CA 2B
29D0 28 FE 22 C2 E8 29 2E EA 06 02 CD 5F 00 E1 CD 6F
29E0 28 E5 21 E4 28 C3 3E 28 FE 2A C2 72 2B 2E E6 06
29F0 06 C3 2D 28 E6 04 CA 21 2A 78 E6 38 0F 0F 0F CD
>

D2A00,2BFF
2A00 60 28 4F DF 78 0F 2E F0 DA 3B 28 0F 2E ED D2 3B
2A10 28 0E 3A DF 0E 3D DF E1 23 7E E5 CD 04 02 C3 4F
2A20 28 68 78 E6 0F FE 09 C2 38 2A 2E F3 06 04 CD 5F
2A30 00 7B CD 36 29 C3 4F 28 7D CD 36 29 7D E6 0F 2E
2A40 ED FE 03 CA 3B 28 2E F0 FE 0B CA 3B 28 0E 3A DF
2A50 0E 3D DF C3 30 28 78 E6 08 C2 5F 2A 0E 4E DF 78
2A60 E6 30 0F 0F 0F 0F 11 5C 29 83 5F 1A 4F DF C9 00
2A70 1F 47 21 40 2B FE F9 CA 44 28 2E 44 FE EB CA 47
2A80 28 2E 4B FE E3 CA 47 28 2E 56 FE FB CA 3E 28 2E
2A90 5A FE F3 CA 3E 28 2E 5E FE E9 CA 41 28 2E 63 FE
2AA0 C9 CA 3B 28 2E 68 FE DB C2 B3 2A 06 04 CD 5F 00
2AB0 C3 17 2A FE D3 C2 C8 2A 0E 47 DF E1 23 7E E5 CD
2AC0 04 02 21 66 2B C3 3B 28 0E 4A FE C3 CA D7 2A FE
2AD0 CD C2 DB 2A DF 0E 53 DF C3 30 28 E6 07 C2 E9 2A
2AE0 0E 52 DF CD 56 2A C3 4F 28 3D C2 F6 2A CD 55 29
2AF0 21 52 2B C3 3E 28 3D C2 03 2B 0E 4A DF CD 56 2A
2B00 C3 30 28 3D 3D C2 10 2B 0E 4A DF 0E 53 C3 FC 2A
2B10 3D C2 23 2B 2F 50 C5 06 04 CD 5F 00 C1 CD 55 29
2B20 C3 4F 28 3D C2 33 2B 0E 41 DF 78 87 47 CD 00 29
2B30 C3 17 2A 0E 4A DF 0E 53 DF DF 78 0F 0F C3 6C 2B
2B40 53 50 3A 3D 48 4C 3A 3D 3A 44 45 48 4C 3A 3D 3A
2B50 53 54 3A 3D 53 54 49 3A 3D 31 49 3A 3D 30 4A 5B
2B60 48 4C 5D 52 45 54 3A 3D 41 3A 3D 47 0F E6 37 C3
2B70 9E 28 FE 37 C2 F4 29 0E 4B DF 21 57 2B C3 3B 28
2B80 7B 0E 2B A7 F2 8C 2B CD 5B 26 0E 2D DF 97 F5 7B
2B90 A7 CA DE 2B 7B D6 28 7A DE 04 FA A9 2B 01 DF 27
2BA0 CD EC 26 F1 3C F5 C3 94 2B 7B D6 20 7A DE 01 F2
2BB0 BE 2B 01 DF 27 CD 4B 26 F1 3D F5 C3 A9 2B 7A 3C
2BC0 16 00 EB 29 EB 29 D2 CA 2B 13 3D F2 C2 2B 01 54
2BD0 00 09 D2 D6 2B 13 4D 09 D2 DC 2B 13 6C 63 7A FE
2BE0 0A CA E9 2B F7 4F C3 ED 2B 0E 31 DF 0D DF CD E6
2BF0 07 0E 45 DF F1 F2 2C 26 F5 0E 2D DF F1 C3 2A 26
>
```

微处理机的程序设计和软件研制

[英] F·G·邓肯 著
白英彩 译

*

上海科学技术文献出版社出版
（上海市武康路2号）
新华书店上海发行所发行
宜兴南漕印刷厂印刷

*

开本 787×1092 1/32 印张 14.25 字数 341,000
1982年8月第1版 1985年2月第3次印刷
印数：24,801—48,600

书号：15192·217 定价：2.00元

《科技新书目》86-234