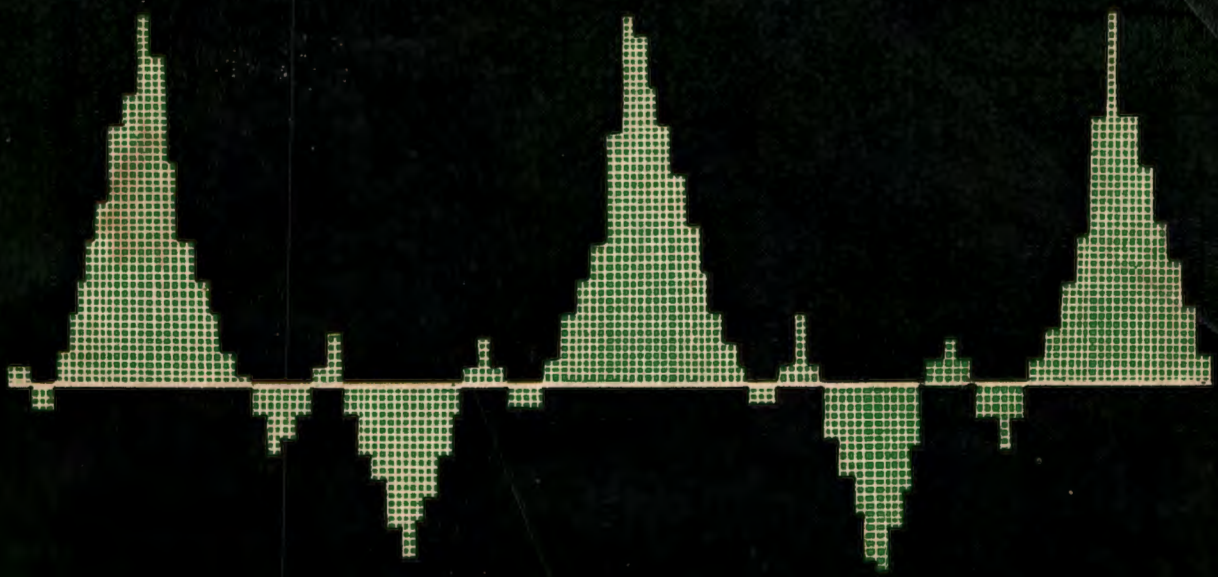
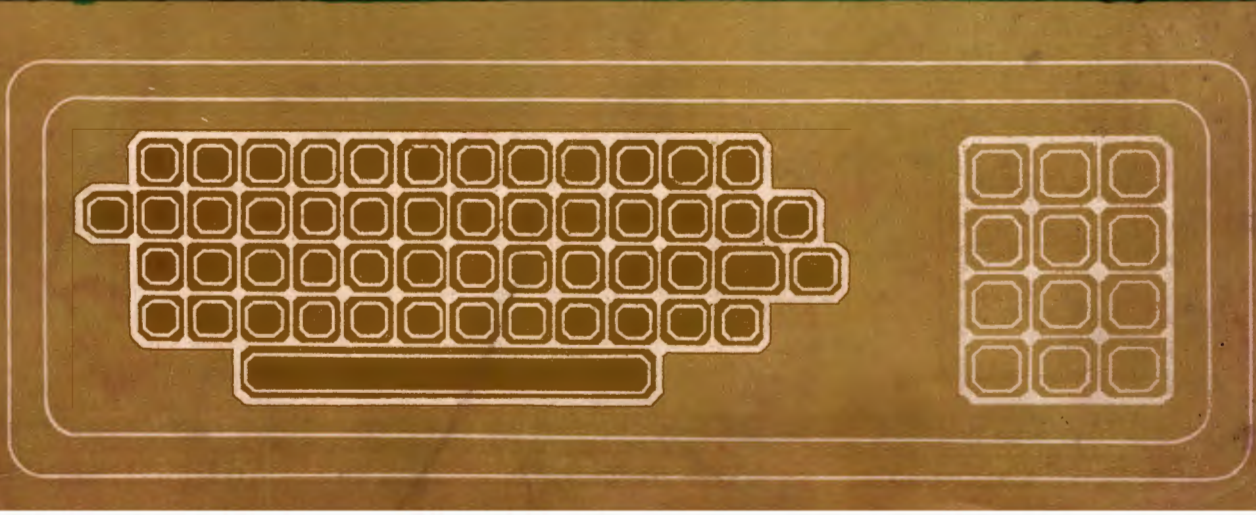


TRS-80
微计算机驻机解释程序
——LEVEL II ROM剖析

[美] J.法沃尔 著

海洋出版社



责任编辑：刘莉蕾

统一书号：13193·0308
定　　价：3.70　元

TRS-80 微计算机驻机解释程序
——LEVEL II ROM 剖析

〔美〕 J. 法沃尔 著

梁祖威 周宝兴 宋知用 译

海 洋 出 版 社

1984年·北京

内 容 简 介

本书详细解释了 TRS-80 I 型的 LEVEL II 驻机 ROM 的分析及程序清单，提供了关于输入/输出管理、各种变换、算术运算、数学函数、BASIC 功能和系统功能等 70 多个有用的子程序入口、使用说明和例题，可供用户在汇编语言级调用。书中还专门介绍了 TRS-80 盒式磁带机、小型软磁盘的工作、记录格式以及系统软件的内部、外部表格，为用户充分利用机内软件资源提供了手段。

本书读者对象：从事微计算机研究、生产的科技人员，特别适合使用 TRS-80 微计算机及同类机器 PS-80，PS-85，SEED-Z80，MDR-Z80，YEE8100，BC3-80 和 DJS-043 等的技术人员和高等院校师生阅读。

TRS-80 微计算机驻机解释程序
——LEVEL II ROM 剖析

〔美〕J. 法沃尔 著
梁祖成 周宝兴 宋知用 译

海洋出版社出版（北京市复兴门外大街）
北京通县科技印刷厂 新华书店北京发行所发行
开本：787×1092 1/16 印张：21 字数：550千字
1984年8月第1版 1984年8月第1次印刷
印数：1—60,000

统一书号：13193·0308 定价：3.70元

译 者 前 言

TRS-80 微型计算机是由美国 TANDY 公司 Radio Shack 分部生产的。它是一种以 Z80 为 CPU 的 8 位微计算机系统。这种微型计算机是当前世界上生产数量较多的机型之一，它拥有很多的用户。这种微型计算机可以通过配接小型软磁盘驱动器、盒式磁带录音机和行式打印机等多种外部设备，组成一个多功能的微计算机系统。在软件方面，除 TRSDOS 操作系统以外，还有 NEWDOS 及 NEWDOS/80 等操作系统以及各种高级语言，它们既适用于数据处理、一般工程和科学计算，也可供家庭使用，此外还配有教育(算术、代数等)、游戏、帐目管理、仓库管理等应用软件。

目前我国已引进相当数量的 TRS-80 微型计算机及软件上兼容的 PS-80, PS-85, SEED-Z80 等微型计算机。另外，国内生产的若干种微型计算机，如 MDR-Z80, YEE 8100, BC3-80 和 DJS-043 等，在软件上也与 TRS-80 兼容。所以，怎样更好地使用这些微型计算机，使它们发挥更大的效用是一件具有现实意义的迫切的任务。出于这个目的，我们翻译了这本书。

本书是一份很有实用价值的参考资料，它为读者提供了 TRS-80 I 型机的 Level II 驻机 ROM 的分析及程序清单的详细解释。给出了关于输入/输出管理、各种变换、算术运算、数学函数、BASIC 功能和系统功能等 70 多个极有用的子程序入口、使用说明和例子，可供使用者在汇编语言级调用。

本书有专门章节给出 TRS-80 的盒式磁带机和小型软磁盘的工作及记录格式，并对系统软件中的各种内部、外部表格(如保留字清单、程序语句表、变量表、通讯区、设备控制块等等)的实际结构作了详细介绍，并用实例予以说明，以帮助读者进一步理解 ROM 内子程序的调用及系统软件中各种表格的使用。这样，就把 TRS-80 微计算机内软件的奥秘全部暴露在读者的面前。这为充分利用机内的软件资源提供了极大的方便，也为扩充系统功能提供了手段。

本书对于从事微计算机研究、生产的科技人员，特别是对使用 TRS-80 微型计算机及类似机器的科技人员有一定的参考价值，也可供高等院校有关专业的师生参考。

在本书的翻译过程中，对于已发现的原文错误均作了改正，除个别情况外，一般印刷及显而易见的错误不再加译注。同时为了方便读者查阅，我们增加了汉英词汇(包括书中常用的缩写字)对照表及全部子程序入口和功能索引。

由于译者的水平所限，难免有错误和不妥之处，欢迎读者批评指正。

译 者

1983 年 3 月

序　　言

几年前，我曾对 J. 法沃尔说："杰姆，你为什么不写一本关于 Microsoft（微软件公司）BASIC 和 TRS-80 的书？你有才干和经验，而数千名 TRS-80 用户需要帮助，特别是我！"没费口舌，他就同意了。虽说是写一本书，但实际上为此要做很多别的事情。

著书立说需要有丰富的想象力，对题材的渊博的知识、才干以及与读者联系的能力。J. 法沃尔是具备这些条件的。

这是本不寻常的书。你将看到它是在这同一题目或任何类似题目的书中论述和提供资料最全面、清楚、详细的一本书。

已出版的论述 TRS-80 BASIC 解释程序和操作系统的其他书和小册子，虽然具有一定的价值，但只适用于有经验的机器语言程序员，而且这些书都有不少缺陷。

本书将会同时受到专家和初学者的欢迎。它除了引导你打开电源和复位（有或没有磁盘）以外，还详细地说明了软件系统操作的每一个领域。你还会看到一些例子、表格和流程图，它们补充说明了程序清单。书中有七千多条对 Microsoft BASIC 解释程序和操作系统的注释。这些注释不是通常的机器语言程序员所作的注释，因为他们的那些不可思议的和暧昧的注释往往使读者如堕云雾之中，而本书所给出的注释是任何人都能懂得的英语注释。不仅如此，而且当一个注释需要进一步加以说明时，还附加了更详细的注释。

本书还有一些内容，可供每一个在以 Z80 为 CPU 的计算机上运行 Microsoft BASIC 的使用者参考。Microsoft 公司有一个部门，以它的巨大智慧为类似的机器生产相似的程序。或许你会发现，一个程序虽然是你的技术人员各自编写的，但程序的大部分却是相同的！

这本书可能成为你所有的书中最有用的书。

H. C. 彭宁顿

1981 年 11 月

目　　录

第一章 绪 论

Level II 由基本操作系统和 BASIC 语言解释程序组成。把它们合在一起称为 Level II ROM 系统。Level II 系统有一个扩展部分，称为磁盘操作系统 DOS，Level II BASIC 也有一个扩展部分，称为磁盘 BASIC。

可以把 Level II 和 DOS 两者看作两个独立的操作系统。两个操作系统如何依赖相存和协调工作，是本书的部分论题。本书主要是描述 Level II ROM 的基本操作，由此可使汇编语言程序员能有效地利用此系统。

没有操作系统的计算机是毫无用处的。需要操作系统的理由是为了在计算机和用户之间提供通讯手段。这一手段使计算机"倾听"键盘，由此知道我们想干什么，并让计算机在显示器上给出信息，告诉我们它正在干什么。当我们编写程序告诉计算机干什么时，在机器内必须有一程序来听取我们的命令，这个程序就称为操作系统。

要给操作系统一个非常确切的定义是不可能的。操作系统有几千种，它们相互之间只是略有差别。这些差别的原因是由于操作系统是为各特定用户提供的，或是为了利用不同机器的硬件特性而设计的。不论操作系统之间存在什么差别，大多数操作系统的基本内部子程序是非常相似的——至少在功能上是如此。

在通用的单用户操作系统中，如 Level II，共同的部分有以下几个：

1. 所有外围设备，如键盘、显示器、打印机和盒式磁带机的驱动程序。

2. 某种语言处理的能力(如 BASIC，COBOL 或 FORTRAN)。

3. 对于提供的每一种语言进行支援的目的码运行时间程序。它应包括数学和算术子程序，它们隐含在语言中。

4. 语言处理程序及其隐含的子程序所使用的辅助支援例行程序。对用户来说，它们通常是看不见的。它们管理着系统资源，如存储器和表格以及控制外部设备的存取。

5. 简单的监控程序。它连续地监控键盘或系统的其他输入设备，搜索用户的输入。

6. 系统实用命令。应注意的是各系统不尽相同。例如，在 Level II 中是 EDIT，LIST，CLOAD 等等。

请记住，这些定义是极广义的。对于不同的操作系统，每个部分的精确定义都是各不相同的。在后面我们将更深入地研究 Level II ROM 的各组成部分。首先，我们要讨论操作系统最初是怎样被装入计算机的。通常，有两种方法可以装入操作系统。一种方法是操作系统可以永久地记录在随系统同时提供的称为只读存储器（ROM）的特殊类型的存储器中。在这种情况下，操作系统是常驻的，只要在它的起点进入，就可以初始化系统并开始接受命令。

操作系统装入机器的另一种方法是将它从某种外存储介质如磁盘或磁带上读入。然而，在这种情况下，我们需要一个程序将操作系统读入机器。这个程序称为初始程序装入程序（IPL），并必须用手工输入或存放在系统内某个地方的 ROM 中。为了简便，我们将

假设所有的机器至少有一个 IPL ROM 或具有操作系统的 ROM。

在 TRS-80 I 型机中，我们将 ROM 和具有操作系统的磁盘两者组合在一起。Level II 机器有一个 ROM 系统，它占用可寻址内存的前 12K 字节。当打开电源或按复位按钮时，控制就被无条件地分别传送到地址 0 或 66H。在这些地址单元中存储的是 JUMP 指令，以转移到 ROM 的其他区域进行系统初始化，然后显示出信息“MEMORY SIZE ?”。

在有磁盘的 Level II 系统中，相同的 ROM 程序依然占用着前 12K 字节内存，但是当打开电源或复位处理时，就从磁盘读取另一个操作系统并装入内存。这个磁盘操作系统（DOS）占用由地址 16384 开始的 5K 字节的 RAM。在装入以后，控制就被转移到 DOS，从而对 DOS 本身进行初始化，并显示提示信息“DOS READY”。因此，即使 ROM 操作系统是常驻的，若机器带有磁盘，则也要装入另一个操作系统。在这种情况下，Level II ROM 的作用相当于 IPL ROM。

应强调的是，DOS 和 ROM 操作系统是相辅相成的。每一个操作系统都提供了另一个操作系统所缺乏的功能。DOS 所需要的基本功能可在 ROM 中找到，同时，DOS 又包含有 ROM 的扩展部分以及它自己的独特功能。

Level II 和 DOS 概述

Level II 是一个独立的操作系统，它自己就能独立运行。它是常驻的，并含有 BASIC 解释程序及执行 BASIC 程序所必需的支援程序。它也有将程序从盒式磁带读出或写入盒式磁带的功能。

磁盘操作系统（如 TRSDOS 或 NEWDOS）是 Level II 的扩充，它在 IPL 过程中从磁盘装入。DOS 与 Level II 有以下几方面不同：首先，DOS 没有 BASIC 解释程序，因此，要键入 BASIC 语句时，必须将控制由 DOS传送到 Level II，这是用打入 DOS 命令 BASIC 来达到的。同样，在把控制由 DOS 传送到 Level II 时，这个命令也完成初始化操作，这方面的内容将在后面讨论。其次，由 DOS 识别的命令通常是一些磁盘实用程序，而不是那些常驻在 ROM 中的子程序——如 Level II 中的子程序。这意味着在使用 DOS 命令之前必须将它们从磁盘装入。这本身就表明必须保留一个 RAM 区域，用以装入和执行这些实用程序。

内存的利用

由 DOS 和 Level II 的描述我们可以知道，RAM 部分的使用将依正在使用的操作系统而不同。在 IPL 以后，内存就立刻按照每个操作系统的要求进行分配，如下面图 1.1 所示。注意中央处理单元（CPU）在每部分图中的位置。

有磁盘的 Level II 系统在执行 BASIC 命令后其内存安排如图 1.2。

内存的前 16K 不论使用哪种操作系统都分配给 Level II 和通讯区。

由 Level II 用来存放 BASIC 程序及其变量的那部分存储器，是从通讯区的终点开始还是从磁盘 BASIC 区的终点开始，这取决于所使用的操作系统。这部分内存也可被程序员用于存放汇编语言程序。下面详细说明没有磁盘的 Level II 系统的这个区域。

虽然图 1.3 所示的 RAM 子区域是固定的，但是它们实际不是这样的！所有的区域会往上或往下移动，而这取决于你执行什么操作。例如，在程序中插入或删去一行时，

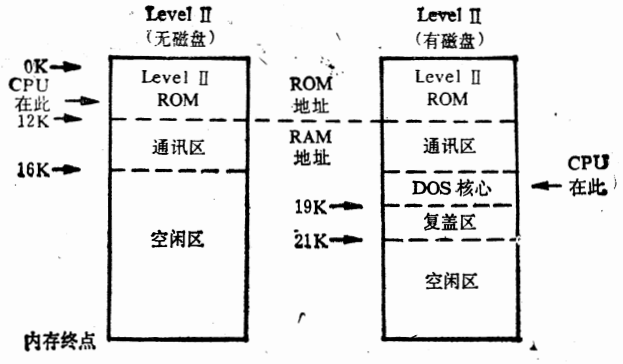


图 1.1　在初始程序装入后的内存安排

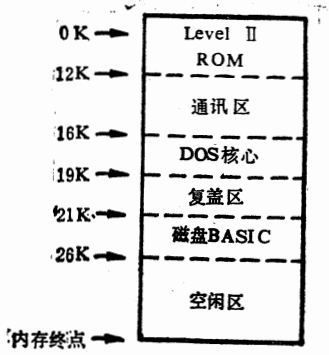


图 1.2　在 BASIC 命令后磁盘系统的内存分配

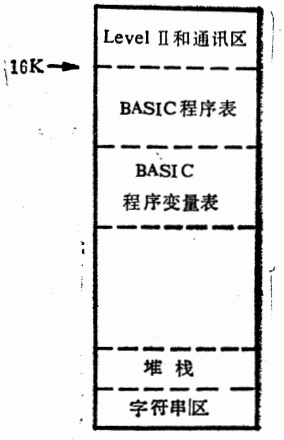


图 1.3　没有磁盘的 Level Ⅱ 系统中内存的分配

将使 BASIC 程序表(称作程序语句表或 PST) 的大小增大或缩小。同时，定义新的变量也将增加变量表的长度。由于这些表格会移动，所以要把它们的地址保存在通讯区中固定的单元内，这就使得表格可以按需要进行移动，并提供了让其他用户知道它们在何处的手段。

程序语句表 (PST) 放有压缩格式的 BASIC 程序源语句(保留字已由代表它们意义的代号所替代)。这表格的首址是固定的，但是它的终址随程序大小而变化。由于程序语句的增删，PST 的终址就相应地移动。对于此表格的详细描述可在第四章中找到。

在 PST 的后面是变量表 (VLT)，它含有 BASIC 程序中使用的全部变量的名称和值。根据下列变量类型，它又分成四个子表格：简单变量(无维的)表、一维表、两维表和三维表。变量名和它们的值按照程序执行过程中遇到它们的次序存放。当把新的变量加入程序时，变量表的大小就会改变，而删除变量将使表格缩减。变量被定义以后就留在表中，直到重新初始化系统时为止。关于本表的详细说明请看第四章。

在图 1.3 中没有绘出的是空闲区 (FSL)，原先它是从通讯区终点延伸到字符串区低端边缘的一段内存。这个表有两个部分，第一部分用于为 PST 和 VLT 分配空间。对于这些区域，其空间的分配是从存储器的低端到高端进行的。FSL 的第二部分用作堆栈区，

这个空间的分配是以相反方向——从字符串区的顶端开始往下朝 Level II 方向进行的。

堆栈区是一个动态(可变的)表格。它被 Level II 和 DOS 系统用作子程序返回地址和硬件寄存器的暂存区域。任何 CALL 或 RST 指令将无条件地使下一条指令的地址存入(PUSH) 堆栈,并且堆栈指针自动减量到下一个相邻低地址。执行 RET 指令(当从子程序返回时使用)就从堆栈移出两个字节(等效于 POP 指令),并且将堆栈指针加 2 。

堆栈中的存储空间可用程序分配,但是必须小心地设计。某些 BASIC 子程序,如 FOR—NEXT 子程序,将把涉及它们操作的所有值存入堆栈。在 FOR—NEXT 情况,一个 18 个字节的块(称为一帧)被推入 (PUSH) 堆栈,并一直保留在堆栈中,直到 FOR—NEXT 循环执行完毕。

FSL 由两部分组成,在给其中任何一个分配空间以前(像 CALL 或 PUSH 这样的堆栈指令除外),要测试(通过调用 ROM 子程序)是否有足够的空间。如果没有足够的空间,就会在显示器上给出"内存出界" (OM) 错误信息。关于调用 ROM 子程序得到 FSL 中可用空间数目的详细说明请看第二章。

内存分配图中给出的最后一个区域是字符串区域。这是一个固定长度的表格,它从内存终端开始往内存的低地址方向工作。这个区域的大小可用 CLEAR 命令指定,它的缺项*大小是 50 个字节。字符串变量就存储在这个区域,而字符串等于字符串**、字符串函数 (string$) 和带引号的字符串都存放在 PST 中。

前面讲过,一个操作系统由六个常见部分组成。因此,Level II 将分布在 ROM 中的某些部分归并成几个独立的片段,而不是集中在单个区域。图 1.4 是 Level II 中内存地址的简图。关于这些区域的精确地址和详细说明,请参看第四章。

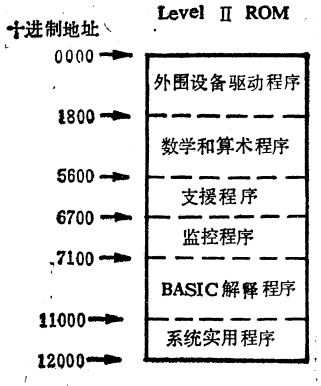


图 1.4 Level II 内存地址简图

通讯区

对于 Level II ROM 来说,通讯区是一个便笺式暂时存储器。以存在这儿的地址为例,有 PST 和变量表的地址。另外,还有 BASIC 支持的变量类型——它们需要的空间比工作寄存器能提供的要多,因此,某些算术操作也需要这个暂存区域。

通讯区的另一个重要用途是提供 Level II 和 DOS 之间的连接——往返传送地址和数据。DOS 出口地址和磁盘 BASIC 地址都保存在这个区域。如前所述,带有磁盘的 Level II 系统是在 DOS 系统中开始执行的。只是在命令 BASIC 被执行以后,控制才从 DOS 传送到 Level II (它也将 DOS 出口和磁盘 BASIC 地址存储起来,以此修改通讯区)。

因为 Level II 在 ROM 中,要想修改它是不可能的。然而,修改操作系统是必须考虑的实际需要。为解决这一问题,Level II 系统写有一些转移到 RAM 区的跳转指令,以便将来的改变能够和 ROM 系统合并在一起。这些跳转称为 DOS 出口,在没有 DOS

* 缺项 (default) 是指用户没有指定参数时系统自动设置的某一固定参数。——译注

** 这儿所说的字符串变量是指它的值,字符串等于字符串是指字符串赋值。——译注

的系统中，它们只是返回 Level II。当有 DOS 时，这些跳转地址变成磁盘 BASIC 的地址，通过这些地址可以改变 Level II 的操作。

在遇到像 GET 或 PUT 这样的磁盘 BASIC 命令时，Level II 就用到磁盘 BASIC 地址，因为在 Level II 中没有支持这些操作的程序。这些命令是装入 RAM 的磁盘 BASIC 的一部分，由于它可以装入任何地方，因此，Level II 需要有找到它的方法。磁盘 BASIC 出口是 Level II 和磁盘 BASIC 两者都知道的一组固定地址，它允许 Level II 将控制传送到磁盘 BASIC 的某些工作子程序。

通讯区的另一个重要方面是它含有一段称作除法支援例行程序(Divide Support Routine) 的程序。这一程序由除法子程序调用，以执行减法和测试操作。在 IPL 过程中，它被从 Level II 拷贝到 RAM 通讯区。当有 DOS 时，就有 DOS 实用程序 BASIC 把它从 ROM 移到 RAM 中。

在磁盘系统上，一个使用 Level II 除法支援程序的汇编语言程序，若没有执行过 BASIC 命令就不能工作，因为除法支援程序不在 RAM 中。所以，在执行带有除法子程序的汇编语言程序时，请首先执行 BASIC 实用程序，或者将支援例行程序拷贝到 RAM 中。

Level II 操作

在本章的前面曾简略地讲述过，在所有操作系统中通常可以找到的六个组成部分。以这些组成部分为借鉴，Level II 可以分成以下六个部分：

第一部分——输入阶段；

第二部分——解释和执行程序；

第三部分——工作子程序；

第四部分——算术和数学运算；

第五部分——I/O 驱动程序；

第六部分——系统实用程序。

另外还有一个所有系统都有的共同部分，它没有包含在上面的表中。这一部分与系统初始化有关（IPL 或复位处理），我们将单独讨论。我们继续讨论 Level II 的六个部分，我们将先从系统准备接受第一条语句或命令开始，并称其为输入阶段。

第一部分——输入阶段

输入阶段是所有操作系统的共同部分。其功能是接受键盘输入和对接收到的命令作出响应。在 Level II 系统的情况下，它具有双重任务，系统命令和 BASIC 程序语句都由这个程序处理。

进入输入扫描程序的入口在 0075H。这是初始入口点，通常只调用它一次。在进入主循环之前，显示出信息“READY”和取得 DOS 出口 (41ACH)。没有磁盘的系统，在 IPL 处理的末尾将自动地跳转到这点。对于有磁盘的系统，是由 DOS 实用程序 BASIC 在它的处理末尾进入这个程序的。输入阶段或扫描阶段摘要归纳如下：

1. 得到由键盘输入的下一行；

2. 用代号替代保留字；

3.测试代号，看它是系统命令（如 RUN, CLOAD 等）还是直接语句（没有行号的 BASIC 语句）。若是后者，则转移到步骤 6；

4.将代号化的语句存储在程序语句表中；

5.回到步骤 1；

6.开始解释和执行。

输入阶段循环从 1A33H 开始。在显示出提示符 >，或行号以后——如果处于自动方式（Auto Mode），则 CALL 0361H 去读下一行。然后，用 CALL 1E5AH 将行号由 ASCII 转换成二进制。接着扫描语句，并将保留字用代号替代（经 CALL 1BC0H）。在代号化以后，立即得到 DOS 出口 41B2H。返回时，测试有无行号。若没有行号，则假设为系统命令或直接语句，控制就传送到 1D5AH 的执行驱动程序。在没有磁盘的系统中，这个测试是在 1AA4H 进行的；在磁盘系统中，此测试和转移是由 DOS 出口 41B2H 调用 1AA1H 进行的。

如果有行号，则把输入的程序行加到 PST 中，连接每一行的指针由 1AFCH 到 1B0EH 的子程序进行修改。如果该行代替已存在的行，则调用在 2BE4H 处的子程序，把被替代行以下的所有程序行进行移动。

在处于自动方式*时，当前行号被存放在 40E2H 和 40E3H 中，行间的增量存放在 40E4H 中。从 1A3FH 到 1A73H 的程序显示和保持自动行号值。空行（语句只有行号）都被舍弃，它们由 1ABFH 处的测试程序进行检测。

第二部分——解释和执行程序

在 Level II 系统中，语句和命令是通过解释执行的。这意味着，调用一个既可用于语句又可用于命令的程序去解释每一行，并完成必要的操作，这是执行系统命令的共同方法。例如，在 DOS 中用 FORMAT 和 COPY 这样的命令，装入各独立的模块。在有些系统中，有关的命令可以合并在一个模块中。当装入该模块后，它判别（解释）调用的名称，由此决定执行哪个子功能。

除微型计算机以外，程序的执行并非都是通过解释的方法进行的，而且也只是在选用的某些语言上，如 BASIC 和 APL 等才如此。代替解释程序的是使用编译程序进行程序的编译和执行。

编译程序将源语句翻译成可直接执行的机器语言代码（称作目的代码）。然后，作为一个独立的步骤，用称为装入程序的实用程序将目的代码装入 RAM。在目的代码装入 RAM 以后，就把控制传送给目的代码，而它的执行几乎与操作系统无关。

不是所有的源码都被编译程序转换成目的代码。某些语句如 READ 和 WRITE，或某些函数如 SIN（正弦）和 COS（余弦），编译程序能识别它们，并产生对特定例行程序的子程序调用，而不是生成它们的目的代码。

这些子程序在库文件中是目的代码形式。当装入程序装入目的代码（编译后的程序）时，所有的子程序调用就从库文件中得到满足（子程序被装入）。从库程序中获取模块的装入程序，称作连接装入程序。

---

* 用 AUTO 命令自动编写行号。——译注

解释程序的操作是十分简单的。它利用比较的方法，扫描每一个源语句的保留字，如IF, FOR, GOTO 等等，将每个保留字用称为代号的唯一的数字值代替，然后存储代号化后的源语句。在 Level II 中，它存储在程序语句表（PST）中。当运行程序时，控制就传送到一个执行驱动程序，它扫描每个语句寻找代号。当找到一个代号时，控制就传送到与此代号有关的子程序。这些代号的子程序(也称工作子程序) 进行句法检查，例如测试数据类型是否有效，逗号位置正确与否，括号是不是闭合的。在编译程序中调用的工作子程序是没有句法检查的，因为编译程序已经进行过句法检查，并且只有在所有的参数都是正确的时候，才去调用子程序。

在 Level II 中，当已经接收到一个没有行号的语句时，或者已经给出 RUN 命令时，才进入执行阶段。这可以是一个系统命令或者是一个要执行的 BASIC 语句。当接收到 RUN 命令时，就执行整个 BASIC 程序。虽然执行驱动程序循环由 1D5AH 开始，在 1DE1H 结束。但是，因为其它一些子程序要共享驱动程序的某些部分，所以对于这些子程序，驱动程序的首址和终址就不一定是上述的地址。

这个阶段中的步骤摘要归纳如下，详细说明参阅图 1.5。

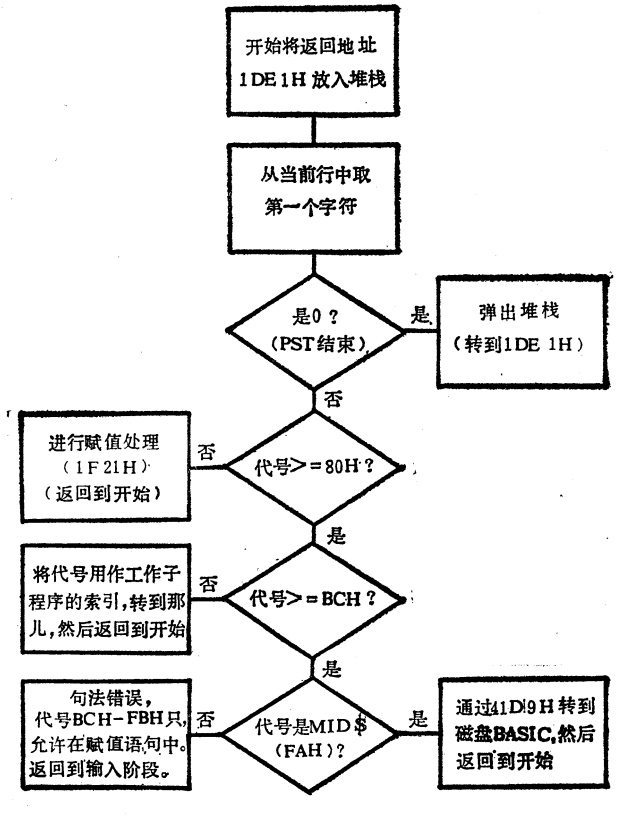


图 1.5　执行驱动程序的流程图

1. 取 PST 中当前行的第一个字符，如果到达了 PST，的末尾，则返回输入阶段；
2. 若该字符不是代号，则去步骤 6；
3. 如果代号大于 BCH，那么它必须是 FAH (MID$ 的代号)，否则给出句法错误；

4. 如果代号小于 BCH，则用它作为进入工作表的索引；

5. 转到工作子程序和返回步骤 1；

6. 分配部分。寻找变量名称，如果变量名没有被定义过，那么建立该变量；

7. 进行表达式求解；

8. 返回步骤 1。

执行驱动程序是由取入 PST 中当前行的第一个字符开始执行的，然后测试这个字符，看它是否是一个代号（80H—FAH)。假设当前行是这样一个赋值语句：

$$A = 1$$

赋值语句子程序由 1F21H 开始。除了它是直接进入而不是通过查表处理的以外，它和别的工作子程序一样。在它进入以前，执行驱动程序中的返回地址 1DE1H 被推入堆栈，因此，它能够像其他工作子程序一样返回。

对赋值子程序应假设当前行指针直接在被赋值变量名称的左边。它寻找或建立该变量名称的登记项，测试变量名后的等号(=)，然后 CALL 2337H，用该处的子程序求解表达式的值。其结果被转换成正确的类型并存放在变量地址。

假设按照第一个字符找到了相应的代号，就要进行第二次测试，看它作为一行中的第一个代号是否有效。能够处于一行开头的有效代号是 80H—BBH。代号 BCH—FCH 只能作为赋值语句的一部分而存在，或在个别语句，如 8FH(IF) '表达式' CAH (THEN) ××××中遇到。MID$ 的代号 FAH 是这个规则中的唯一例外，对它的测试在 2AE7H 进行，在那儿得到了直接跳转到它的磁盘 BASIC 向量 (41D9H)。如果代号在 80H 和 BBH 之间，它就被用作进入工作子程序表的索引，并找到该代号的工作子程序的地址。然后，控制被传送到该工作子程序。工作子程序进行所有的句法检查和完成所需的功能。

工作子程序的参数是语句中代号后边的符号。每个子程序都知道它应该得到的合法字符是什么，并从左往右(就从代号后开始)扫描输入字符串，直到参数的结束。参数的结束必须与语句的结束一致，否则就产生句法错误。

结束参数清单的符号随每个工作子程序而不同。右括号")"结束所有的数学和字符串函数。一个机器零 (00H) 字节结束赋值语句，其他子程序在验证所需的值出现以后，会返回到执行阶段。

在每个工作子程序完成后，控制返回到执行驱动程序，在那儿测试语句的结束 (EOS) 或混合语句的冒号(:)。EOS 是一个字节的机器零。如果检测到 EOS，则从程序语句表中取来下一行，并且它就成为执行驱动程序的当前输入行。

当执行完系统命令或直接语句以后，就没有指向下一语句的指针，因为它们的执行来自输入阶段的输入缓冲器。这是一个不同于 PST 的区域，BASIC 程序语句是存储在 PST 中的。在执行 RUN 命令时，它使执行驱动程序从 PST 中得到它要执行的内容。

当到达 BASIC 程序或系统命令的结束时，控制被无条件地传送到 END 工作子程序，最后它将返回输入阶段。在执行过程中或解释阶段，检测到任何错误都使控制在显示出相应错误代码后，返回到输入阶段。句法错误例外，它将直接返回到编辑方式。

第三部分——工作子程序

工作子程序 (Verb action routine) 是 ROM 中进行实际工作的地方。对于所有系统

命令，如 CLOAD, SYSTEM, CLEAR, AUTO 等，以及 BASIC 动词，如 FOR, IF, THEN, GOTO 等等，都有对应的工作子程序。另外，还有对应所有数学函数和编辑程序子命令的工作子程序。

工作子程序从执行阶段找到动词代号的地方开始，继续分析输入字符串。像执行阶段那样，它们以从左到右的次序检测字符串，寻找特殊的字符，如(,)即逗号，以及被执行动词的唯一代号。如果丢失了所需字符，或者出现不合理的条件，则产生句法错误。

工作子程序使用许多内部子程序，以帮助它们执行程序语句。虽然 Level II 系统的其他许多部分也使用着这些内部子程序，但仍可以将这些内部子程序看作工作子程序的一部分。

内部子程序的一个较好的例子是由 2337H 开始的表达式求值程序。那些允许把表达式作为其参数的，或者已经把它作为一个参数的任何工作子程序都可以调用这个表达式求值子程序。例如，允许表达式作为它们的参数的工作子程序有 IF, FOR 和 PRINT。同样，表达式求值子程序也将调用其他内部子程序（例如，调用 260DH 得到求值表达式中变量的地址）。由于下标变量可以用表达式作它的下标，同样，求地址子程序又会返过来调用表达式求值子程序。

这种处理形式叫做递归，并可用下列表达式表示：

$$c0 = c(1a/bc(2d)/c(1 \times c0))$$

工作子程序使用的其他内部子程序的入口有：跳到语句末尾的 1F05H；检索 FOR 结构堆栈的 1936H 和建立文字串库登记项的 2865H。

任何需要给出的中间结果都存储在通讯区中的工作寄存器区 1(WRA1)。某些动词，如 FOR，建立堆栈结构，是可以由其他动词，如 NEXT，进行检索和识别的。所有的工作子程序，除 MID$ 以外，都是以图 1.6 所示的寄存器设置进入的。

工作子程序的全部清单和它们的入口点在第四章给出。

寄存器	内　　容
AF	从代号后面开始的代码串的下一个元素
	C（进位标志置位）——若是数字
	NC（进位标志复位）——若是字母
BC	工作子程序的地址
DE	工作代号在代码串中的地址
HL	代码串中下一个元素的地址

图 1.6　工作子程序入口的寄存器设置

第四部分——算术和数学运算

在将要讨论算术和数学子程序之前，我们先回顾一下 Z80 CPU 和 BASIC 解释程序的算术能力。

Z80 只支持 8 位和 16 位的整数加和减，不支持乘和除，也不支持浮点运算。它的寄存器组由 7 个 16 位寄存器组成。所有的算术运算都必须在这些寄存器之间进行，不允许在存储器和寄存器之间运算。同样，寄存器之间的运算也受到极大的限制，特别是 16 位量的运算。

BASIC 解释程序支持所有的运算，例如三种变量类型(模式)的加、减、乘和除。这三种变量类型是：整数型、单精度型和双精度型。这种支援是内部子程序提供的，它进行的运算等效于硬件运算。由于软件的复杂性，混合方式的运算，例如整数型和单精度型的混合运算，是不支持的。任何混合变量类型的尝试都将给出不可预计的错误结果。

BASIC 支持的变量类型的大小如下：

整数型	16 个二进制位 (15 位数值，1 位符号位)。
单精度型	32 个二进制位 (8 位偏置指数，加 24 位带符号的尾数)。
双精度型	64 个二进制位 (8 位偏置指数，加 56 位带符号的尾数)。

由此可以清楚地看到，如果要硬件支持浮点运算，寄存器也存不下两个单精度数值或双精度数值。因为对寄存器来说数字可能太大，同时由于软件必须分成若干步来进行，所以必须使用一个 RAM 区域来支持这些运算。

在通讯区中已设置了两个区域来支持这些运算。这两个区域被标记为工作寄存器区 1 (WRA1) 和工作寄存器区 2 (WRA2)。它们分别占有 411DH—4124H 和 4127H—412EH 存储单元。它们用于存放一个或两个运算量，这根据运算量的类型而定，以及存放所有单精度和双精度运算的最后结果。工作寄存器区的详细说明如下：

WRA1 地址	WRA2 地址	整数型	单精度型	双精度型
411DH	4127H			LSB
411EH	4128H			NMSB
411FH	4129H			NMSB
412OH	412AH			NMSB
4121H	412BH	LSB	LSB	NMSB
4122H	412CH	MSB	NMSB	NMSB
4123H	412DH		MSB	MSB
4124H	412EH		指数	指数

图 1.7　工作寄存器区安排

图 1.7 中，LSB = 最低有效字节；　NMSB = 次高有效字节；　MSB = 最高有效字节。

因为不支持混合方式运算，所以整数运算只能在整数之间进行，对于单精度和双精度也同样如此。由于有四种算术运算 (+，−，* 和/) 和三种数值类型，因此必须有 12 个算术子程序。这些程序的每一个都可识别出它们能够运算的数值类型，并且要求在调用以前将那些数值装入相应的 CPU 寄存器或工作寄存器。图 1.8 表明了用于算术子程序的寄存器分配。这些分配对于数学子程序是无效的，因为它们只对单个值进行运算，并且总是假设在 WRA1 中。

整　数　型

存放结果的寄存器	运　算	进行运算的寄存器
HL	加	DE+HL
HL	减	DH—HL
HL	乘	DE*HL
WRA1	除	DE/HL

单　精　度　型

存放结果的寄存器	运　算	进行运算的寄存器
WRA1	加	(BCDE)+WRA1
WRA1	减	(BCDE)−WRA1
WRA1	乘	(BCDE)*WRA1
WRA1	除	(BCDE) / WRA1

双　精　度　型

存放结果的寄存器	运　算	进行运算的寄存器
WRA1	加	WRA1+WRA2
WRA1	减	WRA1—WRA2
WRA1	乘	WRA1*WRA2
WRA1	除	WRA1 / WRA2

图 1.8　用于算术子程序的寄存器分配

数学子程序存在一个问题，即它必须执行算术运算，但是它们不能识别出给予的自变量的据数类型。为解决这个问题，在通讯区中已经保留了另一个字节，以指明 WRA1 中变量的数据类型，这个单元称为类型标志，它的地址是 40AFH，并含有指明 WRA1 当前内容的数据类型的代码。它的代码是：

代　码	数据类型
02	整　数
03	字符串
04	单精度
08	双精度

数学子程序通常不要求自变量为特定的数据类型，但是有一些例外(详细请看第二章)。

第五部分——I/O 驱动程序

驱动程序提供了操作指定设备所必需的基本功能的能力。Level II ROM 内含有键盘、显示器、并行打印机和盒式录音机的输入/输出 (I/O) 驱动程序。磁盘驱动程序是 DOS 系统的一部分，因此，将不予讨论。

Level II 支持的所有设备，除盒式录音机以外，都需要设备控制块 (DCB)。驱动程序使用 DCB 来监视变化的信息，如显示器上的光标位置和打印机的行计数。显示器、键盘和打印机的 DCB 都是 Level II ROM 的一部分。由于必须将信息存入其中，所以在 IPL

过程中，将它们从 ROM 移到 RAM 中的固定单元(在通讯区中)。

对于要传输的每一个字符都必须调用 Level II 驱动程序。驱动程序不能应付记录或文件的概念，因此把所有记录的组合和分解留给了用户。Level II 没有通用的记录管理实用程序。对于 BASIC 程序，必须用子程序，如 PRINT 和 INPUT，去组合每个记录。

例如，当往盒式录音机写入时，PRINT 子程序产生一个引导码，它由 255 个零后边跟一个 A5H 所组成。在引导码写完以后，以 ASCII 字符串的形式写每个单独的变量，每个变量之间是空格，最后用回车结束。非字符串变量都被转换成它们的 ASCII 等效值。

INPUT 操作从搜索引导码的 255 个字节的零开始，然后跳过 A5H，将所有变量读入行缓冲器直到发现回车为止。当 INPUT 完成后，将所有变量转换成它们的正确格式并送入 VLT。

键盘、显示器和行印机的驱动程序可以直接进入，也可以通过通用驱动程序入口点 03C2H 进入。对于为每一个驱动程序规定的调用序列，将在第二章给出。

盒式录音机驱动程序有几个方面与其他驱动程序不同。它以串行位方式进行 I/O，而所有的其他驱动程序是以字节(或字符)方式工作的。这意味着盒式机驱动程序必须按照一位一位的方式传输数据。每一位的传输是非常复杂的，并且包含许多步骤。因为涉及定时，所以在磁盘系统中，盒式机的 I/O 必须在时钟关闭(禁止中断)的情况下进行。关于盒式机 I/O 的更详细说明，见第四章。

第六部分——系统实用程序

在 Level II ROM 中，系统实用程序是一些直接的命令：AUTO，CLEAR，CSAVE，CLOAD，CONT，DELETE，EDIT，LIST，NEW，RUN，SYSTEM，TROFF 和 TRON。这些命令可以和 BASIC 程序语句混合在一起。但是，它们是被立即执行的，而不存入程序语句表 (PST) 中。在执行直接命令以后，控制返回到输入阶段。

在整个 BASIC 程序被输入以后(通过键盘或通过 CLOAD，或者在磁盘系统中通过 LOAD)，必须使用 RUN 命令来执行它。这个命令除了使 PST 中的 BASIC 程序执行以外(进入执行阶段)，与其他系统命令没有区别。如同使用其他系统命令一样，当 BASIC 程序结束时，控制返回到输入阶段。

IPL——初始程序装入程序

IPL 过程作为一般术语已经讨论过。下面对这过程进行全面的论述。说明被分成了磁盘系统和非磁盘系统两个单独的小节。

复位处理(无磁盘)

当按下复位按钮的时候，操作从绝对地址零开始，控制从那儿传送到 0674H，进行以下工作：

A) 从 FFH(255) 到 80H(128) 的所有通道都被预置成零。这使盒式机复位，并选择显示器为每行 64 个字符。

B) 将 06D2H 到 0707H 的代码移到 4000H到 4035H。这预置了在 08H，10H，

18H 以及 20H 的重新启动向量的地址，以跳转到它们在 Level II 中的正常工作单元。400CH 和 400FH 单元被预置成返回 (RET) 指令。

如果磁盘系统正在进行 IPL (初始程序装入)，则 400CH 和 400FH 将由 SYS0 在磁盘部分的 IPL 过程中被改成跳转到相应地址的 JP 指令。键盘、显示器和行印机的 DCB 从 ROM 移到地址 4015H 到 402CH 的 RAM 中。在移完 DCB 以后，402DH，4030H，4032H 和 4033H 单元就被预置为无磁盘用法。如果磁盘系统正在进行 IPL，这些单元就将由 SYS0 进行修改。

C) 把 4036H 到 4062H 的存储器置成机器零 (00H)。

在存储器清零以后，控制被传送到 0075H 单元，在此进行以下工作：

A) 将除法支援程序从 18F7H—191DH 移到 4080H—40A6H。这个范围也包括程序语句表的地址指针。单元 41E5H 被预置成：

LD A, (2C00H)

B) 把扫描子程序的输入缓冲区地址置成 41E8H。这将是输入阶段中用于存储接收到的每一行的缓冲区域。

C) 把磁盘 BASIC 入口向量 4152H—41A5H 预置成跳转到 012DH 的 JP 指令。如果程序使用任何磁盘 BASIC 的功能，那末就引起 L3 错误。接着，将单元 41A6H—41E2H (DOS 出口)置成返回 (RET) 指令。41E8H 置成零，当前堆栈指针 (CSP) 置成 41F8H (在这儿我们需要堆栈是因为在余下的 IPL 过程中将执行 CALL 语句，它需要堆栈来存储返回地址)。

D) 调用在 1B8FH 的子程序。首先它将堆栈重新置成 434CH，并将 40E8H 预置成 (40A0H) — 2，然后将文字串库表预置成空表；将当前输出设备设置为显示器，清除打印缓冲区并关闭盒式录音机；将 FOR 语句标志置成零；将零作为第一个值存入堆栈，并将控制返回到 00B2H。

E) 清除荧光屏，显示出 'MEMORY SIZE'。接着，接受用户的回答并对它进行测试，然后存入 40B1H；分配 50 个字节的内存作字符串区域，并将其低端地址存入 40A0H。

F) 调用 1B4DH 的另一个子程序，关闭跟踪功能；预置简单变量的起始地址 (在 40F9H 中) 和程序语句表起始地址 (在 40A4H 中)。对于所有的变量，把变量类型表 4101H—411AH 置成单精度，并执行 RESTORE。最后，控制返回到 00FCH。

G) 在 00FCH，显示出 'RADIO SHACK LEVEL II BASIC' 信息，并将控制传送到输入阶段。

复位处理(磁盘系统)

这个情况下的操作从单元 0000H 开始并立即跳转到 0674H。在 A) B) C) 等小节描述的复位处理(无磁盘系统)的操作代码，对两个 IPL 过程都是共同的。在 C) 小节描述的过程完成以后，进行确定系统中有无磁盘的测试。如果没有外接磁盘驱动器，那末控制转移到 0075H，否则执行如下的工作：

A) 选择 0 号磁盘驱动器，并定位在 0 磁道 0 扇区。从这个位置上把扇区装入程序 (BOOT/SYS) 读入 RAM 的 4200H—4455H 单元。因为扇区装入程序是以绝对格式写

的，所以在读入完成以后可以立即执行。

在读入完成以后，控制就被传送到扇区装入程序，它将磁盘定位在 11 磁道 4 扇区。于是这个扇区就被读入到 4D00H 的内部缓冲区。扇区读入包括开头 32 个字节中的 SYS0 的目录项。利用这个数据，扇区装入程序计算出 SYS0 的磁道和扇区地址，并将其第一扇区读入 4D00H。

B） 在读入以后，分解二进制数据并将它移到 RAM 中指定地址的区域中。注意，SYS0 不是以绝对格式写的，所以它不能直接读入内存和执行。必须用扇区装入程序将它译码和移动。一旦做完这项工作，控制就被传送到 4200H 地址开始的 SYS0。

C） 下面对 SYS0 的描述只适用于 NEWDOS 系统。它由检测 RAM 存储器的容量开始，并将它本身的键盘驱动程序地址存入 4015H 的键盘 DCB 中。时钟中断向量地址（4012H）被预置成 CALL 4518H。接着，预置更多的地址，并显示出 NEWDOS 题头信息。

D） 在显示出题头以后，测试键盘上键入的回车。如果发现回车，就跳过 AUTO 过程的测试，将控制立即传送到 4400H，在此预置 DOS 的输入扫描程序（SCANNER）。

假设没有检测到回车，则读 gr. 分配表(GAT)扇区（11 磁道 0 扇区），并测试第 E0H 字节是否为回车值。如果发现回车(缺项情况)，控制就转移到 4400H，否则显示出由字节 E0H 开始的 GAT 扇区的 20 个字节的信息。然后，控制被传送到 4405H，并在此启动 AUTO 过程。在执行 AUTO 过程以后，控制将被传送 4400H 开始的 DOS 输入阶段。

磁盘 BASIC

有一个 DOS 命令是称为 BASIC 的实用程序。它除了提供把控制从 DOS 传送到 Level II 的手段外，还包含解释和执行下列磁盘 BASIC 语句的程序。

适用于 TRSDOS 和 NEWDOS 的语句：

CVI	CVS	CVD	MKI$	MKS$	DEFFN
DEFUSR	TIME$	CLOSE	FIELD	GET	PUT
AS	LOAD	SAVE	KILL	MERGE	NAME
LSET	RSET	INSTR	LINE	&H	&O
CMD“S”	CMD“T”	CMD“R”	CMD“D”	CMD“A”	
USR0—USR9		MKD$	MID$（等式左边）		
OPEN“R”	OPEN“O”	OPEN“I”			

只适用于 NEWDOS 的语句为：

OPEN“E” RENUM REF CMD“E” CMD“DOS 命令”

仅 TRSDOS 有的附加的特殊命令*有：

CMD“X”，〈ENTER〉——2.1 版本

CMD“#”〈ENTER〉——2.2 和 2.3 版本

这些隐藏的并且不提供资料的命令，显示出由 Microsoft 公司所作的“秘密”版权标记。CMD“A” 也是不提供资料的，它完成的功能与 CMD“S” 相同。

---

* 经试验，CMD“X” 和 CMD“#” 无效。——译者

磁盘 BASIC 是作为 Level II 的扩充部分运行的，在装入以后，它预置以下通讯区的区段：

1. DOS 出口 41A6H—41E2H 由返回（RET）改变为跳转到磁盘 BASIC 实用程序中的一些单元。

2. 磁盘 BASIC 出口 4152H—41A3H 由 JP 012DH 的 L3 句法错误改变成跳转到磁盘 BASIC 中的工作子程序地址。

在预置通讯区以后，把 DCB 和三个磁盘文件的扇区缓冲器分配在磁盘 BASIC 程序的末尾。然后，控制转到 Level II 中的输入扫描程序（1A19H）。

要执行任何磁盘 BASIC 语句，或从 Level II 取得 DOS 出口的任何时候，就将重新进入磁盘 BASIC。进入磁盘 BASIC 的入口点就象进入工作程序一样。在完成以后，控制返回到执行驱动程序。

注意：磁盘 BASIC 占用内存单元 5200H—5BADH（NEWDOS 系统）。每个保留文件将需要额外的存储空间（32＋256 个字节）*。当汇编程序与 BASIC 程序一起运行时，必须注意不要扰乱了这一区域。

* 每个文件需要 32 字节的 FCB（文件控制块）和 256 字节的文件缓冲区。——译者

第二章 子 程 序

Level II 有许多有用的子程序，它们可以用于汇编语言程序。本章将介绍相当数量的子程序入口点。然而，Level II 所具有的子程序远比这里介绍的多。以提供的地址为指南，可以很容易地探明所有的按各自功能分别处理的 Level II 子程序。

在使用数学或算术子程序以前，应先了解工作寄存器的概念和类型标志(见第一章)。还要回忆一下除法支援程序(见第一章)，它只有在 IPL 一个非磁盘系统时才自动装入。而在磁盘系统中，它是由磁盘 BASIC 实用程序装入的。如果你正在使用磁盘系统和执行汇编语言程序，其中使用了需要除法的任何数学或算术子程序，那么，你必须首先进入 BASIC，或由你的程序去装入除法支援程序。

下面讲述的 I/O 调用序列只适用于 Level II。在 TRSDOS 和磁盘 BASIC 参考手册中有磁盘 I/O 的 DOS 调用序列。

系统子程序和 BASIC 功能有其使用范围，因而，它们不可能始终适用于写入汇编语言。但是，如果你想把汇编程序和 BASIC 程序混合在一起使用，你将发现这些子程序是非常有用的。

I/O 调用序列

在 TRS-80 I 型机中，输入和输出 (I/O) 操作都是直接的，内存映象或口 (或称通道)都是可寻址的。对于 I/O 操作，没有 DMA (直接存储器存取)命令，并且不使用中断处理。

这里选择的入口点不是应有尽有的。它只涉及比较通用的，并将给读者指出一正确方向，如果你需要，可去寻找更特殊的入口点。

在内存映象操作中，从内存单元存或取一个字节，就使数据在 CPU 寄存器和目标设

表 2.1

位	键盘地址							
	3801H	3802H	3804H	3808H	3810H	3820H	3840H	3880H
0	@	H	P	X	0	8	ENTER	SHIFT
1	A	I	Q	Y	1	9	CLEAR	
2	B	J	R	Z	2	:	BREAK	
3	C	K	S		3	;	↑	
4	D	L	T		4	,	↓	
5	E	M	U		5	—	←	
6	F	N	V		6	.	→	
7	G	O	W		7	/	SPACE	

备之间进行传送。内存映象设备的例子有显示器、键盘和磁盘。编程 I/O（通过口）是在寄存器和设备之间直接传送数据。使用口进行 I/O 的唯一设备是盒式录音机。

键盘输入

键盘的内存映象地址在 3800H—3BFFH，并分配如表 2.1 所示。

当按下一个键的时候，相应字节中的对应位就被置位，也清除了由前面的键所置位的那些位。你将发现表中只有 8 个字节（3801H—3880H）是有效的。这会使人误认为可以使用它们之间的那些字节，实际上并非如此，因为任何有效行的字节在所有无用字节中是重复的。因此，已使用了所有字节。

CALL 002BH　扫描键盘

进行键盘的瞬时扫描。如果没有按下键，控制就返回到调用它的程序，A 寄存器和标志寄存器被置零。如果任何键（BREAK 键除外）有动作，则在 A 寄存器中得到该字符的 ASCII 值。如果 BREAK 键动作，则执行系统请求代码为 01 的 RST 28H 指令，RST 指令产生的结果是跳转到 DOS 出口 400CH。在无磁盘系统中，就从该出口返回；而在磁盘系统中，控制被传送到 SYS0，且在那儿请求代码受到检查并被忽略，因为系统请求代码的位 7 必须是 1（置位）。在检查代码以后，控制返回到调用 002BH 的程序。在 002BH 检测到的字符是不显示的。本子程序使用 DE、标志和 A 寄存器。

```
;扫描键盘并测试 BREAK 键或星号。
        PUSH    DE          ;保存 DE。
        PUSH    IY          ;保存 IY。
        CALL    2BH         ;测试任何键的动作。
        DEC     A           ;键有动作，是 BREAK 吗?
        JR      M, NO       ;若没按键，则转移。
        JR      Z, BRK      ;若 BREAK 键动作，Z 标志置位。
        INC     A           ;A 恢复为原先的值。
        CP      2AH         ;测试＊键动作。
        JR      Z, AST      ;若＊键被按下，Z 标志置位。
        ...
```

CALL 0049H　等待键盘输入

当按下键盘上的任何键时，本子程序立即返回到调用它的程序，并在 A 寄存器中得到了该输入字符的 ASCII 值。本子程序将使用 A、标志和 DE 寄存器。

```
;等待键盘输入下一个字符并测试是否是字母。
        PUSH    DE          ;保存 DE 和 IY。
        PUSH    IY
        CALL    49H         ;等待，直到输入下一个字符。
        CP      41H         ;测试是否低于“A”。
        JR      NC, ALPHA   ;若是字母，则转移。
```

...

CALL 05D9H 等待下一行

接受键盘输入并将每个字符存入调用它的程序所提供的缓冲器中。输入要继续进行到打入回车键或 BREAK 键，或者直到缓冲区满了为止。它能识别所有的编辑控制代码，如 TAB，BACKSPACE 等。调用序列是：

```
;由键盘得到下一行。如果敲 BREAK 键，则退出。
;一行不能超过 25 个字符。
SIZE    EQU     25              ;每行中允许的最多字符数。
        LD      HL，BUFF        ;缓冲区地址。
        LD      B，SIZE         ;缓冲区大小。
        CALL    5D9H            ;从键盘读下一行。
        JR      C，BREAK        ;若按了 BREAK 键，就转移。
        ...
BUFF    DEFS    SIZE            ;行缓冲区。
        ...
```

在退出子程序时，寄存器含有：

HL 缓冲区地址；

B 传输的字符数，包括最后一个在内；

C 原缓冲区大小；

A 如果按下回车键或 BREAK 键，则含有接受到的最后一个字符；

进位标志 在用 BREAK 键作为结束时，C 标志置位，否则复位。

如果缓冲区满了，A 寄存器将含有缓冲区的容量。

显示器输出

显示器 I/O 是内存映象 I/O 的另一个例子。它使用的地址是 3C00H 到 3FFFH，其中，3C00H 代表显示器荧光屏的左上角，而 3FFFH 代表荧光屏的右下角。

屏幕控制代码，例如 TAB，CURSOR ON/OFF，BACKSPACE 等都由显示器驱动程序处理。显示器设备本身是不认识任何控制代码的。驱动程序可识别的代码及它们的相应动作是：

代 码	动 作
08H	退格并抹除字符。
0EH	开光标。
0FH	关光标。
17H	选择 32 字符/行。
18H	退回一个字符(左箭头)。
19H	往前跳一个字符(右箭头)。
1AH	往下跳一行(下箭头)。
1BH	往上跳一行(上箭头)。
1CH	光标回到原点，选择 64 字符/行。
1DH	将光标定位在当前行的开头。
1EH	从光标起抹除到行的末尾。
1FH	从光标起抹除到帧的末尾。

选择字符和行的大小(32 或 64 字符每行)是用寻址 FFH 口的屏幕控制器，并发送给它一个指定字符大小的功能字节来完成的。该字节的格式为：

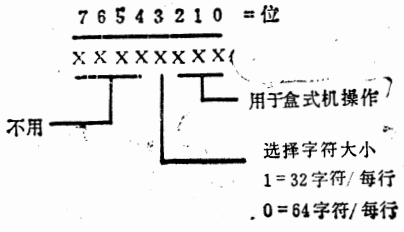


图 2.2

CALL 0033H　荧光屏显示

在荧光屏上显示 A 寄存器中的字符。控制代码都是有效的。本子程序将使用全部寄存器。

```
;在荧光屏上显示信息。
        LD      HL, LIST        ;信息地址。
LOOP    LD      A, (HL)         ;取下个字符。
        OR      A               ;测试信息的结束。
        JR      Z, DONE         ;若是信息结束,则转移到 DONE。
        PUSH    HL              ;不是结束,保存 HL。
        CALL    33H             ;显示出字符。
        POP     HL              ;恢复 HL。
        INC     HL              ;指向下个字符。
        JR      LOOP            ;循环,直到全部印出。
DONE    ...
LIST    DEFM    'THIS   IS  A   TEST'
        DEFB    0DH             ;回车。
        DEFB    0               ;信息结束的指示符。
```

CALL 01C9H　清除屏幕

清除荧光屏,选择每行 64 字符格式,光标回到原点。使用全部寄存器。

```
;清除屏幕,光标复原,选择 32 字符/行,跳四行。
        CALL    01C9H           ;清除屏幕。
        LD      A, 17H          ;选择 32 字符/行。
        CALL    33H             ;将字符大小的控制代码发送给显示器。
        LD      B, 4            ;要跳的行数。
        LD      A, 1AH          ;跳一行的代码。
LOOP    PUSH    BC              ;保存 BC。
        CALL    33H             ;跳一行。
        POP     BC              ;取回计数。
```

```
        DJNZ    LOOP            ;循环,直到跳过四行。
```

CALL 022CH　　闪烁星号

在荧光屏的右上角交替地显示和清除一个星号。本子程序使用全部寄存器。

```
;闪烁星号三次。
        LD      B, 3            ;闪烁的次数。
LOOP    PUSH    BC              ;保存计数。
        CALL    22CH            ;闪烁星号一次。
        POP     BC              ;取回计数。
        DJNZ    LOOP            ;计数闪烁了一次。
DONE    ...
```

打印机输出

打印机是内存映象设备的另一个例子。它的地址是 37E8H。将一个 ASCII 字符存入该地址时,就是把它送给了打印机。从这个地址取数,就得到了打印机状态。如果打印机是可用的,得到的状态为零状态;如果打印机是忙状态,则得到非零状态。

CALL 003BH　　打印字符

将 C 寄存器中含有的字符送往打印机。行计数由驱动程序保存在 DCB 中。当打满一页(66 行)时,行计数被复位,同时,送回给调用它的程序的状态寄存器被置成零。

打印机驱动程序能识别的控制代码是:

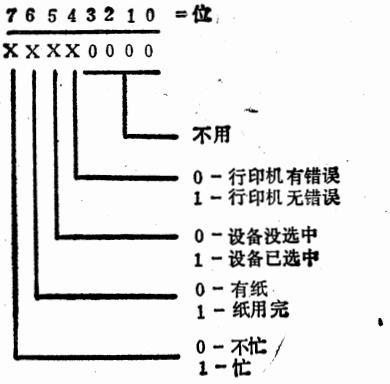
代　码	动　　作
00H	在 A 寄存器的高两位得到打印机状态,如果打印机不忙,则置状态为零;若忙,则置成非零。
0BH	无条件地跳到下页的第一行(换页)。
0CH	复位行计数 (DCB 4),并将其先前的值与每页的行数 (DCB 3) 进行比较。如果行计数为零,则不产生动作;如果行计数为非零,则换页。
0DH	行结束符。使行计数加 1 并测试是否满一页。通常是使打印机开始打印。

```
;在打印机上打印信息,如果在 1.5 秒内打印机没有就绪,就在显示器上显示错误信息。
        LD      HL, LIST        ;待打印行的地址。
START   LD      B, 5            ;准备测试打印机就绪否。
        PUSH    HL              ;保存 HL。
LOAD    LD      DE, 10H         ;装入延迟计数。
TST     CALL    05D1H           ;取打印机状态。
        JR      Z, RDY          ;若打印机就绪,转移。
        DEC     DE              ;没就绪,计数器减 1。
        LD      A, D            ;测试是否过了 1.5 秒。
```

```
        OR      E               ;首先 DE 必须等于 0。
        JR      NZ, TST         ;若 DE 不为 0，转移。
        DJNZ    LOAD            ;循环，直到过了 1.5 秒。
        JP      NTRDY           ;转移去显示'打印机未就绪'。
RDY     POP     HL              ;恢复待打印行地址。
        LD      A, (HL)         ;取下一个打印字符。
        OR      A               ;测试行的结束。
        JR      Z, DONE         ;如果行结束，转移。
        LD      C, A            ;把字符放入相应寄存器中。
        CALL    58DH            ;打印字符。
        INC     HL              ;指向下一个字符。
        JR      START           ;循环，直到打印完全部字符。
NTRDY   LD      HL, NTRDM       ;HL 等于未就绪信息的地址。
        CALL    VIDEO           ;显示信息。
DONE    ···                     ;在打印机上打印了一行。
LIST    DEFM    'THIS IS A TST'
        DEFB    0DH             ;启动打印机所需要的 CR (回车)。
        DEFB    0               ;信息结束标志。
NTRDM   DEFM    'PRINTER NOT READY'
        DEFB    0               ;结束显示信息。
```

CALL 05D1H　　取打印机状态

在状态寄存器中得到打印机的状态，若打印机已准备就绪则为零，否则为非零*。得到的其他状态位如下所示**：



在某些打印机上，无纸和忙状态位是随意的。

---

* 这里的零和非零是调用子程序的结果，即对 F 寄存器的影响：Z 标志置位或复位。以此可判别打印机是否已准备就绪了。而下面提到的其他状态位，是指调用子程序后在 A 寄存器中得到的打印机状态字。——译注

** 原文的位 4 和位 5 所表示的状态有误，因此，对此和下面的调用举例作了修正。——译注

```
;根据上列状态位监控打印机状态,并显示相应错误信息。
        LD      BC, 10H     ;打印机计时计数器。
START   PUSH    BC          ;保存计时计数。
        CALL    5D1H        ;取打印机状态。
        JR      Z, OK       ;若打印机已就绪,则转移。
        BIT     7, A        ;仍在打印吗?
        JR      Z, TIME     ;若位 7 不是零,则打印机忙,去计时。
        BIT     4, A        ;打印机不忙,是否有错?
        JR      NZ, NR      ;若有错误,则位 4 为零,硬件有问题。
        BIT     5, A        ;设备被选中否?
        JR      NZ, NS      ;若位 5 为零,则设备未选中。
;设备已选中、无错且不忙,假设无纸。
OP      LD      HL, OPM     ;显示无纸信息。
        ...
        JP      WAIT        ;转移去等待操作员的回答: 是再试一次还是停止。
NR      BIT     6, A        ;设备有错,测试是否无纸。
        JR      Z, OP       ;若无纸,则转移。
        LD      HL, NRM     ;显示未就绪信息。
        ...
        JP      WAIT        ;等待操作员的回答。
NS      LD      HL, NSM     ;显示没选中信息。
        ...
        JP      WAIT        ;等待操作员的回答。
TIME    POP     BC          ;取回计时计数。
        DEC     BC          ;计数循环了一次。
        PUSH    BC          ;保存新的计数值。
        LD      A, B        ;若计数已变成 0 ,
        OR      C           ;则已计时完毕。
        JR      NZ, START   ;循环,直到完成操作或计时完毕。
        LD      HL, TOM     ;显示计时完毕信息。
        ...
        JP      WAIT        ;等待操作员的回答。
        ...
```

盒式机 I/O

盒式机 I/O 不是内存映象的。在选择适当的设备后,盒式机通过 FFH 口进行寻址,并且 I/O 是一次完成一位,而所有其他设备进行 I/O 是以字节为单位的 (RS-232C 除外)。

因为一位一位地传输数据,所以定时的要求极其严格。当使用下列任何子程序时,应

禁止中断系统，以保证不发生中断而扰乱关键的输出定时。

CALL 0212H　设备选择和开启马达*

选择由 A 寄存器指定的磁带设备并开启盒式机马达。设备编号从 0 开始。本子程序使用全部寄存器。

```
LD      A, 1        ;选择盒式机 1 的代码。
CALL    0212H       ;选择设备 1 ,开马达。
...
```

CALL 0287H　写引导码**

在当前选中的设备上写 Level II 引导码。引导码由 255 个 00H，和它后面的一个 A5H 组成。该子程序使用 B 和 A 寄存器。

```
LD      A, 1        ;选择设备 1 的代码。
CALL    212H        ;选择设备并开马达。
CALL    287H        ;写引导码。
...
```

CALL 0296H　读引导码

读当前选中的设备，直到遇到引导码的结束（A5H）为止。当发现结束时，在显示器的右上角显示一个星号。此子程序使用 A 寄存器。

```
LD      A, 1        ;设备 1 的代码。
CALL    0212H       ;选择设备 1 ,开马达。
CALL    0296H       ;读引导码，在遇到 A5H 时返回。
...
```

CALL 0235H　读一个字节

从当前选中的设备上读入一个字节。在 A 寄存器中得到读入的字节。保存所有寄存器。

```
LD      A, 1        ;要选择的设备。
CALL    212H        ;选择设备，开马达。
CALL    296H        ;跳过引导码。
CALL    235H        ;读后边的字节。
CP      41H         ;测试文件名 A***。
```

---

* 原文遗漏关马达的子程序入口。此入口是 01F8H，当 CALL 01F8H 时，就关掉了马达。——译注

** 原文误为 0284H。根据下面的举例，应是 0287H，它必须考虑选择设备和开马达。而从 0284H 入口，只要先把设备号用"#-××,"形式存放在以 HL 为指针的缓冲区中就行。下面读引导码子程序也有个类似 0284H 的入口 0293H。——译注

*** 这是数据文件而不是程序。关于两者的磁带记录格式，请参考第三章。——译注

```
        JR      Z, YES      ;若是文件 A，则转移。
        ...
```

CALL 0264H　写一个字节

把 A 寄存器中的一个字节写到当前选中的设备上。保存全部寄存器。

```
        LD      A, 1        ;选择设备号。
        CALL    0212H       ;选择设备,开马达。
        CALL    0287H       ;写引导码 (255 个 00H 和 A5H)。
        LD      A, 41H      ;文件名 A。
        CALL    0264H       ;在引导码后写 A。
        ...
```

转换子程序

这些入口点用于将二进制值从一种数据类型或模式转换成另一种，例如，由整数转换成浮点数，和在 ASCII 与二进制表示之间进行转换。这些转换子程序假设要转换的值在 WRA1 中，同时，类型标志 (40AFH) 反映着当前数据类型。结果将留在 WRA1 中，同时，将改变类型标志。

数据类型转换

CALL 0A7FH　浮点数转换成整数*

把 WRA1 中的内容由单精度或双精度数转换成整数。转换中不进行舍入。使用全部寄存器。

;将单精度值转换成整数并把结果送入 IVAL 中。

```
        LD      HL, 4121H     ;WRA1 中 LSB 地址。
        LD      DE, VALUE     ;单精度数的 LSB 地址。
        LD      BC, 4         ;要传送的字节数。
        LDIR                  ;把值送入 WRA1。
        LD      A, 4          ;单精度的类型码。
        LD      (40AFH), A    ;置单精度类型。
        CALL    0A7FH         ;单精度值转换成整数。
        LD      A, (4121H)    ;整数的 LSB。
        LD      (IVAL), A     ;存入整数的单元。
        LD      A,(4122H)     ;整数的 MSB。
        LD      (IVAL + 1),A  ;存入整数的单元。
        ...
VALUE   DEFB    0EH           ; 502.778 (单精度数)的 LSB。
        DEFB    B6H           ; NLSB
```

* 此入口就是 CINT 函数的入口。——译注

```
        DEFB    00H             ; MSB
        DEFB    88H             ;指数。
IVAL    DEFB    0               ;单精度数 502.778 的整数,将存放在这两个单元
        DEFB    0               ;中。
        ...
```

CALL 0AB1H　整数转换成单精度数*

将 WRA1 的内容由整数或双精度数转换成单精度数。子程序使用全部寄存器。

```
;把整数转换成单精度数并送入数值区域。
        LD      A, 59H          ;整数 26457 的 LSB。
        LD      (4121H), A      ;送入 WRA1。
        LD      A, 67H          ;整数 26457 的 MSB。
        LD      (4122H), A
        LD      A, 2            ;整数的类型代码。
        LD      (40AFH), A      ;设置整数类型。
        CALL    0AB1H           ;把整数转换成单精度。
        LD      HL, VALUE       ;数值存放区域的地址。
        CALL    09CBH           ;把单精度值由 WRA1 送入数值区域。
        ...
VALUE   DEFS    4               ;用于存放单精度 26457。
        ...
```

CALL 0ADBH　整数转换成双精度数

将 WRA1 的内容由整数或单精度转换成双精度数。这个子程序要使用全部寄存器。

```
        LD      A, 59H          ;整数 26457 的 LSB。
        LD      (4121H), A
        LD      A, 67H          ;26457 的 MSB。
        LD      (4122H), A      ;送入 WRA1。
        LD      A, 2            ;整数的类型代码。
        LD      (40AFH), A      ;置整数类型。
        CALL    0ADBH           ;把整数转换成双精度。
        LD      DE, VALUE       ;把双精度数从 WRA1 送
        LD      HL, 411DH       ;入存放的区域。
        LD      BC, 8           ;传送的字节数。
        LDIR                    ;传送数值。
        ...
VALUE   DEFS    8               ;存放 26457 的双精度值。
```

* 此入口是 CSNG 函数的入口。因此不但可把整数转换成单精度(实际由 0ACCH 入口),还可把双精度转换成单精度(实际由 0AB9H 入口)。——译注

...

ASCII 转换成数值

下列入口点用于二进制和 ASCII 之间的转换。当从 ASCII 转换成二进制时，假设 HL 寄存器对中存放有 ASCII 串的地址。结果将留在 WRA1 中或 DE 寄存器对中，而类型标志将相应改变。

CALL 1E5AH ASCII 转换成整数

把以 HL 为指针的 ASCII 串转换成它的整型等价值。结果将留在 DE 寄存器对中。转换将在发现第一个非数字字符时停止。

```
        LD      HL, AVAL      ; HL = ASCII 数字的地址。
        CALL    1E5AH         ;把它转换成二进制。
        LD      (BVAL), DE    ;存储二进制值。
        ...
AVAL    DEFM    '26457'       ; ASCII 值 26457。
        DEFB    0             ;非数字停止字节。
BVAL    DEFW    2             ;存放二进制值 26457。
        ...
```

CALL 0E6CH ASCII 转换成二进制值

把以 HL 为指针的 ASCII 串转换成二进制值。如果该值小于 2 的 16 次方 (2^{16})，并且不含有小数点，或者不含有 E 或 D 描述符(指数)，那末 ASCII 串将被转换成它的整数等价值。如果 ASCII 串含有小数点，或者含有 E 或 D 描述符，或者它超过 2^{16}，那末它将被转换成单精度或双精度值。二进制值将留在 WRA1 中，类型标志将置成相应的值。

```
        LD      HL, AVAL      ; ASCII 数字。
        CALL    0E6CH         ;把 ASCII 转换成二进制。
        ...
AVAL    DEFM    '26457'       ;要转换的 ASCII 值。
        DEFB    0             ;非数字停止字节。
        ...
```

CALL 0E65H ASCII 转换成双精度数

把以 HL 为指针的 ASCII 串转换成双精度等价值。该子程序使用全部寄存器。结果将留在 WRA1 中。

```
        LD      HL, AVAL      ;要转换的 ASCII 值的地址。
        CALL    0E65H         ;把值转换成双精度数。
        LD      DE, BVAL      ;然后把数值从 WRA1 送入
        LD      HL, 411DH     ;数值存放区域。
        LD      BC, 8         ;传送的字节数。
        LDIR                  ;把双精度数送入存放区域。
```

```
        ...
AVAL    DEFM    '26457'     ;被转换的 ASCII 值。
        DEFB    0           ;非数字停止字节。
BVAL    DEFS    8           ;存放二进制等价值的区域。
        ...
```

二进制转换成 ASCII 表示

下面一组入口点用于将二进制数转换成为 ASCII 码表示的数。

CALL 0FAFH　把 HL 的内容转换成 ASCII 并显示

把 HL 寄存器对里的值(假设为整数)转换成 ASCII 码，并显示在荧光屏的当前光标位置上。子程序要使用全部寄存器。

```
        LD      HL, 64B8H   ; HL = 25784
        CALL    0FAFH       ;转换成 ASCII 并显示。
        ...
```

CALL 132FH　整数转换成 ASCII

将 WRA1 中的整数转换成 ASCII，并将 ASCII 串存入以 HL 寄存器对作指针的缓冲区内。在入口时，B 和 C 寄存器都应含有 5，以避免逗号或小数点出现在 ASCII 串内。保存全部寄存器。

```
        LD      HL, 500     ;把 500 送入 WRA1。
        LD      (4121H), HL
        LD      BC, 505H    ;抑制逗号或小数点。
        LD      HL, BUFF    ; ASCII 串的缓冲区地址。
        CALL    132FH       ;将 WRA1 中的值转换成 ASCII 并存入缓冲区。
        ...
BUFF    DEFS    5           ; ASCII 值的缓冲区。
        ...
```

CALL 0FBEH　浮点数转换成 ASCII*

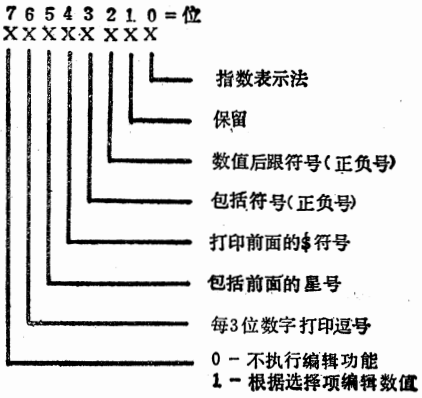
把 WRA1 中的单精度或双精度数转换成它的 ASCII 等价值。ASCII 值被存储在以 HL 寄存器对作指针的缓冲区中。由于把数值从二进制转换成 ASCII，所以它被格式化了，就像使用 PRINT USING 语句时它应该具有的格式那样。格式的模式可以指定，选择的方法是将下列值装入 A，B 和 C 寄存器中。

寄存器 A = 0　不编辑，严格地由二进制转换成 ASCII。

A = X　其中，X 解释如下：

---

* 这个子程序也可以从 0FBDH 入口，这时子程序自动把 A 清零，而不必如 0FBEH 入口时用 LD A,0 指令清零 A 寄存器。——译者



寄存器B等于小数点左边的数字位数。

寄存器C等于小数点后边的数字位数。

```
        LD      HL, AVAL1     ;要转换的 ASCII 值。
        CALL    0E6CH         ;把 ASCII 值转换成二进制值。
        LD      HL, AVAL2     ;转换值的缓冲区地址。
        LD      A, 0          ;表示不编辑。
        CALL    0FBEH         ;将单精度值反转换成 ASCII。
        ...
AVAL1   DEFM    '1103.25'     ;原始的 ASCII 值。
        DEFB    0             ;非数字停止字节。
AVAL2   DEFS    7             ;存放转换后的值。
        ...
```

算术子程序

在两个相同类型的操作数之间，这些子程序执行算术运算。它们假设操作数已装入相应的 CPU 寄存器或工作寄存器区中，并假设将数据类型或模式设置成相应的值。这些子程序中，有些还可能用到除法支援程序(见第一章的说明)。

整数算术子程序

下列程序在 DE 和 HL 寄存器对中进行整数值之间的算术运算。DE 中原来的内容总是保存着，且运算的结果总是留在 HL 寄存器对中。

CALL 0BD2H　　整数加

将 DE 中的整数值与 HL 中的整数相加。其和留在 HL 中，DE 中原来的内容保持不变。如果产生溢出(其和超过 2^{15})，那末先把两个值都转换成单精度值，然后相加。其结果将留在 WRA1 中，同时，类型标志将被改变。

```
        LD      A, 2          ;整数的类型代码。
        LD      (40AFH), A    ;设置整数类型。
```

```
        LD      HL, (VAL1)   ;装入第一个值。
        LD      DE, (VAL2)   ;装入第二个值。
        CALL    0BD2H        ;相加，HL = HL + DE。
        LD      A, (40AFH)   ;测试溢出。
        CP      2            ;类型是否是整数?
        JR      NZ, ···      ;若Z标志不置位，则其和是单精度数，否则是整
                               数。
        ···
VAL1    DEFW    25
VAL2    DEFW    20
        ···
```

CALL 0BC7H　整数减*

从 DE 的值中减去 HL 的值。差值留在 HL 寄存器对中，DE 保持不变。在产生下溢时，把两个值都转换成单精度值，并重新进行减，其结果留在 WRA1 中，类型标志作相应改变。

```
        LD      A, 2         ;整数的类型代码。
        LD      (40AFH), A   ;设置整型。
        LD      HL, (VAL1)   ;值1。
        LD      DE, (VAL2)   ;值2。
        CALL    0BC7H        ; DE 减 HL。
        LD      A, (40AFH)   ;取类型标志。
        CP      2            ;测试下溢。
        JR      NZ, ···      ;不为零，则下溢。
        ···
VAL1    DEFW    25
VAL2    DEFW    20
        ···
```

CALL 0BF2H　整数乘

DE 乘以 HL，乘积留在 HL 中，DE 保持不变。如果发生溢出，两值被转换成单精度值并重新进行运算，乘积将留在 WRA1 中。

```
        LD      A, 2         ;整数的类型代码。
        LD      (40AFH), A   ;设置整数型。
        LD      HL, (VAL1)   ;装入第一个值。
        LD      DE, (VAL2)   ;装入第二个值。
        CALL    0BF2H        ; HL = HL * DE
```

---

* 原文误为 HL 减 DE。——译注

```
        LD      A, (40AFH)   ;取类型标志。
        CP      2            ;测试溢出。
        JR      NZ, ···      ;若Z标志不置位,则为溢出。
        ···
VAL1    DEFW    25
        DEFW    20
        ···
```

CALL 2490H　整数除

DE 除以 HL。两值在开始除以前先转换成单精度值，商留在 WRA1 中，并改变类型标志。DE 和 HL 寄存器组的原来的内容被破坏。

```
        LD      DE, (VAL1)   ;装入值1。
        LD      HL, (VAL2)   ;装入值2。
        CALL    2490H        ; DE 除以 HL，商在 WRA1 中。
        ···
VAL1    DEFW    50
VAL2    DEFW    20
        ···
```

CALL 0A39H　整数比较*

把 DE 和 HL 中的两个整数值作代数比较。DE 和 HL 的内容原封不动，比较的结果留在A寄存器中,并且状态寄存器为:

运　算	A 寄存器
DE>HL	A=−1
DE<HL	A=+1
DE=HL	A=0

```
        LD      DE, (VAL1)   ; DE 和 HL 中是要比较
        LD      HL, (VAL2)   ;的值。
        CALL    0A39H        ;比较 DE 和 HL。
        JR      Z, ···       ;若为0，则 DE = HL。
        JP      P, ···       ;若为正,则 DE < HL。
        ···
```

单精度算术子程序

下面一组入口点用于单精度运算。这些子程序要求一个参量在 BC 和 DE 寄存器中，另一个参量在 WRA1 中。

CALL 0716H　单精度加

BC 和 DE 中的单精度值加 WRA1 中的单精度值,和留在 WRA1 中。

---

* 进行比较的两个量必须符号相同。下面的单精度及双精度比较也要求如此。——译注

```
        LD      HL, VAL1      ;一个单精度值的地址。
        CALL    9B1H          ;把单精度值送入 WRA1。
        LD      HL, VAL2      ;另一个值的地址。
        CALL    9C2H          ;把另一个值送入 BC 和 DE 寄存器中。
        CALL    716H          ;值 1 加值 2，和在 WRA1 中。
        ...
VAL1    DEFS    4             ;存放单精度值。
VAL2    DEFS    4             ;存放单精度值。
        ...
```

CALL 0713H 单精度减*

从 BC 和 DE 中的单精度值中减去 WRA1 中的单精度值，差留在 WRA1 中。

```
        LD      HL, VAL1      ;单精度值 1 的地址。
        CALL    9B1H          ;把值送入 WRA1。
        LD      HL, VAL2      ;单精度值 2 的地址。
        CALL    9C2H          ;把值送入 BC 和 DE。
        CALL    713H          ; BC DE 减 WRA1，差留在 WRA1 中。
        ...
VAL1    DEFS    4             ;存放单精度值。
VAL2    DEFS    4             ;存放单精度值。
        ...
```

CALL 0847H 单精度乘

BC 和 DE 中的值乘以 WRA1 中的当前值，乘积留在 WRA1 中。

```
        LD      HL, VAL1      ;第一个单精度值的地址。
        CALL    9B1H          ;把它送入 WRA1。
        LD      HL, VAL2      ;第二个单精度值的地址。
        CALL    9C2H          ;把它送入 BC 和 DE。
        CALL    847H          ;相乘，乘积留在 WRA1。
        ...
VAL1    DEFS    4             ;存放单精度值。
VAL2    DEFS    4             ;存放单精度值。
        ...
```

CALL 08A2H 单精度除

BC 和 DE 中的单精度值除以 WRA1 中的单精度值，商留在 WRA1 中。

```
        LD      HL, VAL1      ;除数的地址。
        CALL    9B1H          ;把除数送入 WRA1。
```

---

* 原文误为 WRA1 减 BCDE。——译注

```
        LD      HL，VAL2        ;被除数的地址。
        CALL    9C2H            ;把被除数送入 BCDE。
        CALL    08A2H           ; BCDE除以 WRA1，商在 WRA1 中。
        ...
VAL1    DEFS    4               ;存放除数。
VAL2    DEFS    4               ;存放被除数。
        ...
```

CALL 0A0CH 单精度比较

将 BC 和 DE 中的单精度值与 WRA1 中的单精度值进行代数比较，比较的结果在A寄存器中，状态为：

运 算	A 寄存器
BCDE>WRA1	A=−1
BCDE<WRA1	A=+1
BCDE=WRA1	A=0

```
        LD      HL，VAL1        ;要比较的一个值的地址。
        CALL    09B1H           ;把值送入 WRA1。
        LD      HL，VAL2        ;要比较的第二个值的地址。
        CALL    09C2H           ;把值送入 BC 和 DE。
        CALL    0A0CH           ;将 BC 和 DE 与 WRA1 比较。
        JR      Z，...          ;如果 BCDE = WRA1，则Z标志置位。
        JP      P，...          ;如果 BCDE < WRA1，则P标志置位。
        ...
VAL1    DEFS    4               ;存放单精度值。
VAL2    DEFS    4               ;存放单精度值。
        ...
```

双精度算术子程序

下面一组子程序进行两个双精度运算量之间的运算。假设一个运算量在 WRA1 中，而另一个运算量在 WRA2 (4127H—412EH) 中，结果总留在 WRA1 中。

CALL 0C77H 双精度加

将 WRA1 中的双精度值与 WRA2 中的双精度值相加，其和留在 WRA1 中。

```
        LD      A，8            ; DP (双精度)的类型代码。
        LD      (40AFH)，A      ;设置 DP 类型。
        LD      DE，VAL1        ;第一个 DP 值的地址。
        LD      HL，411DH       ; WRA1 的地址。
        CALL    9D3H            ;将第一个 DP 值送入 WRA1。
        LD      DE，VAL2        ;第二个 DP 值的地址。
```

```
        LD      HL, 4127H     ; WRA2 的地址。
        CALL    9D3H          ;将第二个值送入 WRA2。
        CALL    0C77H         ; WRA2 加 WRA1，和在 WRA1 中。
        ...
VAL1    DEFS    8             ;存放双精度值。
VAL2    DEFS    8             ;存放双精度值。
        ...
```

CALL 0C70H　双精度减

从 WRA1 中的双精度值中减去 WRA2 中的双精度值，差留在 WRA1 中。

```
        LD      A, 8          ; DP (双精度)值类型码。
        LD      (40AFH), A    ;置 DP 类型。
        LD      DE, VAL1      ;第一个 DP 值的地址。
        LD      HL, 411DH     ; WRA1 的地址。
        CALL    9D3H          ;将第一个 DP 值送入 WRA1。
        LD      DE, VAL2      ;第二个 DP 值的地址。
        LD      HL, 4127H     ; WRA2 的地址。
        CALL    9D3H          ;将第二个 DP 值送入 WRA2。
        CALL    0C70H         ;从 WRA1 中减去 WRA2，差在 WRA1 中。
        ...
VAL1    DEFS    8             ;存放双精度值。
VAL2    DEFS    8             ;存放双精度值。
        ...
```

CALL 0DA1H　双精度乘

WRA1 中的双精度值乘以 WRA2 中的值，乘积留在 WRA1 中。

```
        LD      A, 8          ; DP (双精度)值类型码。
        LD      (40AFH), A    ;置 DP 数据类型。
        LD      DE, VAL1      ;第一个 DP 值的地址。
        LD      HL, 411DH     ; WRA1 的地址。
        CALL    9D3H          ;将 DP 值 1 送入 WRA1。
        LD      DE, VAL2      ;第二个 DP 值的地址。
        LD      HL, 4127H     ; WRA2 的地址。
        CALL    9D3H          ;将 DP 值 2 送入 WRA2。
        CALL    0DA1H         ; WRA1 乘以 WRA2，乘积在 WRA1 中。
        ...
VAL1    DEFS    8             ;存放双精度值。
VAL2    DEFS    8             ;存放双精度值。
        ...
```

CALL 0DE5H　双精度除

将 WRA1 中的双精度值除以 WRA2 中的双精度值，商留在 WRA1 中。

```
        LD      A, 8            ; DP (双精度)值类型码。
        LD      (40AFH), A      ;置 DP 类型。
        LD      DE, VAL1        ;第一个 DP 值的地址。
        LD      HL, 411DH       ; WRA1 的地址。
        CALL    9D3H            ;将第一个 DP 值送入 WRA1。
        LD      DE, VAL2        ;第二个 DP 值的地址。
        LD      HL, 4127H       ; WRA2 的地址。
        CALL    9D3H            ;将第二个 DP 值送入 WRA2。
        CALL    0DE5H           ; WRA1 除以 WRA2，商留在 WRA1。
        ...
VAL1    DEFS    8               ;存放双精度值 1。
VAL2    DEFS    8               ;存放双精度值 2。
        ...
```

CALL 0A78H　双精度比较

将 WRA1 中的双精度值与 WRA2 中的双精度值进行比较。两个寄存器区域中的内容保持不变，比较的结果留在A寄存器中，状态寄存器为

运　算	A寄存器
WRA1>WRA2	A=−1
WRA1<WRA2	A=+1
WRA1=WRA2	A=0

```
        LD      A, 8            ; DP (双精度)值类型码。
        LD      (40AFH), A      ;置 DP 类型标志。
        LD      DE, VAL1        ;第一个 DP 值的地址。
        LD      HL, 411DH       ; WRA1 的地址。
        CALL    9D3H            ;第一个 DP 值送入 WRA1。
        LD      DE, VAL2        ;第二个 DP 值的地址。
        LD      HL, 4127H       ; WRA2 的地址。
        CALL    9D3H            ;第二个 DP 值送入 WRA2。
        CALL    0A78H           ;比较 WRA1 和 WRA2。
        JR      Z, ···          ;若两者相等，则为零。
        JP      P, ···          ;若 WRA1 < WRA2，则为正。
        ...
```

数学子程序

对下列所有子程序，假设 40AFH 单元中含有变量的数据类型或模式，例如，整型、单

精度型或双精度型，而变量本身在工作寄存器区 1（WRA1）中。浮点除法支援程序也必须装入在 4080H—40A6H 中。

CALL 0977H　绝对值　ABS(N)

将工作寄存器区 1（WRA1）中的值转换成它的正等价值。其结果留在 WRA1 中。如果遇到大于 2 的 15 次方的负整数，那么它就被转换成单精度值。数据类型或模式标志（40AFH）将相应改变，以反映类型上的任何变化。

```
        LD      A, 4            ;单精度的类型码。
        LD      (40AFH),A       ;置单精度类型。
        LD      HL, VAL1        ;求绝对值的单精度值的地址。
        CALL    09B1H           ;将单精度值送入 WRA1。
        CALL    0977H           ;求绝对值。
        ...
VAL1    DEFB    58H             ;单精度值 81.6022。
        DEFB    34H
        DEFB    23H
        DEFB    87H
        ...
```

CALL 0B37H　取整　INT (N)

求得浮点数的整数部分。如果数值是正的，则取其整数部分。如果数值是有小数部分的负数，则在截尾前先四舍五入。整数部分将留在 WRA1 中，类型标志被改变。

```
        LD      A, 4            ;单精度类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, VAL1        ;单精度值的地址。
        CALL    0B37H           ;分离单精度值的整数部分。
        LD      DE, 4121H       ; WRA1的地址(单精度值的整数部分)。
        LD      HL, VAL2        ;整数值的存放地址。
        CALL    09D3H           ;将取整后的单精度值送入存放区域。
        ...
VAL1    DEFB    0E0H            ;单精度值 -41.3418。
        DEFB    5DH
        DEFB    0A5H
        DEFB    86H
VAL2    DEFS    4               ;存放 -41.3418 的整数部分。
        ...
```

CALL 15BDH　反正切　ATN (N)

求 WRA1 中与浮点正切值相应的角度，并以弧度表示。角度值将以单精度值留在

WRA1 中。

```
        LD      A, 4            ;单精度的类型码。
        LD      (40AFH), A      ;设置单精度类型。
        LD      HL, TAN         ;正切值的地址。
        CALL    9B1H            ;将正切值送入 WRA1。
        CALL    15BDH           ;求角度,以弧度表示。
        LD      HL, ANGL        ;存放角度的地址。
        LD      DE, 4121H       ; WRA1 的地址。
        CALL    9D3H            ;把角度从 WRA1 送入存放区域。
        ...
TAN     DEFB    9AH             ; 30 度的正切值。
        DEFB    0C4H
        DEFB    13H
        DEFB    80H             ;正切值的指数。
ANGL    DEFS    4               ;存放 30 度值的地方，以弧度 (0.5235) 表示。
```

CALL 1541H　余弦　COS (N)

计算一个以弧度为单位的角度的余弦。角度必须是浮点值,求得的余弦将在 WRA1 中,以浮点数表示。

```
        LD      A, 4            ;单精度的类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, ANGL        ;角度值的地址。
        CALL    09B1H           ;将角度值送入 WRA1。
        CALL    1541H           ;计算余弦。
        LD      HL, CANGL       ;存放余弦值的地址。
        LD      DE, 4121H       ; WRA1 的地址。
        CALL    09D3H           ;把余弦值从 WRA1 送入存放区域。
        ...
ANGL    DEFB    18H             ;以弧度表示的 30° 角 (0.5235)。
        DEFB    04H
        DEFB    06H
        DEFB    80H             ;指数。
CANGL   DEFS    4               ;将存放 30 度的余弦值。
        ...
```

CALL 1439H　自然数的乘幂　EXP(N)

以 WRA1 中的值为指数，求 e (自然数为底)的乘幂。WRA1 中的值必须是单精度值。得到的结果将以单精度值放在 WRA1 中。

```
        LD      A, 4            ;单精度的类型码。
```

```
        LD      (40AFH), A   ;置单精度类型。
        LD      HL, EXP      ;指数的地址。
        CALL    09B1H        ;把指数送入 WRA1。
        CALL    1439H        ;求 E↑1.5708。
        LD      DE, 4121H    ; WRA1 的地址。
        LD      HL, POW      ;存放地址。
        CALL    09D3H        ;将乘幂送入存放区域。
        ...
EXP     DEFB    0DBH         ;单精度值 1.5708。
        DEFB    00FH
        DEFB    049H
        DEFB    081H
POW     DEFS    4            ;存放 E↑1.5708。
        ...
```

CALL 13F2H　　X的Y次乘幂　　X↑Y

把存在堆栈中的单精度值作底，以 WRA1 中的指定值为指数，求乘方。结果将在 WRA1 中。

```
; 计算 16↑2
        LD      BC, RETADD   ;计算X的Y次方后的返回地址。
        PUSH    BC           ;存入堆栈以便返回。
        LD      A, 4         ;单精度的类型码。
        LD      (40AFH), A   ;置X为单精度类型。
        LD      HL, X        ;底的地址。
        CALL    9B1H         ;把底值送入 WRA1。
        CALL    9A4H         ;由 WRA1 送入堆栈。
        LD      HL, Y        ;指数的地址。
        CALL    9B1H         ;将指数送入 WRA1。
        JP      13F2H        ; WRA1 = X↑Y，在算完以后,返回 RETADD。
RETADD...
X       DEFW    0            ;单精度值 16。
        DEFW    85H
Y       DEFW    0            ;单精度值 2。
        DEFW    82H
        ...
```

CALL 0809H　　对数　　LOG (N)

计算 WRA1 中的单精度值的自然对数(以 e 为底)，得到的结果为单精度值，在 WRA1 中。

```
        LD      A, 4            ;单精度的类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, POW         ;幂的地址。
        CALL    09B1H           ;将幂送入 WRA1。
        CALL    0809H           ;求幂的自然对数。
        LD      DE, 4121H       ; WRA1 的地址。
        LD      HL, NLOG        ;存储区的地址。
        CALL    09D3H           ;将对数值从 WRA1 送入存储区。
        ...
POW     DEFB    00H             ;浮点数 3 (LSB)。
        DEFB    00H
        DEFB    04H
        DEFB    82H             ; 3.0 的指数。
NLOG    DEFS    4               ;将存放 3 的自然对数。
        ...
```

CALL 0B26H　浮点数变换成整数　FIX (N)

无条件地将 WRA1 中的浮点数的小数部分截尾。结果存在 WRA1 中，类型标志置成整型。

```
        LD      A, 4            ;单精度类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, FLPT        ;浮点数值的地址。
        CALL    09B1H           ;把浮点数送入 WRA1。
        CALL    0B26H           ;截尾，转换成整数。
        LD      HL, (4121H)     ;从 WRA1 取来整数部分。
        LD      (INTG), HL      ;存入存储区域。
        ...
FLPT    DEFB    0BAH            ;单精度值 39.7107。
        DEFB    0D7H
        DEFB    01EH
        DEFB    086H
INTG    DEFS    2               ;存放 39.7107 的整数部分。
        ...
```

CALL 01D3H　重播随机数种子　RANDOM

用刷新寄存器的当前内容重播随机数种子 (40ABH 单元)。

```
        CALL    01D3H           ;重播随机数种子。
        ...
```

CALL 14C9H　随机数　RND (N)

根据 WRA1 中传送的参数，产生 0 与 1 之间或 1 与 n 之间的随机数，得到的随机值为整数，在 WRA1 中，同时置类型标志。传送的参数将决定得到的随机数的范围。参数 0 将得到一个 0 和 1 之间的单精度值。大于 0 的参数将被截去所有的小数部分，得到一个 1 与参数的整数部分之间的值。

```
        LD      A, 2            ;整数类型码。
        LD      (40AFH), A      ;置整数类型。
        LD      A, 50
        LD      (4121H), A      ;把整数 50 放入 WRA1。
        CALL    14C9H           ;得到 1 和 50 之间的随机数。
        LD      HL, (4121H)     ;把随机数送入 HL。
        LD      (RVAL), HL      ;并将它送入存放区域。
        ...
RVAL    DEFW    0               ;存放随机数(整数)。
        ...
```

CALL 1547H　正弦　SIN (N)

在 WRA1 中得到单精度值的正弦。角度必须以弧度为单位放在 WRA1 中。

```
        LD      A, 4            ;单精度类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, ANGL        ;以弧度表示的角度的地址。
        CALL    1547H           ;计算角度的正弦。
        LD      DE, 4121H       ; WRA1 中正弦的地址。
        LD      HL, SANGL       ;存放正弦的地址。
        CALL    09D3H           ;将正弦送入存放区域。
        ...
ANGL    DEFB    18H             ; 30 度，以弧度表示为 (0.5235)。
        DEFB    04H
        DEFB    06H
        DEFB    80H             ;指数。
SANGL   DEFS    4               ;将存放 30 度的正弦值。
        ...
```

CALL 13E7H　平方根　SQR (N)

计算 WRA1 中任何值的平方根，方根将留在 WRA1 中，是一个单精度值。

```
        LD      A, 4            ;单精度类型码。
        LD      (40AFH), A      ;置单精度类型。
        LD      HL, VAL1        ;要求平方根的值。
        CALL    09B1H           ;把它送入 WRA1。
```

```
        CALL    13E7H           ;求值的平方根。
        LD      DE，4121H       ; WRA1 中方根的地址。
        LD      HL，ROOT        ;存放区域的地址。
        CALL    09D3H           ;将方根送入存放区域。
        ...
VAL1    DEFB    00H             ;单精度值 4 。
        DEFB    00H
        DEFB    00H
        DEFB    83H             ;浮点数 4 的指数。
ROOT    DEFS    4               ;存放 4 的平方根。
        ...
```

CALL 15A8H　正切　TAN (N)

计算以弧度表示的角度的正切。WRA1 中的角度必须指定为单精度值。正切值将留在 WRA1 中。

```
        LD      A，4            ;单精度类型码。
        LD      (40AFH)，A      ;置单精度类型。
        LD      HL，ANGL        ;以弧度表示的角度的地址。
        CALL    09B1H           ;把角度送入 WRA1。
        CALL    15A8H           ;求角度的正切。
        LD      DE，4121H       ; WRA1的地址。
        LD      HL，TANGL       ;存放正切的存储地址。
        CALL    09D3H           ;将正切从 WRA1 送入存放区域。
        ...
ANGL    DEFB    18H             ;以弧度表示的 30 度的值 (0.5235)。
        DEFB    04H
        DEFB    06H
        DEFB    80H             ;指数。
TANGL   DEFS    4               ;将存放 30 度的正切。
        ...
```

函数推导

Level II 系统支持 16 个算术函数，其中 7 个可以称为数学函数，它们是正弦、余弦、反正切、正切、平方根、指数(以 e 为底)和自然对数。这些函数中的三个按以下恒等式计算：

$$\cos\theta = \sin\left(\theta + \frac{\pi}{2}\right)$$

$$\tan\theta = \frac{\sin\theta}{\cos\theta}$$

$$\sqrt{x} = e^{\frac{\ln x}{2}}$$

还有一个隐含的数学函数，它用下面的恒等式计算乘幂：

$$x^y = e^{y \ln X}$$

放入 Level II 的是正弦、指数、自然对数和反正切的子程序。其他数学函数都用前述恒等式推导它们的值。

正弦

正弦子程序以 5 项近似法为基础：

$$\sin\theta = \theta - \frac{\theta^3}{3!} + \frac{\theta^5}{5!} - \frac{\theta^7}{7!} + \frac{\theta^9}{9!}$$

其中，θ 是以弧度表示的。使用的实际近似法是：

$$\sin\beta(2\pi) = 2\pi\beta - \frac{(2\pi)^3}{3!}\beta^3 + \frac{(2\pi)^5}{5!}\beta^5 - \frac{(2\pi)^7}{7!}\beta^7 + \frac{(2\pi)^9}{9!}\beta^9$$

其中，β 是一个比率，当它乘以 2π 时，就给出以弧度表示的角度。如果 x 是以度表示的角度，那么根据下列规则，β 也用于决定结果的符号：

$$\beta = \frac{x}{360^\circ} \qquad \text{若 } 0^\circ \leqslant x \leqslant 90^\circ$$

$$\beta = \frac{180^\circ}{360^\circ} - \frac{x}{360^\circ} \qquad \text{若 } 90^\circ < x \leqslant 180^\circ$$

$$\beta = \frac{180^\circ}{360^\circ} - \frac{x}{360^\circ} \qquad \text{若 } 180^\circ < x \leqslant 270^\circ$$

$$\beta = \frac{x}{360^\circ} - \frac{360^\circ}{360^\circ} \qquad \text{若 } 270^\circ < x \leqslant 360^\circ$$

用于正弦级数的系数精确到 4 位小数，x 正弦的最大误差小于等于 0.000003，因此，x 正弦的全部值将精确到 4 位。

指数

指数子程序计算 x 所有值的 e^x，其中

$$-88 \leqslant x \leqslant 88$$

这个函数使用的近似法推导如下：

由于 $e^x = 2^{x\log_2 e}$

考虑 $2^{[x\log_2 e]+1}$，其中[表示最大整函数。

于是 $e^x = e^{-t}[2^{[x\log_2 e]+1}]$

若 $t = -x + [x\log_2 e]\ln 2 + \ln 2$

因 $x = \ln e^x = \ln e^{-t}[2^{[x\log_2 e]+1}]$

$x = -t + \{[x\log_2 e] + 1\}\ln 2$

于是 $t = -x\{[x\log_2 e] + 1\}\ln 2$

并且 $0 < t < \ln 2$

同时，因为

$$e^{-t} = 1 - t + \frac{t^2}{2!} - \frac{t^3}{3!} + \frac{t^4}{4!} - \frac{t^5}{5!} + \frac{t^6}{6!} - \frac{t^7}{7!}$$

用下列级数近似 e^{-t}

$$e^{-t} = 1 - t + 0.5t^2 - 0.166t^3 - 0.0416t^4 - 0.0083t^5 + 0.0013298t^6 - 0.0001413t^7$$

于是，用 e^{-t} 的近似值乘以 $2^{[x\log_2 e]+1}$ 就可求得 e^x，其给出的结果通常至少精确到 5 位有效数字或 4 位小数，而无论两者中哪个较大。

反正切

反正切子程序使用下列近似：

$$\arctan x = x - \frac{x^3}{3} + \frac{x^5}{5} - \frac{x^7}{7} + \frac{x^9}{9} - \frac{x^{11}}{11} + \frac{x^{13}}{13} - \frac{x^{15}}{15} + \frac{x^{17}}{17}$$

若 $x < 0$，则用 x 的绝对值计算级数，并把结果的符号反向。若 $x > 1$，则用 $\frac{1}{x}$ 的值计算级数，得到的结果为 $\frac{\pi}{2} - \arctan\frac{1}{x}$。对于 $0 < x < 1$ 的值，就用 x 的原始值计算级数。在计算机中，级数使用的系数从第七项开始与近似级数中的系数不同，第五项和第六项系数的精度也在一定程度上有所不同。实际使用的级数是：

$$\arctan x = x - 0.33331x^3 + 0.199936x^5 - 0.142089x^7 + 0.106563x^9 - 0.0752896x^{11} + 0.0429096x^{13} - 0.01616157x^{15} + 0.00286623x^{17}$$

使用这个近似的最大误差为 0.026。

自然对数

自然对数子程序是以下列级数的三项为基础的：

$$\ln x = 2\left[\left(\frac{x-1}{x+1}\right) + \frac{1}{3}\left(\frac{x-1}{x+1}\right)^3 + \frac{1}{5}\left(\frac{x-1}{x+1}\right)^5 + \cdots\right]$$

这个级数对于 $x < 1$ 的值收敛，因此，必须重新定义为

$$x = x2^n$$

其中，n 是一个整数比例因子，同时

$$\frac{1}{2} \leqslant X < 1$$

通过代数运算(运算过程不在此列出)，X 项可以用 $\frac{X}{\ln 2}$ 代替：

$$\ln x = \frac{2}{\ln 2}\left[\left(\frac{\frac{X}{\ln 2}-1}{\frac{X}{\ln 2}+1}\right) + \frac{1}{3}\left(\frac{\frac{X}{\ln 2}-1}{\frac{X}{\ln 2}+1}\right)^3 + \frac{1}{5}\left(\frac{\frac{X}{\ln 2}-1}{\frac{X}{\ln 2}+1}\right)^5 + \cdots\right]$$

由于

$$\ln x = \ln X2^n = \ln X + n\ln 2$$

同时，因为

$$\ln 2\left(\frac{\ln\frac{X}{\ln 2}}{\ln 2} + \frac{\ln(\ln 2)}{\ln 2}\right) = \ln X$$

根据级数，由此得到

$$\ln x = \left(\frac{\ln \frac{x}{\ln 2}}{\ln 2} - 0.5 + n \right) \ln 2$$

在这个函数中，ln 2 近似为 0.707092，而

$$\frac{\ln (\ln 2)}{\ln 2} \text{ 近似为 } -0.5$$

如果 $x > 0$ 是适当的，那末 ln x 将精确到 4 位有效数字。如果 x 极其接近于 0 或非常大，则它不成立。

系统功能

系统功能是一些可以进入的 ROM 入口点，这就是说，例如磁盘系统中，调用这些入口点的汇编语言程序在 IPL 以后，开始执行 BASIC 实用程序以前，可以首先得到执行。这些入口点与 BASIC 功能不同，因为它们不需要为了操作的正确而预置通讯区。没有磁盘的 Level II 系统，由于它的 IPL 处理，总是有一个预置的通讯区。

这儿提到的某些子程序要使用通讯区，但是它们都不需要预置任何特殊的单元。然而，万一在这些子程序检测到错误时，它们会调用系统错误子程序，系统错误子程序将假设某些字中含有代表错误含义的数据，并将控制返回到 BASIC 解释程序的输入阶段。

RST 08H　比较字符

把用 HL 寄存器指示的输入字符串中的字符与 RST 08H 指令下面的单元中的值进行比较。如果两者相同，那么控制就返回到 RST 08H 指令地址加 2 的地址，此时，在 A 寄存器中含有下一个字符，并且 HL 增加 1*。如果两个字符不一致，那么给出一个句法错误信息，并且控制返回到输入阶段。

```
;测试以 HL 为指针的字符串，看它是否含有字符串，'A = B = C'。
        RST     08H         ;测试 'A'。
        DEFB    41H         ; 'A' 的 16 进制值。
        RST     08H         ;找到 'A'，再测试 '='。
        DEFB    3DH         ; '=' 的 16 进制值。
        RST     08H         ;找到 '='，再测试 'B'。
        DEFB    42H         ; 'B' 的 16 进制值。
        RST     08H         ;找到 'B'，再测试 '='。
        DEFB    3DH         ; '=' 的 16 进制值。
        RST     08H         ;找到 '='，再测试 'C'。
        DEFB    43H         ; 'C' 的 16 进制值。
        ...                 ;找到字符串 'A = B = C'。
```

* 在两者相同时，子程序自动转入执行 RST 10H，然后返回，所以此时 A 含有下一个字符，指针 HL 加 1。——译注

RST 10H　检测下一个字符

将下一个字符从 HL 寄存器对指示的字符串中送入A寄存器，如果它是字母，则复位C标志；如果它是字母数字，则置位C标志。空格和控制代码 09H 及 0BH 将被忽略，仍测试并送入后面的字符。HL 寄存器在送入任何字符之前将先增量，因此，在首次调用时，HL 寄存器应含有字符串地址减1。字符串必须以 0 字节结束。

```
;假设 HL 指示的当前字符串为赋值语句的一部分，它含有一个任意的符号，后面是
  一个常数或变量名。进行必要的测试来确定使用的是常数还是变量。
        RST     08H         ;测试'='。
        DEFB    3DH         ;'='的16进制值。
NEXT    RST     10H         ;取'='后面的字符。
        JR      NC, VAR     ;若是变量名，则C标志不置位。
        CALL    1E5AH       ;取常数值。
        JR      SKIP        ;连接公用的程序。
VAR     CP      2BH         ;不是数字，测试'+'、'-'或字母。
        JR      Z, NEXT     ;跳过'+'号。
        CP      2DH         ;不是'+'，测试'-'号。
        JR      Z, NEXT     ;跳过'-'号。
        CALL    260DH       ;假设它是正确的字母，检索变量名（见本章关于
                             260DH 的说明）。
        ...
SKIP    ...
```

RST 18H　比较 DE 和 HL 的数值

比较 DE 和 HL 中的数值。不比较带符号的整数（正数除外）。只使用A寄存器。在状态寄存器中得到的比较结果为：

C(C 标志置位)——HL < DE

NC(C 标志复位)——HL > DE

NZ(Z 标志复位)——HL ≠ DE

Z(Z 标志置位)——HL = DE

```
;此例测试由 HL 指示的字符串中等号后面的数值大小，以证明它落在100和500
 之间。
        RST     08          ;测试'='。
        DEFB    3DH         ;'='号的16进制值。
        RST     10H         ;找到'='，测试下一个字符。
        JR      NC, ERR     ;若不是数字，则C标志复位。
        CALL    1E5AH       ;取二进制值。
        LD      HL, 500     ;上限值。
        RST     18H         ;与上限值比较。
        JR      C, ERR      ;若数值大于500，则C标志置位。
```

```
        LD      HL, 100         ;下限值。
        RST     18H             ;与下限值比较。
        JR      NC; ERR         ;若数值小于100，则C标志复位。
        ...
```

RST 20H　测试数据类型

根据数据类型标志（40AFH），在A寄存器中得到唯一的数字值，并得到状态标志的组合。通常把这个调用用于测定 WRA1 中当前值的类型。然而，使用它要小心，因为类型标志和 WRA1 中的当前值类型可能不一致，特别是如果使用这儿描述的某些调用把数值装入 WRA1 中时，更要谨慎。

类　型	状　态	A寄存器
02（整数）	NZ/C/M/E	—1
03（字符串）	Z/C/P/E	0
04（单精度）	NZ/C/P/O	1
08（双精度）	NZ/NC/P/E	5

```
;在整数相加以后测试数据类型，以确定是否产生溢出(结果将被转换成单精度)。
        LD      A, 2            ;整数类型码。
        LD      (40AFH), A      ;设置整型。
        LD      BC, (VAL1)      ;第一个量。
        LD      HL, (VAL2)      ;第二个量。
        CALL    0B2DH           ;进行整数加。
        RST     20H             ;测试溢出。
        JP      M, OK           ;结果为整数。
        ...                     ;结果不是整数。
        ...                     ;测试其他类型。
OK      LD      (SUM), HL       ;存储整数结果。
        ...
VAL1    DEFW    125             ; 16位整数值。
VAL2    DEFW    4235            ; 16位整数值。
SUM     DEFW    0               ;存放16位的值。
```

RST 28H　DOS 功能调用

将A寄存器中的请求代码传送给 DOS 进行处理。对于非磁盘系统，则返回。对于磁盘系统，A寄存器必须含有合法的 DOS 功能代码。如果代码是正的，那末忽略此调用，而控制就返回到调用它的程序*。注意，DOS 例行程序丢弃由 RST 指令存入堆栈的返回地址。在处理后，控制将返回到堆栈中的前一个地址。调用程序列为：

```
;装入并执行 DEBUG（查错调试程序）。
        LD      A, 87H          ;装入 DEBUG 的 DOS 代码。
        CALL    DOS
        ...                     ;返回到此。
DOS     RST     28H             ;进行 DOS 调用(将返回到调用它的程序)。
```

* DOS 功能代码必须是位7（最高位）为1，若位7为0（即正数）是无效的。——译注

```
...
```

RST 30H　　装入 DEBUG

这个调用装入 DEBUG (查错调试)程序,并将控制传送到这个程序。当 DEBUG 处理完毕时,控制返回到原先调用它的程序。对于非磁盘系统,控制立即返回。

;若出现不符合逻辑的条件,就装入并执行 DEBUG。

```
        ...             ;测试合法的条件。
        JR      Z, OK   ;若条件正确,则转移。
        RST     30H     ;否则装入并执行 DEBUG。
OK      ...             ;继续。
```

RST 38H　　中断入口点

这是所有中断的系统入口点。它含有转移到通讯区中一段程序的 JP 指令，以进入中断。对于非磁盘系统,这段程序由 DI (禁止进一步中断)指令和它后面的 RET (返回到中断点)指令组成; 如果是 DOS 系统，则是转移到 SYS0 中的中断处理程序的 JP 指令。对于 DOS 系统,中断处理程序由任务调度程序组成,在调度程序中测定中断的确切原因(通常是时钟中断),并且根据任务控制块执行下一个任务。在任务完成以后,控制返回到中断点。

;截取所有时钟中断并测试 ABH 口上的装置，如果 READY线 (位 7) 为真 (高电位或 1),则打开 DEH 口上的咖啡壶电源,否则转移到正常的 DOS 中断处理程序。

```
        ORG     4012H       ;用转移到我们自己的中断处理程序来代替转移
                            到 DOS 中断处理程序。
        JP      HERE
        ORG     0F000H      ;我们的中断处理程序。
HERE    DI                  ;禁止进一步中断。
        PUSH    AF          ;我们将需要使用 AF 寄存器。
        IN      A, (0ABH)   ;取装置的状态。
        OR      A           ;设置位 7 的状态。
        JP      M; TOCP     ;若为负,则装置开。
        POP     AF          ;装置关,恢复寄存器。
        JP      4518H       ;去 DOS 中断处理程序。
TOCP    LD      A, 21H      ;开咖啡壶电源的代码。
        OUT     (0DEH), A   ;将命令送至咖啡壶。
        POP     AF          ;然后恢复寄存器。
        JP      4518H       ;并转移到 DOS 中断处理程序。
        ...
```

CALL 09B4H　　将 BC 和 DE 中的单精度值送入 WRA1

将 BC 和 DE 中的单精度值送入 WRA1。 HL 被破坏，BC 和 DE 保持不变。注

意：类型标志将不改变！

```
        ...
        LD      BC,(PART1)   ;取第一个自变量。
        LD      DE,(PART2)   ;自变量的剩余部分。
                             ;注意，我们已假设 WRA1 当前含有一个单精度
                              值！
        CALL    09B4H        ;将 PART1 送入 WRA1。
        LD      BC,(PART3)   ;取要加的值。
        LD      DE,(PART4)   ;值的剩余部分。
        CALL    0716H        ;求和,结果留在 WRA1 中。
        ...
PART2   DEFW    0000H        ;单精度值 1.5 的 LSB。
PART1   DEFW    8140H        ;单精度值 1.5 的指数及 MSB。
PART4   DEFW    0000H        ;单精度值××的 LSB。
PART3   DEFW    0000H        ;单精度值××的指数和 MSB。
        ...
```

CALL 09B1H　将一个由 HL 指示的单精度值送入 WRA1

把由 HL 指示的单精度值装入 BC 和 DE，然后送入 WRA1。破坏 HL，BC 和 DE 寄存器。

```
        ...
        LD      HL,VAL       ;取要传送值的地址。
        CALL    09B1H        ;将值送入 WRA1。
        ...
VAL     DEFW    8140H        ;单精度值 1.5。
        DEFW    0000H        ; 1.5 的剩余部分。
        ...
```

CALL 09C2H　将单精度值装入 BC 和 DE

把由 HL 指示的单精度值装入 BC 和 DE。本子程序使用所有寄存器。

;计算两个单精度值的乘积,并将乘积送入 BC 和 DE。

```
        LD      HL,VAL1      ;值 1 的地址。
        CALL    09B1H        ;将它送入 WRA1。
        LD      HL,VAL2      ;值 2 的地址。
        CALL    09C2H        ;将它装入 BC 和 DE。
        CALL    0847H        ;求乘积,积留在 WRA1 中。
        LD      BC,(4123H)   ;积的指数和 MSB 装入 BC。
        LD      DE,(4121H)   ;积的 LSB 装入 DE。
        ...
```

```
VAL1    DEFW    ××××
        DEFW    ××××
VAL2    DEFW    ××××
        DEFW    ××××
        ...
```

CALL 09BFH　将单精度值从 WRA1 装入 BCDE

把一个单精度值从 WRA1 装入 BC 和 DE 寄存器。注意：传送子程序不进行类型标志的测试。保证 WRA1 实际含有单精度值是调用程序的责任。

```
        ...
        LD      HL, VAL1        ;要送入 WRA1 的值的地址。
        CALL    09B1H           ;将值 1 送入 WRA1。
        LD      HL, VAL2        ;要加的值的地址。
        CALL    09C2H           ;把要加的值送入 BCDE。
        CALL    0716H           ;进行单精度加。
        CALL    09BFH           ;把结果装入 BCDE。
        LD      (SUM1), DE      ;存储 LSB。
        LD      (SUM2), BC      ;存储指数和 MSB。
        ...
SUM1    DEFW    0               ;存放单精度值的 LSB。
SUM2    DEFW    0               ;存放指数和 MSB。
VAL1    DEFW    0000H           ;单精度值 2.0 的 LSB。
        DEFW    8200H           ;单精度值 2.0 的指数和 MSB。
VAL2    DEFW    0000H           ;单精度值 5.0 的 LSB。
        DEFW    8320H           ;单精度值 5.0 的指数和 MSB。
        ...
```

CALL 09A4H　将 WRA1 的值送入堆栈

把 WRA1 中的单精度值送入堆栈。它按 LSB、MSB、指数的次序存储。所有寄存器保持不变。注意，传送子程序不测试类型标志，它只假设 WRA1 含有单精度值。

;将两个单精度值加起来并把和存入堆栈。

;调用一个子程序，把数值由堆栈装入，执行它自己的操作并返回。

```
        LD      HL, VAL1        ;要送入 WRA1 的值的地址。
        CALL    09B1H           ;将值 1 送入 WRA1。
        LD      HL, VAL2        ;要加的值的地址。
        CALL    09C2H           ;把值 2 送入 BC 和 DE。
        CALL    0716H           ;作单精度加。
        CALL    09A4H           ;把和存储在堆栈中。
        CALL    NSUB            ;调用下一个子程序，返回时，WRA1 中为新的
```

值。

```
        ...
NSUB    POP     HL          ;取返回地址。
        LD      (RET), HL   ;把它送到安全的地方。
        LD      HL, VAL3    ;待加值的地址。
        CALL    09B1H       ;将值 3 送入 WRA1。
        POP     BC          ;取指数和 MSB。
        POP     DE          ;取 LSB。
        CALL    0716H       ;加以前的值。
        LD      HL, (RET)   ;取返回地址。
        JP      (HL)        ;返回到调用它的程序。
VAL1    DEFW    0000H       ;单精度值 2.0 的 LSB。
        DEFW    8200H       ; 2.0 的指数和 MSB。
VAL2    DEFW    0000H       ;单精度值 5.0 的 LSB。
        DEFW    8320H       ; 5.0 的指数和 MSB。
VAL3    DEFW    0A6CH       ;单精度值 —0.333333 的 LSB。
        DEFW    7FAAH       ; —0.333333 的指数和 MSB。
        ...
```

CALL 09D7H　通用传送子程序*

将存储器内容从 DE 寄存器指示的地址传送到由 HL 寄存器给定的地址，传送的字节个数为 B 寄存器中的数目。除 C 寄存器以外该子程序，将使用全部寄存器。

```
;将空格填入一个 DCB 中，然后把一个名称送入其中。
        LD      A, 20H      ;空格的 16 进制值。
        LD      B, 32       ;填充空格的字节数。
        LD      DE, IDCB    ; DE = DCB 的地址。
LOOP    LD      (DE), A     ;将一个空格存入 DCB。
        INC     DE          ;存储地址加 1 。
        DJNZ    LOOP        ;循环，直到 DCB 中填满空格。
        LD      DE, NAME    ;现在，要将文件名送入 DCB。
        LD      HL, IDCB    ; DE = 名称地址，
                            ;  HL = DCB 地址。
        LD      B, LNG      ;要传送的名称中的字符数目。
        CALL    09D7H       ;把名称送入 DCB。
        ...
IDCB    DEFS    32          ;空的 DCB。
LNG     EQU     ENDX — $    ;让汇编程序计算文件名的 LNG (长度)。
```

* 这个子程序也可从 09D6H 入口，此时，应将传送的字节数放入 A 寄存器中。——译注

```
NAME   DEFM   'FILE1/TXT'    ;要送入 DCB 的名称。
ENDX   EQU    $              ;表示名称的结束。
       ...
```

CALL 09D3H　变量传送子程序*

把由类型标志（40AFH）指定的几个字节，从 DE 中的地址传送到 HL 中的地址。本子程序使用 A，DE 和 HL 寄存器。

```
;找一个双精度变量的地址,然后把它送入存储区域。
       LD     HL, NAME1   ;要寻找的变量名。
       CALL   260DH       ;得到字符串 'X' 的地址。
       RST    20H         ;证明它是双精度。
       JR     NC, OK      ;如果是双精度,则转移。
       JP     ERR         ;不是双精度,则错误。
OK     LD     HL, LOCAL   ; HL = 存储地址,
                             DE = 变量地址。
       CALL   09D3H       ;将值从 VLT 送到存储区域。
       ...
ERR    ...
NAME1  DEFM   'X'         ;要寻找的变量名。
       DEFB   0            ;必须用零结束。
LOCAL  DEFS   8            ;用于存储双精度值的足够的空间。
       ...
```

CALL 29C8H　字符串传送子程序

在入口时，HL 指向要传送的字符串的字符串控制块，而 DE 含有目标地址。该子程序要用到所有寄存器。不传送字符串长度和地址。字符串控制块具有下格式：

```
       DEFB   X           字符串长度
       DEFW   ADDR        字符串地址
```

```
;寻找一个叫做 F$ 的字符串变量的地址,将字符串 F$ 传送到叫做 DCB 的局部存
 储区域。
       LD     HL, NAME    ;要寻找的变量名称。
       CALL   260DH       ;寻找字符串 F$ 的地址。
       RST    20H         ;证明它是字符串。
       JR     Z, OK       ;若是字符串,则转移。
       JP     ERR         ;不是字符串,则错误。
OK     LD     A, (DE)     ;取字符串长度。
       CP     33          ;它必须小于 33。
```

* 原文误为 0982H。另外此子程序也可从 09D2H 入口，但此入口的地址指针与 09D3H 相反，即 HL = 源地址，DE = 目标地址。——译注

```
        JP      P, ERR          ;字符串长度大于 32 时,则为错误。
        PUSH    DE              ;将 DE 送入 HL 的简便方法。
        POP     HL              ;把字符串地址送入 HL。
        LD      DE, LOCAL       ; DE = 存储地址。
        CALL    29C8H           ;把字符串变量送入局部存储区域。
        ...
ERR     ...
NAME    DEFM    'F$'            ;要寻找的变量名称。
        DEFB    0               ;结束名称的需要。
LOCAL   DEFS    32              ;局部存储区域。
        ...
```

BASIC 功能

BASIC 功能与系统功能不同,因为它们主要与通讯区中的表格有关,假设通讯区已被预置,并非常好地保持着。这意味着,在调用这些子程序中的任何一个之前,必须已经进入了 BASIC 解释程序, 并且在 RAM 中的 BASIC 实用程序必须是原封未动的。进行 CALL 的汇编程序必须作为 BASIC 程序所调用的子程序运行。

关于通讯区中的表格和存储区域的完整的说明,见第四章。

CALL 1B2CH 搜索行号

搜索程序语句表 (PST) 寻找 BASIC 语句, 它的行号用 DE 寄存器对指定。本子程序使用全部寄存器。其出口条件为:

状 态	条 件	寄 存 器
C/Z	找到程序行	BC=PST 中该程序行的起始地址。 HL=PST 中下一行地址。
NC/Z	程序行不存在行号太大	HL 和 BC 为下一个可用于安放的地址。
NC/NZ	程序行不存在	BC= 大于指定行号的第一个行号地址。 HL=下一行的地址。

;寻找 PST 中 BASIC 语句行号 750 的地址。 如果程序行不存在, 则得到状态 −1 (若大于现有的任何行号);或得到状态 −2 (若存在大于 750 的行号)。如果找到了程序行,那么得到状态 0 。

```
        LD      DE, 750         ;要寻找的行号。
        CALL    1B2CH           ;在 PST 中寻找该程序行。
        JR      NC, NO          ;若程序行不存在,则 C 标志复位。
        LD      HL, 3           ;为了使指针跨过下一行地址和行号而给予的增
                                 量(应增加 4 ,因下面 RST 10H 将加 1 ,所以只
                                 给出 3 )。
```

```
        ADD     HL, BC
        RST     10H             ;取语句的第一个字符。
        ...
        LD      A, 0            ;表示程序行找到。
        RET                     ;返回调用它的程序。
NO      JR      Z, M2           ;若行号太大,则转移。
        LD      A, 0FEH         ;表示程序行不存在 (A = -2)。
        RET                     ;返回调用它的程序。
M2      LD      A, 0FFH         ;表示程序行不存在,行号太大 (A = -1)。
        RET                     ;返回调用它的程序。
```

CALL 260DH　寻找变量的地址

这个入口点搜索变量表 (VLT)，寻找一个与 HL 所指示的字符串中的名称相符合的变量名。如果该变量存在,那么在 DE 中得到它的地址。如果它没有被定义过，那么它以初值 0 建立起来,并且在 DE 中得到它的地址。下标变量和非下标变量都可以寻找，而且数据类型的后缀* 也可以包含在名称字符串中。变量名字符串必须用一个机器 0 字节结束。本子程序使用全部寄存器。

```
;寻找变量 A3 的地址。
        LD      HL, STRNG       ;要寻找的变量名。
        CALL    260DH           ;在 VLT 中找它的地址。
        LD      (ADDR), DE      ;存储起来,供进一步引用。
        ...
STRNG   DEFM    'A3'            ;变量名是 A3。
        DEFB    0               ;结束符。
TSRNG   DEFM    'A(25)'         ;变量名是 A(25)。
        DEFB    0
STRNG   DEFM    'A%'            ;变量名是 A%。
        DEFB    0
        ...
```

CALL 1EB1H　GOSUB

在汇编程序中可以用于执行等效的 GOSUB 语句。它允许在汇编子程序中调用 BASIC 子程序。在 BASIC 子程序执行完以后,控制返回到汇编程序中的下一个指令。本子程序要使用全部寄存器。在入口时，HL 必须含有一个 ASCII 字符串,它是 BASIC 子程序的起始行号。

```
;在汇编语言程序中模拟 GOSUB 语句。
        LD      HL, STRNG       ;进入 GOSUB 的 BASIC 行号的地址。
```

---

* 后缀是指跟在变量名后边的说明数据类型的符号,即 $、%、! 和#号。——译注

```
        CALL    1EB1E           ;等效于 GOSUB 1020。
        ...                     ;当 BASIC 程序执行 RETURN 时，将返回到这
                                 儿。
STRNG   DEFM    '1020'          ; BASIC 子程序的行号。
        DEFB    0               ;结束符。
```

CALL 1DF7H　　TRON

使跟踪功能起作用。由此，将显示出已执行过的每个 BASIC 语句的行号。该子程序使用A寄存器。

```
;打开跟踪，然后执行一个 BASIC 子程序。
        CALL    1DF7H           ;打开跟踪。
        LD      HL, LN          ; GOSUB 的行号。
        CALL    1EB1H           ;执行 GOSUB 1500。
        ...
LN      DEFM    '1500'          ; BASIC 子程序的行号。
        DEFB    0               ;行号结束。
```

CALL 1DF8H　　TROFF

使跟踪功能失去作用。使用A寄存器。

```
;打开跟踪功能，执行 BASIC 子程序，返回后关闭跟踪。
        CALL    1DF7H           ;打开跟踪。
        LD      HL, LN          ; BASIC 子程序的行号。
        CALL    1EB1H           ;执行 GOSUB 2000。
        CALL    1DF8H           ;关闭跟踪。
        RET                     ;返回调用它的程序。
LN      DEFM    '2000'          ; BASIC 子程序的行号。
        DEFB    0               ;结束符。
```

JP 1EDEH　　RETURN

将控制返回到最近的 GOSUB 调用后面的一个 BASIC 语句。如果被 BASIC 子程序调用的汇编程序希望直接返回到原先的调用它的程序，而不通过汇编子程序的入口点返回，那么此出口就可以用于这种返回。在通过 1EDEH 返回之前，必须首先把由于调用汇编程序而存入堆栈中的返回地址清除掉。

```
300     GOSUB 1500,             ;调用 BASIC 子程序。
310     GOSUB 1510              ;从子程序调用返回这儿。
320     ...
        ...
1500    Z = USR1(0)             ;调用汇编子程序及返回。
1510    Z = USR2(0)             ;调用另一个汇编子程序及返回。
```

```
1520    ...
        ...
; USR1 子程序的入口点。
        ...                     ;执行所要求的任何处理。
        POP     AF              ;从堆栈清除返回到 1510 的地址。
        JP      1EDEH           ;直接返回到 310。
; USR2 子程序的入口点。
        ...                     ;执行 USR2 调用的必要处理。
        POP     AF              ;清除返回到 1520 的地址。
        JP      1EDEH           ;直接返回 320。
```

CALL 28A7H　显示信息

在当前的系统输出设备(通常是显示器)上显示以 HL 为指针的信息。要显示的字符串必须用机器零字节或回车代码 0DH 结束。如果用回车结束，那么控制应在获得 41D0H (JP 5B99H) 处的 DOS 出口以后，返回到调用它的程序。这个子程序使用文字串库表和字符串区域。如果没有很好地保持通讯区和字符串区域，则不应调用它。

```
;将 MLIST 中的信息写到当前的系统输出设备上去。
        ...
        LD      HL, MLIST       ; HL = 信息的地址。
        CALL    28A7H           ;送往系统输出设备。
        ...
MLIST   DEFM    'THIS IS A TEST'
        DEFB    0DH             ;这是所要求的结束符。
        ...
```

CALL 27C9H　求空闲内存的数量

计算变量表末尾与堆栈末尾之间所剩余的内存数量。在 WRA1 (4121H—4124H) 中得到单精度的结果。

```
;获得堆栈和 VLT 的末尾之间的全部可用的空闲内存，并将它分成区，以用于比赛
 分类。
        ...
        DI                      ;必须禁止中断，因为没有堆栈空间可用于中断处
                                 理。
        CALL    27C9H           ;得到空闲内存的数量。
        CALL    0A7FH           ;将它转换成整数。
        LD      DE, (4121H)     ;把它送入 DE。
        LD      HL, 500         ;证明它至少有 500 个字节。
        RST     18H
        JR      C, ERR          ;错误——没有足够的空间。
```

```
LD      HL，(40D1H) ；区域的开始。
LD      (EVLT)，HL  ；存储起来，以便恢复。
LD      HL，0       ；于是，可装入 CSP（当前堆栈指针）。
ADD     HL，SP      ；区域的末尾。
LD      (ECSP)，HL  ；存储起来，以便恢复。
...
```

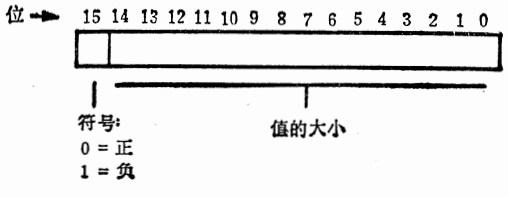
CALL 2B75H　打印信息

把以 HL 为指针的字符串写入当前输出设备。字符串必须用零字节结束。这个调用不同于前述的 28A7H，因为它不使用文字串库区域，但它使用相同的显示子程序，并且获得相同的在 41C1H 的 DOS 出口。本子程序使用全部寄存器。如果把 C9H(RET) 存入 41C1H，那么这个子程序可以在不装入 BASIC 实用程序时调用。

```
；将信息写入当前的输出设备。
        ...
        LD      HL，MLIST   ；信息的地址。
        CALL    2B75H       ；将信息送到系统设备。
        ...
MLIST   DEFM    'THIS IS A TEST'
        DEFB    0           ；要求的结束符。
        ...
```

内部的数值表示法

BASIC 表示的整数为带符号的 16 位量。位 15 放有符号位，而位 0 —位 14 存放值的大小。能够表示的最大正数值是 32767 或 7FFFH。能够表示的最小负数值是—32768 或 8000H。



整数的数值范围为：

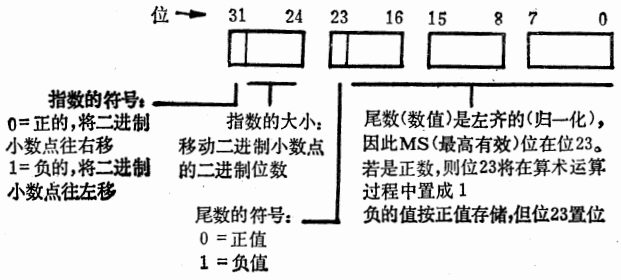
正数值　0000H—7FFFH　或 0 —32767

负数值　FFFFH—8000H　或－1 ——32768

负数值是以正数等价值的补码表示的。

BASIC 支持两种形式的浮点数：一种是单精度，另一种是双精度。两种类型都有带符号的 8 位指数。单精度数有带符号的 24 位尾数，而双精度数有带符号的 56 位尾数。

这两种类型的浮点数格式如下*：



单精度和双精度的唯一差别是尾数中的位数。在正的单精度值中，可以表示的最大有效位数字是 $2^{24}-1$ 或 8388607 或 7FFFFFH。双精度数延长了尾数，因此，可以表示的正值精确到 $2^{56}-1$ 或 3.578×10^{16}。

8388607 和 3.578×10^{16} 这些数字并不是在单精度或双精度数中可以表示的最大数值，但是，它们是在不丢失任何精度时可以表示的最大数值。这是因为两种类型的数还有范围在 2^{-128} 到 2^{127} 之间的指数。这意味着，在理论上，即使尾数只有 24 位或 56 位有效位，二进制小数点对于正值都可以向右延伸 127 位，对于负值都可以向左延伸 128 位**。使用的数据的类型(有效数字的位数)取决于实际的需要。例如，普兰克(Plank) 常数 6.625×10^{-34} J-SEC 焦耳·秒)，可以用单精度值表示而不丢失任何精度，因为它只有 4 位有效数字。然而，如果我们要把相同大小的钱数加起来，那么它必须是双精度值，因为所有的数字都是有效的。

---

* 图中所示为单精度，双精度仅尾数为 56 位而已，两者格式是一样的。另外，对此格式补充说明如下：
1.指数是以 128 为基数的偏移二进制码。若将二进制小数点往右移 n 位，则指数为 $128-n$（若左移，则为 $128+n$)。例如，小数点右移 3 位时指数为 125 即二进制 01111101，若左移 3 位则为 131 即二进制 10000011。若按原文所述把指数分为符号位和指数大小，则在符号为 0 时其指数大小就应用补码表示。例如，小数点右移 3 位，符号为 0，大小为 3 的二进制补码 1111101，合起来即 01111101 即 125。当指数符号为 1 时，其指数大小不用补码表示。
2.尾数是左齐的，其最高位既是尾数最高位又是符号位。例如，正的尾数 111 被表示成 011，而负的 111 仍表示为 111 (以上都是二进制)。——译注

** 正负值是指正负数值。——译注

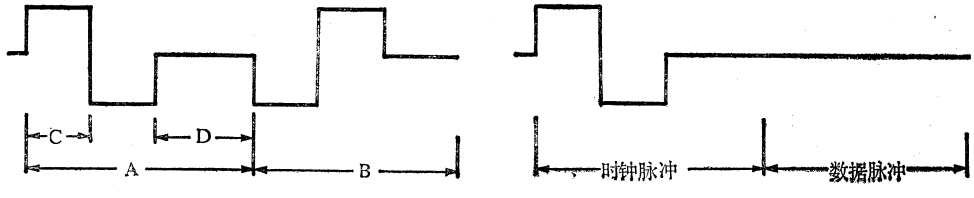
第三章　盒式磁带和磁盘

这一章对盒式磁带和磁盘的实际输入/输出操作作一介绍性的说明。这些实例程序只是作为例子用的，并不适合一般应用。然而也许有些特殊情况，像只需要简单的读/写操作，以及某些作了限制的应用，可以使用这些实例程序。

盒式磁带的输入/输出

从某些方面来看，盒式磁带的输入/输出是有些特点的。首先，每个字节是在软件控制下一位接着一位传送的。这根本不同于其它的输入/输出格式，其它的输入/输出格式是每次传送一个完整的字节。对于大多数输入/输出操作，像访问存储器或执行一条 IN 或 OUT 指令，都需要在 CPU 和外部设备之间传送一个完整的字节。然而，如果外部设备是盒式磁带机，则每个二进制位(待传送字节的)必须依靠软件分别传送。

第二个特点是传送这些位的方法不同于其它输入/输出。它们必须有精确的定时，而且根据被写的是二进制的 0 还是 1，程序必须使用不同的代码。如果表示的是一个二进制的 1，则每一位的记录包括一个时钟脉冲 (CP)，紧接着是一段固定长度的消了磁的磁带，再后面是另一个时钟脉冲；如表示的是一个二进制的 0，则延长消磁的磁带长度。一个二进制的 1 和 0 如下图所示。



二进制的1　　二进制的0

A，B，C 和 D 是按时间单位度量的。因为时间能用机器周期度量，所以给定的间距值将以机器周期来表示，其中每一条指令(任何指令不管它多长)等于一个周期，而且一个周期等于 1 微秒。虽然，这是一种粗糙的估计，但是能使用。在 Level II 中，假定 A，B 之和为 2 毫秒。

利用上述假定和计算用于 Level II 软件中的那些指令，可给出下面这些数值。

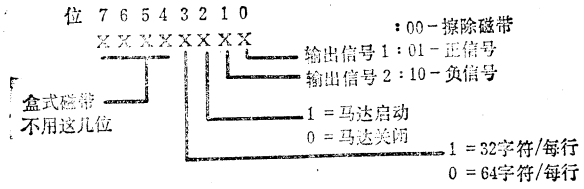
A，B　每半个位为 1.4 毫秒，每位为 2.8 毫秒。

C　每个时钟脉冲是 0.2 毫秒乘 2，即 0.4 毫秒。

D　1 毫秒。

对盒式磁带的输入/输出编程作详细讨论以前，我们应该先回顾一下基本原理。用存 00 (设备 0) 或者 01 (设备 1) 之一到 37E4H 单元来实现设备的选择。马达启动和装入或清除数据锁存器是用发送一个命令值到口地址为 FFH 的盒式磁带控制器来实现的。

命令值表示如下：



当你发送命令到盒式磁带机的时候，要特别留意保护当前荧光屏字符的大小。系统把发送到显示控制器的最后一个命令的拷贝保存在 403DH。403DH 单元内容的位 3 应该并入到送往盒式磁带机的每个命令中。

1 位(称为比特单元)的写操作可以分成两步。首先，写一个时钟脉冲作为位的开始信号。紧接着是一段已擦除干净的磁带，可把这段空白带看作是时钟脉冲的一部分。第二步，如果所写的位是 1，则写另一个时钟脉冲，如果写的位是 0，则再留出一些空白带。

读操作是由搜索时钟脉冲开始的，而后跳到数据脉冲区。如果碰到的是一段空白带，则读数据脉冲区的结果就送回一个 0，如果发现不是空白带，则读回一个 1。下面就是用于盒式磁带操作的一个例子。Level II 使用的操作代码能够在 Level II 程序清单的 01D9H—02A8H 区找到。

汇编程序目的码格式

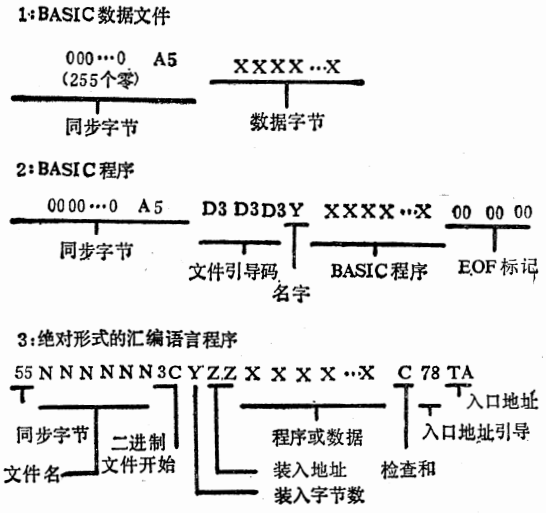
DOS 用一个称为 LOAD 的系统实用程序来装入磁盘目的文件。也能在 DOS 管理下，用打入带有扩充符为 CMD 的文件名来装入目的文件。下面给出磁盘目的文件的格式。它比磁带文件要复杂。这是因为在目的码中夹杂着一些控制代码。在把目的码传送到它的指定地址以前，装入程序先把文件读进缓冲区。控制代码用来为装入程序指出代码应装入什么地方，要装入多少字节，从什么地方开始执行。

```
控制代码：01      (下面是待装入的数据)
计    数：XX      (装入字节数，0 表示 256 个)
装入地址：XX      (装入地址，按 LSB/MSB 次序排列)
          XX
装入数据：XX
          XX
          ⋮
控制代码：02      (下面是开始执行的地址)
          XX      (这个字节不用)
地    址：XX      (执行地址，按 LSB/MSB 次序排列)
          XX
控制代码：03—05   (下面的数据要跳过)
计    数：XX      (要跳过的字节数)
跳过数据：XX      (这个数据要跳过)
          XX
```

⋮

盒式磁带的记录格式

Level II 使用的格式如下：



选择驱动设备(磁带机 1 和磁带机 2) 和打开马达

```
        LD      A, 00           ;磁带机 1 的代码。
        LD      (37E4H), A      ;选择磁带机 1。
        LD      A, 04           ;命令值: 打开马达。
        OUT     (0FFH), A       ;启动马达,清除数据锁存器。
```

写一个已存在 A 寄存器内的字节

```
        PUSH    AF
        PUSH    BC
        PUSH    DE
        PUSH    HL              ;保存调用它的程序的寄存器。
        LD      L, 8            ;要写的位数。
        LD      H, A            ;数据字节放入 H 寄存器。
LOOP    CALL    CP              ;首先写入时钟脉冲。
        LD      A, H            ;得到数据字节。
        RLCA                    ;高位二进制位送入进位标志位。
        LD      H, A            ;保存已改变各二进制位位置的字节。
        JR      NC, WR          ;待写位是 0。一段空白磁带。
        CALL    CP              ;待写位是 1。写一个数据脉冲。
TEST    DEC     L               ;数据字节的各位是否已写完?
        JR      NZ, LOOP        ,没有! 转移到 LOOP。
```

```
        POP     HL          ;是! 恢复调用者的寄存器内容。
        POP     DE
        POP     BC
        POP     AF
        RET                 ;返回到调用它的程序。
WR      LD      B, 135      ;在写空带期间,延迟 135 个周期 (988 微秒)。
WR1     DJNZ    WR1
        JR      TEST        ;去测试是否还有位要写。
CP      LD      A, 05       ;命令值: 马达开, OUTSIG1。
        OUT     (0FFH), A   ;时钟脉冲开始。
        LD      B, 57       ;延迟 57 周期 (417 微秒), 给出时钟信号的一部
                            分。
CP1     DJNZ    CP1
        LD      A, 06       ;命令值: 马达开, OUTSIG2。
        OUT     (0FFH), A   ;时钟脉冲的第二部分。
        LD      B, 57       ;延迟 57 周期 (417 微秒), 给出时钟脉冲的一部
                            分。
CP2     DJNZ    CP2
        LD      A, 04       ;命令值: 马达开,没有 OUTSIG。
        OUT     (0FFH), A   ;开始擦除磁带。
        LD      B, 136      ;延迟 136 周期 (995 微秒), 给出时钟脉冲的尾
                            部。
CP3     DJNZ    CP3
        RET                 ;返回到调用它的程序。
从盒式磁带读下一个字节到 A 寄存器内
        XOR     A           ;清除指定的寄存器。
        PUSH    BC
        PUSH    DE
        PUSH    HL          ;保存调用它的程序的寄存器。
LOOP    LD      B, 8        ;要读的二进制位数。
        CALL    RB          ;读下一个位。装配到目前拼成的字节内。
        POP     HL
        DJNZ    LOOP        ;循环,直到 8 位装完为止。
        POP     DE
        POP     BC          ;恢复调用它的程序的寄存器。
        RET                 ;返回到调用它的程序。
RB      PUSH    BC
        PUSH    AF
RB1     IN      A, (0FFH)   ;读数据锁存器。
```

```
        RLA                 ;测试是空的/不空的磁带。
        JR      NC, RB1     ;空位,继续扫描直到遇见不空为止,假定这时是时钟
                            脉冲开始。
        LD      B, 57       ;在跳过时钟脉冲第一部分的时候,延迟 57 周期。
RB2     DJNZ    RB2
        LD      A, 04       ;命令值: 开马达,清除数据锁存器。
        OUT     (0FFH),A
        LD      B, 193      ;在通过时钟脉冲的尾部时,延迟 193 个周期。
RB3     DJNZ    RB3
        IN      A, (0FFH)   ;定出数据脉冲区。读这个数据脉冲。
        LD      B, A        ;保存数据脉冲。
        POP     AF          ;取回已积累的部分字节。
        RL      B           ;如果是空带, 送到进位位的数据脉冲将是一个 0,
                            如果不是空带,则将是 1。
        RLA                 ;把新的数据脉冲 (1 位)与字节的其余部分合并在
                            一起并存储起来。
        PUSH    AF
        LD      A, 04       ;命令值: 开马达,清除 OUTSIG。
        OUT     (0FFH), A   ;清除数据锁存器。
        LD      B, 240      ;延迟长到足够跳到数据脉冲的结束处。
RB4     DJNZ    RB4
        POP     BC
        POP     AF          ; A 寄存器内放有数据字节。
        RET
```

关闭马达

```
        LD      A, 00       ;命令值: 关马达。
        OUT     (0FFH), A   ;马达关闭。
        RET
```

磁盘输入/输出

这一节讨论的磁盘操作只是些基本原理，因为没有考虑磁盘空间的管理及通常有关磁盘输入/输出的其它一些功能。要介绍的是不经过 DOS 对磁盘任意区进行定位,读出和写入所必须的基本步骤。我们将假定读者对于由 DOS 提供的输入/输出的便利之处是很熟悉的,同时也知道没有通过 DOS 来写磁盘将会产生什么样的缺陷。

通常 TRS-80 Model I 系统带有的磁盘是单面的、35 磁道、$5\frac{1}{4}$ 英寸的小型驱动器。它们能够由比较多的磁道,如 40, 77 或 80 磁道的其它驱动器来代替，但是必须使用经过修改的 DOS 版本。已经有双面小型驱动器的产品出售,最终它们应该代替单面驱动器。

双密度驱动器是可买到的另一种型式的小型驱动器，但是像双面驱动器一样，它们需要修改过的 DOS 版本。

在这个例子中所用的编程形式称为编程输入/输出。这样称呼它是因为程序必须不断地监视控制器的状态，以便确定控制器是否准备好发送或接收下一个数据字节，因此每个字节是在程序控制下分别传送的。代替编程输入/输出的另一种方案是 DMA（存储器直接存取）。使用 DMA 方式，要告诉 DMA 控制器传送的字节数，开始传送的地址，这样控制器就可以控制数据的传送，腾出 CPU，使它可以执行其它的任务。在 Model I 系统中没有 DMA 的能力，因此必须使用编程输入/输出的方式。

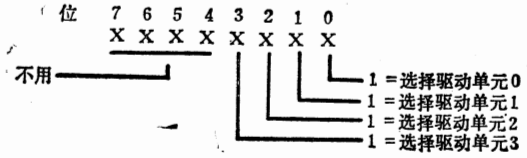
这个例子将假定正在使用的是一个 DOS 格式化的磁盘片。新的磁盘片是消了磁的。磁盘片在使用前必须经过格式化处理，格式化处理就是必须在每个扇区和磁道上记录它的磁道编号和扇区编号，以保证它们的唯一性。编号记录在每个扇区的数据区前面。加在每个扇区前面的编码信息可以有各种不同的规定，所以除非把磁盘片放在原来写它的相同型号的机器上，否则不是总能读得出来的。

像 Model I 的大多数输入/输出设备一样，磁盘也是映象到存储器的。有五个存储单元专供磁盘使用。它们是：

37E1H	磁盘驱动单元选择寄存器；
37ECH	命令/状态寄存器；
37EDH	磁道修改寄存器；
37EEH	扇区寄存器；
37EFH	数据寄存器。

除去磁盘驱动单元选择命令以外，所有磁盘命令都送到 37ECH 单元。如果正要发送的命令需要附加上像磁道编号和扇区编号这样的信息，则在命令发出以前，该数据应该存放在适当的寄存器内。也许已经注意到：命令寄存器和状态寄存器有相同的地址。因此，取状态(取 37ECH 内容)不能在发出一个命令(存入 37ECH) 后的 50 微秒内进行。

驱动单元选择是由存一个驱动单元屏蔽值到存储单元 37E1H 完成的。屏蔽值格式如下：

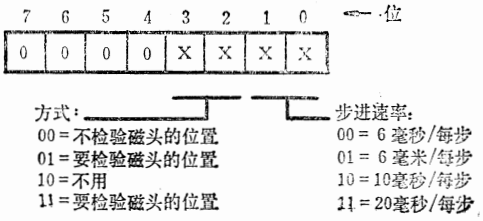


可以一次选择几个驱动单元。例如屏蔽值取 3，将选择设备 0 和设备 1。当任何一个驱动单元被选中时，全部驱动单元的马达都会自动启动。这个操作是由扩展接口机箱内的电路自动完成的。

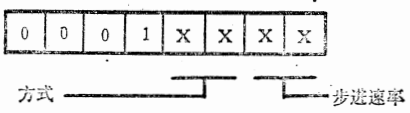
磁盘控制器命令

TRS-80 Model I 使用西方数字（Western Digital）公司制造的 FD1771B-01 软磁盘控制芯片。FD1771B 提供 12 个 8 位的命令。它们是：

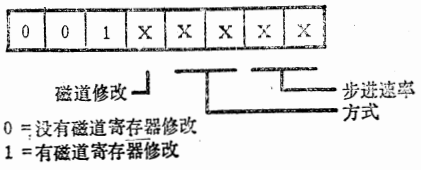
复原：把磁头定位到磁道 0。



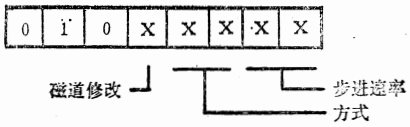
寻道：把磁头定位到由数据寄存器（37EFH）规定的磁道。



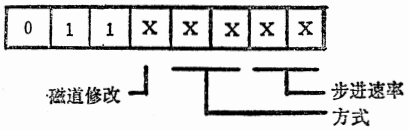
步进：按上次磁头运动的相同方向使磁头移动一步。



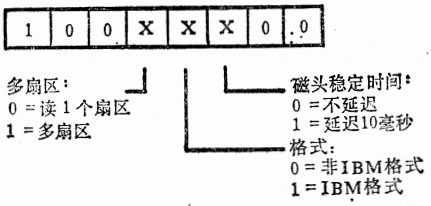
磁头向内步进：使磁头向最里面的磁道方向移动一个位置。



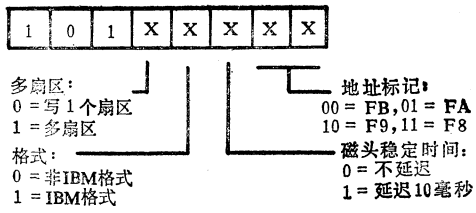
磁头向外步进：使磁头向最外面的磁道方向移动一个位置。



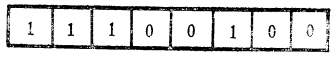
读数据：从扇区寄存器内的值所指定的扇区发送数据的下一个字节。



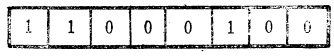
写数据：把数据寄存器内的数据字节发送到扇区的下一个位置，该扇区是由扇区寄存器内的值指定的。



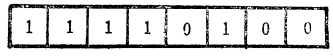
读磁道：读出由索引标记开始的整个磁道。



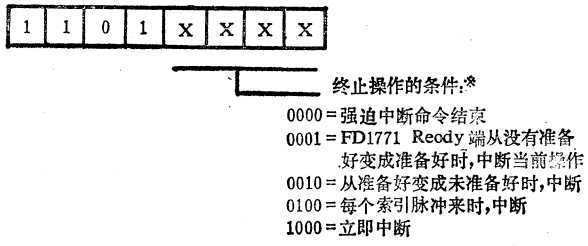
读地址：从磁头下面通过的下一个扇区读地址字段。



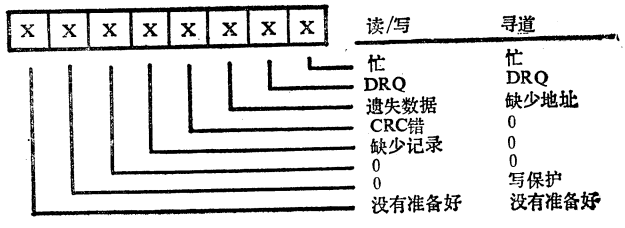
写磁道：写整个磁道，从索引标记开始并继续进行到遇到下一个索引标记为止。



强迫中断：如果下面四个条件中有一个满足，则终止当前的操作和(或)产生一个中断。



读状态：任何时候，只要读 37ECH 单元，就得到软磁盘控制器的状态。状态字有如下格式：



磁盘编程的详细说明

磁盘的编程能够分解成几个很容易处理的步骤。它们是：

---

* 原文图中的条件有错。——译注

1. 选择磁盘驱动单元和等待准备就绪。

2. 把磁头定位在想要的磁道上。

3. 对所需要的扇区发送读/写命令。

4. 传送扇区内有用的数据，每次一个字节。每次传送必须先测试一下，看看控制器是否有下一个数据字节，或者准备接收下一个数据字节。

这个程序演示从 25 磁道第三扇区读单个扇区。

```
        ORG     7000H
        LD      BC，256         ;字节计数。
        PUSH    BC              ; B＝1，C＝0。
        LD      HL，BUFF        ;缓冲器地址。
        LD      A，1            ;驱动单元选择屏蔽字(驱动单元 0)。
        LD      (37E1H)，A      ;选择驱动单元 0，开启马达。
        LD      D，25           ;磁道编号。
        LD      E，3            ;扇区编号。
        LD      (37EEH)，DE     ;确定磁道和扇区，磁道编号放到数据寄存器
                                  (37EFH)，扇区编号放到扇区寄存器。
        LD      A，1BH          ;寻道的操作码，不带检验(带检验时用 17H)。
        LD      (37ECH)，A      ;寻道请求放入命令寄存器。
        LD      B，6            ;询问状态以前，给控制器一个领会命令的机会。
DELAY   DJNZ    DELAY
WAIT    LD      A，(37ECH)      ;得到寻道操作的状态。
        BIT     0，A            ;测试控制器是否处于忙态。
        JR      NZ，WAIT        ;如果是，则不进行寻道。
        LD      A，88H          ;寻道结束。装入读命令，并且发送到控制器。
        LD      (37ECH)，A      ;
        LD      B，6            ;要求得到状态以前，给控制器一个领会命令的机
                                  会。
DELAY1  DJNZ    DELAY1
WAIT1   LD      A，(37ECH)      ;现在要取状态。
        BIT     1，A            ;现在有一个数据字节吗?
        JR      Z，WAIT1        ;没有，一直等待，直到有一个数据字节出现为止。
        LD      A，(37EFH)      ;有，装入数据字节。
        LD      (HL)，A         ;存进缓冲器。
        INC     HL              ;指向下一个缓冲器地址。
        POP     BC              ;恢复 BC 内容*。
        DEC     BC              ;测试 256 个字节有否传送完。
        PUSH    BC              ;保存新的 BC 内容*。
```

---

* 这两条指令在原文中没有，这里是译者为了避免程序发生错误而补充的。——译注

```
LD      A, B          ;把 B 和 C 合并，以便对这两个寄存器进行测
                       试。
OR      C
JR      NZ, WAIT      ;去取下一个字节。
:
```

DOS 出口

在第一章中已对 DOS 出口的一般概念作了讨论。DOS 出口是用于 Level II BASIC 和磁盘 BASIC 之间传递控制的一种方法。出口本身是放置在系统 ROM 区的一条调用 (CALL) 指令，它指向通讯区的一个固定地址。调用地址所包含的内容不是一条返回指令，就一定是转移到磁盘 BASIC 中另一地址的一条转移指令。在不带磁盘的 Level II 系统中，在 IPL 处理期间，这些被调用的单元都设置成返回指令。在磁盘系统中，执行 BASIC 命令之前是不对它们初始化的。在初始化后，调用单元内存放有转移到磁盘 BASIC 中的某些特定地址的转移指令。

DOS 出口这一术语实际上有两种不同的含义。一种 DOS 出口是一些从 ROM BASIC 到磁盘 BASIC 的调用，这些调用可以发生在输入阶段，也可以发生在执行一个系统级命令以及执行一个工作子程序*期间。这些出口使我们能够扩充 ROM 中的子程序。这些出口不安排在关键的地方，是为了能把一个完整的子程序硬掏出来，但是对于截取的大多数 ROM 子程序，这些出口都放置在很便于取用的地方。另一种 DOS 出口是磁盘 BASIC 出口。这种出口和前面讲的那一种出口是根本不同的。它们仅是在执行阶段内，当遇到磁盘 BASIC 的代号时才按要求进入。与这些代号有关的所有处理都包含在磁盘 BASIC 程序中。在 ROM 中没有执行这些代号的程序。

下面的这些说明是供 DOS 出口用的，与磁盘 BASIC 的出口不同。对于每个 DOS 出口，调用序列是不同的。在写一个程序去代替任意一个出口以前，必须先研究有关这个调用的那些代码，并要特别注意寄存器的运用。这里没有讨论在出口处发生的事情。如果有必要，可对磁盘 BASIC 的实用程序反汇编，并且研究分配给该出口的地址中的代码。在第六章中给出了截取这两种类型出口的例子。

所有这些地址都是针对 NEWDOS 2.1 版本的，TRSDOS 的地址是不同的。

Level II 地址 (Hex)	说　　明	DOS 出口 地址 (Hex)	磁盘 BASIC 地址 (Hex)
19EC	为了装入磁盘 BASIC 的错误处理程序所作的调用。错误编号必须在 E 寄存器中。	41A6	
27FE	开始 USR 的处理。	41A9	5679
1A1C	BASIC 开始工作。恰好在 'READY' 信息之前。	41AC	5FFC
0368	开始键盘输入。	41AF	598E
1AA1	当前语句代号化以后输入扫描程序。	41B2	6033
1AEC	修改程序语句表以后输入扫描程序。	41B5	5BD7
1AF2	重新初始化 BASIC 以后输入扫描程序。	41B8	5B8C
1B8C/1DBO	为了新的程序初始化 BASIC。在 END 时进行的	41BB	60A1

* 工作子程序 (Verb action roution) 这里是指 ROM BASIC 中的一些命令 (如 USR, END, STOP… 等)的执行程序。——译注

Level II 地址 (Hex)	说　　明	DOS 出口 地址 (Hex)	磁盘 BASIC 地址 (Hex)
	处理。		
2174	初始化系统输出设备时。	41BE	577C
032C	写系统输出设备时。	41C1	59CD
0358	扫描键盘。在每个 BASIC 语句执行结束处从 INKEY$ 调用。	41C4	59CD
1EA6	RUN NNN 处理的开始。	41C7	5F78
206F	开始 PRINT 处理。	41CA	5A15
20C6	PRINT# 或 PRINT 项目的处理时。	41CD	5B9A
2103	在 BASIC 输出操作期间跳到荧光屏的下一行时。	41D0	5B99
2108/2141	盒式录音机的 PRINT 开始和 PRINT TAB 处理时。	41D3	5B65
219E	开始 INPUT 处理。	41D6	5784
222D	在 READ 处理期间，读了一个变量时。	41 D9	5E63
2278	READ 处理结束。	41DF	579C
2B44	LIST 处理。	41DF	579C
02B2	SYSTEM 命令处理。	41E2	5B51

磁盘 BASIC 出口

在执行阶段内，不论何时遇到 BCH—FAH 范围内的一个代号，都会从 Level II 产生出这些出口。具有这些值的代号都已经分配给了完全由磁盘 BASIC 执行的语句。当找到一个给定的范围内的代号时，控制经工作子程序清单（见第四章）间接地传送到通讯区中相应的磁盘 BASIC 的出口。在工作子程序处理末尾控制返回到 Level II。

代号 (Hex)	工作子程序	通讯区地址 (Hex)	磁盘 BASIC 地址 (Hex)
E6	CVI	4152	5E46
BE	FN	4155	558E
E7	CVS	4158	5E49
B0	DEF	415B	5655
E8	CVD	415E	5E4C
E9	EOF	4161	61EB
EA	LOC	4164	6231
EB	LOF	4167	6242
EC	MKI$	416A	5E2D
ED	MKS$	416D	5E30
EE	MKD$	4170	5E33
85	CMD	4173	56C4
C7	TIME$	4176	5714
A2	OPEN	4179	6349
A3	FIELD	417C	60AB
A4	GET	417F	627C
A5	PUT	4182	627B
A6	CLOSE	4185	606F
A7	LOAD	4188	5F7B
A8	MERGE	418B	600B
A9	NAME	418E	6346

代号 (Hex)	工作子程序	通讯区地址 (Hex)	磁盘 BASIC 地址 (Hex)
AA	KILL	4191	63C0
NONE	&	4194	58B7
AB	LSET	4197	60E6
AC	RSET	419A	60E5
C5	INSTR	419D	582F
AD	SAVE	41A0	6044
9C	LINE	41A3	5756
C1	USR	41A9	5679

磁盘表格

在 TRS-80 Model I 系列中最常用的磁盘是 5¼ 英寸的单面单密度小型软磁盘驱动器。现在可买到各式各样的驱动器单元，并且也能在 TRS-80 上使用，然而必须修改某些硬件和软件。其它一些驱动单元如：5¼ 英寸的双磁头和双密度驱动器；8 英寸单、双磁头以及单、双密度的驱动器；还有各种存储量高达 20 兆字节的硬磁盘驱动器。

单、双磁头这一术语是指驱动单元中读/写磁头的数目。大多数微计算机系统使用单磁头驱动器，但是现在双磁头驱动器用得越来越普遍了。双磁头驱动器有单磁头驱动器两倍的容量，这是因为它能够存取两个磁盘表面，而不是只用一个。

双密度这一术语描述了所用的记录方式。在单密度记录方式中，每位是由一个时钟脉冲紧接着一个数据脉冲组成的；而在双密度记录方式中，如果重复出现数据脉冲，则可省去时钟脉冲。与单密度格式相比，利用这种记录方式，在一个磁道上能够写更多的扇区。所有的记录方法是由控制器和软件决定的，但是双密度驱动器省去了一些时钟脉冲，并且定时比较严格，因此不是所有的驱动器都能作双密度使用。

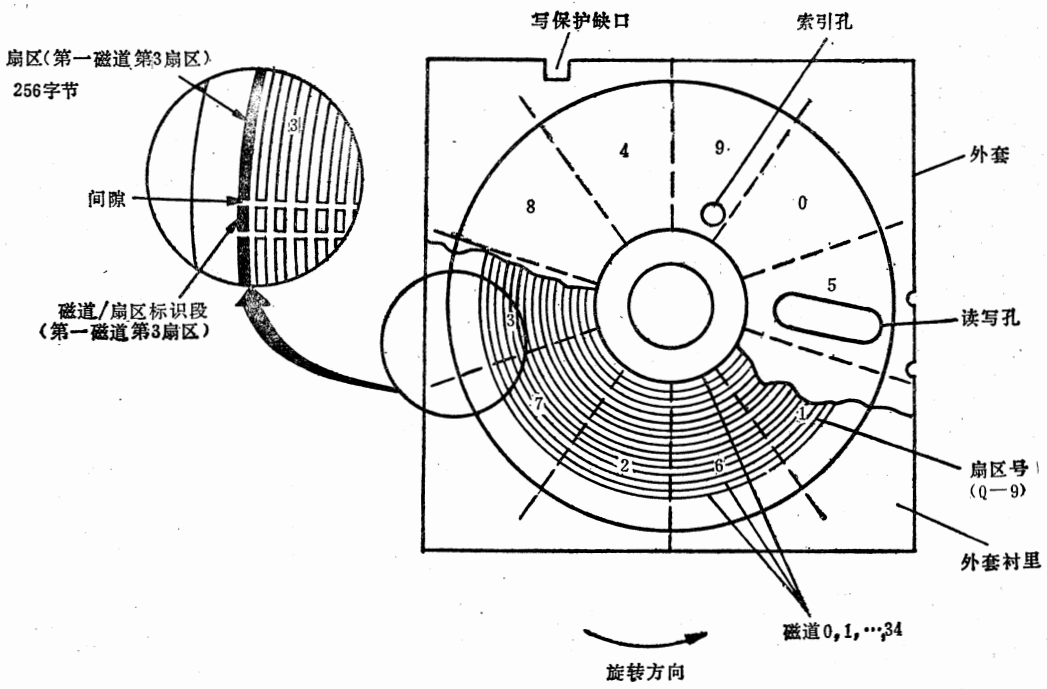
8 英寸驱动器实质上是同 5¼ 英寸驱动器相同的，不同的地方是 8 英寸磁盘通常只有一种磁道数量 (77 磁道)。像较小尺寸的驱动器一样，它们也有单密度和双密度两种。因为 8 英寸磁盘的半径比较大，所以每个磁道上有更多的扇区。8 英寸驱动器的典型磁道容量是 26 个 128 字节的扇区/每个磁道；15 个 256 字节的扇区/每个磁道；8 个 512 字节的扇区/每个磁道；4 个 1024 字节的扇区/每个磁道。

5¼ 英寸单密度的磁道容量是：20 个 128 字节的扇区/每个磁道；10 个 256 字节的扇区/每个磁道；5 个 512 字节的扇区/每个磁道；2 个 1024 字节的扇区/每个磁道。5¼ 英寸双密度驱动器的容量为：32 个 128 字节的扇区/每个磁道；18 个 256 字节的扇区/每个磁道；8 个 512 字节的扇区/每个磁道以及 4 个 1024 字节的扇区/每个磁道。

硬磁盘也有各种类型。一般来说，硬磁盘的容量更大，存取速度更快，同时传输速率更高。但是磁盘本身不能更换，没有一个活动的磁盘，文件的储备就成了一个很大的问题。解决的办法是加装第二个硬磁盘，但费用昂贵。

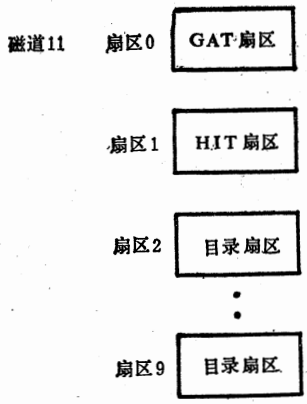
下面是一个 5¼ 英寸 35 磁道磁盘片的图例。

每个磁盘片可以有 35，40，77 或 80 个磁道，这要由所用的驱动器来决定。每个磁道有 10 个 256 字节的扇区。扇区的大小能够改变，每个扇区的大小可以从 2 个字节到 1024 个字节。因为 DOS 采用的扇区大小是 256 字节，所以为了能管理其它的不同于 256 字



节大小的扇区，就必须修改软件。Model I 使用一种部分 IBM 兼容的扇区格式。它不是百分之百的兼容，因为在 IBM 格式的磁盘片上磁道和扇区的编号是从 1 开始的，而 TRSDOS 是由 0 开始的。

DOS 使用一个文件目录，用来保存文件名及一些记录——分配给它们的磁道和扇区的记录。目录占据 11 号磁道的全部 10 个扇区。目录由三部分组成：表示可用扇区的一个磁盘分配图(磁道 11，扇区 0)；用索引表示的文件名，该索引允许从一个超前起始点去搜索目录(称为散列检索表 HIT，磁道 11，扇区 1)；以及目录扇区本身(磁道 11，扇区 2—扇区 9)。见下图。



像目录磁道一样，在磁盘片上还有一个专用的磁道。磁道 0 扇区 0 放有系统装入程序，它是在磁盘 IPL 工作期间用来装入 DOS 用的。由 ROM 内的 IPL 程序把系统装入程序读入 RAM 的 4200H—4300H 单元，然后 ROM 内的 IPL 把控制传递给系统装入程序，于是 DOS 就被装入到系统内。

磁盘磁道的格式

任何磁盘片在使用以前必须经过初始化处理，可以用 FORMAT 或 COPY（如果是用 TRSDOS 操作系统，要用 BACKUP）系统实用程序完成格式化这一工作。格式化处理是对原来已消磁的磁盘片进行初始化。格式化操作为每个可寻址的扇区写上扇区地址并加上同步字节，同步字节是供控制器用于定位特定地址的。另外，格式化操作规定了扇区的大小，每个磁道的扇区数

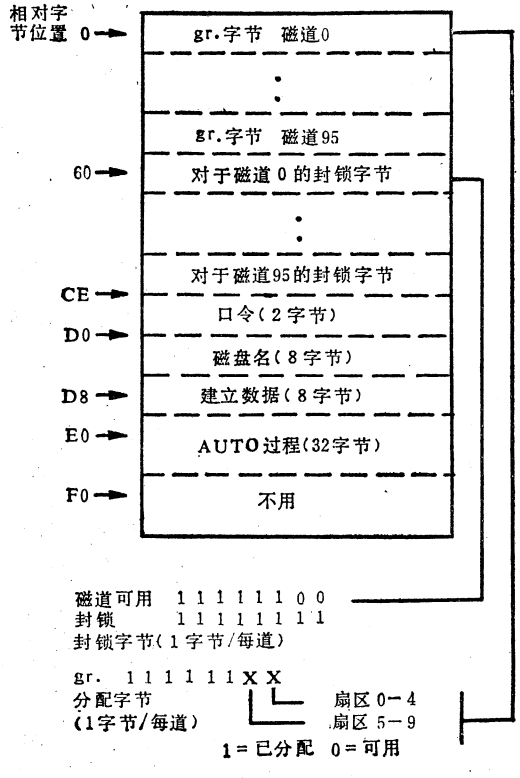
目以及扇区的实际次序。

虽然可以把小型软磁盘的扇区格式化成各种不同的大小，但是通常把扇区格式化成128，256，512，或1024字节。DOS 使用下面所示的磁道格式：

位置	字节数	内容(16进制)
索引	14	FF
	6	00
1扇区	1	FE(地址标记)
	1	磁道编号
	1	磁头编号
	1	扇区编号
	1	扇区长度代码 00 = 128字节 01 = 256字节 02 = 512字节 03 = 1024字节
10扇区/每磁道		
扇区次序是0,5,1,6,2,7,3,8,4,9	2	CRC校验
	11	FF；}这是扇区0的情况
	1	A0；
		(其它扇区为12个FF)
	1	FA(数据区标记)
	256	数据
	2	CRC校验
	12	FF：(不包括最后的第9磁道
	6	00：第9磁道共有130个字节的FF)

gr. 分配表 （磁道 11 扇区 0）

前面我们已经提到了 DOS 使用的文件目录系统。依靠这个目录系统，DOS 能够按需要动态分配磁盘空间。并且，它还必须能够重新使用已经释放的和并非长期需要的那些磁盘空间。用来保存已分配的和可用磁盘空间的磁道的基本记录就是 gr. 分配表 (GAT——Granulae Allocation Table)。显然，如果要保持一个永久性的记录，则 GAT 数据必须存放在机器的外部。GAT 扇区就是用来存放这些信息的存储区域。



在叙述磁盘的时候，已对磁道和扇区作了定义。这些术语将重新定义成在 DOS 中用的专用名词——gr. (granule)。一个 gr. 为5个扇区，或者说是半个磁道。gr. 是磁盘空间分配和重新分配的最小单位。gr. 编号从0到 N，其中 N 是磁盘片上磁道数目的函数。所有已分配的 gr. 记录都保留在 GAT 扇区。回顾一下我们在前面讲过的磁盘容量，就能计算出一个磁盘片上的 gr. 的数目应为

gr. 数 = (磁道数目 × 10)/5

使用一个35磁道的磁盘驱动器，对

于一个不带 DOS 的磁盘，每磁道有 10 个扇区，每个 gr. 有 5 个扇区，由此可算出每个磁盘片有 70 个 gr.

GAT 扇区分成三部分。第一部分是实际的 GAT 表格，其中保存着已分配的 GAT 记录。第二部分保存有磁道封锁表，第三部分则是系统初始化信息。

散列检索表 （磁道 11 扇区 1）

散列检索（Hash Index）是一种迅速定位一个文件的方法，它不必搜索该文件目录前面的所有目录扇区。每个文件有一个根据它的名字算出的唯一的数值。这个数值称为散列代码。在目录系统中一个特定的扇区存有该磁盘片上所有有效文件的散列代码。当建立一个文件时，它的散列代码就存进了散列扇区对应于该文件目录的位置。注意，散列代码的位置并不给出文件的位置，而只给出它的目录扇区的位置。当删去一个文件时，它的散列代码就要从散列扇区中消去。

对文件定位首先要根据文件名算出它的散列值，然后根据这个值搜索散列检索扇区。如果没有找到，则说明该文件不存在。如果找到了散列代码，那么就用散列检索扇区中该代码的位置，计算出记有该文件名登记项的目录扇区的地址。

散列代码值的范围的 01H—FFH。它们是根据 11 个字符的文件名算出来的，并且文件名已经作了向左对齐、填充空格的处理。任何一个文件名扩展都是文件名的最后 3 个字符。用于计算散列值的程序如下：

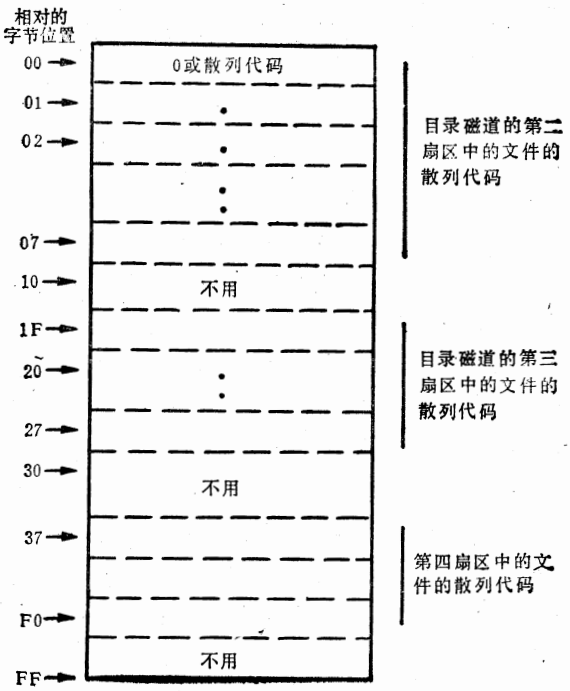
```
        LD      B, 11       ;用作散列计算的字符数。
        LD      C, 0        ;散列寄存器清 0 。
LOOP    LD      A, (DE)     ;得到名字的第一个字符。
        INC     DE          ;指向下一个字符。
        XOR     C           ;散列寄存器与第一个字符逻辑异或。
        RLCA                ;上述结果乘 2 。
        LD      C, A        ;新的散列寄存器值。
        DJNZ    LOOP        ;散列计算全部字符。
        LD      A, C        ;得出散列值。
        OR      A           ;不允许为 0 。
        JP      NZ, DONE*   ;结果不为 0 ,出口,散列值在 A 中。
        INC     A           ;结果为 0 ,强制散列值为 1 。
DONE                        ;出口,散列值在 A 中。
        ⋮
```

在散列扇区中，代码的空间是顺序分配的，并可以从任意点上开始。如果散列扇区已满，则给出一个 DOS 错误代码 1AH，否则以循环方式扫描散列扇区，直到找到一个可用的(零)登记项为止。

并不是散列扇区内的所有字都被用到。在 10H—1FH，30H—3FH，50H—5FH···等等范围内的地址是不用的。只分配那些以数字 00H—07H，20H—27H···等等为结束

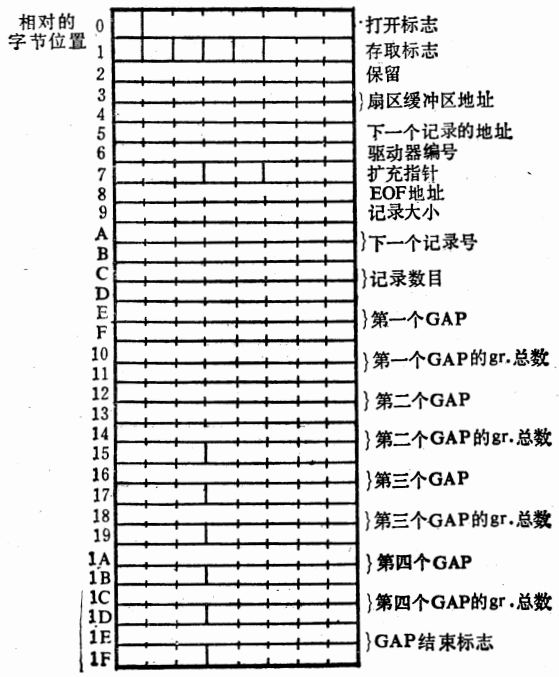
* 原文错写为 JMP DONZ。——译注

的地址。这样做加快了由散列代码值地址计算目录扇区编号的速度。散列扇区表示如下：



磁盘设备控制块（DCB）

每个磁盘文件有一个与它有关的 32 字节的 DCB，DCB 被定义在用户的内存空间。



当一个文件被打开时，DCB 必须包括文件名，文件名扩展(如果有的话)，以及一个任选的驱动器说明。作为 OPEN 语句处理的一部分，用复制一部分目录项到 DCB 中的方法把 DCB 初始化，以便进行 READ 或 WRITE 操作。初始化以后如前页所示。其中：

字节 0　　位 0—6：保留。
　　位 7　　：0＝文件未打开。
　　1＝文件已打开。

字节 1　　位 0—2：存取许可标志。
　　位 3　　：保留。
　　位 4　　：0＝扇区缓冲器可用。
　　1＝在使用以前清除扇区缓冲器。
　　位 5　　：0＝在当前缓冲内寻找记录。
　　1＝无条件读下一个扇区。
　　位 6　　：保留。
　　位 7　　：0＝扇区 I/O。
　　1＝逻辑记录 I/O。

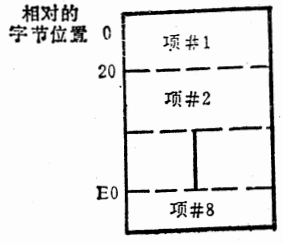
字节 2　　保留。
字节 3—4　　扇区缓冲器地址，依 LSB/MSB 次序排列。
字节 5　　指向缓冲器内的下一个记录。
字节 6　　驱动器编号。
字节 7　　位 0—2 ：扩充项的扇区编号减 2。
　　位 3—4 ：保留。
　　位 5—7 ：相对目录中的主要项的偏移值/16。
字节 8　　在最后扇区中指向文件的结束。
字节 9　　记录大小。
字节10—11　　下一记录编号，依 LSB/MSB 次序排列。
字节12—13　　文件中的记录数。
字节14—15　　第一个 GAP。
字节16—17　　第一个 GAP 中分配的 gr. 总数。
字节18—19　　第二个 GAP。
字节20—21　　第二个 GAP 中分配的 gr. 总数。
字节22—23　　第三个 GAP。
字节24—25　　第三个 GAP 中分配的 gr. 总数。
字节26—27　　第四个 GAP。
字节28—29　　第四个 GAP 中分配的 gr. 总数。
字节30—31　　GAP 字符串标志 (FFFF) 的结束。

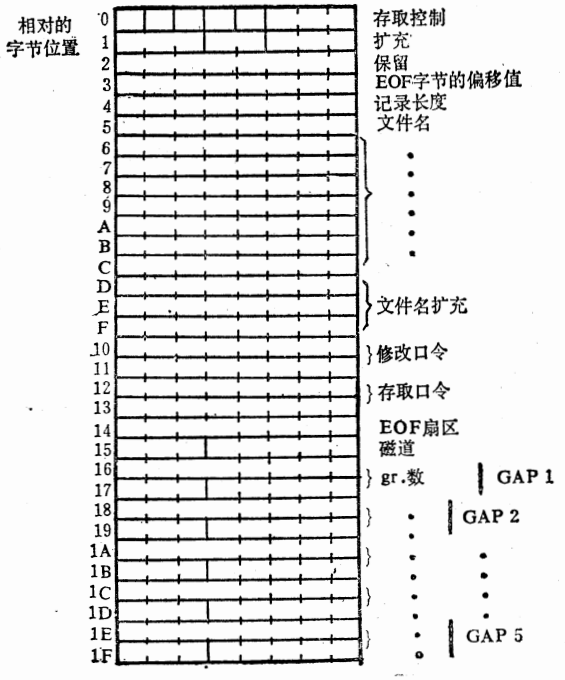
磁盘目录 (磁道 11 扇区 2—磁道 11 扇区 9)

目录扇区放有存取磁盘文件时使用的文件说明。文件说明包括以下内容：文件名，口令，以及该文件所占有的磁盘地址清单。目录扇区分成 8 个固定长度的分区，每个分区

为 32 个字节。每个分区装有一个文件说明。空的分区是用一个放在分区第一字节的标志表示的。

当用 DOS 的 OPEN 或 INIT 调用创建一个文件时，就在目录中分配一个空间。在这样一种分配空间的方法中没有特定的次序，这是因为所用的目录扇区编号是根据文件名推导出的散列代码决定的。扇区中的分区空间是顺序分配的。





上图为目录扇区的图例，其中：

字节 0　　位0—2：文件存取控制标志。

000——不受约束的存取。

001——KILL/RENAME/WRITE/READ/EXECUTE 存取。

010——RENAME/WRITE/READ/EXECUTE 存取。

011——保留。

100——WRITE/READ/EXECUTE 存取。

101——READ/EXECUTE 存取。

110——仅供 EXECUTE 存取。

111——受约束的文件，不能存取。

位 3 = 0，文件是可显示的。

= 1，文件是不可见的。

位 4 = 0，这个登记项是可用的(即文件已删除)。

= 1，这个登记项不可用。

位 5：保留。

位 6 = 0，用户文件。

= 1，系统文件。

位 7 = 0，主要的登记项。

= 1，扩充的登记项。

字节 1　仅供扩大项用。

位 0 — 3，主扇区中相对于该文件项的字节偏移值/10。

位 4 — 7，主项的扇区编号减 2。

字节 2　保留。

字节 3　位 0 — 7，相对于最后一个扇区中文件结束的字节偏移值。

字节 4　位 0 — 7，记录长度。

字节 5—12　ASCII 码表示的文件名，文件名向左对齐，其余用空格补足。

字节13—15　ASCII 码表示的文件名扩展，向左对齐，其余用空格补足。

字节16—17　修改口令(已编码)。

字节18—19　存取口令(已编码)。

字节20—21　文件的最后一个扇区编号。按 LSB/MSB 次序排列。

字节22—31　5 个 2 字节的登记项目，它们称为 gr. 分配对 (GAP——Granule Assignment Pairs)。每个 GAP 由一个起始磁道编号 (字节 1) 和一个邻接 gr. 数的计数(字节 2) 组成。这些按适当次序排列的 GAP 字符串规定了分配给一个文件的磁盘地址。如果分配的 GAP 不多于 5 个，则 GAP 字符串的结束将以字节 1 和字节 2 中的 FFH 来表征，对只有一个 FEH 的情况，则后面紧接着的字节就是另一个目录扇区的磁盘地址，这个扇区包含了 GAP 的剩余部分。包含有扩充 GAP 的目录项称为扩充项，它只包含 GAP 字符串的延伸部分。对于可以分配的扩充项数目没有限制。

GAP 字节的格式如下：

第一个字节　位 0—7 是下列情况之一：

a) 如果第一字节的内容小于 FEH，就假定它是磁道的编号。

b) 如果不再有 GAP 项，则第一字节内容为 FFH。这是 GAP 串的结束。

c) 如果在一个扩充扇区中还有 GAP 项，则第一字节内容是 FEH。而下一个字节就装有扩充扇区的地址。

第二个字节　这个字节的解释取决于前一字节的内容。

如果前一字节的内容为 FFH，则这个字节也是 FFH。

如果前一字节的内容为 FEH，则

位 0 — 3，保留有扩充扇区的扇区编号减 2。

位 4 — 7，在扩充扇区内相对于带有 GAP 剩余部分的项的字节偏移值/10。

如果前一个字节<FEH，则

位 0 — 3，是邻接的 gr. 数 −1。

位 4　，是一个标志，它指出在开始的磁道中，已分配的是第一个还

是第二个 gr.。

位 4 = 0，则分配了第一个 gr.。

位 4 = 1，则第二 gr. 从已分配的扇区开始。

第二个字节值可以从 0 变化到 1FH。

下面是 GAP 串的一个例子：

字节 22=23H	文件从 23 磁道开始
字节 23=06H	有 7 个已分配的 gr. 磁道 (23) 扇区 (0—9)，磁道 (24) 扇区 (0—9)，磁道 (25) 扇区 (0—9)，磁道 (26) 扇区 (0—4)
字节 24=15H	文件在 15 磁道继续下去
字节 25=13H	有 4 个分配的 gr.： 磁道 (15) 扇区 (5—9)， 磁道 (16) 扇区 (0—9)， 磁道 (17) 扇区 (0—4)。
字节 26=FFH	GAP 串结束。
字节 27=FFH	GAP 串结束。

第四章　地址与表格

系统存储器分配图

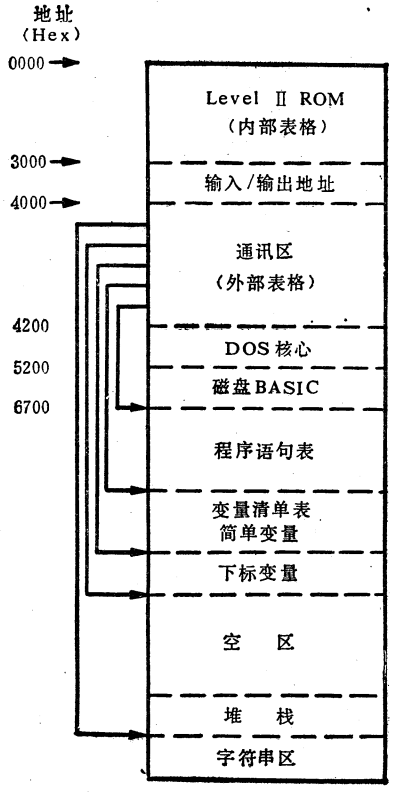


图 4.1　系统存储器分配图

Level II 内部表格

内部表格是指那些驱留在 Level II 系统中的清单和表格。因为它们是驱留在 ROM 中的，故而其内容和地址都是固定的。内部表格可供 BASIC 用于句法分析、数据变换，在表达式计算过程中也要用到它，同时在执行如 FOR 和 IF 语句时也使用。

保留字表　(1650H—1821H)

这张表格包括了所有为 BASIC 解释程序使用的保留字。每一项包括一个位 7 为 1 的保留字。在输入阶段内扫描进入的语句行，以确定是否是这张清单中的字。当发现是其中任何一个时，就用一个表示它的代号来代替它。用 80H 加上检索此表时找到保留字

处的指针就能算出代号。这些保留字的清单及其代号值如下：

保留字	代号 (Hex)	保留字	代号 (Hex)	保留字	代号 (Hex)
END	80	★CLOSE	A6	LPRINT	AF
SET	83	★NAME	A9	PRINT	B2
RANDOM	86	★RSET	AC	LLEST	B5
INPUT	89	★SAVE	AD	CLEAR	B8
LET	8C	★DEF	B0	NEW	BB
IF	8F	CONT	B3	★FN	BE
RETURN	92	DELETE	B6	USR	C1
ELSE	95	CLOAD	B9	STRING$	C4
DEFSTR	98	TABC	BC	★TIME$	C7
DEFDBL	9B	USING	BF	THEN	CA
ERROR	9E	ERL	C2	+	CD
ON	A1	INSTR	C5	/	C0
★GET	A4	MEM	C8	OR	D3
★LOAD	A7	NOT	CB	<	D6
★KILL	AA	—	CE	ABS	D9
FOR	81	↑	D1	POS	DC
CLS	84	>	D4	LOG	DF
NEXT	87	SGN	D7	SIN	E2
DIM	8A	FRE	DA	PEEK	E5
GOTO	8D	SQR	DD	★CVI	E6
RESTORE	90	EXP	E0	★EOF	E9
REM	93	TAN	E3	★MKI$	EC
TRON	96	SYSTEM	AE	CINT	EF
DEFINT	99	POKE	B1	FIX	F2
★LINE	9C	LIST	B4	VAL	F5
RESUM	9F	AUTO	B7	LEFT$	F8
★OPEN	A2	CSAVE	BA	/	FB
★PUT	A5	TO	BD	★CVS	E7
★MERGE	A8	VARPTR	C0	★LOC	EA
★LSET	A8	ERR	C3	★MKS$	ED
RESET	82	POINT	C6	CSNG	F0
★CMD	85	INKEY$	C9	LEN	F3
DATA	88	STEP	CC	ASC	F6
READ	8B	*	CF	RIGHT$	F9
RUN	8E	AND	D2	★CVD	E8
GOSUB	91	=	D5	★LOF	EB
STOP	94	INT	D8	★MKD$	EE
TROFF	97	INP	DB	CDBL	F1
DEFSNG	9A	RND	DE	STR$	F4
EDIT	9D	COS	E1	CHR$	F7
OUT	A0	ATN	E4	★MID$	FA
★FIELD	A3				

★ 是磁盘 BASIC 用的代号

操作符的优先值表 (189AH—18A0H)

这张表格包括了在计算一个表达式时用来确定算术运算次序的数值。当扫描表达式

时，把每对操作符/操作数加上前一个操作数的优先值存入堆栈。当找到一个比前一个优先级别高的操作符时，就执行现在的操作，同时给出一个中间结果值，并推入堆栈，这个中间结果带给后面的运算。下表所示的关系运算值是算出来的。

操作符	功能	优先值
↑	幂	7F
*	乘	7C
/	除	7C
+	加	79
−	减	79
	关系运算	64
<=	关系运算	06
< >	关系运算	05
>=	关系运算	04
<	关系运算	03
=	关系运算	02
>	关系运算	01
AND	逻辑运算	50
OR	逻辑运算	46

算术子程序 （18ABH—18C8H）

这里实际上前前后后有三张表格。在求表达式值的过程中，当找到一个较高优先级算符时，这些表格用来计算中间结果值。

算术子程序地址（Hex）

	整数	单精度	双精度	字符串
加	0BC2	0716	0C77	298F
减	0BC7	0713	0C70	没有
乘	0BF2	0847	0DA1	没有
除	2490	08A2	0DE5	没有
比较	0A39	0A0C	0A78	没有

数据转换子程序 （18A1H—18AAH）

这些程序是把 WRA1 中的值从一种数据类型变换成另外一种类型。当已经得出一个中间计算值时，由表达式求值程序来调用这些程序，使结果的数据类型同表达式的其余部分一致。

变换子程序地址

要得到的数据类型	地址（Hex）
字符串	0AF4
整数	0A7F
单精度	0AB1
双精度	0ADB

工作子程序 （1822H—1899H）

有两个工作子程序地址表。第一张表是开始执行一个新语句时供执行驱动程序使

用。它包括了代号为 80H—BBH 的工作子程序的地址。把语句的第一个代号作为在地址 1822H—1899H 的表格内的一个检索标志(它的范围为 0—60)，在这个表内寻找一个要执行的工作子程序的入口地址。如果语句不用一个代号开始，就作赋值语句处理。第二张表包括那些只能出现在等号右边的工作子程序的地址。如果在求表达式值的过程中，碰到一个处于 D7H—FAH 范围内的代号，就把这个代号作为在地址 1608H—164FH 的表内的一个检索标志，在那里寻找待执行的工作子程序的地址。对于代号 BCH—D6H 没有地址清单，这是因为它们要与另外一些代号联在一起使用，BCH—D6H 跟在那些需要它们和处理它们的代号后面。

(表地址 1822H—1899H)

代号 (Hex)	工作子程序名	地址 (Hex)	代号 (Hex)	工作子程序名	地址 (Hex)
80	END	1DAE	81	FOR	1CA1
82	RESET	0138	83	SET	0315
84	CLS	01C9	85	CMD	4173
86	RANDOM	01D3	87	NEXT	22B6
88	DATA	1F05	89	INPUT	219A
8A	DIM	2608	8B	READ	21EF
8C	LET	1F21	8D	GOTO	1EC2
8E	RUN	1EA3	8F	IF	2039
90	RESTORE	1D91	91	GOSUB	1EB1
92	RETURN	1EDE	93	REM	1F07
94	STOP	1DA9	95	ELSE	1F07
96	TRON	1DF7	97	TROFF	1DF8
98	DEFSTR	1E00	99	DEFINT	1E03
9A	DEFSNG	1E06	9B	DEFDBL	1E09
9C	LINE	41A3	9D	EDIT	2E60
9E	ERROR	1FF4	9F	RESUME	1FAF
A0	OUT	2AFB	A1	ON	1F6C
A2	OPEN	4179	A3	FIELD	417C
A4	GET	417F	A5	PUT	4182
A6	CLOSE	4185	A7	LOAD	4188
A8	MERGE	418B	A9	NAME	418E
AA	KILL	4191	AB	LSET	4197
AC	RSET	419A	AD	SAVE	41A0
AE	SYSTEM	02B2	AF	LPRINT	2067
B0	DEF	415B	B1	POKE	2CB1
B2	PRINT	206F	B3	CONT	1DE4
B4	LIST	2B2E	B5	LLIST	2B29
B6	DELETE	2BC6	B7	AUTO	2008
B8	CLEAR	1E7A	B9	CLOAD	2C1F
BA	CSAVE	2BF5	BB	NEW	1B49

(表地址 1608H—164FH)

代号 (Hex)	工作子程序名	地址 (Hex)	代号 (Hex)	工作子程序名	地址 (Hex)
D7	SGN	098A	D8	INT	0B37
D9	ABS	0977	DA	FRE	27D4

续表

代号 (Hex)	工作子程序名	地址 (Hex)	代号 (Hex)	工作子程序名	地址 (Hex)
DB	INP	2AEF	DC	POS	27F5
DD	SQR	13E7	DE	RND	14C9
DF	LOG	0809	E0	EXP	1439
E1	COS	1541	E2	SIN	1547
E3	TAN	15A8	E4	ATN	15BD
E5	PEEK	2CAA	E6	CVI	4152
E7	CVS	4158	E8	CVD	415E
E9	EOF	4161	EA	LOC	4164
EB	LOF	4167	EC	MKI$	416A
ED	MKS$	416D	EE	MKD$	4170
EF	CINT	0A7F	F0	CSNG	0AB1
F1	CDBL	0ADB	F2	FIX	0B26
F3	LEN	2A03	F4	STR$	2836
F5	VAL	2AC5	F6	ASC	2A0F
F7	CHR$	2A1F	F8	LEFT$	2A61
F9	RIGHT$	2A91	FA	MID$	2A9A

错误代码表 (18C9H—18F6H)

Level II 中，打印出的错误代码是用错误编号作为指针，去检索两个字母的错误简写表来进行解释的。错误代码表的格式如下：

错误编号 (Hex)	代码	原　　因	起因地址 (Hex)
0	NF	有 NEXT 语句缺少 FOR 语句	22C2
2	SN	语法错误(原因很多)	DA, 2C7 EEF, 1C9E 1D32, 1E0E 1E66, 2022 235B, 2615 2AE9, 2DE2
4	RG	有 RETURN 无 GOSUB	1EEC
6	OD	数据出界(读)	2214, 22A2
8	FC	非法调用函数	1E4C
A	OV	数字溢出	7B2
C	OM	内存出界	197C
E	UL	错误的行号	1EDB
10	BS	数组下标出界	273F
12	DD	重复定义数组	2735
14	/0	被 0 除	8A5, DE9 1401
16	ID	INPUT 使用不当	2833
18	TM	数据类型不符	AF8
1A	OS	串空间出界	28DD
1C	LS	字符串太长	29A5
1E	ST	文字串库表格已满	28A3
20	CN	不能继续	1DEB
22	NR	执行了 ON ERROR 语句，但没有 RESUME 语句	198C
24	RW	没有执行 ON ERROR 语句，但有 RESUME 语句	1FB4

续表

错误编号 (Hex)	代码	原　　因	起因地址 (Hex)
26	UE	非法的错误代码	2005
28	MO	遗漏操作数	24A2
2A	FD	盒式磁带数据错误	218C
2C	L3	在 Level II 控制下企图用磁盘 BASIC 语句	12DF

Level II 外部表格

Level II 所用的外部表格是那些保存在 RAM 中的表格。把它们存在 RAM 中是因为它们的内容、容量、以及它们的地址或许要改变。对于每个外部表格都有一个指针保存在通讯区内。

变量类型表 (4101H—411AH)

这个表格是供 BASIC 解释程序使用的，用它确定每个变量的数据类型的模式(整数，字符串，单精度和双精度)。虽然永远不会去移动它，但是当遇到一个 DEF 说明语句时，其内容是会改变的，所以它必须放在 RAM 中。它仅仅是带有固定地址的 RAM 表，因此在通讯区中没有指向它的指针。该表长度为 26 个字，它是用变量名的第一个字符作索引来检索的。在这个表内的每一项包括一个表示该变量类型的代码，如 02——整数，03——字符串，04——单精度，08——双精度。

在 IPL 处理期间，变量表被初始化为所有的变量均为 04。它表示如下：

地址 (Hex)	字母	类 型	地址 (Hex)	字母	类 型
4101	A	04	4102	B	04
4103	C	04	4104	D	04
4105	E	04	4106	F	04
4107	G	04	4108	H	04
4109	I	04	410A	J	04
410B	K	04	410C	L	04
410D	M	04	410E	N	04
410F	O	04	4110	P	04
4111	Q	04	4112	R	04
4113	S	04	4114	T	04
4115	U	04	4116	V	04
4117	W	04	4118	X	04
4119	Y	04	411A	Z	04

程序语句表 (PST)

程序语句表放有作为程序输入的 BASIC 语句。因为程序语句表是驻留在 RAM 中的，而且它的起点也许要随系统不同而有所改变，所以在通讯区的 40A4H 地址有一个指向它的指针。对于打入的每一行，要把它转换成相应的代号(称为代号化)，并存放在 PST 中。语句依行号递增的次序存入，而不管它们输入的次序如何。每一项由指向下一行的两字节指针开始，接着是相当于该行行号的两字节整数，然后是 BASIC 语句的文本。语句正文用称为语句结束标志或 EOS (End Of Statment) 标志的单个 0 字节束结束。PST

的终止地址保存在 40F9H。它是用两字节的 0 来结束的。

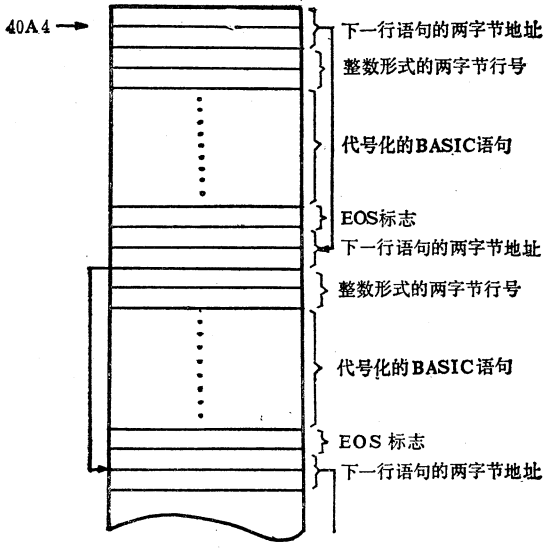


图 4.2　程序语句表（PST）

下面给出两行语句及其在 PST 中的表示形式：

100　A = COS (1.6)

110　IF A > 0.5 THEN 200

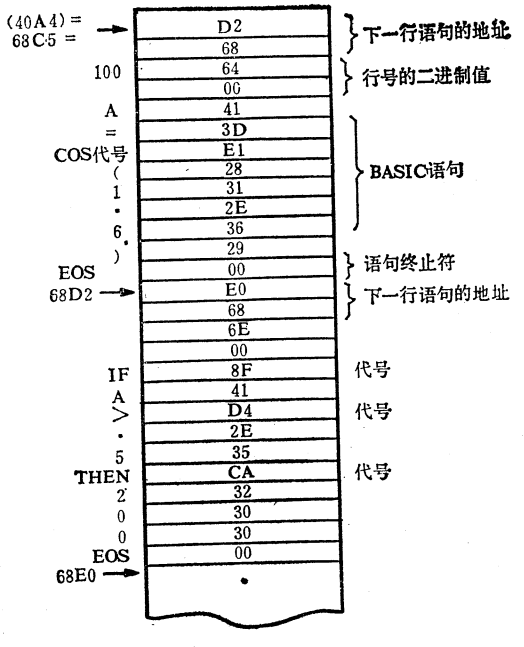


图 4.3　所举例子的 PST

变量表（VLT）

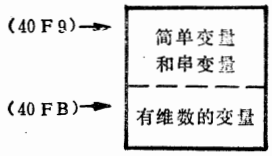
这张表格包括分配给一个 BASIC 程序的全部变量。此表内部分成两个部分。一个

部分包括所有的非下标变量和串变量，而另一部分包括所有的下标变量的值。象 PST 一样，VLT 也是驻留在 RAM 内的，而且在通讯区内有两个指针。40F9H 单元包含有第一部分的首地址，而 40FBH 单元包含有第二部分的首地址。VLT 的起始地址可以认为是 PST 的终点。

一个变量不论定义在哪一个部分，每一项的前三个字节都具有相同的格式。第一个字节为类型代码 (2, 3, 4 或 8)，它还能作为数据项的长度。第二、第三字节包括依后一个字符/前一个字符的次序放置的变量名。接着就是依 LSB/MSB 次序放置的变量本身的值；如果该变量为一个串变量，就是一个指向字符串区域中该串变量的指针。

第二部分包括全部的有维数组。这些项有同样的三字节标题，接着是一个确定数组范围的标题。第二个标题后面是数组的数值，它们按列为主的次序排列存储。

当遇到这些变量时(在一个 DIM 语句中，或者在赋值语句的任何部分)，就将它们分配在 VLT 空间内。它们不以字母数字的次序排列。因为空间是按需要分配的，所以有可能使那些预先定义的变量向下移动。例如，如果已经定义了 A，B 和 C(15)，后面又接着定义了一个 D，因为表的第一部分要增加 D，所以 C(15) 应该往下移动。这将强迫第二部分作移动。



数组是以列为主的次序排列存放。按照这种次序，则最左边的下标最先改变。例如，数组 A(2,3) 应该以如下所示的次序存放在存储器中：

A(0, 0), A(1, 0), A(2, 0),···A(0, 3), A(1, 3), A(2, 3)

任何一个元素的下标都能用下列公式来计算：

下标=(((LRI*0) + URI)*LMI) + UMI)*LLI) + ULI

其中 LRI = 右下标的极限

LMI = 中下标的极限

LLI = 左下标的极限

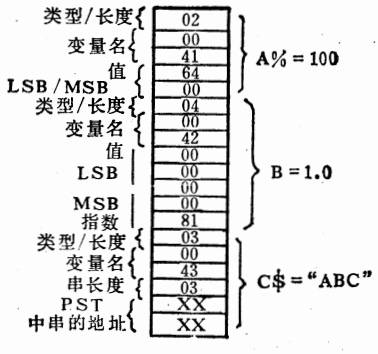


图 4.4a) 简单变量和串变量的存放格式

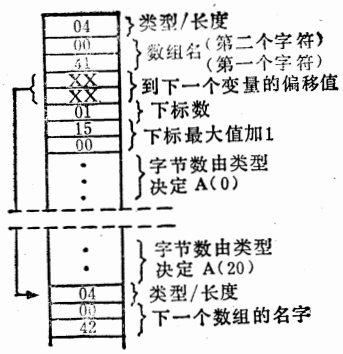


图 4.4b) 一维数组：DIM A(20)

URI = 用户的当前右下标

UMI = 用户的当前中下标

ULI = 用户的当前左下标

用于计算这些下标的程序可以在地址 2595H 到 27C8H 找到。

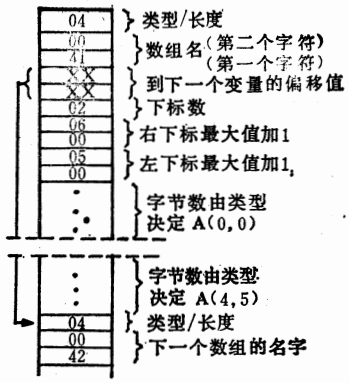


图 4.4c) 二维数组：DIM A(4, 5)

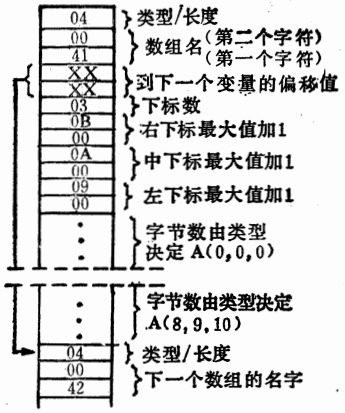


图 4.4d) 三维数组：DIM A(8, 9, 10)

文字串库表 (LSPT) (40B3H—40D2H)

此表是供 BASIC 使用的，它记录了中间结果字符串(象字符串相加和某些打印操作等字符串操作所产生的中间结果字符串)。该表有 11 个 3 字节依顺序分配的项目。表的开始有一个 2 字节的指针，它指向下一个可用项。在 IPL 期间，对该表初始化，以指明清单的首项。

每一项包括通常（但不是必然的）在 PST 内的字符串长度和地址。这些项是由顶向下的方式分配的，而从底部向上的方法来取出的。指向下一个可用项的指针保持在 40B3H 单元内。如果此表溢出，就会给出一个 ST 错误。

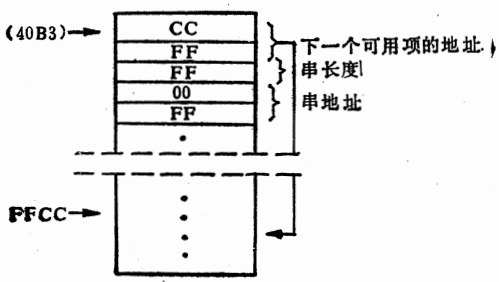


图 4.5 文字串库

通讯区 (4000H—4200H)

通讯区已规定为 RAM 的 4000H 到 4200H 单元。这些地址给出的定义，我们并不认为十分精确。实际上，只有该区域中一部分地址的使用符合通讯区这一含义。选用这一片分界区是因为它们表示了 ROM 的结束和 RAM 中 DOS 的大概的开始地址。在一个不带磁盘的 Level II 系统中没有 DOS，因而象 PST，VLT 等表格应该从比较低的

地址开始。但是他们仍将在 4200H 单元以上的地方开始存放，并把那些区域看作为保留区，因此，PST，VLT 等表格就很安全了。

通讯区有许多用途是以前我们所没有讨论过的。图 4.6 给出了包括这些用途在内的各主要部分。接着的表 4.1 是通讯区内全部字节的说明以及它们已知的用途。

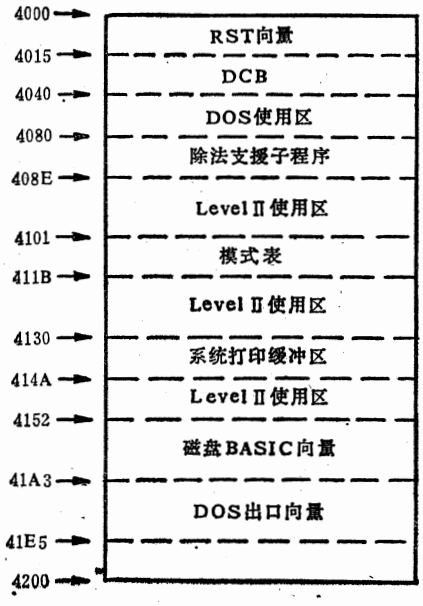


图 4.6　通讯区

表 4.1　通讯区地址表

地址 (Hex)	Level II 内容	DOS 内容	说　明
4000	JP 1C96H	.....	RST 8 向量
4003	JP 1D78H	.....	RST 10H 向量
4006	JP 1C90H	.....	RST 18H 向量
4009	JP 25D9H	.....	RST 20H 向量
400C	RET	JP 4BA2H	RST 28H DOS 请求处理
400F	RET	JP 44B4H	装入 DEBUG (LDA,XX/RST 28H)
4012	DI/RET	CALL 4518H	RST 38H 中断服务调用
4015	.....		键盘 DCB (8 个字节)
401D	.....		显示器 DCB (8 个字节)
4025	.....		打印机 DCB (8 个字节)
402D	JP 5000H	JP 4400H	产生 SYS1(10) DOS 请求
4030	RST 0	LD A, A3H	SYS1 的 DOS 请求代码
4032	LD A, 0	RST 28H	写 'DOS READY' 信息
4033	RET	JP 44BBH	提醒 DOS 调用设备驱动程序
4036	.....		由 SYS0 和键盘驱动程序使用的键盘工作区
403D	.....		显示控制字(普通/大写字体)
403E	.....		DOS 使用
403F	.....		DOS 使用
4040	.....		25 毫秒系统时钟
4041	.....		秒

续表

地址 (Hex)	Level II 内容	DOS 内容	说 明
4042			分
4043			小时
4044			年
4045			日
4046			月
4047			装入系统实用程序的 2 字节地址，由 SYS0/SYS 初始化为 5200H
4049			内存大小。由 SYS0/SYS 计算
404A			保留
404B			当前中断状态字
404C			中断子程序屏蔽
404D			保留(中断 位 0)
404F			保留(中断 位 1)
4051			中断子程序通讯地址
4053			保留(中断 位 3)
4055			保留(中断 位 4)
4057			保留(中断 位 5)
4059		45F7H	磁盘中断子程序的地址
405B		4560H	实时钟中断子程序的地址
405D			IPL 处理期间的堆栈
407D			在 ROM 的 IPL 期间的堆栈起点
407E			保留
407F			保留
4080			除法程序用的减法子程序。在非磁盘 IPL 期间或磁盘系统的 BASIC 实用程序使用时，将除法程序从 18F7H—1904H 移出来。
408E			含有用户子程序 (USR 调用) 的地址
4090			随机数的种子数
4093			IN A, 00
4096			OUT 00, A
4099			保持断点以后打入的最后一个字符
409A			标志(表示进入 RESUME)
409B			在当前打印行内的字符数
409C			输出设备代码 (1——打印机，0——显示器，－1——磁带机)
409D			显示器一行能显示的字符数。
409E			打印机一行能打印的字符数。
409F			保留。
40A0			字符串区分界处的首地址。
40A2			当前行号。
40A4			PST 的地址。
40A6			光标位置。
40A7			键盘缓冲区的地址。
40A9			如是磁带机输入则为 0，否则不为 0。
40AA			随机数的种子数。
40AB			从刷新寄存器送来的数。
40AC			最后一个随机数 (2 字节)。

续表

地址 (Hex)	Level II 内容	DOS 内容	说 明
40AE	.....		标志：0——寻找一个有名字的变量； -1——为有名字的变量建立一项。
40AF	.....		WRA1 中的值的数据类型标志。 2——整数 3——字符串 4——单精度 8——双精度
40B0	.....		在表达式计算过程中，保存中间结果值。
40B1	.....		存储器容量。
40B2	.....		保留。
40B3	.....		在 LSPT 中下一个可用单元的地址。
40B5	.....		LSPT（字符串库的表格）
40D2	.....		LSPT 的结束。
40D3	.....		当一个字符串被传送到字符串区时，下面 3 个字节用于保存串的长度和地址。
40D6	.....		指向串区下一个可用单元的指针。
40D8	.....		1：当前语句中执行的最后一个字节的指针 2：在 PRINT USING 语句中的编辑标志。
40DA	.....		读过的最后一个 DATA 语句的行号。
40DC	.....		FOR 标志 1=FOR 在处理中 0=没有 FOR 在处理中
40DD	.....		在输入阶段为 0，否则为 1。
40DE	.....		读标志： 0=读语句有效； 1=输入语句有效； 也用于 PRINT USING 语句中，保存字符串和变量之间的分隔符。
40DF	.....		为由 DOS 请求装入的程序，保存执行地址。
40E1	.....		AUTO 命令的标志： 0=非 AUTO 模式；非 0，保持下一个行号。
40E2	.....		用二进制表示的当前行号（在输入阶段）。
40E4	.....		AUTO 命令的行增量。
40E6	.....		在输入期间：当前语句的代码串的地址在执行期间：当前语句的行号。
40E8	.....		在执行期内：当语句执行开始时，保存堆栈指针值。
40EA	.....		发生错误行的行号。
40EC	.....		发生错误行的行号。
40ED	.....		当前语句中执行的最后一个字节。
40EF	.....		错误行中位置的地址。
40F0	.....		ON ERROR（语句的目标）地址。

续表

地址 (Hex)	Level II 内容	DOS 内容	说 明
40F2			标志，在 ON ERROR 处理期间为 FFH，用 RESUME 子程序清除。
40F3			PBUF（打印缓冲区）中小数点的地址。
40F5			执行的最后一个行号，由 STOP/END 存入。
40F7			在 ERROR 期间，执行的最后一个字节的地址。
40F9			简单变量的地址。
40FB			有维变量的地址。
40FD			可用区表（FSL）的起始地址。
40FF			在 READ 语句执行过程中，指向读入的最后一个字符后面的一个字节。
4101			变量类型说明表有 26 项(英文字母表的每一个字母占一项)，每项包括一个代码，此代码表示由该字母开始的变量的缺项数据类型。
411A			变量类型表结束。
411B			跟踪标志： 0=不跟踪； 非 0= 跟踪。
411C			当分解一个浮点数时，由数学子程序作暂存用。通常它保存从 LSB 位置移出的最后一个字节。
411D			WRA1——双精度值 LSB。
411E			WRA1——双精度值。
411F			WRA1——双精度值。
4120			WRA1——双精度值。
4121			WRA1——整数或单精度的 LSB。
4122			WRA1。
4123			WRA1——单精度的 MSB。
4124			WRA1——单精度指数。
4125			在数学和算术运算过程中，结果的符号。
4126			在双精度相加过程中用的位存储器。
4127			WRA2——LSB。
4128			WRA2
4129			WRA2
412A			WRA2
412B			WRA2
412C			WRA2
412D			WRA2——MSB。
412E			WRA2——指数。
412F			不用。
4130			在 PRINT 处理期间用的内部打印缓冲区的起点。
4149			打印缓冲区的最后字节。
414A			双精度除法子程序用作暂存，保存除数。
4151			暂存区结束。

★
★
★单元 4152H—41E2H 包含磁盘出口和磁盘 BASIC 出口。在没有磁盘的系统中，这些
★单元初始化为返回指令（RET），而在磁盘系统中这些单元将被初始化为如下情况。
★
★
★
★

地址 (Hex)	Level II 内容	DOS 内容	说 明
4152	RET	JP 5E46H	磁盘 BASIC 出口 (CVI)
4155	RET	JP 558EH	磁盘 BASIC 出口 (FN)
4158	RET	JP 5E49H	磁盘 BASIC 出口 (CVS)
415B	RET	JP 5655H	磁盘 BASIC 出口 (DEF)
415E	RET	JP 5E4CH	磁盘 BASIC 出口 (CVD)
4161	RET	JP 61EBH	磁盘 BASIC 出口 (EOF)
4164	RET	JP 6231H	磁盘 BASIC 出口 (LOC)
4167	RET	JP 6242H	磁盘 BASIC 出口 (LOF)
416A	RET	JP 5E20H	磁盘 BASIC 出口 (MKI$)
416D	RET	JP 5E30H	磁盘 BASIC 出口 (MKS$)
4170	RET	JP 5E33H	磁盘 BASIC 出口 (MKD$)
4173	RET	JP 56C4H	磁盘 BASIC 出口 (CMD)
4176	RET	JP 5714H	磁盘 BASIC 出口 (TIME$)
4179	RET	JP 6349H	磁盘 BASIC 出口 (OPEN)
417C	RET	JP 60ABH	磁盘 BASIC 出口 (FIELD)
417F	RET	JP 627CH	磁盘 BASIC 出口 (GET)
4182	RET	JP 627BH	磁盘 BASIC 出口 (PUT)
4185	RET	JP 606FH	磁盘 BASIC 出口 (CLOSE)
4188	RET	JP 5F7BH	磁盘 BASIC 出口 (LOAD)
418B	RET	JP 600BH	磁盘 BASIC 出口 (MERGE)
418E	RET	JP 6346H	磁盘 BASIC 出口 (NAME)
4191	RET	JP 63C0H	磁盘 BASIC 出口 (KILL)
4194	RET	JP 58B7H	磁盘 BASIC 出口 (&)
4197	RET	JP 60E6H	磁盘 BASIC 出口 (LSET)
419A	RET	JP 60E5H	磁盘 BASIC 出口 (RSET)
419D	RET	JP 582FH	磁盘 BASIC 出口 (INSTR)
41A0	RET	JP 6044H	磁盘 BASIC 出口 (SAVE)
41A3	RET	JP 5756H	磁盘 BASIC 出口 (LINE)
41A6	RET	JP 5679H	磁盘 BASIC 出口 (USR)

★
★
★下面这些地址是 DOS 出口地址。
★
★

地址 (Hex)	Level II 内容	DOS 内容	说 明
41A9	RET	JP *xxxx*H	来自 *xxxx* 地址的 DOS 出口
41AC	RET	JP 5FFCH	来自 1A1CH 的 DOS 出口

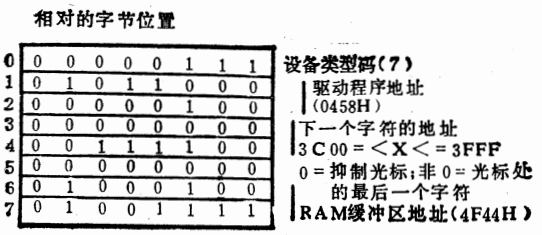
续表

地址 (Hex)	Level II 内容	DOS 内容	说 明
41AF	RET	JP 598EH	来自 0368H 的 DOS 出口
41B2	RET	JP 6033H	来自 ROM 地址 1AA1H 的 DOS 出口
41B5	RET	JP 5BD7H	来自 ROM 地址 1AECH 的 DOS 出口
41B8	RET	JP 5B8CH	来自 ROM 地址 1AF2H 的 DOS 出口
41BB	RET	JP 60A1H	来自 ROM 地址 1B8CH 的 DOS 出口
41BE	RET	JP 577CH	来自 ROM 地址 2174H 的 DOS 出口
41C1	RET	JP *xxxx*H	来自 ROM 地址 032CH 的 DOS 出口
41C4	RET	JP 59CDH	来自 ROM 地址 0358H 的 DOS 出口
41C7	RET	JP 5F78H	来自 ROM 地址 1EA6H 的 DOS 出口
41CA	RET	JP 5A15H	来自 ROM 地址 206FH 的 DOS 出口
41CD	RET	JP 5B9AH	来自 ROM 地址 20C6H 的 DOS 出口
41D0	RET	JP 5B99H	来自 ROM 地址 2103H 的 DOS 出口
41D3	RET	JP 5B65H	来自 ROM 地址 2108H (以及 2141H) 的 DOS 出口
41D6	RET	JP 5784H	来自 ROM 地址 219EH 的 DOS 出口
41DC	RET	JP 5E63H	来自 ROM 地址 222DH 的 DOS 出口
41DF	RET	JP 579CH	来自 ROM 地址 2278H 的 DOS 出口
41E2	RET	JP 5B51H	来自 ROM 地址 02B2H 的 DOS 出口

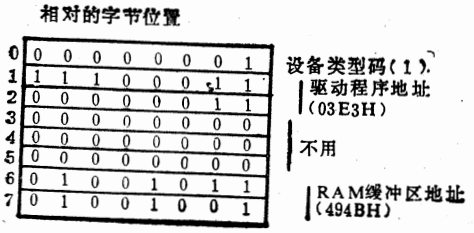
DCB 说明

键盘,荧光屏显示器和打印机的 DCB (设备控制块)是定义在 ROM 中的 06E7H—06FFH 单元。在 IPL 期间,它们被移到通讯区内所给出的地址。

显示器的 DCB (地址 401DH)



键盘 DCB (地址 4015H)



打印机 DCB（地址 4025H）

相对的字节位置

0	0	0	0	0	0	1	1	0	设备类型码（6）
1	1	0	0	0	1	1	0	1	驱动程序地址（058DH）
2	0	0	0	0	0	1	0	1	
3	0	1	0	0	0	0	1	1	行/页（43H=67）
4	0	0	0	0	0	0	0	0	到目前为止打印的行
5	0	0	0	0	0	0	0	0	不用
6	0	1	0	1	0	0	0	0	RAM缓冲区地址（5250H）
7	0	1	0	1	0	0	1	0	

中断向量

中断是一种允许外部事件去中断 CPU 工作的手段，并使 CPU 改变程序运行的方向去执行特定的一部分程序。引起这种偶然发生事件的信号称为中断，作为对中断的响应而被执行的那些程序称为服务程序。服务程序执行完以后，CPU 控制就返回到中断发生的地方，以继续正常的处理。

为了引起中断，系统必须预先准备好接收它们。当系统已经作好准备时，它就处于 ENABLED 状态。ENABLED 是用作允许中断系统的指令的简写形式（EI——启动中断）。不允许中断的系统就处于 DISABLED 状态，它是禁止指令（DI——禁止中断）的简写形式。除了系统要为中断预先作好准备以外，还必须有一些外部事件来激励中断。在 Level II 系统中，外部事件可以是时钟或者是磁盘。实际上，它们两个都能产生中断——时钟每隔 25 毫秒就给出一个中断信号，而磁盘则要根据某些操作的要求发出中断信号。

在运行一个不带磁盘的 Level II 系统时，中断是被禁止的。只有在装入 DOS 时，才许可中断，并装入支持那些中断的服务程序。IPL 指令序列开始时禁止中断，IPL 指令序列对于 Level II 和 DOS 是共用的。对于 Level II，中断将继续关闭，但是在 DOS 系统中，将在 SYS0/SYS 初始化结束时开启中断。

当一个中断发生时，会遇到两种情况。首先，在 37E0H 单元的字节中，一个表示中断确切原因的位被置位。第二，要执行 RST 38H 指令。作为 RST（它象 CALL）的结果，下一条要执行的指令地址被存入（PUSH）堆栈，并且控制被传送到 0038H 单元。存在 0038H 单元内的是一条 JP 4012H 指令。在 IPL 指令序列处理期间，4012H 被初始化为

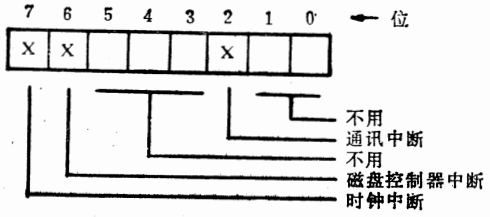
```
4012  DI     禁止以后的中断
4013  RET    返回到中断点
```

这是用于不带磁盘的系统。或

```
4012  CALL   4518H   服务中断
```

这是用于磁盘系统。

在 4518H 单元的服务程序要检查 37E0H 的内容，并执行每一个置位的位（这表示 DOS 有相应的中断服务子程序）所对应的子程序。37E0H 单元内的中断状态字的格式为



内存映象的 I/O

DOS 在 404CH 单元保持一个中断服务屏蔽字，它用作判定每一个中断状态是否有一个要执行的子程序。如取出 404 CH 单元内容为 C0H，它就表示有时钟和磁盘中断子程序。

4518H 单元的服务程序把状态字节和屏蔽字节用逻辑与的方法合并在一起。将得到的结果用作位索引，去检索存放在 404DH—405CH 单元的子程序地址表。子程序地址表的每一项是一个两字节的中断子程序地址。索引的位 0 对应于 404DH/404EH 地址，位 1 对应于 404FH/4050H 地址……等等。

服务程序进入运行，它就从左到右的扫描中断状态，无论何时只要找到一个置 1 的位，就转移到一个对应的子程序。在进入子程序以前，所有的寄存器内容均要保存起来，同时服务程序中的返回地址要推入堆栈，只有这样才能用 RET 指令从该子程序退出。当所有的状态位都已经测试过，控制返回到中断发生的中断点。

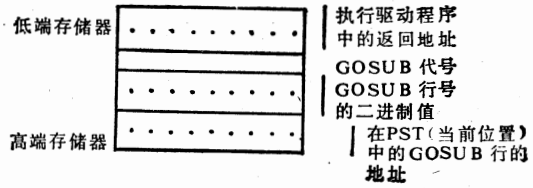
堆栈结构

Level II 通常用通讯区作暂存。然而存在着一些特殊的情况，使得不可能用通讯区作暂存，因为一个程序是可以调用其自身的(称为递归)，并且每个调用将破坏由前一个调用存入的值。这时就要用堆栈存放一些变量。当然也可以使用索引表，但是在这种情况下堆栈比较适用。

FOR 语句的堆栈

与 FOR 循环有关的全部变量地址都放在堆栈中，一直保存到循环完成为止。当处理 NEXT 语句时，它搜索堆栈寻找与当前一个有相同索引地址的 FOR 结构。搜索堆栈的程序从 1936H 单元开始。其仅有的参数是通过 DE 寄存器传送的当前索引的地址。堆栈的搜索是从它当前的位置向后搜索到堆栈的起点。如果没有找到与索引地址一致的 FOR 结构，就产生一个 NF 错误。图 4.8 给出了被搜索的堆栈结构。

GOSUB 语句的堆栈结构



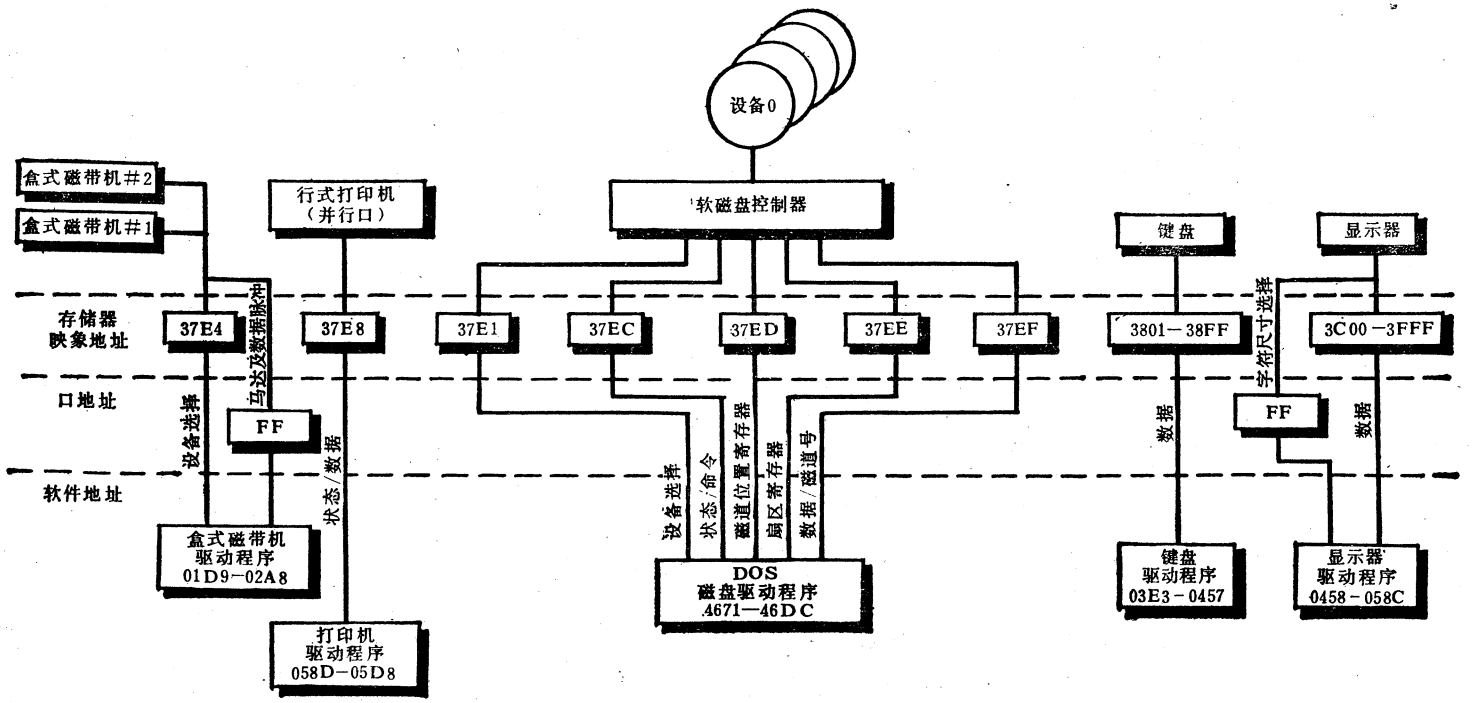


图 4.7　系统外部设备情况

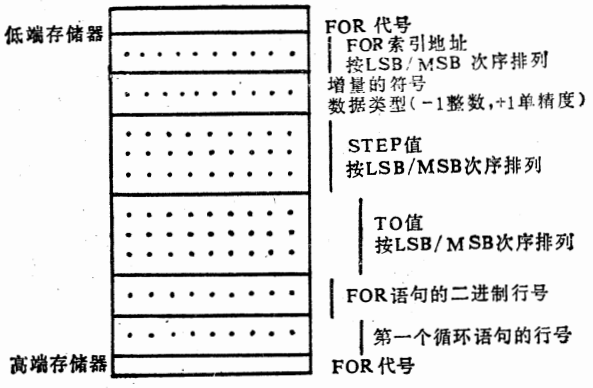


图 4.8 FOR 语句堆栈结构

表达式的计算

表达式的计算包括扫描表达式以及把表达式截断成一些独立的运算，这些运算能按算符级别的相应次序执行。这就是说必须对一个语句扫描，而且具有最高级别值(称为优先级值)的必须首先执行。运算产生的任何新项必须再代入表达式，并且与表达式的剩余部分合在一起继续往下计算。

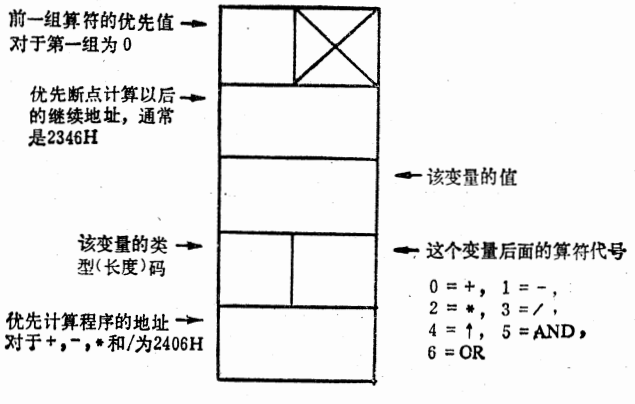
这种计算方法是一种算符优先级分析法。表达式扫描是按从左到右的次序进行的。在找到一个算符代号或 EOS 时，就立即停止扫描。算符左边的变量(称为当前变量)和算符(任何算术记号如+，-，*，/或↑)一起称为一"组"，它们或者

a) 作为一组推入堆栈，或者

b) 如果检测到一个优先断点，则执行已推入堆栈的前组和当前变量之间的运算。该运算的结果就成为当前变量，并且前组从堆栈内消去。这次计算以后，接着就要把新的当前变量和算符作为一个组推入堆栈。

重复这一步骤，直到推入新的一个组，或者，堆栈中再也没有更多的组来与当前值合并时为止。假如那样，说明表达式已经算完了。

推入堆栈的变量/算符组具有如下的格式



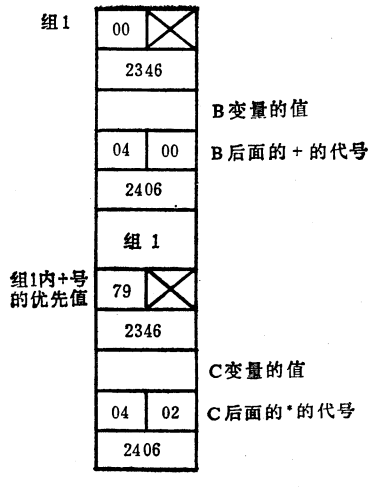
优先断点的测试是很简单的。如果算符(扫描停止处的代号)与推入堆栈的最后一组的优先值相比，有相同或较低的优先值，那末就产生了一个断点，并需要作一次中间结果的计算。随着从堆栈弹出最后一组值，该计算就自动完成了。当发生这种情况时，控制传递到一个程序（通常在 2406H 单元），该程序将执行堆栈弹出的一组中算符所规定的运算，运算将在堆栈弹出的值与当前变量之间进行。其结果就成为当前变量。当计算结束时，控制返回到重复测试优先断点的程序。如这一次是最后的一组(后面没有可引起断点的组了)，于是测试将在与该变量前面相同的算符和前一组中的算符之间进行。如果没有前一组(如表达式的第一组)，则当前的变量和算符就要作为堆栈的下一组被推入堆栈。注意，EOS 或非算术代号都作为优先断点处理。

假定没有发生断点，则当前变量和算符作为堆栈的下一组被推入堆栈，表达式扫描从它停止的这一点继续下去。让我们举一个例子来说明。假定我们有表达式

$$A = B + C * D/E \uparrow 5$$

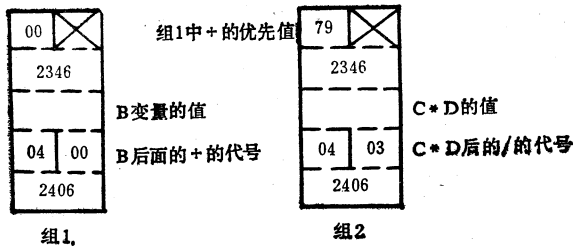
扫描从等号右边的第一个字符开始，并将停在第一个代号＋处。B ＋ 将作为第一组推入堆栈，这是因为：a) 先前没有过变量/算符组，因此就不可能已有一个优先断点。b) 扫描停在算术代号(＋)处。

下次扫描应该停在＊处。再次把变量/算符组 C＊ 推入堆栈，这次就作为组 2，其原因与前稍有不同。＊(乘)的优先值比已经推入的＋(加)号优先值要高，所以没有截断。这时堆栈内容为

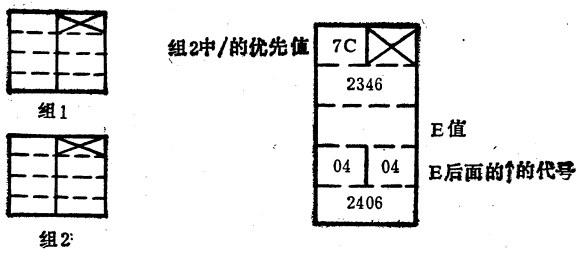


第三次扫描将停在紧接着D的/(除)号处。这一次应当有一个优先断点，因为＊和/有相同的优先值。接着组 2 就从堆栈内弹出，控制传到在 2406H 单元的优先断点程序(也可以用其它的程序，这取决于要执行的操作——要了解详细情况，请查阅清单)。在这里将执行组 2，即 (C＊) 和当前值 (D) 之间的运算。这个运算将产生一个新的当前值，我们称它为 M，M等于 C＊D。

相乘以后，控制回到 2346H 单元(在截断处理以后继续下去)，在此仍使用上面用到的规则。这时将当前值作为组 2 推入堆栈，因为它与组 1 (＋)相比有较高的优先值(/)。现在堆栈的内容为



推入组 2 以后扫描继续进行下去，停在(↑)算符处。该算符(↑)有一个比组 2 中(/)高的优先值，于是第 3 组被加到堆栈内，得出



作下一次扫描，找到一个跟在 5 (这是现在的当前值) 后面的 EOS。根据前面的讨论，EOS 和非算术代号是一个自动的优先断点，因此，组 3 就从堆栈弹出，并计算 E↑5，它的结果就成为一个当前值。控制传到用于推下一个组到堆栈的 2346H 单元，并且，因为当前算符是一个 EOS，所以组 2 就要从堆栈弹出。进行组 2 (M/) 和当前值之间的运算，得到如下的当前值

$$M/E\uparrow 5 \quad 或 \quad C*D/E\uparrow 5$$

因为当前算符是 EOS，所以控制再次传到 2346H 单元，强迫组 1 弹出。当组 1 弹出时，控制就传到计算程序，以便对当前值和组 1 进行运算。得到如下的当前值

$$B+C*D/E\uparrow 5$$

现在控制又传送到 2346H 单元，这时堆栈是空的，使控制返回到调用它的程序。这个表达式已计算完毕，它的值放在 WRA1 中。

DOS 功能代码

DOS 功能码为在程序中间执行系统级命令提供了一种手段。其办法是把与功能码有关的 DOS 复盖模块 SYSX/SYS 装入 4200H—5200H 单元并执行此模块。当完成某一功能后，控制就返回给调用它的程序，就像作了一个子程序调用一样。

把 DOS 功能码装入一个寄存器和执行 RST 28H 指令，就可以执行 DOS 功能。因为用了这种方法，DOS 处理这些功能码时要用到堆栈，把 RST 指令推入堆栈的返回地址冲掉了，于是控制将返回到在堆栈中找到的下一个地址而不是返回到接在 RST 指令后面的地址。例如：

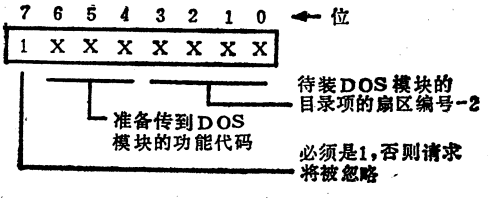
```
LD      A, VAL      ;装入 DOS 功能码。
RST     28H         ;执行 DOS 功能。
```

```
        ⋮                   ;这是我们想返回的地方，实际上不能返回。这是
        ⋮                    由于堆栈是由 DOS 管理着的。
```

这是不能正常工作的，因为返回地址(由 RST 28H 存入堆栈)在处理过程中已被冲掉了。应该用下面的指令序列来代替：

```
        LD      A, VAL      ;装入功能码。
        CALL    DOS         ;把返回地址推入堆栈。
        ⋮
        ⋮
DOS     RST     28H         ;执行 DOS 功能。
                            保留所有的寄存器。将自动返回到 DOS 的调用
                            者。
```

装在A寄存器内的功能码值必须包括待装入的复盖模块目录扇区的扇区号减 2 的值和规定待执行的确切操作的一个代码。功能码格式是



随着对它的执行，由指定目录扇区中的第一项所指定的文件将被装入。例如要装入与第三项和第四项有关的文件是办不到的。系统复盖模块的清单和它的功能如下，但这些说明是不完整的。完整的说明请参看各个模块的说明。

模 块	目录扇区-2	功能码	子功能
SYS1	1	93	10——写 'DOS READY'
		A3	20——写 'DOS READY'
		B3	30——扫描输入字符串。
		C3	40——把输入字符串传送到 DCB。
		D3	50——扫描和传送输入字符串，
		E3	60——添加 DCB 的扩充部分。
		F3	70——保留。
SYS2	2	94	10——OPEN 文件处理。
		A4	20——INIT 文件处理。
		B4	30——建立目录的扩充项。
		C4	40——保留。
		D4	50——保留。
		E4	60——保留。
		F4	70——保留。
SYS3	3	95	10——CLOSE 文件处理。
		A5	20——KILL 文件处理。
		B5	30——保留。
		C5	40——保留。

续表

模　块	目录扇区-2	功能码	子功能
		D5	50——保留。
		E5	60——装入 SYS3/SYS。
		F5	70——格式化磁盘片。
SYS4			
SYS5			

第五章　例 1

BASIC 的 SORT 工作子程序

这一章的内容是一个汇编程序实例，它演示了 ROM 调用及前几章中介绍的那些表格的使用方法。在这个例子中使用了 DOS 出口和磁盘 BASIC 出口，以便添加一个 SORT 工作子程序到 BASIC 中。

如果添加了一个 SORT 工作子程序，于是语句

100 SORT I$, O$, K1$

就可用来读出和分类一个由字符串 I$ 确定的文件。O$ 和 K1$ 是规定输出文件名和分类关键描述符的字符串。实现方法是很简单的。首先，为了识别出 SORT，必须修改输入程序，并且用一个代号来代替 SORT。这些工作可以利用一个 DOS 出口来完成。

在输入程序工作期间，对保留字扫描以后立即可获得 DOS 出口。我们将截取这个出口，对 SORT 一词作进一步的测试，并且用一个代号代替它。然后，处理将像以前那样继续下去。在使用任何 DOS 出口以前，为了确定寄存器正确的用法，应该研究一下出口周围的一些程序。在这种情况下，要特别注意：当获得出口时，进入行的长度是在 BC 寄存器中。如果子程序缩短了这一行(因为用一个代号代替了词 SORT)，那么，它的长度改变了，必须以新的长度来代替 BC 原来的内容。

要作的第二个修改是必须对执行驱动程序 (Execution Driver) 或该程序序列中的某些地方进行修改，使得能够识别出这个新的代号值和转移到 SORT 操作程序。这时会出现一些小问题，因为在调用工作子程序以前，在执行驱动程序中并不存在 DOS 出口，而且驱动程序代码及其表格是在 ROM 中的，它们是不能够改变的。总之，没有一种很容易的方法把新的代号合并到执行程序中。

解决的办法是借用一个磁盘 BASIC 代号，并且在它后面再附加另一个代号。这样，任何一个与借用代号有关的工作子程序的调用都必须截取下来，并对附加的代号进行测试。结果找到了一个附加代号，控制就转向 SORT 工作子程序，否则控制就传递到原定的工作子程序。在所举的这个例子中，将借用代号 FAH，并且在它的后面添加另一个代号 FAH，这就得到一个新代号 FAFAH。

这个例子并不完善，因为 LIST 功能还没有修改成能识别 SORT 代号。如果发出一个 LIST 命令，那末对于 SORT 工作子程序将会给出 MID$ MID$。在讨论工作子程序以前，还有一个细节需要注意。从第一章内的存储器分配图我们可以看到没有指明什么地方可以设法装入汇编程序，而又不干扰 BASIC 区域。工作子程序可能会覆盖字符串区，或者堆栈，甚至于达到更低的 PST 或 VLT 区域，这就要看我们把工作子程序装在哪里。当然，我们也许有运气，能在空闲空间表 (FST) 中找到一个从不使用的区域，但是这样做有点太冒险了。

BASIC 具有设定它要使用的存储空间上限的能力。使用这种特性，我们能够在存储空间的高处保留一个区，以便装入工作子程序而不妨碍任何 BASIC 的表格。现在详细讲述工作子程序。

因为分类可能是一段较长的程序，所以只介绍关于 DOS 出口，磁盘 BASIC 以及第二章中的某些 ROM 调用的详细情况。工作子程序有二段。第一段将被调用一次，这是为了修改通讯区中的 DOS 和磁盘 BASIC 的出口地址(也称为向量)，使它指向工作子程序中的某些单元。在 DOS 系统中，进入 BASIC 以后，必须修改向量地址，因为这些向量地址是用 BASIC 命令初始化的。第二段有二部分。第一部分是输入扫描程序中调用的 DOS 出口代码。第二部分是 SORT 的工作子程序。在执行阶段中遇到一个 FAH 代号时，就进入工作子程序。

假定现在所用的系统有 48K RAM，至少有一个磁盘，并装有 NEWDOS 2.1。工作子程序将占有 E000H—FFFFH 单元。进行向量初始化的入口点将在 E000H。所用的全部缓冲区将动态地分配在空闲空间表 (FST) 的堆栈部分。在退出 DOS 和进入 Level II BASIC 之前装入工作子程序，但是也能用 NEWDOS 的 CMD′ LOAD···′ 从 BASIC 程序装入工作子程序。

1. IPL
2. LOAD, SORT: (把工作子程序装入 E000H—FFFFH)
3. BASIC, 57344: (保护工作子程序区)

```
100  DEF  USR1 (0) = &HE000: 初始化入口点。
110  A = USR1 (0)            : 初始化向量。
RUN                          : 初始化 SORT。

100  I$ = "SORTIN/PAY:1"     : (分类输入文件)
110  O$ = "SORTOUT/PAY:1"    : (分类输出文件)
120  K$ = "A, A, 100—120"    : (分类关键词: 按升序排列的 ASCII 码，分类字段是
                               100—120)。
130  SORT I$, O$, K$         : (分类文件)
RUN
```

```
00100    ORG  0E000H
00110;   初始化 DOS 出口和磁盘 BASIC 地址的初
         始入口点。
00120;
00130    LD  HL, (41B3H)  ;我们处理以后，仍然要
                          用原来的 DOS 出口
                          值。
00140    LD  (ADR1+1), HL
00150;
00160    LD  HL, (41DAH)  ;MID$ 代号 (FAH)
                          的原来磁盘 BASIC 的
                          地址。
00170;
00180    LD  (ADR2+1), HL  ;找到 FAH 代号时，
                           保存起来。
00190    LD  HL, NDX
00200    LD  (41B3H), HL
00210    LD  HL, NDB
00220; 我们的地址。
00230    LD  (41DAH), HL
00240; 具有我们的地址的 FAH 代号。
00250    RET               ;返回到执行驱动程
                           序。
00260;★  得到变量地址
```

```
00270;★ 在输入阶段这一段程序是作为一个 DOS 出
00280;★ 口送入的。其工作为测试'SORT' 命令,并以
00290;★ 'FAFA' 代号来代替 SORT。原来的 DOS
00300;★ 出口地址已经保存起来,并将由 ADR1 处取
00310;★ 出。
00320 NDX CALL SAV        ;保存所有的寄存器。
00330    LD  IX, SORT-1   ;测试字符串。
00340    LD  B, 3         ;匹配的字符数。
00350 NDX1  INC HL        ;循环的起点。
00360    INC IX           ;指向下一个待测字
                           符。
00370    LD  A, (IX+0);   ;取下一个测试字符。
00380    CP  (HL)         ;与输入字符串比较。
00390    JR  NZ, OUT      ;当遇到第一个不匹配
                           字符时,停止。
00400    DJNZ NDX1        ;四个字符必须全部匹
                           配。
00410;★
00420;★ 如果已经匹配,现在用一个代号 'FAFA' 来
00430;★ 代替 'SORT',并压缩字符串。
00440;★
00450    INC HL           ;'SORT' 后面的第一
                           个字符。
00460    PUSH HL          ;为了压缩代码,要保
                           存起来。
00470    LD BC, −3        ;要在输入字符串中退
                           回 3 个字符的位置。
00480    ADD HL, BC       ;'SORT' 的开始。
00490    LD (HL), 0FAH    ;用代号代替 'S'。
00500    INC HL           ;输入字符串中的下一
                           个位置。
00510    LD (HL), 0FAH    ;用代号代替 'O'。
00520    INC HL           ;输入字符串中的下一
                           个位置。
00530    POP DE           ;SORT 后面的字符串
                           地址。
00540    EX DE, HL        ;于是我们能用 RST
00550;★                    10H 去取下一个字
                           符。
00560;★ 现在压缩输入字符串
00570;★
00580    LD BC, 3         ;设定字符的计数,以
00590;★                    便等于跳过的字符
                           数。
00600 NDX2 RST 10H        ;取出下一个字符,去
                           掉空位。
00610;★
00620    LD (DE), A       ;将它往下移。
00630    INC DE           ;指向源地址。
00640    INC C            ;行内的字符计数加1。
00650    OR A             ;测试字符串的终点。
00660    JR NZ, NDX2      ;没有结束,循环。
00670    LD (DE), A       ;每一行必须用 3 字节
00680;★                    的零来结束。
00690    INC DE           ;指向最后一个字节。
00700    LD (DE), A       ;存第三个零。
00710    INC C            ;然后设定 BC= 行长
                           度加 1。
00720    INC C
00730    INC C            ;于是,BASIC 能够
                           移动它。
00740    LD (TEMP), BC    ;保存新的行的长度。
00750    CALL RES         ;恢复寄存器。
00760    LD BC, (TEMP)    ;新的行长度放入 BC。
00770    JR ADR1          ;出口。
00780 OUT CALL RES        ;恢复寄存器。
00790 ADR1 JP 0           ;在原来的 DOS 出口
                           处继续。
00800;★
00810;★ 对于 FAH 代号,则磁盘 BASIC 出口。测试
00820;★ SORT 代号 FAFAH。
00830 NDB CALL SAV        ;保存所有的寄存器。
00840    INC HL           ;跳到代号后面的字
                           符。
00850    LD A, (HL)       ;测试第二个 'FA'。
00860    CP 0FAH          ;下一个字符是 FAH
                           吗?
00870    JR Z, NDB1       ;如是 SORT 的代号,
                           则为 Z。
00880    CALL RES         ;恢复寄存器。
00890 ADR2 JP 0           ;继续 MID$ 处理。
00900;★
00910;★ 我们有一个 SORT 代号。
00920;★
00930 NDB1 INC HL         ;跳过代号的剩余部
                           分。
00940    CALL GADR        ;得到第一个参数的地
                           址。
00950    LD (PARM1), DE   ;保存输入文件名的地
                           址。
00960    RST 08           ;寻找逗号。
00970    DEFM ','         ;要找的符号。
00980    CALL GADR        ;得到第二个参数的地
                           址。
00990    LD (PARM2), DE   ;保存输出文件名的地
                           址。
01000    RST 08           ;寻找逗号。
01010    DEFM ','         ;要找的符号。
01020    CALL GADR        ;得到 SORT 关键词
                           的地址。
01030    LD (PARM3), DE   ;保存 SORT 关键词
                           的地址。
```

```
01040    LD (TEMP), HL      ;保存当前语句中的结
                             束位置。
01050;★
01060;★ 现在用空格填满 I/O DCB
01070;★
01080;★
01090    LD IX, DCBL        ;DCB 的地址清单。
01100    LD C, 2            ;要填空格的 DCB 数。
01110    LD A, 20H          ;空格的 ASCII 代码。
01120  L1 LD L, (IX+0)      ;DCB 地址的 LSB。
01130    LD H, (IX+1)       ;DCB 地址的 MSB。
01140    LD B, 32           ;要填空格的字节数。
01150  L2 LD (HL), A        ;填空格的循环。
01160    INC HL
01170    DJNZ L2            ;循环，直到空格填满
                             为止。
01180    INC IX             ;指向下一个 DCB 地
                             址。
01190    INC IX             ;再增加 DCB 地址。
01200    DEC C              ;所有的 DCB 都填满
                             空格了吗？
01210    JR NZ, L1          ;否，转移到 L1 循环。
01220;★
01230;★ 是，把文件名传送到 DCB 区。
01240;★
01250    LD HL, (PARM1)     ;输入文件名字符串的
                             地址。
01260    LD DE, DCBI        ;输入的 DCB。
01270    CALL 29C8H         ;把文件名移到 DCB。
01280    LD HL, (PARM2)     ;输出文件名字符串的
                             地址。
01290    LD DE, DCBO        ;输出的 DCB。
01300    CALL 29C8H         ;把文件名移到 DCB。
01310    LD HL, (PARM3)     ;得到关键词字符串的
                             地址。
01320    INC HL             ;跳过字节计数。
01330    LD C, (HL)         ;得到字符串地址的
                             LSB。
01340    INC HL             ;指向地址的剩余部
                             分。
01350    LD H, (HL)         ;得到字符串地址的
                             MSB。
01360    LD L, C            ;现在，HL=串地址。
01370    CALL 1E3DH         ;必须是字母。
01380    JR NC, YA1         ;是。
01390    JP ERROR           ;不正确的分类次序。
01400  YA1 LD (ORDER), A    ;保存分类次序。
01410    INC HL             ;跳到结束字符。
01420    RST 08             ;测试逗号。
01430    DEFM ','
01440    CALL 1E3DH         ;必须是字母。
01450    JR NC, YA5         ;是。
01460    JP ERROR
01470  YA5 LD (TYPE), A     ;保存类型。
01480    INC HL             ;跳到结束字符。
01490    RST 8              ;测试逗号。
01500    DEFM ','
01510    CALL 0E6CH         ;得到记录大小。
01520    LD DE, (4121H)     ;从 WRA1 取记录大
                             小。
01530    LD (SIZE), DE      ;将它保存起来。
01540    RST 20H            ;必须是一个整数。
01550    JP M, YA10         ;如是整数，则 M 标
                             志置位。
01560    JP ERROR
01570  YA10 RST 08          ;寻找逗号。
01580    DEFM ','
01590    CALL 0E6CH         ;得到开始位置。
01600    LD DE,(4121H)      ;从 WRA1 取起始位
                             置。
01610    LD (START), DE     ;保存起来。
01620    RST 08             ;寻找'—'符号
01630    DEFM '—'           ;待测试的字符。
01640    CALL 0E6CH         ;得到关键词的结束位
                             置。
01650    LD DE, (4121E)     ;从 WRA1 得到值(结
                             束位置)。
01660    LD (END), DE       ;保存结束位置。
01670    LD HL, (TEMP)      ;达到 EOS 返回时，恢
                             复当前的行地址。
01680;★
01690    CALL RES           ;恢复寄存器。
01700    LD HL, (TEMP)      ;恢复 EOS 地址。
01710    RET                ;返回到 BASIC。
01720;★
01730;★
01740;★
01750  SORT DEFM 'S'        ;SORT 的 S 字母。
01760    DEFB 0D3H          ;SORT 的 OR 的代
                             号。
01770    DEFM 'T'           ;SORT 的 T 字母。
01780;★
01790;★ 保存所有的寄存器
01800;★
01810  SAV EX DE, HL
01820    EX (SP), HL        ;保存DE，返回地址放
                             入 HL。
01830    PUSH BC
01840    PUSH AF
01850    PUSH IX
01860    PUSH DE            ;保存原始的 HL。
01870    EX DE, HL          ;恢复 HL，返回地址
                             送到 DE。
```

```
01880     PUSH DE          ;返回地址送到堆栈。
01890     RET              ;返回到调用它的程
                           序。
01900;★
01910;★ 恢复所有的寄存器
01920;★
01930  RES POP HL          ;返回地址送到 HL。
01940     POP DE           ;实在的 HL。
01950     POP IX
01960     POP AF
01970     POP BC
01980     EX (SP), HL      ;返回地址送到堆栈。
01990     EX DE, HL
02000     RET
02010     JP (HL)          ;返回到调用它的程
                           序。
02020;★
02030;★ 将下一个变量地址放进 DE
02040;★
02050  GADR LD A, (HL)     ;从输入字符串得到下
02060;★                    一个字符,测试文字。
02070     CP 22H           ;是个引号吗?引号是
02080;★                    一个文字串的开始。
02090     JR NZ, GADR2     ;否,去找变量的地址。
02100     CALL 2866H       ;是,去建立一个LSPT
                           项。
02110     JR GADR5         ;然后转移到 GADR5
                           的公用程序。
02120  GADR2 CALL 2540H    ;得到下一个变量的地
                           址。
02130  GADR5 RST 20H       ;这是一个字符串吗?
02140     LD DE, (4121H)   ;下一个变量的地址。
02150     RET Z            ;如果是串变量就返
                           回。
02160     POP HL           ;清除堆栈。
02170     POP HL           ;清除堆栈。
02180     LD A, 2          ;语法错误的错误代
                           码。
02190     JP 1997H         ;转移到 ERROR 子程
                           序。
02200;★
02210;★ ERROR 出口
02220;★
02230  ERROR CALL RES      ;恢复寄存器。
02240     POP HL           ;清除堆栈。
02250     LD A, 2          ;语法错误代码。
02260     JP 1997H         ;打印错误信息。
02270;★
02280;★
02290;★
02300  DCBL DEFW DCBI
           DEFW DCBO
02320  PARM1 DEFW 0        ;输入文件名字符串地
                           址。
02330  PARM2 DEFW 0        ;输出文件名字符串地
                           址。
02340  PARM3 DEFW 0        ;关键词字符串地址。
02350  TYPE DEFB 0         ;记录类型。
02360  ORDER DEFB 0        ;分类次序。
02370  SIZE DEFW 0         ;记录大小。
02380  START DEFW 0        ;关键词的开始位置。
02390  END DEFW 0          ;关键词的结束位置。
02400  TEMP DEFW 0         ;存放 EOS 地址。
02410  DCBI DEFS 32        ;输入 DCB。
02420  DCBO DEFS 32        ;输出 DCB。
02430        END
```

第六章　例 2

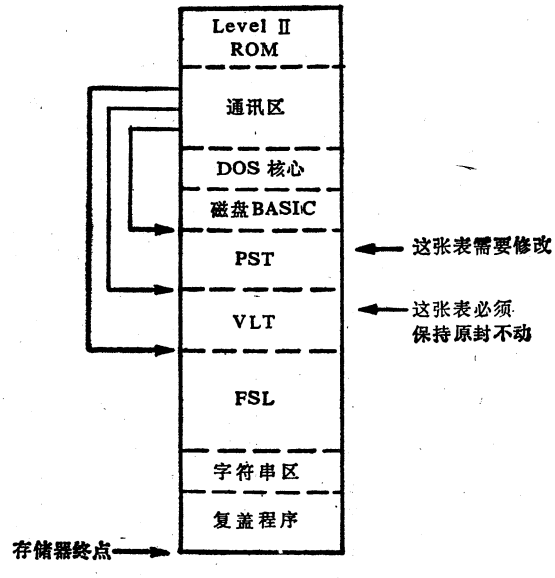
BASIC 覆盖程序

这个例子教你怎样使用通讯区内的这些表格，使得一个 BASIC 程序能够装入和执行覆盖程序。覆盖程序把一些语句加到执行中的 BASIC 程序内，而保持所有的当前变量不受影响。使用的调用指令序列是：

```
100   DEF   USR1=&HE000        :覆盖程序的地址。
            ⋮                  :应用程序的主体。
300   F$="FILE1/BAS"           :包括有覆盖程序的文件。
310   Z=USR1 (500)             :这个程序结束时，用 FILE1/BAS 中的语句来代替行500。
320   GOSUB  500               :执行覆盖程序。
            ⋮
500   REM  START  OF  OVERLAY  AREA
```

该例的操作过程中所作的假定将同第五章中的假定相同。注意：包含 ASCII 文件的那些覆盖文件必须是已经按 A 方式(即 ASCII 码的形式)存放的文件。

然而，程序本身与前例是有很大差别的。例如，这里将不使用 DOS 出口。这就是说将不需要修改通讯区，所以就不需要起始的入口点。本例将把一个参数放在调用指令序列中进行传送，另外一个将有一个约定的名字，这样才能在 VLT 中找到它。



执行 BASIC 程序时，要用到三张主要的表格。第一张表是 PST，在那里已经存有待执行的 BASIC 语句。第二张表是 VLT，在那里存放着分配给程序的那些变量，第三张表是 FSL，它给出了可用的存储单元。这些表是按上面叙述的次序放置的。为了支持覆盖程序，我们需要克服的问题是：寻找一种方法去改变第一张表，而保持第二张表的内容不变。标明这些表格的存储器分配图示于前页。

幸而上述问题十分容易解决。采用移动 VLT 到 FSL 高端的方法，我们就可以把 VLT 与 PST 分隔开。然后就能从磁盘读出覆盖语句，并加到 PST 中。显然，这样做时 PST 或者要增长或者会缩短，除非现在装入的覆盖模块的大小同以前的那一块完全一样。在加入了覆盖语句以后要移回 VLT，于是 VLT 就紧接着 PST 了。然后要修改移动后的表的指针，并使控制返回到 BASIC 执行驱动程序。

用在这个例子中的覆盖装入程序假定：包含覆盖语句的文件是 ASCII 码格式的。这就是说，每个进入行在移入 PST 以前必须先代号化。为了提高处理速度，应该修改装入程序，以便接受代号化的文件。

能装入的覆盖语句的数量没有限制。如果发现有行号小于开始行号这样的错误，程序就将退出。对于较高级别覆盖破坏一个较低级别覆盖的这种情况，装入程序不作测试，这会成为一种灾难——因为返回路径将被破坏。

作为一个例子，这里给出了装入三个独立的覆盖程序的例程序。

```
100   A=1.2345
110   B=1
120   IF B=1 THEN F$ ="FILE1"
130   IF B=2 THEN F$ ="FILE2"
140   IF B=3 THEN F$ ="FILE3"
150   Z=USR1 (500)
160   GOSUB 500
170   B=B+1
180   IF B>3 THEN 110
190   GOTO 120
```

```
500   PRINT "OVERLAY #1 ENTERED"
510   PRINT A
520   C=25
530   D=30
540   E=C+D+A
550   PRINT "C="; C
560   PRINT "D="; D
570   PRINT "E="; E
580   RETURN
```

文件1的内容

```
500   PRINT "OVERLAY #2 ENTERED"
510   PRINT A
520   C=C+1
530   D=D+1
540   E=E+1
550   REM
```

文件2的内容

```
560   REM
570   REM
580   REM
590   PRINT "C, D, E"; C, D, E
600   RETURN
```

```
500   PRINT "OVERLAY #3 ENTERED"
510   A=A+1
520   PRINT "A="; A
530   RETURN
```

文件3的内容

```
00100         ORG 0F000H
00110   OPEN EQU 4424H   ; DOS 地址。
00120   READ EQU 4436H   ; DOS 地址。
00130   ERN  EQU 12      ; 磁盘 DCB 地址。
00140   NRN  EQU 10      ; 磁盘 DCB 地址。
00150   EOF  EQU 8       ; 磁盘 DCB 地址。
00160;★
00170;★对于 BASIC 程序的覆盖装入的入口点
00180;★
00190   PUSH AF          ; 保存所有的寄存器。
00200   PUSH BC
00210   PUSH DE
00220   PUSH HL
00230   LD HL,-1         ; 预置扇区计数为-1。
00240   LD (RCOUNT), HL
00250   LD HL, 00        ; 于是我们能加载 CSP
                         (当前堆栈指针)。
```

```
00260   ADD HL, SP          ; 装 CSP。
00270   LD (CSP), HL        ; 存储以备复原。
00280   LD DE, (4121H)      ; 开始覆盖的行号。
00290   LD (LINE), DE       ; 存储以备今后参考。
00300   LD A, (40AFH)       ; 函数值类型必须在最
                              后恢复。
00310   LD (TYPE), A
 0320;★
00330;★ 在把名字移进 DCB 以前，要把 DCB 填满空格。
00340;★
00350   LD B, 32            ; 填空格的字节数。
00360   LD HL, DCB          ; DCB 的地址。
00370   LD A, 20H           ; 空格的 ASCII 码。
00380   BFL LD (HL), A      ; 传送一个空格。
00390   INC HL              ; 指向下一个字。
00400   DJNZ BFL            ; 循环，直到 DCB 填
                              满空格。
00410;★
00420;★ 从变量 F$ 得到覆盖文件名。把它传送到已经
00430;★ 清除干净的 DCB。
00440;★
00450   LD HL, LFN          ; 公共变量名的字符
                              串。
00460   CALL 2540H          ; 得到 F$ 的地址。
00470   RST 20H             ; 确认它是一个字符
                              串。
00480   JR Z, OK            ; 如是字符串，则Z标
                              志置位。
00490   JP ERR              ; 错误的变量形式。
00500   OK LD HL, (4121H) ; 把F$的地址放进HL。
00510   LD DE, DCB          ; DCB 地址。
00520   CALL 29C8H          ; 把 F$ 名传送到
                              DCB。
00530;★
00540;★ 初始化全部局部变量。
00550;★
00560   LD A, 0             ; 设定传送标志为 0。
00570   LD (PF), A          ; 传送标志。
00580   LD (PI), A          ; 扇区缓冲器索引。
00590;★
00600;★ 寻找分配给函数调用的变量地址。
00610;★ 覆盖装入完毕以后，必须重新计算这个地址，因
00620;★ 为 VLT 要移动。接着，给用来读覆盖文件的
00630;★ 扇区缓冲区在 FSL 中分配空间。
00640;★
00650;★
00660   LD HL, 00           ; 于是我们能装 CSP。
00670   ADD HL, SP          ; HL = CSP。
00680   PUSH HL             ; 保存 CSP。
00690   LD BC, 20           ; CSP 退回的总数。
00700   ADD HL, BC          ; 给出 CPS-20，即函
00710;★                       数变量的地址。
00720   LD (VARADR), HL    ; 保存变量的堆栈地
                              址。
00730   POP HL              ; 恢复 CSP，存入 HL
                              中。
00740   LD BC, —256         ; 在 FSL 内给扇区缓
00750;★                       冲区分配的空间的总
                              数。
00760   ADD HL, BC          ; 计算新的 CSP。
00770   LD (BADDR), HL      ; 扇区缓冲区的开始。
00780   LD SP, HL           ; 也是新的 CSP。
00790   PUSH HL
00800   LD DE, (40F9H)      ; PST 现在的终点。
00810   LD (CEPST), DE      ; 存储以备计算用。
00820   LD HL, (40FBH)      ; 数组的开始。
00830   XOR A               ; 清除进位标志。
00840   SBC HL, DE          ; 计算从 VLT 的起点
00850;★                       到数组起点间的偏移
                              值。
00860   LD (LSVLT), HL      ; 保存偏移值。
00870;★
00880;★
00890;★
00900   LD DE, (LINE)       ; 寻找 PST 中覆盖开
00910   CALL 1B2CH          ; 始处的行地址。
00920   LD (40F9H), BC      ; 用它作为 PST 的临
                              时终点。
00930;★
00940;★计算 VLT 的长度
00950;★
00960   LD DE, (CEPST)      ; PST 原来的终点。
00970   LD HL, (40FDH)      ; FSL 的起点。
00980   XOR A               ; 清除进位标志。
00990   SBC HL, DE          ; 给出 VLT 的长度
                              —1。
01000   INC HL              ; 修正—1使它为 VLT
                              的长度。
01010   LD (LVLT), HL       ; 保存 VLT 的长度。
01020   POP HL              ; 把 CSP 重新放到
                              HL。
01030   LD BC, —50          ; 假定需要的堆栈长
                              度。
01040   ADD HL, BC          ; 给出临时的 VLT 的
                              终点。
01050   LD BC, (LVLT)       ; 现在从终点中减去
                              VLT 的长度，得到起
                              始地址。
01060   XOR A
01070   SBC HL, BC
01080   LD (SNVLT), HL      ; 保存临时 VLT 的终
                              点。
```

```
01090   PUSH HL          ; 现在我们可以把它装
                           进 DE。
01100   POP DE
01110   LD HL, (CEPST)   ; 老的 PST 的起点。
01120   LD BC, (LVLT)    ; VLT 的大小。
01130   LDIR             ; 把 VLT 移到临时的
                           位置。
01140;
01150; 开始覆盖装入
01160;
01170   LD DE, DCB       ; 覆盖文件的 DCB。
01180   LD HL, (BADDR)   ; 扇区缓冲区地址。
01190   LD BC, 0         ; 指定扇区 I/O。
01200   CALL OPEN        ; 打开覆盖文件。
01210 LOOP CALL GNL      ; 得到文件的下一行。
01220   JR Z, OUT        ; 如果覆盖文件中不再
                           有行，则为0。
01230;★
01240   CALL ATOB        ; 把行加到 PST 中。
01250   JR LOOP          ; 循环，直到文件取完
                           为止。
01260;★
01270;★已经加了覆盖语句。
01280;★VLT 往下移(紧接着 PST) 以后，重新设置
01290;★VLT 的指针。
01300 OUT LD HL, (SNVLT); 临时 VLT 的开始地
                           址。
01310   LD DE, (40F9H)   ; PST 的当前终点。
01320   INC DE           ; 在 PST 的终点
01330   INC DE           ; 留下二个字节的 0。
01340   LD (40F9H), DE   ; 保存新的 VLT 的开
                           始地址。
01350   LD BC, (LVLT)    ; VLT 的长度。
01360   LDIR             ; 将 VLT 移到 PST
                           的终点。
01370   INC DE           ; 给出 FSL 的地址。
01380   PUSH DE          ; 保存 FSL 的地址。
01390   LD HL, (40F9H)   ; VLT 的开始地址。
01400   LD BC, (LSVLT)   ; 加简单变量的长度。
01410   ADD HL, BC       ; 给出数组指针的地
                           址。
01420   LD (40FBH), HL   ; 保存新的数组指针。
01430   POP HL           ; HL= 新的 FSL 地
                           址。
01440   LD (40FDH), HL   ; 修改 FSL。
01450;★
01460;★计算 VLT 已经移动的距离，修改正要推入堆
01470;★栈的函数变量的地址。
01480;★
01490   LD DE, (CEPST)   ; VLT 原来的起点。
01500   LD HL, (40F9H)   ; VLT 当前的起点。
01510   RST 18H          ; 比较地址。
01520   JR NC, UP        ; 向上移动新的 VLT。
01530   PUSH HL          ; 保留操作数。
01540   PUSH DE
01550   XOR A            ; 清除进位标志。
01560   POP HL           ; 恢复操作数。
01570   POP DE
01580   JR UP1           ; 计算距离。
01590 UP XOR A           ; 为了进行减，清除进
                           位标志。
01600 UP1 SBC HL, DE     ; 计算 VLT 已经移动
                           过的总数。
01610   PUSH HL          ; 保存距离。
01620   LD HL, (VARADR)  ; 取出变量的堆栈地
                           址。
01630   LD C, (HL)
01640   INC HL           ; 指向地址的 MSB。
01650   LD B, (HL)       ; BC= 放在堆栈的变
                           量地址。
01660;★
01670   POP HL           ; 得到偏移量。
01680   ADD HL, BC       ; 得到新的地址
01690;★                    (因为 VLT 已经移
                           动过)。
01700   PUSH HL          ; 于是能把它装进去。
01710   POP DE           ; 把新地址装进 DE。
01720   LD HL, (VARADD)  ; 再取堆栈地址。
01730   LD (HL), E       ; 函数变量地址的
                           LSB。
01740   INC HL           ; 堆栈中下一个字节的
                           地址。
01750   LD (HL), D       ; 函数变量地址的
                           MSB。
01760;★
01770;★重新设置数据类型为它原来的值
01780;★
01790   LD A, (TYPE)     ; 得到进入时的类型标
                           志。
01800   LD (40AFH), A    ; 类型恢复为它原来的
                           值。
01810   LD HL, (CSP)     ; 恢复 CSP 为它原来
                           的值。
01820   LD SP, HL
01830   POP HL           ; 恢复寄存器。
01840   POP DE
01850   POP BC
01860   POP AF
01870   RET              ; 返回到 BASIC。
01880;★
01890;★GNL——从一个文件中得到 BASIC 程序的下
01900;★      一行，把它传送到 BASIC 行的缓冲
01910;★      区，然后把它变成代号。
```

```
01920;★文件假定是以 ASCII 格式表示的。行由一个
01930;★回车来结束的 (0DH)。
01940;★
01950 GNL LD A, (PF)      ; 得到传送标志。
01960   OR A               ; 读扇区吗？
01970   JR NZ, GNL5        ; 如果不为 0，则不读。
01980 GNL3 LD A, 0         ; 复位扇区缓冲区索引
                           为 0。
01990   LD (FI), A
02000   LD HL, (RCOUNT)    ; 准备测试文件的终
                           点。
02010   INC HL             ; 指向读的扇区计数。
02020   LD (RCOUNT), HL
02030   LD BC, O           ; 读下一个扇区。
02040   LD DE, DCB         ; 覆盖 DCB 地址。
02050   LD HL, (BADDR)     ; 扇区缓冲区地址。
02060   CALL READ          ; 读下一个区段。
02070   LD A, 1            ; 重新设置缓冲器
02080   LD (PF), A         ; 中数据的传送标志。
02090 GNL5 LD DE, (RCOUNT)
                           ; 现在测试文件的终
                           点。
02100   LD HL, (DCB+ERN)   ; 从 DCB 得到最后一
                           个扇区号。
02110   XOR A              ; 为了减，先清除进位
                           标志。
02120   SBC HL, DE         ; 已读了最后一个扇区
                           吗?
02130   JR NZ, GNL10       ; 如不是最后一个，则
                           不为 0。
02140   LD A, (DCB+EOF)    ; 在最后一个扇区内，
                           EOD (数据结束)。
02150   LD B, A            ; 到达数据结束处吗？
02160   LD A, (FI)         ; 当前扇区的索引。
02170   SUB B              ; 必须小于等于 EOD
                           索引。
02180   JR C, GNL10        ; 如数据未结束，进位
                           标志置位。
02190   XOR A              ; 表示文件结束。
02200   RET                ; 返回到主程序。
02210 GNL10 LD HL, (BADDR)
                           ; 扇区缓冲区地址。
02220   LD A, (FI)         ; 当前缓冲区索引。
02230   LD C, A            ; 用于 16 位算术运算。
02240   LD B, 0            ; 同上。
02250   ADD HL, BC         ; 缓冲器中的当前行地
                           址。
02260   LD DE, (40A7H)     ; BASIC 行缓冲区地
                           址。
02270 GNL15 LD A, (HL)     ; 把行从扇区缓冲区移
                           到 BASIC 行缓冲区。
02280   LD (DE), A
02290   INC DE             ; 指向目的地址。
02300   INC C              ; 移动一个字符的计
                           数。
02310   JR C, GNL3         ; 如行溢出该扇区，则
02320;★                    转移。
02330   INC HL             ; 不溢出，源地址加 1。
02340   SUB 0DH            ; 测试行的终点。
02350   JR NZ, GNL15       ; 循环，直到行结束。
02360   DEC DE             ; 在行缓冲区内，往回
                           退一个字符，并以一
                           个 0 来结束。
02370   LD (DE), A
02380   LD A, C            ; 为下一行保存结束缓
                           冲区索引。
02390   LD (FI), A
02400   OR A               ; 表示还有数据。
02410   RET                ; 返回到调用它的程
                           序。
02420;★
02430;★把缓冲区内的行进行代号化。然后加到 PST
02440;★中。
02450 ATOB LD HL, (40A7H)
                           ; 行缓冲区地址。
02460   CALL 1E5AH         ; 得到二进制的行号。
02470   PUSH DE            ; 保存 DE 中的二进
                           制行号。
02480   PUSH HL            ; 保存行缓冲器地址。
02490   LD HL, (LINE)      ; 开始的覆盖行号。
02500   RST 18H            ; 与当前行比较。
02510   JR Z, ATOB5        ; 如果相等，则转移。
02520   JR NC, ERR         ; 如进入行号小于覆盖
02530;★                    行号，则错误。
02540 ATOB5 POP HL         ; 恢复行地址。
02550   CALL 1BC0H         ; 程序行代号化。
02560   LD HL, (40F9H)     ; 当前 PST 的终点。
02570   PUSH HL            ; 保存这行的地址。
02580   ADD HL, BC         ; 加新的行长度。
02590   LD (40F9H), HL     ; 下一行的开始。
02600   PUSH HL            ; 于是我们能把它装入
                           DE。
02610   POP DE
02620;★
02630;★在加入的新的行中修改指针到下一行。然后把
02640;★这一行的二进制代号传送到 PST 中。
02650;★
02660   POP HL             ; 在 PST 中这一行的
                           地址。
02670   LD (HL), E         ; 下一行地址的 LSB。
02680   INC HL
02690   LD (HL), D         ; 下一行地址的 MSB。
```

```
02700   INC HL              ; 二进制行号的起点。
02710   POP DE              ; 二进制行号。
02720   LD (HL), E          ; 行号的 LSB。
02730   INC HL
02740   LD (HL), D          ; 行号的 MSB。
02750   INC HL              ; 指向行内的第一个字
                              符。
02760   EX DE, HL           ; DE=该行的 PST。
02770   LD HL, (40A7H)      ; 代号化的行地址。
02780   DEC HL
02790   DEC HL
02800 ATOB10 LD A, (HL)     ; 得到一个代号化的字
                              节。
02810   LD (DE), A          ; 把它移到 PST。
02820   INC HL
02830   INC DE
02840   OR A                ; 测试 EOS (语句结
                              束)。
02850   JR NZ, ATOB10       ; 如不是语句结束(字
                              节内容为 0)，则转
                              移。
02860   LD (DE), A          ; 用两字节 0 作程序结
                              束。
02870   INC DE
02880   LD (DE), A
02890   RET                 ; 返回到调用它的程
                              序。
02900;★
02910;★ERROR 处理——恢复堆栈空间
02920;★
02930 ERR POP AF            ; 清除堆栈。
02940   POP AF               ; 清除堆栈。
02950   POP AF               ; 清除堆栈。
02960   LD HL, 0             ; 撤消扇区缓冲区分
                              配。
02970   ADD HL, SP           ; CSP (当前堆栈指
                              针)。
02980   LD BC, 256           ; 扇区缓冲区的大小。
02990   ADD HL, BC           ; 计算新的 CSP。
03000   LD SP, HL            ; 设置新的 CSP。
03010 ERR10 POP AF           ; 清除堆栈。
03020   POP AF               ; 清除堆栈。
03030   POP AF               ; 清除堆栈。
03040   POP AF               ; 清除堆栈。
03050   POP AF               ; 清除堆栈。
03060   LD A, 2              ; 语法错误代码。
03070   JP 1997H             ; 给出错误信息，返回
                              BASIC。
03080;★
03090;★常数和计数器
03100;★
03110 LINE DEFW 0            ; 覆盖行号。
03120 CSP DEFW 0             ; 保存入口时的 CSP。
03130 TYPE DEFB 0            ; 原来的数据类型。
03140 LFN DEFM 'F$'          ; 公用变量名。
03150 DEFB 0
03160 DCB DEFS 32            ; 覆盖 DCB。
03170 BADDR DEFW 0           ; 堆栈中扇区缓冲区地
                              址。
03180 VARADR DEFW 0          ; 堆栈中变量地址。
03190 CEPST DEFW 0           ; PST 的当前终点。
03200 LVLT DEFW 0            ; VLT 的长度。
03210 SNVLT DEFW 0           ; 新的 VLT 的起始地
                              址。
03220 LSVLT DEFW 0           ; 简单变量 VLT 的
                              长度。
03230 PF    DEFB 0           ; 传送标志。
03240 FI    DEFB 0           ; 扇区缓冲区索引。
03250 RCOUNT DEFW -1         ; 读扇区计数。
03260   END
```

第七章　新的 ROM BASIC 解释程序

第八章里的注释是根据早期的三片 ROM 芯片给出的，假如你现在的机器是两片 ROM 的结构，则反汇编出来稍有一点不同。

最新给出“MEM SIZE?”信息的 ROM 和老的 ROM 之间的差别将在下面给出。目的码前面标注着星号的那些存储单元中的内容将和下一章给出的内容不同。当运行反汇编程序时一定要按页地小心检查，看程序的有哪些地方不同。

本章给出的注释也是按反汇编的格式编排的。

地址	目的码	源程序		注　释
0050:	0D	DEC	C	; 不按 |SHIFT| 时回车（|ENTER|）的代码（0DH）。★★★ ASCII 数值★★★★★★★★★★★★★★★★
0051:	0D	DEC	C	; 按 |SHIFT| 时回车（|ENTER|）的代码（0DH）。
0052:	1F	RRA		; 不按 |SHIFT| 时 |CLEAR| 的代码（1FH）。
0053:	1F	RRA		; 按 |SHIFT| 时 |CLEAR| 的代码（1FH）。
0054:	01015B	LD	BC, 5B01H	; 不按(和按) |SHIFT| 时 |BREAK| 的代码（01H）；不按 |SHIFT| 时 |↑| 的代码（5BH）。
0057:	1B	DEC	DE	; 按 |SHIFT| 时 |↑| 的代码（1BH）。
0058:	0A	LD	A, (BC)	; 不按 |SHIFT| 时 |↓| 的代码（0AH）。
0059:*	00	NOP		; 按 |SHIFT| 时 |↓| 的代码（00H）。
005A:	08	EX	AF,AF'	; 不按 |SHIFT| 时 |←| 的代码（08H）。
005B:	1809	JR	0066H	; 按 |SHIFT| 时 |←| 的代码（18H），不按 |SHIFT| 时 |→| 代码（09H）。
005D:	19	ADD	HL, DE	; 按 |SHIFT| 时 |→| 的代码（19H）。
005E:	2020	JR	NZ, 0080H	;不按和按 |SHIFT| 时空格的代码（20H）。
00FC:*	210E01	LD	HL, 010EH	; 'R/S L2 BASIC' 信息的首地址。
0105:	4D	LD	C, L	; M ★★★ MEM SIZE ★★★★★★★★★★★★★★
0106:	45	LD	B, L	; E
0107:	4D	LD	C, L	; M
0108:	*2053	JR	NZ, 015DH	; 空格,S
010A:	*49	LD	C, C	; I
010B:	*5A	LD	E, D	; Z
010C:	*45	LD	B, L	; E

地址	目的码	源程序		注　释
010D:	*00	NOP		; 信息的结束符。
010E:	*52	LD	D, D	; R ★★★ R/S L2 BASIC ★★★★★★★★★★★★
010F:	*2F	CPL		; /
0110:	*53	LD	D, E	; S
0111:	*204C	JR	NZ, 015FH	; 空格, L
0113:	*322042	LD	(4220H), A	; 2, 空格, B
0116:	*41	LD	B, C	; A
0117:	53	LD	D, E	; S
0118:	*49	LD	C, C	; I
0119:	*43	LD	B, C	; C
011A:	*0D	DEC	C	; 回车。
011B:	*00	NOP		; 信息的结束符。
011C:	*C5	PUSH	BC	; 把键盘编码扫描中有效的那一行地址暂存起来。
011D:	*010005	LD	BC, 0500H	; 需要延迟的计数值。
0120:	*CD6000	CALL	0060H	; 延迟 7.33 毫秒。★★★防止键盘抖动的程序★★★★
0123:	*C1	POP	BC	; 取回该行有效地址。
0124:	*0A	LD	A, (BC)	; 从该有效地址上重新取回原标志位。
0125:	*A3	AND	E	; 把原标志位与现有标志位合在一起。
0126:	*C8	RET	Z	; 当为零时, 即第二次测试不是该行有效, 返回到调用它的程序。
0127:	*7A	LD	A, D	; 否则,我们认为这一行有效。
0128:	*07	RLCA		; 行的指针乘 2。
0129:	*07	RLCA		; 行的指针乘 4。
012A:	*C3FE03	JP	03FEH	; 返回到键盘驱动程序的其余部分中去。
0248:	*0660	LD	B, 60H	; 将延迟 476 (或 703) 毫秒,取决于 CPU 时钟频率。
024F:	*0685	LD	B, 85H	; 然后延迟 865 (或 975) 毫秒。
02E2:	*20ED	JR	NZ, 02D1H	; 若不相同,跳到磁带机上的下一个程序去。
02E4:	*23	INC	HL	; 有一个字符相同时, 地址加 1, 并指向输入名称的下一个字符。
03FB:	*C31C01	JP	011CH	; 转到防止按键后抖动的程序。若是合法的键入字符,则返回到 3FEH; 否则返回到调用它的程序。
0683:	*20F1	JR	NZ, 0676H	; 把整个数据块传送程序循环 128 次。
1225:	E7	RST	20H	; 测试数据类型是否为双精度或字符串。
1226:	*300B	JR	NC, 1233H	; 若是双精度,则转移。

地址	目的码	源程序		注释
124D:	*B7	OR	A	; 置标志位。
1265:	*F24312	JP	P, 1243H	; 原说明不变。
2067:	3E01	LD	A, 01H	; A = 打印机的设备代码。★★★ LPRINT 子程序 ★★
2069:	329C40	LD	(409CH), A	; 设置当前的系统设备为打印机。
206C:	*C37C20	JP	207CH	
206F:	CDCA41	CALL	41CAH	; DOS 出口。★★★ PRINT 子程序 ★★★★★★★★
2072:	*FE23	CP	23H	; 测试#符号。
2074:	*2006	JR	NZ, 207CH	; 若不是 PRINT #,则转移。
2076:	*CD8402	CALL	0284H	; 在磁带机文件上录制引导码。★★★ PRINT #子程序 ★★★★★★★★★★★★★★★★★★★★★★★★★
2079:	*329C40	LD	(409CH), A	; 设置当前系统设备为磁带机。
207C:	*2B	DEC	HL	; 指向字符串代码中前一个字符。
207D:	*D7	RST	10H	; 重新测试前一个字符。
207E:	*CCFE20	CALL	Z, 20FEH	; 若字符串结束,写一个回车。
2081:	*CA6921	JP	Z, 2169H	; 若字符串结束,关闭磁带机,并返回。
2084:	*F620	OR	20H	; 字符串没有结束,将可能的 40H 转换为 60H。
2086:	*FE60	CP	60H	; 然后测试@符号
2088:	*201B	JR	NZ, 20A5H	; 若不是 PRINT@, 转移
208A:	*CD012B	CALL	2B01H	; 计算@表达式,结果放在 DE 中。★★★ PRINT @ 子程序 ★★★★★★★★★★★★★★★★★★★★★★
208D:	*FE04	CP	04H	; A=MSB, 测试@后的数值是否大于 1023。
208F:	*D24AIE	JP	NC, 1E24H	; 若@后数值 >1023, FC 错误。
2092:	*E5	PUSH	HL	; 把当前代码字符串地址暂存起来。
2093:	*21003C	LD	HL, 3C00H	; HL=显示缓冲器的首地址。
2096:	*19	ADD	HL, DE	; 得到显示的位置。
2097:	*222040	LD	(4020H), HL	; 把地址存到显示器 DCB 中作为光标地址。
209A:	*7B	LD	A, E	; 然后取一行中的位置。
209B:	*E63F	AND	3FH	; 把它截在 63 以内。
209D:	*32A640	LD	(40A6H), HL	; 然后,作为行内当前位置存储起来。
20A0:	*E1	POP	HL	; 取回代码字符串地址(项目清单的首地址)。
20A1:	*CF	RST	08H	; 验证显示位置后面的逗号。
20A2:	*2C	INC	L	; 给出逗号的代码 2CH——',',。
20A3:	*18C7	JR	206CH	; 从项目清单中取出第一个变量。
20A5:	*7E	LD	A, (HL)	; 从代码字符串中取下一个字符。
20A6:	*FEBF	CP	BFH	; 测试是否为 USING 的代号。
20A8:	*CABD2C	JP	Z, 2CBDH	; 若是 USING 的代号,则转移。
20AB:	*FEBC	CP	BCH	; 测试是否用 TAB 的代号。
20AD:	*CA3721	JP	Z, 2137H	; 若是 TAB 的代号,则转移。
20B0:	*E5	PUSH	HL	; 暂存代码字符串地址。

地址	目的码	源程序		注　释
20B1:	*FE2C	CP	2CH	; 测试是否有一个逗点。
20B3:	*2853	JR	Z, 2108H	; 若有一个逗点,则去取下一项。
20B5:	*FE3B	CP	3BH	; 若没有逗点,则测试有无分号。
20B7:	*285E	JR	Z, 2117H	; 若有分号,则去取下一项。
20B9:	CD3723	CALL	2337H	; 计算下一项显示的值。
20BC:	*E3	EX	(SP), HL	; 暂存当前代码字符串地址, HL=当前项的地址。
20F6:	*C37C20	JP	207CH	; 在找到语句结束 (EOS) 前,循环。
213A:	*E67F	AND	7FH	; 结果在 A 寄存器中,使它不超过 127。
2166:	*C38120	JP	2081H	; 处理 PRINT TAB 语句的后一部分。
226A:	*00	NOP		; 取消原有的对 FD 错误的测试。
226B:	*00	NOP		;
226C:	*00	NOP		;
226D:	*00	NOP		;
226E:	*00	NOP		;
2C1F:	*D6B2	SUB	B2H	; 测试是否 CLOAD? ★★★ CLOAD 子程序 ★★★★
2C21:	*2802	JR	Z, 2C25H	; 若是 "CLOAD?", 转移。
2C23:	*AF	XOR	A	; 表示是 CLOAD。
2C24:	*012F23	LD	BC, 232FH	; 2C25: CPL, 若是 CLOAD?, 则 A=−1, 若是 CLOAD, 则 A=0。
2C27:	*F5	PUSH	AF	; 2C26: INC HL, HL 指向文件名称, 暂存 CLOAD? 或 CLOAD 标志。
2C28:	*7E	LD	A, (HL)	; 从代码字符串中取下一个字符,它应是文件名称。
2C29:	*B7	OR	A	; 置标志位。
2C2A:	*2807	JR	Z, 2C33H	; 若一行已结束,转移。
2C2C:	*CD2723	CALL	2327H	; 求表达式的数值(取文件名称)。
2C2F:	*CD132A	CALL	2A13H	; 把文件名称的地址放入 DE 中。
2C32:	*1A	LD	A, (DE)	; 取文件名称。
2C33:	*6F	LD	L, A	; 把它移入 L 寄存器。
2C34:	*F1	POP	AF	; 取回 CLOAD? 或 CLOAD 标志。
2C35:	*B7	OR	A	; 按照标志,置状态寄存器。
2C36:	*67	LD	H, A	; H=CLOAD? 或 CLOAD 标志, L=文件名称。
2C37:	*222141	LD	(4121H), HL	; 把标志和文件名称存入 WRA1 中。
2C3A:	*CC4D1B	CALL	Z, 1B4DH	; 若是 CLOAD, 调用 NEW 子程序,把系统变量初始化。
2C3D:	*210000	LD	HL, 0000H	; 在寻找引导码和同步字节时,
2C40:	*CD9302	CALL	0293H	; 选择指定的设备。
2C43:	2A2141	LD	HL, (4121H)	; 取回 CLOAD? 或 CLOAD 标志和文件名称。

第八章　老的 ROM BASIC 解释程序

本章给出的存储单元和注释都是针对老的三片 ROM 芯片的，它与最新的两片 ROM 芯片之间的差别，请参阅第七章。

地址	目的码	源程序		注　释
0000:	F3	DI		; 接通电源时 IPL 的入口点，关闭时钟和磁盘的中断。
0001:	AF	XOR	A	; 清除 A 寄存器和标志位。
0002:	C37406	JP	0674H	; 转移到 IPL 程序的起点。
0005:	C30040	JP	4000H	; ★★★ 比较 ★★★★★★★★★★★★★★★★★★
0008:	C30040	JP	4000H	; RST 08H (JP 1C96H)，把 RST 08 后边的数值与下一个字符进行比较，不相等时给出句法错误。
000B:	E1	POP	HL	; } 这两条指令 LEVEL II 没有使用。
000C:	E9	JP	(HL)	;
000D:	C39F06	JP	069FH	; 转移去装入并执行扇区的装入程序。
0010:	C30340	JP	4003H	; RST 10H (JP 1D78H)，取入并检查下一个字符。
0013:	C5	PUSH	BC	; 保存 BC——键盘子程序。
0014:	0601	LD	B, 01H	; B=入口代码。
0016:	182E	JR	0046H	; 转入驱动子程序 (03C2H)。
0018:	C30640	JP	4006H	; RST 18H (JP 1C90H)，比较 DE 与 HL。
001B:	C5	PUSH	BC	; 保存 BC——显示器和打印机子程序。
001C:	0602	LD	B, 02H	; B=入口代码。
001E:	1826	JR	0046H	; 转入驱动子程序 (03C2H)。
0020:	C30940	JP	4009H	; RST 20H (JP 25D9H)，确定数据的类型。
0023:	C5	PUSH	BC	; 保存 BC。
0024:	0604	LD	B, 04H	; B=入口代码。
0026:	181E	JR	0046H	; 转入驱动子程序 (03C2H)。
0028:	C30C40	JP	400CH	; RST 28H (没有 DOS 时便返回；工作于 DOS2.0 时—JP 4BA2H)。
002B:	111540	LD	DE, 4015H	; 把键盘的 DCB 地址装入 DE。★★★键盘扫描★★★
002E:	18E3	JR	0013H	; 转移到键盘驱动程序。
0030:	C30F40	JP	400FH	; RST 30H (没有 DOS 时便返回；工作于 DOS2.0 时—JP 44B4H)。
0033:	111D40	LD	DE, 401DH	; 把显示器的 DCB 地址装入 DE。★★★荧光屏显示★
0036:	18E3	JR	001BH	; 转移到显示器驱动程序。
0038:	C31240	JP	4012H	; RST 38H (没有 DOS 时便关闭中断和返回；工作于 DOS 2.0 时——JP 4518H)，所有中断的入口点。
003B:	112540	LD	DE, 4025H	; 把打印机 DCB 指针装入 DE。★★★★★★★★★

地址	目的码	源程序		注　释
003E:	18DB	JR	001BH	; 转入打印机驱动程序。
0040:	C3D905	JP	05D9H	; 转入判断键入字符程序。
0043:	C9	RET		;
0044:	00	NOP		; } LEVEL II 没有使用这三条指令。
0045:	00	NOP		;
0046:	C3C203	JP	03C2H	; 转入驱动程序。
0049:	CD2B00	CALL	002BH	; 选通键盘。★★★等待键盘输入 ★★★★★★★★★
004C:	B7	OR	A	; 测试按键与否。
004D:	C0	RET	NZ	; 若按了键,则返回。
004E:	18F9	JR	0049H	; 没有按,则循环等待。
0050:	0D	DEC	C	; ★★★ 以下是 03E3H 键盘子程序的表, |ENTER|、|CLEAR|、|BREAK|、|↑|、|↓|、|←|、|→| 和空格的 ASCII 代码值 ★★★★★★★★★★★★★★★★ 没有按 |SHIFT| 时 |ENTER| 的代码 (0DH)。
0051:	0D	DEC	C	; 按 |SHIFT| 时 |ENTER| 的代码 (0DH)。
0052:	1F	RRA		; 没有按 |SHIFT| 时 |CLEAR| 的代码 (1FH)。
0053:	1F	RRA		; 按 |SHIFT| 时 |CLEAR| 的代码 (1FH)。
0054:	01015B	LD	BC, 5B01H	; 没有按和按 |SHIFT| 时 |BREAK| 的代码 01H, 没按 |SHIFT| 时 |↑| 的代码 (5BH)。
0057:	1B	DEC	DE	; 按 |SHIFT| 时 |↑| 的代码 (1BH)。
0058:	0A	LD	A, (BC)	; 没按 |SHIFT| 时 |↓| 的代码 (0AH)。
0059:	1A	LD	A, (DE)	; 按 |SHIFT| 时 |↓| 的代码 (1AH)。
005A:	08	EX	AF,AF'	; 没按 |SHIFT| 时的 |←| 代码 (08H)。
005B:	1809	JR	0066H	; 按 |SHIFT| 时 |←| 的代码 (18H), 没按 |SHIFT| 时 |→| 的代码 (09H)。
005D:	19	ADD	HL, DE	; 按 |SHIFT| 时 |→| 的代码 (19H)。
005E:	2020	JR	NZ, 0080H	; 没按 |SHIFT| 和按 |SHIFT| 时空格的代码 (20H)。
0060:	0B	DEC	BC	; 减循环计数★★★延迟 ((BC−1)×26+17)×2.255 个 T 状态 ★★★★★★★★★★★★★★★★★★★★★
0061:	78	LD	A, B	; 测试循环计数是否为零。
0062:	B1	OR	C	; 把计数的 MSB 和 LSB 合并。
0063:	20FB	JR	NZ, 0060H	; 循环,直到延迟计数为零
0065:	C9	RET		; 返回到调用它的程序。
0066:	310006	LD	SP, 0600H	; 复位 IPL 入口。★★★复位处理 ★★★★★★★★★
0069:	3AEC37	LD	A, (37ECH)	; 把控制器的状态取来。状态=00H 若有 EI (扩展接口) 和磁盘准备就绪了。状态 =80H 若 EI 和磁盘没有准备就绪。状态 =FFH 若 EI 是关着,或者没有 EI。
006C:	3C	INC	A	; 测试控制器的状态。

地址	目的码	源程序		注　释
006D:	FE02	CP	02H	; 如果没有 EI，状态通常为 FFH。
006F:	D20000	JP	NC, 0000H	; 若控制器可寻址，C 标志复位，转入公用 IPL 程序。
0072:	C3CC06	JP	06CCH	; 系统没有联接磁盘，转移去给出 BASIC′ READY′ 的提示符。
0075:	118040	LD	DE, 4080H	; 这里是没联磁盘时，打开电源和按 BREAK 键的入口点。
0078:	21F718	LD	HL, 18F7H	; 把初始数据移到通讯区。
007B:	012700	LD	BC, 0027H	; 传送的字节数。
007E:	EDB0	LDIR		; 把 ROM 中 18F7H—191DH 的内容移入 RAM 中 4080H—40A6H。装入除法支援程序。把公用区初始化为： 4080—408D 除法支援程序。 408E 1E4AH 用户子程序地址。 4090 E64DDB 随机数的种子。 4093 IN A, (00H) INP（输入）的主要指令。 4095 RET 4096 OUT (00H), A OUT（输出）的主要指令。 4098 RET 4099 00 最后一个键入的字符。 409A 00 错误计数器。 409B 00 当前一行字符的计数。 409C 00 输出设备类型。 409D 00 显示器上一行的容量(最多为 64 个字符)。 409E 30H 在 PRINT 时行的容量。 409F 00 保留 40A0—40A1 字符串区的首地址。 40A2 FFFFH 最初的 BASIC 行号。 40A4 429EH 程序语句表（PST）的地址
0080:	21E541	LD	HL, 41E5H	; 继续使通讯区初始化。
0083:	363A	LD	(HL), 3AH	; 把 3AH 放到 41E5H 中。
0085:	23	INC	HL	; 指向 41E6H。
0086:	70	LD	(HL), B	; 把 0 放入 41E6H 中。
0087:	23	INC	HL	; 指向 41E7H。
0088:	362C	LD	(HL), 2CH	; 41E7H 为 2CH。
008A:	23	INC	HL	; HL=41E8H，把输入缓冲区指针（40A7H）
008B:	22A740	LD	(40A7H), HL	; 设置成键盘缓冲区（41E8H）。
008E:	112D01	LD	DE, 012DH	; JP 指令的地址字段。
0091:	061C	LD	B, 1CH	; 将 4152—41A5H 都预置成 JP 12DH，这样当有磁盘 BASIC 的命令输入时，将给出 L3 的错误信息。
0093:	215241	LD	HL, 4152H	;
0096:	36C3	LD	(HL), C3H	; 把 JP 指令代码 C3H 放到 4152H 中。
0098:	23	INC	HL	; 地址增 1，指向地址字段的 LSB。
0099:	73	LD	(HL), E	; 把 2DH 放到 4153H 中。

地址	目的码	源程序		注　释
009A:	23	INC	HL	; 地址加 1，指向地址字段的 MSB。
009B:	72	LD	(HL), D	; 把 01 放到 4154H 中，给出 JP 012DH。
009C:	23	INC	HL	; 指向下一个 JP 指令的地址。
009D:	10F7	DJNZ	0096H	; 重复 28 次(共 84 个存储单元)，在 4152—41A5H 中给出 JP 12DH。
009F:	0615	LD	B, 15H	; 为 DOS 出口返回的循环计数。
00A1:	36C9	LD	(HL), C9H	; 将 C9H 放入 41A6H 中，给出 RET 指令。
00A3:	23	INC	HL	; 41A9: RET 把 DOS 的出口向量变为返回指令。
00A4:	23	INC	HL	; ⋮
00A5:	23	INC	HL	; 41E2: RET
00A6:	10F9	DJNZ	00A1H	; 重复，在 41A6, 41A9, …, 41E2H 中给出 RET 指令。
00A8:	21E842	LD	HL, 42E8H	; 把地址装入 HL
00AB:	70	LD	(HL), B	; 这样就能把 0 存储在 42E8H 中。
00AC:	31F841	LD	SP, 41F8H	; 在 IPL 过程中，堆栈地址为 41F8H。
00AF:	CD8F1B	CALL	1B8FH	; 对 BASIC 的指针和变量初始化。
00B2:	CDC901	CALL	01C9H	; 清除荧光屏。
00B5:	210501	LD	HL, 0105H	; 信息 'MEMORY SIZE?' 的指针。
00B8:	CDA728	CALL	2BA7H	; 输出信息。
00BB:	CDB31B	CALL	1BB3H	; 显示'?'，等待用户的输入。
00BE:	38F5	JR	C,00B5H	; 如果输入 BREAK 键，则重新询问 MEMORY SIZE? 要求用户输入内存保护。
00C0:	D7	RST	10H	; 对输入的字符进行识别。
00C1:	B7	OR	A	; 置标志位。
00C2:	2012	JR	NZ, 00D6H	; 若不是输入字符串的结束，便转移。
00C4:	214C43	LD	HL, 434CH	; 若只输入 CR (回车)，动态地确定存储器的容量。
00C7:	23	INC	HL	; 从 17220 开始到 65535 测试存储器。
00C8:	7C	LD	A, H	; 下一个测试地址的 LSB。
00C9:	B5	OR	L	; 与下一个测试地址的 MSB 合并在一起。
00CA:	281B	JR	Z, 00E7H	; 扫描到存储器顶端 65535，测试需要的最小存储量。
00CC:	7E	LD	A, (HL)	; 取存储器测试单元中的原始内容。
00CD:	47	LD	B, A	; 把它保存起来以便复原。
00CE:	2F	CPL		; 取反码(给出测试图样)。
00CF:	77	LD	(HL),A	; 存储测试图样。
00D0:	BE	CP	(HL)	; 把存储单元的内容与测试图样相比较。
00D1:	70	LD	(HL), B	; 把原始内容存放回去。
00D2:	28F3	JR	Z, 00C7H	; 地址存在，转去测试存储器的最小量。
00D4:	1811	JR	00E7H	; 该地址不存在，地址加 1，再次测试。
00D6:	CD5A1E	CALL	1E5AH	; 得到二进制的等效值放入 DE 和 A 中。
00D9:	B7	OR	A	;
00DA:	C29719	JP	NZ, 1997H	; 若 Z 标志复位，则给出 SN 错误信息。
00DD:	EB	EX	DE, HL	; HL=存储器容量。

地址	目的码	源程序		注　释
00DE:	2B	DEC	HL	; 存储器容量减1。}测试存储器容量,以确保它的存在。
00DF:	3E8F	LD	A, 8FH	; 比较数值。
00E1:	46	LD	B, (HL)	; 取存储器的内容保存在 B 寄存器中。
00E2:	77	LD	(HL), A	; 存储测试图样。
00E3:	BE	CP	(HL)	; 把存储的测试图样与 A 寄存器中的图样比较。
00E4:	70	LD	(HL), B	; 把存储器的原始内容放回去。
00E5:	20CE	JR	NZ, 00B5H	; 并不存在指定的存储器容量,重新询问。
00E7:	2B	DEC	HL	; 存储器的总量−2。
00E8:	111444	LD	DE, 4414H	; DE=17428
00EB:	DF	RST	18H	; 测试是否是最小的存储器容量(17428)。
00EC:	DA7A19	JP	C, 197AH	; 若 C 置位, OM 错误,没有足够的存储器。
00EF:	11CEFF	LD	DE, FFCEH	; 为字符串区保留 50 个字节,将这个缺项常数值放入 DE 中。
00F2:	22B140	LD	(40B1H), HL	; 保存存储器的容量。
00F5:	19	ADD	HL, DE	; 从可使用的存储器末址减去字符串容量。
00F6:	22A040	LD	(40A0H), HL	; 把字符串区的首地址存放起来。
00F9:	CD4D1B	CALL	1B4DH	; 把 BASIC 的所有变量和指针初始化。
00FC:	211101	LD	HL, 010EH	; 'RADIO…BASIC' 的信息指针。
00FF:	CDA728	CALL	28A7H	; 输出信息。
0102:	C3191A	JP	1A19H	; 转移到 READY 程序
0105:	4D	LD	C, L	; M ★★★ 'MEMORY SIZE' 的信息 ★★★★★★★★
0106:	45	LD	B, L	; E
0107:	4D	LD	C, L	; M
0108:	4F	LD	C, A	; O
0109:	52	LD	D, D	; R
010A:	59	LD	E, C	; Y
010B:	2053	JR	NZ, 0160H	; 空格, S
010D:	49	LD	C, C	; I
010E:	5A	LD	C, D	; Z
010F:	45	LD	B, L	; E
0110:	00	NOP		; 00——信息结束符。
0111:	52	LD	D, D	; R ★★★ 'RADIO SHACK LEVEL II BASIC' 信息 ★★
0112:	41	LD	B, C	; A
0113:	44	LD	B, H	; D
0114:	49	LD	C, C	; I
0115:	4F	LD	C, A	; O
0116:	2053	JR	NZ, 016BH	; 空格, S
0118:	48	LD	C, B	; H
0119:	41	LD	B, C	; A
011A:	43	LD	B, E	; C
011B:	4B	LD	C, E	; K

地址	目的码	源程序		注释
011C:	204C	JR	NZ, 016AH	; 空格, L
011E:	45	LD	B, L	; E
011F:	56	LD	D (HL)	; V
0120:	45	LD	B, L	; E
0121:	4C	LD	C, H	; L
0122:	2049	JR	NZ, 016DH	; 空格, I
0124:	49	LD	C, C	; I
0125:	2042	JR	NZ, 0169H	; 空格, B
0127:	41	LD	B, C	; A
0128:	53	LD	D, E	; S
0129:	49	LD	C, C	; I
012A:	43	LD	B, E	; C
012B:	0D	DEC	C	; 0DH——回车代码。
012C:	00	NOP		; 00H——信息结束符。
012D:	1E2C	LD	E, 2CH	; L3 错误的代码。★★★★★★★★★★★★★★★★★
012F:	C3A219	JP	19A2H	; 转移到错误程序,显示出 L3 错误。
0132:	D7	RST	10H	; 定位于下一个字符。★★★ 用于测试 (POINT/SET/RESET) ★★★★★★★★★★★★★★★★★★★★★
0133:	AF	XOR	A	; 若是 POINT, 则 A=0。POINT (x, y)
0134:	013E80	LD	BC, 803EH	; 否则, 0135: LD A, 80H SET 程序, A=−1 SET (x, y)
0137:	013E01	LD	BC, 013EH	; 0138: LD A, 01H RESET 程序,A=1 RESET (x, y)
013A:	F5	PUSH	AF	; 暂存指明 POINT/SET/RESET 入口的标志。
013B:	CF	RST	08H	; 检测下一个字符,寻找左括号 '('。
013C:	28CD	JR	Z, 010BH	; 013C: 定义左括号代码, 28H——'(', 用作 RST 08H 的参数。
013E:	1C	INC	E	; 013D: CALL 2B1CH, 求出第一个变量 x 的数值。
013F:	2B	DEC	HL	; 结果在 A 寄存器中。
0140:	FE80	CP	80H	; 把 x 的坐标值与 128 比较。
0142:	D24A1E	JP	NC, 1E4AH	; 若 x=>128, 产生 FC 错误。
0145:	F5	PUSH	AF	; 把 x 坐标保存起来。
0146:	CF	RST	08H	; 检测输入字符串的下一个字符。
0147:	2C	INC	L	; 是否是逗号——', 。
0148:	CD1C2B	CALL	2B1CH	; 求第二个变量 y 的数值。
014B:	FE30	CP	30H	; 把结果放在 A 寄存器中,与 48 比较。
014D:	D24A1E	JP	NC, 1E4AH	; 若 y=>48, 则是 FC 是错误。
0150:	16FF	LD	D, FFH	; 准备把 y 坐标值除以 3, 给出 Q+R。

说明 计算指定点的存储器地址。存储器的作图区域在地址 3C00H—3FFFH 之间。每 6 比特 (2×3) 的一个图块位置是用地址 3C00H 开始的一个 8 比特的字节表示的。这个图块位置以 6 位二进制数 (比特串)存储在存储器中,在一个字节中是向右排齐的。在一个字节中二进制位从右向左编号 (就如我们已

地址	目的码	源程序		注释

想得的那样),在作图中我们用位 0 至位 5,位 6 没有用到,位 7 为 1,表示作图方式。在每个图形块内,各个小方块的坐标以一个图形块"字节"表示如下:位 0 和位 1 分别表示第 1 行的第 0 点和第 1 点(第 0 点在左),而位 2 和位 3 分别表示第二行的第 0 点和第 1 点,依次类推。

```
0152:  14      INC   D          ; D=Q。
0153:  D603    SUB   03H        ; 通过减来进行除。
0155:  30FB    JR    NC, 0152H  ; 循环,直到余数<3。
0157:  C603    ADD   A, 03H     ; 使余数为正。
0159:  4F      LD    C, A       ; 把它保存在 C 中。
015A:  F1      POP   AF         ; A=x 的坐标。
015B:  87      ADD   A, A       ; 乘 2。
015C:  5F      LD    E, A       ; E=2x。
015D:  0602    LD    B, 02H     ; B= 位移的次数。
015F:  7A      LD    A, D       ; D 和 E (即 Q 和 2X) 右移 2 位。
0160:  1F      RRA              ; 使得 E 的位 1 的值
0161:  57      LD    D, A       ; 留在进位位中。
0162:  7B      LD    A, E       ; 若我们在第一列的(方形)图块位置,
0163:  1F      RRA              ; 那这一位为零。
0164:  5F      LD    E, A       ; 若在第二列(方形)图块位置,这一位为 1。
0165:  10F8    DJNZ  015FH      ;
0167:  79      LD    A, C       ; 现在,按照公式 2R+1+(0 或 1,对应于第一或第二列)。
0168:  8F      ADC   A, A       ; 计算该字节中点的位置。
0169:  3C      INC   A          ;
016A:  47      LD    B, A       ; 把比特位置计数存起来。
016B:  AF      XOR   A          ; 清除 A 寄存器和进位标志位。
016C:  37      SCF              ; 把进位标志置 1。
016D:  8F      ADC   A, A       ; 产生位的掩码值,
016E:  10FD    DJNZ  016DH      ; 以便确定我们要寻找的那个点的位置,把掩码留在 C
                                  中,
0170:  4F      LD    C, A       ; 计算含有那个图块位置的字节地址,存储在 DE 中。
0171:  7A      LD    A, D       ; 对寻找的那一位掩码。
0172:  F63C    OR    3CH        ; 从 Y/3 求得 Q, A=Q。
0174:  57      LD    D, A       ; 再存入 D, DE=我们要求的那个图块位置字节的地址。
0175:  1A      LD    A, (DE)    ; 取该字节的数值(比特串)。
0176:  B7      OR    A          ; 清除标志位。
0177:  FA7C01  JP    M, 017CH   ; 如果有图示标记,则转移。
017A:  3E80    LD    A, 80H     ; 否则,置图示标记。
017C:  47      LD    B, A       ; B= 这个图块位置的值(比特串)。
017D:  F1      POP   AF         ; 取出入口时的标志。
017E:  B7      OR    A          ; 测试标志位。
017F:  78      LD    A, B       ; A=这个图块位置字节的内容。
0180:  2810    JR    Z, 0192H   ; 若是 POINT 调用,则转移。
```

地址	目的码	源程序		注　释
0182:	12	LD	(DE), A	; 把图块字节内容再存储回去。
0183:	FA8F01	JP	M, 018FH	; 若是 SET 调用,则转移。
0186:	79	LD	A, C	; 否则,这一定是 RESET 调用。
0187:	2F	CPL		; 把要 RESET 的那一位关闭掉。
0188:	4F	LD	C, A	; 把关掉的那位保留在 C 中。
0189:	1A	LD	A, (DE)	; 从存储器中取那一个图块位置。
018A:	A1	AND	C	; 把指定的那一位关闭。
018B:	12	LD	(DE), A	; 然后再存放回去,这样我们就做完了。
018C:	CF	RST	08H	; 经测试右括号——')'后便可返回了。
018D:	29	ADD	HL, HL	; 给出右括号的代码 29H——')'。
018E:	C9	RET		; 返回到调用它的程序。
018F:	B1	OR	C	; SET 程序的继续。★★★把图块中的小方块点亮 ★★
0190:	18F9	JR	018BH	; 把这值再放回去,并返回调用它的程序。
0192:	A1	AND	C	; POINT 程序的继续。★★★ 把测试的那一位单独提取出来 ★★★★★★★★★★★★★★★★★★★★★★
0193:	C6FF	ADD	A, FFH	; 若这一位是点亮了,则产生溢出。
0195:	9F	SBC	A, A	; 若这一位没有点亮, A=0; 这一位点亮, A=−1。
0196:	E5	PUSH	HL	; 保存当前代码字符串地址。
0197:	CD8D09	CALL	098DH	; 把 00(假)或 −1(真)作为当前值保存起来。
019A:	E1	POP	HL	; 取回代码字符串地址。
019B:	18EF	JR	018CH	; 测试结束的括号,返回到调用它的程序。
019D:	D7	RST	10H	; 确定代码字符串中下一个字符。★★★ INKEY$ 子程序 ★★★★★★★★★★★★★★★★★★★★★★★
019E:	E5	PUSH	HL	; 把当前代码字符串地址暂存起来。
019F:	3A9940	LD	A, (4099H)	; 取键盘扫描中输入的最后一个字符,是否 |SHIFT| @。
01A2:	B7	OR	A	; 置标志位。
01A3:	2006	JR	NZ, 01ABH	; 若按了 |SHIFT| @, 则转移。
01A5:	CD5803	CALL	0358H	; 否则,重新扫描键盘。
01A8:	B7	OR	A	; 置标志位,以便测试结果。
01A9:	2811	JR	Z, 01BCH	; 若没有键入,则转移。
01AB:	F5	PUSH	AF	; 把键入的字符暂存起来。
01AC:	AF	XOR	A	; 清除 A 寄存器和标志位。
01AD:	329940	LD	(4099H), A	; 清除 |SHIFT| @键。
01B0:	3C	INC	A	; A=1, 要建立的字符串的长度。
01B1:	CD5728	CALL	2857H	; 确定有无可存放字符串的空余内存, 保存字符串长度, 地址在 4023H。
01B4:	F1	POP	AF	; A= 键入的字符。
01B5:	2AD440	LD	HL, (40D4H)	; HL=文字串库中的字符串的地址。
01B8:	77	LD	(HL), A	; 存储该字符。
01B9:	C38428	JP	2884H	; 把字符串移到文字串库区中。
01BC:	212819	LD	HL, 1928H	; 取 'READY' 信息的地址。★★★★★★★★★★★

地址	目的码	源程序		注释
01BF:	222141	LD	(4121H), HL	; 并移入当前字符串变量位置。
01C2:	3E03	LD	A, 03H	; 数据类型等于字符串。
01C4:	32AF40	LD	(40AFH), A	; 设置当前数据类型为字符串。
01C7:	E1	POP	HL	; 把信息地址传给 HL。
01C8:	C9	RET		; 返回调用它的程序。
01C9:	3E1C	LD	A, 1CH	; 清除荧光屏。★★★ 把光标位置置在原点 ★★★★★
01CB:	CD3A03	CALL	033AH	; 输出到荧光屏。
01CE:	3E1F	LD	A, 1FH	; 清除荧光屏的命令。
01D0:	C33A03	JP	033AH	; 输出到荧光屏,然后返回
01D3:	ED5F	LD	A, R	; 取当前 RAM 的刷新地址。★★★ RANDOM 子程序(使用刷新地址) ★★★★★★★★★★★★★★★★★
01D5:	32AB40	LD	(40ABH), A	; 存储随机数值。
01D8:	C9	RET		; 返回调用它的程序。
01D9:	2101FC	LD	HL, FC01H	; 在 4 位数据锁存器中把位 0 置1。★★★★★★★★★
01DC:	CD2102	CALL	0221H	; OUT (0FFH), 01
01DF:	060B	LD	B, 0BH	; B=延迟循环计数。
01E1:	10FE	DJNZ	01E1H	; 延迟 80 微秒。

说明 向磁带机写一位，假设这时磁带机的马达已启动运转了。 则写一个时钟脉冲，需由以下三步来完成: OUT (0FFH), 01; OUT, (0FFH), 02; OUT (0FFH), 00。一个时钟脉冲的总时间是 836 微秒。

01E3:	2102FC	LD	HL, FC02H	; 在 4 位数据锁存器中把位 1 置1。
01E6:	CD2102	CALL	0221H	; OUT (0FFH), 02
01E9:	060B	LD	B, 0BH	; B=延迟循环计数。
01EB:	10FE	DJNZ	01EBH	; 延迟 $3.25\times10^{-6}\times11\times2.26=80$ 微秒。
01ED:	2100FC	LD	HL, FC00H	; 把 4 位数据锁存器位 0 和位 1 复位。
01F0:	CD2102	CALL	0221H	; OUT (0FFH), 00
01F3:	065C	LD	B, 5CH	; B=延迟循环计数=92。
01F5:	10FE	DJNZ	01F5H	;延迟=$3.25\times10^{-6}\times92\times2.26=676$ 微秒。
01F7:	C9	RET		; 返回到调用它的程序。
01F8:	E5	PUSH	HL	; 关闭磁带机的入口点 ★★★★★★★★★★★★★★★
01F9:	2100FB	LD	HL, FB00H	; HL=关闭磁带机的命令。
01FC:	181B	JR	0219H	; 转到磁带机的驱动程序。
01FE:	7E	LD	A, (HL)	; 取输入字符串中的下一个代号。 ★★★★★★★★★
01FF:	D623	SUB	23H	; 判断是否为 # 符号。
0201:	3E00	LD	A, 00H	; 若没有指定为 # x 时, A=0 (设备号为 0)。
0203:	200D	JR	NZ, 0212H	; 若没有 # 号,则转移。
0205:	CD012B	CALL	2B01H	; 得到设备号,放在 DE 中(作为整数在"当前的区域"中)。
0208:	CF	RST	08H	; 寻找设备号后的逗号。
0209:	2C	INC	L	; 给出逗点的代码 2CH——','。
020A:	7B	LD	A, E	; 把设备代号——××。
020B:	A2	AND	D	; 变为相等的正数。
020C:	C602	ADD	A, 02H	;

地址	目的码	源程序		注释
020E:	D24A1E	JP	NC, 1E4AH	; 若 NC, 则产生 FC 错误。
0211:	3D	DEC	A	; A=设备号的正数值。
0212:	32E437	LD	(37E4H), A	; 规定设备的入口。★★★ 选择磁带机设备 ★★★★★
0215:	E5	PUSH	HL	; 保存当前代码字符串地址。
0216:	2104FF	LD	HL, FF04H	; 启动磁带机运转的代码。
0219:	CD2102	CALL	0221H	; 将马达打开或关闭。
021C:	E1	POP	HL	; 取回代码字符串的地址。
021D:	C9	RET		; 返回调用它的程序。
021E:	2100FF	LD	HL, FF00H	; 为了保护荧光屏控制器标志所设的掩码。
0221:	3A3D40	LD	A, (403DH)	; 得到荧光屏的控制位 (32 或 64 字符)。
0224:	A4	AND	H	; 和磁带器的控制位合并在一起。
0225:	B5	OR	L	;
0226:	D3FF	OUT	(FFH), A	; 把寄存器 A 的内容写到口 255 去(磁带和荧光屏的控制器)。
0228:	323D40	LD	(403DH), A	; 把新的数值作为当前的控制数值存放起来。
022B:	C9	RET		; 返回到调用它的程序。
022C:	3A3F3C	LD	A, (3C3FH)	; 在读磁带机时的闪烁'*'。 ★★★ 从荧光屏上取来有*的字节 ★★★★★★★★★★★★★★★★★★★★
022F:	EE0A	XOR	0AH	; 给出 2AH/20H/2AH…, 相当于 *,␣,*,␣,…。
0231:	323F3C	LD	(3C3FH), A	; 把新的显示数值存储回去。
0234:	C9	RET		; 返回到调用它的程序。
0235:	C5	PUSH	BC	; 从磁带机读的入口。★★★从磁带机上读入一个字节，然后返回 ★★★★★★★★★★★★★★★★★★★★
0236:	E5	PUSH	HL	; 保存调用它的程序的寄存器。
0237:	0608	LD	B, 08H	; B=读入的位数。
0239:	CD4102	CALL	0241H	; 读入一位,在 A 寄存器中合并成一个字节。
023C:	10FB	DJNZ	0239H	; 循环 8 次,直到读入一个字节为止。
023E:	E1	POP	HL	; } 恢复调用它的程序的寄存器。
023F:	C1	POP	BC	;
0240:	C9	RET		; 返回。
0241:	C5	PUSH	BC	; 从磁带机上读入一位。★★★ 调用 8 次便读入一个字节 ★★★★★★★★★★★★★★★★★★★★
0242:	F5	PUSH	AF	; 把调用程序的寄存器暂存起来。
0243:	DBFF	IN	A, (FFH)	; 磁带机开始运转,当出现第一个起动脉冲时便停止。
0245:	17	RLA		; 读入并测试是否时钟脉冲。
0246:	30FB	JR	NC, 0243H	; 如果不是,将继续循环,直到时钟脉冲出现。
0248:	0660	LD	B, 60H	; 在检测到起动脉冲后,延迟 476 毫秒。
024A:	10FE	DJNZ	024AH	;
024C:	CD1E02	CALL	021EH	; 复位 OUTSIG 触发器,于是我们便能读入数据脉冲。
024F:	0685	LD	B, 76H	; 然后在读入数据脉冲之前,
0251:	10FE	DJNZ	0251H	; 延迟 865 毫秒。

地址	目的码	源程序		注　释
0253:	DBFF	IN	A, (FFH)	; 读入数据脉冲。
0255:	47	LD	B, A	; 保存在 B 中。
0256:	F1	POP	AF	; A=这一字节的前几位。
0257:	CB10	RL	B	; 把数据移入进位位。
0259:	17	RLA		; 把这一数据位与前几位合并。
025A:	F5	PUSH	AF	; 把该字节暂存起来。
025B:	CD1E02	CALL	021EH	; 复位 OUTSIG 触发器。
025E:	F1	POP	AF	; 取回数据字节。
025F:	C1	POP	BC	; 取回暂存的寄存器。
0260:	C9	RET		; 返回。
0261:	CD6402	CALL	0264H	; 调用 264H 去写一个时钟脉冲。
0264:	E5	PUSH	HL	; 写一个字节的入口。以下条指令是把调用它程序的寄存器暂存起来。

说明 写一个字节是采用把一个字节的每一位串行的方法进行的。每一位的前面有一个时钟脉冲，若这一位是1，则其后跟着另一个时钟脉冲；若是0便没有时钟脉冲。从时钟脉冲到该位数据脉冲的时间近似1毫秒。

0265:	C5	PUSH	BC	;
0266:	D5	PUSH	DE	;
0267:	F5	PUSH	AF	;
0268:	0E08	LD	C, 08H	; C=要写入的位数。
026A:	57	LD	D, A	; D=要一位一位写入的数据字节。
026B:	CDD901	CALL	01D9H	; 写时钟位。
026E:	7A	LD	A, D	; 得到要写入的字节。
026F:	07	RLCA		; 若最高位是1，则进位位将置位，否则进位位是0。
0270:	57	LD	D, A	; 把移位后的数据字节保存起来。
0271:	300B	JR	NC, 027EH	; 若最高位是0，便转移去延迟1毫秒。
0273:	CDD901	CALL	01D9H	; 否则把一个是1的数据写入。
0276:	0D	DEC	C	; 对这个字节写过的位计数。
0277:	20F2	JR	NZ, 026BH	; 若没有写完一个字节，则写入另一个时钟脉冲，然后再测试下一数据位。
0279:	F1	POP	AF	; 恢复调用它的程序的寄存器 AF，
027A:	D1	POP	DE	; DE，
027B:	C1	POP	BC	; BC
027C:	E1	POP	HL	; 和 HL。
027D:	C9	RET		; 返回到调用它的程序。
027E:	0687	LD	B, 87H	; B=时间延迟计数。 ★★★★★★★★★★★★★★
0280:	10FE	DJNZ	0280H	; 延迟 $3.25\times10^{-6}\times135\times2.26=991$ 微秒。
0282:	18F2	JR	0276H	; 转回到计数写入位数的程序。
0284:	CDFE01	CALL	01FEH	; 取来设备号，并启动马达。 ★★★★★★★★★★★
0287:	06FF	LD	B, FFH	; 写引导码和同步字节的入口。
0289:	AF	XOR	A	; A=要写入的数据(都是0)。

地址	目的码	源程序		注　释
028A:	CD6402	CALL	0264H	; 写 255 个 0 字节。
028D:	10FB	DJNZ	028AH	; 写一个 0 字节计数减 1，直到写完 255 个字节。
028F:	3EA5	LD	A, A5H	; 紧接要写入的字节是 A5H。
0291:	18D1	JR	0264H	; 写入同步字节 A5H，然后返回到调用它的程序。
0293:	CDFE01	CALL	01FEH	; 取设备号，启动马达 ★★★★★★★★★★★★★★★
0296:	E5	PUSH	HL	; 寻找引导码和同步字节的入口。
0297:	AF	XOR	A	; 寄存器 A 和状态标志清零。
0298:	CD4102	CALL	0241H	; 从磁带机读入，直到找到 A5H。
029B:	FEA5	CP	A5H	; 在取得 A5H 之前，必须跳过 255 个 0 字节。
029D:	20F9	JR	NZ, 0298H	;
029F:	3E2A	LD	A, 2AH	; A = '*'的 ASCII 码。
02A1:	323E3C	LD	(3C3EH), A	; 在荧光屏上显示出'**'。
02A4:	323F3C	LD	(3C3FH), A	;
02A7:	E1	POP	HL	; 取回代码字符串的地址。
02A8:	C9	RET		; 返回到调用它的程序。
02A9:	CD1403	CALL	0314H	; 从磁带机上读入两个字节。★★★ 用于读入汇编程序 ★★★★★★★★★★★★★★★★★★★★★★★★★
02AC:	22DF40	LD	(40DFH), HL	; 把执行地址保存起来。
02AF:	CDF801	CALL	01F8H	; 关闭磁带机。
02B2:	CDE241	CALL	41E2H	; DOS 出口 (JP 5B51H)。
02B5:	318842	LD	SP, 4288H	; 在假设装入地址后设置 CSP。
02B8:	CDFE20	CALL	20FEH	; 显示 CR (回车)。
02BB:	3E2A	LD	A, 2AH	; A='*' 的 ASCII 码。
02BD:	CD2A03	CALL	032AH	; 显示*。
02C0:	CDB31B	CALL	1BB3H	; 等待从键盘输入，应是要装入的文件名称。
02C3:	DACC06	JP	C, 06CCH	; 若按了 BREAK 键，则转移。
02C6:	D7	RST	10H	; 检测输入字符串的下一个字符。
02C7:	CA9719	JP	Z, 1997H	; 若是 EOS，则 SN 错误。
02CA:	FE2F	CP	2FH	; 是'/'吗?
02CC:	284F	JR	Z, 031DH	; 若是'/'，便转移
02CE:	CD9302	CALL	0293H	; 启动磁带机，跳过引导码，直到找到 U，然后定位于第一个数据字节。
02D1:	CD3502	CALL	0235H	; 读入一个字节。
02D4:	FE55	CP	55H	; 测试是否为 'U' (U 的 ASCII 代码是 55H，即文件名称的引导码)。
02D6:	20F9	JR	NZ, 02D1H	; 循环，直到读入 'U' 的 ASCII 代码。
02D8:	0606	LD	B, 06H	; B=待匹配的字符数。
02DA:	7E	LD	A, (HL)	; 取键入的一个字符(在执行 02C0H 时键入的)。
02DB:	B7	OR	A	; 测试是否为 0。
02DC:	2809	JR	Z, 02E7H	; 是 0，即文件名称结束，开始装入程序。
02DE:	CD3502	CALL	0235H	; 否则，从磁带上读入一个字节。

地址	目的码	源程序		注　释
02E1:	BE	CP	(HL)	; 与键入的相比较。
02E2:	20ED	JR	NZ, 02D1H	; 若文件名称不相同,跳到磁带机上的下一个程序去。
02E4:	23	INC	HL	; 指向下一个键入的字符。
02E5:	10F3	DJNZ	02DAH	; 循环,直到六个字符都相同,或者键入的命令已结束。
02E7:	CD2C02	CALL	022CH	; 装入程序时,在荧光屏上闪烁*。
02EA:	CD3502	CALL	0235H	; 读入一个字节。
02ED:	FE78	CP	78H	; 现在判断读入的字节是否为 78H (78H是入口地址的引导码)。
02EF:	28B8	JR	Z, 02A9H	; 是,读入后两个字节并存入 40DFH, 等待输入。
02F1:	FE3C	CP	3CH	; 是否为 '<'("'<' 的代码是 3CH, 它是数据块的引导码)。
02F3:	20F5	JR	NZ, 02EAH	; 否,继续读入,直到找到 78H 或 3CH。
02F5:	CD3502	CALL	0235H	; 读入要装入的字节数。
02F8:	47	LD	B, A	; 要装入的字节数保存在计数器中。
02F9:	CD1403	CALL	0314H	; 读入后面两个字节(地址)放入 HL 中。
02FC:	85	ADD	A, L	; 从地址开始计算检查和。
02FD:	4F	LD	C, A	; 保存 8 位检查和。
02FE:	CD3502	CALL	0235H	; 读入一个字节。
0301:	77	LD	(HL), A	; 把它存储起来。
0302:	23	INC	HL	; 修改存储地址。
0303:	81	ADD	A, C	; 把数据字节算入检查和。
0304:	4F	LD	C, A	; 保存检查和。
0305:	10F7	DJNZ	02FEH	; 装入了一个字节。
0307:	CD3502	CALL	0235H	; 读入检查和。
030A:	B9	CP	C	; 把读入的检查和与计算出的检查和比较。
030B:	28DA	JR	Z, 02E7H	; 相等时,继续读入,并寻找 78H, 直到找到为止。
030D:	3E43	LD	A, 43H	; 检查和错误,显示一个 'C' 字母。
030F:	323E3C	LD	(3C3EH), A	; 把 'C' 显示在荧光屏上。
0312:	18D6	JR	02EAH	; 扫描,直到下一个程序开始。
0314:	CD3502	CALL	0235H	; 从磁带机上读入一个字节, 作为 LSB 存入 L 寄存器。
				★★★★★★★★★★★★★★★★★★★★★★★★★★★★
0317:	6F	LD	L, A	; 从磁带机上读入两个字节,合并成一个 16 位数。
0318:	CD3502	CALL	0235H	; 从磁带机上再读入一个字节。
031B:	67	LD	H, A	; 作为 MSB 存入 H 寄存器。
031C:	C9	RET		; 返回到调用它的程序。
031D:	EB	EX	DE, HL	; DE=输入的响应地址。★★★★★★★★★★★★★
031E:	2ADF40	LD	HL, (40DFH)	; 40DFH 存放执行地址。
0321:	EB	EX	DE, HL	; HL=输入地址, DE=执行地址。
0322:	D7	RST	10H	; 测试是否为 CR, 若不是 CR,
0323:	C45A1E	CALL	NZ, 1E5AH	; 则把 ASCII 码变成二进制数,结果留在 DE 中。
0326:	208A	JR	NZ, 02B2H	; 若发现不是数字,则转移。

地址	目的码	源程序		注　释
0328:	EB	EX	DE, HL	; 否则,数字便是执行地址。
0329:	E9	JP	(HL)	; 转移到1×××命令中给出的地址
032A:	C5	PUSH	BC	; 把 A 的内容输出到荧光屏、打印机或磁带机上。 ★★
032B:	4F	LD	C, A	; 保存输出的字符。
032C:	CDC141	CALL	41C1H	; 若没有 DOS, 则返回
032F:	3A9C40	LD	A, (409CH)	; 取设备类型代码。−1 是磁带机; 0 是显示器; +1 是行式打印机。
0332:	B7	OR	A	; 根据设备类型设置状态寄存器。
0333:	79	LD	A, C	; A=要写的字符。
0334:	C1	POP	BC	; 恢复调用它的程序的 BC 内容。
0335:	FA6402	JP	M, 0264H	; 写到磁带机上去。
0338:	2062	JR	NZ, 039CH	; 写到打印机上去。
033A:	D5	PUSH	DE	; 输出到荧光屏上去显示。
033B:	CD3300	CALL	0033H	;
033E:	F5	PUSH	AF	; 把要写的字符暂存起来。
033F:	CD4803	CALL	0348H	; 测试显示缓冲区是否满。
0342:	32A640	LD	(40A6H), A	; 更新光标位置 (0—3FH)。
0345:	F1	POP	AF	; 取回要写的字符。
0346:	D1	POP	DE	; 恢复调用它的程序的 DE 内容。
0347:	C9	RET		; 返回到调用它的程序。
0348:	3A3D40	LD	A, (403DH)	; 取荧光屏显示的控制字。 ★★★★★★★★★★★★
034B:	E608	AND	08H	; 测试每行是 32 个还是 64 个字符的显示格式。
034D:	3A2040	LD	A, (4020H)	; 取光标的地址。
0350:	2803	JR	Z, 0355H	; 若每行是 64 个字符,则转移。
0352:	0F	RRCA		; 迫使光标位置在 3C00H—3FFFH 之间。
0353:	E61F	AND	1FH	;
0355:	E63F	AND	3FH	;
0357:	C9	RET		; 返回到调用它的程序。
0358:	CDC441	CALL	41C4H	; DOS 出口 (JP 59CDH)。 ★★★★★★★★★★★★
035B:	D5	PUSH	DE	; 暂存调用它的程序的 DE 内容。
035C:	CD2B00	CALL	002BH	; 扫描键盘。
035F:	D1	POP	DE	; 取回 DE。
0360:	C9	RET		; 返回到调用它的程序。
0361:	AF	XOR	A	; 键盘输入子程序。 ★★★★★★★★★★★★★★★
0362:	329940	LD	(4099H), A	; 清除 BREAK 后键入的字符。
0365:	32A640	LD	(40A6H), A	; 并清除当前光标位置。
0368:	CDAF41	CALL	41AFH	; DOS 出口 (JP 598EH)。
036B:	C5	PUSH	BC	; 保存 BC。
036C:	2AA740	LD	HL, (40A7H)	; (通常)缓冲区=41E8H。
036F:	06F0	LD	B, F0H	; 缓冲区长度 =240 个字节。
0371:	CDD905	CALL	05D9H	; 观察在缓冲区内键入的是什么。

地址	目的码	源程序		注释
0374:	F5	PUSH	AF	; 保存标志位。
0375:	48	LD	C, B	; C=输入的长度。
0376:	0600	LD	B, 00H	; BC=输入的长度。
0378:	09	ADD	HL, BC	; HL=输入缓冲区的结束指针。
0379:	3600	LD	(HL), 00H	; 以 00 表示输入结束的标志。
037B:	2AA740	LD	HL, (40A7H)	; HL=输入缓冲区的指针。
037E:	F1	POP	AF	; 取回状态标志。
037F:	C1	POP	BC	; 取回 BC。
0380:	2B	DEC	HL	; HL=输入缓冲区指针 −1 (RST 16H 程序所要求的)。
0381:	D8	RET	C	; 若按了 BREAK 键,则进位标志置位,返回。
0382:	AF	XOR	A	; 否则清除所有的状态标志。
0383:	C9	RET		; 返回,此时 HL=输入缓冲区 −1。
0384:	CD5803	CALL	0358H	; 扫描键盘。 ★★★★★★★★★★★★★★★★★★★★
0387:	B7	OR	A	; 测试有否按了任何一个键。
0388:	C0	RET	NZ	; 若有一个键被按了,返回。
0389:	18F9	JR	0384H	; 否则循环,直到按了键为止。
038B:	AF	XOR	A	; A 清零。 ★★★★★★★★★★★★★★★★★★★★
038C:	329C40	LD	(409CH), A	; 然后置输入设备=显示器。
038F:	3A9B40	LD	A, (409BH)	; 取打印机托架的位置。
0392:	B7	OR	A	; 置状态标志。
0393:	C8	RET	Z	; 若打印机缓冲区空,返回。
0394:	3E0D	LD	A, 0DH	; 装入打印字符(回车)。
0396:	D5	PUSH	DE	; 把调用它的程序的 DE 暂存。
0397:	CD9C03	CALL	039CH	; 调用打印机驱动程序。
039A:	D1	POP	DE	; 取回 DE。
039B:	C9	RET		; 返回到调用它的程序。
039C:	F5	PUSH	AF	; 把调用它的程序的 AF 暂存。 ★★★★★★★★★★

说明 调用打印机驱动程序的入口,要打印的字符在 A 寄存器中。若 A='C', 跳一行,把当前的字符计数器清零。若 A='A' 或 'D', 打印回车,并对行字符计数器清零。

039D:	D5	PUSH	DE	; 把调用它的程序的 DE 和 BC 暂存。
039E:	C5	PUSH	BC	;
039F:	4F	LD	C, A	; C=要打印的字符。
03A0:	1E00	LD	E, 00H	; E=新的一行的字符计数器。测试打印的是否为 'C', 'D' 或 'A'。
03A2:	FE0C	CP	0CH	; 测试是否跳一行。
03A4:	2810	JR	Z, 03B6H	; 若要跳一行,则转移。
03A6:	FE0A	CP	0AH	; 测试是否要换行 (0AH)。
03A8:	2003	JR	NZ, 03ADH	; 不换行,测试是否 'D'——回车。
03AA:	3E0D	LD	A, 0DH	; 置下一个字符为打印机回车。
03AC:	4F	LD	C, A	; 暂存打印机的回车符。

地址	目的码	源程序		注　释
03AD:	FE0D	CP	0DH	; 测试第二种回车符。
03AF:	2805	JR	Z, 03B6H	; 若是 'A' 或 'D'——回车,则转移。
03B1:	3A9B40	LD	A, (409BH),	; 取本行的字符计数。
03B4:	3C	INC	A	; 计数加 1, 以输出下一个字符。
03B5:	5F	LD	E, A	; 把计数移到 E 寄存器中,
03B6:	7B	LD	A,E	; 因而我们可以使用通用的程序。
03B7:	329B40	LD	(409BH), A	; 将更新后的本行字符计数存储起来。
03BA:	79	LD	A, C	; 把要打印的字符放入 A 寄存器
03BB:	CD3B00	CALL	003BH	; 调用行印机驱动程序。
03BE:	C1	POP	BC	; 恢复调用它的程序的 BC。
03BF:	D1	POP	DE	; DE
03C0:	F1	POP	AF	; 和 AF 寄存器。
03C1:	C9	RET		; 返回到调用它的程序。
03C2:	E5	PUSH	HL	; 驱动程序入口子程序。★★★ RST 14H, 1CH, 24H 都进入本程序 ★★★★★★★★★★★★★★★★★★★
03C3:	DDE5	PUSH	IX	; 暂存 IX 寄存器
03C5:	D5	PUSH	DE	; B=入口代码, DE=DCB 地址。
03C6:	DDE1	POP	IX	; 把 DCB 地址装入 IX 中。
03C8:	D5	PUSH	DE	; 暂存 DE 的原始内容。
03C9:	21DD03	LD	HL, 03DDH	; HL 是返回的地址。
03CC:	E5	PUSH	HL	; 把返回地址推入堆栈。
03CD:	4F	LD	C, A	; 把送到外设去的字符存起来。
03CE:	1A	LD	A, (DE)	; 从 DCB 中取来第一个字。
03CF:	A0	AND	B	; 分离出设备代码位。
03D0:	B8	CP	B	; 把该数值与入口代码 (B) 比较。
03D1:	C23340	JP	NZ, 4033H	; 若不相等,则经 DOS 出口转移,到设备驱动程序。
03D4:	FE02	CP	02H	; 清除状态标志位。
03D6:	DD6E01	LD	L, (IX+01H)	; HL=从 DCB 中取来的驱动程序地址。
03D9:	DD6602	LD	H, (IX+02H)	; 取驱动程序地址的 MSB。
03DC:	E9	JP	(HL)	; 转移到驱动程序。
03DD:	D1	POP	DE	; 从驱动程序返回,
03DE:	DDE1	POP	IX	; 恢复 IX, HL 和 BC。
03E0:	E1	POP	HL	;
03E1:	C1	POP	BC	;
03E2:	C9	RET		; 返回到调用它的程序。
03E3:	213640	LD	HL, 4036H	; 键盘驱动程序。★★★ HL=键盘工作区指针 ★★★
03E6:	010138	LD	BC, 3801H	; BC=行的指针。
03E9:	1600	LD	D, 00H	; D=列的指针。
03EB:	0A	LD	A, (BC)	; 取第 N 行,
03EC:	5F	LD	E, A	; 共有 8 列对应的位,
03ED:	AE	XOR	(HL)	; 与前一个键入的值异或。

地址	目的码	源程序		注　释
03EE:	73	LD	(HL), E	; 把列的位存储在缓冲区中。
03EF:	A3	AND	E	; 测试哪一行按了键。
03F0:	2008	JR	NZ, 03FAH	; 若是按了第 N 行中的一个键,则转移。
03F2:	14	INC	D	; 改变列指针。
03F3:	2C	INC	L	; 共有七个字节的缓冲区用于行的检索。
03F4:	CB01	RLC	C	; 把行地址从 3801H 到 3840H 一步一步地改变。
03F6:	F2EB03	JP	P, 03EBH	; 测试下一行。
03F9:	C9	RET		; 没有按任何键,返回。
03FA:	5F	LD	E, A	; 暂存列的位。 ★★★★★★★★★★★★★★★★★★★★
03FB:	7A	LD	A, D	; 行指针在 0—7。
03FC:	07	RLCA		; 行 ×2,
03FD:	07	RLCA		; 行 ×4,
03FE:	07	RLCA		; 行 ×8,
03FF:	57	LD	D, A	; 存入 D 中。
0400:	0E01	LD	C, 01H	; 从第 0 位开始,
0402:	79	LD	A, C	; 屏蔽不用的位,选出合适的位。
0403:	A3	AND	E	; 寻找非零的列。
0404:	2005	JR	NZ, 040BH	; 若找到,便转移。
0406:	14	INC	D	; 列数加 1。
0407:	CB01	RLC	C	; 调整掩码。
0409:	18F7	JR	0402H	; 再寻找。
040B:	3A8038	LD	A, (3880H)	; 取 |SHIFT| 位,
040E:	47	LD	B, A	; 把 |SHIFT| 位存在 B 中。
040F:	7A	LD	A, D	; 行×8+列数(在 0—7 之间)。
0410:	C640	ADD	A, 40H	; 行×8+列数 +64,
0412:	FE60	CP	60H	; 测试前四行 (@, A—Z)。
0414:	3013	JR	NC, 0429H	; 若是后三行,即数字或特殊字符,就转移。
0416:	CB08	RRC	B	; 将 |SHIFT| 位右移到进位位。
0418:	3031	JR	NC, 044BH	; 若没有 |SHIFT| ,则转移。
041A:	C620	ADD	A, 20H	; 给出小写字母。
041C:	57	LD	D, A	; 调整后的字符。
041D:	3A4038	LD	A, (3840H)	; 取行 6 中的列数,
0420:	E610	AND	10H	; 寻找向下箭头或 CR (回车)
0422:	2828	JR	Z, 044CH	; 若找不到向下箭头或 CR, 转移。
0424:	7A	LD	A, D	; 又取入调整后的键入值。
0425:	D660	SUB	60H	; 调整为 ASCII 的 CR。
0427:	1822	JR	044BH	; 转移去返回。
0429:	D670	SUB	70H	; 测试最后一行 (|ENTER| — |SPACE|)。
042B:	3010	JR	NC, 043DH	; 若是最后一行,则转移。
042D:	C640	ADD	A, 40H	; 重新调整行 4, 5。

地址	目的码	源程序		注　释
042F:	FE3C	CP	3CH	; 把行 4, 5 进行转换。
0431:	3802	JR	C, 0435H	; 若是按了 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, :, ;, , 等键中的一个,则转移。
0433:	EE10	XOR	10H	; 将行 5 的各位取反。
0435:	CB08	RRC	B	; 判断是否按了 |SHIFT| 。
0437:	3012	JR	NC, 044BH	; 没有按,就转移。
0439:	EE10	XOR	10H	; 然后,再将行 5 的各位取反。
043B:	180E	JR	044BH	; 转到出口处。
043D:	07	RLCA		; 现在是(行×8+列 −48)×2。
043E:	CB08	RRC	B	; 测试有否按 |SHIFT| 。
0440:	3001	JR	NC, 0443H	; 若没按,则转移。
0442:	3C	INC	A	; 现在变为(行×8+列−48)×2+5=列×2+1。
0443:	215000	LD	HL, 0050H	; 给出最后一行的代码表地址。
0446:	4F	LD	C, A	; 由 43DH 或 442H 处来的数值(取决于有否按 |SHIFT|),放到 C 中。
0447:	0600	LD	B, 00H	; 令 B=0。
0449:	09	ADD	HL, BC	; 得到表的变址数,
044A:	7E	LD	A, (HL)	; 取得 ASCII 的代码。
044B:	57	LD	D, A	; 保存该字符。
044C:	01AC0D	LD	BC, 0DACH	; 装入延迟量。
044F:	CD6000	CALL	0060H	; 延迟 20 毫秒。
0452:	7A	LD	A, D	; A=ASCII 的字符。
0453:	FE01	CP	01H	; 是 |BREAK| 吗?
0455:	C0	RET	NZ	; 否,返回。
0456:	EF	RST	28H	; 是 |BREAK| ,
0457:	C9	RET		; 返回。
0458:	DD6E03	LD	L, (IX+03H)	; ★★★ 显示驱动程序★★★★★★★★★★★★★★ HL=光标位置的指针。 装入当前显示器缓冲区地址的 LSB。
045B:	DD6604	LD	H, (IX+04H)	; 装入当前显示器缓冲区地址的 MSB。
045E:	383A	JR	C, 049AH	; 若得到所需的最后一个字符是回车,则转移。
0460:	DD7E05	LD	A, (IX+05H)	; 得到光标开或关的标志。
0463:	B7	OR	A	; 置光标开或关的状态标志。
0464:	2801	JR	Z, 0467H	; 若光标关闭,则转移。
0466:	77	LD	(HL), A	; 把被光标覆盖的字符移到字符缓冲区中。
0467:	79	LD	A, C	; 取来要显示的字符。
0468:	FE20	CP	20H	; 是否为空格?
046A:	DA0605	JP	C, 0506H	; 若是控制字符,则转移。
046D:	FE80	CP	80H	; 测试是否为图示字符或压缩代码。
046F:	3035	JR	NC, 04A6H	; 若是图示字符或空格的压缩代码,则转移。
0471:	FE40	CP	40H	; 把该字符与字母@比较。

地址	目的码	源程序		注释
0473:	3808	JR	C, 047DH	; 若不是字母@—Z，则转移。
0475:	D640	SUB	40H	; 为求出字母，减@值，得到 0—26。
0477:	FE20	CP	20H	; 测试是否小写字母。
0479:	3802	JR	C, 047DH	; 若不是小写字母，则转移。
047B:	D620	SUB	20H	; 把小写字母变成大写字母。
047D:	CD4105	CALL	0541H	; 把一个新的字符显示在荧光屏上，若需要，可把整个图象往上移。
0480:	7C	LD	A, H	; 使下一个字符的地址一定在 3C00H≤x≤3FFFH 的范围中。
0481:	E603	AND	03H	;
0483:	F63C	OR	3CH	; 使缓冲区地址的 MSB 在 3C—3FH 内。
0485:	67	LD	H, A	; 把更新的缓冲区地址 MSB 放入 HL 中。
0486:	56	LD	D, (HL)	; 取得在光标位置处的字符值。
0487:	DD7E05	LD	A, (IX+05H)	; 得到光标开或关的状态。
048A:	B7	OR	A	; 置光标状态标志。
048B:	2805	JR	Z, 0492H	; 若光标关闭转移。
048D:	DD7205	LD	(IX+05H), D	; 否则，存储将被光标代替的字符。
0490:	365F	LD	(HL), 5FH	; 把光标(—)移到下一个字符的位置上。
0492:	DD7503	LD	(IX+03H), L	; 把下一个字符位置的地址存放在荧光屏的 DCB 中第 3，4 字节内。
0495:	DD7404	LD	(IX+04H), H	;
0498:	79	LD	A, C	; 取回最新显示的字符。
0499:	C9	RET		; 返回到调用它的程序。
049A:	DD7E05	LD	A, (IX+05H)	; 取光标开或关状态，送还由光标替代的当前字符或下一个字符。
049D:	B7	OR	A	; 置该状态标志。
049E:	C0	RET	NZ	; 若光标开着，则带着字符返回，
049F:	7E	LD	A, (HL)	; 否则取上一次显示的字符，放在 A 寄存器中返回。
04A0:	C9	RET		;
04A1:	7D	LD	A, L	; 取当前显示器缓冲区地址的 LSB。★★★★★★★★ 假设为每行 64 个字符，使显示器缓冲区指针退回到当前行的开始。
04A2:	E6C0	AND	C0H	; 把最低 6 位去掉，在每行 64 个字符时，
04A4:	6F	LD	L, A	; 得到 ××00H, ××40H, ××80H 和 ××C0H。
04A5:	C9	RET		; 把新的显示器缓冲地址放在 HL 中返回。
04A6:	FEC0	CP	C0H	; 检查空格的压缩代码。★★★★★★★★★★★★★
04A8:	38D3	JR	C, 047DH	; 图示符号，转移。
04AA:	D6C0	SUB	C0H	; 减去压缩代码中的变换量。
04AC:	28D2	JR	Z, 0480H	; 若显示空格数为 0，则转移。
04AE:	47	LD	B, A	; B=要显示的空格数。
04AF:	3E20	LD	A, 20H	; A=空格的代码。

地址	目的码	源程序		注　释
04B1:	CD4105	CALL	0541H	; 显示一个空格。
04B4:	10F9	DJNZ	04AFH	; 循环,直到显示了 B 个空格。
04B6:	18C8	JR	0480H	; 更新显示缓冲区指针,并返回。
04B8:	7E	LD	A, (HL)	; ★★★取入当前位置处的字符,来打开或关闭光标(控制字符的处理)。作为光标状态★★★★★★★★★★
04B9:	DD7705	LD	(IX+05H), A	; 把光标开或关状态标志放在 BCD 中。
04BC:	C9	RET		; 返回到调用它的程序。
04BD:	AF	XOR	A	; 置光标状态为关。
04BE:	18F9	JR	04B9H	; 更新显示器的 DCB, 返回。
04C0:	21003C	LD	HL, 3C00H	; HL=显示区的首地址。★★★光标复原★★★★★★
04C3:	3A3D40	LD	A, (403DH)	; 令每行 64 个字符。
04C6:	E6F7	AND	F7H	; 消除命令字中每行 32 个字符的那一位。
04C8:	323D40	LD	(403DH), A	; 存储命令字。
04CB:	D3FF	OUT	(FFH), A	; 把命令字送到显示器控制器中。
04CD:	C9	RET		; 返回到调用它的程序。
04CE:	2B	DEC	HL	; 光标左移一个字符。★★★光标左移一个字符(控制字符的处理)★★★★★★★★★★★★★★★★★★★★
04CF:	3A3D40	LD	A, (403DH)	; 取显示器的控制字。
04D2:	E608	AND	08H	; 测试每行 32 个或 64 个字符。
04D4:	2801	JR	Z, 04D7H	; 若是 64 字符/每行,则转移。
04D6:	2B	DEC	HL	; 若是 32 字符/每行,则再左移一个字符。
04D7:	3620	LD	(HL), 20H	; 用空格代替上一个字符。
04D9:	C9	RET		; 返回到调用它的程序。
04DA:	3A3D40	LD	A, (403DH)	; 取显示器的控制字。★★★向左箭头,光标左移一个字符(控制字符的处理) ★★★★★★★★★★★★★★
04DD:	E608	AND	08H	; 分离出每行的字符数。
04DF:	C4E204	CALL	NZ, 04E2H	; 若每行 32 个字符,则使光标左移两次。
04E2:	7D	LD	A, L	; 把现在光标位置的 LSB 保存起来。
04E3:	E63F	AND	3FH	; 判断光标的 LSB 是否退到上一行。
04E5:	2B	DEC	HL	; 然后光标左移一个字符。
04E6:	C0	RET	NZ	; 若光标在同一行中,则返回。
04E7:	114000	LD	DE, 0040H	; 用当前光标位置加 64 的方法,
04EA:	19	ADD	HL, DE	; 使光标往下跳一行。
04EB:	C9	RET		; 然后返回到调用它的程序。
04EC:	23	INC	HL	; 当前的光标地址加 1。★★★向右箭头,光标向右移一个区(控制字符的处理) ★★★★★★★★★★★★★★
04ED:	7D	LD	A, L	; 取地址的 LSB。
04EE:	E63F	AND	3FH	; 测试是否溢出到下一行。
04F0:	C0	RET	NZ	; 不溢出,返回到调用它的程序。
04F1:	11C0FF	LD	DE, FFC0H	; 向上移一行,把当前光标地址加-64。
04F4:	19	ADD	HL, DE	;

地址	目的码	源程序		注　释
04F5:	C9	RET		; 返回到调用它的程序。
04F6:	3A3D40	LD	A, (403DH)	; 取显示器控制字。★★★★★★★★★★★★★★★★
04F9:	F608	OR	08H	; 打开每行 32 个字符的方式。
04FB:	323D40	LD	(403DH), A	; 送回显示器的控制字。
04FE:	D3FF	OUT	(FFH), A	; 选择每行 32 个字符的方式。
0500:	23	INC	HL	; 在显示缓冲区中当前位置的地址加 1。
0501:	7D	LD	A, L	; 在每行 32 个字符方式中，
0502:	E6FE	AND	FEH	; 使地址的 LSB 一定是偶数值。
0504:	6F	LD	L, A	; 把更新的行地址放入 HL 中。
0505:	C9	RET		; 返回到调用它的程序。
0506:	118004	LD	DE, 0480H	; ★★★ 显示器控制字符（字符代码小于 20H）的处理★★★★★★★★★★处理控制字后，送回地址。
0509:	D5	PUSH	DE	; 放入堆栈。
050A:	FE08	CP	08H	; 测试是否左移一格，并清除上一个字符（08H 是退格控制码）。
050C:	28C0	JR	Z, 04CEH	; 若是左移一格，则转移。
050E:	FE0A	CP	0AH	; 不是，再测试是否为 0AH（0AH 是换行控制码）。
0510:	D8	RET	C	; 除 08H 外，控制代码小于 0AH 时不予处理。
0511:	FE0E	CP	0EH	; 测试是否开光标（0EH 是开光标控制码）。
0513:	384F	JR	C, 0564H	; 若在 0AH—0DH（回车）之间，则转移。
0515:	28A1	JR	Z, 04B8H	; 若是开光标，则转移。
0517:	FE0F	CP	0FH	; 测试是否关光标（0FH 是关光标控制码）。
0519:	28A2	JR	Z, 04BDH	; 若是关光标，则转移。
051B:	FE17	CP	17H	; 测试是否选择每行 32 个字符（17H 是选择 32 字符/行）。
051D:	28D7	JR	Z, 04F6H	; 若是选择每行 32 个字符，则转移。
051F:	FE18	CP	18H	; 是否向左箭头（18H, ← 控制码）。
0521:	28B7	JR	Z, 04DAH	; 若是向左箭头，则转移。
0523:	FE19	CP	19H	; 是否向右箭头（19H, → 控制码）。
0525:	28C5	JR	Z, 04ECH	; 若是向右箭头，则转移。
0527:	FE1A	CP	1AH	; 是否向下箭头（1AH, ↓ 控制码）。
0529:	28BC	JR	Z, 04E7H	; 是，转移。
052B:	FE1B	CP	1BH	; 是否向上箭头（1BH, ↑ 控制码）。
052D:	28C2	JR	Z, 04F1H	; 是，转移。
052F:	FE1C	CP	1CH	; 是否光标复原控制码（1CH, 光标复原控制码）。
0531:	288D	JR	Z, 04C0H	; 是，转移。
0533:	FE1D	CP	1DH	; 是否向左移到一行的开始位置（1DH, 左移到行开头的控制码）。
0535:	CAA104	JP	Z, 04A1H	; 若是左移到行开头，则转移。
0538:	FE1E	CP	1EH	; 是否把这一行光标右边的显示部分都消除（1EH, 删除右部控制码）。

地址	目的码	源程序		注　释
053A:	2837	JR	Z, 0573H	; 是,转移。
053C:	FE1F	CP	1FH	; 是否清除光标以下的荧光屏显示内容（1FH，清除屏幕上显示的余下部分)。
053E:	283C	JR	Z, 057CH	; 是,转移。
0540:	C9	RET		; 对其它字符不予处理。
0541:	77	LD	(HL), A	; 把一个新的字符送到显示器的缓冲区中。 ★★★★★
0542:	23	INC	HL	; 把缓冲区的地址加 1。
0543:	3A3D40	LD	A, (403DH)	; 得到显示器的状态字。
0546:	E608	AND	08H	; 取得字符/行的标志。
0548:	2801	JR	Z, 054BH	; 若是每行 64 个字符,则转移。
054A:	23	INC	HL	; 若每行 32 个字符,则地址还要加 1，并得到下个显示地址。
054B:	7C	LD	A, H	; 测试是否到了显示缓冲区的终地址。
054C:	FE40	CP	40H	; 若下一个显示地址的 MSB=40H，那末达到了终地址。
054E:	C0	RET	NZ	; 若没有达到,则返回。
054F:	11C0FF	LD	DE, FFC0H	; DE=－64
0552:	19	ADD	HL, DE	; 把显示器的内存指针退回一行,准备将荧光屏的显示卷动上移。
0553:	E5	PUSH	HL	; 把最底部一行的首地址暂存。
0554:	11003C	LD	DE, 3C00H	; DE=第一行地址。
0557:	21403C	LD	HL, 3C40H	; HL=第二行地址。
055A:	C5	PUSH	BC	; 暂存 BC。
055B:	01C003	LD	BC, 03C0H	; BC=要移动的字符数(等于 15 行的字符)。
055E:	EDB0	LDIR		; 把荧光屏上显示的内容向上移一行。
0560:	C1	POP	BC	; 取回 BC。
0561:	EB	EX	DE, HL	; HL=第 16 行(最后一行)的地址。
0562:	1819	JR	057DH	; 对第 16 行都填空格。
0564:	7D	LD	A, L	; 取当前字符地址的 LSB。
0565:	E6C0	AND	C0H	; 使它成为当前行的首地址(控制字符的处理)。
0567:	6F	LD	L, A	;
0568:	E5	PUSH	HL	; 把当前字符行的首地址暂存。
0569:	114000	LD	DE, 0040H	; DE=一行的字符数。
056C:	19	ADD	HL, DE	; 给出下一行的首地址。
056D:	7C	LD	A, H	; 测试下一行的地址的 MSB。
056E:	FE40	CP	40H	; 是否已达到荧光屏的终地址。
0570:	28E2	JR	Z, 0554H	; 是,转移(使屏幕显示往上推一行)。
0572:	D1	POP	DE	; 取回当前行的首地址。
0573:	E5	PUSH	HL	; 删到行的末尾。 HL=下一行的首地址。
0574:	54	LD	D, H	; 计算末地址,
0575:	7D	LD	A, L	; 以便用于下面在行内填空格的程序中。
0576:	F63F	OR	3FH	; 取放在 HL 中的地址。

地址	目的码	源程序		注　释
0578:	5F	LD	E, A	; 使它恰好成为到下一行的字符数目,
0579:	13	INC	DE	; 放到 DE 中。
057A:	1804	JR	0580H	; 然后转到对一行填空格的程序。
057C:	E5	PUSH	HL	; 把荧光屏上光标以后的显示清除。
057D:	110040	LD	DE, 4000H	; 测试地址,用于循环检查是否达到终地址。
0580:	3620	LD	(HL), 20H	; 把一个空格放到该行中当前字符的位置上。
0582:	23	INC	HL	; 当前地址加 1。
0583:	7C	LD	A, H	; 将当前地址的 MSB 与 D 的内容(终地址的 MSB=40H) 比较。
0584:	BA	CP	D	; 测试是否达到终地址的 MSB。
0585:	20F9	JR	NZ, 0580H	; 否,循环。
0587:	7D	LD	A, L	; 然后比较地址的 LSB。
0588:	BB	CP	E	;
0589:	20F5	JR	NZ, 0580H	; 若不是终地址的 LSB, 则循环。
058B:	E1	POP	HL	; 取回 HL (当前字符位置的地址)。
058C:	C9	RET		; 返回到调用它的程序。
058D:	79	LD	A, C	; 取一个要打印的字符。★★★ 打印机驱动程序 ★★★

说明 行打印机托架的控制代码, 0AH = 回车+换行; 0BH = 换页; 0CH = 有条件的换页; 0DH = 回车

058E:	B7	OR	A	; 置状态标志。
058F:	2840	JR	Z, 05D1H	; 若是零,则取得打印机的状态并返回。
0591:	FE0B	CP	0BH	; 是否换页代码?
0593:	280A	JR	Z, 059FH	; 是,进行换行,一直达到下一页的开头。
0595:	FE0C	CP	0CH	; 判断是否为条件换页。
0597:	201B	JR	NZ, 05B4H	; 若是数据字符,则转移。
0599:	AF	XOR	A	; 清除 A 寄存器(打印一个空字符)。
059A:	DDB603	OR	(IX+03H)	; 得到每页的行数。
059D:	2815	JR	Z, 05B4H	; 若是零,则不要换页。
059F:	DD7E03	LD	A, (IX+03H)	; 取每页的行数。
05A2:	DD9604	SUB	A, (IX+04H)	; 减去本页已打印的行数。
05A5:	47	LD	B, A	; 给出将要换多少行,方可达到下一页的开头。该数放在 B 中。
05A6:	CDD105	CALL	05D1H	; 取得打印机的状态。
05A9:	20FB	JR	NZ, 05A6H	; 若打印机处于忙的状态,便循环。
05AB:	3E0A	LD	A, 0AH	; 取换行的字符。
05AD:	32E837	LD	(37E8H), A	; 输出给打印机。
05B0:	10F4	DJNZ	05A6H	; 循环,直到换了一页。
05B2:	1818	JR	05CCH	; 清除新页的行计数器,并返回。
05B4:	F5	PUSH	AF	; 暂存要打印的字符。
05B5:	CDD105	CALL	05D1H	; 取打印机状态。
05B8:	20FB	JR	NZ, 05B5H	; 若打印机忙,则循环。

地址	目的码	源程序		注　释
05BA:	F1	POP	AF	; 取回要打印的字符。
05BB:	32E837	LD	(37E8H), A	; 把它输出给打印机。
05BE:	FE0D	CP	0DH	; 是否为回车?
05C0:	C0	RET	NZ	; 若是数据字符,则返回到调用它的程序。
05C1:	DD3404	INC	(IX+04H)	; 本页打印的行计数加 1。
05C4:	DD7E04	LD	A, (IX+04H)	; 取本页的行计数值。
05C7:	DDBE03	CP	(IX+03H)	; 与每页的行数比较。
05CA:	79	LD	A, C	; 把要打印的字符重新放回到 A 中(是一个回车字符)。
05CB:	C0	RET	NZ	; 若一页没有打满,则返回。
05CC:	DD360400	LD	(IX+04H), 00H	; 一页打满,清除下一页的行计数器。
05D0:	C9	RET		; 返回到调用它的程序。
05D1:	3AE837	LD	A, (37E8H)	; 取得打印机的状态字。 ★★★★★★★★★★★★★
05D4:	E6F0	AND	F0H	; 分离出状态。
05D6:	FE30	CP	30H	; 测试是否联接了打印机以及是否准备好了。
05D8:	C9	RET		; 若已联接好,并准备好,状态为零。返回。
05D9:	E5	PUSH	HL	; HL 指向输入缓冲区。★★★ 接收键盘的输入 ★★★

说明　HL = 缓冲区的首地址,B = 缓冲区的大小,若按 |BREAK| 键,则返回,此时C标志置位。

05DA:	3E0E	LD	A, 0EH	; 取打开光标代码。
05DC:	CD3300	CALL	0033H	; 打开光标。
05DF:	48	LD	C, B	; C=缓冲区大小。
05E0:	CD4900	CALL	0049H	; 按一个键后返回到这里。
05E3:	FE20	CP	20H	; 测试是否为空格。
05E5:	3025	JR	NC, 060CH	; 若 NC, 则不是空格,但如果是一个可显示的字符,便转移。
05E7:	FE0D	CP	0DH	; 测试是否为回车。
05E9:	CA6206	JP	Z, 0662H	; 是,转移。
05EC:	FE1F	CP	1FH	; 测试是否为 |CLEAR|。
05EE:	2829	JR	Z, 0619H	; 是,转移。
05F0:	FE01	CP	01H	; 测试是否为 |BREAK|。
05F2:	286D	JR	Z, 0661H	; 是,转移。
05F4:	11E005	LD	DE, 05E0H	; 因为以下都不是可显示的字符了,
05F7:	D5	PUSH	DE	; 把返回地址 05E0H 放入堆栈,
05F8:	FE08	CP	08H	; 测试是否为向左移一格,删除字符。
05FA:	2834	JR	Z, 0630H	; 是,转移。
05FC:	FE18	CP	18H	; 测试是否为 |←|。
05FE:	282B	JR	Z, 062BH	; 是,转移。
0600:	FE09	CP	09H	; 测试是否为 TAB 字符。
0602:	2842	JR	Z, 0646H	; 是,转移。
0604:	FE19	CP	19H	; 测试是否选择每行 32 个字符。
0606:	2839	JR	Z, 0641H	; 是,转移。
0608:	FE0A	CP	0AH	; 测试是否为换行。

地址	目的码	源程序		注　释
060A:	C0	RET	NZ	; 否,回到 05E0H。
060B:	D1	POP	DE	; 取出返回地址 05E0H。
060C:	77	LD	(HL), A	; 键入的是可显示的字符,将它存起来。
060D:	78	LD	A, B	; 取来字符计数,即 240 减已取的字符数。
060E:	B7	OR	A	; 置标志位。
060F:	28CF	JR	Z, 05E0H	; 若已达缓冲区底,则不处理键入的字符,除非 BREAK 或 CR (回车)。
0611:	7E	LD	A, (HL)	; 取回刚键入的字符。
0612:	23	INC	HL	; 缓冲区的地址加 1。
0613:	CD3300	CALL	0033H	; 显示刚接收的字符。
0616:	05	DEC	B	; 计数减 1。
0617:	18C7	JR	05E0H	; 取下一个字符。
0619:	CDC901	CALL	01C9H	; 若按了 CLEAR 或 CLS，则清除荧光屏上显示的内容。
061C:	41	LD	B, C	; 重新设置传输字符的计数。
061D:	E1	POP	HL	; 重新设置缓冲区的地址。
061E:	E5	PUSH	HL	; 把缓冲区的首地址放入堆栈。
061F:	C3E005	JP	05E0H	; 取下一个字符(缓冲区的第一个字符)。
0622:	CD3006	CALL	0630H	; 等待下一个字符的键入。
0625:	2B	DEC	HL	; 退回到前一个字符 (CR 之前的一个)。
0626:	7E	LD	A, (HL)	; 将它取来并判断是否为换行。
0627:	23	INC	HL	; 恢复缓冲区的地址为下一个单元。
0628:	FE0A	CP	0AH	; 前一个字符是否为换行?
062A:	C8	RET	Z	; 是,返回。
062B:	78	LD	A, B	; 不是,测试缓冲区是否满。A=接收到的字符个数。
062C:	B9	CP	C	; 接收到的字符个数减缓冲区大小。
062D:	20F3	JR	NZ, 0622H	; 若还有空余单元可放数据,则循环。
062F:	C9	RET		; 若缓冲区满,则返回。
0630:	78	LD	A, B	; B=接收到的字符数，C=缓冲区的大小。 ★★★★★
0631:	B9	CP	C	; 测试缓冲区满否。
0632:	C8	RET	Z	; 是,返回。
0633:	2B	DEC	HL	; 退回到前一个字符。
0634:	7E	LD	A, (HL)	; 把它取出。
0635:	FE0A	CP	0AH	; 测试是否为换行。
0637:	23	INC	HL	; 指向刚接收到的字符。
0638:	C8	RET	Z	; 若前一个字符是换行,则返回。
0639:	2B	DEC	HL	; 再回到缓冲区前一个字符。
063A:	3E08	LD	A, 08H	; 发出左移前一个字符的命令。
063C:	CD3300	CALL	0033H	; 左移前一个字符。
063F:	04	INC	B	; 调整字符接收计数。
0640:	C9	RET		; 返回。

地址	目的码	源程序		注释
0641:	3E17	LD	A, 17H	; 送出位置命令。★★★★★★★★★★★★★★★★★★★★
0643:	C33300	JP	0033H	; 到显示器控制器并返回。
0646:	CD4803	CALL	0348H	; 等待下一个键的输入。★★★输入过程中 HT 键★★

说明 若在键入过程中，按了 HT（水平 TAB 键，它的代码是 09H），则产生空格。将在缓冲器填补规定数目的空格，或者把缓冲器填满。填补的空格个数如下：当光标在水平位置 0 时，按 HT，则使光标移到位置 8 处（共有光标位置 0—64），即填了 8 个空格；在位置 1 处，也移到位置 8 处，此时填补了 7 个空格，……。当光标在水平位置 8 时，将光标到位置 16 处。以下依此类推。

0649:	E607	AND	07H	; 取 ASCII 码的低三位。
064B:	2F	CPL		; 取其反码。
064C:	3C	INC	A	; 给出数值 K=x<=8。
064D:	C608	ADD	A, 08H	; 清除计数器的高位。
064F:	5F	LD	E, A	; 把要加的空格数暂存起来。
0650:	78	LD	A, B	; 得到缓冲区余下空间的总数。
0651:	B7	OR	A	; 测试缓冲区是否满了。
0652:	C8	RET	Z	; 若满了，便返回。
0653:	3E20	LD	A, 20H	; 把空格的 ASCII 码放入 A 寄存器中。
0655:	77	LD	(HL), A	; 将空格存入缓冲区中。
0656:	23	INC	HL	; 缓冲区中地址加 1，指向下个单元。
0657:	D5	PUSH	DE	; 暂存调用它程序的 DE。
0658:	CD3300	CALL	0033H	; 显示空格。
065B:	D1	POP	DE	; 恢复 DE。
065C:	05	DEC	B	; 把缓冲区中剩下的字节数减 1。
065D:	1D	DEC	E	; 要加到缓冲区中的空格数减 1。
065E:	C8	RET	Z	; 若已加到了规定的空格数，便返回。
065F:	18EF	JR	0650H	; 循环，直到计数为零，或缓冲区已满。
0661:	37	SCF		; ★★★ 输入过程中，按了 |BREAK| 键。★★★★★★
				在输入过程中，若按 |BREAK|，则进位标志置位，否则进位标志清零。
0662:	F5	PUSH	AF	; 在输入时按 | CR | 键。
0663:	3E0D	LD	A, 0DH	; A=CR 代码，结束缓冲区。
0665:	77	LD	(HL), A	; 把结束符放入缓冲区。
0666:	CD3300	CALL	0033H	; 显示 CR。
0669:	3E0F	LD	A, 0FH	; 关光标代码。
066B:	CD3300	CALL	0033H	; 调用驱动程序，关闭光标。
066E:	79	LD	A, C	; C=缓冲区的大小。
066F:	90	SUB	A, B	; 缓冲区的大小减去处理过的字符数。
0670:	47	LD	B, A	; 给出缓冲区的字符数。
0671:	F1	POP	AF	; 取回 C 标志，若是 |BREAK| 键，则 C 标志置 1，若是 | CR |，则为 0。
0672:	E1	POP	HL	; HL=缓冲区的首地址。
0673:	C9	RET		; 返回到调用它的程序。

地址	目的码	源程序		注释
0674:	D3FF	OUT	(FFH), A	; 把 0 输出到磁带机。★★★显示器控制程序 ★★★★
0676:	21D206	LD	HL, 06D2H	; 显示器/键盘/打印机的 DCB 的地址。
0679:	110040	LD	DE, 4000H	; 通讯区的首地址。
067C:	013600	LD	BC, 0036H	; 为数据块传送而设置计数。
067F:	EDB0	LDIR		; 把 6D2H—707H 的内容移到 4000H—4035H 中。
0681:	3D	DEC	A	; 将传送到口地址 FFH 的数值,
0682:	3D	DEC	A	; 改变成 FFFDH, …, FFFBH, …。
0683:	20F1	JR	NZ, 0676H	; 把这整个过程循环 128 次。
0685:	0627	LD	B, 27H	; A=0
0687:	12	LD	(DE), A	; 把 0 存放在 4036H—4062H 中。
0688:	13	INC	DE	; 目标指针加 1。
0689:	10FC	DJNZ	0687H	; 若没有做完,便循环。
068B:	3A4038	LD	A, (3840H)	; 测试键盘输入是否为 BREAK 键。
068E:	E604	AND	04H	;
0690:	C27500	JR	NZ, 0075H	; 是,转移。
0693:	317D40	LD	SP, 407DH	; 置新的堆栈区。
0696:	3AEC37	LD	A, (37ECH)	; 读入磁盘状态。
0699:	3C	INC	A	; 测试扩展接口,
069A:	FE02	CP	02H	; 和磁盘驱动器。
069C:	DA7500	JP	C, 0075H	; 若没有联接磁盘,便转移。
069F:	3E01	LD	A, 01H	; 驱动器 0 的设备选择掩码。
06A1:	32E137	LD	(37E1H), A	; 选择驱动器 0。
06A4:	21EC37	LD	HL, 37ECH	; 磁盘命令/状态寄存器的地址。
06A7:	11EF37	LD	DE, 37EFH	; 磁盘数据寄存器地址。
06AA:	3603	LD	(HL), 03H	; 把 3 放入磁盘命令寄存器,于是复原,定位于 0 道。
06AC:	010000	LD	BC, 0000H	; 延时计数。
06AF:	CD6000	CALL	0060H	; 延迟近 3 秒。
06B2:	CB46	BIT	0, (HL)	; 测试控制器是否忙。
06B4:	20FC	JR	NZ, 06B2H	; 若忙,则循环等待。
06B6:	AF	XOR	A	; A=0。
06B7:	32EE37	LD	(37EEH), A	; 把 0 放入扇区寄存器。
06BA:	010042	LD	BC, 4200H	; BC=缓冲区的地址。
06BD:	3E8C	LD	A, 8CH	; A=读命令。
06BF:	77	LD	(HL), A	; 把 0 道 0 扇区读入 4200H—42FFH。
06C0:	CB4E	BIT	1, (HL)	; 测试是否准备好数据。
06C2:	28FC	JR	Z, 06C0H	; 若没有数据,则转移。
06C4:	1A	LD	A, (DE)	; 从磁盘取下一个字节。
06C5:	02	LD	(BC), A	; 把数据传送到 4200H 开始的缓冲区。
06C6:	0C	INC	C	; 缓冲区指针加 1。
06C7:	20F7	JR	NZ, 06C0H	; 若还不到 256 个字节,则循环。
06C9:	C30042	JP	4200H	; 传送完,转到 TRSDOS 的装入程序。

地址	目的码	源程序		注　释
06CC:	01181A	LD	BC, 1A18H	; BASIC READY 程序的地址(返回地址)★★★★★★★
06CF:	C3AE19	JP	19AEH	; 把 BASIC 的变量和指针初始化，然后转到 1A18H (BASIC READY 程序)。
06D2:	C3961C	JP	1C96H	; 4000H——执行 RST 08H 转到 JP 1C96H (比较)。

说明 在执行 IPL 程序的第一步时，便把存储单元 6D2H—707H 中的内容移到通讯区 4000H—4035H 单元中。

06D5:	C3781D	JP	1D78H	; 4003H——执行 RST 10H 转到 JP 1D78H (取下一个字符)。
06D8:	C3901C	JP	1C90H	; 4006H——执行 RST 18H 转到 JP 1C90H (比较 DE: HL)。
06DB:	C3D925	JP	25D9H	; 4009H——执行 RST 20H 转到 JP 25D9H (测试数据类型)。
06DE:	C9	RET		; 400CH——执行 RST 28H 转成 RET (对 DOS 是 JP 4BA2H)。
06DF:	00	NOP		; 400FH——执行 RST 30H 转成 RET (对 DOS 是 JP
06E0:	00	NOP		; 44B4H)。
06E1:	C9	RET		;
06E2:	00	NOP		; 4012H——执行 RST 38H 变成中断向量的入口点:
06E3:	00	NOP		; DI 和 RET (对 DOS 是 JP 4518H)。
06E4:	FB	EI		;
06E5:	C9	RET		;
06E6:	00	NOP		;
06E7:	01E303	LI	BC, 03E3H	; 4015H——键盘的 DCB，驱动程序地址=3E3H。 ★★
06EA:	00	NOP		;
06EB:	00	NOP		;
06EC:	00	NOP		;
06ED:	4B	LD	C, E	; 'K'
06EE:	49	LD	C, C	; 'I''
06EF:	07	RLCA		; 401DH——显示器的 DCB，驱动程序地址=458H。 ★
06F0:	58	LD	E, B	;
06F1:	04	INC	B	;
06F2:	00	NOP		;
06F3:	3C	INC	A	;
06F4:	00	NOP		;
06F5:	44	LD	B, H	; D'
06F6:	4F	LD	C, A	; 'O'
06F7:	068D	LD	B, 8DH	; 4025H——行式打印机的 DCB，驱动程序地址=58DH。
				★★★★★★★★★★★★★★★★★★★★★★★
06F9:	05	DEC	B	;
06FA:	43	LD	B, E	;
06FB:	00	NOP		;

地址	目的码	源程序		注释
06FC:	00	NOP		;
06FD:	50	LD	D, B	;
06FE:	52	LD	D, D	;
06FF:	C30050	JP	5000H	; 402DH——被 SYS0 改变为 JP 4400H。
0702:	C7	RST	00H	; 4030H——被 SYS0 改变为 LD A, A3H。
0703:	00	NOP		;
0704:	00	NOP		; 4032H——被 SYS0 改变为 RST 28H。
0705:	3E00	LD	A, 00H	; 4033H——被 SYS0 改变为 JP 44BBH。
0707:	C9	RET		;
0708:	218013	LD	HL, 1380H	; 单精度运算子程序的地址。★★★★★★★★★★

说明 单精度加子程序(有五个入口点)

0708H 从本地址入口时,把数值 0.5 放入 BC 和 DE 中,然后把它与 WRA1 中的值相加。

070BH 从本地址入口时,把 HL 指针所指的单精度数值放入 BC 和 DE 中,然后把它与 WRA1 中的值相加。

0710H 从本地址入口时,把 HL 指针所指的单精度数值放入 BC 和 DE 中,然后在 BC 和 DE 与 WRA1 相加之前,改变 WRA1 中的数值的符号,再作加法。

0713H 从本地址入口时,在把 BC 和 DE 与 WRA1 中的数值相加之前,先改变 WRA1 中的数值的符号,再作加法。

0716H 从本地址入口时,把 WRA1 与 BC 和 DE 中的数值相加,其和留在 WRA1 中。

070B:	CDC209	CALL	09C2H	; 将由 HL 指针所指向的单精度数放入 BC 和 DE 中,
070E:	1806	JR	0716H	; 转移去把 BC 和 DE 中的单精度数和 4121H—4124 H 的内容相加。
0710:	CDC209	CALL	09C2H	; 把当前值放到 BC 和 DE 中。
0713:	CD8209	CALL	0982H	; 改变 WRA1 中数值的符号。
0716:	78	LD	A, B	; 取寄存器中数值的指数项。
0717:	B7	OR	A	; 置指数项的标志状态。
0718:	C8	RET	Z	; 若指数 =0, 表示寄存器中的数是 0。
0719:	3A2441	LD	A, (4124H)	; 然后取另一个数的指数。
071C:	B7	OR	A	; 并测试它的指数。
071D:	CAB409	JP	Z, 09B4H	; 若是 0, 转移。
0720:	90	SUB	A, B	; A=当前指数−寄存器指数=要换算的位数。
0721:	300C	JR	NC, 072FH	; 寄存器的指数小,因此数值也小。
0723:	2F	CPL		; 把指数的差值变为正数,
0724:	3C	INC	A	; 同时把当前值与寄存器中的值交换,
0725:	EB	EX	DE, HL	; 使最小的一个值在寄存器中。
0726:	CDA409	CALL	09A4H	; 把 4121H—4124H 中的单精度值放入堆栈中。
0729:	EB	EX	DE, HL	; 恢复 HL 为第二值的地址。
072A:	CDB409	CALL	09B4H	; 把寄存器中的单精度值放入 4121H—4124H 中。
072D:	C1	POP	BC	; } 把上面 0726H 放入堆栈的单精度数值取出来。
072E:	D1	POP	DE	; }
072F:	FE19	CP	19H	; 若指数的差值大于 24, 则由于超出了量级的范围,
0731:	D0	RET	NC	; 因此不能相加。

地址	目的码	源程序		注　释
0732:	F5	PUSH	AF	; 把寄存器值要右移的位数保存,于是它的指数＝当前值的指数。
0733:	CDDF09	CALL	09DFH	; 把要加的两个值的最高位置 1。
0736:	67	LD	H, A	; 把测定的符号放在 H 中， A=BC 和 DE 中数值要右移的位数，
0737:	F1	POP	AF	; 于是，使寄存器的值换算成与当前值（WRA1）具有同样的量级。
0738:	CDD707	CALL	07D7H	; 转移去分解 BC 和 DE 中的值。
073B:	B4	OR	H	; 置寄存器中值的符号的状态标志。
073C:	212141	LD	HL, 4121H	; 取 WRA1 的地址。
073F:	F25407	JP	P, 0754H	; 若寄存器中的数值是负数,则转移。
0742:	CDB707	CALL	07B7H	; 把 CDE 中的单精度数与 HL 指针指出的单精度数相加,和放在 CDE 中,若两个尾数大小相同,则转移。
0745:	D29607	JP	NC, 0796H	;
0748:	23	INC	HL	;
0749:	34	INC	(HL)	; 否则指数加 1。
074A:	CAB207	JP	Z, 07B2H	; 若指数溢出为 0, 则有错误,转移到 07B2H 给出 OV 信息。
074D:	2E01	LD	L, 01H	; L＝要移位的个数。
074F:	CDEB07	CALL	07EBH	; 把尾数向右移一位。
0752:	1842	JR	0796H	; 进行数值归一化,并返回到调用它的程序。
0754:	AF	XOR	A	; 清除 A 和标志位。★★★把 BC 和 DE 中的负单精度值与 HL 指针给的正单精度值相加,和在 BC 和 DE 中。 ★★★★★★★★★★★★★★★★★★★★★★
0755:	90	SUB	A, B	; 0－指数＝－指数。
0756:	47	LD	B, A	; 把负指数暂存起来。
0757:	7E	LD	A, (HL)	; 取内存数值的 LSB。
0758:	9B	SBC	A, E	; 减寄存器数值的 LSB。
0759:	5F	LD	E, A	; E=新的寄存器值的 LSB。
075A:	23	INC	HL	; 地址加 1, 指向内存值的中间一个字节。
075B:	7E	LD	A, (HL)	; 取内存值的中间字节。
075C:	9A	SBC	A, D	; 减去寄存器值的中间字节的数值。
075D:	57	LD	D, A	; D=新的寄存器值的中间字节。
075E:	23	INC	HL	; 地址加 1, 指向内存值的 MSB。
075F:	7E	LD	A, (HL)	; 取内存值的 MSB。
0760:	99	SBC	A, C	; 减去寄存器值的 MSB。
0761:	4F	LD	C, A	; C=新的寄存器值的 MSB。
0762:	DCC307	CALL	C, 07C3H	; 若进位标志置位,把它转换成正的对应值。
0765:	68	LD	L, B	; L=原寄存器值的指数。

说明　把整数转换成单精度数的第一部分：在入口处，要转换的整数分别放在 CDE 中，它们是:C=MSB, D=NMSB, E=LSB。若这三个字节都为零,则将指数也置成零(在 4124H 中)，而其它三个字节已经

地址	目的码	源程序		注　释

为零了。若该整数不为零，则取第一个非零字节，并转到 785H—77DH 中去进行归一化的处理(把它进行移位，直到 MSB 为 1)。

```
0766:  63      LD    H, E          ; H=LSB
0767:  AF      XOR   A             ; 清除 A 寄存器和标志位。
0768:  47      LD    B, A          ; B=测试过的字节的计数。
0769:  79      LD    A, C          ; 取新的寄存器值的下一个字节 (MSB/NMSB/LSB)
076A:  B7      OR    A             ; 测试 MSB 是否为 0。
076B:  2018    JR    NZ, 0785H     ; 若 MSB 不为 0，转移(归一化寄存器的值)。
076D:  4A      LD    C, D          ; 这是一个 8 位的循环左移。把寄存器值旋转左移 8 位，
                                   若整个值是 0，
076E:  54      LD    D, H          ; 则将指数也置为 0，并返回。左移 8 位：C←D←H←L
                                   ←A。
076F:  65      LD    H, L          ; C←D←H
0770:  6F      LD    L, A          ; H←L←A
0771:  78      LD    A, B          ; 把 B 中的零进行递推，直到测试到一个非零的字节，
0772:  D608    SUB   08H           ; 或寄存器值的三个字节都经过测试了。
0774:  FEE0    CP    E0H           ; 测试待测数值的三个字节是否都测试过了。
0776:  20F0    JR    NZ, 0768H     ; 否，转移。
0778:  AF      XOR   A             ; 是，将数值清零。
0779:  322441  LD    (4124H), A    ; 指数清零。
077C:  C9      RET                 ; 返回到调用它的程序。
077D:  05      DEC   B             ; 计数左移一位。
```

说明　把整数转换成单精度数的第二部分：把 CDHL 作为一个整体进行左移，即把 L 的最高位移入 H，H 的最高位移入 D，D 的最高位移入 C，一直循环，直到 C 的最高位被移入位 15 时才停止左移。必须将移位的计数以负数形式放在 B 寄存器中。

```
077E:  29      ADD   HL, HL        ; 把 HL 左移一位。
077F:  7A      LD    A, D          ; 然后把 D 左移一位。
0780:  17      RLA                 ; 把 HL 寄存器移到进位中的那一位取来，
0781:  57      LD    D, A          ; 重新放回到 D 中，
0782:  79      LD    A, C          ; 然后再把 C 左移一位。
0783:  8F      ADC   A, A          ; 把 D 寄存器移到进位中的那一位取来，
0784:  4F      LD    C, A          ; 重新放回到 C 中。
0785:  F27D07  JP    P, 077DH      ; 循环，直到把 CDHL 归一化了为止。
0788:  78      LD    A, B          ; A=左移位数的计数。
0789:  5C      LD    E, H          ; 把 HL 暂存起来，
078A:  45      LD    B, L          ; 这样可将它用作指数的地址。
078B:  B7      OR    A             ; 测试移位的计数。
078C:  2808    JR    Z, 0796H      ; 若寄存器值已经归一化，或负数时，便转移。
078E:  212441  LD    HL, 4124H     ; HL=原寄存器值的指数项地址。
0791:  86      ADD   A, (HL)       ; 将移位计数与偏置值相加。
0792:  77      LD    (HL), A       ; 把结果作为指数存起来。
```

地址	目的码	源程序		注　释
0793:	30E3	JR	NC, 0778H	; 若数值小于 2^{24}，则置数为 0。
0795:	C8	RET	Z	; 当指数为 0，以 WRA1=0 返回。
0796:	78	LD	A, B	; 取数值的 LSB。
0797:	212441	LD	HL, 4124H	; 把指数项地址放入 HL 中。

说明 把整数转换成单精度数的第三部分：清除尾数的符号位(在上述归一化处理中，它被置成负数)，然后准备好存储结果的寄存器。

079A:	B7	OR	A	; 测试 LSB 中是否有数。
079B:	FCA807	CALL	M, 07A8H	; 若有数，则测试是否溢出。
079E:	46	LD	B, (HL)	; 否则把指数放入 B 中。
079F:	23	INC	HL	; 地址加 1，指向 4025H (含有结果的符号)。
07A0:	7E	LD	A, (HL)	; 然后取其符号，将符号分离出来，
07A1:	E680	AND	80H	; 以便可以和新的指数合在一起。
07A3:	A9	XOR	C	; 清除 MSB 的符号位。
07A4:	4F	LD	C, A	; B=指数，C=MSB，D=NMSB，E=LSB。
07A5:	C3B409	JP	09B4H	; 把 BC 和 DE 中的单精度数值放入 4121H—4124H 中。
07A8:	1C	INC	E	; LSB 加 1。

说明 在返回调用它的程序时，负数中的 0 都变成了 1，现在要把尾数中尾随的 0 (现在都是 1) 再变为 0，同时，在构成一个新的单精度数时，用于测试是否溢出。

07A9:	C0	RET	NZ	; 若没有溢出，则返回。
07AA:	14	INC	D	; 处理下一个字节，该数加 1。
07AB:	C0	RET	NZ	; 若不溢出，则返回。
07AC:	0C	INC	C	; 处理下一个字节，该数加 1。
07AD:	C0	RET	NZ	; 若不溢出，则返回。
07AE:	0E80	LD	C, 80H	; 把值置成负零。
07B0:	34	INC	(HL)	; 指数加 1。
07B1:	C0	RET	NZ	; 若没有溢出，则返回。
07B2:	1E0A	LD	E, 0AH	; OV 错误代码。
07B4:	C3A219	JP	19A2H	; 输出 OV 错误信息。
07B7:	7E	LD	A, (HL)	; 取存储器数值的 LSB。

说明 在 CDE 中的三个字节的单精度数，与 HL 指针的三个字节的单精度数相加，这两个数中必须有一个数已按比例经过换算了，因此它们的指数相同。在 LSB 相加时产生的进位要加到 MSB 中，这样从 LSB 加到 MSB。在出口时，A=MSB，若相加后尾数增加，则 C 标志置 1，这时必须调整指数值。否则 C 标志是零，其和留在 CDE 中。

07B8:	83	ADD	A, E	; 加寄存器值的 LSB。
07B9:	5F	LD	E, A	; 存储新的 LSB。
07BA:	23	INC	HL	; 指向存储器数值的中间字节。
07BB:	7E	LD	A, (HL)	; 取存储器数值的中间字节。
07BC:	8A	ADC	A, D	; 加寄存器数值的中间字节。
07BD:	57	LD	D, A	; 存储新的中间字节。
07BE:	23	INC	HL	; 指向存储器数值的 MSB。

地址	目的码	源程序		注释
07BF:	7E	LD	A, (HL)	; 取存储器数值的 MSB。
07C0:	89	ADC	A, C	; 加寄存器数值的 MSB。
07C1:	4F	LD	C, A	; 存储新的 MSB。
07C2:	C9	RET		; 返回到调用它的程序。
07C3:	212541	LD	HL, 4125H	; 重新设置符号位。 ★★★★★★★★★★★★★★★★

说明 本段程序把 4 个字节的负整数求补，变成两个正数，这样就可以将它转换成单精度数。单精度数的符号位(在 4125H 中)也已求了反。这样保证归一化以后依然是个负的尾数。

07C6:	7E	LD	A, (HL)	; 于是尾数将有一个负号。
07C7:	2F	CPL		; 把符号求反。
07C8:	77	LD	(HL), A	; 把符号位存回去。
07C9:	AF	XOR	A	; A 清零。
07CA:	6F	LD	L, A	; 把它暂存起来。
07CB:	90	SUB	A, B	; 把 B 求补 (0—B)，
07CC:	47	LD	B, A	; 把新的值放回 B 中。
07CD:	7D	LD	A, L	; A 重新置 0。
07CE:	9B	SBC	A, E	; 把 E 求补 (0—E)，
07CF:	5F	LD	E, A	; 把新的值放回 E 中。
07D0:	7D	LD	A, L	; A 重新置 0。
07D1:	9A	SBC	A, D	; 把 D 求补 (0—D)，
07D2:	57	LD	D, A	; 把新的值放回 D 中。
07D3:	7D	LD	A, L	; A 再置 0。
07D4:	99	SBC	A, C	; 把 C 求补 (0—C)，
07D5:	4F	LD	C, A	; 把新的值放回 C 中。
07D6:	C9	RET		; 返回到调用它的程序。
07D7:	0600	LD	B, 00H	; 入口时，★★★分解一个单精度数★★★★★★★★
07D9	D608	SUB	08H	; A=右移的位数。
07DB:	3807	JR	C, 07E4H	; 若 C 标志置位，则右移 (A) 位，
07DD:	43	LD	B, E	; 否则将数右移一个字节。从这里到 07E2H 为止，
07DE:	5A	LD	E, D	; 把 00CDE 这样位移，
07DF:	51	LD	D, C	; 移位后，变成 E00CD。
07E0:	0E00	LD	C, 00H	;
07E2:	18F5	JR	07D9H	; 再循环，判断要不要再右移一个字节。
07E4:	C609	ADD	A, 09H	; 使移位计数为正。
07E6:	6F	LD	L, A	; 把它放入 L 中。
07E7:	AF	XOR	A	; 清除标志位。
07E8:	2D	DEC	L	; 把移位数减 1。
07E9:	C8	RET	Z	; 如果完成了，从这里返回。
07EA:	79	LD	A, C	; 现在把 BCDE 作为一个整体，一次右旋一位。
07EB:	1F	RRA		; 把 C 右旋一位，而 C 的位 0 进入进位标志。
07EC:	4F	LD	C, A	; 把新的值放入 C 中。
07ED:	7A	LD	A, D	; 把 D 右移一位，而 C 的位 0 变成 D 的位 7。

地址	目的码	源程序		注释
07EE:	1F	RRA		; D 的位 0 移入进位标志。
07EF:	57	LD	D, A	; 把新的值放入 D 中。
07F0:	7B	LD	A, E	; 把 E 右移一位。
07F1:	1F	RRA		; D 的位 0 变成 E 的位 7，E 的位 0 移入进位标置。
07F2:	5F	LD	E, A	; 把新的值放入 E 中。
07F3:	78	LD	A, B	; 最后把 B 右移一位，
07F4:	1F	RRA		; E 的位 0 变成 B 的位 7，B 的位 0 便移出去了。
07F5:	47	LD	B, A	: 把新的值放回 B 中。
07F6:	18EF	JR	07E7H	; 循环，直到移了（L）位。整数部分留在 CDE 中，小数部分留在 B 中。
07F8:	00	NOP		; ★★★★★★★★★★★★★★★★★★★★★★★★
07F9:	00	NOP		; 07F8H—07FBH 是单精度浮点数 1.0。
07FA:	00	NOP		;
07FB:	81	ADD	A, C	;
07FC:	03	INC	BC	; 03 表示以下三个单精度数值是用在 LN 幂级数中的系数。
07FD:	AA	XOR	D	: 07FDH—0800H 是单精度浮点数 0.5988。
07FE:	56	LD	D, (HL)	;
07FF:	19	ADD	HL, DE	;
0800:	80	ADD	A, B	;
0801:	F1	POP	AF	; 0801H—0804H 是单精度浮点数 0.96145。
0802:	227680	LD	(8076H), HL	;
0805:	45	LD	B, L	; 0805H—0808H 是单精度浮点数 2.88539。
0806:	AA	XOR	D	;
0807:	3882	JR	C, 078BH	;
0809:	CD5509	CALL	0955H	; 测试当前单精度数值的符号位。★★★ LOG 子程序 ★★★★★★★★★★★★★★★★★★★★★★★★

说明 使用方法：

1. 测试数值的符号，若是负号，则 FC 错误返回。
2. 把数值按比例进行换算，使它在 0.5 至 1.0 之间，然后把换算中使用的位计数存起来。
3. 按以下公式重新计算换算后的值。
   $x=1-(2\ln 2/(x+\ln 2))$
4. 计算：
   $((x^2\times c_0+c_1)\times x^2+c_2)\times x$
5. 把级数的最后结果减 0.5。
6. 把第 5 步的计算结果加上移位计数。
7. 把第 6 步的结果乘以 ln 2。

080C:	B7	OR	A	; 根据符号位，置状态标志。
080D:	EA4A1E	JP	PE, 1E4AH	; 若是负数，则产生错误。
0810:	212441	LD	HL, 4124H	; HL＝当前值的指数项的地址。
0813:	7E	LD	A, (HL)	; A＝当前数值的指数值。
0814:	013580	LD	BC, 8035H	; BC 和 DE＝0.707092，

地址	目的码	源程序		注释
0817:	11F304	LD	DE, 04F3H	; (近似于 ln 2)。
081A:	90	SUB	A, B	; 换算数值,使它<1。
081B:	F5	PUSH	AF	; 保存换算(比例)因子。
081C:	70	LD	(HL), B	; 使当前数值的指数值与 BC 和 DE 中的常数一样大。
081D:	D5	PUSH	DE	; 把 BC 和 DE 中的常数放入堆栈。
081E:	C5	PUSH	BC	;
081F:	CD1607	CALL	0716H	; 把当前值与 BC 和 DE 中常数相加。
0822:	C1	POP	BC	; 取回 BC 和 DE 中常数相加。
0823:	D1	POP	DE	;
0824:	04	INC	B	; 把指数加 1,等于常数乘以 2。
0825:	CDA208	CALL	08A2H	; 1.4141 (近似 ln 4) 除以换算后的值 +ln2。
0828:	21F807	LD	HL, 07F8H	; HL=单精度数值 1.0 的地址。
082B:	CD1007	CALL	0710H	; 把 1.0 取入 BC 和 DE 中,并从当前值中减去这个值。
082E:	21FC07	LD	HL, 07FCH	; 把 HL 指向三个系数的首地址。
0831:	CD9A14	CALL	149AH	; 调用计算级数的程序,求级数之和。

说明 计算级数之和:

$(((x^2\times c_0+c_1)\times x^2+c_2)\times x^2+\cdots+c_N)\times x$

其中由 I 规定了计算多少项, $I=0,\cdots,N$。这里调用时 $I=2$。

0834:	018080	LD	BC, 8080H	; BC 和 DE=−0.5
0837:	110000	LD	DE, 0000H	;
083A:	CD1607	CALL	0716H	; 把当前值加−0.5。
083D:	F1	POP	AF	; 取回在 81AH 处推入堆栈的换算因子。
083E:	CD890F	CALL	0F89H	; 把当前值换算成原先的值。
0841:	013180	LD	BC, 8031H	; 使 BC 和 DE=0.693115 (=ln2),
0844:	111872	LD	DE, 7218H	; 然后把级数和与 0.693115 相乘。
0847:	CD5509	CALL	0955H	; 测试当前 SP 符号位和指数项。★★★ 乘法子程序 ★

说明 单精度相乘:把当前值与 BC 和 DE 相乘。使用移位和相加的方法。首先分解每个数,然后再移位相加。

084A:	C8	RET	Z	; 若指数为零,便返回。
084B:	2E00	LD	L, 00H	; L=00H, 表示指数相加。
084D:	CD1409	CALL	0914H	; 把指数相加,将每个数的 MSB 字节中的最高位置1。
0850:	79	LD	A, C	;
0851:	324F41	LD	(414FH), A	; 414FH=寄存器数值的 MSB。
0854:	EB	EX	DE, HL	;
0855:	225041	LD	(4150H), HL	; 4150H—4151H 是寄存器数值的后两位字节。
0858:	010000	LD	BC, 0000H	; BC=0
085B:	50	LD	D, B	; DE=0
085C:	58	LD	E, B	;
085D:	216507	LD	HL, 0765H	; 在相乘以后调用整数转换单精度数的程序,
0860:	E5	PUSH	HL	; 将结果转换成单精度数。

地址	目的码	源程序		注　释
0861:	216908	LD	HL, 0869H	; 我们将转移到后面的 0869H 处进行分解单精度数的工作。
0864:	E5	PUSH	HL	; 把 0869H 放入堆栈两次，
0865:	E5	PUSH	HL	; 这样我们就可以对每一个单精度数值进行分解处理。
0866:	212141	LD	HL, 4121H	; HL=当前值的地址。
0869:	7E	LD	A, (HL)	; 测试 LSB 是否为零。
086A:	23	INC	HL	; HL=NMSB 地址。
086B:	B7	OR	A	; A=当前单精度值的 LSB。
086C:	2824	JR	Z, 0892H	; 若 LSB 为零，转移(作一个字节的循环右移)，然后再取下一个字节。
086E:	E5	PUSH	HL	; 把 NMSB 的地址暂存起来。

说明 用右移和测试 C 标志状态的方法来检查当前值中的 1。当发现一个 1 时，便把当前值与寄存器的值(现在在 414FH—4151H 中)相加，在当前值的各位都测试过一遍以前，一直重复这个处理过程。

086F:	2E08	LD	L, 08H	; L=右移 SP 位移计数(把工作区的值位移成朝右排齐)。
0871:	1F	RRA		; 把 LSB 右移一位。
0872:	67	LD	H, A	; 暂存右移后的 LSB。
0873:	79	LD	A, C	; 把 MSB 取入 A 中。
0874:	300B	JR	NC, 0881H	; 若从 LSB 中没有移出 1，则转移到 0881H。
0876:	E5	PUSH	HL	; 否则把移位后的 LSB 和计数暂存起来。
0877:	2A5041	LD	HL, (4150H)	; 原寄存器数值的 NMSB 和 LSB 的地址。
087A:	19	ADD	HL, DE	; 将它们与当前的部分和相加(部分加)。
087B:	EB	EX	DE, HL	; 把部分和留在相应寄存器中。
087C:	E1	POP	HL	; 取回 LSB 和移位计数。
087D:	3A4F41	LD	A, (414FH)	; 然后把原寄存器值的 MSB 加到累加总数中去。
0880:	89	ADC	A, C	;
0881:	1F	RRA		; 右移 MSB，
0882:	4F	LD	C, A	; 暂存移位后的 MSB。

说明 取寄存器值的 MSB，并与当前值的 MSB 相加，然后继续。

0883:	7A	LD	A, D	; 取来中间字节，

说明 把在寄存器中的当前数值向右对齐，得到一个整数值。右移 D 和 E，先右移 D 寄存器，把 D 寄存器中的位 0 移入 C 标志。再在移位 E 时，移到 E 寄存器中。这样重复，结果留在 BC 和 DE 中，这是一个没有归一化的浮点数。当前值的指数在 4124H 中(保持经过调整后的指数)。

0884:	1F	RRA		; 右移一位，
0885:	57	LD	D, A	; 把移位后的中间字节暂存起来。
0886:	7B	LD	A, E	; 取 LSB，
0887:	1F	RRA		; 右移一位，
0888:	5F	LD	E, A	; 把移位后的 LSB 存回原处。
0889:	78	LD	A, B	; 取指数，
088A:	1F	RRA		; 右移一位，
088B:	47	LD	B, A	; 放回去。

地址	目的码	原程序		注释
088C:	2D	DEC	L	; 计数减 1,
088D:	7C	LD	A, H	; 把原来的 LSB 放回到 A 中。
088E:	20E1	JR	NZ, 0871H	; 继续循环,直到测试完全部 8 位。
0890:	E1	POP	HL	; HL 恢复为下一个字节的地址。
0891:	C9	RET		; 返回。
0892:	43	LD	B,E	; ★★★在相乘以前分解 SP, 由相乘 SP 程序调用 ★★
0893:	5A	LD	E, D	; 循环左移 BC 和 DE 一个字节, B 的值丢失了, A 的值放在 C 中,
0894:	51	LD	D, C	; A 与 BC 和 DE 的位移是这样的:
0895:	4F	LD	C, A	; A→C C→D D→E E→B
0896:	C9	RET		; 返回。
0897:	CDA409	CALL	09A4H	; 把 WRA1 中的值放入堆栈。★★★★★★★★★★★
089A:	21D80D	LD	HL, 0DD8H	; 浮点数 10.0 的地址。
089D:	CDB109	CALL	09B1H	; 把浮点数 10.0 取入 BC 和 DE, 再移入 4121H—4124H。
08A0:	C1	POP	BC	; 把 WRA1 中的值重新放入 BC 和 DE 中。
08A1:	D1	POP	DE	;
08A2:	CD5509	CALL	0955H	; 测试 WRA1 中数值的符号。★★★单精度除法程序★
08A5:	CA9A19	JP	Z, 199AH	; 被零除,错误。
08A8:	2EFF	LD	L, FFH	; L=FFH 表示减指数。
08AA:	CD1409	CALL	0914H	; 用加法求出新的指数,把每个数值最高位置 1,
08AD:	34	INC	(HL)	; 结果的符号位留在 4125H 中。
08AE:	34	INC	(HL)	; 被除数的指数加 2。
08AF:	2B	DEC	HL	; HL=4123H=当前数值的 MSB。
08B0:	7E	LD	A, (HL)	; 把 WRA1 中 MSB 值取来。
08B1:	328940	LD	(4089H), A	; 4089H=当前值的 MSB。
08B4:	2B	DEC	HL	; HL=NMSB 的地址。
08B5:	7E	LD	A, (HL)	; 取 NMSB。
08B6:	328540	LD	(4085H), A	; 4085H=当前值的 NMSB。
08B9:	2B	DEC	HL	; HL=LSB 的地址。
08BA:	7E	LD	A, (HL)	; 取 LSB,
08BB:	328140	LD	(4081H), A	; 并把它放入 4081H 中。
08BE:	41	LD	B, C	; B=寄存器数值的 MSB。
08BF:	EB	EX	DE, HL	; DE=4122H, HL=寄存器数值的 NMSB 和 LSB。
08C0:	AF	XOR	A	; A 和标志位清零。
08C1:	4F	LD	C, A	; 把寄存器数值的 MSB, NMSB 和 LSB 清零。
08C2:	57	LD	D, A	;
08C3:	5F	LD	E, A	;
08C4:	328C40	LD	(408CH), A	; 对 B 和 HL 的溢出进行加倍处理的次数计数器清零。
08C7:	E5	PUSH	HL	; 把 BC 和 HL 中的除数放入堆栈。
08C8:	C5	PUSH	BC	; B 是寄数器数值的 MSB, C=0。
08C9:	7D	LD	A, L	; A 是寄存器数值的 LSB, 现在计算被除数—除数。

地址	目的码	源程序		注释
08CA:	CD8040	CALL	4080H	; 把寄存器的数值减去当前(4081H—4089H)的值,结果在 B 和 HL 中。
08CD:	DE00	SBC	A, 00H	; 若寄存器数值<当前值,则出口时, A=0, C 标志=1。
08CF:	3F	CCF		; 若寄存器数值>当前值,把进位标志求反, C 标志=1。
08D0:	3007	JR	NC, 08D9H	; 若寄存器数值<当前值,转移去除数加倍处理。
08D2:	328C40	LD	(408CH), A	; 把对 B 和 HL 的溢出次数的计数保存起来。
08D5:	F1	POP	AF	; 把上一次除法中保存在堆栈中的一些值清除掉,
08D6:	F1	POP	AF	; 因为我们不再需要了。
08D7:	37	SCF		; 置进位标置。
08D8:	D2C1E1	JP	NC, E1C1H	; 08D9H: POP BC 取出上次的除数,以便将它加倍。
08DB:	79	LD	A, C	; 08DAH: POP HL
08DC:	3C	INC	A	; 首先测试有否可能在相除时 HL 溢出而进入 BC。
08DD:	3D	DEC	A	;
08DE:	1F	RRA		; 测试 C 的位 0,
08DF:	FA9707	JP	M, 0797H	; 若它为 1, 则转移去把结果归一化。
08E2:	17	RLA		; 清除可能的进位置 1。
08E3:	7B	LD	A, E	; 把 E 左移一位。把 CDE 作为一个整体都左移一位,从 E 移出的将移入 D 中,……依次类推。
08E4:	17	RLA		; 把 A 的位 7 取出。
08E5:	5F	LD	E, A	; 移位后的值放回 E 中,它的原最高位在进位位中。
08E6:	7A	LD	A, D	; 把 D 左移一位,
08E7:	17	RLA		; 原 E 的位 7 进入 D 的位 0。
08E8:	57	LD	D, A	; 把 D 放回, D 的原最高位在进位位中。
08E9	79	LD	A, C	; 把 C 左移一位。
08EA:	17	RLA		; 原 D 的位 7 进入 C 的位 0。
08EB:	4F	LD	C, A	; 放回 C 中。
08EC:	29	ADD	HL, HL	; 把除数加倍,使它最后超过了被除数。
08ED:	78	LD	A, B	; 当超过时,商和余数将在 B 和 HL 寄存器中,作为寄存器数值。
08EE:	17	RLA		; 把 HL 左移一位,产生的任何溢出,放入 B 中。
08EF:	47	LD	B, A	; 然后 B 左移一位。把 B 的溢出总数的计数,
08F0:	3A8C40	LD	A, (408CH)	; 作为比特流保存在 408CH 中。
08F3:	17	RLA		; 也就是 1 的个数等于溢出发生的次数。
08F4:	328C40	LD	(408CH), A	;
08F7:	79	LD	A, C	;
08F8:	B2	OR	D	; 现在把寄存器值的所有字节合并,
08F9:	B3	OR	E	;
08FA:	20CB	JR	NZ, 08C7H	; 并循环,直到除数溢出。
08FC:	E5	PUSH	HL	; 暂存 HL。
08FD:	212441	LD	HL, 4124H	; 取工作区中数值的指数。
0900:	35	DEC	(HL)	; 减 1, 为了求 $(A^x)/(B^y)=(A/B)^{(x-y)}$

地址	目的码	源程序		注释
0901:	E1	POP	HL	; 取回 HL。
0902:	20C3	JR	NZ, 08C7H	; 把移位和减 1 的运算循环。
0904:	C3B207	JP	07B2H	; OV 错误(指数达到零)。
0907:	3EFF	LD	A, FFH	; 计算浮点数相乘时新的指数值。

说明 当从 907H 入口时,调用它的程序应根据 WRA1 中数值的符号,确定 L 寄存器中的数值。当 WRA1>=0, L=0; 当 WRA1<0, L=FFH。

0909:	2EAF	LD	L, AFH	; 090A: XOR A, 把 A 和标志清零。
090B:	212D41	LD	HL, 412DH	; HL=在 WRA2 内双精度值 MSB 的地址。
090E:	4E	LD	C, (HL)	; C=WRA2 中数值的 MSB。
090F:	23	INC	HL	; HL=在 WRA2 内双精度值指数的地址。
0910:	AE	XOR	(HL)	; 根据使用的入口,使指数为正或负。
0911:	47	LD	B, A	; 把指数暂存在 B 中。
0912:	2E00	LD	L, 00H	; 用于测试 WRA1 的指数符号的掩码(强迫符号为+)。
0914:	78	LD	A, B	; 取回指数。
0915:	B7	OR	A	; 测试是否为零,并置标志。
0916:	281F	JR	Z, 0937H	; WRA1 的值是零。
0918:	7D	LD	A, L	; 不为零, 取 WRA1 值的指数。
0919:	212441	LD	HL, 4124H	; 我们已经知道它不为零,把 WRA1 的指数的符号
091C:	AE	XOR	(HL)	; 与 L 中的掩码合并,这是在 907H 的第二个入口。
091D:	80	ADD	A, B	; 把两个要相乘的数值的指数相加,
091E:	47	LD	B, A	; 并将结果保存在 B 中。
091F:	1F	RRA		; 相加后或许产生进位,现在来测试有无进位。循环右移
0920:	A8	XOR	B	; 把进位位移到位 7, 并和新的指数进行异或。(由于它产生了我们正测试的进位,所以位 7 应是 0。)
0921:	78	LD	A, B	;
0922:	F23609	JP	P, 0936H	; 若指数的和超出范围,转移。
0925:	C680	ADD	A, 80H	; 把新的指数放入 A, 并把位 7 置 1。
0927:	77	LD	(HL), A	; 存储新的指数,
0928:	CA9008	JP	Z, 0890H	; 若该值是零,则转移。
092B:	CDDF09	CALL	09DFH	; 把当前值的最高位置 1, 于是可以将它分解, 以便重复相加。
092E:	77	LD	(HL), A	;
092F:	2B	DEC	HL	; HL=NMSB
0930:	C9	RET		; 返回到调用它的程序。
0931:	CD5509	CALL	0955H	; 测试 WRA1 中浮点数的符号。★★★★★★★★★★
0934:	2F	CPL		; 将结果求反,因此若数是正值,则 A 为负数;
0935:	E1	POP	HL	; 若数为负值,则 A 为正数,
0936:	B7	OR	A	; 按新的指数置标志。
0937:	E1	POP	HL	; 清除堆栈。
0938:	F27807	JP	P, 0778H	; 把当前浮点数值置成零,并返回。
093B:	C3B207	JP	07B2H	; OV 错误,退出。

地址	目的码	源程序		注　释
093E:	CDBF09	CALL	09BFH	; 取 4121H—4124H 中的值。★★★★★★★★★★★★

说明 把 WRA1 中的浮点数乘 10，先把指数加 2，等于乘 4，再加原值，等于乘 5。

0941:	78	LD	A, B	; B=指数，C=MSB，D=NMSB，E=LSB。
0942:	B7	OR	A	; 按新的指数设置标志。
0943:	C8	RET	Z	; 若数为零，则返回。
0944:	C602	ADD	A, 02H	; 把寄存器的数乘 4。
0946:	DAB207	JP	C, 07B2H	; 若指数溢出，则错误。
0949:	47	LD	B, A	; 把新的指数放回去。
094A:	CD1607	CALL	0716H	; 加原值，等于原值乘 5。
094D:	212441	LD	HL, 4124H	; 4124H 是指数的地址，
0950:	34	INC	(HL)	; 把指数加 1，就等于增加一倍，
0951:	C0	RET	NZ	; 也就是原始数值乘 10。
0952:	C3B207	JP	07B2H	; OV 错误，转移。
0955:	3A2441	LD	A, (4124H)	; 测试工作区中 SP 的符号，返回时负数 A=－1，正数 A=＋1。
0958:	B7	OR	A	; 置指数的状态标志。
0959:	C8	RET	Z	; 若指数为零，则返回。
095A:	3A2315	LD	A, (4123H)	; 不为零，取 SP 的 MSP。
095D:	FE2F	CP	2FH	; 095E: CPL A
095F:	17	RLA		; 把符号位放入进位位。
0960:	9F	SBC	A, A	; 给出 0－符号位。
0961:	C0	RET	NZ	; 若 MSB 为负值，则 A=－1，返回。
0962:	3C	INC	A	; 若 MSB 为正，则 A=＋1，返回
0963:	C9	RET		; 返回到调用它的程序。
0964:	0688	LD	B, 88H	; B=80＋要转换的位数。★★★★★★★★★★★★

说明 把整数转换成单精度数。指数位存在 4124H 中，对于正的系数，置位符号标志(在 4125H 中)。整数的 MSB 在 C 中，LSB 在 D 中，而令 C 标志为 MSB 的符号。再调用归一化程序。若从 0969H 入口，则 B 必须等于整数值的二进制位数加 80H。

0966:	110000	LD	DE, 0000H	; 使用在归一化程序中的寄存器清零。
0969:	212441	LD	HL, 4124H	; 取 WRA1 指数的地址。
096C:	4F	LD	C, A	; C=整数的 MSB。
096D:	70	LD	(HL), B	; 把初始指数值暂存，放在 4124H 中。
096E:	0600	LD	B, 00H	; 在进入归一化程序前，B 必须为 0。
0970:	23	INC	HL	; 指向 WRA1 的符号字节，
0971:	3680	LD	(HL), 80H	; 得到它的正值。
0973:	17	RLA		; 置 C 标志为整数值的符号，
0974:	C36207	JP	0762H	; 去进行归一化
0977:	CD9409	CALL	0994H	; 测试当前值的符号。★★★ 把负数转换成正数★★★
097A:	F0	RET	P	; 若正数便返回，若负数，测定数据类型。
097B:	E7	RST	20H	; 测试数据类型。
097C:	FA5B0C	JP	M, 0C5BH	; 整数，则转换成正值，若溢出，则为单精度值并返回。

地址	目的码	源程序		注释
097F:	CAF60A	JP	Z, 0AF6H	; 若Z标志置1，则 TM 错误。
0982:	212341	LD	HL, 4123H	: 若是一个单精度或双精度数值，把 MSB 的符号位——位 7 置 0，使它变为正数。
0985:	7E	LD	A, (HL)	;
0986:	EE80	XOR	80H	; 若当前值为正，则将当前值置成 1。
0988:	77	LD	(HL), A	; 若该值为负，则将置成 0。
0989:	C9	RET		; 返回到调用它的程序。
098A:	CD9409	CALL	0994H	; 测试当前值的符号。★★★若该值是正数，则 A=+1，若该值是负数，则 A=−1 ★★★★★★★★★★★★★
098D:	6F	LD	L, A	; 这样设置 HL：若当前为正，则 HL=0000。
098E:	17	RLA		; 若当前值为负，则 HL=FFFFH。
098F:	9F	SBC	A, A	; 若 C 标志 =0，则 A=0，若 C 标志=1，
0990:	67	LD	H, A	; 则给出 A=FFH，把进位状态放入 H 中，
0991:	C39A0A	JP	0A9AH	; 把 HL 作为当前数值存储起来，得到整数类型并返回调用程序。
0994:	E7	RST	20H	; 测定当前数据类型，★★★测试数值符号，正数时 A=+1，负数时 A=−1 ★★★★★★★★★★★★★★
0995:	CAF60A	JP	Z, 0AF6H	; 若Z置1，是字符串，则 TM 错误。
0998:	F25509	JP	P,0955H	; 若单精度或双精度，则转移去测定符号，并返回。
099B:	2A2141	LD	HL, (4121H)	; 把整数取入 HL 中。
099E:	7C	LD	A, H	; 把 MSB 和 LSB 合并，
099F:	B5	OR	L	; 以测试整数是否为零。
09A0:	C8	RET	Z	; 若整数为零，则返回。
09A1:	7C	LD	A, H	; A=整数的 MSB。
09A2:	18BB	JR	095FH	; 去测试符号，并返回调用它的程序(正数 A=+1，负数 A=−1)。
09A4:	EB	EX	DE, HL	; 暂存 HL。★★★把 4121H—4124H (WRA1) 放入堆栈 ★★★★★★★★★★★★★★★★★★★★★★
09A5:	2A2141	LD	HL,(4121H)	; 取第一个数，
09A8:	E3	EX	(SP), HL	; 把要传送的数放入堆栈，返回地址放入 HL 中，堆栈=(4121H)。
09A9:	E5	PUSH	HL	; 返回地址给堆栈。
09AA:	2A2341	LD	HL, (4123H)	; 取第二个数，
09AD:	E3	EX	(SP), HL	; 把第二个要传送的数值放入堆栈中。
09AE:	E5	PUSH	HL	; 返回地址给堆栈。
09AF:	EB	EX	DE, HL	; 恢复 HL
09B0	C9	RET		; 返回调用它的程序。
09B1:	CDC209	CALL	09C2H	; 把以 HL 为指针的单精度数放入 BC 和 DE。 ★★★

说明 由 9B1H 入口时，把以 HL 为指针的单精度数→BC 和 DE→WRA1。出口时，B=指数，C=MSB，D=NMSB，E=LSB，HL=指数地址 +1。

09B4:	EB	EX	DE, HL	; 然后把它放入 WRA1 工作区中。

地址	目的码	源程序		注 释

说明 由 9B4 入口时，把 BC 和 DE 中的单精度数 →WRA1 中。入口前必须将单精度按以下位置放在寄存器中：B=指数，C=MSB，D=NMSB，E=LSB。

09B5:	222141	LD	(4121H),HL	; 把 HL 的值 (H=NMSB, L=LSB) 存储在 WRA1 中。
09B8:	60	LD	H, B	; H=B=指数。
09B9:	69	LD	L, C	; L=C=MSB。
09BA:	222341	LD	(4123H), HL	; 把它们存储在 WRA1 中。
09BD:	EB	EX	DE, HL	; 取回 HL 的原始内容。
09BE:	C9	RET		; 返回到调用它的程序。
09BF:	212141	LD	HL, 4121H	; 把 WRA1 (4121H—4124H) 中的单精度数放入 BC 和 DE 中。★★★★★★★★★★★★★★★★★★★★★

说明 由 9BFH 入口时，把 WRA1 中的单精度数送入 BC 和 DE。出口时，B=(4124H)=指数，C=(4123H)=MSB，D=(4122H)=NMSB，E=(4121H)=LSB，HL=4125H。

09C2:	5E	LD	E, (HL)	; E=LSB (或=(4121H))。★★★把以 HL 为指针的单精度数放入 BC 和 DE ★★★★★★★★★★★★★★

说明 由 9C2H 入口时，把以 HL 为指针的单精度数送入 BC 和 DE 中。入口前，HL 应指向单精度值的 LSB。出口时，B=指数，C=MSB，D=NMSB，E=LSB，HL=指数地址+1。

09C3:	23	INC	HL	; 指向下一个地址。
09C4:	56	LD	D, (HL)	; D=NMSB (或 4122H)。
09C5:	23	INC	HL	; 指向下一个地址。
09C6:	4E	LD	C, (HL)	; C=MSB或 (4123H)。
09C7:	23	INC	HL	; 指向下一个地址。
09C8:	46	LD	B, (HL)	; B=指数(或 4124H)。
09C9	23	INC	HL	; 指向下一代一个地址。
09CA	C9	RET		; 然后返回调用它的程序。
09CB:	112141	LD	DE, 4121H	; ★★★将单精度数从 WRA1(4121H—4124H) 送入 HL 为指针的内存单元中 ★★★★★★★★★★★★★★★ 单精度数的源地址。
09CE:	0604	LD	B,04H	; 要移动的字节数。
09D0:	1805	JR	09D7H	: 转移去，将单精度数送入由 HL 规定的单元中，并返回调用它的程序。
09D2:	EB	EX	DE, HL	: 传送子程序的入口点。★★★ HL = 源地址，DE = 目的地址 ★★★★★★★★★★★★★★★★★★★★★
09D3:	3AAF40	LD	A, (40AFH)	: 取类型说明(它也是要移动的字段长度，即字节个数)。
09D6:	47	LD	B, A	; 把要移动的字数放入 B 中。
09D7:	1A	LD	A, (DE)	: 从源区取一个字节。★★★通用传送子程序：DE = 源地址，HL = 目的地址，B = 要传送的字节数 ★★★★ ★★★★★★★★★★★★★★★★★★★★★★★★★
09D8:	77	LD	(HL), A	; 把该字节放入目的区中。
09D9:	13	INC	DE	; 源地址加 1。

地址	目的码	源程序		注　释
09DA:	23	INC	HL	; 目的地址加 1。
09DB:	05	DEC	B	; 计数移动了一个字节。
09DC:	20F9	JR	NZ, 09D7H	; 若还有数据要移动,则转移。
09DE:	C9	RET		; 否则返回到调用它的程序。
09DF:	212341	LD	HL, 4123H	; 指向单精度数值的 MSB。★★★ 将单精度值的最高位置 1 ★★★★★★★★★★★★★★★★★★★★★★
09E2:	7E	LD	A, (HL)	; 把 MSB 取来。
09E3:	07	RLCA		; 位 7 移入 C 标志。
09E4:	37	SCF		; 把位 7 置 1, 并把数放回去,
09E5:	1F	RRA		; 这样,原来的符号位还在 C 标志中。
09E6:	77	LD	(HL), A	; 把 MSB 放回去,此时它的最高位是 1。
09E7:	3F	CCF		; 把原位 0 的数求反,
09E8:	1F	RRA		; 并把它放入 MSB 的位 7 中(符号位,也是最高位)。
09E9:	23	INC	HL	; HL=4125H, 是结果的符号位,
09EA:	23	INC	HL	; 将在下面确定。
09EB:	77	LD	(HL), A	; 把原符号求反的值放入 4125H 中。
09EC:	79	LD	A, C	; 把在 BC 和 DE 寄存器中的单精度值
09ED:	07	RLCA		; MSB 中的位 7 (最高位)置 1。
09EE:	37	SCF		; 然后把 C 标志置 1, 这样就能得到位 7=1 的字节。
09EF:	1F	RRA		; 原符号位到 C 标置中。
09F0:	4F	LD	C, A	; 再放回到 C=MSB。
09F1:	1F	RRA		; 原符号位到位 7 中。
09F2:	AE	XOR	(HL)	; 设置符号标置,当两者符号相等, 则 (4125H)=80H。
09F3:	C9	RET		; 否则为 00H, 返回。
09F4:	212741	LD	HL, 4127H	; WRA2 的地址。★★★ WRA2→WRA1* ★★★★★★
09F7:	11D209	LD	DE, 09D2H	; 返回地址。

说明 由 09D2H 入口时, HL=源地址, DE=目的地址,传送 (40FAH) 个字节。

09FA:	1806	JR	0A02H	; 把数值从 WRA2 移到 WRA1 中。
09FC:	212741	LD	HL, 4127H	; WRA2 的地址。 ★★★ WRA1→WRA2 ★★★★★★
09FF:	11D309	LD	DE, 09D3H	; 把数值从 WRA1 移到 WRA2 中
0A02:	D5	PUSH	DE	; 9D3H 为返回地址。

说明 由 09D3H 入口时, HL=目的地址, DE=源地址,移动的字节=(40FAH)。

0A03:	112141	LD	DE, 4121H	; 在 WRA1 中当前变量的地址。
0A06:	E7	RST	20H	; 确定变量的类型。
0A07:	D8	RET	C	; 若是整数,字符串或单精度数,转到传送子程序。
0A08:	111D41	LD	DE, 411DH	; 取双精度变量的地址。
0A0B:	C9	RET		; 转到传送子程序。
0A0C:	78	LD	A, B	; ★☆★把 BC 和 DE 中的单精度数和 WRA1 中的作比

* 原文在 09F4H 处有错,把源和目的搞反了。——译者

地址	目的码	源程序		注　释

较 ★★★★★★★★★★★★★★★★★★★★★★★★

说明 两个数比较时符号必须相同。若两者符号不相同，或存储器值>寄存器的值，则为负数。

```
0A0D:  B7       OR      A          ; 测试寄存器值的指数，
0A0E:  CA5509   JP      Z, 0955H   ; 若指数(和其它几个字节)为零，则转移。
0A11:  215E09   LD      HL, 095EH  ; 从本子程序出口时的返回地址。
0A14:  E5       PUSH    HL         ; 放入堆栈。
0A15:  CD5509   CALL    0955H      ; 测试单精度数 MSB 的符号。
0A18:  79       LD      A, C       ; A=寄存器单精度数的 MSB。
0A19:  C8       RET     Z          ; 若在 4121H—4124H 中不是一个单精度数，则返回。
0A1A:  212341   LD      HL, 4123H  ; 取 WRA1 中的 MSB 地址。
0A1D:  AE       XOR     (HL)       ; 把 4123H 中的 MSB 与寄存器值的 MSB 比较。
0A1E:  79       LD      A, C       ; 重新取寄存器值的 MSB。
0A1F:  F8       RET     M          ; 若符号不同，则返回。
0A20:  CD260A   CALL    0A26H      ; 比较 BC 和 DE 中的单精度值与 4121H—4124H 中
                                   的值。
0A23:  1F       RRA                ; 比较以后取 C 标志，
0A24:  A9       XOR     C          ; 把它和寄存器的符号位合并在一起。
0A25:  C9       RET                ; 返回到调用它的程序。
0A26:  23       INC     HL         ; HL=WRA1 的指数项地址。 ★★★★★★★★★
```

说明 把 BC 和 DE 中的单精度数与 4121H—4124H 中的单精度数作比较时，两个数的符号必须相同。并不是只比较指数，而是先比较这两个数的指数，并一直比较到 LSB。一旦发现两个数中有不相同的地方，便返回，HL= 在比较时不相同处的那个字节的地址。若两个数完全相同，则返回时，HL=411FH，A=0，Z 标志置 =1。

若两个数不相等，则 C 标志 =0 时，存储器中的数=<寄存器中的数。

C 标志 =1 时，存储器中的数>寄存器中的数。

```
0A27:  78       LD      A, B       ; A=寄存器值的指数。
0A28:  BE       CP      (HL)       ; 指数作比较。
0A29:  C0       RET     NZ         ; 若不相等，返回
0A2A:  2B       DEC     HL         ; 给出 WRA1 中 MSB 的地址。
0A2B:  79       LD      A, C       ; A=寄存器值的 MSB。
0A2C:  BE       CP      (HL)       ; 比较 MSB。
0A2D:  C0       RET     NZ         ; 若不相等，返回。
0A2E:  2B       DEC     HL         ; 给出 WRA1 中的 NMSB 的地址。
0A2F:  7A       LD      A, D       ; A=寄存器值的 NMSB。
0A30:  BE       CP      (HL)       ; 比较 NMSB。
0A31:  C0       RET     NZ         ; 若不相等，返回。
0A32:  2B       DEC     HL         ; 给出 WRA1 中的 LSB 的地址。
0A33   7B       LD      A,E        ; A=寄存器值的 LSB。
0A34:  96       SUB     A, (HL)    ; 比较 LSB。
0A35:  C0       RET     NZ         ; 若不相等，则返回。
0A36:  E1       POP     HL         ; 说明数值相等，
```

地址	目的码	源程序		注　释
0A37:	E1	POP	HL	; 清除堆栈中的 095EH。
0A38:	C9	RET		; 返回到调用 0A0CH 的程序中去。
0A39:	7A	LD	A, D	; 测试符号。★★★比较整数值★★★★★★★★★★

说明 比较 HL 和 DE 中的整数值。若两者符号不等，返回时为 −1。当两者符号相同，DE>HL 时，A=−1, DE<HL 时，A=+1, DE=HL 时，A=0。

0A3A:	AC	XOR	H	; 把 D 的符号与 H 的符号作比较。
0A3B:	7C	LD	A, H	; 再准备作减法。
0A3C:	FA5F09	JP	M, 095FH	; 若符号不等,则转移,
0A3F:	BA	CP	D	; 否则比较 MSB。
0A40:	C26009	JP	NZ, 0960H	; 若不相等,则转移。
0A43:	7D	LD	A, L	; 准备比较整数的 LSB。
0A44:	93	SUB	A, E	; 比较 LSB。
0A45:	C26009	JP	NZ, 0960H	; 若不相等,则转移。
0A48:	C9	RET		; 数值相等，A=00H，返回。
0A49:	212741	LD	HL, 4127H	; WRA2 的地址。★★★比较两个双精度的数值 ★★★
0A4C:	CDD309	CALL	09D3H	; 把 DE 指定的数值移到 4127H—412EH 中。
0A4F:	112E41	LD	DE, 412EH	; 取移动后数值的指数项地址。
0A52:	1A	LD	A, (DE)	; 取指数。
0A53:	B7	OR	A	; 按指数置标志。
0A54:	CA5509	JP	Z,0955H	; 若指数为零,测试 MSB，并返回到调用它的程序。
0A57:	215E09	LD	HL, 095EH	; 把返回地址 095EH 放入堆栈,
0A5A:	E5	PUSH	HL	; 以便在 WRA1 与 WRA2 不相等时,返回到 95EH。
0A5B:	CD5509	CALL	0955H	; 测试 WRA1 是否为零。若为零，则在 0A61 处返回。
0A5E:	1B	DEC	DE	; DE=移动后数值的 MSB 地址。
0A5F:	1A	LD	A, (DE)	; 取 MSB。
0A60:	4F	LD	C, A	; 把它放入 C 中。
0A61:	C8	RET	Z	; 若 WRA1 中的 MSB 为零,则返回。
0A62:	212341	LD	HL, 4123H	; HL=当前值的 MSB 地址。
0A65:	AE	XOR	(HL)	; 把移动后数值与当前数值(即 (WRA2) 与 (WRA1)) 的符号比较。
0A66:	79	LD	A, C	; 又取回 WRA2 中数值(即移动后数值)的 MSB。
0A67:	F8	RET	M	; 若符号不相同,返回。
0A68:	13	INC	DE	; DE=移动后数值的指数地址。
0A69:	23	INC	HL	; HL=WRA1 值的指数地址。
0A6A:	0608	LD	B, 08H	; 准备对这两个数值进行比较。
0A6C:	1A	LD	A, (DE)	; 开始一个字节、一个字节地进行比较,
0A6D:	96	SUB	A, (HL)	; 比较的方法是用 WRA2 中的值减去 WRA1 中的值。
0A6E:	C2230A	JP	NZ, 0A23H	; 若不相等,则转移。
0A71:	1B	DEC	DE	; WRA2 的指针减 1。
0A72:	2B	DEC	HL	; WRA1 的指针减 1。

地址	目的码	源程序		注　释
0A73:	05	DEC	B	; 要比较的字节计数减 1。
0A74:	20F6	JR	NZ, 0A6CH	; 若所有的字节还没有比较完,则循环。
0A76:	C1	POP	BC	; 数值相等,从堆栈中清除返回地址 095EH。
0A77:	C9	RET		; 返回到调用它的程序。
0A78:	CD4F0A	CALL	0A4FH	; 把当前值与 WRA2 中的值比较。★★★比较两个双精度值 ★★★★★★★★★★★★★★★★★★★★★★
0A7B:	C25E09	JP	NZ, 095EH	; 若不相等,则置标志。
0A7E:	C9	RET		; 相等,返回时, A=00H, Z 标志=1。
0A7F:	E7	RST	20H	; 测试数据类型。★★★ CINT 程序 ★★★★★★★★
0A80:	2A2141	LD	HL, (4121H)	; HL=WRA1 中单精度值的 LSB 地址。
0A83:	F8	RET	M	; 已经是一个整数,返回。
0A84:	CAF60A	JP	Z, 0AF6H	; 若 Z 标志置位,是字符串,给出 TM 错误。
0A87:	D4B90A	CALL	NC, 0AB9H	; 若是双精度,调用 CSNG 程序。
0A8A:	21B207	LD	HL, 07B2H	; 取来 OV 错误程序的地址。
0A8D:	E5	PUSH	HL	; 有错误时的返回地址暂存起来。
0A8E:	3A2441	LD	A, (4124H)	; 取 WRA1 中当前值的指数。
0A91:	FE90	CP	90H	; 测试是否大于 16。
0A93:	300E	JR	NC, 0AA3H	; 若指数大于 16 (整数不能大于 16 位),则转移。
0A95:	CDFB0A	CALL	0AFBH	; 把一个正的单精度数转换成整数。
0A98:	EB	EX	DE, HL	; 整数从 DE 换到 HL 中。
0A99:	D1	POP	DE	; 从堆栈中清除错误返回地址,或多余的操作数。
0A9A:	222141	LD	(4121H), HL	; 把 HL 中的数值放到 WRA1 中。
0A9D:	3E02	LD	A, 02H	; 给整数标记。
0A9F:	32AF40	LD	(40AFH), A	; 置数据类型为整数。
0AA2:	C9	RET		; 返回到调用它的程序。
0AA3:	018090	LD	BC, 9080H	; BC 和 DE=-2^{16} ★★★★★★★★★★★★★★★★
0AA6:	110000	LD	DE, 0000H	; 以单精度格式表示。
0AA9:	CD0C0A	CALL	0A0CH	; 把当前数值与-2^{16}比较。
0AAC:	C0	RET	NZ	; 若数值不相等,则返回。
0AAD:	61	LD	H, C	; 若相等,则把当前值,即 -2^{16} 变成整数。
0AAE:	6A	LD	L, D	; 返回到调用它的程序。
0AAF:	18E8	JR	0A99H	; 返回时,数值为 8000H, 并置整数类型。
0AB1:	E7	RST	20H	; 测试数据类型。★★★ CSNG 程序★★★★★★★★
0AB2:	E0	RET	PO	; 已是单精度,返回。
0AB3:	FACC0A	JP	M, 0ACCH	; 若是整数,则转移。
0AB6:	CAF60A	JP	Z, 0AF6H	; 若是字符串 (Z 标志置位),则 TM 错误。
0AB9:	CDBF09	CALL	09BFH	; 把双精度数的前一半,从 WRA1 中放入 BC 和 DE 中
0ABC:	CDEF0A	CALL	0AEFH	; 把该值的数据类型标志置成单精度。
0ABF:	78	LD	A, B	; 取双精度的指数,
0AC0:	B7	OR	A	; 置标志位,测试指数。
0AC1:	C8	RET	Z	; 若 Z 标志置位,返回。

地址	目的码	源程序		注释
0AC2:	CDDF09	CALL	09DFH	; 把 WRA1 的和寄存器的 MSB 中的最高位置 1。
0AC5:	212041	LD	HL, 4120H	; HL=WRA1 中双精度值的中间地址。
0AC8:	46	LD	B, (HL)	; 取双精度值中间部分,作 LSB。
0AC9:	C39607	JP	0796H	; 把双精度值在寄存器的部分变成单精度值,并返回。
0ACC:	2A2141	LD	HL, (4121H)	; 把整数转换成单精度数。 ★★★★★★★★★★★★
0ACF:	CDEF0A	CALL	0AEFH	; 把 WRA1 中的值标志置成单精度。
0AD2:	7C	LD	A, H	; A=整数的 MSB。
0AD3:	55	LD	D, L	; D=整数的 LSB。
0AD4:	1E00	LD	E, 00H	; E=0 (值的其余部分)。
0AD6:	0690	LD	B, 90H	; B=初始的最大指数。
0AD8:	C36909	JP	0969H	; 进行归一化处理,再返回到调用它的程序。
0ADB:	E7	RST	20H	; 测试数据类型。★★★ 把整数或单精度数转换成双精度数 ★★★★★★★★★★★★★★★★★★★★★★
0ADC:	D0	RET	NC	; 已是双精度数,返回。
0ADD:	CAF60A	JP	Z, 0AF6H	; 若是字符串,则转移。
0AE0:	FCCC0A	CALL	M, 0ACCH	; 若是整数,则调用子程序,将整数转换成单精度。
0AE3:	210000	LD	HL, 0000H	; 把 WRA1 中的后四个字节充以零。
0AE6:	221D41	LD	(411DH), HL	; 这是双精度数值中的后四个字节。
0AE9:	221F41	LD	(411FH), HL	;
0AEC:	3E08	LD	A, 08H	; 双精度标志。
0AEE:	013E04	LD	BC, 043EH	; OAEF: LD A, 04 单精度标志。
0AF1:	C39F0A	JP	0A9FH	; 把 A 放入数据类型中,然后返回。
0AF4:	E7	RST	20H	; 测试数据类型。 ★★★★★★★★★★★★★★★★
0AF5:	C8	RET	Z	; 若是字符串,不带错误信息返回。
0AF6:	1E18	LD	E, 18H	; 若不是字符串,则 TM 错误。
0AF8:	C3A219	JP	19A2H	; 输出 TM 错误信息。
0AFB:	47	LD	B, A	; 把单精度正数变成整数。 ★★★★★★★★★★★★
0AFC;	4F	LD	C, A	; 把指数从 A 移到 BC,
0AFD:	57	LD	D, A	; D
0AFE:	5F	LD	E, A	; 和 E 中。
0AFF:	B7	OR	A	; 测试指数值。
0B00:	C8	RET	Z	; 若数值为零,则返回。
0B01:	E5	PUSH	HL	; 保存错误返回地址。
0B02:	CDBF09	CALL	09BFH	; 把 WRA1 中的单精度值放入 BC 和 DE 中。
0B05:	CDDF09	CALL	09DFH	; 准备把当前值和寄存器值进行算术运算
0B08:	AE	XOR	(HL)	; (把数值的最高位置 1, 并比较符号相同否)。
0B09:	67	LD	H, A	; H 是数值的符号,当正时位 7=0; 当负时位 7=1。
0B0A:	FC1F0B	CALL	M, 0B1FH	; 若负数,则转移。
0B0D:	3E98	LD	A, 98H	; A=允许的最大指数值。
0B0F:	90	SUB	A, B	; 指数一偏移值=要得到整数而右移的位数。
0B10:	CDD707	CALL	07D7H	; 在 CDE 中得到向右对齐的整数。

地址	目的码	源程序		注释
0B13:	7C	LD	A, H	; A=原始符号,若正数时,位 7 为 0;若负数时,位 7 为 1。
0B14:	17	RLA		; 把符号移入 C 标志。
0B15:	DCA807	CALL	C, 07A8H	; 若是负数,则把跟在后面的 1 变成 0。
0B18:	0600	LD	B, 00H	; 指数为 0。
0B1A:	DCC307	CALL	C, 07C3H	; 若是负数,则把它变成为负整数。
0B1D:	E1	POP	HL	; 恢复调用它程序的 HL。
0B1E:	C9	RET		; 返回到调用它的程序。
0B1F:	1B	DEC	DE	; 把单精度数的 NMSB 和 LSB 减 1。★★★把单精度数舍入 ★★★★★★★★★★★★★★★★★★★★★★
0B20:	7A	LD	A, D	; 将新的 NMSB 和 LSB 合并。
0B21:	A3	AND	E	; 若它们是零,则结果为 FFFFH。
0B22:	3C	INC	A	; 测试是否为 FFFFH (即 LSB 和 NMSB 都为零)。
0B23:	C0	RET	NZ	; 若它们不为零,则返回,
0B24:	0B	DEC	BC	; 否则 MSB 减 1。
0B25:	C9	RET		; 然后返回。
0B26:	E7	RST	20H	; 确定数据类型。★★★ FIX 程序★★★★★★★★★
0B27:	F8	RET	M	; 若是整数便返回。
0B28:	CD5509	CALL	0955H	; 测试 WRA1 中浮点数值的符号。
0B2B:	F2370B	JP	P, 0B37H	; 若是正数,则转移。
0B2E:	CD8209	CALL	0982H	; 把该值的符号位清除(使它变正)。
0B31:	CD370B	CALL	0B37H	; 把单精度或双精度值变成整数,不四舍五入。
0B34:	C37B09	JP	097BH	; 把数的整数部分再变成单精度或双精度值,然后返回。
0B37:	E7	RST	20H	; 把单精度或双精度数变成整数,决定数据类型。★★★
0B38:	F8	RET	M	; 已是整数,返回。
0B39;	301E	JR	NC, 0B59H	; 若是双精度数,则转移。
0B3B:	28B9	JR	Z, 0AF6H	; 若 Z 标志置 1 (是字符串),则 TM 错误。
0B3D:	CD8E0A	CALL	0A8EH	; 把单精度数转换成整数,返回到调用它的程序。
0B40:	212441	LD	HL, 4124H	; HL←WRA1 中单精度数的地址。

说明 把单精度数的整数部分分离出来,把整数部分放在 A 寄存器中。再把整数转换成单精度数,并把它(单精度数)放在 WRA1 中,若是以 WRA1 中的双精数来调子转换时,则返回时, C 标志不置位。

0B43:	7E	LD	A, (HL)	; A=该值的指数。
0B44:	FE98	CP	98H	; 测试整数部分是否超过 16 位。
0B46:	3A2141	LD	A, (4121H)	; A=该值的 LSB。
0B49:	D0	RET	NC	; 若整数部分超过 16 位,返回。
0B4A:	7E	LD	A, (HL)	; A=指数。
0B4B:	CDFB0A	CALL	0AFBH	; 把单精度数变成整数,这样,给出了整数。
0B4E:	3698	LD	(HL), 98H	;
0B50:	7B	LD	A, E	; 再把它变成单精度数。
0B51:	F5	PUSH	AF	; 把整数值移 8 位,
0B52:	79	LD	A, C	; 把 E 放到 A 中,再放入堆栈中,
0B53:	17	RLA		; 然后把 C 的位 7 (符号位)放入 C 标志中。

地址	目的码	源程序		注　释
0B54:	CD6207	CALL	0762H	; 把数值归一化,并调整指数。
0B57:	F1	POP	AF	; 取回整数值。
0B58:	C9	RET		; 返回到调用它的程序。
0B59:	212441	LD	HL, 4124H	; 把双精度数转换成整数。 ★★★★★★★★★★★★
0B5C:	7E	LD	A, (HL)	; 取指数。
0B5D:	FE90	CP	90H	; 与允许的最大整数的指数(偏移)值比较。
0B5F:	DA7F0A	JP	C, 0A7FH	; 若数小于 16 位精度,则转移,将使用单精度转换成整数的子程序。
0B62:	2014	JR	NZ, 0B78H	; 若数大于 16 位精度,则转移。
0B64:	4F	LD	C, A	; C=指数=90H, 数将是 16 位整数。
0B65:	2B	DEC	HL	; 退回一个字节,指向 WRA1 中的 MSB。
0B66:	7E	LD	A, (HL)	; A=MSB 字节。
0B67:	EE80	XOR	80H	; 把 MSB 中的符号位取反。
0B69:	0606	LD	B, 06H	; 测试是否为零。若 A 和以后几个字节累加的和为零,
0B6B:	2B	DEC	HL	; 则数值是负零。
0B6C:	B6	OR	(HL)	; 指向双精度数的下一个字节。
0B6D:	05	DEC	B	; 测试所有的字节。
0B6E:	20FB	JR	NZ, 0B6BH	; 没有测试完,继续循环。
0B70:	B7	OR	A	; 对双精度值的全部字节经 OR 运算,置状态标志。
0B71:	210080	LD	HL, 8000H	; HL=−0 (整数)
0B74:	CA9A0A	JP	Z, 0A9AH	; 以 −0 值返回,给出整数类型,再返回到调用它的程序。
0B77:	79	LD	A, C	; 双精度的指数放入 A 寄存器。
0B78:	FEB8	CP	B8H	; 与 56 比较。
0B7A:	D0	RET	NC	; 若双精度数超过 56 位,则错误。
0B7B:	F5	PUSH	AF	; 暂存指数。
0B7C:	CDBF09	CALL	09BFH	; 把双精度数的前一半放在 BC 和 DE 中。
0B7F:	CDDF09	CALL	09DFH	; 把数值的最高位置 1。确定结果的符号。
0B82:	AE	XOR	(HL)	; 测试数值的符号,若正数状态为正,否则状态为负。
0B83:	2B	DEC	HL	; HL=4124H, WRA1 中数值的指数地址*。
0B84:	36B8	LD	(HL), B8H	; 把最大的指数值放入指数单元。
0B86:	F5	PUSH	AF	; 把数值的符号位暂存。
0B87:	FCA00B	CALL	M, 0BA0H	; 若是负数,把后几个字节的 1 变成 0。
0B8A:	212341	LD	HL, 4123H	; HL=双精度数值 MSB 的地址。
0B8D:	3EB8	LD	A, B8H	; A=双精度数的最大指数。
0B8F:	90	SUB	A, B	; 减去当前值的指数,给出了求整数的右移位数。
0B90:	CD690D	CALL	0D69H	; 把数值分解,并向右对齐。
0B93:	F1	POP	AF	; 取回符号位。
0B94:	FC200D	CALL	M, 0D20H	; 若是负数,把后几个字节的 0 变成 1。
0B97:	AF	XOR	A	; 清除 A 寄存器。

* 从 09DFH 子程序返回时,HL = 4125H = 符号位字节,所以经 DEC HL 指令得 4124H,存放指数的单元,而 4123H 是存放 MSB 的单元。原文中有误。——译者

地址	目的码	源程序		注释
0B98:	321C41	LD	(411CH), A	; 放回尾数的符号。
0B9B:	F1	POP	AF	; 取回原指数。
0B9C:	D0	RET	NC	; 若尾数大于 56 位,错误。
0B9D:	C3D80C	JP	0CD8H	; 将结果作归一化处理,并返回。
0BA0:	211D41	LD	HL, 411DH	; ★★★把双精度负数后几个字节的 1 转变成 0 ★★★ HL=双精度值 LSB 的地址。
0BA3:	7E	LD	A, (HL)	; 从指针处取一个字节。
0BA4:	35	DEC	(HL)	; 指针处的内容减 1。
0BA5:	B7	OR	A	; 测试原取来的字节。
0BA6:	23	INC	HL	; 指针指向下一个字节。
0BA7:	28FA	JR	Z, 0BA3H	; 当内容为零时循环。
0BA9:	C9	RET		; 返回到调用它的程序。
0BAA:	E5	PUSH	HL	; 保存调用程序的 HL。★★★ 两个在 BC 和 DE 中的 16 位数二进制相乘 ★★★★★★★★★★★★★★★★

说明 乘积是留在 DE 中,是以移位相加的方法来进行乘法的。将在 BASIC 计算下标变量地址时,调用本程序。

0BAB:	210000	LD	HL, 0000H	; 累加器清零。
0BAE:	78	LD	A, B	; 测试在 BC 中的值。
0BAF:	B1	OR	C	; 若是零,则把零放入 DE, 并返回。
0BB0:	2812	JR	Z, 0BC4H	; 若 BC=0, 则转移。
0BB2:	3E10	LD	A, 10H	; A=16=左移的次数。
0BB4:	29	ADD	HL, HL	; 把结果左移一位。
0BB5:	DA3D27	JP	C, 273DH	; 若 C 标志置位,则 BS 错误。
0BB8:	EB	EX	DE, HL	; 准备将被乘数左移一位。
0BB9:	29	ADD	HL, HL	; 移一位,
0BBA:	EB	EX	DE, HL	; 并将它又放到 DE 中。
0BBB:	3004	JR	NC, 0BC1H	; 若 C 标志没有置 1, 就表示没有找到一个 1,所以不要相加,
0BBD:	09	ADD	HL, BC	; 否则把乘数加到目前为止的结果上。
0BBE:	DA3D27	JP	C, 273DH	; 若 C 标志置 1, 则 BS 错误。
0BC1:	3D	DEC	A	; 要移 16 次,是否移完?
0BC2:	20F0	JR	NZ, 0BB4H	; 否,继续循环。
0BC4:	EB	EX	DE, HL	; 把结果放入 DE 中。
0BC5:	E1	POP	HL	; 恢复调用它程序的 HL。
0BC6:	C9	RET		; 返回到调用它的程序。
0BC7:	7C	LD	A, H	; 测试 HL 中数值的符号。★★★两个在 HL 和 DE 中十六位数二进制减 ★★★★★★★★★★★★★★★★
0BC8:	17	RLA		; 把符号留在 B 中。
0BC9:	9F	SBC	A, A	; 若 HL 为正 B=0, 若 HL 为负 B=—1。
0BCA:	47	LD	B, A	; 把符号位移入 B 中。
0BCB:	CD510C	CALL	0C51H	; HL 求反码,然后使用加法的程序。

地址	目的码	源程序		注　释
0BCE:	79	LD	A, C	; 把 A 清零,置 A 为差值的符号,若 HL 为正,
0BCF:	98	SBC	A, B	; 则 A=+0, 若 HL 为负,则 A=−1。
0BD0:	1803	JR	0BD5H	; 使用加法程序,若结果太小而下溢,则转换成单精度数。
0BD2	7C	LD	A, H	; B=HL 的符号。★★★ 两个在 HL 和 DE 中的整数二进制相加 ★★★★★★★★★★★★★★★★★★★

说明 两个在 HL 和 DE 中的整数相加,结果留在 HL 中。若求和结果溢出，则把两个数先转换成单精度数,然后再相加。确定溢出的方法如下: C=求和后的C标志, C=0 表示没有溢出, C=1, 则若 A=0, B=0 溢出; 若 A<>B, 是负数。

0BD3:	17	RLA		; 符号进入 C 标志。
0BD4:	9F	SBC	A, A	; 若 HL 为正 B=0, 否则 B=−1。
0BD5:	47	LD	B, A	; 把符号位放入 B 中。
0BD6:	E5	PUSH	HL	; 暂存 HL, 以便在必须把它转换成单精度时使用。
0BD7:	7A	LD	A, D	; 取寄存器的值的 MSB, 以测试它的符号。
0BD8:	17	RLA		; 置 A=DE 的符号,若 DE 为正,则 A=0,
0BD9:	9F	SBC	A, A	; 否则 A=−1。
0BDA:	19	ADD	HL, DE	; 两个寄存器相加,
0BDB:	88	ADC	A, B	; 结果的符号与两个符号之和再求和。
0BDC:	0F	RRCA		; 结果的符号放入位 7,
0BDD:	AC	XOR	H	; 并把它与 HL 的符号合并在一起。
0BDE:	F2990A	JP	P, 0A99H	; 没有溢出,说明结果是整数,存在 4121H 中, 并返回。
0BE1:	C5	PUSH	BC	; 把符号状态放入堆栈。
0BE2:	EB	EX	DE, HL	; 为了转换,把原 DE 和 HL 交换。
0BE3:	CDCF0A	CALL	0ACFH	; 把原 DE 的值转换成单精度,放入 4121H—4124H 中。
0BE6:	F1	POP	AF	; 清除堆栈。
0BE7:	E1	POP	HL	; 取回原 HL 的值。在以上加法中已把原 HL 的值冲掉了。
0BE8:	CDA409	CALL	09A4H	; 把转换后的 DE 值(已在 WRA1 中)放到堆栈中。
0BEB:	EB	EX	DE, HL	; 恢复 HL。
0BEC:	CD6B0C	CALL	0C6BH	; 把 HL 中的值变成单精度。
0BEF:	C38F0F	JP	0F8FH	; HL 和 DE 中的值变成单精度数值后相加。
0BF2:	7C	LD	A, H	; 测试 HL 的数值。★★★整数乘法程序★★★★★★

说明 整数相乘: DE=第一个数值, HL=第二个数值。计算结果留在 HL 中。若两个数的符号相同,则其乘积为正号。若两个数的符号不同,其乘积是负号。在乘法运算之前,先要把负数变成正数。乘法中使用的方法是移位和相加。在移位 DE 时, 每找到一个 1, 便把 HL 中的原数值加到累加寄存器中(这里还是用 HL 寄存器对),并左移 1 位。这样重复 16 次(一定要把 DE 中的 16 位都测试一遍)。若产生溢出,要把两个数都变成单精度数,然后调用单精度数的乘法程序。

0BF3:	B5	OR	L	; 若是 0,
0BF4:	CA9A0A	JP	Z, 0A9AH	; 则在 HL 中的结果是 0, 返回。
0BF7:	E5	PUSH	HL	; 保存原始数值,
0BF8:	D5	PUSH	DE	; 以便在必要时可转换成单精度数。
0BF9:	CD450C	CALL	0C45H	; 置结果为单精度类型,把负数变成正数。

地址	目的码	源程序		注　释
0BFC:	C5	PUSH	BC	; BC=结果的符号,放入堆栈。
0BFD:	44	LD	B, H	; B=第二个数值的 MSB。
0BFE:	4D	LD	C, L	; BC=第二个数值。
0BFF:	210000	LD	HL, 0000H	; HL=累加器。
0C02:	3E10	LD	A, 10H	; 左移的次数。
0C04:	29	ADD	HL, HL	; 把结果移位,并测试是否溢出。
0C05:	381F	JR	C, 0C26H	; 若溢出,则 C 标志置位。
0C07:	EB	EX	DE, HL	; 没有溢出,则 DE 左移一位。
0C08:	29	ADD	HL, HL	; 并测试是否为二进制的 1,
0C09:	EB	EX	DE, HL	; 在 C 标志中。
0C0A:	3004	JR	NC, 0C10H	; 若 C 标志为 0, 则没有二进制 1, 转移。
0C0C:	09	ADD	HL, BC	; 把原 HL 中的值加到累加器中。
0C0D:	DA260C	JP	C, 0C26H	; 对 DE 中每一个 1, 都加一次。
0C10:	3D	DEC	A	; DE 是否移了 16 次?
0C11:	20F1	JR	NZ, 0C04H	; 否,继续循环。
0C13:	C1	POP	BC	; 是,取乘积的符号。
0C14:	D1	POP	DE	; 在 DE 中的原始值。
0C15:	7C	LD	A, H	; 测试结果的符号。
0C16:	B7	OR	A	; 根据结果置标志。
0C17:	FA1F0C	JP	M, 0C1FH	; 若结果为负值,则转移(可能溢出了)。
0C1A:	D1	POP	DE	; 清除堆栈。
0C1B:	78	LD	A, B	; 把结果的符号放入 A 中。
0C1C:	C34D0C	JP	0C4DH	; 把 HL 变为相应的符号,存储结果,并返回调用它的程序。
0C1F:	EE80	XOR	80H	; 清除符号位,测试数值的其余部分是否为零。★★★★
0C21:	B5	OR	L	; 若为 0, 这是一个负数,
0C22:	2813	JR	Z, 0C37H	; 否则把它转换成单精度数。
0C24:	EB	EX	DE, HL	; 0C26: POP BC 清除结果的符号位, 数溢出, 转换成单精度后再乘。
0C25:	01C1E1	LD	BC, E1C1H	; 0C27: POP HL 取回原 HL 的数值。
0C28:	CDCF0A	CALL	0ACFH	; 把原 HL 的值变成单精度。
0C2B:	E1	POP	HL	; HL=原 DE 的值。
0C2C:	CDA409	CALL	09A4H	; 转换后的 HL 放入堆栈。
0C2F:	CDCF0A	CALL	0ACFH	; 将原 DE 值(现在在 HL 中)转换为单精度值。
0C32:	C1	POP	BC	; 从堆栈中取出转换后的 HL 值,
0C33:	D1	POP	DE	; 放入 BC 和 DE 中。
0C34:	C34708	JP	0847H	; 进行单精度相乘。
0C37:	78	LD	A, B	; 取结果的符号状态。★★★★★★★★★★★★★★
0C38:	B7	OR	A	; 把状态标志作为结果的符号。
0C39:	C1	POP	BC	; 清除堆栈中的返回地址。
0C3A:	FA9A0A	JP	M, 0A9AH	; 若假设的符号为负,则转移。

地址	目的码	源程序		注　释
0C3D:	D5	PUSH	DE	; 保存原 DE 值。
0C3E:	CDCF0A	CALL	0ACFH	; 把结果变成单精度。
0C41:	D1	POP	DE	; 取回原 DE 值。
0C42:	C38209	JP	0982H	; 取回符号,并返回到调用它的程序。
0C45:	7C	LD	A, H	; 取第二个操作数中 MSB 的符号。★★★若 HL 为负,取反码,若 DE 为负,也取反码★★
0C46:	AA	XOR	D	; 与第一个操作数的符号合并在一起。
0C47:	47	LD	B, A	; 若两者符号相等(++)或(--),则 B 为+,当两者不同时, B 为-。
0C48:	CD4C0C	CALL	0C4CH	; 测试 HL 操作数的符号,若为负,转换成正。
0C4B:	EB	EX	DE, HL	; 交换 HL 和 DE,于是可测试 DE 的符号,若是负数,则转为正。
0C4C:	7C	LD	A, H	; 取 DE 值的符号字节。
0C4D:	B7	OR	A	; 按 DE 值的符号来置标志。
0C4E:	F29A0A	JP	P, 0A9AH	; 置整数类型,结果在 4121H 中,返回到调用它的程序。
0C51:	AF	XOR	A	; 清除 A 寄存器和 C 标志。
0C52:	4F	LD	C, A	; C 寄存器清零。
0C53:	95	SUB	A, L	; LSB 取反码,
0C54:	6F	LD	L, A	; 放回到 L 寄存器。
0C55:	79	LD	A, C	; A 寄存器清零。
0C56:	9C	SBC	A, H	; MSB 取反码。
0C57:	67	LD	H, A	; 放回到 H 寄存器
0C58:	C39A0A	JP	0A9AH	; 置数据类型为整数(02), 把数值放入 4121H—4122H 中,返回。
0C5B:	2A2141	LD	HL, (4121H)	; 取一个二进制表示的整数 ★★★★★★★★★★★★
0C5E:	CD510C	CALL	0C51H	; 转换成正数,
0C61:	7C	LD	A, H	, 测试数值是否小于 2^{15}。
0C62:	EE80	XOR	80H	; 若位 15 不是零,而其它位都为零,
0C64:	B5	OR	L	; 则数值大于 2^{15}。
0C65:	C0	RET	NZ	; 若整数=或 <32768, 则返回
0C66:	EB	EX	DE, HL	, 数值大于 2^{15} , 把它放入 DE 中。
0C67:	CDEF0A	CALL	0AEFH	; 置单精度标志。
0C6A:	AF	XOR	A	; 置指数为零。
0C6B:	0698	LD	B, 98H	; 单精度值的最大指数。
0C6D:	C36909	JP	0969H	; 把数值转换成单精度值,返回到调用它的程序。
0C70:	212D41	LD	HL, 412DH	, 取 WRA2 双精度值地址。★★★双精度减程序★★★
0C73:	7E	LD	A, (HL)	; 取 WRA2 数值的 MSB。
0C74:	EE80	XOR	80H	; 改变符号。
0C76:	77	LD	(HL), A	; 放回到原存储单元。
0C77:	212E41	LD	HL, 412EH	; HL=WRA2 指数的地址。★★★双精度相加程序★★

把一个负整数转换成反码,为正数。

说明　是当前值与保存值相加,即 WRA1 与 WRA2 中的值相加。

地址	目的码	源程序		注　释
0C7A:	7E	LD	A, (HL)	; 取 WRA2 中的指数,
0C7B:	B7	OR	A	; 置指数标志。
0C7C:	C8	RET	Z	; 若 WRA2 中的数为零,便返回。
0C7D:	47	LD	B, A	; B=WRA2 数值的指数。
0C7E:	2B	DEC	HL	; 退回去指向 WRA2 的 MSB。
0C7F:	4E	LD	C, (HL)	; C=WRA2 的 MSB 数值。
0C80:	112441	LD	DE, 4124H	; DE=WRA1 中数值的指数地址。
0C83:	1A	LD	A, (DE)	; 取 WRA1 中值的指数。
0C84:	B7	OR	A	; 置指数标志。
0C85:	CAF409	JP	Z, 09F4H	; 若 WRA1 值为零,则转移。
0C88:	90	SUB	A, B	; 否则比较指数 WRA1—WRA2。
0C89:	3016	JR	NC, 0CA1H	; 若 WRA1 的指数 >WRA2 的指数,则转移。整数部分位数少的指数也小。
0C8B:	2F	CPL		; 使指数的差值变成正数。
0C8C:	3C	INC	A	; WRA2 的值大于 WRA1 的值,
0C8D:	F5	PUSH	AF	; 保存指数的差值。
0C8E:	0E08	LD	C, 08H	; 交换这两个数,于是 WRA1←WRA2。
0C90:	23	INC	HL	; 并反之, WRA1=WRA2。
0C91:	E5	PUSH	HL	; HL=WRA2 的指数地址。
0C92:	1A	LD	A, (DE)	; 从 WRA1 中取一个字节。★★★交换 WRA1 和 WRA2 中的双精度数 ★★★★★★★★★★★★★★★★★★
0C93:	46	LD	B, (HL)	; 从 WRA2 中取一个字节。
0C94:	77	LD	(HL), A	; 把 WRA1 的字节放入 WRA2 中。
0C95:	78	LD	A, B	; 把 WRA2 的字节放入 WRA1 中。
0C96:	12	LD	(DE), A	; 使大的一个数放在 WRA1 中。
0C97:	1B	DEC	DE	; WRA1 的地址减 1,
0C98:	2B	DEC	HL	; WRA2 的地址减 1,
0C99:	0D	DEC	C	; 字节移动计数计 1。
0C9A:	20F6	JR	NZ, 0C92H	; 若还没有把工作区中双精度数的 8 个字节都移到,便循环。
0C9C:	E1	POP	HL	; 把 WRA2 的地址放回到 HL 中。
0C9D:	46	LD	B, (HL)	; B=新的 WRA2 中值的指数。
0C9E:	2B	DEC	HL	; HL=WRA2 的 MSB 地址。
0C9F:	4E	LD	C, (HL)	; C=WRA2 数的 MSB。
0CA0:	F1	POP	AF	; A=指数的差值。
0CA1:	FE39	CP	39H	; 指数的差值是否大于 56?
0CA3:	D0	RET	NC	; 若指数的差值大于 56, 则返回。
0CA4:	F5	PUSH	AF	; 把指数的差值暂存起来。
0CA5:	CDDF09	CALL	09DFH	; 把 WRA1 中的 MSB 的最高位置 1。
0CA8:	23	INC	HL	; 在归一化时, HL=充以零的字节的地址。
0CA9:	3600	LD	(HL), 00H	; 然后把它置零。

地址	目的码	源程序		注　释
0CAB:	47	LD	B, A	; 把 WRA2 的符号保存起来。
0CAC:	F1	POP	AF	; 取回指数的差值。
0CAD:	212D41	LD	HL, 412DH	; HL=WRA2 中 MSB 的地址。
0CB0:	CD690D	CALL	0D69H	; 按比例换算该值,使两者指数相等。
0CB3:	3A2641	LD	A, (4126H)	; 这样两个数才能相加。
0CB6:	321C41	LD	(411CH), A	; 取右移 WRA2 时刚移出去的 8 位。
0CB9:	78	LD	A, B	; 取 WRA2 值的符号位。
0CBA:	B7	OR	A	; 按 WRA2 的符号置标志。
0CBB:	F2CF0C	JP	P, 0CCFH	; 符号不同,必须相减。
0CBE:	CD330D	CALL	0D33H	; 把 4127H—412DH 中的双精度数加到 411D—4123H 中。
0CC1:	D200D	JP	NC, 0D0EH	; 若 C 标志没有置位,则调整结果的符号,并返回。
0CC4:	EB	EX	DE, HL	; C 标志置位,把当前值的指数加 1。
0CC5:	34	INC	(HL)	; 若溢出,则有错误。
0CC6:	CAB207	JP	Z, 07BH	; 转移去给出 OV 信息。
0CC9:	CD900D	CALL	0D90H	; 把结果右移一位,
0CCC:	C30E0D	JP	0D0EH	; 调整结果的符号,并返回。
0CCF:	CD450D	CALL	0D45H	; 求两个数的差值。★★★ WRA1—WRA2, 差留在 WRA1中★★★★★★★★★★★★★★★★★★★★

说明 在求出差值后,对差值归一化。测试 MSB, 若 MSB 为零,整个数值左移一个字节;若不为零, 把它每次左移一位,直到 MSB 的位 7 为 1。

0CD2:	212541	LD	HL, 4125H	; HL=结果的符号标志。
0CD5:	DC570D	CALL	C, 0D57H	; 若 C 标志置位,则取差的反码。
0CD8:	AF	XOR	A	; 预置计数的初始值。
0CD9:	47	LD	B, A	; B 清零,为以下的归一化循环作准备。
0CDA:	3A2341	LD	A, (4123H)	; 取 MSB。
0CDD:	B7	OR	A	; 测试是否为零。
0CDE:	201E	JR	NZ, 0CFEH	; 若不是零, 转移去进行左移, 直到位 7 为 1。否则将整个数从 LSB 起向指数方向左移一个字节。
0CE0:	211C41	LD	HL, 411CH	; HL=在 WRA1 中双精度数 LSB 的地址 −1。
0CE3:	0E08	LD	C, 08H	; C=要移位的字节数。
0CE5:	56	LD	D, (HL)	; 取要移的下一个字节。
0CE6:	77	LD	(HL), A	; 把现在的字节保存起来。
0CE7:	7A	LD	A, D	; 把要移的字节暂存在 A 中, 以便放在下一个地址单元中。
0CE8:	23	INC	HL	; 指向 WRA1 中的下一个字节。
0CE9:	0D	DEC	C	; 是否把双精度数整个左移一个字节?
0CEA:	20F9	JR	NZ, 0CE5H	; 否,循环。
0CEC:	78	LD	A, B	; 是,当数值是零时,不再循环了。
0CED:	D608	SUB	08H	; 是否把 LSB 移过了 8 个字节,

地址	目的码	源程序		注　释
0CEF:	FEC0	CP	C0H	; 移到了指数的位置?
0CF1:	20E6	JR	NZ, 0CD9H	; 否,继续寻找非零的 MSB。
0CF3:	C37807	JP	0778H	; 是,将指数清零,并返回。
0CF6:	05	DEC	B	; 保持左移的位数和字节数的计数。
0CF7:	211C41	LD	HL, 411CH	; 要左移一位的 8 字节数的 LSB 地址。
0CFA:	CD970D	CALL	0D97H	; 将数左移一位(移位后放回原处)。
0CFD:	B7	OR	A	; 测试 MSB 的位 7。
0CFE:	F2F60C	JP	P, 0CF6H	; 继续移位,直到位 7 为 1。
0D01:	78	LD	A, B	; 测试左移位数的计数。
0D02:	B7	OR	A	; 置计数的标志。
0D03:	2809	JR	Z, 0D0EH	; 若数值已归一化,则转移。
0D05:	212441	LD	HL, 4124H	; HL=指数的地址。
0D08:	86	ADD	A, (HL)	; 加左移的位数,得到偏移值。
0D09:	77	LD	(HL), A	; 存储新的指数。
0D0A:	D27807	JP	NC, 0778H	; 若不溢出,则置指数为零。
0D0D:	C8	RET	Z	; 返回到调用它的程序。
0D0E:	3A1C41	LD	A, (411CH)	; 取当前值的 MSB。
0D11:	B7	OR	A	; 置状态标志。
0D12:	FC200D	CALL	M, 0D20H	; 若数值为负,则把后几个字节的 0 恢复成 1。
0D15:	212541	LD	HL, 4125H	; 取结果的符号。
0D18:	7E	LD	A, (HL)	; 放入 A 寄存器中。
0D19:	E680	AND	80H	; 分离出结果的符号。
0D1B:	2B	DEC	HL	; 指向尾数的符号。
0D1C:	2B	DEC	HL	; 故 HL=HL−2。
0D1D:	AE	XOR	(HL)	; 把结果的符号置成尾数的符号。
0D1E:	77	LD	(HL), A	; 以正确的符号放回 MSB 中。
0D1F:	C9	RET		; 返回到调用它的程序。
0D20:	211D41	LD	HL, 411DH	; HL=WRA1 中双精度数 LSB 的地址。 ★★★★★★

说明　把 1 与 WRA1 中的双精度数相加。从 LSB 加 1 开始，若溢出(其和为 0)，那末把 C 标志与下一个字节相加,这样重复下去。若指数产生溢出,则整个数便溢出。

0D23:	0607	LD	B, 07H	; 用于双精度值加 1 的字节计数。
0D25:	34	INC	(HL)	; LSB 加 1。
0D26:	C0	RET	NZ	: 若不溢出,返回。
0D27:	23	INC	HL	; 否则, C 标志加下一个字节。
0D28:	05	DEC	B	;
0D29:	20FA	JR	NZ, 0D25H	; 直至不发生溢出。
0D2B:	34	INC	(HL)	; 指数加 1。
0D2C:	CAB207	JP	Z, 07B2H	; OV 错误。
0D2F:	2B	DEC	HL	; 数已变成负数。
0D30:	3680	LD	(HL), 80H	; 重新设置 MSB=80H, 其它几个字节=00H。
0D32:	C9	RET		; 返回调用它的程序。

地址	目的码	源程序		注　释
0D33:	212741	LD	HL, 4127H	; 加数的地址。★★★★★★★★★★★★★★★★★★★★

说明 两个双精度数求和。只把这两个数的尾数相加，而指数不相加。第一个数的地址在 DE 中，第二个数的地址在 HL 中，而结果还是存放在由 HL 指针给出的单元中，代替了原数。

0D36:	111D41	LD	DE, 411DE	; 被加数的地址。
0D39:	0E07	LD	C, 07H	, 要加的字节数。
0D3B:	AF	XOR	A	; 清除 C 标志。
0D3C:	1A	LD	A, (DE)	; 开始相加。
0D3D:	8E	ADC	A, (HL)	, 从 LSB 开始，
0D3E:	12	LD	(DE), A	; 一直加到 MSB，
0D3F:	13	INC	DE	; 把结果放入 WRA1 (411DH—4124H) 中。
0D40:	23	INC	HL	; 在相加之前，数值必须进行分解，
0D41:	0D	DEC	C	; 计数已加了一个字节。
0D42:	20F8	JR	NZ, 0D3CH	; 循环，直到所有的字节都加完。
0D44:	C9	RET		; 返回到调用它的程序。
0D45:	212741	LD	HL, 4127H	; ★★★两个双精度数相减。从 411DH—4123H 的内容中减去 ★★★★ 4127H—412DH，结果放在 411DH—4123H 中*★★★★★★★★★★★★★★★★★★★★ WRA2 的首址。
0D48:	111D41	LD	DE, 411DH	; WRA1 的首址。
0D4B:	0E07	LD	C, 07H	; 要减的的字节数
0D4D:	AF	XOR	A	; 消除 C 标志。
0D4E:	1A	LD	A, (DE)	; 取 WRA1 的 LSB。
0D4F:	9E	SBC	A, (HL)	; 减去 WRA2 的 LSB。
0D50:	12	LD	(DE), A	; 结果放入 WRA1 中。
0D51:	13	INC	DE	; WRA1 和 WRA2 的地址加 1。
0D52:	23	INC	HL	;
0D53:	0D	DEC	C	; 减的字节计数将计 1。
0D54:	20F8	JR	NZ, 0D4EH	; 循环，直到所有字节都减完。
0D56:	C9	RET		; 返回到调用它的程序。
0D57:	7E	LD	A, (HL)	; 符号标志放 A 中。★★★ WRA1 中的正双精度数求反程序 ★★★★★★★★★★★★★★★★★★★★
0D58:	2F	CPL		; 求反。
0D59:	77	LD	(HL), A	; 放回符号标志。
0D5A:	211C41	LD	HL, 411CH	; HL＝当前双精度数的 LSB 地址。
0D5D:	0608	LD	B, 08H	, 求反的字节数。
0D5F:	AF	XOR	A	; A 清零，并清除 C 标志。
0D60:	4F	LD	C, A	; 把零暂存起来，以便取回。
0D61:	79	LD	A, C	; 取回零值，C 标志保持不变。
0D62:	9E	SBC	A, (HL)	; 把一个字节求反。

* 原文说明是 WRA2—WRA1→WRA1。——译者

地址	目的码	源程序		注　释
0D63:	77	LD	(HL), A	; 把它存放回去。
0D64:	23	INC	HL	; 指向下一个字节的地址。
0D65:	05	DEC	B	; 8个字节是否都进行了求反?
0D66:	20F9	JR	NZ, 0D61H	; 否,继续循环。
0D68:	C9	RET		; 是,返回到调用它的程序。
0D69:	71	LD	(HL), C	; 存储 MSB。

说明 把一个双精度数进行分解，该数放在 HL 指针给的存储单元中(开始时 HL 指向 MSB)。C=MSB，A=右移的位数。将数值进行右移,先是一次移一个字节，直到移位计数 <0，然后再一次移一位地移动。

0D6A:	E5	PUSH	HL	; 把数值的开始地址保存起来,数值从 MSB 开始。
0D6B:	D608	SUB	08H	; 移位计数是否≥8?
0D6D:	380E	JR	C, 0D7DH	; 否,转移到进行移位的程序。
0D6F:	E1	POP	HL	; HL 恢复为数据的开始地址。
0D70:	E5	PUSH	HL	; 再暂存数据的开始地址。
0D71:	110008	LD	DE, 0800H	; D=要移动(向右移一个字节)的字节数。
0D74:	4E	LD	C, (HL)	; 把数据右移一个字节，
0D75:	73	LD	(HL), E	; 而左边填入零，C 是被移动的字节，
0D76:	59	LD	E, C	; E 是前一个被移出的字节(开始时为零)。
0D77:	2B	DEC	HL	; 地址减 1。
0D78:	15	DEC	D	; 计数减 1。
0D79:	20F9	JR	NZ, 0D74H	; 还没有移完 7 个字节,便循环。
0D7B:	18EE	JR	0D6BH	; 循环,直到移位计数 <8。
0D7D:	C609	ADD	A, 09H	; 上述分解程序的继续,将双精度值以位右移进行调整。
0D7F:	57	LD	D, A	; D=右移的位数。
0D80:	AF	XOR	A	; A 清零。
0D81:	E1	POP	HL	; HL=MSB 的地址。
0D82:	15	DEC	D	; 计数位移的个数。
0D83:	C8	RET	Z	; 若已位移完了,则从分解程序返回。
0D84:	E5	PUSH	HL	; 暂存 MSB 的地址。
0D85:	1E08	LD	E, 08H	; 要移动的字节数。
0D87:	7E	LD	A, (HL)	; 取一个字节,把它右移,位 0 进入 C 标志,
0D88:	1F	RRA		; 然后变成下一个字节的位 7。
0D89:	77	LD	(HL), A	; 放回移位后的字节,
0D8A:	2B	DEC	HL	; 指向下一个地址。
0D8B:	1D	DEC	E	; 位移所有的字节。
0D8C:	20F9	JR	NZ, 0D87H	; 若没有移完,则继续循环。
0D8E:	18F0	JR	0D80H	; 移完了,测试是否按规定的数值移位了。
0D90:	212341	LD	HL, 4123H	; 指数的地址。★★★ 把 HL 指针指的双精度数右移一位 ★★★★★★★★★★★★★★★★★★★★★★
0D93:	1601	LD	D, 01H	; 右移的位数。
0D95:	18ED	JR	0D84H	; 转移到位移程序。在 0D83H 返回到调用它的程序。

地址	目的码	源程序		注　释
0D97:	0E08	LD	C, 08H	; 左移的字节数。★★★ 把 HL 指针指的双精度数左移一位 ★★★★★★★★★★★★★★★★★★★★★★
0D99:	7E	LD	A, (HL)	; 取 LSB。
0D9A:	17	RLA		; 左移一位，位 7 进入 C 标志，然后 C 标志进入下一个字节的位 0。
0D9B:	77	LD	(HL), A	; 把移位后的值放回原处。
0D9C:	23	INC	HL	; 指向 LSB+1。
0D9D:	0D	DEC	C	; 计数移位了一个字节。
0D9E:	20F9	JR	NZ, 0D99H	; 若 8 个字节还没有移完，则继续循环。
0DA0:	C9	RET		; 否则便返回。
0DA1:	CD5509	CALL	0955H	; 测试当前值的指数。★★★ 用重复加实现双精度相乘 ★★★★★★★★★★★★★★★★★★★★★★★★
0DA4:	C8	RET	Z	; 若该值为零，便返回。
0DA5:	CD0A09	CALL	090AH	; 调整指数，新的指数放入 4124H 中。
0DA8:	CD390E	CALL	0E39H	; 把当前值移到 414AH—4150H（暂存单元），而把原单元清零。
0DAB:	71	LD	(HL), C	; 411CH 清零。
0DAC:	13	INC	DE	; DE=414AH= 移入暂存单元中的单精度数值的首地址。
0DAD:	0607	LD	B, 07H	; B=要加的字节计数。
0DAF:	1A	LD	A, (DE)	; 取一个字节，从 LSB 开始取起。
0DB0:	13	INC	DE	; 指向下一个字节。
0DB1:	B7	OR	A	; 测试当前取入的字节是否为零。
0DB2:	D5	PUSH	DE	; 把当前字节的地址暂存起来。
0DB3:	2817	JR	Z, 0DCCH	; 若该字节为零，则把整个数值右移一个字节。
0DB5:	0E08	LD	C, 08H	; 要右移一个字节的次数。
0DB7:	C5	PUSH	BC	; 把处理过字节的计数暂存起来，初始时 B=7, C=8。
0DB8:	1F	RRA		; 右移 LSB，然后测试当前的位 0 是否为 1。
0DB9:	47	LD	B, A	; 若为 1，则把两个已分解的单精度数相加。
0DBA:	DC330D	CALL	C, 0D33H	; 把 WRA1 与 WRA2 中的数相加，结果留在 WRA1 中。
0DBD:	CD900D	CALL	0D90H	; 把和右移一位。
0DC0:	78	LD	A, B	; 取回右移后的 LSB，于是可测试其它几位。
0DC1:	C1	POP	BC	; 取回要测试的位数。
0DC2:	0D	DEC	C	; 计测试了一位。
0DC3:	20F2	JR	NZ, 0DB7H	; 如果对当前字节中各位还没有都进行测试，则继续循环。
0DC5:	D1	POP	DE	; 取来下一个要测试字节的地址。
0DC6:	05	DEC	B	; 是否所有的字节都已向右移调整了?
0DC7:	20E6	JR	NZ, 0DAFH	; 否，继续循环。
0DC9:	C3D80C	JP	0CD8H	; 是，转去将结果归一化，并返回到调用它的程序。
0DCC:	212341	LD	HL, 4123H	; HL=WRA1 的地址，A=0。

地址	目的码	源程序		注释
0DCF:	CD700D	CALL	0D70H	；把 WRA1 右移一个字节。
0DD2:	18F1	JR	0DC5H	；继续位移和相加的循环。
0DD4:	00	NOP		；双精度数 10 ★★★★★★★★★★★★★★★★★★
0DD5:	00	NOP		；
0DD6:	00	NOP		；
0DD7:	00	NOP		；
0DD8:	00	NOP		；
0DD9:	00	NOP		；
0DDA:	2084	JR	NZ，0D60H	；
0DDC:	11D40D	LD	DE，0DD4H	；取双精度数 10 的地址。
0DDF:	212741	LD	HL，4127H	；目的地址（WRA2 的）。
0DE2:	CDD309	CALL	09D3H	；把双精度数 10 移入 WRA2。
0DE5:	3A2E41	LD	A，(412EH)	；取除数的指数。★★★双精度除：WRA1 被 WRA2 除，用相减和移位的方法 ★★★★★★★★★★★★★★★★★★★★★★
0DE8:	B7	OR	A	；测试指数是否为零。
0DE9:	CA9A19	JP	Z，199AH	；若是零，则为 /0（被零除）的错误。
0DEC:	CD0709	CALL	0907H	；计算新的指数，置 WRA1 为负数。
0DEF:	34	INC	(HL)	；把 WRA1 的指数恢复为原数值。
0DF0:	34	INC	(HL)	；
0DF1:	CD390E	CALL	0E39H	；把 WRA1 的值移到 414AH—4150H 中(被除数)。
0DF4:	215141	LD	HL，4151H	；HL＝被除数(移动后的 WRA1）的指数地址。
0DF7:	71	LD	(HL)，C	；把指数清零。
0DF8:	41	LD	B，C	；把 B 寄存器清零。
0DF9:	114A41	LD	DE，414AH	；被除数(移动后的 WRA1）LSB 的地址。
0DFC:	212741	LD	HL，4127H	；除数（WRA2）LSB 的地址。
0DFF:	CD4B0D	CALL	0D4BH	；从被除数中减去除数。
0E02:	1A	LD	A，(DE)	；差值移入 414AH—4151H 中。
0E03:	99	SBC	A，C	；若 WRA2 的值大于 414A—4151H 中的值，
0E04:	3F	CCF		；414AH—4151H 数值的 MSB 减 1。
0E05:	380B	JR	C，0E12H	；若除数大于被除数，转移。否则，把差值加到移动后的当前值上。
0E07:	114A41	LD	DE，414AH	；DE＝移动后 WRA1 数值的 LSB 地址(被除数)。
0E0A:	212741	LD	HL，4127H	；HL＝WRA2 的 LSB 地址(除数)。
0E0D:	CD390D	CALL	0D39H	；把它们相加，并放入 414AH—4151H 中。
0E10:	AF	XOR	A	；清除所有的状态标志，以便不返回。
0E11:	DA1204	JP	C，0412H	；0E12：LD (DE)，A 存储新的指数(被除数)。
0E14:	3A2341	LD	A，(4123H)	；0E13：INC B 表示进行一次减法。
0E17:	3C	INC	A	；然后取被除数的 MSB。
0E18:	3D	DEC	A	；
0E19:	1F	RRA		；C 标志进入符号位。

地址	目的码	源程序		注释
0E1A:	FA110D	JP	M, 0D11H	; 已经完成,转移去把结果归一化。
0E1D:	17	RLA		; 恢复 C 标志。
0E1E:	211D41	LD	HL, 411DH	; HL=原被除数的 LSB 地址。
0E21:	0E07	LD	C, 07H	; 要移动的字节个数。
0E23:	CD990D	CALL	0D99H	; 把整个被除数左移一位。
0E26:	214A41	LD	HL, 414AH	; HL=移动后除数的地址。
0E29:	CD970D	CALL	0D97H	; 把移动后的除数左移一位,于是除数和被除数两者一致了。
0E2C:	78	LD	A, B	; 取减法计数,
0E2D:	B7	OR	A	; 置状态标志,
0E2E:	20C9	JR	NZ, 0DF9H	; 若除数<被除数,便转移。
0E30:	212441	LD	HL, 4124H	; 否则除数>被除数,把除数除以 2。
0E33:	35	DEC	(HL)	; 把指数减 1。
0E34:	20C3	JR	NZ, 0DF9H	; 然后重复减,如果除数趋于零,
0E36:	C3B207	JP	07B2H	; 将有 OV 错误。
0E39:	79	LD	A, C	; 取回 WRA2 值的 MSB。在此我们需要 C 寄存器。
0E3A:	322D41	LD	(412DH), A	; 放入 WRA2 的 MSB 中。
0E3D:	2B	DEC	HL	; HL=当前数值的 MSB 地址。
0E3E:	115041	LD	DE, 4150H	; DE=当前单精度数值的暂存区地址。
0E41:	010007	LD	BC, 0700H	; B = 要移动的字节数, C = 要传送给当前数值的值(00H)。
0E44:	7E	LD	A, (HL)	; 从当前值中取一个字节。
0E45:	12	LD	(DE), A	; 把它移入 414AH—4150H 中。
0E46:	71	LD	(HL), C	; 而把当前的字节清零。
0E47:	1B	DEC	DE	; 两个地址都减 1, 从 MSB 开始移起,
0E48:	2B	DEC	HL	; 一直移到 LSB。
0E49:	05	DEC	B	; 是否已移了 7 个字节?
0E4A:	20F8	JR	NZ, 0E44H	; 否,继续循环。
0E4C:	C9	RET		; 是,返回到调用它的程序。
0E4D:	CDFC09	CALL	09FCH	; 把当前数值移到暂存单元中。 ★★★★★★★★★★

说明 本子程序把一个当前的双精度数值乘以 10 的程序*, 用这个数加其本身的方法来完成。先是把当前数值移到暂存单元中,再调用双精度求和的程序,把当前值和暂存单元中的数相加。

0E50:	EB	EX	DE, HL	; HL=当前数值的最后一个地址
0E51:	2B	DEC	HL	; 指向指数的地址。
0E52:	7E	LD	A, (HL)	; 取指数。
0E53:	B7	OR	A	; 测试是否为零。
0E54:	C8	RET	Z	; 若数值是零,或者不是一个浮点数,则返回。
0E55:	C602	ADD	A, 02H	; 为以下的加法调整指数。

* 原文中为乘以 2, 而从 0E4DH—0E62H 中可看到是乘以 10, 原文有误。——译者

地址	目的码	源程序		注　释
0E57:	DAB207	JP	C, 07B2H	; 若指数溢出,则产生错误。
0E5A:	77	LD	(HL), A	; 把调整后的指数存储起来。
0E5B:	E5	PUSH	HL	; 并把该存储的指数的地址暂存起来。
0E5C:	CD770C	CALL	0C77H	; 把当前数值和暂存单元中的双精度数相加,和留在当前值工作区中。
0E5F:	E1	POP	HL	; 取回指数的地址。
0E60:	34	INC	(HL)	; 调整指数。
0E61:	C0	RET	NZ	; 若没有产生溢出,便返回。
0E62:	C3B207	JP	07B2H	; 若指数为零,则产生 OV 错误。
0E65:	CD7807	CALL	0778H	; 把单精度数值的指数清零。 ★★★ ASCII 码变成二进制数的子程序 ★★★★★★★★★★★★★★★★★★★
0E68:	CDEC0A	CALL	0AECH	; 标志为双精度数。
0E6B:	F6AF	OR	AFH	; 0E6C: XOR A
0E6D:	EB	EX	DE, HL	; 暂存 HL (当前输入符号的地址)。
0E6E:	01FF00	LD	BC, 00FFH	; 预置 HL=00, B=0, C=−0。
0E71:	60	LD	H, B	; 把 H 和 L 清零。
0E72:	68	LD	L, B	;
0E73:	CC9A0A	CALL	Z, 0A9AH	; 标志为整数,累加器清零。
0E76:	EB	EX	DE, HL	; 把当前输入符号的地址取回到 HL 中, DE=00。
0E77:	7E	LD	A, (HL)	; 取数字的第一个字符。
0E78:	FE2D	CP	2DH	; 测试是否为负号。
0E7A:	F5	PUSH	AF	; 把 MSB 作为符号暂存起来。
0E7B:	CA830E	JP	Z, 0E83H	; 若是负号,则转移去取下一个字符。
0E7E:	FE2B	CP	2BH	; 测试是否为'+',
0E80:	2801	JR	Z, 0E83H	; 若是正号,则转移去取下一个字符。
0E82:	2B	DEC	HL	; 地址减 1, 以抵消 RST 10 中的加 1。
0E83:	D7	RST	10H	; 检测下一个字符。
0E84:	DA290F	JP	C, 0F29H	; 若字符是一个数目字,转移。
0E87:	FE2E	CP	2EH	; 测试是否为小数点。
0E89:	CAE40E	JP	Z, 0EE4H	; 若是小数点,则转移。
0E8C:	FE45	CP	45H	; 测试是否为 E
0E8E:	2814	JR	Z, 0EA4H	; 若是 E, 则为指数型单精度数,转移。
0E90:	FE25	CP	25H	; 测试是否为%,
0E92:	CAEE0E	JP	Z, 0EEEH	, 若是 %, 则是整数,转移。
0E95:	FE23	CP	23H	; 测试是否为 #,
0E97:	CAF50E	JP	Z, 0EF5H	; 若是#,是双精度数,转移。
0E9A:	FE21	CP	21H	; 测试是否为!,
0E9C:	CAF60E	JP	Z, 0EF6H	; 若是!,是单精度数,转移。
0E9F:	FE44	CP	44H	; 测试是否为 D,
0EA1:	2024	JR	NZ, 0EC7H	; 若不是 D, 则转移,否则是指数型双精度数。
0EA3:	B7	OR	A	; 若是 D, 则 A 寄存器是非零。对于 E, 标志状态=0。

地址	目的码	源程序		注　释
				E 或 D 的处理。
0EA4:	CDFB0E	CALL	0EFBH	; 把数字转变成单精度或双精度数。
0EA7:	E5	PUSH	HL	; 暂存 HL，于是它可用于存放将被推入堆栈的地址。
0EA8:	21BD0E	LD	HL，0EBDH	; 把 0EBDH 作返回地址，放入堆栈。
0EAB:	E3	EX	(SP)，HL	; 取回 HL＝下一个字符的地址，堆栈＝0EBDH。
0EAC:	D7	RST	10H	; 测试输入流中的下一个字符，寻找符号。
0EAD:	15	DEC	D	; 若以下任何一个测试是真的，则 D＝－1。
0EAE:	FECE	CP	CEH	; 控制转到 0EBDH，否则将一直执行到 0EBDH。
0EB0:	C8	RET	Z	; 若是－(取－)的代号，则取 D＝－1，返回。
0EB1:	FE2D	CP	2DH	; 若不是取负的代号，则测试 ASCII 的负号代码。
0EB3:	C8	RET	Z	; 若是'－'字符，则返回 (D＝－1)。
0EB4:	14	INC	D	; 若后边是'＋'符号，则 D＝0，若后边是'－'符号，则 D＝－1。
0EB5:	FECD	CP	CDH	; 测试是否加(＋)的代号。
0EB7:	C8	RET	Z	; 若是＋的代号，则 D＝0，返回。
0EB8:	FE2B	CP	2BH	; 不是＋代号，则测试 ASCII 码的正号。
0EBA:	C8	RET	Z	; 若是正号字符 (D＝0)，则返回。
0EBB:	2B	DEC	HL	; 把地址减 1，使输入指针指向 E 或 D。
0EBC:	F1	POP	AF	; 从堆栈中消除 0EBDH 的返回地址。
0EBD:	D7	RST	10H	; 测试输入流中的下一个字符。
0EBE:	DA940F	JP	C，0F94H	; 若下一个字符是数字，则转移。
0EC1:	14	INC	D	; 最后的指数值：当－号时，D＝0；当＋号时，D＝＋1。
0EC2:	2003	JR	NZ，0EC7H	; 若指数为正，则转移。
0EC4:	AF	XOR	A	; 清除 A 寄存器。
0EC5:	93	SUB	A，E	; A＝0 － 指数值。
0EC6:	5F	LD	E，A	; E＝指数。
0EC7:	E5	PUSH	HL	; 把代码字符串的当前指针地址暂存起来。
0EC8:	7B	LD	A，E	; E＝指数。
0EC9:	90	SUB	A，B	; B＝除小数点以外的数字个数，A＝乘或除的次数。
0ECA:	F40A0F	CALL	P，0F0AH	; 把数值乘以 10。
0ECD:	FC180F	CALL	M，0F18H	; 对于在 0F68H—0F6FH 中每次乘和加，都要在 0F18H 处将数值除以 10，并且对 A 寄存器自动增 1。
0ED0:	20F8	JR	NZ，0ECAH	; 按 A 寄存器中的数目对数值进行循环换标。
0ED2:	E1	POP	HL	; 取回下一个字符的地址。
0ED3:	F1	POP	AF	; 取回可能的符号。
0ED4:	E5	PUSH	HL	; 把下一个字符的地址再暂存起来。
0ED5:	CC7B09	CALL	Z，097BH	; 在数值之前放了一个负号。
0ED8:	E1	POP	HL	; 取回代码字符串的地址。
0ED9:	E7	RST	20H	; 确定数据转换的类型。
0EDA:	E8	RET	PE	; 若不是单精度，则返回。
0EDB:	E5	PUSH	HL	; 暂存代码字符串的地址。

地址	目的码	源程序		注释
0EDC:	219008	LD	HL, 0890H	; 取返回地址。
0EDF:	E5	PUSH	HL	; 放入堆栈。
0EE0:	CDA30A	CALL	0AA3H	; 判断数值不是 -2^{16}，如果是 -2^{16}，则置类型为整数，数值为 8000H。
0EE3:	C9	RET		; 返回到 0890H。
0EE4:	E7	RST	20H	; 确定数据类型。★★★★★★★★★★★★★★★★
0EE5:	0C	INC	C	; C=0，
0EE6:	20DF	JR	NZ, 0EC7H	; 若整数后有小数点，或小数点是第一个字符，则转移。
0EE8:	DCFB0E	CALL	C, 0EFBH	; 若不是双精度，则转换成单精度。
0EEB:	C3830E	JP	0E83H	; 转移，去取下一个数字。
0EEE:	E7	RST	20H	; 确定数据类型。★★★发现%，把数值作最后处理并返回 ★★★★★★★★★★★★★★★★★★★★★★★★
0EEF:	F29719	JP	P, 1997H	; 若 P 置位(即不是整数)，则 SN 错误。
0EF2:	23	INC	HL	; 指向代码字符串的下一个字符。
0EF3:	18D2	JR	0EC7H	; 转移去作数据的最后处理和返回。
0EF5:	B7	OR	A	; 使 A 为非零值。★★★发现＃或！★★★★★★★★
0EF6:	CDFB0E	CALL	0EFBH	; 把数值转换成单精度或双精度。
0EF9:	18F7	JR	0EF2H	; 返回到调用它的程序。
0EFB:	E5	PUSH	HL	; 把输入字符串中的当前位置暂存起来。★★★★★★
0EFC:	D5	PUSH	DE	: 把字符串数值的整数部分暂存起来。
0EFD:	C5	PUSH	BC	; B=00, C=00。
0EFE:	F5	PUSH	AF	; 保存说明数据类型的标志，A=长度。
0EFF:	CCB10A	CALL	Z, 0AB1H	; 把当前数值转换成单精度。
0F02:	F1	POP	AF	; 取回标志。
0F03:	C4DB0A	CALL	NZ, 0ADBH	; 把当前数值转换成双精度。
0F06:	C1	POP	BC	; 取回 BC=0000H。
0F07:	D1	POP	DE	; 取回数值的整数部分。
0F08:	E1	POP	HL	; 取回输入字符串的当前位置。
0F09:	C9	RET		; 返回。
0F0A:	C8	RET	Z	; 把单(双)精度数乘 10，若是整数，则返回。★★★★★
0F0B:	F5	PUSH	AF	; 保存调用它程序的 AF 状态。
0F0C:	E7	RST	20H	; 确定数据的类型。
0F0D:	F5	PUSH	AF	; 暂存数据的类型。
0F0E:	E43E09	CALL	PO, 093EH	; 单精度:把当前数值乘以 10。
0F11:	F1	POP	AF	; 取回数据类型。
0F12:	EC4D0E	CALL	PE, 0E4DH	; 双精度:把当前数值乘以 10。
0F15:	F1	POP	AF	; 恢复调用它的程序的 AF。
0F16:	3D	DEC	A	; 把要乘的次数计数减 1。
0F17:	C9	RET		; 返回到调用它的程序。
0F18:	D5	PUSH	DE	; 把当前的单精度或双精度数除以 10。★★★★★★★
0F19:	E5	PUSH	HL	; 暂存调用它的程序的寄存器。

地址	目的码	源程序		注　释
0F1A:	F5	PUSH	AF	; 有 DE, AF 和 HL。
0F1B:	E7	RST	20H	; 确定数据的类型。
0F1C:	F5	PUSH	AF	; A 表示类型。
0F1D:	E49708	CALL	PO, 0897H	; 把当前数值除以 10。
0F20:	F1	POP	AF	; 取回数据的类型,这样可以转移到其它子程序去。
0F21:	ECDC0D	CALL	PE, 0DDCH	; 双精度:当前数值除以 10。
0F24:	F1	POP	AF	; 恢复用户的寄存器,
0F25:	E1	POP	HL	; 有 AF, HL
0F26:	D1	POP	DE	; 和 DE。
0F27:	3C	INC	A	; 然后把要除的次数的计数加 1。
0F28:	C9	RET		; 返回到调用它的程序。
0F29:	D5	PUSH	DE	; DE=0000H ★★★★★★★★★★★★★★★★★★★★
0F2A:	78	LD	A, B	; B=00H
0F2B:	89	ADC	A, C	; 当进入时, C 标志已设置,对单精度 C=00H, 对整数 C=FFH。
0F2C:	47	LD	B, A	; 对整数转换 B=0。对小数点后的单精度转换 B=数字的计数。
0F2D:	C5	PUSH	BC	; 把 C=0, B=计数暂存起来。
0F2E:	E5	PUSH	HL	; 暂存输入字符串的指针。
0F2F:	7E	LD	A, (HL)	; 取来一个字符。
0F30:	D630	SUB	30H	; 使 A=0—9 之间的数。
0F32:	F5	PUSH	AF	; 保存当前数字的二进制值。
0F33:	E7	RST	20H	; 确定我们正在转换成的数据类型。
0F34:	F25D0F	JP	P, 0F5DH	; 若不是整数,则转移。A=当前数字的值。
0F37:	2A2141	LD	HL, (4121H)	; 由 ASCII 码转换成的整数。
0F3A:	11CD0C	LD	DE, 0CCDH	; DE=3277
0F3D:	DF	RST	18H	; 把当前值与 3277 比较。
0F3E:	3019	JR	NC, 0F59H	; 若数值 >3277, 则转移。
0F40:	54	LD	D, H	; DE=当前值。
0F41:	5D	LD	E, L	; 乘 10:
0F42:	29	ADD	HL, HL	; 乘 2
0F43:	29	ADD	HL, HL	; 乘 4
0F44:	19	ADD	HL, DE	; 乘 5
0F45:	29	ADD	HL, HL	; 乘 10
0F46:	F1	POP	AF	; 取来当前数字的二进制值。
0F47:	4F	LD	C, A	; 放入 C 中。
0F48:	09	ADD	HL, BC	; 当前数字的二进制值与合并的值相加。
0F49:	7C	LD	A, H	; 测试目前已合并成的数值的符号。
0F4A:	B7	OR	A	; 置标志。
0F4B:	FA570F	JP	M, 0F57H	; 若数值超过 2^{15}, 则转移。
0F4E:	222141	LD	(4121H), HL	; 存储二进制数值。

地址	目的码	源程序		注　释
0F51:	E1	POP	HL	; 恢复 HL，BC 和 DE。
0F52:	C1	POP	BC	; B=小数点后数字的个数，C 在没有遇上小数点时为 FFH。
0F53:	D1	POP	DE	; 可能的符号状态。
0F54:	C3830E	JP	0E83H	; 取下一个数字。
0F57:	79	LD	A，C	; A=当前的数字。
0F58:	F5	PUSH	AF	; 把它暂存起来，于是我们可将它转换成单精度，然后在当前值转换成单精度后，两者相加。
0F59:	CDCC0A	CALL	0ACCH	; 把当前的数值变成单精度。
0F5C:	37	SCF		; 于是将分支转移到转换成双精度数的子程序。
0F5D:	3018	JR	NC，0F77H	; 若是双精度，则转移。
0F5F:	017494	LD	BC，9474H	; ASCII 码变成单精度，在 BC 和 DE 中放一个单精度数 16×10^6。
0F62:	110024	LD	DE，2400H	; 把16×10^6与(4121H—4124H)中的当前单精度值作比较。
0F65:	CD0C0A	CALL	0A0CH	;
0F68:	F2740F	JP	P，0F74H	; 若当前值大于 2^{16}，则转换为双精度。
0F6B:	CD3E09	CALL	093EH	; 把当前值乘以 10，我们将在以后除去增大的倍数。
0F6E:	F1	POP	AF	; A=当前的数字。
0F6F:	CD890F	CALL	0F89H	; 把当前数字变成单精度格式，然后加到工作区的数值上。
0F72:	18DD	JR	0F51H	; 取下一个数字，并计算在B寄存器中小数点后的位数。
0F74:	CDE30A	CALL	0AE3H	; 把双精度值的工作区 411DH=411FH 初始化，置双精度标志。
0F77:	CD4D0E	CALL	0E4DH	; 把当前单精度数值乘以 10。
0F7A:	CDFC09	CALL	09FCH	; 把（411DH—4124H）中的双精度值存入（4127H—412EH)中。
0F7D:	F1	POP	AF	; A=当前数字的二进制值。
0F7E:	CD6409	CALL	0964H	; 把它转换成单精度。
0F81:	CDE30A	CALL	0AE3H	; 把双精度值的工作区 411DH—411FH 初始化为零。
0F84:	CD770C	CALL	0C77H	; 把当前单精度数字加到当前精度数值上。
0F87:	18C8	JR	0F51H	; 去取下一个数字。
0F89:	CDA409	CALL	09A4H	; ★★★把 A 寄存器中的 8 位数变成单精度数，并加到 WRA1 的数值中去 ★★★★★★★★★★★★★★★★ 把（4121H—4124H）的当前值放入堆栈中去。
0F8C:	CD6409	CALL	0964H	; 把 A 寄存器的值变成单精度值。
0F8F:	C1	POP	BC	; 把堆栈中的当前单精度值放入 BC 和 DE 中。
0F90:	D1	POP	DE	; B=指数，C=MSB，D=NMSB，E=LSB。
0F91:	C31607	JP	0716H	; 把寄存器的值与 WRA1（4121H—4124H）中的当前值相加，并返回。
0F94:	7B	LD	A，E	; A=当前的指数。★★★★★★★★★★★★★★★★

说明　计算 E 寄存器中的指数值，使该值不超过 50。在处理 E 和 D 形式的指数时，将调用本子程序。

地址	目的码	源程序		注释
0F95:	FE0A	CP	0AH	; 与10比较。
0F97:	3009	JR	NC, 0FA2H	; 若≥10，使它为常数32H。
0F99:	07	RLCA		; 然后把当前数值乘以10。
0F9A:	07	RLCA		; ×4
0F9B:	83	ADD	A, E	; +1，变成乘5。
0F9C:	07	RLCA		; ×2，变成乘10。
0F9D:	86	ADD	A, (HL)	; 取当前数字（ASCII 码）
0F9E:	D630	SUB	30H	; 将它变成二进制数，加到当前数值中。
0FA0:	5F	LD	E, A	; 把当前数字放入 E 中。
0FA1:	FA1E32	JP	M, 321EH	; 0FA2: LD E,32H。
0FA4:	C3BD0E	JP	0EBDH	; 取输入字符串的下一个数字，返回到 0F94H。
0FA7:	E5	PUSH	HL	; 暂存代码字符串的地址。★★★★★★★★★★★★
0FA8:	212419	LD	HL, 1924H	; 取 IN 的信息地址。
0FAB:	CDA728	CALL	28A7H	; 输出信息。
0FAE:	E1	POP	HL	; 取回代码字符串地址。
0FAF:	CD9A0A	CALL	0A9AH	; 把 HL 中的数值作为当前值保存起来，置整数类型。★★★把 HL 中的数变成 ASCII 码，并显示在荧光屏上 ★★★★★★★★★★★★★★★★★★★★★★
0FB2:	AF	XOR	A	; 当转换时表明不能编辑。
0FB3:	CD3410	CALL	1034H	; 把显示缓冲区初始化。
0FB6:	B6	OR	(HL)	; 置非零状态，以便在 0EF7H 进行测试。
0FB7:	CDD90F	CALL	0FD9H	; 把当前值变成 ASCII 码。
0FBA:	C3A628	JP	28A6H	; 显示数值，并返回到调用它的程序。
0FBD:	AF	XOR	A	; 清除编辑标志。★★★ 把二进制数转换成 ASCII 子程序 ★★★★★★★★★★★★★★★★★★★★★★

说明 利用 A 寄存器中的编辑标志建立输出缓冲区。在入口时，B=在小数点前的'#'的个数，C=小数点之后的'#'个数。

0FBE:	CD3410	CALL	1034H	; 将输出缓冲区地址送 HL，编辑标志送入 40D8H。
0FC1:	E608	AND	08H	; 测试在输出中要求的符号。
0FC3:	2802	JR	Z, 0FC7H	; 若没有所要求的前导+号，则转移。
0FC5:	362B	LD	(HL), 2BH	; 正号。
0FC7:	EB	EX	DE, HL	; 把输出缓冲区的地址放入 DE 中。
0FC8:	CD9409	CALL	0994H	; 确定当前值的符号。
0FCB:	EB	EX	DE, HL	; 又把输出缓冲区的地址放回 HL 中。
0FCC:	F2D90F	JP	P, 0FD9H	; 若数值是正，则转移。
0FCF:	362D	LD	(HL), 2DH	; 把负号放入输出缓冲区中。
0FD1:	C5	PUSH	BC	; 把小数点前后的'#'号个数暂存起来。
0FD2:	E5	PUSH	HL	; 把显示缓冲区中的当前位置暂存起来。
0FD3:	CD7B09	CALL	097BH	; 把负数变成正数。
0FD6:	E1	POP	HL	; 取回显示缓冲区的地址。
0FD7:	C1	POP	BC	; 恢复计数。

地址	目的码	源程序		注　释
0FD8:	B4	OR	H	; 把正的 MSB 与 41H 合并在一起。
0FD9:	23	INC	HL	; HL＝4131H。
0FDA:	3630	LD	(HL), 30H	; ASCII 码的零放入显示缓冲区的下一个单元中。
0FDC:	3AD840	LD	A, (40D8H)	; A＝编辑标志。
0FDF:	57	LD	D, A	; 把编辑标志留在 D 中。
0FE0:	17	RLA		; 准备转移到测试位 2^{15} 的 (PRINT USING) 子程序。
0FE1:	3AAF40	LD	A, (40AFH)	; A＝当前变量的类型和长度的标志。
0FE4:	DA9A10	JP	Z, 109AH	; 若是由 PRINT USING 来调用本程序，则转移。
0FE7:	CA9210	JP	Z, 1092H	; 若编辑标志为零，则转去编辑。
0FEA:	FE04	CP	04H	; 测试数据类型。
0FEC:	D23D10	JP	NC, 103DH	; 若是单精度或双精度，则转移。
0FEF:	010000	LD	BC, 0000H	; BC＝没有逗点或没有小数点的标志。
0FF2:	CD2F13	CALL	132FH	; 把工作区中的整数变成 ASCII 码。
0FF5:	213041	LD	HL, 4130H	; ASCII 码缓冲区的首地址。
0FF8:	46	LD	B, (HL)	; B＝缓冲区中第一个 ASCII 码字符。
0FF9:	0E20	LD	C, 20H	; C＝空格的代码。
0FFB:	3AD840	LD	A, (40D8H)	; 取编辑参数字，看看是否需要测试，
0FFE:	5F	LD	E, A	; 及判别数是否超出范围。
0FFF:	E620	AND	20H	; 测试是否有所需的前导＊号。
1001:	2807	JR	Z, 100AH	; 不判断数是否超出范围。
1003:	78	LD	A, B	; 若在显示缓冲区第一个字符不等于空格，则数溢出。
1004:	B9	CP	C	; 把缓冲区的第一个字符与空格比较。
1005:	0E2A	LD	C, $2A_H$	; 若不相等，则以'＊'号 (C＝'＊') 代替缓冲区的第一个字符。
1007:	2001	JR	NZ, 100AH	; 数没有溢出。
1009:	41	LD	B, C	; 数溢出了。
100A:	71	LD	(HL), C	; 用'＊'号代替缓冲区的第一个字符。
100B:	D7	RST	10H	; 若不作范围检查，则无条件地以空格代替缓冲区中的第一个字符。
100C:	2814	JR	Z, 1022H	; 若是二进制的零(缓冲区的结束)，则转移。
100E:	FE45	CP	45H	; 测试是否为 E。
1010:	2810	JR	Z, 1022H	; 是，转移
1012:	FE44	CP	44H	; 测试是否为 D。
1014:	280C	JR	Z, 1022H	; 是，转移。
1016:	FE30	CP	30H	; 测试是否为 0。
1018:	28F0	JR	Z, 100AH	; 是 ASCII 的零，转移。
101A:	FE2C	CP	2CH	; 测试是否逗号。
101C:	28EC	JR	Z, 100AH	; 是逗号，转移。
101E:	FE2E	CP	2EH	; 测试是否为小数点。
1020:	2003	JR	NZ, 1025H	; 不是小数点，转移。
1022:	2B	DEC	HL	; 有一个小数点，或者缓冲区结束，或者有一个 D 或 E 时，

扫描输出缓冲区，寻找 E, D, 0, ·, , 和缓冲区的结束字符，将以空格代替零。

地址	目的码	源程序		注　释
1023:	3630	LD	(HL), 30H	; 把地址指针减1，指向上一个字节，用 ASCII 码的零代替该字符。
1025:	7B	LD	A, E	; A＝编辑标志。
1026:	E610	AND	10H	; 测试是否要插入前导 $ 号。
1028:	2803	JR	Z, 102DH	; 否，转移。
102A:	2B	DEC	HL	; 是，地址指针再减1。
102B:	3624	LD	(HL), 24H	; 插入一个$号。
102D:	7B	LD	A, E	; 再取回编辑标志。
102E:	E604	AND	04H	; 测试数值的符号是否放在数的后面。
1030:	C0	RET	NZ	; 否，返回到调用它的程序。
1031:	2B	DEC	HL	; 是，缓冲区指针减1。
1032:	70	LD	(HL), B	; 把符号存放进去。
1033:	C9	RET		; 返回到调用它的程序，
1034:	32D840	LD	(40D8H), A	; 存储编辑标志。★★★★★★★★★★★★★★★★
1037:	213041	LD	HL, 4130H	; HL＝显示输出缓冲区的首地址。
103A:	3620	LD	(HL), 20H	; 在显示缓冲区的第一个字符是空格。
103C:	C9	RET		; 返回到调用它的程序。
103D:	FE05	CP	05H	; ★★★把单(或双)精度转换成 ASCII 码，若是双精度，则 C 标志置位★★★★★★★★★★★★★★★★★
103F:	E5	PUSH	HL	; 把显示缓冲区的当前位置暂存起来。
1040:	DE00	SBC	A, 00H	; 若是单精度，则 A＝4，若是双精度，则 A＝8。
1042:	17	RLA		; 若是单精度，则 A＝8，若是双精度，则 A＝10H。
1043:	57	LD	D, A	; D 是调整后的类型标志。
1044:	14	INC	D	; D＝9 (单精度)，D＝11H (双精度)。
1045:	CD0112	CALL	1201H	; 把数值按比例换算，使 $99.999<x<999.99$。
1048:	010003	LD	BC, 0300H	; 在换算后，A＝双精度按比例增大(正数)
104B:	82	ADD	A, D	; 或按比例缩小(负数)的次数。
104C:	FA5710	JP	M, 1057H	; 若按比例缩小的次数大于9或 11H，则转移。
104F:	14	INC	D	; D＝0AH (单精度)或 12H (双精度)。
1050:	BA	CP	D	; 测试数值是否完全未经换算。
1051:	3004	JR	NC, 1057H	; 若已经按比例增大或缩小，则转移。
1053:	3C	INC	A	; A＝数值中数字的个数。
1054:	47	LD	B, A	; 把它存在 B 中。
1055:	3E02	LD	A, 02H	; 使指数为零，
1057:	D602	SUB	02H	; 计算指数值。
1059:	E1	POP	HL	; 取回显示缓冲区的地址。
105A:	F5	PUSH	AF	; 暂存指数。
105B:	CD9112	CALL	1291H	; 对逗号和小数点的初始化的程序。
105E:	3630	LD	(HL), 30H	; 把 ASCII 码的零放到显示缓冲区当前位置上。
1060:	CCC909	CALL	Z, 09C9H	; 若数已经换算完了，则 HL 加1。
1063:	CDA412	CALL	12A4H	; 把二进制数转成 ASCII 码，结果放在显示缓冲区中。

地址	目的码	源程序		注　释
1066:	2B	DEC	HL	; 指向缓冲区的上一个字符，使退到缓冲区中第一个非零值。
1067:	7E	LD	A, (HL)	; 取上一个字符，
1068:	FE30	CP	30H	; 与 ASCII 码的零比较，
106A:	28FA	JR	Z, 1066H	; 在发现非零字符之前，循环。
106C:	FE2E	CP	2EH	; 测试小数点。
106E:	C4C909	CALL	NZ, 09C9H	; 若不是小数点，则调用 09C9H（HL 加 1，指向小数点后的第一个字符）。
1071:	F1	POP	AF	; 取回指数。
1072:	281F	JR	Z, 1093H	; 若指数是零，则转移。
1074:	F5	PUSH	AF	; 把指数暂存起来。
1075:	E7	RST	20H	; 测试数据类型。
1076:	3E22	LD	A, 22H	; 这将根据数值是单精度或双精度，
1078:	8F	ADC	A, A	; 而变成 E 或 D 的形式。
1079:	77	LD	(HL), A	; 暂存指数的标志。
107A:	23	INC	HL	; 缓冲区中地址加 1，指向指数的第一个位置。
107B:	F1	POP	AF	; 再取回指数值。
107C:	362B	LD	(HL), 2BH	; 正(＋)的指数。
107E:	F28510	JP	P, 1085H	; 若指数为正，则转移。
1081:	362D	LD	(HL), 2DH	; 负(－)的指数。
1083:	2F	CPL		; 把负指数变成正指数。
1084:	3C	INC	A	;
1085:	062F	LD	B, 2FH	; B＝ASCII 值的 0,1,…,9 的起始值（指 0 前面的 '/' 的 ASCII 值）。
1087:	04	INC	B	; 开始用混合减法循环来除以 10：把 A 中的当前值变成一个 ASCII 码表示的数字。
1088:	D60A	SUB	0AH	; 减 10，
108A:	30FB	JR	NC, 1087H	; 直到余数 ＜10，B＝商。
108C:	C63A	ADD	A, 3AH	; 把余数变成一个 ASCII 码的数字。
108E:	23	INC	HL	; 显示缓冲区的地址加 1，指向下一个位置。
108F:	70	LD	(HL), B	; 指数的第一位数字。
1090:	23	INC	HL	; 缓冲区的地址加 1。
1091:	77	LD	(HL), A	; 指数的第二位数字。
1092:	23	INC	HL	; 缓冲区的地址加 1。
1093:	3600	LD	(HL), 00H	; 00 表示 ASCII 字符串的结束。
1095:	EB	EX	DE, HL	; DE＝显示缓冲区的结束地址。
1096:	213041	LD	HL, 4130H	; HL＝显示缓冲区的首地址。
1099:	C9	RET		; 返回到调用它的程序。
109A:	23	INC	HL	; 显示缓冲区的地址加 1。★★★ PRINT USING 的编辑操作 ★★★★★★★★★★★★★★☆★★★★★
109B:	C5	PUSH	BC	; B＝小数点前 '#' 个数，C＝小数点后 '#' 个数。

地址	目的码	源程序		注　释
109C:	FE04	CP	04H	; A=数据类型,测试整数或浮点数。
109E:	7A	LD	A, D	; A=编辑标志。
109F:	D20911	JP	NC, 1109H	; 若是单(或双)精度数,则转移。
10A2:	1F	RRA		; 置指数项的标志。
10A3:	DAA311	JP	C, 11A3H	; 若当前变量是字符串变量,则转移,否则必定是整数。
10A6:	010306	LD	BC, 0603H	; B=前导数目字的个数, C=逗号的计数。PRINT USING 的整数编辑。
10A9:	CD8912	CALL	1289H	; 测试逗号标志,若不是逗号, C 清零。
10AC:	D1	POP	DE	; D=小数点清'#'的个数。
10AD:	7A	LD	A, D	; 把计数放入 A 中。
10AE:	D605	SUB	05H	; 与5比较(整数所允许的最大数字个数)。
10B0:	F46912	CALL	P, 1269H	; 若大于5位数字,则把显示缓冲区前边充以零。
10B3:	CD2F13	CALL	132FH	; 把当前值转换成 ASCII 码。
10B6:	7B	LD	A, E	; 把小数点后'#'的个数放在 A 中。
10B7:	B7	OR	A	; 置状态标志。
10B8:	CC2F09	CALL	Z, 092FH	; 若后面没有'#'号, 把显示器的地址指针减 1。
10BB:	3D	DEC	A	; 测试计数是否为零。
10BC:	F46912	CALL	P, 1269H	; 否则,小数点后添加 (A) 个零。
10BF:	E5	PUSH	HL	; 暂存显示缓冲区地址。
10C0:	CDF50F	CALL	0FF5H	; 编辑 ASCII 缓冲区,并把数变换成这种格式。
10C3:	E1	POP	HL	; 取回显示缓冲区地址。
10C4:	2802	JR	Z, 10C8H	; 若符号位放在数字之后,转移。
10C6:	70	LD	(HL), B	; 否则在数值后存放一个空格。
10C7:	23	INC	HL	; 显示缓冲区地址加 1, 指向下一个地址。
10C8:	3600	LD	(HL), 00H	; 用一个零字节表示缓冲区结束。
10CA:	212F41	LD	HL, 412FH	; ASCII 码显示缓冲区首址减 1。
10CD:	23	INC	HL	; 显示缓冲区地址加 1, 指向下一个地址。

说明　在显示缓冲区中寻找字段的开始地址,然后返回到调用它的程序。若字段以'+'、'-'或'$'开始, 则在返回之前,先转移到 10DFH。搜索字段是从小数点的地址开始, 并逐步往后退回一个字段长度 (在 D 寄存器中)。

10CE:	3AF340	LD	A, (40F3H)	; A=显示缓冲区中小数点地址的 LSB。
10D1:	95	SUB	A, L	; 与现在的显示缓冲区 LSB 比较。
10D2:	92	SUB	A, D	; 再减去字段长度。
10D3:	C8	RET	Z	; 若找到了字段的开始地址,则返回。
10D4:	7E	LD	A, (HL)	; 不是字段的开始地址,则取字符,
10D5:	FE20	CP	20H	; 并测试是否为空格。
10D7:	28F4	JR	Z, 10CDH	; 测试到字段开始,或者发现'+','-'或'$',否则循环。
10D9:	FE2A	CP	2AH	; 测试是否为'*'。
10DB:	28F0	JR	Z, 10CDH	; 是,则不理睬空格和'*'。
10DD:	2B	DEC	HL	; 地址减 1, 指向上一个字符,再测试。
10DE:	E5	PUSH	HL	; 把缓冲区的地址暂存起来。

地址	目的码	源程序		注　释
10DF:	F5	PUSH	AF	; 把 A 中的字符暂存起来。
10E0:	01DF10	LD	BC, 10DFH	; 在'+'、'-'和'$'的情形中的返回地址。
10E3:	C5	PUSH	BC	; 把它放入堆栈。
10E4:	D7	RST	10H	; 再测试这个字符，
10E5:	FE2D	CP	2DH	; 与'-'作比较。
10E7:	C8	RET	Z	; 若是负号，则返回到 10DFH，
10E8:	FE2B	CP	2BH	; 不是，再测试是否'+'。
10EA:	C8	RET	Z	; 若是正号，则返回到 10DFH，
10EB:	FE24	CP	24H	; 不是'+'和'-'，是否'$'?
10ED:	C8	RET	Z	; 是'$'，返回到 10DFH。
10EE:	C1	POP	BC	; 清除返回地址 10DFH。
10EF:	FE30	CP	30H	; 测试是否 ASCII 码的零(前导 0)。
10F1:	200F	JR	NZ, 1102H	; 若不是前导零，则转移。
10F3:	23	INC	HL	; 指向下一个字符。
10F4:	D7	RST	10H	; 并测试它后边的一个字符。
10F5:	300B	JR	NC, 1102H	; 若不是数字，则转移。
10F7:	2B	DEC	HL	; 再把地址减 1，退回一个字符来测试。
10F8:	012B77	LD	BC, 772BH	; 10F9: DEC HL 再退一个字符。
10FB:	F1	POP	AF	; 10FA: LD (HL), A 把数字移动一个位置。
10FC:	28FB	JR	Z, 10F9H	; 在找到字段的结束之前，继续循环。
10FE:	C1	POP	BC	; 清除堆栈。
10FF:	C3CE10	JP	10CEH	; 重新扫描。
1102:	F1	POP	AF	; 恢复字段开始处的字符。
1103:	28FD	JR	Z, 1102H	; 在发现字段的首地址之前，继续循环。
1105:	E1	POP	HL	; 取回字段的首地址。
1106:	3625	LD	(HL), 25H	; 用'%'代替。
1108:	C9	RET		; 返回到调用程序。
1109:	E5	PUSH	HL	; 把当前显示缓冲区地址暂存起来。★★★ 浮点数编辑 ★★★★★★★★★★★★★★★★★★★★★★★
110A:	1F	RRA		; 为 PRINT USING，测试编辑标志的位 0。
110B:	DAAA11	JP	C, 11AAH	; 若浮点数以指数表示，则转移。
110E:	2814	JR	Z, 1124H	; 若是单精度数值，则转移。
1110:	118413	LD	DE, 1384H	; DE＝双精度数 1×10^{16} 的地址。
1113:	CD490A	CALL	0A49H	; 将数值与 1×10^{16} 比较。
1116:	1610	LD	D, 10H	; D＝双精度字段的数字个数。
1118:	FA3211	JP	M, 1132H	; 若数值小于 0×10^{16}，则转移。
111B:	E1	POP	HL	; 取回显示缓冲区的当前地址。
111C:	C1	POP	BC	; B＝小数点前'#'的个数，C＝小数点之后'#'的个数。
111D:	CDBD0F	CALL	0FBDH	; 重新进入编辑程序，使数值小于 1×10^{16}。
1120:	2B	DEC	HL	; 得到缓冲区的当前位置。
1121:	3625	LD	(HL), 25H	; 存储一个'%'(空格字段的开始)。

地址	目的码	源程序		注　释
1123:	C9	RET		; 返回到调用它的程序。
1124:	010EB6	LD	BC, B60EH	; 编辑单(双)精度数值小于 1×10^{16} ★★★★★★★★
1127:	11CA1B	LD	DE, 1BCAH	; BC 和 DE=1×10^{16}
112A:	CD0C0A	CALL	0A0CH	; 把要编辑的数值与 1×10^{16} 比较。
112D:	F21B11	JP	P, 111BH	; 若要编辑的数值大于 1×10^{16},转移。
1130:	1606	LD	D, 06H	; D=要打印的数字个数(字段大小)。
1132:	CD5509	CALL	0955H	; 测试数值的符号。
1135:	C40112	CALL	NZ, 1201H	; 把单精度值按比例换算，使它在 99,999 至 999,999 之间,返回时，A=换算次数(+表示增大,-表示缩小)。
1138:	E1	POP	HL	; HL=ASCII 码缓冲区的开始。
1139:	C1	POP	BC	; B=小数点前'#'的个数， C=小数点后'#'的个数。
113A:	FA5711	JP	M, 1157H	; 若数值按比例增大(乘以 10)，则转移。
113D:	C5	PUSH	BC	; 把小数点前后的'#'个数暂存起来。
113E:	5F	LD	E, A	; E=原数值除以 10 的次数。
113F:	78	LD	A, B	; B=用户指定小数点前'#'的个数。
1140:	92	SUB	A, D	; D=6。
1141:	93	SUB	A, E	; E=编辑数值除以 10 的次数。
1142:	F46912	CALL	P, 1269H	; 把前导的 ASCII 码的零放入显示缓冲区中。
1145:	CD7D12	CALL	127DH	; 计算小数点和逗号的计数。
1148:	CDA412	CALL	12A4H	; 把单精度的整数部分转换成 ASCII 码。
114B:	B3	OR	E	; 测试按比例换算数值的次数。
114C:	C47712	CALL	NZ, 1277H	; 对每次数值换算加后导零。
114F:	B3	OR	E	; 置状态标志。
1150:	C49112	CALL	NZ, 1291H	; 把小数点和逗号放入数值缓冲区。
1153:	D1	POP	DE	; 取回编辑计数。
1154:	C3B610	JP	10B6H	; 去把数值的小数部分转换成 ASCII 码。
1157:	5F	LD	E, A	; E=数值按比例增大(乘以 10)的次数。 ★★★★★★
1158:	79	LD	A, C	; C=小数点后要打印的数字个数。数值已按比例增大了。调整小数点后位数的比例。
1159:	B7	OR	A	; 测试计数。
115A:	C4160F	CALL	NZ, 0F16H	; 若不为零,则把小数点后'#'的个数减 1。
115D:	83	ADD	A, E	; A=(小数点后'#'个数-1)+(-数值按比例增大的次数)
115E:	FA6211	JP	M, 1162H	; 若数值需要按比例缩小,则转移。
1161:	AF	XOR	A	; 表示没有按比例缩小。
1162:	C5	PUSH	BC	; 把小数点前后的计数暂存。
1163:	F5	PUSH	AF	; 暂存换算计数。
1164:	FC180F	CALL	M, 0F18H	; 当前值被 10 除以 (A) 次。
1167:	FA6411	JP	M, 1164H	; 每除一次，A 加 1。
116A:	C1	POP	BC	; 取回原换算计数。
116B:	7B	LD	A, E	; A=数值乘以 10 的次数。

数值被按比例缩小,或者根本没有换算。调整小数点前位数的比例。

地址	目的码	源程序		注释
116C:	90	SUB	A, B	; 减换算计数。
116D:	C1	POP	BC	; 取回小数点前后的计数。
116E:	5F	LD	E, A	; 调整比例因子。
116F:	82	ADD	A, D	; 加字段的长度(置符号标志)。
1170:	78	LD	A, B	; A=小数点前'#'的个数。
1171:	FA7F11	JP	M, 117FH	; 若没有前导数字,则转移,
1174:	92	SUB	A, D	; 否则减去字段长度(单精度为 6, 双精度为 10H)。
1175:	93	SUB	A, E	; 再减去调整后的比例因子。
1176:	F46912	CALL	P, 1269H	; 添加后导零。
1179:	C5	PUSH	BC	; 把小数点前后'#'的计数暂存起来。
117A:	CD7D12	CALL	127DH	; 设置 B 和 C 为小数点和逗号的计数。
117D:	1811	JR	1190H	; 转移去编辑小数点前的数。
117F:	CD6912	CALL	1269H	; 在显示缓冲区中插入零。 ★★★★★★★★★★★★
1182:	79	LD	A, C	; 把逗号计数暂存起来, 否则它将在调用 1294H 时被清除。
1183:	CD9412	CALL	1294H	; 把小数点加到显示缓冲区中,给出 0。
1186:	4F	LD	C, A	; 取出逗号计数放入 C 寄存器中。
1187:	AF	XOR	A	; 把 A 寄存器清零。
1188:	92	SUB	A, D	; 求出要求的字段长度与换算的字段长度之差,
1189:	93	SUB	A, E	;
118A:	CD6912	CALL	1269H	; 然后把许多零加到显示缓冲区中。
118D:	C5	PUSH	BC	; 把小数点前后的'#'个数暂存起来。
118E:	47	LD	B, A	; 把 B 清零,
118F:	4F	LD	C, A	; 把 C 清零。
1190:	CDA412	CALL	12A4H	; 把单精度的整数部分转变成整数的 ASCII 码。
1193:	C1	POP	BC	; 取回计数。
1194:	B1	OR	C	; 对小数点后'#'的个数设置标志。
1195:	2003	JR	NZ, 119AH	; 若小数点后有数字,转移。
1197:	2AF340	LD	HL, (40F3H)	; 否则,取显示缓冲区中小数点的地址。
119A:	83	ADD	A, E	; 给出小数点前数字的个数。
119B:	3D	DEC	A	; 减 1,
119C:	F46912	CALL	P, 1269H	; 把许多零添加到显示缓冲区中。
119F:	50	LD	D, B	; 置 D=小数点前'#'的个数。
11A0:	C3BF10	JP	10BFH	; 转移去编辑 ASCII 数值。
11A3:	E5	PUSH	HL	; 暂存显示缓冲区的当前地址。★★★ PRINT USING 的指数格式。11A3H 是整数的入口点, 11AAH 是单(双)精度的入口点 ★★★★★★★★★★★★★★★★★★
11A4:	D5	PUSH	DE	; 暂存编辑标志。
11A5:	CDCC0A	CALL	0ACCH	; 把整数转变成单精度数。
11A8:	D1	POP	DE	; 取回编辑标志。
11A9:	AF	XOR	A	; 清除状态标志,作单精度处理。

地址	目的码	源程序		注释
11AA:	CAB011	JP	Z, 11B0H	; 单(双)精度入口点。若是单精度,则转移。
11AD:	1E10	LD	E, 10H	; E=双精度时要打印的数字个数。
11AF:	011E06	LD	BC, 061EH	; 11B0: LD E,6 E=单精度时要打印的数字个数。
11B2:	CD5509	CALL	0955H	; 测试当前数值的符号。
11B5:	37	SCF		; 置 C 标志,以便在第一次通过时,在 11F3H 强迫转移。
11B6:	C40112	CALL	NZ, 1201H	; 若当前值不为零,去换算该数值。
11B9:	E1	POP	HL	; 取回显示缓冲区的地址。
11BA:	C1	POP	BC	; 取回小数点前后'#'的个数。
11BB:	F5	PUSH	AF	; 暂存标志,以便在 11F3H 处测试。
11BC:	79	LD	A, C	; A=小数点后'#'的个数。
11BD:	B7	OR	A	; 置标志,以测试是否为零。
11BE:	F5	PUSH	AF	; 暂存小数点后数字个数。
11BF:	C4160F	CALL	NZ, 0F16H	; 若小数点后数字不为零,则把它减 1。
11C2:	80	ADD	A, B	; 把小数点前后数字计数合并在一起。
11C3:	4F	LD	C, A	; 把总数暂存起来。
11C4:	7A	LD	A, D	; 取编辑标志。
11C5:	E604	AND	04H	; 分离出在数值标志后面的符号。
11C7:	FE01	CP	01H	; 若数值后有符号,则 C 标志不置位。
11C9:	9F	SBC	A, A	; 若有符号,则 A=0, 否则 A=FFH。
11CA:	57	LD	D, A	; 暂存新的编辑标志。
11CB:	81	ADD	A, C	; 若有符号放在后面,调整显示的数字个数。
11CC:	4F	LD	C, A	; 暂存调整后的计数。
11CD:	93	SUB	A, E	; A=除以 10 的次数。
11CE:	F5	PUSH	AF	; 暂存除数计数。
11CF:	C5	PUSH	BC	; 暂存字符计数。
11D0:	FC180F	CALL	M, 0F18H	; 把数值除以 10, 共除 (A) 次(每除一次, A 加 1)。
11D3:	FAD011	JP	M, 11D0H	; 还能除的话,便再环。
11D6:	C1	POP	BC	; 取回'#'的计数。
11D7:	F1	POP	AF	; 取回除数计数。
11D8:	C5	PUSH	BC	; 再次把它们暂存。
11D9:	F5	PUSH	AF	;
11DA:	FADE11	JP	M, 11DEH	; 若后边有零,则转移。
11DD:	AF	XOR	A	; 清除 A 和标志。
11DE:	2F	CPL		; 使后导零的计数为正。
11DF:	3C	INC	A	; 再变成补码。
11E0:	80	ADD	A, B	; 加小数点之前字段的长度。
11E1:	3C	INC	A	; 再加 1,
11E2:	82	ADD	A, D	; 加字段的长度(单精度为 6, 双精度为 10H)。
11E3:	47	LD	B, A	; B=小数点前数字的个数。
11E4:	0E00	LD	C, 00H	; 表示没有逗号。
11E6:	CDA412	CALL	12A4H	; 把数值变成 ASCII 码。

地址	目的码	源程序		注释
11E9:	F1	POP	AF	; 取回原小数点前'#'的个数。
11EA:	F47112	CALL	P,1271H	; 添加后导零。
11ED:	C1	POP	BC	; 取回小数点前后数字个数的计数。
11EE:	F1	POP	AF	; 取回小数点前的数字个数。
11EF:	CC2F09	CALL	Z, 092FH	; 若在小数点之前没有数字，则显示缓冲区的地址减 1。
11F2:	F1	POP	AF	; 取第一次的标志，若 C 标志置位，则清除堆栈，
11F3:	3803	JR	C, 11F8H	; 加指数，并转入公共的编辑程序。
11F5:	83	ADD	A, E	; 否则，在正的时候，把缺项字段长度加 1。
11F6:	90	SUB	A, B	; 在负的时候加 1，然后减去字段中实际的字符数。
11F7:	92	SUB	A, D	; 得到指数的大小。
11F8:	C5	PUSH	BC	; 暂存 BC。
11F9:	CD7410	CALL	1074H	; 计算指数，并加到显示缓冲区中。
11FC:	EB	EX	DE, HL	; 恢复 HL。
11FD:	D1	POP	DE	; 清除堆栈。
11FE:	C3BF10	JP	10BFH	; 转移去编辑 ASCII 码的值。
1201:	D5	PUSH	DE	; 测试单(双)精度数值的大小。 ★★★★★★★★★★

说明 把数值换算的次数清除掉。将一个单精度或双精度数进行换算，使数值在 99999 到 999999 之间。在出口时，A>0 表示按比例缩小除以 10 的次数，A<0 表示按比例增大乘以 10 的次数。

1202:	AF	XOR	A	; 把 A 和标志清零。
1203:	F5	PUSH	AF	; 把零放入堆栈。
1204:	E7	RST	20H	; 测试数据类型。
1205:	E22212	JP	PO, 1222H	; 若是单精度，则转移。
1208:	3A2441	LD	A, (4124H)	; 若是双精度，把指数放入 A 中。
120B:	FE91	CP	91H	; 计算数据整数部分的位数。
120D:	D22212	JP	NC, 1222H	; 若双精度数整数部分的位数大于或等于 17，则转移。
1210:	116413	LD	DE, 1364H	; DE＝双精度数 10×10^9 的地址。
1213:	212741	LD	HL, 4127H	; 目的工作区 (WRA2) 的地址。
1216:	CDD309	CALL	09D3H	; 送入存储数值的单元 (WRA2)。
1219:	CDA10D	CALL	0DA1H	; 把当前数值 (WRA1 中)乘以 10×10^9。
121C:	F1	POP	AF	; A＝按比例增大双精度数而进行乘的次数计数。
121D:	D60A	SUB	0AH	; A＝计数 −10。
121F:	F5	PUSH	AF	; 为了测试，暂存起来。
1220:	18E6	JR	1208H	; 在整数部分不超过 2^{16} 时，循环。
1222:	CD4F12	CALL	124FH	; 把当前数值与 999999 比较，若数值小于 999999，则返回。 ★★★ 把单(双)精度数进行换算，使数值在 99999 至 999999 之间 ★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★★
1225:	E7	RST	20H	; 测试数据类型。
1226:	EA3412	JP	PE, 1234H	; 若不是单精度，则转移。
1229:	014391	LD	BC, 9143H	; BC 和 DE 为单精度的 99999。
122C:	11F94F	LD	DE, 4FF9H	;

地址	目的码	源程序		注释
122F:	CD0C0A	CALL	0A0CH	; 把当前数值与 99999 比较。
1232:	1806	JR	123AH	; 测试比较结果。
1234:	116C13	LD	DE,136CH	; DE 是单精度数，1.44×10^{17} 的地址。
1237:	CD490A	CALL	0A49H	; 把当前值与 1.44×10^{17} 比较。
123A:	F24C12	JP	P, 124CH	; 若数值大于 99999，则转移（整数大于 5 位，双精度小于 17 位）。
123D:	F1	POP	AF	; A＝换算计数。
123E:	CD0B0F	CALL	0F0BH	; 当前数值乘以 10。
1241:	F5	PUSH	AF	; A＝负的乘 10 的次数。
1242:	18E1	JR	1225H	; 在数值落入 99999 至 999999 之间前，循环。
1244:	F1	POP	AF	; A＝换算计数，
1245:	CD180F	CALL	0F18H	; 若数值 ＞999999，则除以 10。
1248:	F5	PUSH	AF	; 暂存除的次数。
1249:	CD4F12	CALL	124FH	; 在数值小于 999999 之前，循环。
124C:	F1	POP	AF	; A＝＋(除的次数)或 A＝－(乘的次数)。
124D:	D1	POP	DE	; 恢复调用它的程序的 DE。
124E:	C9	RET		; 返回到调用它的程序。
124F:	E7	RST	20H	; 测试数据类型。 ★★★★★★★★★★★★★★★★
1250:	EA5E12	JP	PE, 125EH	; 若是双精度数，则转移。
1253:	017494	LD	BC, 9474H	; BC 和 DE＝999999。
1256:	11F823	LD	DE, 23F8H	;
1259:	CD0C0A	CALL	0A0CH	; 把当前数值与 999999 比较。
125C:	1806	JR	1264H	; 测试比较结果。
125E:	117413	LD	DE, 1374H	; DE＝1×10^{16} 数值的地址。 ★★★★★★★★★★
1261:	CD490A	CALL	0A49H	; 与当前数值进行比较。
1264:	E1	POP	HL	; 取消返回地址，使能转移到 1244H。
1265:	F24412	JP	P, 1244H	; 若当前数值的整数部分大于 6 位数，则转移。
1268:	E9	JP	(HL)	; 否则返回到调用它的程序。
1269:	B7	OR	A	; 测试零标志。 ★★★把 A 作的数作计数，把（A）个 ASCII 码的零放入以 HL 为指针的显示缓冲区中★★
126A:	C8	RET	Z	; 若 A 为零，则返回。
126B:	3D	DEC	A	; 把一个 ASCII 码的零放入了显示缓冲区，A 减 1。
126C:	3630	LD	(HL), 30H	; 存放一个 ASCII 码的零。
126E:	23	INC	HL	; 存放地址加 1。
126F:	18F9	JR	126AH	; 若 A 寄存器不为零，则循环，直到传送（A）个 ASCII 码的零。
1271:	2004	JR	NZ, 1277H	; 若后导零还没有加完，则转移去继续加后导零。
1273:	C8	RET	Z	; 若后导零已加完，则返回到调用它的程序。
1274:	CD9112	CALL	1291H	; 把小数点和逗号放入数值缓冲区中。
1277:	3630	LD	(HL), 30H	; 把一个后导的 ASCII 码的零放入显示缓冲区。
1279:	23	INC	HL	; 缓冲区地址加 1。

地址	目的码	源程序		注释
127A:	3D	DEC	A	; 计数减 1，表示已加了一个后导零。
127B:	18F6	JR	1273H	; 测试是否完成。
127D:	7B	LD	A, E	; A=数值换算中乘或除的次数。

说明 计算在小数点之前数字个数以及要加多少个逗号。在入口时，D=字段的长度（6 位或 16 位），E=乘以或除以 10 的次数。在返回时，B=小数点前数字个数，C=在小数点之前这部分数据要加逗号的个数。

127E:	82	ADD	A, D	; D=要显示的数据位数。
127F:	3C	INC	A	; A 加 1，给出了小数点之前的位数。
1280:	47	LD	B, A	; B=小数点前的位数。
1281:	3C	INC	A	; 小数点之前的位数加 1。
1282:	D603	SUB	03H	; 以 3 个为一组(以便加入逗号)。
1284:	30FC	JR	NC, 1282H	; 在 A=−1, −2 或 −3 之前，循环。
1286:	C605	ADD	A, 05H	; A 加 5，得到正的余数，给出为 4, 3, 2。
1288:	4F	LD	C, A	; C=逗号计数。
1289:	3AD840	LD	A, (40D8H)	; A=编辑标志，测试加逗号的标志。
128C:	E640	AND	40H	; 从编辑标志中取出加逗号的标志位。
128E:	C0	RET	NZ	; 若要加逗号，则 C=逗号计数，返回，
128F:	4F	LD	C, A	; 否则使逗号计数为零。
1290:	C9	RET		; 返回到调用它的程序。
1291:	05	DEC	B	; 计数一个前导数字。★★★计算小数点之前的数 ★★
1292:	2008	JR	NZ, 129CH	; 若还没有把所有的前导数存入显示缓冲区，则转移。
1294:	362E	LD	(HL), 2EH	; 前导数已存完，存入一个小数点。
1296:	22F340	LD	(40F3H), HL	; 把小数点的地址放入存储器中。
1299:	23	INC	HL	; 把地址加 1，指向数值小数部分的第一个字符位置。
129A:	48	LD	C, B	; 把 C 和 B 置为零，禁止再存入小数点和逗号。
129B:	C9	RET		; 返回到调用它的程序。
129C:	0D	DEC	C	; 已存入一个字符，计数减 1。
129D:	C0	RET	NZ	; 若以 3 个字符为一组还没有存完，则返回。
129E:	362C	LD	(HL), 2CH	; 每 3 位数字加一个逗号。
12A0:	23	INC	HL	; 指向缓冲区的下一个地址。
12A1:	0E03	LD	C, 03H	; 重新设置逗号计数。
12A3:	C9	RET		; 返回到调用它的程序。
12A4:	D5	PUSH	DE	; 暂存编辑标志。 ★★★★★★★★★★★★★★★★
12A5:	E7	RST	20H	; 测试数据类型。
12A6:	E2EA12	JP	PO, 12EAH	; 若单精度数便转移。只把双精度数的整数部分变成 ASCII 码值。
12A9:	C5	PUSH	BC	; 暂存小数点前位数计数和逗号计数。
12AA:	E5	PUSH	HL	; 暂存缓冲区地址。
12AB:	CDFC09	CALL	09FCH	; 把 WRA1 中的数据移到 WRA2 中。
12AE:	217C13	LD	HL, 137CH	; HL=双精度数 0.5 的地址。

地址	目的码	源程序		注　释
12B1:	CDF709	CALL	09F7H	; 把它移入 WRA1 中。
12B4:	CD770C	CALL	0C77H	; 把 0.5 加到 WRA2 中的数值,结果放在 WRA1 中。
12B7:	AF	XOR	A	; 清除状态标志。
12B8:	CD7B0B	CALL	0B7BH	; 分解 WRA1 中的数值,暂存在当前工作区中。
12BB:	E1	POP	HL	; 取回缓冲区地址。
12BC:	C1	POP	BC	; 取回计数。
12BD:	118C13	LD	DE, 138CH	; DE 等于 1×10^{15} 至 1×10^{6} 的以 10 为幂的表地址。
12C0:	3E0A	LD	A, 0AH	; A=当前数值除以 10 的幂的次数。
12C2:	CD9112	CALL	1291H	; 把小数点和逗号放入缓冲器。
12C5:	C5	PUSH	BC	; 把小数点之前的数字个数暂存起来。
12C6:	F5	PUSH	AF	; 暂存除计数。
12C7:	E5	PUSH	HL	; 暂存当前数据缓冲区地址。
12C8:	D5	PUSH	DE	; 把 10 的幂次表放入堆栈。
12C9:	062F	LD	B, 2FH	; B=每次除以后,以 ASCII 码表示的商。
12CB:	04	INC	B	; B 从 30H 开始(代表 ASCII 码的零)。
12CC:	E1	POP	HL	; HL=10 的幂次表的地址,是除数的地址。
12CD:	E5	PUSH	HL	; 把它暂存起来,以便在循环中可以取回。
12CE:	CD480D	CALL	0D48H	; 把当前数值(整数)除以 10 的幂次,从 10^{15} 开始,降到 10^{6}。
12D1:	30F8	JR	NC, 12CBH	; 循环,直到余数小于当前的幂次。
12D3:	E1	POP	HL	; 取回 10 的幂次表地址。
12D4:	CD360D	CALL	0D36H	; 把余数和当前 10 的幂次相加,使余数为正。
12D7:	EB	EX	DE, HL	; 把当前幂次表地址放入 DE 中。
12D8:	E1	POP	HL	; HL=当前显示缓冲区的地址。
12D9:	70	LD	(HL), B	; 数据放入缓冲区。
12DA:	23	INC	HL	; 地址加 1,指向下一个缓冲区位置。
12DB:	F1	POP	AF	; 取回状态标志,于是可以测试是否循环了 10 次。
12DC:	C1	POP	BC	; 取回计数。
12DD:	3D	DEC	A	; 循环了一次,计数减 1。
12DE:	20E2	JR	NZ, 12C2H	; 若还没有做完 10 次,继续循环。
12E0:	C5	PUSH	BC	; 暂存计数,
12E1:	E5	PUSH	HL	; 以及暂存当前缓冲区地址。
12E2:	211D41	LD	HL, 411DH	; 把双精度值的后一半数作为单精度值。
12E5:	CDB109	CALL	09B1H	; 移入 WRA1 中,
12E8:	180C	JR	12F6H	; 并把它转换成 ASCII 码。
12EA:	C5	PUSH	BC	; ★★★单精度数转换成整数 ★★★★★★★★★★

说明　将此数的整数除以 100,000 及 10,000,利用在 1335H 处的程序把余数变成 ASCII 码。

12EB:	E5	PUSH	HL	; 把计数和缓冲区地址暂存起来。
12EC:	CD0807	CALL	0708H	; 当前数值加 0.5,结果放在 BC 和 DE 中。
12EF:	3C	INC	A	; MSB 加 1。
12F0:	CDFB0A	CALL	0AFBH	; 把正的单精度数转换成整数,结果在 BC 和 DE 中。

地址	目的码	源程序		注释
12F3:	CDB409	CALL	09B4H	; 把 BC 和 DE 中的值放到 WRA1 中，它是原单精度数的整数部分。
12F6:	E1	POP	HL	; 取回 HL,即缓冲区的地址。
12F7:	C1	POP	BC	; 取回计数。
12F8:	AF	XOR	A	; A 和标志清零。
12F9:	11D213	LD	DE, 13D2H	; DE=整数 100,000 的地址。
12FC:	3F	CCF		; C 标志=用于除循环 12FCH—1327H 中的第一次开关量，
12FD:	CD9112	CALL	1291H	; 在数据缓冲区中置小数点和逗号。
1300:	C5	PUSH	BC	; 暂存计数。
1301:	F5	PUSH	AF	; 暂存循环计数的 C 标志。
1302:	E5	PUSH	HL	; 暂存缓冲区的地址。
1303:	D5	PUSH	DE	; 暂存除法表的地址。
1304:	CDBF09	CALL	09BFH	; 把当前的单精度的值放入 BC 和 DE 中。
1307:	E1	POP	HL	; HL=整数 100,000 的地址。
1308:	062F	LD	B, 2FH	; B=ASCII 码 0 减 1(30H−1)。
130A:	04	INC	B	; 给出 30H, 31H,… 相当于 ASCII 码的 0, 1,…。
130B:	7B	LD	A, E	; 整数值的 LSB。★★★ 本段程序使用减法来将当前值的整数部分除以 100,000，商在 B 寄存器中，以 ASCII 码表示 ★★★★★★★★★★★★★★★★★★★★★
130C:	96	SUB	A, (HL)	; 减去 100,000 的 LSB。
130D:	5F	LD	E, A	; 把差值放回去,准备下一次减法。
130E:	23	INC	HL	; 指向 100,000 的下一个字节。
130F:	7A	LD	A, D	; 取整数值的中间字节。
1310:	9E	SBC	A, (HL)	; 减 100,000 的中间字节。
1311:	57	LD	D, A	; 把差值放回去,准备下一次减法。
1312:	23	INC	HL	; 地址加 1, 指向 100,000 的 MSB。
1313:	79	LD	A, C	; 取整数的 MSB。
1314:	9E	SBC	A, (HL)	; 减去 100,000 的 MSB。
1315:	4F	LD	C, A	; 把差值放回去,准备下一次减法。
1316:	2B	DEC	HL	; 把 HL 恢复为指向 100,000 的 LSB。
1317:	2B	DEC	HL	;
1318:	30F0	JR	NC, 130AH	; 循环,直到整型等效值小于 100,000。
131A:	CDB707	CALL	07B7H	; 将 100,000 加 C 和 DE 中的值,使余数为正。
131D:	23	INC	HL	; HL 加 1, 指向常数 10,000 的地址。
131E:	CDB409	CALL	09B4H	; 把余数作为当前值存储起来(存入 WRA1)。
1321:	EB	EX	DE, HL	; 将常数 10,000 的地址送入 DE。
1322:	E1	POP	HL	; HL=当前 PBUF (显示缓冲区)地址。
1323:	70	LD	(HL), B	; 存储 ASCII 商。
1324:	23	INC	HL	; HL 加 1, 指向 PBUF 的下一个位置。
1325:	F1	POP	AF	; 恢复 C 标志(开关)。

地址	目的码	源程序		注　释
1326:	C1	POP	BC	; 恢复 BC，由此，随后可将它存储起来。
1327:	38D3	JR	C, 12FCH	; 若 C 标志置位，则将它复位，并将余数除以 10,000。
1329:	13	INC	DE	; 在 C 标志不置位时，我们已将单精度值的整数部分除以了 100,000 和 10,000。余数为正，并已作为当前值被存储起来。
132A:	13	INC	DE	; DE 加 1，指向常数 1,000。
132B:	3E04	LD	A, 04H	; A＝数字的个数。
132D:	1806	JR	1335H	; 转移，去将余数转换成 4 个 ASCII 数字。
132F:	D5	PUSH	DE	; 存储编辑标志。★★★将整数转换成 ASCII ★★★★
1330:	11D813	LD	DE, 13D8H	; DE＝从 10,000 开始的 10 的乘幂的降序表。
1333:	3E05	LD	A, 05H	; A＝要建立的 ASCII 数字的位数。
1335:	CD9112	CALL	1291H	; 把小数点或逗号加到缓冲区中。
1338:	C5	PUSH	BC	; 保存计数。
1339:	F5	PUSH	AF	; 保存位数的计数。
133A:	E5	PUSH	HL	; 保存缓冲区地址。
133B:	EB	EX	DE, HL	; HL＝乘幂表的地址。
133C:	4E	LD	C, (HL)	; 将 10 的乘幂送入 BC。
133D:	23	INC	HL	; 指向 MSB 或乘幂。
133E:	46	LD	B, (HL)	; 装入 MSB 或乘幂。
133F:	C5	PUSH	BC	; 保存乘幂。
1340:	23	INC	HL	; HL 加 1，指向乘幂表中下一个值。
1341:	E3	EX	(SP), HL	; HL＝刚装入的值，下个值的地址送入堆栈。
1342:	EB	EX	DE, HL	; DE＝已装入的值——部分。
1343:	2A2141	LD	HL, (4121H)	; HL＝当前值(整数)。
1346:	062F	LD	B, 2FH	; 将当前值除以从 10,000 开始的 10 的乘幂，
1348:	04	INC	B	; 一直进行到除以 10。
1349:	7D	LD	A, L	; 将每次除的余数加起来，
134A:	93	SUB	A, E	; 其和就成为下次除法的被除数。
134B:	6F	LD	L, A	; 除法是用复合减法进行的。
134C:	7C	LD	A, H	; 商 ＋30H＝商的 ASCII 等效值。
134D:	9A	SBC	A, D	; B 寄存器＝商。
134E:	67	LD	H A	; HL＝下一个被除数。
134F:	30F7	JR	NC, 1348H	; 循环，直到商 (HL) 小于当前 10 的乘幂。
1351:	19	ADD	HL, DE	; 余数＋除数＝被除数。
1352:	222141	LD	(4121H), HL	; 存储下一个被除数。
1355:	D1	POP	DE	; DE＝下一个 10 的乘幂的地址。
1356:	E1	POP	HL	; 恢复输出缓冲区的地址。
1357:	70	LD	(HL), B	; 将 ASCII 数字送入缓冲区。
1358:	23	INC	HL	; 指向缓冲区中下一个地址。
1359:	F1	POP	AF	; A＝转换位数的计数值。
135A:	C1	POP	BC	; 恢复小数点前后的＃号的计数。

地址	目的码	源程序		注　释
135B:	3D	DEC	A	; 已经变换5位了吗?
135C:	20D7	JR	NZ, 1335H	; 没有,则循环。
135E:	CD9112	CALL	1291H	; 将小数点和逗号送入数字缓冲区。
1361:	77	LD	(HL), A	; 零结束符送入 PBUF。
1362:	D1	POP	DE	; 恢复调用它的程序的 DE。
1363:	C9	RET		; 返回到调用它的程序。 ★★★★★★★★★★★★★
1364:	00	NOP		; (1364H—136BH)=1.0×10E9 (双精度)。
1365:	00	NOP		;
1366:	00	NOP		;
1367:	00	NOP		;
1368:	F9	LD	SP, HL	;
1369:	02	LD	(BC), A	;
136A:	15	DEC	D	;
136B:	A2	AND	D	;
136C:	FD	?		; (136DH—1373H)=1.0×10D15 (双精度)。
136D:	FF	RST	38H	;
136E:	9F	SBC	A, A	;
136F:	31A95F	LD	SP, 5FA9H	;
1372:	63	LD	H, E	;
1373:	B2	OR	D	;
1374:	FEFF	CP	FFH	; (1374H—137BH)=1.0×10D16 (双精度)。
1376:	03	INC	BC	;
1377:	BF	CP	A	;
1378:	C9	RET		;
1379:	1B	DEC	DE	;
137A:	0EB6	LD	C, B6H	;
137C:	00	NOP		; (137CH—1383H) = 0.5(双精度)。
137D:	00	NOP		;
137E:	00	NOP		;
137F:	00	NOP		;
1380:	00	NOP		; (1380H—1383H)=0.5 (单精度)。
1381:	00	HOP		;
1382:	00	NOP		;
1383:	80	ADD	A, B	;
1384:	00	NOP		; (1384H—138BH)=1.0×10D16 (双精度)。
1385:	00	NOP		;
1386:	04	INC	B	;
1387:	BF	CP	A	;
1388:	C9	RET		;
1389:	1B	DEC	DE	;
138A:	0EB6	LD	C, B6H	; (138AH—138DH)=0.502778 (单精度)

地址	目的码	源程序		注　释
138C:	00	NOP		;(138CH—1392H)=1.0×10D15(双精度值的整数部分)
138D:	80	ADD	A, B	;
138E:	C6A4	ADD	A, A4H	;
1390:	7E	LD	A, (HL)	;
1391:	8D	ADC	A, L	;
1392:	03	INC	BC	;
1393:	00	NOP		; (1393H—1399H) = 1.0 × 10E14 (双精度值的整数部分)。
1394:	40	LD	B, B	;
1395:	7A	LD	A, D	;
1396:	10F3	DJNZ	138BH	;
1398:	5A	LD	E, D	;
1399:	00	NOP		;
139A:	00	NOP		; (139AH—13A0H) = 1.0 × 10E13 (双精度值的整数部分)。
139B:	A0	AND	B	;
139C:	72	LD	(HL), D	;
139D:	4E	LD	C, (HL)	;
139E:	1809	JR	13A9H	;
13A0:	00	NOP		;
13A1:	00	NOP		; (13A1H—13A7H) = 1.0 × 10E12 (双精度值的整数部分)。
13A2:	10A5	DJNZ	1349H	;
13A4:	D4E800	CALL	NC, 00E8H	;
13A7:	00	NOP		;
13A8:	00	NOP		; (13A8H—13AEH) = 1.0 × 10E11 (双精度值的整数部分)。
13A9:	E8	RET	PE	;
13AA:	76	HALT		;
13AB:	48	LD	C, B	;
13AC:	17	RLA		;
13AD:	00	NOP		;
13AE:	00	NOP		;
13AF:	00	NOP		; (13AFH—13B5H) = 1.0 × 10E10 (双精度值的整数部分)。
13B0:	E40B54	CALL	PO, 540BH	;
13B3:	02	LD	(BC), A	;
13B4:	00	NOP		;
13B5:	00	NOP		;
13B6:	00	NOP		; (13B6H—13BCH) = 1.0 × 10E9 (双精度值的整数部分)。

地址	目的码	源程序		注释
13B7:	CA9A3B	JP	Z, 3B9AH	;
13BA:	00	NOP		;
13BB:	00	NOP		;
13BC:	00	NOP		;
13BD:	00	NOP		; (13BDH—13C3H)=1.0×10E8(双精度值的整数部分)。
13BE:	E1	POP	HL	;
13BF:	F5	PUSH	AF	;
13C0:	05	DEC	B	;
13C1:	00	NOP		;
13C2:	00	NOP		;
13C3:	00	NOP		;
13C4:	80	ADD	A, B	; (13C4H—13CAH) = 1.0 × 10E7 (双精度值的整数部分)。
13C5:	96	SUB	A, (HL)	;
13C6:	98	SBC	A, B	;
13C7:	00	NOP		;
13C8:	00	NOP		;
13C9:	00	NOP		;
13CA:	00	NOP		;
13CB:	40	LD	B, B	; (13CBH—13D1H)= 1,000,000 (双精度值的整数部分)。
13CC:	42	LD	B, D	;
13CD:	0F	RRCA		;
13CE:	00	NOP		;
13CF:	00	NOP		;
13D0:	00	NOP		;
13D1:	00	NOP		;
13D2:	A0	AND	B	; (13D2H—13D3H)=100,000
13D3:	86	ADD	A, (HL)	;
13D4:	011027	LD	BC, 2710H	; (13D5H—13D6H)=10,000
13D7:	00	NOP		;
13D8:	1027	DJNZ	1401H	; (13D8H—13D9H) = 10,000 ★★★ 10 的乘幂的整数表 ★★★★★☆★☆★☆★★★★★★★★★★★★★★★★
13DA:	E8	RET	PE	; (13DAH—13DBH)=1,000
13DB:	03	INC	BC	;
13DC:	64	LD	H, H	; (13DCH—13DDH)=100
13DD:	00	NOP		;
13DE:	0A	LD	A, (BC)	; (13DEH—13DFH)=10
13DF:	00	NOP		;
13E0:	010021	LD	BC, 2100H	; 13E1H:NOP ★★★★★★★★★★★★★★★★★★★
13E3:	82	ADD	A, D	; 13E2H:LD HL ,0982H; 浮点数由负转换为正的地址。

地址	目的码	源程序		注　释
13E4:	09	ADD	HL, BC	;
13E5:	E3	EX	(SP), HL	; 把转换子程序的地址送入堆栈。
13E6:	E9	JP	(HL)	; 返回到调用它的程序。
13E7:	CDA409	CALL	09A4H	; 将当前单精度值送入堆栈。★★★ SQR 子程序★★★
13EA:	218013	LD	HL, 1380H	; HL=单精度值0.5的地址。★★★计算 $X^{0.5}$ (用13F2H通用子程序)。
13ED:	CDB109	CALL	09B1H	; 把0.5送入 BC 和 DE 并送入 WRA1。
13F0:	1803	JR	13F5H	; 进入用于求 X^Y 的通用程序。
13F2:	CDB10A	CALL	0AB1H	; ★★★ X^Y 子程序★★使用方法是 $e^{Y\ln X}$ ★★★★将4121H—4122H 中的整数转换成单精度数，并存入4121H—4124H。
13F5:	C1	POP	BC	; 把要求乘方的值送入 BC 和 DE,
13F6:	D1	POP	DE	; (即把底数送入 BCDE)。
13F7:	CD5509	CALL	0955H	; 测试指数的符号。
13FA:	78	LD	A, B	; A=要求乘方的数的 MSB。
13FB:	283C	JR	Z, 1439H	; 若指数为0，则转移。
13FD:	F20414	JP	P, 1404H	; 若指数为正，则转移。
1400:	B7	OR	A	; 测试求乘方的值。
1401:	CA9A19	JP	Z, 199AH	; 若求0的负数乘方，则转移。
1404:	B7	OR	A	; 再测试求乘方的值。
1405:	CA7907	JP	Z, 0779H	; 求0的正数乘方。
1408:	D5	PUSH	DE	; 把求乘方的值存入堆栈。
1409:	C5	PUSH	BC	; 底值的 EXP 和 MSB。
140A:	79	LD	A, C	; A=求乘方的值的 MSB。
140B:	F67F	OR	7FH	; 测试底数的符号，在它是负数时，置位位0—位6。
140D:	CDBF09	CALL	09BFH	; 把指数(幂)装入 BC 和 DE。
1410:	F22114	JP	P, 1421H	; 若底是正数，则转移。
1413:	D5	PUSH	DE	; 把指数存入堆栈。
1414:	C5	PUSH	BC	; 指数值的 EXP 和 MSB。
1415:	CD400B	CALL	0B40H	; 将指数的整数部分放入 A，截尾后的浮点部分放入 WRA1。
1418:	C1	POP	BC	; 恢复指数，
1419:	D1	POP	DE	; 以单精度值存放在 BC 和 DE 中。
141A:	F5	PUSH	AF	; 保存指数的整数部分。
141B:	CD0C0A	CALL	0A0CH	; 原始的指数与截尾后的指数比较，由此可知是否是整个的指数。
141E:	E1	POP	HL	; H=指数(整数)。
141F:	7C	LD	A, H	; A=指数。
1420:	1F	RRA		; 若指数是奇数，则置位 C 标志。
1421:	E1	POP	HL	; 取回单精度的指数，
1422:	222341	LD	(4123H), HL	; 送入 WRA1。

地址	目的码	源程序		注　释
1425:	E1	POP	HL	; 取指数的剩余部分，
1426:	222141	LD	(4121H), HL	; 并送入 WRA1。
1429:	DCE213	CALL	C, 13E2H	; 若指数是奇数并且底数是负数，则调用 13E2H 子程序
142C:	CC8209	CALL	Z, 0982H	; 若指数是整数并且底数是负数，则调用 0982H 子程序
142F:	D5	PUSH	DE	; 保存指数。
1430:	C5	PUSH	BC	; 指数值的 EXP 和 MSB
1431:	CD0908	CALL	0809H	; 求底值的对数。在负底数求小数的乘方时，
1434:	C1	POP	BC	; 给出 'ILLEGAL FUNCTION CALL'。
1435:	D1	POP	DE	; 恢复指数。
1436:	CD4708	CALL	0847H	; ln (值)乘以指数，然后计算 $e^{\ln(值)\times指数}$
1439:	CDA409	CALL	09A4H	; 把指数送入堆栈。★★★ 计算 e^X ★★★★★★★★
143C:	013881	LD	BC, 8138H	; BCDE=1.4427 (约 ln2+1≈2)。
143F:	113BAA	LD	DE, AA3BH	;
1442:	CD4708	CALL	0847H	; 指数值乘以 1.4427(2ln2)。
1445:	3A2441	LD	A, (4124H)	; A=乘积的指数。
1448:	FE88	CP	88H	; 测试指数，看它整数部分是否多于 8 位。

说明　方法：1.计算 $X=X\times 2\ln 2$

2.分离出 X 的整数部分，如果它大于 88，则退出，并给出溢出错误。

3.用步骤 2 得到的整数，计算 $Y=(2^{整数})\times 2$。

4.步骤 2 得到的整数加 1。

5.步骤 4 的结果乘以 ln2。

6. X 的原始值减去步骤 5 的结果，并反转结果的符号。

7.用步骤 1 计算的 X 值计算级数：

$$(((((((X\times C_0+C_1)X+C_2)X+C_3)X+C_4)X+C_5)X+C_6)X+C_7)$$

8.级数的最后项乘以步骤 3 中计算出的值。

144A:	D23109	JP	NC, 0931H	; 如果指数的整数部分多于 8 位，则转移。
144D:	CD400B	CALL	0B40H	; 整数部分少于 8 位，取整数部分，
1450:	C680	ADD	A, 80H	; 并放入 A 寄存器，
1452:	C602	ADD	A, 02H	; 然后测试它。
1454:	DA3109	JP	C, 0931H	; 如果指数× 2 ln2≥126，则转移。
1457:	F5	PUSH	AF	; 将(整数+82H) 保存在堆栈。
1458:	21F807	LD	HL, 07F8H	; 单精度值 1.0 的地址。
145B:	CD0B07	CALL	070BH	; 加 INT (EXP×2 ln2)。其中 EXP 为指数。
145E:	CD4108	CALL	0841H	; 乘 ln2
1461:	F1	POP	AF	; 清除堆栈(取整的 EXP×2ln2)。
1462:	C1	POP	BC	; 然后把原始的指数装入 BC 和 DE。
1463:	D1	POP	DE	;
1464:	F5	PUSH	AF	; 保存取整的 EXP×2ln2。
1465:	CD1307	CALL	0713H	; 取整后的指数减去原始的指数。
1468:	CD8209	CALL	0982H	; 使差成为正数。
146B:	217914	LD	HL, 1479H	; 八个系数的地址。
146E:	CDA914	CALL	14A9H	; 计算级数

地址	目的码	源程序		注　释
1471:	110000	LD	DE, 0000H	; 把 EXP×2ln2 的整数值装入 BC 和 DE。
1474:	C1	POP	BC	;
1475:	4A	LD	C, D	; C 寄存器清零。
1476:	C34708	JP	0847H	; 乘级数的和，并返回调用它的程序。
1479:	08	EX	AF, AF′	; 计算 e^X 的级数中使用的系数表中系数的个数(08)
147A:	40	LD	B, B	; (147AH—147DH)=—1.41316×10E—4
147B:	2E94	LD	L, 94H	;
147D:	74	LD	(HL), H	;
147E:	70	LD	(HL), B	; (147EH—1481H)=1.32988×10E—3=1/6!
147F:	4F	LD	C, A	;
1480:	2E77	LD	L, 77H	;
1482:	6E	LD	L, (HL)	; (1482H—1485H)=—8.30136×10E—3=—1/5!
1483:	02	LD	(BC), A	;
1484:	88	ADC	A, B	;
1485:	7A	LD	A, D	;
1486:	E6A0	AND	A0H	; (1486H—1489H)=0.0416574≒1/4!
1488:	2A7C50	LD	HL, (507CH)	; (148AH—148DH)=—0.166665≒—1/3!
148B:	AA	XOR	D	;
148C:	AA	XOR	D	;
148D:	7E	LD	A, (HL)	;
148E:	FF	RST	38H	; (148EH—1491H)=0.5
148F:	FF	RST	38H	;
1490:	7F	LD	A, A	;
1491:	7F	LD	A, A	;
1492:	00	NOP		; (1492H—1495H)=—1.0
1493:	00	NOP		;
1494:	80	ADD	A, B	;
1495:	81	ADD	A, C	;
1496:	00	NOP		; (1496H—1499H)=1.0
1497:	00	NOP		;
1498:	00	NOP		;
1499:	81	ADD	A, C	;
149A:	CDA409	CALL	09A4H	; 将 X 值送入堆栈。★★★ 计算级数和的通用求和程序
				★★★★★★★★★★★★★★★★★★★★★★★★★

说明　当在 149AH 进入时，对于 I=0 到 N，计算级数和：

$$((((X^2\times C_0+C_1)X^2+C_2X^2+\cdots C_N)X$$

在 149AH 的第二个入口点可用于求级数和：

$$((((X\times C_0+C_1)X+C_2)X+C+C_3)X+\cdots C_N,$$

I=0 到 N。在入口时，X 项在 BC 和 DE 中，HL 指向系数表，表的第一个数为项数，其后是系数。

149D:	11320C	LD	DE, 0C32H	; 然后将返回地址 0C32H 推入堆栈，
14A0:	D5	PUSH	DE	; 在返回前，它将计算最后一项。

地址	目的码	源程序		注　释
14A1:	E5	PUSH	HL	; 保存项数、系数的地址。
14A2:	CDBF09	CALL	09BFH	; 把值送入 BC 和 DE。
14A5:	CD4708	CALL	0847H	; X 值自乘。
14A8:	E1	POP	HL	; 取回系数的地址。
14A9:	CDA409	CALL	09A4H	; 将 X 值或 X^2 值送入堆栈。
14AC:	7E	LD	A, (HL)	; A＝项数。
14AD:	23	INC	HL	; HL＝下一个系数的地址。
14AE:	CDB109	CALL	09B1H	; 取入 HL 指向的系数，并将它送入 WRA1。HL 指向下个系数值。
14B1:	06F1	LD	B, F1H	; 14B2H: POP AF　　;取回剩余系数的个数。
14B3:	C1	POP	BC	; BC 和 DE＝在 14A9H 存入的 X 值。
14B4:	D1	POP	DE	;
14B5:	3D	DEC	A	; 计算了一项，计数减 1 。
14B6:	C8	RET	Z	; 如果计算了所有项，则返回。
14B7:	D5	PUSH	DE	; BCDE＝X 值，
14B8;	C5	PUSH	BC	; 将 X 值存入堆栈，由此能重新使用。
14B9:	F5	PUSH	AF	; 保存要计算的剩余项个数，
14BA:	E5	PUSH	HL	; 以及指向下一个系数的指针 HL。
14BB:	CD4708	CALL	0847H	; 计算 C(I)×X。
14BE:	E1	POP	HL	; 恢复系数表地址。
14BF:	CDC209	CALL	09C2H	; 将下个系数从 HL 指示的表中送入 BCDE，随后指向下一个值。
14C2:	E5	PUSH	HL	; 保存下一个系数的地址。
14C3:	CD1607	CALL	0716H	; 计算 C(I)×X+C(I+1)。
14C6:	E1	POP	HL	; 取回系数表地址。
14C7:	18E9	JR	14B2H	; 继续求级数。WRA1＝当前项。
14C9:	CD7F0A	CALL	0A7FH	; 把值转换成整数。★★★ RND 子程序 ★★★★★★
14CC:	7C	LD	A, H	; A＝自变量的 MSB。
14CD:	B7	OR	A	; 置状态标志。
14CE:	FA4A1E	JP	M, 1E4AH	; 如果是负数，即 RND(A)中 A 是负数，则为 FC 错误。
14D1:	B5	OR	L	; 合并 MSB 和 LSB，置状态标志。
14D2:	CAF014	JP	Z, 14F0H	; 若参数为 0 ，即 RND(0)，则转移。
14D5:	E5	PUSH	HL	; 保存参数(用于计算 RND(X)的 X 值)。
14D6:	CDF014	CALL	14F0H	; 计算 RND(0)。
14D9:	CDBF09	CALL	09BFH	; 将随机数送入 BCDE。
14DC:	EB	EX	DE, HL	; 现在将随机数存入堆栈。
14DD:	E3	EX	(SP), HL	; 把原始参数送入 HL。
14DE:	C5	PUSH	BC	; 存 RND(0) 值于堆栈。
14DF:	CDCF0A	CALL	0ACFH	; 把原始参数转换成单精度值。
14E2:	C1	POP	BC	; 取回在 14D6H 调用子程序时求得的 RND(0) 值。
14E3:	D1	POP	DE	;

地址	目的码	源程序		注释
14E4:	CD4708	CALL	0847H	; 然后，RND(0) 乘以参数。
14E7:	21F807	LD	HL, 07F8H	; HL=单精度值 0.1 的地址。
14EA:	CD0B07	CALL	070BH	; 将 1.0 加当前值。
14ED:	C3400B	JP	0B40H	; 转换成整数，并返回到调用它的程序
14F0:	219040	LD	HL, 4090H	; HL=3 个字节标志表的地址。★★★ RND(0) ★★★
14F3:	E5	PUSH	HL	; 把标志表地址存入堆栈。
14F4:	110000	LD	DE, 0000H	; DE=起始值的中间字节和 LSB。
14F7:	4B	LD	C, E	; C=起始值的 MSB。
14F8:	2603	LD	H, 03H	; H=全部外循环的次数计数。
14FA:	2E08	LD	L, 08H	; L=全部内循环的次数。
14FC:	EB	EX	DE, HL	; 把当前值的中间字节及 LSB 送入 HL。
14FD:	29	ADD	HL, HL	; 加倍。
14FE:	EB	EX	DE, HL	; 然后将它们送回 DE。
14FF:	79	LD	A, C	; 取当前值的 MSB。
1500:	17	RLA		; 加倍。
1501:	4F	LD	C, A	; 并送回它的源寄存器。
1502:	E3	EX	(SP), HL	; 保存计数器，将标志字的地址送入 HL。
1503:	7E	LD	A, (HL)	; A=标志字。
1504:	07	RLCA		; 乘以 2 。
1505:	77	LD	(HL), A	; 再送回原处。
1506:	E3	EX	(SP), HL	; 计数器送回 HL。
1507:	D21615	JP	NC, 1516H	; 若标志字起初不溢出，则转移。
150A:	E5	PUSH	HL	; 标志字溢出，保存计数器。
150B:	2AAA40	LD	HL, (40AAH)	; 种子的最低两位有效字节。
150E:	19	ADD	HL, DE	; 将种子加起始值。
150F:	EB	EX	DE, HL	; 把新的种子送入 DE。
1510:	3AAC40	LD	A, (40ACH)	; 种子的 MSB。
1513:	89	ADC	A, C	; 加起始值的 MSB。
1514:	4F	LD	C, A	; 新的起始值 MSB 送回源寄存器。
1515:	E1	POP	HL	; 恢复计数器。
1516:	2D	DEC	L	; 全部内循环次数的计数减 1 。
1517:	C2FC14	JP	NZ, 14FCH	; 如果不是 8 次，则转移。
151A:	E3	EX	(SP), HL	; 保存计数器，HL=标志字的地址。
151B:	23	INC	HL	; HL 加 1 ，指向下一个标志字。
151C:	E3	EX	(SP), HL	; 恢复计数器，新的标志字地址送入堆栈。
151D:	25	DEC	H	; 全部外循环的次数减 1 。
151E:	C2FA14	JP	NZ, 14FAH	; 若不够 3 次，则转移。
1521:	E1	POP	HL	; 从堆栈中清除标志表地址。
1522:	2165B0	LD	HL, B065H	; HL=原始种子的中间字节和 LSB。
1525:	19	ADD	HL, DE	; 加当前值。
1526:	22AA40	LD	(40AAH), HL	; 并作为新的种子值存储起来。

地址	目的码	源程序		注释
1529:	CDEF0A	CALL	0AEFH	; 置当前数据类型为单精度。
152C:	3E05	LD	A, 05H	; 当前值的 MSB 加 5，
152E:	89	ADC	A, C	; 并作为种子的 MSB 存储起来。
152F:	32AC40	LD	(40ACH), A	;
1532:	EB	EX	DE, HL	; 把中间字节和 LSB 送入 DE，于是，我们有了 BCDE 这样的排列。
1533:	0680	LD	B, 80H	; B=符号标志和指数。
1535:	212541	LD	HL, 4125H	; HL=符号标志字地址。
1538:	70	LD	(HL), B	; 置符号标志为正。
1539:	2B	DEC	HL	; HL 减 1，指向指数。
153A:	70	LD	(HL), B	; 置指数为 80H，所以值将小于 1。
153B:	4F	LD	C, A	; C=新的 MSB（在 152EH 计算的）。
153C:	0600	LD	B, 00H	; B=0
153E:	C36507	JP	0765H	; 将值归一化并转移到 14D9H，如果不是 RND(0) 则返回调用它的程序。
1541:	218B15	LD	HL, 158BH	; 1.57 即 $\pi/2$ 的地址。★★★ COS 子程序 ★★★★★
1544:	CD0B07	CALL	070BH	; 1.5 加当前值。
1547:	CDA409	CALL	09A4H	; 把当前值存入堆栈。★★★ SIN 子程序★★★★★★

说明 方法：1.设 $X\leqslant 360°$
2.按 $X=X/360°$ 重新计算 X，于是 $X\leqslant 1$。
3.若 $X\leqslant 90°$ 至步骤 7。
4.若 $X\leqslant 180°$，则 $X=0.5-X$，至步骤 7。
5.若 $X\leqslant 270°$，则 $X=0.5-X$，至步骤 7。
6.按 $X=X-1.0$ 重新计算 X。
7.用幂级数计算 SIN。

154A:	014983	LD	BC, 8349H	; BCDE=单精度数 6.28（即 2π）。
154D:	11DB0F	LD	DE, 0FDBH	;
1550:	CDB409	CALL	09B4H	; 将 2π 送入 WRA1。
1553:	C1	POP	BC	; 把求 SIN 的值送入 BC 和 DE。
1554:	D1	POP	DE	;
1555:	CDA208	CALL	08A2H	; 值 $/2\pi$ 给出 $X/360°$。
1558:	CDA409	CALL	09A4H	; 把值 $/2\pi$ 送入堆栈。
155B:	CD400B	CALL	0B40H	; 将结果转换成整数，于是可以分离出余数。
155E:	C1	POP	BC	; BCDE=值/2π 的商和余数。
155F:	D1	POP	DE	;
1560:	CD1307	CALL	0713H	; 原始值减去值的整数部分（分离出 X 的小数部分）。
1563:	218F15	LD	HL, 158FH	; 单精度值（0.25）的地址。
1566:	CD1007	CALL	0710H	; 小数部分减去 0.25。测试是否 $\leqslant 90°$。
1569:	CD5509	CALL	0955H	; 测试差的符号。
156C:	37	SCF		; 如果是正的，跳过 1582H 的符号反向子程序。
156D:	F27715	JP	P, 1577H	; 若 $<90°$，转移，去加上减数 0.250。
1570:	CD0807	CALL	0708H	; 差加 0.5。

地址	目的码	源程序		注释
1573:	CD5509	CALL	0955H	; 测试当前值的符号，看是否大于 0.75（即小于 270°）。
1576:	B7	OR	A	; 置状态标志。
1577:	F5	PUSH	AF	; 保存符号指针（正＝＋1，负＝－1）。
1578:	F48209	CALL	P, 0982H	; 若是正的，使它为负（给出 $X-1.0$）
157B:	218F15	LD	HL, 158FH	; 单精度值（0.250）的地址。
157E:	CD0B07	CALL	070BH	; WRA1 中的当前值加 0.250。
1581:	F1	POP	AF	; 取符号反向标志。
1582:	D48209	CALL	NC, 0982H	; 根据象限置 X 项的符号。
1585:	219315	LD	HL, 1593H	; 系数的地址。
1588:	C39A14	JP	149AH	; 计算级数，并返回调用它的程序。
158B:	DB0F	IN	A, (0FH)	; (158BH—158EH)＝1.5 单精度★★★★★★★★★★
158D:	49	LD	C, C	;
158E:	81	ADD	A, C	;
158F:	00	NOP		; (158FH—1592H)＝0.25（单精度）
1590:	00	NOP		;
1591:	00	NOP		;
1592:	7F	LD	A, A	;
1593:	05	DEC	B	; 其后面的系数值的个数（05）。★★★ 计算 SIN 幂级数的系数 ★★★★★★★★★★★★★★★★★★★★
1594:	BA	CP	D	; (1594H—1597H)＝39.7107（单精度）。
1595:	D7	RST	10H	;
1596:	1E86	LD	E, 86H	;
1598:	64	LD	H, H	; (1598H—159BH)＝－76.576（单精度）。
1599:	2699	LD	H, 99H	;
159B:	87	ADD	A, A	;
159C:	58	LD	E, B	; (159CH—159FH)＝81.6022（单精度）。
159D:	34	INC	(HL)	;
159E:	23	INC	HL	;
159F:	87	ADD	A, A	;
15A0:	E0	RET	PO	; (15A0H—15A3H)＝－41.3417（单精度）。
15A1:	5D	LD	E, L	;
15A2:	A5	AND	L	;
15A3:	86	ADD	A, (HL)	;
15A4:	DA0F49	JP	C, 490FH	; (15A4H—15A7H)＝6.28319（单精度）。
15A7:	83	ADD	A, E	;
15A8:	CDA409	CALL	09A4H	; 将 WRA1 送入堆栈。★★★ TAN 子程序★★★★★ 利用恒等式：TAN(X) = SIN(X)/COS(X)
15AB:	CD4715	CALL	1547H	; 计算 SIN(X)。
15AE:	C1	POP	BC	; 恢复 BC 和 DE 的原始值。
15AF:	E1	POP	HL	;
15B0:	CDA409	DALL	09A4H	; 将 SIN(X) 送入堆栈。

地址	目的码	源程序		注释
15B3:	EB	EX	DE, HL	; 在 BC 和 DE 中给出了原始值。
15B4:	CDB409	CALL	09B4H	; 把原始值送入 WRA1。
15B7:	CD4115	CALL	1541H	; 计算 COS(X)。
15BA:	C3A008	JP	08A0H	; 计算 TAN(X)=SIN(X)/COS(X)，并返回。
15BD:	CD5509	CALL	0955H	; 测试正切的符号。★★★ ATN 子程序 ★★★★★★

说明 方法：1.测试正切的符号，若是负的，则角度在第二和第四象限。在出口时，置位标志，使结果为正。如果值是负的，则反转其符号。

2.测试正切的大小，若小于 1，则去步骤 3，否则计算它的倒数，并将返回地址放入堆栈，在返回地址将计算 $\pi/2$ 减级数值。

3.计算级数：$(((X^2\times C_0+C_1)X^2+C_2)\cdots C_8)X$

4.如果步骤 1 得到的标志没置位，则反转级数结果的符号。

5.若原始值小于 1，则返回调用它的程序，否则计算 $\pi/2$ 减步骤 4 得到的值，然后返回。

15C0:	FCE213	CALL	M, 13E2H	; 若是负的，则将由正到负的变换地址放入堆栈，以给出适当的结果。
15C3:	FC8209	CALL	M, 0982H	; 将当前值由负变换成正。
15C6:	3A2441	LD	A, (4124H)	; 取入正切的指数。
15C9:	FE81	CP	81H	; 测试该值是否大于 1。
15CB:	380C	JR	C, 15D9H	; 若值小于 1，则转移。
15CD:	010081	LD	BC, 8100H	; 在 BCDE 中设置一个浮点数+1。
15D0:	51	LD	D, C	;
15D1:	59	LD	E, C	;
15D2:	CDA208	CALL	08A2H	; 取正切的倒数。
15D5:	211007	LD	HL, 0710H	; 在级数后面调用的减法子程序的地址，将
15D8:	E5	PUSH	HL	; 从 $\pi/2$ 中减去最后项。
15D9:	21E315	LD	HL, 15E3H	; HL=单精度系数的地址。
15DC:	CD9A14	CALL	149AH	; 计算级数。
15DF:	218B15	LD	HL, 158BH	; 1.5708 (即 $\pi/2$) 的地址。
15E2:	C9	RET		; 从 $\pi/2$ 中减去最后项，并返回。关于返回，见步骤 2。
15E3:	09	ADD	HL, BC	; 下面单精度系数的个数 (09)。★★★ 求 TAN 的幂级数的系数 ★★★★★★★★★★★★★★★★★★★★
15E4:	4A	LD	C, D	; (15E4H—15E7H)=2.86623×10E−3
15E5:	D7	RST	10H	;
15E6:	3B	DEC	SP	;
15E7:	78	LD	A, B	;
15E8:	02	LD	(BC), A	; (15E8H—15EBH)=−0.0161657
15E9:	6E	LD	L, (HL)	;
15EA:	84	ADD	A, H	;
15EB:	7B	LD	A, E	;
15EC:	FEC1	CP	C1H	; (15ECH—15EFH)=0.0429096
15EE:	2F	CPL		;

地址	目的码	源程序		注　释
15EF:	7C	LD	A, H	;
15F0:	74	LD	(HL), H	; (15F0H—15F3H)=—0.0752896
15F1:	319A7D	LD	SP, 7D9AH	;
15F4:	84	ADD	A, H	; (15F4H—15F7H)=0.105586
15F5:	3D	DEC	A	;
15F6:	5A	LD	E, D	;
15F7:	7D	LD	A, L	;
15F8:	C8	RET	Z	; (15F8H—15FBH)=—0.142089
15F9:	7F	LD	A, A	;
15FA:	91	SUB	A, C	;
15FB:	7E	LD	A, (HL)	;
15FC:	E4BB4C	CALL	PO, 4CBBH	; (15FCH—15FFH)=0.199936
15FF:	7E	LD	A, (HL)	;
1600:	6C	LD	L, H	; (1600H—1603H)=—0.333331
1601:	AA	XOR	D	;
1602:	AA	XOR	D	;
1603:	7F	LD	A, A	;
1604:	00	NOP		; (1604H—1607H)=1.0000
1605:	00	NOP		;
1606:	00	NOP		;
1607:	81	ADD	A, C	;
1608:	8A	ADC	A, D	; SGN (098AH) ★★★内部函数的地址★★★★★★★
1609:	09	ADD	HL, BC	;
160A:	37	SCF		; INT (0B37H)
160B:	0B	DEC	BC	;
160C:	77	LD	(HL), A	; ABS (0977H)
160D:	09	ADD	HL, BC	;
160E:	D427EF	CALL	NC, EF27H	; FRE (27D4H)。1610H: INP (2AEFH)。
1611:	2AF527	LD	HL, (27F5H)	; 1612H: POS (27F5H)
1614:	E7	RST	20H	; SQR (13E7H)
1615:	13	INC	DE	;
1616:	C9	RET		; RND (14C9H)
1617:	14	INC	D	;
1618:	09	ADD	HL, BC	; LOG (0809H)
1619:	08	EX	AF, AF'	;
161A:	39	ADD	HL, SP	; EXP (1439H)
161B:	14	INC	D	;
161C:	41	LD	B, C	; COS (1541H)
161D:	15	DEC	D	;
161E:	47	LD	B, A	; SIN (1547H)
161F:	15	DEC	D	;

地址	目的码	源程序		注释
1620:	A8	XOR	B	; TAN (15A8H)
1621:	15	DEC	D	;
1622:	BD	CP	L	; ATN (15BDH)
1623:	15	DEC	D	;
1624:	AA	XOR	D	; PEEK (2CAAH)
1625:	2C	INC	L	;
1626:	52	LD	D, D	; CVI (4152H)
1627:	41	LD	B, C	;
1628:	58	LD	E, B	; CVS (4158H)
1629:	41	LD	B, C	;
162A:	5E	LD	E, (HL)	; CVD (415EH)
162B:	41	LD	B, C	;
162C:	61	LD	H, C	; EOF (4161H)
162D:	41	LD	B, C	;
162E:	64	LD	H, H	; LOC (4164H)
162F:	41	LD	B, C	;
1630:	67	LD	H, A	; LOF (4167H)
1631:	41	LD	B, C	;
1632:	6A	LD	L, D	; MKI$ (416AH)
1633:	41	LD	B, C	;
1634:	6D	LD	L, L	; MKS$ (416DH)
1635:	41	LD	B, C	;
1636:	70	LD	(HL), B	; MKD$ (4170H)
1637:	41	LD	B, C	;
1638:	7F	LD	A, A	; CINT (0A7FH)
1639:	0A	LD	A, (BC)	;
163A:	B1	OR	C	; CSNG (0AB1H)
163B:	0A	LD	A, (BC)	;
163C:	DB0A	IN	A, (0AH)	; CDBL (0ADBH)
163E:	260B	LD	H, 0BH	; FIX (0B26H)
1640:	03	INC	BC	; LEN (2A03H)
1641:	2A3628	LD	HL, (2836H)	; 1642H: STR$ (2836H)
1644:	C5	PUSH	BC	; VAL (2AC5H)
1645:	2A0F2A	LD	HL, (2A0FH)	; 1646H: ASC (2A0FH)
1648:	1F	RRA		; CHR$ (2A1FH)
1649:	2A612A	LD	HL, (2A61H)	; 164AH: LEFT$ (2A61H)
164C:	91	SUB	A, C	; RIGHT$ (2A91H)
164D:	2A9A2A	LD	HL,(2A9AH)	; 164EH: MID$ (2A9AH)
1650:	C5	PUSH	BC	; 代号 保留字 ★★★ 保留字表 ★★★★★★★★
				80H END
1651:	4E	LD	C, (HL)	;

地址	目的码	源程序			注 释
1652:	44	LD	B, H	;	
1653:	C64F	ADD	A, 4FH	; 81H	FOR
1655:	52	LD	D, D	;	
1656:	D24553	JP	NC, 5345H	; 82H	RESET
1659:	45	LD	B, L	;	
165A:	54	LD	D, H	;	
165B:	D345	OUT	(45H), A	; 83H	SET
165D:	54	LD	D, H	;	
165E:	C34C53	JP	534CH	; 84H	CLS
1661:	C34D44	JP	444DH	; 85H	CMD
1664:	D2414E	JP	NC, 4E41H	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★
				86H	RANDOM
1667:	44	LD	B, H	;	
1668:	4F	LD	C, A	;	
1669:	4D	LD	C, L	;	
166A:	CE45	ADC	A, 45H	; 87H	NEXT
166C:	58	LD	E, B	;	
166D:	54	LD	D, H	;	
166E:	C44154	CALL	NZ, 5441H	; 88H	DATA
1671:	41	LD	B, C	;	
1672:	C9	RET		; 89H	INPUT
1673:	4E	LD	C, (HL)	;	
1674:	50	LD	D, B	;	
1675:	55	LD	D, L	;	
1676:	54	LD	D, H	;	
1677:	C4494D	CALL	NZ, 4D49H	; 8AH	DIM
167A:	D24541	JP	NC, 4145H	; 8BH	READ
167D:	44	LD	B, H	;	
167E:	CC4554	CALL	Z, 5445H	; 8CH	LET
1681:	C7	RST	00H	; 8DH	GOTO
1682:	4F	LD	C, A	;	
1683:	54	LD	D, H	;	
1684:	4F	LD	C, A	;	
1685:	D2554E	JP	NC, 4E55H	; 8EH	RUN
1688:	C9	RET		; 8FH	IF
1689:	46	LD	B, (HL)	;	
168A:	D24553	JP	NC, 5345H	; 90H	RESTORE
168D:	54	LD	D, H	;	
168E:	4F	LD	C, A	;	
168F:	52	LD	D, D	;	
1690:	45	LD	B, L	;	

地址	目的码	源程序		注释		
1691:	C7	RST	00H	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★	
				91H	GOSUB	
1692:	4F	LD	C, A	;		
1693:	53	LD	D, E	;		
1694:	55	LD	D, L	;		
1695:	42	LD	B, D	;		
1696:	D24554	JP	NC, 5445H	; 92H	RETURN	
1699:	55	LD	D, L	;		
169A:	52	LD	D, D	;		
169B:	4E	LD	C, (HL)	;		
169C:	D2454D	JP	NC, 4D45H	; 93H	REM	
169F:	D354	OUT	(54H) ,A	; 94H	STOP	
16A1:	4F	LD	C, A	;		
16A2:	50	LD	D, B	;		
16A3:	C5	PUSH	BC	; 95H	ELSE	
16A4:	4C	LD	C, H	;		
16A5:	53	LD	D, E	;		
16A6:	45	LD	B, L	;		
16A7:	D4524F	CALL	NC, 4F52H	; 96H	TRON	
16AA:	4E	LD	C, (HL)	;		
16AB:	D4524F	CALL	NC, 4F52H	; 97H	TROFF	
16AE:	46	LD	B, (HL)	;		
16AF:	46	LD	B, (HL)	;		
16B0:	C44546	CALL	NZ, 4645H	; 98H	DEFSTR	
16B3:	53	LD	D, E	;		
16B4:	54	LD	D, H	;		
16B5:	52	LD	D, D	;		
16B6:	C44546	CALL	NZ, 4645H	; 99H	DEFINT	
16B9:	49	LD	C, C	;		
16BA:	4E	LD	C, (HL)	;		
16BB:	54	LD	D, H	;		
16BC:	C44546	CALL	NZ, 4645H	; 9AH	DEFSNG	
16BF:	53	LD	D, E	;		
16C0:	4E	LD	C, (HL)	;		
16C1:	47	LD	B, A	;		
16C2:	C44546	CALL	NZ, 4645H	; 9BH	DEFDBL	
16C5:	44	LD	B, H	;		
16C6:	42	LD	B, D	;		
16C7:	4C	LD	C, H	;		
16C8:	CC494E	CALL	Z, 4E49H	; 9CH	LINE	
16CB:	45	LD	B, L	;		

地址	目的码	源 程 序			注 释
16CC:	C5	PUSH	BC	; 9D	EDIT
16CD:	44	LD	B, H	;	
16CE:	49	LD	C, C	;	
16CF:	54	LD	D, H	;	
16D0:	C5	PUSH	BC	; 9EH	ERROR
16D1:	52	LD	D, D	;	
16D2:	52	LD	D, D	;	
16D3:	4F	LD	C, A	;	
16D4:	52	LD	D, D	;	
16D5:	D24553	JP	NC, 5345H	; 9FH	RESUME
16D8:	55	LD	D, L	;	
16D9:	4D	LD	C, L	;	
16DA:	45	LD	B ,L	;	
16DB:	CF	RST	08H	; A0H	OUT
16DC:	55	LD	D, L	;	
16DD:	54	LD	D, H	;	
16DE:	CF	RST	08H	; A1H	ON
16DF:	4E	LD	C, (HL)	;	
16E0:	CF	RST	08H	; A2H	OPEN
16E1:	50	LD	D, B	;	
16E2:	45	LD	B, L	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★
16E3:	4E	LD	C, (HL)	;	
16E4:	C649	ADD	A, 49H	; A3H	FIELD
16E6:	45	LD	B, L	;	
16E7:	4C	LD	C, H	;	
16E8:	44	LD	B, H	;	
16E9:	C7	RST	00H	; A4H	GET
16EA:	45	LD	B, L	;	
16EB:	54	LD	D, H	;	
16EC:	D0	RET	NC	; A5H	PUT
16ED:	55	LD	D, L	;	
16EE:	54	LD	D, H	;	
16EF:	C34C4F	JP	4F4CH	; A6H	CLOSE
16F2:	53	LD	D, E	;	
16F3:	45	LD	B, L	;	
16F4:	CC4F41	CALL	Z, 414FH	; A7H	LOAD
16F7:	44	LD	B, H	;	
16F8:	CD4552	CALL	5245H	; A8H	MERGE
16FB:	47	LD	B, A	;	
16FC:	45	LD	B, L	;	
16FD:	CE41	ADC	A, 41H	; A9H	NAME

地址	目的码	源程序			注释
16FF:	4D	LD	C, L	;	
1700:	45	LD	B, L	;	
1701:	CB49	BIT	1, C	; AAH	KILL
1703:	4C	LD	C, H	;	
1704:	4C	LD	C, H	;	
1705:	CC5345	CALL	Z, 4553H	; ABH	LSET
1708:	54	LD	D, H	;	
1709:	D25345	JP	NC, 4553H	; ACH	RSET
170C:	54	LD	D, H	;	
170D:	D341	OUT	(41H), A	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★
				ADH	SAVE
170F:	56	LD	D, (HL)	;	
1710:	45	LD	B, L	;	
1711:	D359	OUT	(59H), A	; AEH	SYSTEM
1713:	53	LD	D, E	;	
1714:	54	LD	D, H	;	
1715:	45	LD	B, L	;	
1716:	4D	LD	C, L	;	
1717:	CC5052	CALL	Z, 5250H	; AFH	LPRINT
171A:	49	LD	C, C	;	
171B:	4E	LD	C, (HL)	;	
171C:	54	LD	D, H	;	
171D:	C44546	CALL	NZ, 4645H	; B0H	DEF
1720:	D0	RET	NC	; B1H	POKE
1721:	4F	LD	C, A	;	
1722:	4B	LD	C, E	;	
1723:	45	LD	B, L	;	
1724:	D0	RET	NC	; B2H	PRINT
1725:	52	LD	D, D	;	
1726:	49	LD	C, C	;	
1727:	4E	LD	C, (HL)	;	
1728:	54	LD	D, H	;	
1729:	C34F4E	JP	4E4FH	; B3H	CONT
172C:	54	LD	D, H	;	
172D:	CC4953	CALL	Z, 5349H	; B4H	LIST
1730:	54	LD	D, H	;	
1731:	CC4C49	CALL	Z, 494CH	; B5H	LLIST
1734:	53	LD	D, E	;	
1735:	54	LD	D, H	;	
1736:	C4454C	CALL	NZ, 4C45H	; B6H	DELETE
1739:	45	LD	B, L	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★

地址	目的码	源程序			注释	
173A:	54	LD	D, H	;		
173B:	45	LD	B, L	;		
173C:	C1	POP	BC	; B7H	AUTO	
173D:	55	LD	D, L	;		
173E:	54	LD	D, H	;		
173F:	4F	LD	C, A	;		
1740:	C34C45	JP	454CH	; B8H	CLEAR	
1743:	41	LD	B, C	;		
1744:	52	LD	D, D	;		
1745:	C34C4F	JP	4F4CH	; B9H	CLOAD	
1748:	41	LD	B, C	;		
1749:	44	LD	B, H	;		
174A:	C35341	JP	4153H	; BAH	CSAVE	
174D:	56	LD	D, (HL)	;		
174E:	45	LD	B, L	;		
174F:	CE45	ADC	A, 45H	; BBH	NEW	
1751:	57	LD	D, A	;		
1752:	D44142	CALL	NC, 4241H	; BCH	TAB(	
1755:	28D4	JR	Z, 172BH	; BDH	TO	(1756H 起)
1757:	4F	LD	C, A	;		
1758:	C64E	ADD	A, 4EH	; BEH	FN	
175A:	D5	PUSH	DE	; BFH	USING	
175B:	53	LD	D, E	;		
175C:	49	LD	C, C	;		
175D:	4E	LD	C, (HL)	;		
175E:	47	LD	B, A	;		
175F:	D641	SUB	41H	; C0H	VARPTR	
1761:	52	LD	D, D	;		
1762:	50	LD	D, B	;		
1763:	54	LD	D, H	;		
1764:	52	LD	D, D	;		
1765:	D5	PUSH	DE	; C1H	USR	
1766:	53	LD	D, E	;		
1767:	52	LD	D, D	;		
1768:	C5	PUSH	BC	; C2H	ERL	
1769:	52	LD	D, D	;		
176A:	4C	LD	C, H	;		
176B:	C5	PUSH	BC	; C3H	ERR	
176C:	52	LD	D, D	;		
176D:	52	LD	D, D	;		
176E:	D354	OUT	(54H), A	; C4H	STRING$	

地址	目的码	源程序			注释	
1770:	52	LD	D, D	;		
1771:	49	LD	C, C	;		
1772:	4E	LD	C, (HL)	;		
1773:	47	LD	B, A	;		
1774:	24	INC	H	;		
1775:	C9	RET		;	C5H	INSTR
1776:	4E	LD	C, (HL)	;		
1777:	53	LD	D, E	;		
1778:	54	LD	D, H	;		
1779:	52	LD	D, D	;		
177A:	D0	RET	NC	;	C6H	POINT
177B:	4F	LD	C, A	;		
177C:	49	LD	C, C	;		
177D:	4E	LD	C, (HL)	;		
177E:	54	LD	D, H	;		
177F:	D4494D	CALL	NC, 4D49H	;	C7H	TIME$
1782:	45	LD	B, L	;		
1783:	24	INC	H	;		
1784:	CD454D	CALL	4D45H	;	代号	保留字 ★★★ 保留字表 ★★★★★★★★
					C8H	MEM
1787:	C9	RET		;	C9H	INKEY$
1788:	4E	LD	C, (HL)	;		
1789:	4B	LD	C,E	;		
178A:	45	LD	B, L	;		
178B:	59	LD	E, C	;		
178C:	24	INC	H	;		
178D:	D44845	CALL	NC, 4548H	;	CAH	THEN
1790:	4E	LD	C, (HL)	;		
1791:	CE4F	ADC	A, 4FH	;	CBH	NOT
1793:	54	LD	D, H	;		
1794:	D354	OUT	(54H), A	;	CCH	STEP
1796:	45	LD	B, L	;		
1797:	50	LD	D, B	;		
1798:	AB	XOR	E	;	CDH	+
1799:	AD	XOR	L	;	CEH	—
179A:	AA	XOR	D	;	CFH	*
179B:	AF	XOR	A	;	D0H	/
179C:	DBC1	IN	A, (C1H)	;	D1H	↑
179E:	4E	LD	C, (HL)	;	D2H	AND (179DH 起)
179F:	44	LD	B, H	;		
17A0:	CF	RST	08H	;	D3H	OR

地址	目的码	源程序			注　释
17A1:	52	LD	D, D	;	
17A2:	BE	CP	(HL)	; D4H	>
17A3:	BD	CP	L	; D5H	=
17A4:	BC	CP	H	; D6H	<
17A5:	D347	OUT	(47H), A	; D7H	SGN
17A7:	4E	LD	C, (HL)	;	
17A8:	C9	RET		; D8H	INT
17A9:	4E	LD	C, (HL)	;	
17AA:	54	LD	D, H	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★
17AB:	C1	POP	BC	; D9H	ABS
17AC:	42	LD	B, D	;	
17AD:	53	LD	D, E	;	
17AE:	C652	ADD	A, 52H	; DAH	FRE (字符串)
17B0:	45	LD	B, L	;	
17B1:	C9	RET		; DBH	INP
17B2:	4E	LD	C, (HL)	;	
17B3:	50	LD	D, B	;	
17B4:	D0	RET	NC	; DCH	POS
17B5:	4F	LD	C, A	;	
17B6:	53	LD	D, E	;	
17B7:	D351	OUT	(51H), A	; DDH	SQR
17B9:	52	LD	D, D	;	
17BA:	D24E44	JP	NC, 444EH	; DEH	RND
17BD:	CC4F47	CALL	Z, 474FH	; DFH	LOG
17C0:	C5	PUSH	BC	; E0H	EXP
17C1:	58	LD	E, B	;	
17C2:	50	LD	D, B	;	
17C3:	C34F53	JP	534FH	; E1H	COS
17C6:	D349	OUT	(49H), A	; E2H	SIN
17C8:	4E	LD	C, (HL)	;	
17C9:	D4414E	CALL	NC, 4E41H	; E3H	TAN
17CC:	C1	POP	BC	; E4H	ATN
17CD:	54	LD	D, H	;	
17CE:	4E	LD	C, (HL)	;	
17CF:	D0	RET	NC	; E5H	PEEK
17D0:	45	LD	B, L	;	
17D1:	45	LD	B, L	;	
17D2:	4B	LD	C, E	;	
17D3:	C35649	JP	4956H	; E6H	CVI
17D6:	C35653	JP	5356H	; E7H	CVS
17D9:	C35644	JP	4456H	; E8H	CVD

地址	目的码	源程序			注　释
17DC:	C5	PUSH	BC	; E9H	EOF
17DD:	4F	LD	C, A	;	
17DE:	46	LD	B, (HL)	;	
17DF:	CC4F43	CALL	Z, 434FH	; EAH	LOC
17E2:	CC4F46	CALL	Z, 464FH	; EBH	LOF
17E5:	CD4B49	CALL	494BH	; ECH	MKI$
17E8:	24	INC	H	;	
17E9:	CD4B53	CALL	534BH	; EDH	MKS$
17EC:	24	INC	H	;	
17ED:	CD4B44	CALL	444BH	; EEH	MKD$
17F0:	24	INC	H	;	
17F1:	C3494E	JP	4E49H	; EFH	CINT
17F4:	54	LD	D, H	;	
17F5:	C3534E	JP	4E53H	; F0H	CSNG
17F8:	47	LD	B, A	;	
17F9:	C34442	JP	4244H	; F1H	CDBL
17FC:	4C	LD	C, H	;	
17FD:	C649	ADD	A, 49H	; F2H	FIX
17FF:	58	LD	E, B	;	
1800:	CC454E	CALL	Z, 4E45H	; F3H	LEN
1803:	D354	OUT	(54H), A	; F4H	STR$ (表达式)
1805:	52	LD	D, D	;	
1806:	24	INC	H	;	
1807:	D641	SUB	41H	; F5H	VAL (字符串)
1809:	4C	LD	C, H	;	
180A:	C1	POP	BC	; F6H	ASC (字符串)
180B:	53	LD	D, E	;	
180C:	43	LD	B, E	; 代号	保留字 ★★★ 保留字表 ★★★★★★★★
180D:	C34852	JP	5248H	; F7H	CHR$ (表达式)
1810:	24	INC	H	;	
1811:	CC4546	CALL	Z, 4645H	; F8H	LEFT$ (字符串,*n*)
1814:	54	LD	D, H	;	
1815:	24	INC	H	;	
1816:	D24947	JP	NC, 4749H	; F9H	RIGHT$ (字符串, *n*)
1819:	48	LD	C, B	;	
181A:	54	LD	D, H	;	
181B:	24	INC	H	;	
181C:	CD4944	CALL	4449H	; FAH	MID$ (字符串,位置, *n*)
181F:	24	INC	H	;	
1820:	A7	AND	A	; FBH	
1821:	80	ADD	A, B	; 保留字表结束。★★★工作子程序向量地址,每个2字	

地址	目的码	源程序		注　释
				节 ★★★★★★★★★★★★★★★★★★★★★★★★★
1822:	AE	XOR	(HL)	; 1DAEH——END（子程序地址——名称）
1823:	1D	DEC	E	;
1824:	A1	AND	C	; 1CA1H——FOR
1825:	1C	INC	E	;
1826:	3801	JR	C, 1829H	; 0138H——RESET
1828:	35	DEC	(HL)	; 0135H——SET
1829:	01C901	LD	BC, 01C9H	; 182AH:01C9H—CLS
182C:	73	LD	(HL), E	; 4173H——CMD
182D:	41	LD	B, C	;
182E:	D301	OUT	(01H), A	; 01D3H——RANDOM
1830:	B6	OR	(HL)	; 22B6H——NEXT
1831:	22051F	LD	(1F05H), HL	; 1832H:1F05H——DATA
1834:	9A	SBC	A, D	; 219AH——INPUT
1835:	210826	LD	HL, 2608H	; 1836H:2608H——DIM
1838:	EF	RST	28H	; 21EFH——READ
1839:	21211F	LD	HL, 1F21H	; 183AH: 1F21H——LET
183C:	C21EA3	JP	NZ, A31EH	; 1EC2H——GOTO。183EH: 1EA3H——RUN
183F:	1E39	LD	E, 39H	; 1840H:2039H——IF
1841:	2091	JR	NZ, 17D4H	; 1842H: 1D91H——RESTORE
1843:	1D	DEC	E	;
1844:	B1	OR	C	; 1EB1H——GOSUB
1845:	1EDE	LD	E, DEH	; 1846H: 1EDEH——RETURN
1847:	1E07	LD	E, 07H	; 1848H: 1F07H——REM
1849:	1F	RRA		;
184A:	A9	XOR	C	; 1DA9H——STOP
184B:	1D	DEC	E	;
184C:	07	RLCA		; 1F07H——ELSE
184D:	1F	RRA		;
184E:	F7	RST	30H	; 1DF7H——TRON
184F:	1D	DEC	E	;
1850:	F8	RET	M	; 1DF8H——TROFF
1851:	1D	DEC	E	;
1852:	00	NOP		; 1E00H——DEFSTR
1853:	1E03	LD	E, 03H	; 1854H: 1E03H——DEFINT
1855:	1E06	LD	E, 06H	; 1856H: 1E06H——DEFSNG
1857:	1E09	LD	E, 09H	; 1858H: 1E09H——DEFDBL
1859:	1EA3	LD	E, A3H	; 185AH: 41A3H——LINE
185B:	41	LD	B, C	;
185C:	60	LD	H, B	; 2E60H——EDIT
185D:	2EF4	LD	L, F4H	; 185EH: 1FF4H——ERROR

地址	目的码	源程序		注释
185F:	1F	RRA		;
1860:	AF	XOR	A	; 1FAFH——RESUME
1861:	1F	RRA		;
1862:	FB	EI		; 2AFBH——OUT
1863:	2A6C1F	LD	HL, (1F6CH)	; 1864H: 1F6CH——ON
1866:	79	LD	A, C	; 4179H——OPEN
1867:	41	LD	B, C	;
1868:	7C	LD	A, H	; 417CH——FIELD
1869:	41	LD	B, C	;
186A:	7F	LD	A, A	; 417FH——GET
186B:	41	LD	B, C	;
186C:	82	ADD	A, D	; 4182H——PUT
186D:	41	LD	B, C	;
186E:	85	ADD	A, L	; 4185H——CLOSE
186F:	41	LD	B, C	;
1870:	88	ADC	A, B	; 4188H——LOAD
1871:	41	LD	B, C	;
1872:	8B	ADC	A, E	; 418BH——MERGE
1873:	41	LD	B, C	;
1874:	8B	ADC	A, (HL)	; 418EH——NAME
1875:	41	LD	B, C	;
1876:	91	SUB	A, C	; 4191H——KILL
1877:	41	LD	B, C	;
1878:	97	SUB	A, A	; 4197H——LSET
1879:	41	LD	B, C	;
187A:	9A	SBC	A, D	; 419AH——RSET
187B:	41	LD	B, C	;
187C:	A0	AND	B	; 41A0H——SAVE
187D:	41	LD	B, C	;
187E:	B2	OR	D	; 02B2H——SYSTEM
187F:	02	LD	(BC), A	;
1880:	67	LD	H, A	; 2067H——LPRINT
1881:	205B	JR	NZ, 18DEH	; 1882H: 415BH——DEF
1883:	41	LD	B, C	;
1884:	B1	OR	C	; 2CB1H——POKE
1885:	2C	INC	L	;
1886:	6F	LD	L, A	; 206FH——PRINT
1887:	20E4	JR	NZ, 186DH	; 1888H: 1DE4H——CONT
1889:	1D	DEC	E	;
188A:	2E2B	LD	L, 2BH	; 2B2EH——LIST
188C:	29	ADD	HL, HL	; 2B29H——LLIST

地址	目的码	源程序		注　释
188D:	2B	DEC	HL	;
188E:	C62B	ADD	A, 2BH	; 2BC6H——DELETE
1890:	08	EX	AF, AF'	; 2008H——AUTO
1891:	207A	JR	NZ, 190DH	; 1892H: 1E7AH——CLEAR
1893:	1E1F	LD	E, 1FH	; 1894H: 2C1FH——CLOAD
1895:	2C	INC	L	;
1896:	F5	PUSH	AF	; 2BF5H——CSAVE
1897:	2B	DEC	HL	;
1898:	49	LD	C, C	; 1B49H——NEW
1899:	1B	DEC	DE	;
189A:	79	LD	A, C	; + ★★★ 算符优先值 ★★★★★★★★★★★★★
189B:	79	LD	A, C	; —
189C:	7C	LD	A, H	; *
189D:	7C	LD	A, H	; /
189E:	7F	LD	A, A	; ↑
189F:	50	LD	D, B	; AND
18A0:	46	LD	B, (HL)	; OR
18A1:	DB0A	IN	A, (0AH)	; ★★★ 由算术子程序用来进行数据转换和算术运算★ 0ADBH——转换成双精度。
18A3:	00	NOP		; 以下两个单元没有用。
18A4:	00	NOP		;
18A5:	7F	LD	A, A	; 0A7FH——转换成整数。
18A6:	0A	LD	A, (BC)	;
18A7:	F40AB1	CALL	P, B10AH	; 0AF4H——测试数据类型。若不是字符串，则为 TM 错误。
18AA:	0A	LD	A, (BC)	; 18A9H: 0AB1H——转换成单精度。
18AB:	77	LD	(HL), A	; 0C77H——双精度加法子程序。
18AC:	0C	INC	C	;
18AD:	70	LD	(HL), B	; 0C70H——双精度减法子程序。
18AE:	0C	INC	C	;
18AF:	A1	AND	C	; 0DA1H——双精度乘法子程序。
18B0:	0D	DEC	C	;
18B1:	E5	PUSH	HL	; 0DE5H——双精度除法子程序。
18B2:	0D	DEC	C	;
18B3:	78	LD	A, B	; 0A78H——双精度比较子程序。
18B4:	0A	LD	A, (BC)	;
18B5:	1607	LD	D, 07H	; 0716H——单精度加法子程序。
18B7:	13	INC	DE	; 0713H——单精度减法子程序。
18B8:	07	RLCA		;
18B9:	47	LD	B, A	; 0847H——单精度乘法子程序。
18BA:	08	EX	AF, AF'	;

地址	目的码	源程序		注释
18BB:	A2	AND	D	; 08A2H——单精度除法子程序。
18BC:	08	EX	AF, AF'	;
18BD:	0C	INC	C	; 0A0CH——单精度比较子程序。
18BE:	0A	LD	A, (BC)	;
18BF:	D20BC7	JP	NC, C70BH	; 0BD2H——整数加法子程序。
18C2:	0B	DEC	BC	; 18C1H: 0BC7H——整数减法子程序。
18C3:	F20B90	JP	P, 900BH	; 0BF2H——整数乘法子程序。
18C6:	24	INC	H	; 18C5H: 2490H——整数除法子程序。
18C7:	39	ADD	HL, SP	; 0A39H——整数比较子程序。
18C8:	0A	LD	A, (BC)	;
18C9:	4E	LD	C, (HL)	; 0——NF (有 NEXT 没有 FOR)。★★★ 错误代码★
18CA:	46	LD	B, (HL)	;
18CB:	53	LD	D, E	; 2——SN (句法错误)。
18CC:	4E	LD	C, (HL)	;
18CD:	52	LD	D, D	; 4——RG (有 RETURN 没有 GOSUB)。
18CE:	47	LD	B, A	;
18CF:	4F	LD	C, A	; 6——OD (DATA不够)。
18D0:	44	LD	B, H	;
18D1:	46	LD	B, (HL)	; 8——FC (非法的函数调用)。
18D2:	43	LD	B, E	;
18D3:	4F	LD	C, A	; 10——OV (溢出)。
18D4:	56	LD	D, (HL)	;
18D5:	4F	LD	C, A	; 12——OM (内存不够)。
18D6:	4D	LD	C, L	;
18D7:	55	LD	D, L	; 14——UL (没有定义过的行号)。
18D8:	4C	LD	C, H	;
18D9:	42	LD	B, D	; 16——BS (下标超出范围)。
18DA:	53	LD	D, E	;
18DB:	44	LD	B, H	; 18——DD (重复定义数组)。
18DC:	44	LD	B, H	;
18DD:	2F	CPL		; 20——/0 (被零除)。
18DE:	3049	JR	NC, 1929H	; 22——ID (非法的直接操作)。
18E0:	44	LD	B, H	;
18E1:	54	LD	D, H	; 24——TM (类型不符合)。
18E2:	4D	LD	C, L	;
18E3:	4F	LD	C, A	; 26——OS (字符串空间不够)。
18E4:	53	LD	D, E	;
18E5:	4C	LD	C, H	; 28——LS (字符串太长)。
18E6:	53	LD	D, E	;
18E7:	53	LD	D, E	; 30——ST (字符串公式太复杂)。
18E8:	54	LD	D, H	;

地址	目的码	源程序		注释
18E9:	43	LD	B, E	; 32——CN (不能继续)。
18EA:	4E	LD	C, (HL)	;
18EB:	4E	LD	C, (HL)	; 34——NR (执行了 ON ERROR 但没有 RESUME)。
18EC:	52	LD	D, D	;
18ED:	52	LD	D, D	; 36——RW (在执行 ON ERROR 以前遇到了 RESUME)。
18EE:	57	LD	D, A	;
18EF:	55	LD	D, L	; 38——UE (不能显示出的错误)。
18F0:	45	LD	B, L	;
18F1:	4D	LD	C, L	; 40——MO (遗漏运算量)。
18F2:	4F	LD	C, A	;
18F3:	46	LD	B, (HL)	; 42——FD (坏的文件数据)。
18F4:	44	LD	B, H	;
18F5:	4C	LD	C, H	; 44——L3 (磁盘 BASIC 命令)。
18F6:	33	INC	SP	;
18F7:	D600	SUB	00H	; 减 LSB。★★★除法支援子程序 ★★★★★★★★★

说明 在无磁盘 IPL 过程中，18F7H 到 191DH 的代码被送入 4080H 到 40A6H 单元。这一段代码含有用于单精度除法的除法支援子程序和通讯区单元 408EH 到 40A4H 的初始值。

18F9:	6F	LD	L, A	; 并将值重新存入 L。
18FA:	7C	LD	A, H	; 取中间字节。
18FB:	DE00	SBC	A, 00H	; 减中间字节。
18FD:	67	LD	H, A	; 把差送入 H。
18FE:	78	LD	A, B	; 取 MSB。
18FF:	DE00	SBC	A, 00H	; 减 MSB,
1901:	47	LD	B, A	; 并存回原处。
1902:	3E00	LD	A, 00H	; 清除 A。
1904:	C9	RET		; 返回到调用它的程序。
1905:	4A	LD	C, D	; 408EH: 用户子程序的地址。
1906:	1E40	LD	E, 40H	;
1908:	E64D	AND	4DH	; 4090H: RND 使用的 3 字节表,以跟踪原先的 RND 值。
190A:	DB00	IN	A, (00H)	; 4093H: 用于 INP (××)。
190C:	C9	RET		; 4095H: RET
190D:	D300	OUT	(00H), A	; 4096H: 用于 OUT P,××。
190F:	C9	RET		; 4098H: RET
1910:	00	NOP		; 4099H: 00
1911:	00	NOP		; 409AH: 00
1912:	00	NOP		; 409BH: 00
1913:	00	NOP		; 409CH: 00
1914:	40	LD	B, B	; 409DH: 40H
1915:	3000	JR	NC, 1917H	;
1917:	4C	LD	C, H	; 40A0H: 用于无磁盘 IPL 的初始堆栈地址 (434CH)。
1918:	43	LD	B, E	;

地址	目的码	源程序		注释
1919:	FEFF	CP	FFH	; 40A2H: 初始 BASIC 行号 (FFFEH)。
191B:	E9	JP	(HL)	; 40A4H: PST 的初始地址 (42E9H)。
191C:	42	LD	B, D	;
191D:	2045	JR	NZ, 1964H	; 空格, E ★★★ ERROR 信息 ★★★★★★★★★★
191F:	72	LD	(HL), D	; R
1920:	72	LD	(HL), D	; R
1921:	6F	LD	L,A	; O
1922:	72	LD	(HL), D	; R
1923:	00	NOP		; 结束符。
1924:	2069	JR	NZ, 198FH	; 空格, I ★★★ IN 信息 ★★★★★★★★★★★★
1926:	6E	LD	L, (HL)	; N
1927:	2000	JR	NZ, 1929H	; 空格, 00——结束符。
1929:	52	LD	D, D	; R ★★★ READY 信息 ★★★★★★★★★★★★★
192A:	45	LD	B, L	; E
192B:	41	LD	B, C	; A
192C:	44	LD	B, H	; D
192D:	59	LD	E, C	; Y
192E:	0D	DEC	C	; 回车。
192F:	00	NOP		; 结束符。
1930:	42	LD	B, D	; B ★★★ BREAK 信息 ★★★★★★★★★★★★★
1931:	72	LD	(HL), D	; R
1932:	65	LD	H, L	; E
1933:	61	LD	H, C	; A
1934:	6B	LD	L, E	; K
1935:	00	NOP		; 信息结束符。
1936:	210400	LD	HL, 0004H	; HL = 4 ★★★ 寻找与调用它的程序指定的索引相匹配的索引 ★★★★★★★★★★★★★★★★★★★★

说明 调用时 DE=NEXT 索引的地址。往后扫描堆栈寻找推入的 FOR。若找到 FOR，则得到索引的地址，并与调用它的程序的 DE 进行比较。若相等，就返回，此时 A=0，HL = 变量的地址。若不相等，则继续扫描，直到找不到 FOR，此时返回时，A<>0。

1939:	39	ADD	HL, SP	; 于是，我们可以将当前堆栈指针退回 4 个字节。
193A:	7E	LD	A, (HL)	; A=(当前堆栈指针减 4)。
193B:	23	INC	HL	; 在寻找 FOR 代号时，再退回 1 个字节。
193C:	FE81	CP	81H	; (当前堆栈指针－ 1)=FOR 的代号吗?
193E:	C0	RET	NZ	; 如果不是 FOR，则返回，此时 A 不为 0。
193F:	4E	LD	C, (HL)	; C=索引变量的地址的 LSB。
1940:	23	INC	HL	; 当前堆栈指针再退回 1 个字节。
1941:	46	LD	B, (HL)	; B=索引变量地址的 MSB。
1942:	23	INC	HL	; HL=堆栈中 FOR 索引的地址。
1943:	E5	PUSH	HL	; 将 FOR 索引指针的地址存入堆栈。

地址	目的码	源程序		注释
1944:	69	LD	L, C	; L=索引地址的 LSB。
1945:	60	LD	H, B	; H=索引地址的 MSB。(见 1936H 注释)
1946:	7A	LD	A, D	; 测试用户指定的变量地址。
1947:	B3	OR	E	; 置状态标志。
1948:	EB	EX	DE, HL	; DE=由堆栈得到的索引地址。
1949:	2802	JR	Z, 194DH	; 如果用户指定的地址为 0,则转移。
194B:	EB	EX	DE, HL	; HL=由堆栈得到的索引地址。
194C:	DF	RST	18H	; 把调用它的程序的 DE 与由堆栈得到的索引地址进行比较。
194D:	010E00	LD	BC, 000EH	; 到下一个 FOR 代号的后退数量。
1950:	E1	POP	HL	; HL=增量标志的符号的堆栈地址。
1951:	C8	RET	Z	; 如果 FOR 索引=NEXT 索引,返回。
1952:	09	ADD	HL, BC	; 否则,退回到下一个可能的 FOR。
1953:	18E5	JR	193AH	; 继续寻找。
1955:	CD6C19	CALL	196CH	; 确认字符串区域中有空间。★★★在入口时 DE=上限★★★★★★★★★★★★★★★★★★★★★★★★

说明 这个子程序把一个变量(通常为字符串)传送到调用它的程序所指定的另一个区域。

在入口时: BC=被传送的表的终址;
DE=被传送的表的首址;
HL=表所送入区域的终址。

地址	目的码	源程序		注释
1958:	C5	PUSH	BC	; 源地址(表的终址)存入堆栈。
1959:	E3	EX	(SP), HL	; 源地址(表的终址)送入 HL。
195A:	C1	POP	BC	; BC=目标地址(终址)。
195B:	DF	RST	18H	; 测试传送的结束。
195C:	7E	LD	A, (HL)	; 从源表取一个字节。
195D:	02	LD	(BC), A	; 存入目标表。
195E:	C8	RET	Z	; 如果表被传送完,则返回。
195F:	0B	DEC	BC	; 源地址减 1。
1960:	2B	DEC	HL	; 目标地址减 1。
1961:	18F8	JR	195BH	; 循环,直到表被送完。
1963:	E5	FUSH	HL	; 保存代码串地址。★★★ 计算 HL 与内存终端 FFC6H 间的空间数量 ★★★★★★★★★★★★★★★★★★
1964:	2AFD40	LD	HL, (40FDH)	; 空闲内存指针的首址。
1967:	0600	LD	B, 00H	; B=00, C=需要的双字节数。
1969:	09	ADD	HL, BC	; 将需要的字节数乘 2 加空闲区的首址。
196A:	09	ADD	HL, BC	; HL=空闲区域的终址。
196B:	3EE5	LD	A, E5H	; 196CH: PUSH HL ; 保存新的空闲区指针(首址)。
196D:	3EC6	LD	A, C6H	; 计算堆栈首址 FFC6H (即 65478) 与新的
196F:	95	SUB	A, L	; 空闲内存指针首址之间的内存数量,
1970:	6F	LD	L, A	; 用 FFC6H 减空闲内存新首址来计算。
1971:	3EFF	LD	A, FFH	; 如果空闲内存超过堆栈首址的范围溢出,

地址	目的码	源程序		注释
1973:	9C	SBC	A, H	; 那末就是空间不够。
1974:	3804	JR	C, 197AH	; 如果 C 标志置位——空闲区表超过 65478 (FFC6H), 则为 OM 错误。
1976:	67	LD	H, A	; 准备测定空闲区是否溢出堆栈区域。
1977:	39	ADD	HL, SP	;
1978:	E1	POP	HL	;
1979:	D8	RET	C	; 如果 C 标志置位,表示没有溢出,返回。
197A:	1E0C	LD	E, 0CH	; OM 错误代码。
197C:	1824	JR	19A2H	; 输出 OM 错误信息。
197E:	2AA240	LD	HL, (40A2H)	; HL=当前行号。★★★★★★★★★★★★★★★★
1981:	7C	LD	A, H	; 把 MSB 与 LSB 合并。
1982:	A5	AND	L	;
1983:	3C	INC	A	; 如果当前行=FFFFH, 那末我们还没有开始执行 BASIC 程序。
1984:	2808	JR	Z, 198EH	; 如果没有执行 BASIC 程序,则转移。仍在输入阶段。
1986:	3AF240	LD	A, (40F2H)	; 取错误忽略标志。
1989:	B7	OR	A	; 置状态标志。
198A:	1E22	LD	E, 22H	; NO RESUME 错误代码。
198C:	2014	JR	NZ, 19A2H	; 若在输入阶段中发生没有 RESUME 地址的错误, 则输出 NR 错误信息。
198E:	C3C11D	JP	1DC1H	; 再进入 BASIC 的 'READY' 子程序——装入当前数据行号。
1991:	2ADA40	LD	HL, (40DAH)	; 取入最后的 DATA 语句的行号。
1994:	22A240	LD	(40A2H), HL	; 把它作为当前行指针存起来。
1997:	1E02	LD	E, 02H	; SN 错误代码。
1999:	011E14	LD	BC, 141EH	; 199AH: LD E, 14H ; /0 错误代码。
199C:	011E00	LD	BC, 001EH	; 199DH: LD E, 0 ; NF 错误代码。
199F:	011E24	LD	BC, 241EH	; 19A0H: LD E, 24H ; RW 错误代码。
19A2:	2AA240	LD	HL, (40A2H)	; HL=有错误的行的地址。★★★★★★★★★★★★
19A5:	22EA40	LD	(40EAH), HL	; 将错误行号存储起来。
19A8:	22EC40	LD	(40ECH), HL	; 再存一个。
19AB:	01B419	LD	BC, 19B4H	; BC=重新初始化后的继续行号。
19AE:	2AE840	LD	HL, (40E8H)	; HL=起始语句的堆栈指针。
19B1:	C39A1B	JP	1B9AH	; 去重新初始化系统变量。返回到 19B4H。
19B4:	C1	POP	BC	; BC=0000
19B5:	7B	LD	A, E	; A=错误号。
19B6:	4B	LD	C, E	; C=错误号。
19B7:	329A40	LD	(409AH), A	; 存储错误号。
19BA:	2AE640	LD	HL, (40E6H)	; HL=当前行中执行过的最后字节的地址。
19BD:	22EE40	LD	(40EEH), HL	; 存储执行过的最后字节的地址。
19C0:	EB	EX	DE, HL	; 保存 HL

地址	目的码	源程序		注释
19C1:	2AEA40	LD	HL, (40EAH)	; HL=执行过的最后行的行号。
19C4:	7C	LD	A, H	; 把执行过的最后行号的 LSB
19C5:	A5	AND	L	; 与 MSB 合并,
19C6:	3C	INC	A	; 然后测试是否行号=FFFFH。
19C7:	2807	JR	Z, 19D0H	; 如果行号=FFFFH, 则仍在输入阶段中。
19C9:	22F540	LD	(40F5H), HL	; 存储错误地址。
19CC:	EB	EX	DE, HL	; 恢复执行过的最后字节。
19CD:	22F740	LD	(40F7H), HL	; 存储执行过的最后字节。
19D0:	2AF040	LD	HL, (40F0H)	; 取 ON ERROR 地址。
19D3:	7C	LD	A, H	; 将 LSB 与 MSB 合并,
19D4:	B5	OR	L	; 于是可以测试它是否为 0 。
19D5:	EB	EX	DE, HL	; DE=ON ERROR 地址。
19D6:	21F240	LD	HL, 40F2H	; 在 ON ERROR 处理过程中标志字的地址。
19D9:	2808	JR	Z, 19E3H	; 若没有 ON ERROR 地址,则转移。
19DB:	A6	AND	(HL)	; 测试程序中是否有 RESUME 处理。
19DC:	2005	JR	NZ, 19E3H	; 是,那末不能嵌套 RESUME。
19DE:	35	DEC	(HL)	; 标志一个错误,因此 RESUME 将进行工作。
19DF:	EB	EX	DE, HL	; HL=要转入的语句的地址。
19E0:	C3361D	JP	1D36H	; 去执行驱动程序。
19E3:	AF	XOR	A	; A 清零。★★★★★★★★★★★★★★★★★★★★
19E4:	77	LD	(HL), A	; 清除错误忽略标志。
19E5:	59	LD	E, C	; 错误号送入 E。
19E6:	CDF920	CALL	20F9H	; 显示器定位在下一行上。
19E9:	21C918	LD	HL, 18C9H	; HL=错误代码表地址。
19EC:	CDA641	CALL	41A6H	; DOS 出口(装入和执行 BASIC 错误子程序)。
19EF:	57	LD	D, A	; D 清零。
19F0:	3E3F	LD	A, 3FH	; A=ASCII 的'?'。
19F2:	CD2A03	CALL	032AH	; 显示'?'。
19F5:	19	ADD	HL, DE	; HL=地址(某错误代码的地址)。
19F6:	7E	LD	A, (HL)	; 取错误代码的一个字符。
19F7:	CD2A03	CALL	032AH	; 显示该字符。
19FA:	D7	RST	10H	; 取错误代码的下一个字符。
19FB:	CD2A03	CALL	032AH	; 显示该字符。
19FE:	211D19	LD	HL, 191DH	; 'ERROR' 信息的地址。
1A01:	E5	PUSH	HL	; 保存 'ERROR' 信息的地址。
1A02:	2AEA40	LD	HL, (40EAH)	; HL=产生错误的语句的行号。
1A05:	E3	EX	(SP), HL	; 行号送入堆栈, HL='ERROR' 信息的地址。
1A06:	CDA728	CALL	28A7H	; 输出信息,其地址在 HL 中。
1A09:	E1	POP	HL	; HL=STOP、END 或有错误的行的二进制行号。
1A0A:	11FEFF	LD	DE, FFFEH	; DE=65534
1A0D:	DF	RST	18H	; 当前行号=65534 吗?

地址	目的码	源程序		注　释
1A0E:	CA7406	JP	Z, 0674H	; 是,则 IPL 系统。
1A11:	7C	LD	A, H	; 否,则测试行号是否为 0 。
1A12:	A5	AND	L	; 将当前行号的 LSB 与 MSB 合并。
1A13:	3C	INC	A	;
1A14:	C4A70F	CALL	NZ, 0FA7H	; 若不为 0 ,则显示出当前行号。
1A17:	3EC1	LD	A, C1H	; 1A18H: POP BC
1A19:	CD8B03	CALL	038BH	; 置输出设备为显示器。★★★ 刷新当前行缓冲器 ★★
1A1C:	CDAC41	CALL	41ACH	; DOS 出口 (JP 5FFCH)。
1A1F:	CDF801	CALL	01F8H	; 关盒式录音机。
1A22:	CDF920	CALL	20F9H	; 跳到显示器的下一行。
1A25:	212919	LD	HL, 1929H	; 'READY' 信息的地址。
1A28:	CDA728	CALL	28A7H	; 输出 'READY' 信息。
1A2B:	3A9A40	LD	A, (409AH)	; 取错误号。
1A2E:	D602	SUB	02H	; 测试是否句法错误。
1A30:	CC532E	CALL	Z, 2E53H	; 若是句法错误,则进入 EDIT 子程序。
1A33:	21FFFF	LD	HL, FFFFH	; HL=当前行号。
1A36:	22A240	LD	(40A2H), HL	; 置当前行号为－1 ,表示执行尚未开始。
1A39:	3AE140	LD	A, (40E1H)	; 自动输入标志字段——若是自动,则为非零,若不是自动,则为 00H。
1A3C:	B7	OR	A	; 置状态标志。
1A3D:	2837	JR	Z, 1A76H	; 若无自动增量,则转移并显示出 '>'提示符。
1A3F:	2AE240	LD	HL, (40E2H)	; 否则将当前行号取入 HL。
1A42:	E5	PUSH	HL	; 当前行号存入堆栈。
1A43:	CDAF0F	CALL	0FAFH	; 输出一个行号。
1A46:	D1	POP	DE	; 将当前行号装入 DE, 供检索子程序使用,
1A47:	D5	PUSH	DE	; 并将它留在堆栈中。
1A48:	CD2C1B	CALL	1B2CH	; 检索相同的行号。
1A4B:	3E2A	LD	A, 2AH	; '*' 号的代码(表示相同的行号)。
1A4D:	3802	JR	C, 1A51H	; 如果找到相同的行号,则转移去显示'*'号。
1A4F:	3E20	LD	A, 20H	; 否则显示出一个空格。
1A51:	CD2A03	CALL	032AH	; 显示一个空格或'*'号。
1A54:	CD6103	CALL	0361H	; 接受输入并送到缓冲区内。
1A57:	D1	POP	DE	; DE=当前行号。
1A58:	3006	JR	NC, 1A60H	; 若没有按 BREAK 键,则转移。
1A5A:	AF	XOR	A	; 否则清除 AUTO 增量标志。
1A5B:	32E140	LD	(40E1H), A	; 关闭 AUTO 增量。
1A5E:	18B9	JR	1A19H	; 转移到 'READY' 子程序。
1A60:	2AE440	LD	HL, (40E4H)	; 取增量值。 ★★★★★★★★★★★★★★★★★★★
1A63:	19	ADD	HL, DE	; 加当前行号并测试溢出否。
1A64:	38F4	JR	C, 1A5AH	; 如果行号超过 2^{15}, 则转移,清除 AUTO 增量。
1A66:	D5	PUSH	DH	; 把未增量的行号存入堆栈。

地址	目的码	源程序		注　释
1A67:	11F9FF	LD	DE, FFF9H	; DE=65529
1A6A:	DF	RST	18H	; 将增量后的行号与65529比较。
1A6B:	D1	POP	DE	; DE=未增量的行号。
1A6C:	30EC	JR	NC, 1A5AH	; 如果增量后的行号 ≥65529，则转移。
1A6E:	22E240	LD	(40E2H), HL	; 把增量后的行号作为当前行号存储起来。
1A71:	F6FF	OR	FFH	; 置 A=FFH (即255)。
1A73:	C3EB2F	JP	2FEBH	; 使用 EDIT 程序将缓冲区地址装入 HL，然后转移到1A98H。
1A76:	3E3E	LD	A, 3EH	; A='>'(提示符)。★★★ 不用 AUTO 增量输入行号 ★★★★★★★★★★★★★★★★★★★★★★★★
1A78:	CD2A03	CALL	032AH	; 显示出'>'号。
1A7B:	CD6103	CALL	0361H	; 接受输入，返回时 HL=缓冲区地址。
1A7E:	DA331A	JP	C, 1A33H	; 如果按 BREAK 键，则转移，去取下一行。
1A81:	D7	RST	10H	; 从缓冲区取一个字符，跳过空格和控制代码。
1A82:	3C	INC	A	; 置状态标志，但保存 C 标志，
1A83:	3D	DEC	A	; 于是，我们可以测试语句的结束。
1A84:	CA331A	JP	Z, 1A33H	; 若语句结束，则转移。
1A87:	F5	PUSH	AF	; 保存状态 (C 标志)。
1A88:	CD5A1E	CALL	1E5AH	; 把二进制表示的行号取入 DE。
1A8B:	2B	DEC	HL	; 后退输入缓冲区，越过所有的跟在行号后边的空格。
1A8C:	7E	LD	A, (HL)	; 取下个字符。
1A8D:	FE20	CP	20H	; 检查是不是空格。
1A8F:	28FA	JR	Z, 1A8BH	; 循环，直到找到行号的最后一位数字。
1A91:	23	INC	HL	; HL=行号后边第一个字符的地址。
1A92:	7E	LD	A, (HL)	; 取行号后的第一个字符。
1A93:	FE20	CP	20H	; 若它是一个空格，
1A95:	CCC909	CALL	Z, 09C9H	; 那末缓冲区地址加1，指向下个字符。
1A98:	D5	PUSH	DE	; 保存二进制行号。
1A99:	CDC01B	CALL	1BC0H	; 将输入编码成代号，BC=被编码语句的长度。
1A9C:	D1	POP	DE	; DE=二进制行号。
1A9D:	F1	POP	AF	; 取由 1A81H 得到的 C 标志。
1A9E:	22E640	LD	(40E6H), HL	; 编码语句的指针。
1AA1:	CDB241	CALL	41B2H	; DOS 出口 (JP 6033H)。
1AA4:	D25A1D	JP	NC, 1D5AH	; 如果没有行号，则转移。必须是直接语句或系统命令。
1AA7:	D5	PUSH	DH	; 保存二进制行号。
1AA8:	C5	PUSH	BC	; 保存代码串的长度。
1AA9:	AF	XOR	A	; A 清零，
1AAA:	32DD40	LD	(40DDH), A	; 并清除进入输入阶段的标志。
1AAD:	D7	RST	10H	; 扫描第一个代号。
1AAE:	B7	OR	A	; 置状态标志，
1AAF:	F5	PUSH	AF	; 并将它们保存起来。

地址	目的码	源程序		注释
1AB0:	EB	EX	DE, HL	; HL=行号的二进制值。
1AB1:	22EC40	LD	(40ECH), HL	; 把行号存储在通讯区中。
1AB4:	EB	EX	DE, HL	; DE=行号,供检索子程序使用。
1AB5:	CD2C1B	CALL	1B2CH	; 检索相同的行号。
1AB8:	C5	PUSH	BC	; 在检索后,如果该行号已存在,则 BC=行号在缓冲区中的地址。
1AB9:	DCE42B	CALL	C, 2BE4H	; 如果没有找到相同的行,就将最靠近的行,在内存中往上移,
1ABC:	D1	POP	DE	; 以便给新的行留出空间。DE=缓冲区中行的地址。
1ABD:	F1	POP	AF	; 恢复在 1AADH 扫描代号所得到的状态。
1ABE:	D5	PUSH	DE	; 保存缓冲区中行的地址。
1ABF:	2827	JR	Z, 1AE8H	; 如果找到相同的行,则转移,否则添加新行。
1AC1:	D1	POP	DE	; DE=最后行或大于新行的行的地址。
1AC2:	2AF940	LD	HL, (40F9H)	; HL=程序行终址的指针。
1AC5:	E3	EX	(SP), HL	; HL=代码串的长度,堆栈=要移动的行的地址。
1AC6:	C1	POP	BC	; BC=新行的长度。
1AC7:	09	ADD	HL, BC	; HL=程序行新终址的指针。
1AC8:	E5	PUSH	HL	; 保存程序结束的地址。
1AC9:	CD5519	CALL	1955H	; 证明有足够的空间供新行使用。测试堆栈区中 PST 溢出否。
1ACC:	E1	POP	HL	; HL=PST 的终址。
1ACD:	22F940	LE	(40F9H), HL	; PST 的新终址。
1AD0:	EB	EX	DE, HL	; HL=要往上移的行的地址。
1AD1:	74	LD	(HL), H	; 把要移动行的地址的 MSB 作为行的首字节存储起来。
1AD2:	D1	POP	DE	; DE=二进制形式的新行号。
1AD3:	E5	PUSH	HL	; 如果程序行要往上移,把地址保存起来。
1AD4:	23	INC	HL	; HL+1, 指向行号项的 LSB。
1AC5:	23	INC	HL	; 指向行号项的 MSB。
1AD6:	73	LD	(HL), E	; DE=新行行号的二进制值。存储 LSB。
1AD7:	23	INC	HL	; HL+1, 指向 MSB。
1AD8:	72	LD	(HL), D	; 把新行号的 MSB 存在老行号的位置。
1AD9:	23	INC	HL	; HL=语句指针(过去的行号)。
1ADA:	EB	EX	DE, HL	; DE=行号后边的第一个数据字节的地址。
1ADB:	2AA740	LD	HL, (40A7H)	; HL=输入区域指针(取出地址)。
1ADE:	EB	EX	DE, HL	; DE=输入区域指针, HL=程序区中第一个数据位置(存储地址)。
1ADF:	1B	DEC	DE	; DE=输入区域指针－1。
1AE0:	1B	DEC	DE	; DE=输入区域指针－2。
1AE1:	1A	LD	A, (DE)	; 从输入缓冲区取入程序的一个字节。
1AE2:	77	LD	(HL), A	; 将它送入程序存储区 (PST)。
1AE3:	23	INC	HL	; 存储地址加1。

地址	目的码	源程序		注　释
1AE4:	13	INC	DE	; 取出地址加 1 。
1AE5:	B7	OR	A	; 测试代码串的结束。
1AE6:	20F9	JR	NZ, 1AE1H	; 如果不是要移动的语句的结束,则转移。
1AE8:	D1	POP	DE	; DE=程序表中行的地址。
1AE9:	CDFC1A	CALL	1AFCH	; 修改新行后边的所有行的行指针。
1AEC:	CDB541	CALL	41B5H	; DOS 出口 (JP 5BD7H)。
1AEF:	CD5D1B	CALL	1B5DH	; 修改 40FBH, 40FBH 行指针=40F9H。
1AF2:	CDB841	CALL	41B8H	; DOS 出口 (JP 5B8CH)。
1AF5:	C3331A	JP	1A33H	; 循环,重复输入过程。
1AF8:	2AA440	LD	HL, (40A4H)	; HL=PST 的首址(从磁盘 BASIC 进入的)。
1AFB:	EB	EX	DE, HL	; 把 PST 地址送入 DE。
1AFC:	62	LD	H, D	; ★★★ 修改新行后边的所有行的行指针★★★★★★ DE=程序语句表的地址。HL=当前行指针。
1AFD:	6B	LD	L, E	; 每行的前两字节含有下一行的地址。 0000 两个字节表示程序结束。
1AFE:	7E	LD	A, (HL)	; 取入当前行的首字节,
1AFF:	23	INC	HL	; 与第二字节合并,
1B00:	B6	OR	(HL)	; 寻找程序的结束字节。
1B01:	C8	RET	Z	; 如果结束,则返回。
1B02:	23	INC	HL	; HL=语句开始的指针(跳过下个语句指针和行号)。
1B03:	23	INC	HL	; 跳过当前行的第三字节,
1B04:	23	INC	HL	; 和第四字节,它们含有该行的行号。
1B05:	AF	XOR	A	; A=0, 清除状态标志。
1B06:	BE	CP	(HL)	; 扫描当前行的结束,它用 00H 结束。
1B07:	23	INC	HL	; 当找到结束时, HL+1 将是下一行的地址。
1B08:	20FC	JR	NZ, 1B06H	; 循环,直到找到语句的结束为止。
1B0A:	EB	EX	DE, HL	; DE=语句终址+ 1 (指向下个语句的指针), HL=当前行指针。
1B0B:	73	LD	(HL), E	; 把下一行地址送入当前行的前两个字节,
1B0C:	23	INC	HL	; 存储下一行地址的 LSB。
1B0D:	72	LD	(HL), D	; 存储下一行地址的 MSB。
1B0E:	18EC	JR	1AFCH	; 循环,直到找到程序的结束为止。
1B10:	110000	LD	DE, 0000H	; 不指定任何行号时,预置起始行为 0 。 ★★★★★★

说明 被 LIST 和 DELETE 调用。将起始行号和结束行号 (X—Y)转换成二进制,并将结束行号存入堆栈。然后进入下边的程序,去寻找起始行的程序表地址。把起始行的地址留在 BC 中,结束行的地址在堆栈中。

1B13:	D5	PUSH	DE	; 存入堆栈。
1B14:	2809	JR	Z, 1B1FH	; 如果没有给定行号,则转移。
1B16:	D1	POP	DE	; 清除临时的初始值。
1B17:	CD4F1E	CALL	1E4FH	; 取给定的起始行号,转换成二进制放入 DE。
1B1A:	D5	PUSH	DE	; 保存起始行号。

地址	目的码	源程序		注释
1B1B:	280B	JR	Z,1B28H	; 若没有指定结束行,则转移。
1B1D:	CF	RST	08H	; 测试行号后的破折号。
1B1E:	CE11	ADC	A, 11H	; 1B1EH: CE 是破折号代码。
1B20:	FAFFC4	JP	M, C4FFH	; 1B1FH: LD DE, FFFAH ; 缺项结束行号。
1B23:	4F	LD	C, A	; 1B22H: CALL NZ, 1E4FH; 将结束行号取入 DE。
1B24:	1EC2	LD	E, C2H	; 1B25H: JP NZ, 1997H;若没有结束符,则为 SN 错误。
1B26:	97	SUB	A, A	;
1B27:	19	ADD	HL, DE	;
1B28:	EB	EX	DE, HL	; HL=结束行号。
1B29:	D1	POP	DE	; DE=起始行号。
1B2A:	E3	EX	(SP), HL	; 结束行号存入堆栈,返回地址放入 HL。
1B2B:	E5	PUSH	HL	; 返回地址存入堆栈,于是,我们可以在下面返回。
1B2C:	2AA440	LD	HL, (40A4H)	; HL=PST 的首址。★★★检索相同行号的子程序★★

说明 出口条件: 没有找到行,但遇到了 PST 的结束: NC/Z/HL=BC。
找到了行: DE=HL/C/Z, BC=PST 中行的地址,
HL=下一行的地址。
没有找到行,发现行号已大于要找的行号: DE>HL/NC/NZ,
BC=当前行的地址, HL=下一行的地址。

1B2F:	44	LD	B, H	; DE=要寻找的行号。
1B30:	4D	LD	C, L	; BC=当前行在 PST 中的地址。
1B31:	7E	LD	A, (HL)	; A=下一行地址的 LSB。
1B32:	23	INC	HL	; HL+1, 指向下一行地址的 MSB。
1B33:	B6	OR	(HL)	; 合并 MSB 和 LSB, 并置状态标志。
1B34:	2B	DEC	HL	; 恢复 HL, 指向当前行的开头。
1B35:	C8	RET	Z	; 如果 PST 结束,则返回。
1B36:	23	INC	HL	; 否则 HL+2。
1B37:	23	INC	HL	; 指向当前行的行号。
1B38:	7E	LD	A, (HL)	; A=当前行行号的 LSB。
1B39:	23	INC	HL	; 指向 MSB。
1B3A:	66	LD	H, (HL)	; H=当前行行号的 MSB。
1B3B:	6F	LD	L, A	; L=当前行行号的 LSB。
1B3C:	DF	RST	18H	; 从当前行行号中减去 DE 中的行号。
1B3D:	60	LD	H, B	; 置 HL=当前行首址。
1B3E:	69	LD	L, C	; L=当前行首址的 LSB。
1B3F:	7E	LD	A, (HL)	; 把下一行的地址送入 HL。
1B40:	23	INC	HL	; 指向下一行地址的 MSB。
1B41:	66	LD	H, (HL)	; H=下一行地址的 MSB。
1B42:	6F	LD	L, A	; L=下一行地址的 LSB。在 HL 中形成了下一行的地址。
1B43:	3F	CCF		; 如果当前行号<DE 中的值, 则置位 C 标志。在 CCF 以后, C 标志为零。

地址	目的码	源程序		注释
1B44:	C8	RET	Z	; 若行号相同，则返回，BC=当前行地址，HL=下一行地址。
1B45:	3F	CCF		; 行号不相同，如果 DE 中的行号<当前行号，则 C 标志求反，
1B46:	C0	RET	NC	; 并返回，BC=当前行地址，HL=下一行地址。
1B47:	18E6	JR	1B2FH	; 循环，直到程序结束，或行号大于要寻找的行号。
1B49:	D0	RET	NZ	; 若是 NEW××，则句法错误。★★★ NEW 子程序★
1B4A:	CDC901	CALL	01C9H	; 清除荧光屏。
1B4D:	2AA440	LD	HL, (40A4H)	; HL=程序语句表 (PST) 的首址。
1B50:	CDF81D	CALL	1DF8H	; 关闭跟踪功能。
1B53:	32E140	LD	(40E1H), A	; 清除 AUTO 增量标志。
1B56:	77	LD	(HL), A	; 将 PST 初始化成空表，方法是把开始两个字节清零。
1B57:	23	INC	HL	; 指向第二个字节。
1B58:	77	LD	(HL),A	; 清零第二个字节。
1B59:	23	INC	HL	; 然后，把变量表的首址
1B5A:	22F940	LD	(40F9H), HL	; 初始化为 PST 的终址。
1B5D:	2AA440	LD	HL, (40A4H)	; 把 PST 地址重新装入 HL。★★★ RUN ★★★★★
1B60:	2B	DEC	HL	; 后退一个字节，
1B61:	22DF40	LD	(40DFH), HL	; 这将是程序开始执行的地址。
1B64:	061A	LD	B, 1AH	; 共有 26 个英文字母。★★★ RUN行号 由此开始★★
1B66:	210141	LD	HL, 4101H	; 把缺项字母表中的所有项初始化成 04H (单精度类型码)。
1B69:	3604	LD	(HL), 04H	; HL=字母表地址。把类型码送入表中。
1B6B:	23	INC	HL	; HL+1，指向下一项。
1B6C:	10FB	DJNZ	1B69H	; 循环，直到字母类型表被预置完毕。
1B6E:	AF	XOR	A	; 清除 A 寄存器。
1B6F:	32F240	LD	(40F2H), A	; 对于 RESUME 工作子程序表示没有错误的标志。
1B72:	6F	LD	L, A	; 然后，把 HL 清零。
1B73:	67	LD	H, A	; H 清零。
1B74:	22F040	LD	(40F0H), HL	; 置 ON ERROR 地址为零。
1B77:	22F740	LD	(40F7H), HL	; 指向 BREAK, STOP 或 END 后的下个语句。
1B7A:	2AB140	LD	HL, (40B1H)	; 最高内存指针。
1B7D:	22D640	LD	(40D6H), HL	; 字符串工作区指针。
1B80:	CD911D	CALL	1D91H	; 执行 RESTORE。
1B83:	2AF940	LD	HL, (40F9H)	; HL=BASIC 程序的结束。
1B86:	22FB40	LD	(40FBH), HL	; 简单变量指针。
1B89:	22FD40	LD	(40FDH), HL	; 数组指针。
1B8C:	CDBB41	CALL	41BBH	; DOS 出口 (JP 60A1H)。
1B8F:	C1	POP	BC	; 取返回地址，因为我们将改变堆栈指针。
1B90:	2AA040	LD	HL, (40A0H)	; HL=字符串数据指针的首址。
1B93:	2B	DEC	HL	; HL=字符串数据指针的首址－1。

地址	目的码	源程序		注释
1B94:	2B	DEC	HL	; HL=字符串数据指针的首址－2。
1B95:	22E840	LD	(40E8H), HL	; 堆栈指针＝字符串数据指针的首址－2。
1B98:	23	INC	HL	; HL 恢复为字符串数据指针的首址。
1B99:	23	INC	HL	;
1B9A:	F9	LD	SP, HL	; SP＝字符串数据指针的首址。
1B9B:	21B540	LD	HL, 40B5H	; 预置文字串库表（LSPT）为空表。
1B9E:	22B340	LD	(40B3H), HL	; 把 LSPT 的首址送入 40B3H。
1BA1:	CD8B03	CALL	038BH	; 输出设备＝显示器，检查打印机缓冲区空否。
1BA4:	CD6921	CALL	2169H	; 关闭盒式磁带机，并置输出设备为显示器。
1BA7:	AF	XOR	A	; A 清零，
1BA8:	67	LD	H, A	; 然后清除 HL，
1BA9:	6F	LD	L, A	; 以便把零推入堆栈，表示推入 'RUN'。
1BAA:	32DC40	LD	(40DCH), A	; 清除 'FOR' 语句标志。
1BAD:	E5	PUSH	HL	; 表示将 'RUN' 推入了堆栈。
1BAE:	D5	PUSH	BC	; 继续执行代码串的返回地址。
1BAF:	2ADF40	LD	HL, (40DFH)	; 把代码串地址重新送入 HL。
1BB2:	C9	RET		; 返回到调用它的程序。
1BB3:	3E3F	LD	A, 3FH	; A＝ASCII 的'?' 号。 ★★★★★★★★★★★★★★
1BB5:	CD2A03	CALL	032AH	; 显示'?'号。
1BB8:	3E20	LD	A, 20H	; A＝空格的 ASCII 码。
1BBA:	CD2A03	CALL	032AH	; 显示空格。
1BBD:	C36103	JP	0361H	; 等待键盘输入，并返回到调用它的程序。
1BC0:	AF	XOR	A	; 清零 A 寄存器。★★★★★★★★★★★★★★★★
1BC1:	32B040	LD	(40B0H), A	; 清除 DATA 语句标志。
1BC4:	4F	LD	C, A	; 清除 C 寄存器。
1BC5:	EB	EX	DE, HL	; DE＝行号后边第一个字符的地址。
1BC6:	2AA740	LD	HL, (40A7H)	; HL＝输入区指针＝代号化的字符串地址。
1BC9:	2B	DEC	HL	; 后退一个字节。
1BCA:	2B	DEC	HL	; 再后退一个字节。
1BCB:	EB	EX	DE, HL	; DE ＝ 输入字符串地址减 2，HL ＝ 当前输入字符串地址。
1BCC:	7E	LD	A, (HL)	; 从输入字符串中取下一个字符。
1BCD:	FE20	CP	20H	; 测试是否空格。
1BCF:	CA5B1C	JP	Z, 1C5BH	; 若是空格，则转移。
1BD2:	47	LD	B, A	; 保存输入字符。
1BD3:	FE22	CP	22H	; 测试是否引号(")。
1BD5:	CA771C	JP	Z, 1C77H	; 若是引号，则把引号中的整个字段送入代码串。
1BD8:	B7	OR	A	; 置状态标志。
1BD9:	CA7D1C	JP	Z, 1C7DH	; 若字符串结束，则转移。
1BDC:	3AB040	LD	A, (40B0H)	; A＝DATA 语句标志。
1BDF:	B7	OR	A	; 置状态标志。

地址	目的码	源程序		注释
1BE0:	7E	LD	A, (HL)	; 从输入字符串中取下一个字符。
1BE1:	C25B1C	JP	NZ, 1C5BH	; 若遇到 DATA 语句,则转移。
1BE4:	FE3F	CP	3FH	; 是否 PRINT 的缩写'?'。
1BE6:	3EB2	LD	A, B2H	; 用 PRINT 的代号代替问号。
1BE8:	CA5B1C	JP	Z, 1C5BH	; 若是'?' (PRINT 代号),则转移。
1BEB:	7E	LD	A, (HL)	; 再取入当前字符。
1BEC:	FE30	CP	30H	; 测试是否字母数字
1BEE:	3805	JR	C, 1BF5H	; 若字符<30H——表示它不是字母或数字。
1BF0:	FE3C	CP	3CH	; 若字符<3CH——表示是0—9,:(冒号),;(分号),
1BF2:	DA5B1C	JP	C, 1C5BH	; <(小于号)等数字或特殊符号。将它送入代号区。
1BF5:	D5	PUSH	DE	; 把指针存入缓冲区原点-2,-1, …。
1BF6:	114F16	LD	DE, 164FH	; DE=句法树的地址。(即保留字表地址)
1BF9:	C5	PUSH	BC	; 保存 BC。
1BFA:	013D1C	LD	BC, 1C3DH	; 在句法树与输入字符串
1BFD:	C5	PUSH	BC	; 相符合时的返回地址。
1BFE:	067F	LD	B, 7FH	; B=句法树控制字符计数。
1C00:	7E	LD	A, (HL)	; 当前输入字符。
1C01:	FE61	CP	61H	; 测试是否大写字母。
1C03:	3807	JR	C, 1C0CH	; 若不是小写字母,则转移。
1C05:	FE7B	CP	7BH	; 测试是否大写字母。
1C07:	3003	JR	NC, 1C0CH	; 若不是小写字母,则转移。
1C09:	E65F	AND	5FH	; 构成大写字母。
1C0B:	77	LD	(HL), A	; 存储转换后的字母。
1C0C:	4E	LD	C, (HL)	; 再取当前字符。
1C0D:	EB	EX	DE, HL	; HL=句法表地址。DE=前当字符串的地址。
1C0E:	23	INC	HL	; HL+1, 指向句法表中下一个字符。
1C0F:	B6	OR	(HL)	; 为从句法表中得到当前字符,设置状态标志。
1C10:	F20E1C	JP	P, 1C0EH	; 扫描句法表,直到找到控制字符。
1C13:	04	INC	B	; 经过的句法控制字符的计数。
1C14:	7E	LD	A, (HL)	; 取入句法元素(即保留字)。
1C15:	E67F	AND	7FH	; 清除符号位。
1C17:	C8	RET	Z	; 若是0,则是句法表的结束,转移到 1C3DH。
1C18:	B9	CP	C	; 将输入元素与句法元素进行比较。
1C19:	20F3	JR	NZ, 1C0EH	; 若不同,则继续扫描,直到通过控制元素。
1C1B:	EB	EX	DE, HL	; HL=输入字符串中当前符号的首址。
1C1C:	E5	PUSH	HL	; 存储当前符号的首址。
1C1D:	13	INC	DE	; 指向句法表中下个字符。
1C1E:	1A	LD	A, (DE)	; 取下一个句法表元素。
1C1F:	B7	OR	A	; 为了结束名称的测试,置状态标志。
1C20:	FA391C	JP	M, 1C39H	; 若是控制元素,则转移,我们取得完全的匹配。
1C23:	4F	LD	C, A	; 保存下个句法元素。

地址	目的码	源程序		注　释
1C24:	78	LD	A, B	; 如果要测定的关键字的计数是 8DH,
1C25:	FE8D	CP	8DH	; 那末我们正在测试 'GOTO'。
1C27:	2002	JR	NZ, 1C2BH	; 若不是 'GOTO' 代号,则转移。
1C29:	D7	RST	10H	; 如果它是空格,跳到下边的字符。
1C2A:	2B	DEC	HL	; 减 1,以便下跳一个字符。
1C2B:	23	INC	HL	; 跳到下个字符。
1C2C:	7E	LD	A, (HL)	; 从输入字符串中取下一个元素。
1C2D:	FE61	CP	61H	; 测试是否大写字母。
1C2F:	3802	JR	C, 1C33H	; 若不是小写,则转移。
1C31:	E65F	AND	5FH	; 强迫构成大写字母。
1C33:	B9	CP	C	; 比较输入元素和句法元素。
1C34:	28E7	JR	Z, 1C1DH	; 若相同,则转移。
1C36:	E1	POP	HL	; 不相同,则从句法表中的
1C37:	18D3	JR	1C0CH	; 刚才位置重新开始扫描
1C39:	48	LD	C, B	; 句法表索引。
1C3A:	F1	POP	AF	; 去掉在 1C1CH 处推入堆栈的 HL 的内容。
1C3B:	EB	EX	DE, HL	; HL=这个字符串的句法树地址,DE=当前字符串地址。
1C3C:	C9	RET		; 返回到 1C3DH。
1C3D:	EB	EX	DE, HL	; HL=当前字符串地址。
1C3E:	79	LD	A, C	; A=句法表索引。
1C3F:	C1	POP	BC	; 清除堆栈中的返回地址。
1C40:	D1	POP	DE	; DE=在 1CF5H 取入的输入字符串缓冲区原地址−2。
1C41:	EB	EX	DE, HL	; HL=缓冲区原点 −2, DE=当前字符串地址。
1C42:	FE95	CP	95H	; 测试是否 ELSE 代号。
1C44:	363A	LD	(HL), 3AH	; 把':'号送入缓冲区原地址−2。
1C46:	2002	JR	NZ, 1C4AH	; 若不是 'ELSE' 代号,则转移。
1C48:	0C	INC	C	; 代号缓冲区中字符计数加 1。
1C49:	23	INC	HL	; HL+1, 指向代号缓冲区中下一个位置。
1C4A:	FEFB	CP	FBH	; 测试 'REM' 缩写(')的代号。
1C4C:	200C	JR	NZ, 1C5AH	; 若不是(')号的代号,则转移。
1C4E:	363A	LD	(HL), 3AH	; 将':'送入代号缓冲区。
1C50:	23	INC	HL	; 代号缓冲区的下一个位置。
1C51:	0693	LD	B, 93H	; 'REM' 的代号。
1C53:	70	LD	(HL), B	; 送入代号缓冲区。
1C54:	23	INC	HL	; 指向代号缓冲区中的下一个位置。
1C55:	EB	EX	DE, HL	; HL=输入字符串地址, DE=代号缓冲区地址。
1C56:	0C	INC	C	; 计数加 2。
1C57:	0C	INC	C	; 要把更多字符送入代号缓冲区。
1C58:	181D	JR	1C77H	; 将注释送入代号缓冲区。
1C5A:	EB	EX	DE, HL	; DE=缓冲区地址 −2, HL=当前字符串地址。
1C5B:	23	INC	HL	; HL+1, 指向输入字符串中的下一个字符。

地址	目的码	源程序		注　释
1C5C:	12	LD	(DE), A	; 把句法树索引送入缓冲区原来地址 −2。或者，可能是空格，也将其送入。
1C5D:	13	INC	DE	; DE=缓冲区原地址 −1。
1C5E:	0C	INC	C	; C=下个句法元素的索引。
1C5F:	D63A	SUB	3AH	; 测试多语句行。
1C61:	2804	JR	Z, 1C67H	; 若是多语句行，则转移。
1C63:	FE4E	CP	4EH	; 测试 DATA 语句。
1C65:	2003	JR	NZ, 1C6AH	; 若不是 'DATA' 代号，则转移。
1C67:	32B040	LD	(40B0H), A	; 把句法表索引送入 'DATA' 语句的标志。
1C6A:	D659	SUB	59H	; 测试 'REM' 代号。
1C6C:	C2CC1B	JP	NZ, 1BCCH	; 若不是 'REM' 代号，则转移，去分析语句的其余部分。
1C6F:	47	LD	B, A	; B=00H
1C70:	7E	LD	A, (HL)	; 从输入字符串中取下一个字符。
1C71:	B7	OR	A	; 置状态标志，于是，我们可以测试 EOS (语句结束)。
1C72:	2809	JR	Z, 1C7DH	; 若是 EOS, 则转移。
1C74:	B8	CP	B	; 把语句从输入缓冲区移入输入缓冲区 −2，
1C75:	28E4	JR	Z, 1C5BH	; 循环，直到检测到 EOS。传送过的
1C77:	23	INC	HL	; 字符的计数放入 BC, 如果检测到 '' 字符串，
1C78:	12	LD	(DE), A	; 即空字符串，也送入。
1C79:	0C	INC	C	; 计数加 1，已有一个字符加到了代号缓冲区中。
1C7A:	13	INC	DE	; 代号缓冲区地址加 1。
1C7B:	18F3	JR	1C70H	; 循环，直到找到 EOS 或结束的引号。
1C7D:	210500	LD	HL, 0005H	; 现在，将目前的
1C80:	44	LD	B, H	; 代号缓冲区的
1C81:	09	ADD	HL, BC	; 长度加 5，
1C82:	44	LD	B, H	; 然后将新的
1C83:	4D	LD	C, L	; 计数放入 BC。
1C84:	2AA740	LD	HL, (40A7H)	; 取输入字符串区的首址。
1C87:	2B	DEC	HL	; 后退三个字节。
1C88:	2B	DEC	HL	;
1C89:	2B	DEC	HL	;
1C8A:	12	LD	(DE), A	; 然后，清零代号化字符串的后三个字节。
1C8B:	13	INC	DE	; 地址加 1。
1C8C:	12	LD	(DE), A	; 第二个字节清零。
1C8D:	13	INC	DE	; 地址加 1。
1C8E:	12	LD	(DE), A	; 第三个字节清零。
1C8F:	C9	RET		; 返回到调用它的程序。
1C90:	7C	LD	A, H	; 计算 HL−DE。★★★ RST 18H ★★★★★★★★★

说明 RST 18H 将转移到这儿，计算 HL−DE，若两者相等，则 Z 标志置位；若 DE>HL，则 C 标志置位。

地址	目的码	源程序		注解
1C91:	92	SUB	A, D	;
1C92:	C0	RET	NZ	; 若相等,则返回。
1C93:	7D	LD	A, L	; 计算 L—E。
1C94:	93	SUB	A, E	;
1C95:	C9	RET		; 返回到调用它的程序。
1C96:	7E	LD	A, (HL)	; 取要比较的值。★★★ RST 08H 子程序 ★★★★★
1C97:	E3	EX	(SP), HL	; 保存返回地址。
1C98:	BE	CP	(HL)	; 将 (HL) 与 RST 08H 后的值进行比较。
1C99:	23	INC	HL	; 返回地址加 1。
1C9A:	E3	EX	(SP), HL	; 把返回地址再存入堆栈，HL=当前代码串指针
1C9B:	CA781D	JP	Z, 1D78H	; 如果找到所需的字符,则调用 RST 10H 子程序。
1C9E:	C39719	JP	1997H	; 如果所需的字符没找到,则为 SN 错误。
1CA1:	3E64	LD	A, 64H	; FOR 标志值。★★★ FOR 子程序 ★★★★★★★★
1CA3:	32DC40	LD	(40DCH), A	; 标志 FOR 语句
1CA6:	CD211F	CALL	1F21H	; 计算 $x=y$ (索引)
1CA9:	E3	EX	(SP), HL	; 保存代码串地址。DE=索引变量的地址。
1CAA:	CD3619	CALL	1936H	; 往后扫描堆栈，寻找具有相同索引的 FOR/NEXT 代号(若找到,则错误)。
1CAD:	D1	POP	DE	; DE=当前代码串地址 ('TO' 代号的地址)。
1CAE:	2005	JR	NZ, 1CB5H	; 若堆栈中没有嵌套的 FOR，则转移。 如果找到嵌套的 FOR，出口时，HL=FOR 在堆栈中的首址。
1CB0:	09	ADD	HL, BC	; BC=对于堆栈终址的偏置,相加后,处在第一个 FOR 结构的末尾。
1CB1:	F9	LD	SP, HL	; 将 CSP 重新设置成这个地址，收回堆栈空间，并给出 NF 错误。
1CB2:	22E840	LD	(40E8H), HL	; 把 CSP 地址存入 40E8H。
1CB5:	EB	EX	DE, HL	; HL=当前代码串地址。
1CB6:	0E08	LD	C, 08H	; C=所需空间的一半。
1CB8:	CD6319	CALL	1963H	; 验证有 16 个字节的空闲内存。
1CBB:	E5	PUSH	HL	; 保存 'TO' 以前的代码串地址。
1CBC:	CD051F	CALL	1F05H	; 扫描,直到语句结束。
1CBF:	E3	EX	(SP), HL	; 堆栈=语句的末址，HL=语句中当前位置。
1CC0:	E5	PUSH	HL	; 存入堆栈的代码串地址应指向 'TO' 代号。
1CC1:	2AA240	LD	HL, (40A2H)	; HL=二进制形式的当前行号。
1CC4:	E3	EX	(SP), HL	; 堆栈=行的结束地址，FOR 语句的二进制行号。
1CC5:	CF	RST	08H	; 测试 'TO' 代号
1CC6:	BD	CP	L	; 定义 'TO' 的代号: BDH。
1CC7:	E7	RST	20H	; 测试索引变量的数据类型。
1CC8:	CAF60A	JR	Z, 0AF6H	; 若 Z 标志置位(字符串)，TM 错误。
1CCB:	D2F60A	JP	NC, 0AF6H	; 若 C 标志复位(双精度)，TM 错误。

地址	目的码	源程序		注　释
1CCE:	F5	PUSH	AF	; 保存类型标志。
1CCF:	CD3723	CALL	2337H	; 计算 FOR 语句的 TO 边界。
1CD2:	F1	POP	AF	; 恢复类型标志。
1CD3:	E5	PUSH	HL	; 保存 TO 代号后的代码串中的当前位置。
1CD4:	F2EC1C	JP	P, 1CECH	; 若索引是单精度数,则转移。
1CD7:	CD7F0A	CALL	0A7FH	; 将当前 TO 值转换成整数。
1CDA:	E3	EX	(SP), HL	; 把整数值存入堆栈,把堆栈中的当前位置重新装入 HL。
1CDB:	110100	LD	DE, 0001H	; DE=在不指定 STEP 时的增量。
1CDE:	7E	LD	A, (HL)	; 从代码串中取入下个元素。
1CDF:	FECC	CP	CCH	; 与 'STEP' 代号比较。
1CE1:	CC012B	CALL	Z, 2B01H	; 若是 'STEP' 代号,则调子程序——将步长值送入 DE。
1CE4:	D5	PUSH	DE	; 保存步长值。
1CE5:	E5	PUSH	HL	; 保存代码串位置。
1CE6:	EB	EX	DE, HL	; 把 STEP 值送入 HL, 以便测试它的大小。
1CE7:	CD9E09	CALL	099EH	; 把 STEP 值的符号取入 A。若是正的, 则 A=+1, 若是负的, A=−1。
1CEA:	1822	JR	1D0EH	; 跳过计数和 STEP 值的单精度码。
1CEC:	CDB10A	CALL	0AB1H	; 把 TO 值转换成单精度。
1CEF:	CDBF09	CALL	09BFH	; 把计数装入 BC 和 DE。
1CF2:	E1	POP	HL	; HL=TO 表达式的结束地址。
1CF3:	C5	PUSH	BC	; 保存 TO 值(极限)的 4 个字节。
1CF4:	D5	PUSH	DE	;
1CF5:	010081	LD	BC, 8100H	; BC=单精度值的 01H=缺项 STEP 值。
1CF8:	51	LD	D, C	; DE=0000H。
1CF9:	5A	LD	E, D	;
1CFA:	7E	LD	A, (HL)	; A=代码串中的下个元素。
1CFB:	FECC	CP	CCH	; 测试 'STEP' 代号。
1CFD:	3E01	LD	A, 01H	; 缺项步长=1。
1CFF:	200E	JR	NZ, 1D0FH	; 若不是 'STEP' 代号,则转移。
1D01:	CD3823	CALL	2338H	; 计算 STEP 表达式。
1D04:	E5	PUSH	HL	; 保存代码串地址。
1D05:	CDB10A	CALL	0AB1H	; 将 STEP 值转换成单精度。
1D08:	CDBF09	CALL	09BFH	; 把 STEP 表达式的值送入 BC 和 DE。
1D0B:	CD5509	CALL	0955H	; 把 STEP 值的符号取入 A, +1 表示正, −1 表示负。
1D0E:	E1	POP	HL	; HL=当前代码串地址。
1D0F:	C5	PUSH	BC	; 把 STEP 值保存在堆栈中。
1D10:	D5	PUSH	DE	;
1D11:	4F	LD	C, A	; 取 STEP 值的符号标志放入 C。
1D12:	E7	RST	20H	; 测试 STEP 值的数据类型。
1D13:	47	LD	B, A	; B=STEP 值的类型: −1(整数), +1(单精度)。
1D14:	C5	PUSH	BC	; 保存数据类型和符号标志。

地址	目的码	源程序		注　释
1D15:	E5	PUSH	HL	; 把当前代码串地址存入堆栈。
1D16:	2ADF40	LD	HL, (40DFH)	; HL 为由 FOR *x*=*y* 得到的索引的地址。
1D19:	E3	EX	(SP), HL	; HL=代码串地址,堆栈=*x* 变量的地址。
1D1A:	0681	LD	B, 81H	; B='FOR' 代号
1D1C:	C5	PUSH	BC	; 保存 'FOR' 代号和 STEP 增量的符号。
1D1D:	33	INC	SP	; 在堆栈中留一个字节的间隙。
1D1E:	CD5803	CALL	0358H	; 继续处理代码串。测试键盘输入。
1D21:	B7	OR	A	; 为输入设置状态标志。
1D22:	C4A01D	CALL	NZ, 1DA0H	; 如果按了键,检测是否是 shift @
1D25:	22E640	LD	(40E6H), HL	; 存储当前行中刚执行过的上一个字节的地址。
1D28:	ED73E840	LD	(40E8H), SP	; 存储 CSP (当前堆栈指针)。
1D2C:	7E	LD	A, (HL)	; 从输入字符串中取下一个字符。
1D2D:	FE3A	CP	3AH	; 并测试是否是多语句。
1D2F:	2829	JR	Z, 1D5AH	; 若是冒号':'——这行是多语句,则转移。
1D31:	B7	OR	A	; 否则,验证是否代码串结束。置状态标志。
1D32:	C29719	JP	NZ, 1997H	; 若有一个零字节,且 C 标志复位,则为 SN 错误。
1D35:	23	INC	HL	; HL+1, 指向下一个字符。
1D36:	7E	LD	A, (HL)	; 取入指向下个语句的指针的 LSB。
1D37:	23	INC	HL	; 并利用把它和下个语句的指针的 MSB 字节合并的方
1D38:	B6	OR	(HL)	; 法,来测试是否非零。
1D39:	CA7E19	JP	Z, 197EH	; 如果是最后一个可执行语句,则转移。
1D3C:	23	INC	HL	; 否则,把下个语句的行号取入 DE。
1D3D:	5E	LD	E, (HL)	; 取入下个语句的行号的 LSB。
1D3E:	23	INC	HL	; HL+1, 指向下个语句行号的 MSB。
1D3F:	56	LD	D, (HL)	; DE=下个语句的二进制行号。
1D40:	EB	EX	DE, HL	; HL=下个语句的行号。
1D41:	22A240	LD	(40A2H), HL	; 把刚执行过的行修改成当前行。
1D44:	3A1B41	LD	A, (411BH)	; 取跟踪功能的标志。
1D47:	B7	OR	A	; 置状态标志。
1D48:	280F	JR	Z, 1D59H	; 若是 TROFF, 则转移。若是 TRON, 则执行下边的程序。
1D4A:	D5	PUSH	DE	; 保存 DE, 因为显示子程序将使用它。
1D4B:	3E3C	LD	A, 3CH	; '<' 号的 ASCII 码。
1D4D:	CD2A03	CALL	032AH	; 显示'<'号。
1D50:	CDAF0F	CALL	0FAFH	; 把二进制行号转换成 ASCII 码并显示行号。
1D53:	3E3E	LD	A, 3EH	; '>' 号的 ASCII 码。
1D55:	CD2A03	CALL	032AH	; 显示'>'号(于是给出了:<行号>)。
1D58:	D1	POP	DE	; 恢复 DE。
1D59:	EB	EX	DE, HL	; HL=代码串的当前行。
1D5A:	D7	RST	10H	; 取下个代号。★★★执行阶段由此开始 ★★★★★★

说明　在代码串中寻找下个非空格字符。方法:

地址	目的码	源 程 序		注 释

1.在当前语句中寻找下个代号,并转移到工作子程序。在执行工作子程序以后强迫返回到 1D1EH。
2.在工作子程序每次完成对 BREAK 键、行的结束(指向下一行)、程序的结束(返回到输入阶段)、或 TRON 选择项等的测试以后,至步骤 1。

```
1D5B: 111E1D   LD    DE, 1D1EH  ; 在执行一个工作子程序以后的返回地址。
1D5E: D5       PUSH  DE         ; 把返回地址推入堆栈。
1D5F: C8       RET   Z          ; 若是 EOS (语句结束),则返回——返回到 1D1EH。
1D60: D680     SUB   80H        ; (代号的范围是 80H—FBH) 计算相对的代号索引。
1D62: DA211F   JP    C, 1F21H   ; 不是代号——必须是赋值语句。
1D65: FE3C     CP    3CH        ; 测试代号是否在 TAB 代号的下边。
1D67: D2E72A   JP    NC, 2AE7H  ; 如果代号=>BCH (TAB—MID$ 和'),则转移。
1D6A: 07       RLCA             ; 子程序地址偏移值的双精度余数。
1D6B: 4F       LD    C, A       ; BC=子程序地址偏移值。
1D6C: 0600     LD    B, 00H     ; B=00H, C=2×代号。
1D6E: EB       EX    DE, HL     ; 保存 HL (代码串中当前位置)。
1D6F: 212218   LD    HL, 1822H  ; 工作子程序的地址表。
1D72: 09       ADD   HL, BC     ; HL=子程序表地址指针。
1D73: 4E       LD    C, (HL)    ; C=工作子程序地址的 LSB。
1D74: 23       INC   HL         ; HL+1, 指向 MSB。
1D75: 46       LD    B, (HL)    ; B=工作子程序地址的 MSB。
1D76: C5       PUSH  BC         ; 将子程序地址存入堆栈。(它将在下边被弹出。)
1D77: EB       EX    DE, HL     ; 恢复代码串地址。
1D78: 23       INC   HL         ; HL+1, 指向下一个字符。★★★ RST 10H 工作程序
                                ★★★★★★★★★★★★★★★★★★★★★★
1D79: 7E       LD    A, (HL)    ; 取下个字符。
1D7A: FE3A     CP    3AH        ; 将它与冒号(:)比较。
1D7C: D0       RET   NC         ; 如果字符是:, ;, <,… A—Z, 则返回。
1D7D: FE20     CP    20H        ; 否则,测试是否空格。
1D7F: CA781D   JP    Z, 1D78H   ; 如果是一个空格,则取下一个字符。
1D82: FE0B     CP    0BH        ; 将它与垂直的 TAB 比较。
1D84: 3005     JR    NC, 1D8BH  ; 如果 A>=0BH (不是控制代码), 则转移。
1D86: FE09     CP    09H        ; 测试水平 TAB。
1D88: D2781D   JP    NC, 1D78H  ; 若不是水平 TAB 或换行,则转移。
1D8B: FE30     CP    30H        ; 与 ASCII '0' 比较。
1D8D: 3F       CCF              ; 若是数字 (>=30H), 置位 C 标志。
1D8E: 3C       INC   A          ; 若不是数字 (<30H), 清除 C 标志。
1D8F: 3D       DEC   A          ; 根据刚装入的字符置状态标志 (C 标志除外)。
1D90: C9       RET              ; 返回到调用它的程序。
1D91: EB       EX    DE, HL     ; 保存 HL。★★★ RESTORE子程序 ★★★★★★★
1D92: 2AA440   LD    HL, (40A4H); HL=程序首址的指针。
1D95: 2B       DEC   HL         ; 后退一个字节,并存储 HL。
1D96: 22FF40   LD    (40FFH), HL ; 数据指针=程序首址-1。
```

地址	目的码	源程序		注　释
1D99:	EB	EX	DE, HL	; 恢复 HL。
1D9A:	C9	RET		; 返回到调用它的程序。
1D9B:	CD5803	CALL	0358H	; 扫描键盘一次。★★★★★★★★★★★★★★★★★
1D9E:	B7	OR	A	; 置状态标志,以确定是否有键入字符。
1D9F:	C8	RET	Z	; 若没有按键,则返回。
1DA0:	FE60	CP	60H	; 是 shift @ 吗?
1DA2:	CC8403	CALL	Z, 0384H	; 若是,则等待,直到用户打入一个字符为止。
1DA5:	329940	LD	(4099H), A	; 存储打入的字符。
1DA8:	3D	DEC	A	; A−1, 是否 BREAk 键。
1DA9:	C0	RET	NZ	; 若不是,则返回。★★★ STOP 子程序 ★★★★★★
1DAA:	3C	INC	A	; 置 A=1, 状态为非零。
1DAB:	C3B41D	JP	1DB4H	; 使用 END 程序。
1DAE:	C0	RET	NZ	; 若是 END ××, 则为句法错误。★★★ END 子程序 ★★★★★★★★★★★★★★★★★★★★★★★★★
1DAF:	F5	PUSH	AF	; 保存零状态 (END 处理)。
1DB0:	CCBB41	CALL	Z, 41BBH	; DOS 出口 (JP 60A1H)。
1DB3:	F1	POP	AF	; 将 END 状态重新存入 A 和状态寄存器。
1DB4:	22E640	LD	(40E6H), HL	; STOP 或 END 的当前代码串地址。
1DB7:	21B540	LD	HL, 40B5H	; HL=文字串区的首址。
1DBA:	22B340	LD	(40B3H), HL	; 重新设置指向文字串区首址的指针。
1DBD:	21F6FF	LD	HL, FFF6H	; 1DBEH: OR FFH
1DC0:	C1	POP	BC	; 清除堆栈。
1DC1:	2AA240	LD	HL, (40A2H)	; 二进制形式的当前行号。
1DC4:	E5	PUSH	HL	; 保存 STOP 或 END 语句的二进制行号。
1DC5:	F5	PUSH	AF	; A=0 (END), 1(STOP)。
1DC6:	7D	LD	A, L	; 将当前行号的 LSB 与当前行号的
1DC7:	A4	AND	H	; MSB 合并,于是我们可以
1DC8:	3C	INC	A	; 测试是否为行号 FFFFH。
1DC9:	2809	JR	Z, 1DD4H	; 如果行号=FFFFH, 即程序还没开始执行,则转移。
1DCB:	22F540	LD	(40F5H), HL	; 否则,把终止执行的行号存储起来。
1DCE:	2AE640	LD	HL, (40E6H)	; HL=当前行号,
1DD1:	22F740	LD	(40F7H), HL	; 存入 40F7H 单元。
1DD4:	CD8B03	CALL	038BH	; 预置输出 DCB 为显示器。
1DD7:	CDF920	CALL	20F9H	; 输出一个回车符。
1DDA:	F1	POP	AF	; 恢复 A=0 (END), 1 (STOP)。
1DDB:	213019	LD	HL, 1930H	; 'BREAk' 信息的地址。
1DDE:	C2061A	JP	NZ, 1A06H	; 如果遇到的是 STOP, 则转移,给出 'BREAK' 信息。
1DE1:	C3181A	JP	1A18H	; 如果是 END 语句或命令方式中的错误,则转移, 给出 'READY' 信息。
1DE4:	2AF740	LD	HL, (40F7H)	; HL=扫描过的最后语句字节。★★★ CONT 子程序 ★★★★★★★★★★★★★★★★★★★★★★★★★

地址	目的码	源程序		注　释
1DE7:	7C	LD	A, H	; 把执行过的最后语句的地址
1DE8:	B5	OR	L	; 的 LSB 和 MSB 合并。
1DE9:	1E20	LD	E, 20H	; CN 错误代码。
1DEB:	CAA219	JP	Z, 19A2H	; 若没有继续地址，则输出 CN 错误信息。
1DEE:	EB	EX	DE, HL	; 把继续行号的地址放入 DE。
1DEF:	2AF540	LD	HL, (40F5H)	; HL＝刚执行的最后的行号。
1DF2:	22A240	LD	(40A2H), HL	; 把有错误的行号存储起来。
1DF5:	EB	EX	DE, HL	; 然后，置 HL＝继续行号的地址。
1DF6:	C9	RET		; 转移到继续行去开始执行。
1DE7:	3EAF	LD	A, AFH	; 置A寄存器为非零，作为 TRON 标志。 ★★★ TRON 子程序 ★★★★★★★★★★★★★★★★★★★★
1DF9:	321B41	LD	(411BH), A	; 1DF8H: XOR A; 置 A 为零，作为 TROFF 标志。
1DFC:	C9	RET		; 存储 TRON 或 TROFF 标志，并返回到解释程序。
1DFD:	F1	POP	AF	; 以下这些指令，LEVEL II 没有使用。
1DFE:	E1	POP	HL	; •
1DFF:	C9	RET		; •
1E00:	1E03	LD	E, 03H	; E＝字符串值的类型码。 ★★★ DEFSTR 子程序 ★★
1E02:	011E02	LD	BC, 021EH	; 1E03H: LD E, 02; DEFINT 子程序。
1E05:	011E04	LD	BC, 041EH	; 1E06H: LD E, 04; DEFSNG 子程序。
1E08:	011E08	LD	BC, 081EH	; 1E09H: LD E, 08; DEFDBL 子程序。
1E0B:	CD3D1E	CALL	1E3DH	; 测试代码串中的下个元素，证明它是字母。
1E0E:	019719	LD	BC, 1997H	; 当它不是字母时的错误信息地址。
1E11:	C5	PUSH	BC	; 把错误信息地址推入堆栈。
1E12:	D8	RET	C	; 若 DEFSTR 等语句后边没有字母，则为句法错误。
1E13:	D641	SUB	41H	; 减去 'A' 的 ASCII 码，给出范围在 0—25 之间的值。
1E15:	4F	LD	C, A	; 把范围值存入 C,
1E16:	47	LD	B, A	; 存入 B。
1E17:	D7	RST	10H	; 测试代码串中的下个元素。
1E18:	FECE	CP	CEH	; 是否破折号(即减号－)的代号。
1E1A:	2009	JR	NZ, 1E25H	; 没有指定字母范围，转移。
1E1C:	D7	RST	10H	; 指定了一个字母范围，取结束字母。
1E1D:	CD3D1E	CALL	1E3DH	; 检查是否字母。
1E20:	D8	RET	C	; 若不是字母，则为句法错误。
1E21:	D641	SUB	41H	; A＝0—25，对应字母 A—Z。
1E23:	47	LD	B, A	; 放入 B。
1E24:	D7	RST	10H	; 取下个字符。
1E25:	78	LD	A, B	; 证实第二个字母在第一个字母的后边。
1E26:	91	SUB	A, C	; 第二个字母减去第一个字母。
1E27:	D8	RET	C	; 如果字母范围不是升序的，则为句法错误6
1E28:	3C	INC	A	; A＝要改变类型的项数。
1E29:	E3	EX	(SP), HL	; 清除错误处理地址，保存当前代码串地址。

地址	目的码	源程序		注　释
1E2A:	210141	LD	HL, 4101H	; HL=类型表地址。
1E2D:	0600	LD	B, 00H	; B=00H, C=第一个字母的值。
1E2F:	09	ADD	HL, BC	; 找到类型表中的下一项。
1E03:	73	LD	(HL), E	; 在类型表中设置数据类型。
1E31:	23	INC	HL	; 指向下个项目。
1E32:	3D	DEC	A	; 已修改了一项,计数减 1。
1E33:	20FB	JR	NZ, 1E30H	; 循环,直到所指定范围中的项目都改变完了为止。
1E35:	E1	POP	HL	; 恢复代码串指针,
1E36:	7E	LD	A, (HL)	; 并寻找其他字符。
1E37:	FE2C	CP	2CH	; 测试是否逗号。
1E39:	C0	RET	NZ	; 若不是逗号,返回。
1E3A:	D7	RST	10H	; 取下一个元素,
1E3B:	18CE	JR	1E0BH	; 并测试是否字母。
1E3D:	7E	LD	A, (HL)	; 从代码串中取入下个元素。 ★★★★★★★★★★★
1E3E:	FE41	CP	41H	; 与 ASCII 的 A 比较。
1E40:	D8	RET	C	; 若不是字母,则返回。
1E41:	FE5B	CP	5BH	; 与'↑'的 ASCII 码比较,若是字母,则 C 标志置位。
1E43:	3F	CCF		; 若不是字母,则 C 标志被置位。
1E44:	C9	RET		; 若是字母,则 C 标志被复位。
1E45:	D7	RST	10H	; 从输入中取下个字符。★★★ 在计算被引用变量的下标时调用 ★★★★★★★★★★★★★★★★★★★★
1E46:	CD022B	CALL	2B02H	; 得到下个表达式的值,作为整数放入 DE,
1E49:	F0	RET	P	; 若是正的,则设置为下标,返回。
1E4A:	1E08	LD	E, 08H	; 若是负的,则为 FC 错误。
1E4C:	C3A219	JP	19A2H	; 输出 FC 错误信息。
1E4F:	7E	LD	A, (HL)	; 取下一个字符。★★★ ASCII 转换成二进制 ★★★★
1E50:	FE2E	CP	2EH	; 检查小数点。
1E52:	EB	EX	DE, HL	; DE=当前输入符号的地址。
1E53:	2AEC40	LD	HL, (40ECH)	;
1E56:	EB	EX	DE, HL	; DE=小数点地址, HL=当前符号的地址。
1E57:	CA781D	JP	Z, 1D78H	; 若是小数点,则转移。
1E5A:	2B	DEC	HL	; ★★★ ASCII 转换成二进制 ★★★★★★★★★★★ 退回到当前字符,从小数点(·)开始,并往后倒退着进行。
1E5B:	110000	LD	DE, 0000H	; 累加值预置成零。
1E5E:	D7	RST	10H	; 重新处理前边的字符。
1E5F:	D0	RET	NC	; 若不是数字,则返回。
1E60:	E5	PUSH	HL	; 保存当前字符指针(数字)。
1E61:	F5	PUSH	AF	; 保存由 RST 10H 得到的数字和标志。
1E62:	219819	LD	HL, 1998H	; HL=6552。
1E65:	DF	RST	18H	; 累加值 >6552 吗?

地址	目的码	源程序		注释
1E66:	DA9719	JP	C, 1997H	; 若累加值 >6552, 则为 SN 错误。
1E69:	62	LD	H, D	; 若不大于,则继续进行。
1E6A:	6B	LD	L, E	; 把当前值送入 HL。
1E6B:	19	ADD	HL, DE	; DE×2
1E6C:	29	ADD	HL, HL	; DE×4
1E6D:	19	ADD	HL, DE	; DE×5
1E6E:	29	ADD	HL, HL	; HL=DE×10
1E6F:	F1	POP	AF	; 取回上一个 ASCII 数字。
1E70:	D630	SUB	30H	; 将它转换成二进制,
1E72:	5F	LD	E, A	; 并存入 E 寄存器。
1E73:	1600	LD	D, 00H	; DE=0000—0009 (数字的二进制值)。
1E75:	19	ADD	HL, DE	; 把最后的一位数字加到目前的总数上。
1E76:	EB	EX	DE, HL	; DE=DE×10+A
1E77:	E1	POP	HL	; 恢复下个数字的指针。
1E78:	18E4	JR	1E5EH	; 处理下个数字。
1E7A:	CA611B	JP	Z, 1B61H	; 若没有字节个数,则转移。★★★ CLEAR 子程序★★
1E7D:	CD461E	CALL	1E46H	; 把字节数送入 DE。
1E80:	2B	DEC	HL	; 代码串地址后退一个字节。
1E81:	D7	RST	10H	; 检查输入流中的下一个字符。
1E82:	C0	RET	NZ	; 若不是行的结束,则返回。
1E83:	E5	PUSH	HL	; 保存当前代码串指针。
1E84:	2AB140	LD	HL, (40B1H)	; 把内存终端指针送入 HL。
1E87:	7D	LD	A, L	; DE=为字符串保留的字节数 (n)。
1E88:	93	SUB	A, E	; 内存终端指针的 LSB 减去 n 的 LSB。
1E89:	5F	LD	E, A	; 保存 LSB 的差。
1E8A:	7C	LD	A, H	; 取入内存终端指针的 MSB。
1E8B:	9A	SBC	A, D	; 内存终端指针的 MSB 减去 n 的 MSB。
1E8C:	57	LD	D, A	; 把 MSB 的差保存在 D 中。
1E8D:	DA7A19	JP	C, 197AH	; 如果要保留的字节比可用的字节多, 则产生 OM 错误。
1E90:	2AF940	LD	HL, (40F9H)	; HL=程序终址的指针。
1E93:	012800	LD	BC, 0028H	; BC=需要的最小数量的变量空间。
1E96:	09	ADD	HL, BC	; 加程序终址的指针,给出最初的字符串区。
1E97:	DF	RST	18H	; 与字符串区的首址进行比较。
1E98:	D27A19	JP	NC, 197AH	; 如果字符串表覆盖变量表,则产生 OM错误。
1E9B:	EB	EX	DE, HL	; HL=字符串区的新的首址。
1E9C:	22A040	LD	(40A0H), HL	; 存储字符串区首址的指针。
1E9F:	E1	POP	HL	; 恢复代码串指针。
1EA0:	C3611B	JP	1B61H	; 转入在 RUN 子程序中的公用程序。
1EA3:	CA5D1B	JP	Z, 1B5DH	; 若没有指定行号,则转移。★★★ RUN 子程序★★★
1EA6:	CDC741	CALL	41C7H	; DOS 出口 (JP 5F78H)。
1EA9:	CD611B	CALL	1B61H	; 初始化运行时变量。

地址	目的码	源程序		注释
1EAC:	011E1D	LD	BC, 1D1EH	; 在执行驱动程序中的继续地址。
1EAF:	1810	JR	1EC1H	; 用 GOTO 程序在指定的行号开始执行。
1EB1:	0E03	LD	C, 03H	; 需要的双字节数。★★★ GOSUB 子程序 ★★★★★
1EB3:	CD6319	CALL	1963H	; 证明至少有 6 个字节的可用内存空间。
1EB6:	C1	POP	BC	; BC=执行驱动程序中的返回地址。
1EB7:	E5	PUSH	HL	; 保存代码串地址。
1EB8:	E5	PUSH	HL	; 并建立一个空区，它将在后边被填充。
1EB9:	2AA240	LD	HL, (40A2H)	; HL=当前行号的二进制值。
1EBC:	E3	EX	(SP), HL	; 把它存在堆栈的空区中，在 HL 中恢复代码串指针。
1EBD:	3E91	LD	A, 91H	; 将 145 存入堆栈，
1EBF:	F5	PUSH	AF	; 作为 GOSUB 的标志。
1EC0:	33	INC	SP	; 堆栈指针后退跳过状态标志。
1EC1:	C5	PUSH	BC	; 保存执行驱动程序中的返回地址。使用 GOTO 程序。
1EC2:	CD5A1E	CALL	1E5AH	; 把要转入的行号取入 DE。★★★ GOTO 子程序 ★★
1EC5:	CD071F	CALL	1F07H	; 跳到这一行的末尾。
1EC8:	E5	PUSH	HL	; 保存下行的代码串地址。
1EC9:	2AA240	LD	HL, (40A2H)	; HL=最后行号的二进制值。
1ECC:	DF	RST	18H	; 将目标行号与当前行号进行比较。
1ECD:	E1	POP	HL	; 恢复代码串地址。
1ECE:	23	INC	HL	;
1ECF:	DC2F1B	CALL	C, 1B2FH	; 若当前行号大于目标行号，则目标行在前边，去寻找在 DE 中指定的行号。
1ED2:	D42C1B	CALL	NC, 1B2CH	; 否则目标行在后边，要往后寻找 DE 中的指定行号。
1ED5:	60	LD	H, B	; 在返回时，BC=所需行号的地址。
1ED6:	69	LD	L, C	; 把目标行代码串地址送入 HL。
1ED7:	2B	DEC	HL	; 退到行的开头。
1ED8:	D8	RET	C	; 返回到执行驱动程序，开始执行新的行。
1ED9:	1E0E	LD	E, 0EH	; UL 错误代码。行号没有找到。
1EDB:	C3A219	JP	19A2H	; 输出 UL 错误信息。
1EDE:	C0	RET	NZ	; 如果是 RETURN××，则为句法错误。★★★ RETURN 子程序 ★★★★★★★★★★★★★★★★★★★★★★★
1EDF:	16FF	LD	D, FFH	; 置 DE 为虚地址，供搜索子程序使用。并且 A−1，供扫描子程序使用。
1EE1:	CD3619	CALL	1936H	; 堆栈指针退后 4 个字节，将值装入 A。
1EE4:	F9	LD	SP, HL	; 将堆栈指针设置成已退后的地址。
1EE5:	22E840	LD	(40E8H), HL	; 存储已后退的堆栈地址。
1EE8:	FE91	CP	91H	; 并寻找 GOSUB 标记，
1EEA:	1E04	LD	E, 04H	; 如果有 RETURN 而没有 GOSUB，则为 RG 错误，
1EEC:	C2A219	JP	NZ, 19A2H	; 显示错误信息。
1EEF:	E1	POP	HL	; HL=GOSUB 子程序的二进制行号。
1EF0:	22A240	LD	(40A2H), HL	; 把它作为当前行号存储起来。

地址	目的码	源程序		注释
1EF3:	23	INC	HL	; 指向下一出。
1EF4:	7C	LD	A, H	; 证明行号没有溢出。
1EF5:	B5	OR	L	;
1EF6:	2007	JR	NZ, 1EFFH	; 若没有溢出,则转移。
1EF8:	3ADD40	LD	A, (40DDH)	; 否则,我们有一行程序。
1EFB:	B7	OR	A	; 取输入阶段的标志,并测试它。
1EFC:	C2181A	JP	NZ, 1A18H	; 若仍在输入阶段,则转移。
1EFF:	211E1D	LD	HL, 1D1EH	; HL=执行驱动程序中的返回地址。
1F02:	E3	EX	(SP), HL	; 把返回地址存入堆栈, HL=GOSUB 子程序的代码串地址。
1F03:	3EE1	LD	A, E1H	; 1F04H: POP HL; 现在扫描 GOSUB 语句的结束,并返回到执行驱动程序。
1F05:	013A0E	LD	BC, 0E3AH	; ★★★ DATA 子程序★★★★★★★★★★★★★★★

说明 设置停止扫描字符为:搜索代码串直到找到行的结束(00H),或者遇到停止扫描值 (00H) 或(:)为止。
对于引号或 'IF' 代号,执行下列操作:
引号时,则无条件地把停止扫描字符复位成 (00H);
'IF' 代号时,对于停止扫描字符=(00H), 则不干什么;
对于停止扫描字符=(:),则 D 寄存器加 1。

1F08:	00	NOP		; 1F07H: LD C, 00; 设置停止扫描字符为 00。
1F09:	0600	LD	BC,00H	; B=00
1F0B:	79	LD	A, C	; 保存原来的停止扫描字符。
1F0C:	48	LD	C, B	; 重新设置停止扫描字符为 00。
1F0D:	47	LD	B, A	; B=停止扫描值。
1F0E:	7E	LD	A, (HL)	; 从代码串中取入一个元素。
1F0F:	B7	OR	A	; 测试是否是行的结束。
1F10:	C8	RET	Z	; 如果是行结束,则返回。
1F11:	B8	CP	B	; 测试停止扫描符。
1F12:	C8	RET	Z	; 若遇到停止扫描符,则返回。
1F13:	23	INC	HL	; HL+1, 指向代码串中的下个元素。
1F14:	FE22	CP	22H	; 测试引号(")。
1F16:	28F3	JR	Z, 1F0BH	; 若是引号,则再置停止扫描符为 00。
1F18:	D68F	SUB	8FH	; 不是引号,测试 'IF' 代号。
1F1A:	20F2	JR	NZ, 1F0EH	; 若不是 'IF' 代号,则转移。
1F1C:	B8	CP	B	; A=0, 若 B=0, 则 C 标志= 0,
1F1D:	8A	ADC	A, D	; 并且,加指令不改变 D 的值。
1F1E:	57	LD	D, A	; 若B<>0, 则 C 标志= 1,
1F1F:	18ED	JR	1F0EH	; 并且循环一次, D 就加 1。
1F21:	CD0D26	CALL	260DH	; 把变量地址取入 DE。★★★ LET 子程序 ★★★★★
1F24:	CF	RST	08H	; 测试变量名后是否跟着 '='号,若没有,则有错误。
1F25:	D5	PUSH	DE	; D5H 是 '='的代号。
1F26:	EB	EX	DE, HL	; 变量名的地址送入 HL。

地址	目的码	源程序		注　释
1F27:	22DF40	LD	(40DFH), HL	; 把赋值变量的地址存起来。
1F2A:	EB	EX	DE, HL	; 把变量的下一个输入地址重新存入 HL。
1F2B:	D5	PUSH	DE	; 保存变量的地址。
1F2C:	E7	RST	20H	; 测定数据类型。
1F2D:	F5	PUSH	AF	; 保存类型和标志： A=−1（整），0（字符串），1(单精度），5（双精度）。
1F2E:	CD3723	CALL	2337H	; 计算表达式,把结果作为当前变量存起来。
1F31:	F1	POP	AF	; 恢复原先的数据。
1F32:	E3	EX	(SP), HL	; 将当前代码串地址推入堆栈，HL=变量的地址。
1F33:	C603	ADD	A, 03H	; 把数据复原成：2（整），3（字符串），4（单精度），8（双精度）。
1F35:	CD1928	CALL	2819H	; 把表达式结果转换成相应的类型。
1F38:	CD030A	CALL	0A03H	; 把结果送入“当前”数值区。
1F3B:	E5	PUSH	HL	; 保存变量的地址。
1F3C:	2028	JR	NZ, 1F66H	; 若结果不是字符串,则转移。
1F3E:	2A2141	LD	HL, (4121H)	; HL=指向字符串项的指针。
1F41:	E5	PUSH	HL	; 将它存入堆栈。
1F42:	23	INC	HL	; 跳过长度。
1F43:	5E	LD	E, (HL)	; E=字符串地址的 LSB。
1F44:	23	INC	HL	; 指向字符串地址的 MSB。
1F45:	56	LD	D, (HL)	; D=字符串地址的 MSB。
1F46:	2AA440	LD	HL, (40A4H)	; HL=程序首地的指针。
1F49:	DF	RST	18H	; 将程序堆栈指针与字符串变量的地址进行比较。
1F4A:	300E	JR	NC, 1F5AH	; 若字符串在程序前边,则转移。
1F4C:	2AA040	LD	HL, (40A0H)	; HL=字符串数据指针。
1F4F:	DF	RST	18H	; 将字符串地址与字符串区的低端边界比较。
1F50:	D1	POP	DE	; DE=字符串地址的指针。
1F51:	300F	JR	NC, 1F62H	; 若字符串不在字符串区中,则转移。
1F53:	2AF940	LD	HL, (40F9H)	; HL=程序终址的指针。
1F56:	DF	RST	18H	; 将字符串地址与 PST 的终址进行比较。
1F57:	3009	JR	NC, 1F62H	; 若字符串是程序中的文字,则转移。
1F59:	3ED1	LD	A, D1H	; 1F5AH: POP DE ; DE=字符串项的指针。
1F5B:	CDF529	CALL	29F5H	; 退回到先前的文字串库项。
1F5E:	EB	EX	DE, HL	; DE=字符串表区中字符串项的地址。
1F5F:	CD4328	CALL	2843H	; 将字符串送入永久的字符串区。
1F62:	CDF529	CALL	29F5H	; 文字串库表退回一项。
1F65:	E3	EX	(SP), HL	; 由堆栈取出文字串项的指针。
1F66:	CDD309	CALL	09D3H	; 把结果送到被赋值变量的单元。
1F69:	D1	POP	DE	; DE=被赋值变量的地址。
1F6A:	E1	POP	HL	; HL=代码串地址。
1F6B:	C9	RET		; 返回到调用它的程序。

地址	目的码	源程序		注　释
1F6C:	FE9E	CP	9EH	; 测试 'ERROR' 的代号。★★★ ON 子程序 ★★★★
1F6E:	2025	JR	NZ, 1F95H	; 若不是 ON ERROR，则转移。
1F70:	D7	RST	10H	; 测试输入缓冲区中的下个字符。★★★ ON ERROR ★
1F71:	CF	RST	08H	; 测试它是否是 8DH。
1F72:	8D	ADC	A, L	; 目的码 8DH 是 GOTO 的代号。
1F73:	CD5A1E	CALL	1E5AH	; 若是，则是 'GOTO'，将其后边的常数转换成二进制，结果在 DE 中。
1F76:	7A	LD	A, D	; 测试是否为 ON ERROR GOTO 0000。清除 ON ERROR 条件。
1F77:	B3	OR	E	; 将地址的 LSB 和 MSB 合并。
1F78:	2809	JR	Z, 1F83H	; 若是 GOTO 0000，则转移。
1F7A:	CD2A1B	CALL	1B2AH	; 在 BASIC 程序表中寻找行号的地址。
1F7D:	50	LD	D, B	; 把 BASIC 语句的地址送入 DE。
1F7E:	59	LD	E, C	; E=地址的 LSB。
1F7F:	E1	POP	HL	; HL=输入流中的当前位置，HL是在 1B2AH 处存入堆栈的。
1F80:	D2D91E	JP	NC, 1ED9H	; 若行号没有找到，则为 UL 错误。
1F83:	EB	EX	DE, HL	; HL=GOTO BASIC 行的地址，DE=输入流中的位置。
1F84:	22F040	LD	(40F0H), HL	; 40F0H=恢复执行的语句的地址。
1F87:	EB	EX	DE, HL	; 把代码串地址再放入 HL。
1F88:	D8	RET	C	; 若不是 GOTO 0000，则返回到执行驱动程序，否则进入下列程序。
1F89:	3AF240	LD	A, (40F2H)	; 取拒绝的错误信息。
1F8C:	B7	OR	A	; 置状态标志。
1F8D:	C8	RET	Z	; 若拒绝标志没置位，则返回到执行驱动程序。
1F8E:	3A9A40	LD	A, (409AH)	; 否则取错误代码，
1F91:	5F	LD	E, A	; 并送入 E 寄存器。
1F92:	C3AB19	JP	19ABH	; 转移到错误子程序。
1F95:	CD1C2B	CALL	2B1CH	; 把 n 值放入 DE。 ★★★★★★★★★★★★★★★
1F98:	7E	LD	A, (HL)	; A=从代码串中取入的下一个代号：ON *n* GOTO, ON *n* GOSUB。
1F99:	47	LD	B, A	; 保存代号。
1F9A:	FE91	CP	91H	; 测试 GOSUB 代号。
1F9C:	2803	JR	Z, 1FA1H	; 若是 'ON *n* GOSUB'，则转移。
1F9E:	CF	RST	08H	; 测试 GOTO 代号。
1F9F:	8D	ADC	A, L	; 目的码 8DH 是 GOTO 代号。
1FA0:	2B	DEC	HL	; 将代码串指针退回到 GOTO 代号。
1FA1:	4B	LD	C, E	; C=ON *n* 中的 *n* 值。
1FA2:	0D	DEC	C	; *n*−1（跳行的计数）。
1FA3:	78	LD	A, B	; A=GOSUB 或 GOTO 的代号。
1FA4:	CA601D	JP	Z, 1D60H	; 我们已跳了 n 行，返回到执行驱动程序。

地址	目的码	源程序		注释
1FA7:	CD5B1E	CALL	1E5BH	; 把 GOTO 的行号作为二进制数放入 DE。
1FAA:	FE2C	CP	2CH	; 寻找行号后边的逗号,否则它是语句的结束。
1FAC:	C0	RET	NZ	; 若没有逗号,则返回。
1FAD:	18F3	JR	1FA2H	; 循环,直到跳过 n 个行号。
1FAF:	11F240	LD	DE, 40F2H	; 取错误标志的地址。★★★ RESUME 子程序★★★★
1FB2:	1A	LD	A, (DE)	; 取入错误标志(若有错误则为 FFH, 否则为 0H)。
1FB3:	B7	OR	A	; 置状态标志。
1FB4:	CAA019	JP	Z, 19A0H	; 若执行 RESUME 而没有错误,则产生 RN 错误。
1FB7:	3C	INC	A	; 置错误标志为 0 。
1FB8:	329A40	LD	(409AH), A	; 把它存储起来。
1FBB:	12	LD	(DE), A	; 重新设置错误标志。
1FBC:	7E	LD	A, (HL)	; 从代码串中取入下个元素。
1FBD:	FE87	CP	87H	; 测试 NEXT 代号。
1FBF:	280C	JR	Z, 1FCDH	; 若是 'RESUME NEXT', 则转移。
1FC1:	CD5A1E	CALL	1E5AH	; 把行号的二进制值送入 DE。
1FC4:	C0	RET	NZ	; 若没有行号,则返回到执行驱动程序。
1FC5:	7A	LD	A, D	; 合并行号的 MSB 和 LSB,
1FC6:	B3	OR	E	; 测试行号是否为 0 。
1FC7:	C2C51E	JP	NZ, 1EC5H	; 若是 'RESUME ××××', 则在 GOTO 子程序继续执行。
1FCA:	3C	INC	A	; 否则是 'RESUME 0'。置 A=1, 表示是 RESUME 0。
1FCB:	1802	JR	1FCFH	; 转移到 RESUME 0 程序。
1FCD:	D7	RST	10H	; 在 RESUME NEXT 时,测试是否是多语句。
1FCE:	C0	RET	NZ	; 若是':'号,则返回到执行驱动程序,否则进入下列程序。
1FCF:	2AEE40	LD	HL, (40EEH)	; 取错误行中当前位置的地址。★★★ RESUME 0 ★★
1FD2:	EB	EX	DE, HL	; 存入 DE。
1FD3:	2AEA40	LD	HL, (40EAH)	; (40EAH)=发生错误的语句的行号,
1FD6:	22A240	LD	(40A2H), HL	; 我们就将在发生错误的语句行恢复执行。
1FD9:	EB	EX	DE, HL	; 恢复错误行中当前位置地址,以返回到执行驱动程序。
1FDA:	C0	RET	NZ	; 若是 RESUME 0, 则返回到执行驱动程序。
1FDB:	7E	LD	A, (HL)	; 否则,是 RESUME NEXT。
1FDC:	B7	OR	A	; 测试行的结束。
1FDD:	2004	JR	NZ, 1FE3H	; 若不是行的结束,则转移。
1FDF:	23	INC	HL	; 行结束了。跳过结束符——0 字节。
1FE0:	23	INC	HL	; 跳过下个
1FE1:	23	INC	HL	; 语句的指针。
1FE2:	23	INC	HL	; 跳过发生错误行的
1FE3:	23	INC	HL	; 二进制行号。
1FE4:	7A	LD	A, D	; DE=发生错误的语句的行号。
1FE5:	A3	AND	E	; 测试程序的结束。
1FE6:	3C	INC	A	; 若是程序结束,则给出 0 。

地址	目的码	源 程 序		注　　释
1FE7:	C2051F	JP	NZ, 1F05H	; 不是程序的结束,则跳到错误行的末尾,并继续。
1FEA:	3ADD40	LD	A, (40DDH)	; 取进入输入阶段的标志。
1FED:	3D	DEC	A	; 测试是否开始输入阶段。
1FEE:	CABE1D	JP	Z, 1DBEH	; 没开始,则转移到该处。
1FF1:	C3051F	JP	1F05H	; 在返回前跳到语句的末尾。
1FF4:	CD1C2B	CALL	2B1CH	; 若是 ERROR *n*, 计算 *n*。 ★★★ ERROR 子程序★★
1FF7:	C0	RET	NZ	; 若语句没结束,则返回。
1FF8:	B7	OR	A	; 置错误号状态标志。
1FF9:	CA4A1E	JP	Z, 1E4AH	; 若 n 为零,则为 FC 错误。
1FFC:	3D	DEC	A	; A=*n*−1
1FFD:	87	ADD	A, A	; A=2(*n*−1)
1FFE:	5F	LD	E, A	; 将加倍后的错误号存入 E。
1FFF:	FE2D	CP	2DH	; 与 45 进行比较。
2001:	3802	JR	C, 2005H	; 若错误号在范围内 (<45), 则转移。
2003:	1E26	LD	E, 26H	; 26H 是 UL 错误代码。
2005:	C3A219	JP	19A2H	; 输出错误信息。
2008:	110A00	LD	DE, 000AH	; 缺项起始行号为 10。 ★★★ AUTO 子程序 ★★★★
200B:	D5	PUSH	DE	; 保存起始行号。
200C:	2817	JR	Z, 2025H	; 若没有指定参数,则使用缺项值。
200E:	CD4F1E	CALL	1E4FH	; 将第一个参数从 ASCII 转换成二进制。
2011:	EB	EX	DE, HL	; 把用户指定的起始行号存入 HL。
2012:	E3	EX	(SP), HL	; 然后将它与堆栈中的 10 交换。
2013:	2811	JR	Z, 2026H	; 若只指定一个参数,则转移。
2015:	EB	EX	DE, HL	; DE=10
2016:	CF	RST	08H	; 测试第一个参数后边的逗号。
2017:	2C	INC	L	; 定义逗号的代码: 2CH——','。
2018:	EB	EX	DE, HL	; DE=当前代码语句地址。
2019:	2AE440	LD	HL, (40E4H)	; HL=以前的自动增量值。
201C:	EB	EX	DE, HL	; DE=以前的自动增量值, HL=代码串地址。
201D:	2806	JR	Z, 2025H	; 若没有第二个参数,则转移。
201F:	CD5A1E	CALL	1E5AH	; 转换第二个参数——增量值。
2022:	C29719	JP	NZ, 1997H	; 若 Z 标志复位,则为 SN 错误。
2025:	EB	EX	DE, HL	; HL=自动增量值。
2026:	7C	LD	A, H	; 测试自动增量值是否为零。
2027:	B5	OR	L	;
2028:	CA4A1E	JP	Z, 1E4AH	; 若是零,则为 FC 错误。
202B:	22E440	LD	(40E4H), HL	; 自动增量值。
202E:	32E140	LD	(40E1H), A	; 设置 BASIC 自动增量标志。
2031:	E1	POP	HL	; HL=起始行号。
2032:	22E240	LD	(40E2H), HL	; 当前输入行号。
2035:	C1	POP	BC	; 清除堆栈。

地址	目的码	源程序		注　释
2036:	C3331A	JP	1A33H	; 返回到输入阶段。
2039:	CD3723	CALL	2337H	; 计算表达式 ★★★ IF ★★★★★★★★★★★★★
203C:	7E	LD	A, (HL)	; 表达式后边的
203D:	FE2C	CP	2CH	; 元素是逗号吗?
203F:	CC781D	CALL	Z, 1D78H	; 是,则取入下一个元素。
2042:	FECA	CP	CAH	; 并测试 'THEN' 代号。
2044:	CC781D	CALL	Z, 1D78H	; 若是 'THEN' 代号,则往前跳,于是下面的后退,
2047:	2B	DEC	HL	; 将使我们定位于 'THEN' 代号处,否则定位
2048:	E5	PUSH	HL	; 于表达式后边的元素处。
2049:	CD9409	CALL	0994H	; 测试条件的真或假。
204C:	E1	POP	HL	; 恢复语句中当前位置的地址。
204D:	2807	JR	Z, 2056H	; 如果是零,则表达式为假,去寻找 ELSE 或行的结束。
204F:	D7	RST	10H	; 检测代码串中的下个元素。
2050:	DAC21E	JP	C, 1EC2H	; 若是数字,则它必须是 GOTO 的地址。
2053:	C35F1D	JP	1D5FH	; 返回到执行驱动程序去分析语句串的剩余部分。
2056:	1601	LD	D, 01H	; 扫描行结束的计数次数。★★★ IF 语句的假分支 ★
2058:	CD051F	CALL	1F05H	; 扫描行的结束。
205B:	B7	OR	A	; A=停止扫描值。
205C:	C8	RET	Z	; 若行结束,则返回到 BASIC。
205D:	D7	RST	10H	; 取下个元素,
205E:	FE95	CP	95H	; 并测试 ELSE 代号。
2060:	20F6	JR	NZ, 2058H	; 若不是 ELSE 代号,则再扫描。
2062:	15	DEC	D	; 匹配 IF 和 ELSE。
2063:	20F3	JR	NZ, 2058H	; 循环,直到通过所有的 ELSE。
2065:	18E8	JR	204FH	; 执行语句的剩余部分。
2067:	3E01	LD	A, 01H	; A=打印机设备码。★★★ LPRINT 子程序 ★★★★
2069:	329C40	LD	(409CH), A	; 存入当前设备类型的单元。
206C:	C39B20	JP	209BH	; 去分析语句的剩余部分。
206F:	CDCA41	CALL	41CAH	; DOS 出口 (JP 5A15H)。★★★ PRINT @ ★★★★★
2072:	FE40	CP	40H	; 测试下个元素是否@的代码。
2074:	2019	JR	NR, 2089H	; 若不是 PRINT @, 则转移。
2076:	CD012B	CALL	2B01H	; 计算@表达式。★★★ PRINT @ 子程序 ★★★★★
2079:	FE04	CP	04H	; A=MSB, 测试@值>1023 否。
207B:	D24A1E	JP	NC, 1E4AH	; 若@位置 >1023, 则为 FC 错误。
207E:	E5	PUSH	HL	; 堆栈=当前代码串地址。
207F:	21003C	LD	HL, 3C00H	; HL=显示区指针。
2082:	19	ADD	HL, DE	; HL=显示区首址加@位置。
2083:	222040	LD	(4020H), HL	; 把光标位置存入显示器 DCB。
2086:	7B	LD	A, E	; E=行内位置。
2087:	E63F	AND	3FH	; 使它不超过 63, 并将它作为
2089:	32A640	LD	(40A6H), A	; 修正光标的偏移量存储起来。

地址	目的码	源程序		注释
208C:	E1	POP	HL	; 恢复代码串地址(项目表的指针)。
208D:	CF	RST	08H	; 证实@表达式后的逗号 ','。
208E:	2C	INC	L	; 定义逗号的代码：2CH——','。
208F:	FE23	CP	23H	; 寻找'#'的代码。
2091:	2008	JR	NZ, 209BH	; 若不是 PRINT #,则转移。
2093:	CD8402	CALL	0284H	; 分析剩余的字符串。打开设备。★★★ PRINT# ★★
2096:	3E80	LD	A, 80H	; 设置写入盒式磁带标志。
2098:	329C40	LD	(409CH), A	; 盒式磁带标志(=−1)。
209B:	2B	DEC	HL	; 退回到输入流中的前面符号。
209C:	D7	RST	10H	; 重新检查输入流中的下个字符。
209D:	CCFE20	CALL	Z, 20FEH	; 若是零,则输出回车符(语句结束),设备换行。
20A0:	CA6921	JP	Z, 2169H	; 若是 PRINT, 则写同步字节，清除输出设备标志 (409CH),
20A3:	FEBF	CP	BFH	; 并返回到执行驱动程序。
20A5:	CABD2C	JP	Z, 2CBDH	; 若是 PRINT USING, 则转移。
20A8:	FEBC	CP	BCH	; 测试 TAB 代号。
20AA:	CA3721	JP	Z, 2137H	; 若是 PRINT TAB, 则转移,打印项目表。
20AD:	E5	PUSH	HL	; 保存输入流中的当前位置。★★★ PRINT# ★★★★
20AE:	FE2C	CP	2CH	; 测试逗号。
20B0:	CA0821	JP	Z, 2108H	; 若是逗号,取下一项。
20B3:	FE3B	CP	3BH	; 测试分号。
20B5:	CA6421	JP	Z, 2164H	; 若是分号,则转移。
20B8:	C1	POP	BC	; BC=输入流中的当前位置。
20B9:	CD3723	CALL	2337H	; 取要打印的下一项的值或地址。
20BC:	E5	PUSH	HL	; 保存结束符号的地址。
20BD:	E7	RST	20H	; 测定数据类型。
20BE:	2832	JR	Z, 20F2H	; 若是字符串,则转移。
20C0:	CDBD0F	CALL	0FBDH	; 将二进制转换成 ASCII, 并送入输出缓冲区。
20C3:	CD6528	CALL	2865H	; 建立 ASCII 数字的文字串库项目。
20C6:	CDCD41	CALL	41CDH	; DOS 出口 (JP 5B9AH)。
20C9:	2A2141	LD	HL, (4121H)	; HL=当前打印串的地址。
20CC:	3A9C40	LD	A, (409CH)	; A=输出设备标志。
20CF:	B7	OR	A	; 测试设备类型标志。
20D0:	FAE920	JP	M, 20E9H	; 若是写入盒式磁带 (PRINT#), 则转移。
20D3:	2808	JR	Z, 20DDH	; 若不是 LPRINT, 则转移。
20D5:	3A9B40	LD	A, (409BH)	; A=当前行位置。★★★ LPRINT (续) ★★★★★
20D8:	86	ADD	A, (HL)	; 加上新行中的字符数。
20D9:	FE84	CP	84H	; 并测试行溢出否。
20DB:	1809	JR	20E6H	; 去测试比较的结果。
20DD:	3A9D40	LD	A, (409DH)	; 取显示行的容量,★★★ PRINT 项目(续)★★★★★
20E0:	47	LD	B, A	; 并把它送入 B, 于是我们可以与它比较。

地址	目的码	源程序		注释
20E1:	3AA640	LD	A, (40A6H)	; 取当前行的光标偏移量。
20E4:	86	ADD	A, (HL)	; 加上新行的长度，
20E5:	B8	CP	B	; 并与最大的行容量进行比较。
20E6:	D4FE20	CALL	NC, 20FEH	; 若 C 标志复位，则新行将溢出缓冲区，跳到新的一行（换行）。
20E9:	CDAA28	CALL	28AAH	; 将行写到当前输出设备上去。★★★ PRINT #(续)★
20EC:	3E2D	LD	A, 20H	; A=ASCII 空格。
20EE:	CD2A03	CALL	032AH	; 输出空格。返回时得到非零。
20F1:	B7	OR	A	; 置状态标志。
20F2:	CCAA28	CALL	Z, 28AAH	; 如当前数据类型是字符串，则将它写到输出设备上去。
20F5:	E1	POP	HL	; 把当前代码串地址重新取入 HL。
20F6:	C39B20	JP	209BH	; 并循环，直到语句结束 (EOS)。
20F9:	3AA640	LD	A, (40A6H)	; A=从显示器的当前位置到下一行的光标偏移量。
20FC:	B7	OR	A	; 置状态标志。
20FD:	C8	RET	Z	; 如果在行的开头，则返回。
20FE:	3E0D	LD	A, 0DH	; 否则跳到下一行。
2100:	CD2A03	CALL	032AH	; 调用显示器驱动程序。
2103:	CDD041	CALL	41D0H	; DOS 出口 (JP 5B99H)。
2106:	AF	XOR	A	; 清除 A 寄存器状态标志和进位 (C) 标志。
2107:	C9	RET		; 返回到调用它的程序。
2108:	CDD341	CALL	41D3H	; DOS 出口 (JP 5B65H)。★★★写盒式磁带 ★★★★
210B:	3A9C40	LD	A, (409CH)	; 取当前输出设备标志，
210E:	B7	OR	A	; 并测试类型。
210F:	F21921	JP	P, 2119H	; 若当前设备不是盒式磁带，则转移。
2112:	3E2C	LD	A, 2CH	; A=ASCII 逗号。
2114:	CD2A03	CALL	032AH	; 在打印机上或显示器上打印逗号。
2117:	184B	JR	2164H	; 从代码串中取下个字符。
2119:	2808	JR	Z, 2123H	; 若当前设备是显示器，则转移。
211B:	3A9B40	LD	A, (409BH)	; 设备是打印机，把当前打印位置取入 A。
211E:	FE70	CP	70H	; 将打印位置与 112 比较。
2120:	C32B21	JP	212BH	; 去测试是否跳行。
2123:	3A9E40	LD	A, (409EH)	; A=行的容量。
2126:	47	LD	B, A	; 存入 B。
2127:	3AA640	LD	A, (40A6H)	; A=行中当前位置。
212A:	B8	CP	B	; 测试这一行内有无空间，从当前位置中减去行的容量。
212B:	D4FE20	CALL	NC, 20FEH	; 若没有空间，就换行。我们正处在行的末尾。
212E:	3034	JR	NC, 2164H	; 若标志行的末尾，则转移。
2130:	D610	SUB	10H	; 测试是否至少留下了 10 个打印位置。
2132:	30FC	JR	NC, 2130H	; 循环，直到位于行末的 10 个空之内。
2134:	2F	CPL		; 给出要打印的空格数(负数)。
2135:	1823	JR	215AH	; 去打印空格。

地址	目的码	源程序		注　释
2137.	CD1B2B	CALL	2B1BH	; 取 TAB 数，★★★ PRINT TAB 处理 ★★★★★★
213A:	E63F	AND	3FH	; 计算表达式，结果在 A 中。不让它超过 63。
213C:	5F	LD	E, A	; 把 TAB 值存入 E。
213D:	CF	RST	08H	; 寻找最靠近的括号。
213E:	29	ADD	HL, HL	; 定义括号的代码：29H——')'。
213F:	2B	DEC	HL	; 将代码串指针重新定位于'('号。
2140:	E5	PUSH	HL	; 将地址存入堆栈。
2141:	CDD341	CALL	41D3H	; DOS 出口（JP 5B65H）。
2144:	3A9C40	LD	A, (409CH)	; A＝输出设备类型码。
2147:	B7	OR	A	; 测试设备类型码。
2148:	FA4A1E	JP	M, 1E4AH	; 若是负的(磁带)，则产生 FC 错误。
214B:	CA5321	JP	Z, 2153H	; 若输出设备是显示器，则转移。
214E:	3A9B40	LD	A, (409BH)	; A＝当前行中的打印位置。
2151:	1803	JR	2156H	; 跳过重新装入 A 寄存器的指令。
2153:	3AA640	LD	A, (40A6H)	; A＝当前显示器行中光标位置。
2156:	2F	CPL		; A＝负的当前位置。
2157:	83	ADD	A, E	; A＝－当前位置＋TAB。
2158:	300A	JR	NC, 2164H	; 若 TAB 小于当前位置，则转移。
215A:	3C	INC	A	; A＝要打印的空格数。
215B:	47	LD	B, A	; B＝打印的空格计数。
215C:	3E20	LD	A, 20H	; A＝ASCII 空格。
215E:	CD2A03	CALL	032AH	; 打印一个空格，
2161:	05	DEC	B	; 计数减 1。
2162:	20FA	JR	NZ, 215EH	; 循环，直到打印了 B 个空格。
2164:	E1	POP	HL	; 恢复输入串中的位置。
2165:	D7	RST	10H	; 检测下个字符。
2166:	C3A020	JP	20A0H	; 处理 PRINT TAB 的剩余部分。
2169:	3A9C40	LD	A, (409CH)	; A＝设备类型码。关闭盒式磁带机，并重新设置当前设备为显示器。
216C:	B7	OR	A	; 测试是否是盒式磁带机。
216D:	FCF801	CALL	M, 01F8H	; 若是，则关闭盒式磁带机。
2170:	AF	XOR	A	; 清除 A 和状态标志。
2171:	329C40	LD	(409CH), A	; 并重新设置当前设备代码为显示器。
2174:	CDBE41	CALL	41BEH	; DOS 出口（JP 577CH）。
2177:	C9	RET		; 返回到调用它的程序。
2178:	3F	CCF		; ?——问号。★★★? REDO 错误信息 ★★★★★★★
2179:	52	LD	D, D	; R
217A:	45	LD	B, L	; E
217B:	44	LD	B, H	; D
217C:	4F	LD	C, A	; O
217D:	0D	DEC	C	; 回车。

地址	目的码	源程序		注　释
217E:	00	NOP		; 信息结束符。
217F:	3ADE40	LD	A, (40DEH)	; 取读标志。★★★ 输出读和输入错误信息★★★★★
2182:	B7	OR	A	; 设置状态标志。
2183:	C29119	JP	NZ, 1991H	; 如 READ 有效,则在 NNN 处为 SN 错误。
2186:	3AA940	LD	A, (40A9H)	; 取输入标志的类型。
2189:	B7	OR	A	; 测试零。
218A:	1E2A	LD	E, 2AH	; FD 错误码。
218C:	CAA219	JP	Z, 19A2H	; 在盒式磁带输入时,输出 FD 错误信息。
218F:	C1	POP	BC	; 清除堆栈。
2190:	217821	LD	HL, 2178H	; REDO 信息的地址。
2193:	CDA728	CALL	28A7H	; 输出 REDO 信息。
2196:	2AE640	LD	HL, (40E6H)	; 恢复代码串地址。
2199:	C9	RET		; 返回到调用它的程序。
219A:	CD2828	CALL	2828H	; 检查非法的直接命令。★★★ INPUT 子程序 ★★★
219D:	7E	LD	A, (HL)	; (没有给出行号的 INPUT 语句)
219E:	CDD641	CALL	41D6H	; DOS 出口 (JP 5784H)。
21A1:	D623	SUB	23H	; 检查磁带机的标志 # 。
21A3:	32A940	LD	(40A9H), A	; 如是 INPUT #, 则 (40A9H)=0。
21A6:	7E	LD	A, (HL)	; A=代码串的下一个元素。
21A7:	2020	JR	NZ, 21C9H	; 如是从控制台来的 INPUT, 则转移。
21A9:	CD9302	CALL	0293H	; 寻找引导码和同步字节。
21AC:	E5	PUSH	HL	; 保存代码串地址。
21AD:	06FA	LD	B, FAH	; B=待读的最大字节数(250)。
21AF:	2AA740	LD	HL, (40A7H)	; HL=输入区指针。
21B2:	CD3502	CALL	0235H	; 从磁带上读一个字节。
21B5:	77	LD	(HL), A	; 存储刚读入的字节。
21B6:	23	INC	HL	; HL 加 1, 指向缓冲区的下一个位置。
21B7:	FE0D	CP	0DH	; 读进缓冲区一直到遇到 CR 为止。
21B9:	2802	JR	Z, 21BDH	; 如遇到 CR, 则转移。
21BB:	10F5	DJNZ	21B2H	; 或者循环,一直到读入 250 个字节为止。
21BD:	2B	DEC	HL	; 定位于缓冲区的最后位置。
21BE:	3600	LD	(HL), 00H	; 在结束处放 00H。
21C0:	CDF801	CALL	01F8H	; 关闭磁带机。
21C3:	2AA740	LD	HL, (40A7H)	; 输入缓冲区地址送入 HL。
21C6:	2B	DEC	HL	; 后退一个字节。
21C7:	1822	JR	21EBH	; 在这里存入一个逗号,于是我们能用 READ 处理。
21C9:	01DB21	LD	BC, 21DBH	; 把继续运行的地址 21DBH, ★★★ INPUT 项目清单 ★★★★★★★★★★★★★★★★★★★★★★★
21CC:	C5	PUSH	BC	; 放到堆栈。
21CD:	FE22	CP	22H	; 寻找引号。
21CF:	C0	RET	NZ	; 如果在输入语句中没有提示信息,则转移到 21DBH。

地址	目的码	源程序		注　释
21D0:	CD6628	CALL	2866H	; 有引号(输入语句中的提示信息)时,为引号建立文字串库项。
21D3:	CF	RST	08H	; 寻找尾随的分号。
21D4:	3B	DEC	SP	; 给出分号的代码: 3BH——';'
21D5:	E5	PUSH	HL	; 保存代码串地址。
21D6:	CDAA28	CALL	28AAH	; 写提示信息。
21D9:	E1	POP	HL	; 恢复代码串地址。
21DA:	C9	RET		; 返回到 21DBH。
21DB:	E5	PUSH	HL	; 保存代码串地址。★★★★★★★★★★★★★★★★★
21DC:	CDB31B	CALL	1BB3H	; 显示'?' 和等待接收输入，返回时 HL=缓冲区地址减1。
21DF:	C1	POP	BC	; BC=代码串地址。
21E0:	DABE1D	JP	C, 1DBEH	; 如果按下 BREAK 键,则转移。
21E3:	23	INC	HL	; 定位于缓冲区内数据的第一个字节的位置。
21E4:	7E	LD	A, (HL)	; 取第一个数据字节。
21E5:	B7	OR	A	; 设置状态标志。
21E6:	2B	DEC	HL	; 后退一个单元到(缓冲区起点减1)处。
21E7:	C5	PUSH	BC	; 保存代码串地址。
21E8:	CA041F	JP	Z, 1F04H	; 如果第一个数据字符是二进制0，则跳到行结束处，返回 BASIC。
21EB:	362C	LD	(HL), 2CH	; 存入一个逗号,使 READ 知道已到达 DATA 语句中的一个值的结束处。
21ED:	1805	JR	21F4H	;
21EF:	E5	PUSH	HL	; 保存 PST 的当前位置。★★★ READ 子程序 ★★★
21F0:	2AFF40	LD	HL, (40FFH)	; HL=DATA 语句的开始地址。
21F3:	F6AF	OR	AFH	; 21F4H: XOR A; A 清零。若 A 为0,表示 INPUT, A 为非0,表示 READ。
21F5:	32DE40	LD	(40DEH), A	; 如果是读,则不是0 0。
21F8:	E3	EX	(SP), HL	; 如果是输入,则是0 0，HL=返回地址，堆栈=DATA 地址。
21F9:	1802	JR	21FDH	; 联接公共程序。
21FB:	CF	RST	08H	; 测试逗号。
21FC:	2C	INC	L	; 21FCH: 给出逗号的代码 2CH——','
21FD:	CD0D26	CALL	260DH	; 把当前变量的地址取入 DE。
2200:	E3	EX	(SP), HL	; 弹出指向 DATA 语句中当前单元的指针。
2201:	D5	PUSH	DE	; 用变量地址来代替它。
2202:	7E	LD	A, (HL)	; 从 DATA 语句取下一个字符。
2203:	FE2C	CP	2CH	; 测试终止逗号。
2205:	2826	JR	Z, 222DH	; 如是逗号,则转移。
2207:	3ADE40	LD	A, (40DEH)	; A=读标志。
220A:	B7	OR	A	; 测试是 READ 处理还是 INPUT 处理。

地址	目的码	源程序		注释
220B:	C29622	JP	NZ, 2296 H	; 如果是 READ, 则转移——去找下一个 DATA 语句。
220E:	3AA940	LD	A, (40A9H)	; 测试是否是 INPUT 调用中指定的设备号。
2211:	B7	OR	A	;
2212:	1E06	LD	E, 06H	; OD 错误——在调用中没给出设备号。
2214:	CAA219	JP	Z, 19A2H	; 如果没有指定磁带机,则输出 OD 信息。
2217:	3E3F	LD	A, 3FH	; 在处理 INPUT 语句时,显示'?'及数据中的错误。
2219:	CD2A03	CALL	032AH	; 显示'?'和接收输入。
221C:	CDB31B	CALL	1BB3H	; 从键盘接收输入。HL 内为缓冲区地址减 1。
221F:	D1	POP	DE	; DE=下一个变量的地址。
2220:	C1	POP	BC	; BC=代码串中下一个元素的地址。
2221:	DABE1D	JP	C, 1DBEH	; 如果按 BREAK 键,则转移。
2224:	23	INC	HL	; HL 加 1,指向缓冲区的第一个数据字节。
2225:	7E	LD	A, (HL)	; 取第一个数据字节。
2226:	B7	OR	A	; 设置状态标志。
2227:	2B	DEC	HL	; 缓冲区指针退回到缓冲区首址减 1 处。
2228:	C5	PUSH	BC	; 保存代码串地址。
2229:	CA041F	JP	Z, 1F04H	; 缓冲区内没有数据, 跳到这一行的结束处, 并返回 BASIC。
222C:	D5	PUSH	DE	; 保存变量地址。
222D:	CDDC41	CALL	41DCH	; DOS 出口 (JP 5E63H)。
2230:	E7	RST	20H	; 测试数据类型。
2231:	F5	PUSH	AF	; 保存从数据类型测试中获得的状态。
2232:	2019	JR	NZ, 224DH	; 进行数据变换:变成二进制,单精度或双精度。
2234:	D7	RST	10H	; 否则,是字符串数据。检验 DATA 语句中的下一个字符。
2235:	57	LD	D, A	; 把该字符保存在 B, D 中。
2236:	47	LD	B,A	;
2237:	FE22	CP	22H	; 测试是否为引号。
2239:	2805	JR	Z, 2240H	; 如果是一个引号——表明是字符串,则转移,
223B:	163A	LD	D, 3AH	; 否则扫描 DATA 语句,寻找一个:号或者逗号,
223D:	062C	LD	B, 2CH	; 并为它建立一个文字串库项。
223F:	2B	DEC	HL	;
2240:	CD6928	CALL	2869H	; 为 DATA 字符串建立一个文字串库项。
2243;	F1	POP	AF	; A=最终数据类型的标志。
2244:	EB	EX	DE, HL	; 保存 HL。
2245:	215A22	LD	HL, 225AH	; 把继续运行的地址 225AH 推入堆栈。
2248:	E3	EX	(SP), HL	; 清除堆栈。
2249:	D5	PUSH	DE	; 保存变量地址。
224A:	C3331F	JP	1F33H	; 把结果传送到目标变量,在 225AH 处继续执行。
224D:	D7	RST	10H	; 检验 DATA 语句中的下一个字符,并把 DATA 中的下一个值从 ASCII 码变换成二进制。★★★★★★★★

地址	目的码	源程序		注　释
224E:	F1	POP	AF	; 取回根据数据类型测试得到的标志，
224F:	F5	PUSH	AF	; 并重新保存起来。把返回地址 2243H 推入堆栈，
2250:	014322	LD	BC, 2243H	; 以便返回到下一个 DATA 变换。
2253:	C5	PUSH	BC	; 把 2243H 放入堆栈。
2254:	DA6C0E	JP	C, 0E6CH	; 把 ASCII 码变换成二进制——不是双精度。
2257:	D2650E	JP	NC, 0E65H	; 把 ASCII 码变换成二进制——双精度。
225A:	2B	DEC	HL	; 在 DATA 语句中后退一个字符。★★★★★★★★★
225B:	D7	RST	10H	; 检验终止字符。
225C:	2805	JR	Z, 2263H	; 如是行的结束，则转移。
225E:	FE2C	CP	2CH	; 如不是行的结束，测试是否为逗号。
2260:	C27F21	JP	NZ, 217FH	; 如不是逗号，输出错误信息。
2263:	E3	EX	(SP), HL	; HL=READ 语句中的下一个字节，堆栈内为 DATA 语句中的下一个字节。
2264:	2B	DEC	HL	; 向后退一个字符，
2265:	D7	RST	10H	; 且重新对它检验。
2266:	C2FB21	JP	NZ, 21FBH	; 如不是 0，它必定是一个逗号，去处理下一个变量。
2269:	D1	POP	DE	; 清除堆栈。
266A:	3AA940	LD	A, (40A9H)	; 检查 FD 错误。
226D:	B7	OR	A	; 设置状态标志。
226E:	C8	RET	Z	; 没有错误，返回到 BASIC。
226F:	3ADE40	LD	A, (40DEH)	; 取 READ/INPUT 标志。
2272:	B7	OR	A	; 设置状态标志。
2273:	EB	EX	DE, HL	; DE=代码串地址。
2274:	C2961D	JP	NZ, 1D96H	; 如果 READ 错误，则转移。
2277:	D5	PUSH	DE	; 保存代码串地址。测试 INPUT 错误。
2278:	CDDF41	CALL	41DFH	; DOS 出口 (JP 579CH)。
227B:	B6	OR	(HL)	; 测试是否 INPUT 的结束。
227C:	218622	LD	HL, 2286H	; EXTRA IGNORED 信息
227F:	C4A728	CALL	NZ, 28A7H	; 如不是 INPUT 的结束，输出信息。
2282:	E1	POP	HL	; 恢复代码串的地址。
2283:	C36921	JP	2169H	; 关闭盒式磁带机，恢复视频显示，并返回 BASIC。
2286:	3F	CCF		; EXTRA IGNORED 信息。★★★★★★★★★★★★
2287:	45	LD	B, L	; E
2288:	78	LD	A, B	; X
2289:	74	LD	(HL), H	; T
228A:	72	LD	(HL), D	; R
228B:	61	LD	H, C	; A
228C:	2069	JR	NZ, 22F7H	; 空格，I
228E:	67	LD	H, A	; G
228F:	6E	LD	L, (HL)	; N
2290:	6F	LD	L, A	; O

地址	目的码	源程序		注释
2291:	72	LD	(HL), D	; R
2292:	65	LD	H, L	; E
2293:	64	LD	H, H	; D
2294:	0D	DEC	C	; 回车。
2295:	00	NOP		; 信息终止符。
2296:	CD051F	CALL	1F05H	; 搜索下一个 DATA 语句。★★★ 调用 DATA ★★★
2299:	B7	OR	A	; 扫描到当前 DATA 行的结束。
229A:	2012	JR	NZ, 22AEH	; 如用:号结束这一行,则转移。
229C:	23	INC	HL	; 跳到下一个 BASIC 语句的地址。
229D:	7E	LD	A, (HL)	; 取下一个语句的地址。
229E:	23	INC	HL	;
229F:	B6	OR	(HL)	; 如果它是 0 ,则已达程序的结束。
22A0:	1E06	LD	E, 06H	; 如在找到下一个 DATA 语句以前,达到了程序的终点,
22A2:	CAA219	JP	Z, 19A2H	; 则给出 OD 错误。
22A5:	23	INC	HL	; HL 加1,指向这行的行号,
22A6:	5E	LD	E, (HL)	; 并把它装入 DE。
22A7:	23	INC	HL	; HL 加 1, 指向行号的 MSB。
22A8:	56	LD	D, (HL)	; DE=该语句的二进制行号。
22A9:	EB	EX	DE, HL	; HL=DATA 语句的代码串。
22AA:	22DA40	LD	(40DAH), HL	; 保存 DATA 语句的二进制行号。
22AD:	EB	EX	DE, HL	; 把 BASIC 语句地址重新放到 HL 内。
22AE:	D7	RST	10H	; 检验下一个代号。
22AF:	FE88	CP	88H	; 测试是否为 DATA 的代号。
22B1:	20E3	JR	NZ, 2296H	; 如果不是 DATA 代号,则转移,继续寻找,直到有 DATA 代号或程序结束。
22B3:	C32D22	JP	222DH	; 找下一个 DATA 语句。
22B6:	110000	LD	DE, 0000H	; 在没有指定索引的情况下, 继续循环。★★★ NEXT 子程序 ★★★★★★★★★★★★★★★★★★★★
22B9:	C40D26	CALL	NZ, 260DH	; 如给出索引,把它的地址放到 DE。
22BC:	22DF40	LD	(40DFH), HL	; 保存当前代码串的地址。
22BF:	CD3619	CALL	1936H	; 在堆栈内寻找推入的 FOR, 返回时 HL = 类型 (调整后的)/符号标志的堆栈地址。
22C2:	C29D19	JP	NZ, 199DH	; 如果没有 FOR, 则 NF 错误。
22C5:	F9	LD	SP, HL	; 把堆栈指针设置为 STEP 值的类型/符号的地址。
22C6:	22E840	LD	(40E8H), HL	; 把 CSP (当前堆栈指针)保存于 40E8H 中。
22C9:	D5	PUSH	DE	; 保存索引地址,重写 FOR 索引的地址。
22CA:	7E	LD	A, (HL)	; A=增量的符号标志。
22CB:	23	INC	HL	; 跳过调整过的类型标志。
22CC:	F5	PUSH	AF	; 保存符号标志。
22CD:	D5	PUSH	DE	; DE=索引的地址。
22CE:	7E	LD	A, (HL)	; A=调整过的STEP 增量的类型标志,如是单精度为+1,

地址	目的码	源程序		注　释
				而整数则为 −1。
22CF:	23	INC	HL	; 后退到 STEP 增量的结束。
22D0:	B7	OR	A	; 测试 STEP 增量的调整后的类型标志。
22D1:	FAEA22	JP	M, 22EAH	; 如是整数类型,则转移。
22D4:	CDB109	CALL	09B1H	; 装入从堆栈取来的 STEP 增量。作为当前值保存起来。
22D7:	E3	EX	(SP), HL	; HL=索引地址。堆栈=TO 终值的结束地址。
22D8:	E5	PUSH	HL	; 保存索引地址。
22D9:	CD0B07	CALL	070BH	; 把索引装进 BC 和 DE 并加到当前变量上。
22DC:	E1	POP	HL	; 把索引地址重新放入 HL。
22DD:	CDCB09	CALL	09CBH	; 把当前值(新的索引)传送到它的地址。
22E0:	E1	POP	HL	; HL=TO 终值的结束地址。
22E1:	CDC209	CALL	09C2H	; 把 TO 值装进 BC 和 DE。
22E4:	E5	PUSH	HL	; 保存指向 FOR 语句的二进制行号的指针地址。
22E5:	CD0C0A	CALL	0A0CH	; 把 BC 和 DE 中的 TO 值与当前值新的索引作比较。
22E8:	1829	JR	2313H	; 对比较结果进行检验。
22EA:	23	INC	HL	; 堆栈后退 4 个字节——跳过单精度 TO 值这个区。★
22EB:	23	INC	HL	;
22EC:	23	INC	HL	;
22ED:	23	INC	HL	; 准备取一个整数增量。
22EE:	4E	LD	C, (HL)	; C=增量的 LSB。
22EF:	23	INC	HL	; HL 加 1, 指向 MSB。
22F0:	46	LD	B, (HL)	; B=增量的 MSB。
22F1:	23	INC	HL	; HL=TO 终值的堆栈地址。
22F2:	E3	EX	(SP), HL	; HL=索引的地址。堆栈=TO 终值的结束地址。
22F3:	5E	LD	E, (HL)	; E=索引的 LSB。
22F4:	23	INC	HL	; HL 加 1, 指向 MSB。
22F5:	56	LD	D, (HL)	; D=索引的 MSB。
22F6:	E5	PUSH	HL	; 保存索引 MSB 的地址。
22F7:	69	LD	L, C	; L=增量的 LSB。
22F8:	60	LD	H, B	; H=增量的 MSB。
22F9:	CDD20B	CALL	0BD2H	; 把 DE 中的值加到 HL。如果是整数, 如放在 HL 中, 即索引加增量。
22FC:	3AAF40	LD	A, (40AFH)	; 取数据类型标志。
22FF:	FE04	CP	04H	; 测试是否为单精度。
2301:	CAB207	JP	Z, 07B2H	; 如是单精度,则为 OV 错误。
2304:	EB	EX	DE, HL	; DE=新的索引值。
2305:	E1	POP	HL	; HL=变量区中的索引地址。
2306:	72	LD	(HL), D	; 保存新的索引的 MSB。
2307:	2B	DEC	HL	; 往下跳到 LSB。
2308:	73	LD	(HL), E	; 保存新索引的 LSB。
2309:	E1	POP	HL	; HL=在 FOR 语句中 TO 值的地址。

地址	目的码	源程序		注　释
230A:	D5	PUSH	DE	; 保存新的索引。
230B:	5E	LD	E, (HL)	; E=TO 值的 LSB。
230C:	23	INC	HL	; HL 加 1，指向 MSB。
230D:	56	LD	D, (HL)	; D=TO 值的 MSB。
230E:	23	INC	HL	; HL 加 1，指向行号的地址。
230F:	E3	EX	(SP), HL	; HL=TO 值，把行号地址存在堆栈中。
2310:	CD390A	CALL	0A39H	; 把新的索引与终值比较。
2313:	E1	POP	HL	; HL=FOR 语句二进制行号的地址。
2314:	C1	PQP	BC	; BC=索引的符号标志。
2315:	90	SUB	A, B	; 把比较的符号与要求的符号作比较。
2316:	CDC209	CALL	09C2H	; 装入 BC，BC=循环中第一语句的地址。 DE=FOR 语句的二进制行号。
2319:	2809	JR	Z, 2324H	; 如索引< >终值，则转移。
231B:	EB	EX	DE, HL	; HL=FOR 语句的二进制行号。
231C:	22A240	LD	(40A2H), HL	; 保存 FOR 语句的行号。
231F:	69	LD	L, C	; 传送第一循环语句的 LSB。
2320:	60	LD	H, B	; 传送第一循环语句的 MSB。
2321:	C31A1D	JP	1D1AH	; 继续执行。恢复 FOR 代号和 FOR 语句的 GAP。
2324:	F9	LD	SP, HL	; 恢复堆栈指针，★★★表达式计算的开始★★★★★
2325:	22E840	LD	(40E8H), HL	; 并保存在 40E8H。
2328:	2ADF40	LD	HL, (40DFH)	; HL=NEXT I 以后的代码串地址。
232B:	7E	LD	A, (HL)	; 取下一个代号。
232C:	FE2C	CP	2CH	; 与逗号比较。
232E:	C21E1D	JP	NZ, 1D1EH	; 如不是逗号，则转移。
2331:	D7	RST	10H	; 定位到下一个索引。
2332:	CDB922	CALL	22B9H	; 再进入和执行 NEXT。
2335:	CF	RST	08H	; 测试输入语句中的左括号。
2336:	282B	JR	Z, 2363H	; 2336H：给出左括号的代码 28H——'('
2338:	1600	LD	D, 00H	; 2337H：DEC HL
233A:	D5	PUSH	DE	; D=优先值，E=算符代号。
233B:	0E01	LD	C, 01H	; 需要的空闲存储器字节数。
233D:	CD6319	CALL	1963H	; 检查空闲存储器的范围。
2340:	CD9F24	CALL	249FH	; 取表达式中下一个元素的值。 如是变量：地址放到 4121H。 如是常数：值放到 4127H。
2343:	22F340	LD	(40F3H), HL	; 下一个代号的地址。
2346:	2AF340	LD	HL, (40F3H)	; 处理后的再入点。
2349:	C1	POP	BC	; BC = DE = 优先值（最后一个操作数/最后一个算符代号）。

E	0	1	2	3	4	5	6
代号	+	−	*	/	@@	AND	OR

地址	目的码	源程序		注　释
234A:	7E	LD	A, (HL)	; 取下一个代号(算符或函数)。
234B:	1600	LD	D, 00H	; 清除遇到的关系代号标志。
234D:	D6D4	SUB	D4H	; 测试是算术算符还是关系算符。
234F:	3813	JR	C, 2364H	; 算符+,−,*,/,↑, AND, OR。
2351:	FE03	CP	03H	; 测试是否为>,=,<代号。
2353:	300F	JR	NC, 2364H	; 如是 SGN 到 MID$ 的代号,则转移。
2355:	FE01	CP	01H	; 如是>, C 标志置位。测试<=,>=。
2357:	17	RLA		; 调整后的代号给出 1(>), 2(=), 4(<)。
2358:	AA	XOR	D	; 前面的调整后的代号与当前的调整后的代号合并。
2359:	BA	CP	D	; 测试是否为允许的合并<=,=>。
235A:	57	LD	D, A	; 合并必须比前面的值要大。

说明

	<	=	>
	4	2	1
<4	0	6	5
=2	6	0	3
>1	5	3	0

关系表

235B:	DA9719	JP	C,1997H	; 如是<<,>>或==,则错误。
235E:	22D840	LD	(40D8H), HL	; <,=或>代号的地址放到 40D8H 内。
2361:	D7	RST	10H	; 取下一个代号。
2362:	18E9	JR	234DH	; 把两个关系算符作为一个来处理。
2364:	7A	LD	A, D	; 取关系算符标志。
2365:	B7	OR	A	; 然后设置状态标志。
2366:	C2EC23	JP	NZ, 23ECH	; 如果先碰到<,=,>代号,则转移。
2369:	7E	LD	A, (HL)	; A=算符代号。
236A:	22D840	LD	(40D8H), HL	; 算术算符的地址放到 40D8H 单元。
236D:	D6CD	SUB	CDH	; 测试是否为算术算符代号。
236F:	D8	RET	C	; 如果不是算术算符代号,则返回。
2370:	FE07	CP	07H	; 测试是否为+到 OR 的代号。
2372:	D0	RET	NC	; 如果是>到 MID$ 的代号,则返回。
2373:	5F	LD	E, A	; E=0—7。
2374:	3AAF40	LD	A, (40AFH)	; 取当前变量的类型标志。
2377:	D603	SUB	03H	; −1(整数), 0(字符串), 1(单精度), 5(双精度)。
2379:	B3	OR	E	; 把算符代号和类型合并起来。
237A:	CA8F29	JP	Z, 298FH	; 于是就能测试是否字符串相加。
237D:	219A18	LD	HL, 189AH	; 优先算符值的表。
2380:	19	ADD	HL, DE	; 加上本代号(0—7)。
2381:	78	LD	A, B	; 计算这个算符的地址。
2382:	56	LD	D, (HL)	; 得到上一个算符的优先值。
2383:	BA	CP	D	; 得到这一个算符的优先值。

地址	目的码	源程序		注　释
2384:	D0	RET	NC	; 如果这个算符的优先值比上一个算符和上个算符代号的优先值级别高，则返回。
2385:	C5	PUSH	BC	;
2386:	014623	LD	BC, 2346H	; 在优先断点情况下的继续地址。
2389:	C5	PUSH	BC	; 推入堆栈。
238A:	7A	LD	A, D	; A=这个算符的优先值。
238B:	FE7F	CP	7FH	; 测试是否是幂运算。
238D:	CAD423	JP	Z, 23D4H	; 如果是幂运算，则转移。
2390:	FE51	CP	51H	; 测试是否是逻辑算符。
2392:	DAE123	JP	C, 23E1H	; 如是 AND 或 OR，则转移。
2395:	212141	LD	HL, 4121H	; HL=第一个操作数的二进制值的地址。
2398:	B7	OR	A	; 清除状态标志。
2399:	3AAF40	LD	A, (40AFH)	; 取数据类型。 −1（整数），0（字符串），1（单精度），5（双精度）。
239C:	3D	DEC	A	; 减 1。
239D:	3D	DEC	A	; 减 2。
239E:	3D	DEC	A	; 减 3。
239F:	CAF60A	JP	Z, 0AF6H	; 如 Z 标志置位(即字符串)，则是 TM 错误。
23A2:	4E	LD	C, (HL)	; 现在，装入算符的二进制值。C=值的 LSB。
23A3:	23	INC	HL	; HL 加 1，指向 MSB。
23A4:	46	LD	B, (HL)	; BC=二进制值。
23A5:	C5	PUSH	BC	; 保存二进制值。
23A6:	FAC523	JP	M, 23C5H	; 如果存的是整型操作数，则转移。
23A9:	23	INC	HL	; 否则，取值的剩余部分，
23AA:	4E	LD	C, (HL)	; 放进 BC，同时也把它放入堆栈。
23AB:	23	INC	HL	; C=单精度值的 MSB。
23AC:	46	LD	B, (HL)	; B=单精度值的指数。
23AD:	C5	PUSH	BC	; 保存数字的剩余部分。
23AE:	F5	PUSH	AF	; 保存类型(−3)。
23AF:	B7	OR	A	; 清除状态标志，于是能测试是否为双精度值。
23B0:	E2C423	JP	PO, 23C4H	; 如不是双精度，则转移。
23B3:	F1	POP	AF	; 清除堆栈。
23B4:	23	INC	HL	; 指向值的下一个字节。
23B5:	3803	JR	C, 23BAH	; 如果值的剩余部分不在 WRA1 中，则转移。
23B7:	211D41	LD	HL, 411DH	; 恢复 HL 为 WRA1 的起点。
23BA:	4E	LD	C, (HL)	; 装入双精度值的剩余部分，
23BB:	23	INC	HL	; 并保存在堆栈中。
23BC:	46	LD	B, (HL)	; B=下一个 LSB。
23BD:	23	INC	HL	; 指向下一个数字。
23BE:	C5	PUSH	BC	; 保存双精度值的 LSB 和 NMSB。
23BF:	4E	LD	C, (HL)	; 然后把双精度值的中间字节装入 BC，

地址	目的码	源程序		注　释
23C0:	23	INC	HL	;
23C1:	46	LD	B, (HL)	;
23C2:	C5	PUSH	BC	; 并保存在堆栈中。
23C3:	06F1	LD	B, F1H	; 23C4H: POP AF 清除堆栈内的类型(−3)和状态。
23C5:	C603	ADD	A, 03H	; A=类型。
23C7:	4B	LD	C, E	; 算术算符(0—7)的代号。
23C8:	47	LD	B, A	; 加操作数的长度。
23C9:	C5	PUSH	BC	; 操作数放入堆栈。
23CA:	010624	LD	BC, 2406H	; 重排操作优先级的地址。
23CD:	C5	PUSH	BC	; 放入堆栈。
23CE:	2AD840	LD	HL, (40D8H)	; 恢复 HL 为所遇到的最后一个代号的地址。
23D1:	C33A23	JP	233AH	; 注意，D=优先值，E=算符值(0—7)。
23D4:	CDB10A	CALL	0AB1H	; 把整数(4121H—4122H)变换为单精度数，并存入4121H—4124H。
23D7:	CDA409	CALL	09A4H	; 把单精度数从 4121H—4124H 传送到堆栈。
23DA:	01F213	LD	BC, 13F2H	; 单精度指数的子程序的地址。
23DD:	167F	LD	D, 7FH	; D=↑的优先值。
23DF:	18EC	JR	23CDH	; 继续指数计算。
23E1:	D5	PUSH	DE	; 保存优先值代号。 ★★★★★★★★★★★★★★★
23E2:	CD7F0A	CALL	0A7FH	; 把当前值变换为整数，留在 HL 内。
23E5:	D1	POP	DE	; 恢复优先值代号。
23E6:	E5	PUSH	HL	; 保存当前值(整数)。
23E7:	01E925	LD	BC, 25E9H	; 逻辑算符子程序地址。
23EA:	18E1	JR	23CDH	; 继续句法分析。
23EC:	78	LD	A, B	; A=先前算符的优先值。★★★关系代号子程序★★★
23ED:	FE64	CP	64H	; 测试是否为关系算符。
23EF:	D0	RET	NC	; 如先前的算符是关系算符，则返回。
23F0:	C5	PUSH	BC	; B=前一个算符的优先值，C=代号。
23F1:	D5	PUSH	DE	; D=6，5 或 3，E=代号。
23F2:	110464	LD	DE, 6404H	; D='<=', '>=' 的优先值，E=代号(关系符组合形式)。
23F5:	21B825	LD	HL, 25B8H	; 比较逻辑量的程序的地址，
23F8:	E5	PUSH	HL	; 并放入堆栈。
23F9:	E7	RST	20H	; 测试数据类型。
23FA:	C29523	JP	NZ, 2395H	; 如不是字符串，则转移。
23FD:	2A2141	LD	HL, (4121H)	; HL=字符串地址。把变量放于堆栈。
2400:	E5	PUSH	HL	; 保存字符串地址于堆栈。
2401:	018C25	LD	BC, 258CH	; BC=字符串比较子程序的地址。
2404:	18C7	JR	23CDH	; 把 BC 中的地址保存在堆栈中。继续分析语句。
2406:	C1	POP	BC	; B=数据类型，C=算术算符代号(0—7)。★★★语句结束或优先断点 ★★★★★★★★★★★★★★★

地址	目的码	源程序		注释
2407:	79	LD	A, C	; A=C=代号。
2408:	32B040	LD	(40B0H), A	; (40B0H)=最后一个操作数的算术算符代号(待执行的这一个)。
240B:	78	LD	A, B	; 第一个操作数的数据类型。
240C:	FE08	CP	08H	; 测试第一个操作数的数据类型。
240E:	2828	JR	Z, 2438H	; 如第一个操作数是双精度数,则转移。
2410:	3AAF40	LD	A, (40AFH)	; 不是,则测试当前操作数。
2413:	FE08	CP	08H	; 测试数据类型。
2415:	CA6024	JP	Z, 2460H	; 如是双精度数,则转移。
2418:	57	LD	D, A	; D=当前操作数的数据类型。
2419:	78	LD	A, B	; A=第一个操作数的数据类型。
241A:	FE04	CP	04H	; 测试当前操作数数据类型。
241C:	CA7224	JP	Z, 2472H	; 如第一个操作数是单精度数,则转移。
241F:	7A	LD	A, D	; A=当前操作数的数据类型。
2420:	FE03	CP	03H	; 是 CR 串变量吗?
2422:	CAF60A	JP	Z, 0AF6H	; 如是串变量,则为 TM 错误。
2425:	D27C24	JP	NC, 247CH	; 如是单精度,则转移,否则为整数。
2428:	21BF18	LD	HL, 18BFH	; 计算算术子程序的地址。
242B:	0600	LD	B, 00H	; 算术算符代号乘 2 ,加上算术子程序表的首址,
242D:	09	ADD	HL, BC	; 给出包含有算术子程序地址的存储单元地址。
242E:	09	ADD	HL, BC	;
242F:	4E	LD	C, (HL)	; 整数算术子程序的地址。C=LSB。
2430:	23	INC	HL	; HL 加1,指向地址的下一个单元。
2431:	46	LD	B, (HL)	; B=算术子程序的地址的 MSB。
2432:	D1	POP	DE	; DE=第一个操作数的值。
2433:	2A2141	LD	HL, (4121H)	; HL=当前操作数的值。
2436:	C5	PUSH	BC	; 保存算术子程序地址到堆栈中,用于后面的 POP 指令。
2437:	C9	RET		; 返回算术子程序。
2438:	CDDB0A	CALL	0ADBH	; 把当前值变换为双精度值。
243B:	CDFC09	CALL	09FCH	; 把当前值变换为单精度值。 ★★★★★★★★★★★
243E:	E1	POP	HL	; 把当前值送到 WRA1 的 411FH 和 4120H。
243F:	221F41	LD	(411FH), HL	;
2442:	E1	POP	HL	; HL=双精度值的 NMSB,
2443:	221D41	LD	(411DH), HL	; 并放到 WRA1 的 411DH 和 411EH。
2446:	C1	POP	BC	; BC=双精度值的剩余部分。
2447:	D1	POP	DE	; DE=双精度值的剩余部分。
2448:	CDB409	CALL	09B4H	; 把 BC 和 DE 存入 WRA1 的高端部分, 把 DE 传送到 4121H 单元, BC 传送到 4123H 单元。
244B:	CDDB0A	CALL	0ADBH	; 把第一个值变换为双精度值。
244E:	21AB18	LD	HL, 18ABH	; 双精度子程序的基地址。
2451:	3AB040	LD	A, (40B0H)	; 取代号值。用它来计算算术子程序的地址。

地址	目的码	源程序		注释
2454:	07	RLCA		; 代号×2。
2455:	C5	PUSH	BC	; 保存 BC 值,于是我们就能用它作 16 位算术运算。
2456:	4F	LD	C, A	; C=2×代号。
2457:	0600	LD	B, 00H	; B=0
2459:	09	ADD	HL, BC	; (代号值×2)×18ABH=算术子程序的表地址。
245A:	C1	POP	BC	; 恢复 BC。
245B:	7E	LD	A, (HL)	; 装入算术子程序地址的 LSB。
245C:	23	INC	HL	; HL 加 1, 指向 MSB。
245D:	66	LD	H, (HL)	; 把算术子程序地址的 MSB 装入 HL。
245E:	6F	LD	L, A	; HL=算术子程序的地址。
245F:	E9	JP	(HL)	; 转移到算术子程序。返回 2346H。
2460:	C5	PUSH	BC	; 保存第一个操作数数据类型/算术算符代号★★★第一个操作数不是双精度,第二个操作数是双精度★★★★
2461:	CDFC09	CALL	09FCH	; 把当前的值移到存放区。存放区指的是 WAR2。
2464:	F1	POP	AF	; A=另一个操作数的数据类型。
2465:	32AF40	LD	(40AFH), A	; 把 A 的内容存起来,
2468:	FE04	CP	04H	; 并测试它是否为单精度。
246A:	28DA	JR	Z, 2446H	; 如是单精度,则转移,把数值变换成双精度,并作运算。
246C:	E1	POP	HL	; 值必定是整数。将它从堆栈弹出,
246D:	222141	LD	(4121H), HL	; 然后作为当前值存起来,
2470:	18D9	JR	244BH	; 把它变换成双精度,并执行运算。
2472:	CDB10A	CALL	0AB1H	; 把当前操作数变换成单精度数。 ★★★★★★★★★
2475:	C1	POP	BC	; 把现在的算符放到 BC,
2476:	D1	POP	DE	; 放到 DE。
2477:	21B518	LD	HL, 18B5H	; 单精度算术子程序的基地址。
247A:	18D5	JR	2451H	; 去执行运算。
247C:	E1	POP	HL	; 把整型操作数装进 HL。 ★★★ 第一个操作数是单精度,第二个操作数是整数★★★★★★★★★★★★★
247D:	CDA409	CALL	09A4H	; 把当前单精度值放入堆栈。
2480:	CDCF0A	CALL	0ACFH	; 把 HL 中的整数值变换成单精度值。
2483:	CDBF09	CALL	09BFH	; 把由整数值变换成的单精度值装入 BC 和 DE。
2486:	E1	POP	HL	; 弹出堆栈中的单精度值, H=指数, L=MSB。
2487:	222341	LD	(4123H), HL	; 作为当前值的指数和 MSB。
248A:	E1	POP	HL	; 弹出单精度值, H=NMSB, L=LSB。
248B:	222141	LD	(4121H), HL	; 作为当前值的 NMSB 和 LSB。
248E:	18E7	JR	2477H	; 去执行运算。
2490:	E5	PUSH	HL	; 保存 HL, 供以后变换用。★★★ 整数除 ★★★★★ 把 HL 和 DE 变换成单精度,用单精度除。
2491:	EB	EX	DE, HL	; 准备把 DE 变换为单精度。
2492:	CDCF0A	CALL	0ACFH	; 把 DE 变换为单精度。
2495:	E1	POP	HL	; 恢复原来的 HL。

地址	目的码	源程序		注释
2496:	CDA409	CALL	09A4H	; 把变换过的 DE 移到堆栈。
2499:	CDCF0A	CALL	0ACFH	; 把 HL 变换为单精度。
249C:	C3A008	JP	08A0H	; 去执行单精度除。
249F:	D7	RST	10H	; 测试下一个符号。 ★★★★★★★★★★★★★★★★
24A0:	1E28	LD	E, 28H	; 如果是字符串的结束,则为 MO 错误。
24A2:	CAA219	JP	Z, 19A2H	; 如 Z 标志置位,则转移到 19A2H。
24A5:	DA6C0E	JP	C, 0E6CH	; 如是数字,则转移——把 ASCII 码变成二进制值。
24A8:	CD3D1E	CALL	1E3DH	; 检验是否是字母。
24AB:	D24025	JP	NC, 2540H	; 如是字母,则转移。
24AE:	FECD	CP	CDH	; 测试是否是+的代号。
24B0:	28ED	JR	Z, 249FH	; 如是+的代号,则转移——寻址下一个数字。
24B2:	FE2E	CP	2EH	; 测试是否是小数点。
24B4:	CA6C0E	JP	Z, 0E6CH	; 如是小数点,则转移。
24B7:	FECE	CP	CEH	; 测试是否是一的代号。
24B9:	CA3225	JP	Z, 2532H	; 如是一的代号,则转移。
24BC:	FE22	CP	22H	; 测试是否为引号。
24BE:	CA6628	JP	Z, 2866H	; 如是引号,则转移——建立一个文字串指针项。
24C1:	FECB	CP	CBH	; 测试是否为 NOT 的代号。
24C3:	CAC425	JP	Z, 25C4H	; 如是 NOT 的代号,则转移。
24C6:	FE26	CP	26H	; 测试是否为 & 的 ASCII码。
24C8:	CA9441	JP	Z, 4194H	; 如是 &, 则转移。
24CB:	FEC3	CP	C3H	; 测试是否为 ERR 的代号。
24CD:	200A	JR	NZ, 24D9H	; 如不是 ERR 的代号,则转移。
24CF:	D7	RST	10H	; 定位于代码串中的下一个元素。
24D0:	3A9A40	LD	A, (409AH)	; 取当前的错误编号。
24D3:	E5	PUSH	HL	; 保存当前代码串的地址。
24D4:	CDF827	CALL	27F8H	; 把错误编号作为当前值(整数),保存起来。
24D7:	E1	POP	HL	; 恢复代码串地址。
24D8:	C9	RET		; 返回到表达式计算。
24D9:	FEC2	CP	C2H	; 测试是否是 ERL 的代号。 ★★★★★★★★★★★
24DB:	200A	JR	NZ, 24E7H	; 如果不是 ERL 的代号,则转移。
24DD:	D7	RST	10H	; 定位于代码串中的下一个元素。
24DE:	E5	PUSH	HL	; 保存当前代码串的地址。
24DF:	2AEA40	LD	HL, (40EAH)	; 取有错误的行号。
24E2:	CD660C	CALL	0C66H	; 把行号变换成单精度值,并作为当前值保存起来。
24E5:	E1	POP	HL	; 恢复代码串地址。
24E6:	C9	RET		; 返回到表达式计算。
24E7:	FEC0	CP	C0H	; 测试是否是 VARPTR 的代号。 ★★★ VARPTR ★★
24E9:	2014	JR	NZ, 24FFH	; 如果不是 VARPTR 的代号,则转移。
24EB:	D7	RST	10H	; 从代码串得到下一个字符。
24EC:	CF	RST	08H	; 测试下一个字符是否是左括号,并跳过它。

地址	目的码	源 程 序		注 释
24ED:	28CD	JR	Z, 24BCH	; 24EDH: 给出左括号的值 28H——'('
24EF:	0D	DEC	C	; 24EEH: CALL 260DH——测试变量名。
24F0:	26CF	LD	H, CFH	; 24F1H: RST 08——测试下一个字符是否是右括号，并跳过它。
24F2:	29	ADD	HL, HL	; 24F2H:给出右括号的值29 H——')'
24F3:	E5	PUSH	HL	; 保存当前代码串地址。
24F4:	EB	EX	DE, HL	; 把变量地址移到 HL。
24F5:	7C	LD	A, H	; 然后测试是否为 0 地址(非定义变量)。
24F6:	B5	OR	L	; 把地址的 LSB 和 MSB 合并起来。
24F7:	CA4A1E	JP	Z, 1E4AH	; 如变量没有被定义过，则是 FC 错误。
24FA:	CD9A0A	CALL	0A9AH	; 把地址作为当前变量保存起来，设定类型为整数。
24FD:	E1	POP	HL	; 恢复当前代码串地址。
24FE:	C9	RET		; 返回到执行驱动程序。
24FF:	FEC1	CP	C1H	; 测试是否是 USR 的代号。 ★★★★★★★★★★★
2501:	CAFE27	JP	Z, 27FEH	; 如是 USR 的代号，则转移。
2504:	FEC5	CP	C5H	; 测试是否是 INSTR 的代号。
2506:	CA9D41	JP	Z, 419DH	; 如是 INSTR, 则转移: 磁盘 BASIC 用 (JP 582FH)。
2509:	FEC8	CP	C8H	; 测试是否是 MEM 的代号。
250B:	CAC927	JP	Z, 27C9H	; 如是 MEM 的代号，则转移。
250E:	FEC7	CP	C7H	; 测试是否是 TIME$ 的代号。
2510:	CA7641	JP	Z,4176H	; 如是 TIME$ 的代号，则转移。
2513:	FEC6	CP	C6H	; 测试是否是 POINT 的代号。
2515:	CA3201	JP	Z, 0132H	; 如是 POINT 的代号，则转移。
2518:	FEC9	CP	C9H	; 测试是否是 INKEY$ 的代号。
251A:	CA9D01	JP	Z, 019DH	; 如是 INKEY$ 的代号，则转移。
251D:	FEC4	CP	C4H	; 测试是否是 STRING$ 的代号。
251F:	CA2F2A	JP	Z, 2A2FH	; 如是 STRING$ 的代号，则转移。
2522:	FEBE	CP	BEH	; 测试是否是 FN 的代号。
2524:	CA5541	JP	Z, 4155H	; 如是 FN, 则转移:磁盘 BASIC 用 (JP 558EH)。
2527:	D6D7	SUB	D7H	; 测试是否是 SGN 到 MID$ 范围内的代号。
2529:	D24E25	JP	NC, 254EH	; 如是 SGN 到 MID$ 范围内的代号，则转移。
252C:	CD3523	CALL	2335H	; 在代号值小于 D7H 和超过 BCH 时，调用暂停，当表达式算完就返回，
252F:	CF	RST	08H	; 测试下一个字符是否为右括号')'。
2530:	29	ADD	HL, HL	; 2530H: 给出右括号的值 29H——')'。
2531:	C9	RET		; 返回到调用它的程序。
2532:	167D	LD	D, 7DH	; 取优先值。★★★二进制减去子程序★★★★★★★
2534:	CD3A23	CALL	233AH	; 计算变量。
2537:	2AF340	LD	HL, (40F3H)	; 取代码串中的下一个元素的地址。
253A:	E5	PUSH	HL	; 保存继续执行的地址。
253B:	CD7B09	CALL	097BH	; 当前值的符号取反。

地址	目的码	源程序		注　释
253E:	E1	POP	HL	; 恢复代码串地址。执行函数 SGN—MID$ 以后，返回到这里。
253F:	C9	RET		; 返回到表达式计算。
2540:	CD0D26	CALL	260DH	; 取变量的地址。★★★找变量地址。由 HL 作指针给出变量名 ★★★★★★★★★★★★★★★★★★★★
2543:	E5	PUSH	HL	; 保存代码串地址。
2544:	EB	EX	DE, HL	; 把变量地址放到 HL 中。
2545:	222141	LD	(4121H), HL	; 把它存在 4121H 和 4122H 单元。
2548:	E7	RST	20H	; 确定数据类型。
2549:	C4F709	CALL	NZ, 09F7H	; 如是数字数据，要调用 09F7H：把数字值移到 4127H 单元。
254C:	E1	POP	HL	; HL＝输入字符串中下一个符号的地址。
254D:	C9	RET		; 返回到调用它的程序。
254E:	0600	LD	B, 00H	; B＝0 ★★★ SGN—MID$ ★★★★★★★★★★★★
2550:	07	RLCA		; A＝2×(代号－ D7H)。
2551:	4F	LD	C, A	; 保存新的代号。
2552:	C5	PUSH	BC	; 把 B＝0，C＝2×(代号－ D7H) 保存到堆栈中。
2553:	D7	RST	10H	; 从代号化的字符串中，取下一个字符。
2554:	79	LD	A, C	; 寻找从 SGN 到 CHR$ 的代号。
2555:	FE41	CP	41H	; 测试是否是调整过的代号。
2557:	3816	JR	C, 256FH	; 如是 SGN—CHR$ 的代号，则转移，否则它就是 LEFT$ 到 MID$ 范围内的代号。
2559:	CD3523	CALL	2335H	; 计算调用程序的表达式部分。调用程序中有 2 或 3 个参数。
255C:	CF	RST	08H	; 测试下一个字符是否是逗号。
255D:	2C	INC	L	; 255DH：给出逗号的值 2CH——','。
255E:	CDF40A	CALL	0AF4H	; 验证当前变量是一个串变量，否则有错误。
2561:	EB	EX	DE, HL	; 确认当前变量是一个串变量。
2562:	2A2141	LD	HL, (4121H)	; DE＝程序语句中的当前位置。HL＝字符串的地址。
2565:	E3	EX	(SP), HL	; 把字符串的地址放到堆栈，后跟字符串。
2566:	E5	PUSH	HL	; 把 H＝00，L＝2×（代号－ D7H）存入堆栈。
2567:	EB	EX	DE, HL	; 把程序语句中的当前位置放到 HL 中。
2568:	CD1C2B	CALL	2B1CH	; 计算字符串函数的 *n* 部分。
256B:	EB	EX	DE, HL	; DE＝语句中的当前位置。HL＝*n*。
256C:	E3	EX	(SP), HL	; 把 *n* 传送到堆栈。HL＝2×(代号－ D7H)。
256D:	1814	JR	2583H	; 转移到代号的工作子程序。
256F:	CD2C25	CALL	252CH	; 计算表达式。简单变量参数的调用。
2572:	E3	EX	(SP), HL	; HL＝0＋2×(代号－ D7H)。
2573:	7D	LD	A, L	; A＝2×(代号－ D7H)。
2574:	FE0C	CP	0CH	; 测试是否是 SGN 到 SQR 范围内的代号。
2576:	3807	JR	C, 257FH	; 如是 SGN 到 SQR 范围内的代号，则转移。

地址	目的码	源程序		注　释
2578:	FE1B	CP	1BH	; 测试调整后的代号。
257A:	E5	PUSH	HL	; 然后保存 0 + 2 ×(代号－ D7H)。
257B:	DCB10A	CALL	C, 0AB1H	; 如是 SQR 到 ATN 范围内的代号,则把 4121H 单元内的整数变换成单精度数。
257E:	E1	POP	HL	; 把代号重新放到 HL。
257F:	113E25	LD	DE, 253EH	; 把 253EH 的返回地址推入堆栈,
2582:	D5	PUSH	DE	; 于是在执行函数功能以后,就能返回。
2583:	010816	LD	BC, 1608H	; 函数 SGN 到 MID$ 的地址。
2586:	09	ADD	HL, BC	; 加上待求函数的索引。
2587:	4E	LD	C, (HL)	; C=函数地址的 LSB。
2588:	23	INC	HL	; HL 加 1, 指向 MSB。
2589:	66	LD	H, (HL)	; H=函数地址的 MSB。
258A:	69	LD	L, C	; HL=函数的地址。
258B:	E9	JP	(HL)	; 转移入 SGN 到 MID$ 范围内的函数。
258C:	CDD729	CALL	29D7H	; 确认字符串装进字符串数据区。★★★ 两个字符串作关系运算的比较 ★★★★★★★★★★★★★★★★
258F:	7E	LD	A, (HL)	; A=长度。
2590:	23	INC	HL	; HL 加 1, 指向字符串地址的 LSB。
2591:	4E	LD	C, (HL)	; 取字符串地址的 LSB。
2592:	23	INC	HL	; HL 加 1, 指向字符串地址的 MSB。
2593:	46	LD	B, (HL)	; BC=字符串地址。
2594:	D1	POP	DE	; 清除堆栈。
2595:	C5	PUSH	BC	; 保存第一个字符串的地址。
2596:	F5	PUSH	AF	; A=第一个字符串的长度。
2597:	CDDE29	CALL	29DEH	; 把第二个字符串地址放进 HL。
259A:	D1	POP	DE	; D=第一个字符串地址的长度。
259B:	5E	LD	E, (HL)	; E=第二个字符串中的字符数。
259C:	23	INC	HL	; HL 加 1, 指向第二个字符串地址的 LSB。
259D:	4E	LD	C, (HL)	; C=字符串 2 的地址的 LSB。
259E:	23	INC	HL	; HL 加 1, 指向地址的 MSB。
259F:	46	LD	B, (HL)	; BC=字符串 2 的地址。
25A0:	E1	POP	HL	; HL=字符串 1 的地址。
25A1:	7B	LD	A, E	; A=字符串 2 的剩余字符。
25A2:	B2	OR	D	; D=字符串 1 的剩余字符。
25A3:	C8	RET	Z	; 如所有的字符都比较过了,则返回。
25A4:	7A	LD	A, D	; 重新装入第一字符串剩下的字符数。
25A5:	D601	SUB	01H	; 测试是否计数为 0 。
25A7:	D8	RET	C	; 如字符串 1 已比较完了,则返回。
25A8:	AF	XOR	A	; 清除 A 寄存器。
25A9:	BB	CP	E	; 给出 0 ——剩余的字符串 2 的字符数。
25AA:	3C	INC	A	; 测试是否在字符串 2 中已没有剩余字符。

地址	目的码	源程序		注　释
25AB:	D0	RET	NC	; 如字符串 2 已比较完,则返回。
25AC:	15	DEC	D	; 字符串 1 剩余字符数减 1 。
25AD:	1D	DEC	E	; 字符串 2 剩余字符数减 1 。
25AE:	0A	LD	A, (BC)	; 把字符串 1 中的一个字符同字符串 2 的一个字符比较。
25AF:	BE	CP	(HL)	; 进行比较。
25B0:	23	INC	HL	; 字符串 1 地址加 1。
25B1:	03	INC	BC	; 字符串 2 地址加 1 。
25B2:	28ED	JR	Z, 25A1H	; 如字符相同,则转移。
25B4:	3F	CCF		; 否则进位标志求反,于是在 960H 将给出 +1 或 −1。
25B5:	C36009	JP	0960H	; 返回调用它的程序。
25B8:	3C	INC	A	; 当前算符的值加 1。
25B9:	8F	ADC	A, A	; ★★★ 比较两个逻辑量★★★★★★★★★★★★★★ 如为 0 ,则 C 标志不置位, 并给出 1, 如为 FFH, 则 C 标志置位,并给出 0 。
25BA:	C1	POP	BC	; 取另一个操作数的值。
25BB:	A0	AND	B	; 把两个值合并在一起。
25BC:	C6FF	ADD	A, FFH	; 如两者相等,得到 0 ,如不相等,置位进位标志。
25BE:	9F	SBC	A, A	; 如相等,设定 A=0 如果不相等,则置 1 。
25BF:	CD8D09	CALL	098DH	; 如果 A 是正,置当前值为 0 ,如 A 是负, 则置 FFH。
25C2:	1812	JR	25D6H	; 继续进行表达式的计算。
25C4:	165A	LD	D, 5AH	; D=优先值。★★★ NOT 子程序 ★★★★★★★★

说明 在计算一个表达式时,从加法程序进入。

25C6:	CD3A23	CALL	233AH	; 计算表达式的剩余部分, 直到有一个较高优先值时为止。
25C9:	CD7F0A	CALL	0A7FH	; 把当前值变换成整数。
25CC:	7D	LD	A, L	; 结果放在 HL 中。
25CD:	2F	CPL		; 整数的 LSB 求反。
25CE:	6F	LD	L, A	; LSB 重新存到 L。
25CF:	7C	LD	A, H	; 然后再取 MSB。
25D0:	2F	CPL		; 整数的 MSB 求反。
25D1:	67	LD	H, A	; MSB 重新存到 H。
25D2:	222141	LD	(4121H), HL	; 把已求反的数作为当前值存起来。
25D5:	C1	POP	BC	; 清除堆栈。
25D6:	C34623	JP	2346H	; 继续进行表达式的计算。
25D9:	3AAF40	LD	A, (40AFH)	; 取 WRA1 中的值的数据类型★★★ RST 20H子程序★

说明 放在 40AFH 单元的数据类型如下:

类型	代码	Z 标志	C 标志	N 标志	P 标志	A 寄存器
整　数	02	NZ	C	N	E	−1
字符串	03	Z	C	P	E	0
单精度	04	NZ	C	P	O	1
双精度	08	NZ	NC	P	E	5

地址	目的码	源程序		注　释
25DC:	FE08	CP	08H	; 准备设置数据标志。
25DE:	3005	JR	NC, 25E5H	; 如果是双精度，则转移。
25E0:	D603	SUB	03H	; 不是双精度，则减 3 。
25E2:	B7	OR	A	; 然后根据结果设置状态标志，
25E3:	37	SCF		; 进位标志 C 置 1 ，
25E4:	C9	RET		; 并返回。
25E5:	D603	SUB	03H	; 是双精度，减 3 。
25E7:	B7	OR	A	; 然后，根据结果设置状态标志，
25E8:	C9	RET		; 并且不置位进位标志，返回。
25E9:	C5	PUSH	BC	; ★★★ 逻辑算符子程序——从表达式计算进入★★★ B=最后一个算符的优先值。
25EA:	CD7F0A	CALL	0A7FH	; 把当前值变换成整数。
25ED:	F1	POP	AF	; 把 BC 值放进 AF。
25EE:	D1	POP	DE	; 把返回地址放入 DE。
25EF:	01FA27	LD	BC, 27FAH	; 把新的返回地址放入 BC。
25F2:	C5	PUSH	BC	; 返回地址存到堆栈。
25F3:	FE46	CP	46H	; 是 'OR' 的代号吗?
25F5:	2006	JR	NZ, 25FDH	; 不是，转移到比较子程序。
25F7:	7B	LD	A, E	; 把 HL 同 DE 比较。结果放在 HL 内。
25F8:	B5	OR	L	; E 和 L 比较，结果在 A 内。
25F9:	6F	LD	L, A	; 重新存入 L。
25FA:	7C	LD	A, H	; H 和 D 比较，结果留在 A 内。
25FB:	B2	OR	D	; 将在 27FAH 处被移到 H 中。
25FC:	C9	RET		; 返回到 27FAH，把结果变换成整数，返回 2346H。
25FD:	7B	LD	A, E	; 将 DE 和 HL 作逻辑比较，结果在 HL 中。
25FE:	A5	AND	L	; E 和 L 逻辑与。
25FF:	6F	LD	L, A	; 结果放在 L 中。
2600:	7C	LD	A, H	; 把 H 的内容放入 A 内，
2601:	A2	AND	D	; 于是可以与 D 比较。留在 A 中的结果将在 27FAH 传送到 H 中。
2602:	C9	RET		; 返回到 27FAH。使结果为一个整数，返回到 2346H。
2603:	2B	DEC	HL	; 把代码串指针后退一字节。 ★★★★★★★★★★★
2604:	D7	RST	10H	; 重新测试最后符号。
2605:	C8	RET	Z	; 如果语句结束，则返回。
2606:	CF	RST	08H	; 测试下一个字符是否是单引号。
2607:	2C	INC	L	; 2607H: 给出单引号的值 2CH——(')
2608:	010326	LD	BC, 2603H	; 寻找变量的地址。 ★★★ 使得返回到 2603H ★★★
260B:	C5	PUSH	BC	; 260CH: OR AF——设定建立方式。
260C:	F6AF	OR	AFH	; 260DH: XOR A——使 A 为 0，设定 (40AEH)=寻找方式。
260E:	32AE40	LD	(40AEH), A	; 设 (40AEH)=寻找或建立方式。

地址	目的码	源程序		注　释
2611:	46	LD	B, (HL)	; 保存变量名的第一个字符。
2612:	CD3D1E	CALL	1E3DH	; 检查是否是字母。
2615:	DA9719	JP	C, 1997H	; 如 C 标志置位(HL 内不是一个字母),则为 SN 错误: 变量名不由一个字母开始。
2618:	AF	XOR	A	; 清除 A 寄存器和 C 标志。
2619:	4F	LD	C, A	; 使 C 为 0。
261A:	D7	RST	10H	; 取输入字符串中的下一个字符。
261B:	3805	JR	C, 2622H	; 如是数字,则转移。
261D:	CD3D1E	CALL	1E3DH	; 测试是否是字母数字。如失败,则置位 C 标志。
2620:	3809	JR	C, 262BH	; 如果不是一个字母,则转移。如果不是一个字母,数字或(,则为错误。
2622:	4F	LD	C, A	; 名字的第二个字符放到 C。
2623:	D7	RST	10H	; 测试第二个字符后面的符号,直到找到一个非数字符号为止。
2624:	38FD	JR	C, 2623H	; 如字符是数字,则转移。
2626:	CD3D1E	CALL	1E3DH	; 测试是否是字母。
2629:	30F8	JR	NC, 2623H	; 如是一个字母,则转移。
262B:	115226	LD	DE, 2652H	; 现在定位在变量名的结束处。仅最前面的两个字符有用。
262E:	D5	PUSH	DE	; 把返回地址 2652H 放入堆栈。
262F:	1602	LD	D, 02H	; 测试名字后面的字符是否是%。
2631:	FE25	CP	25H	; 如果是%,设置 D 为数据类型 2。
2633:	C8	RET	Z	; 如果是%(整数): D=2, 则返回(转移到 2652H)。
2634:	14	INC	D	; 如变量是一个串变量时,使 D 成为 3。
2635:	FE24	CP	24H	; 测试变量名后面是否为 $。
2637:	C8	RET	Z	; 如是 $ (字符串): D=3, 则返回。
2638:	14	INC	D	; 在变量是单精度时,使 D 成为 4。
2639:	FE21	CP	21H	; 测试变量名后面是否是!
263B:	C8	RET	Z	; 如果是!(单精度): D=4, 则返回。
263C:	1608	LD	D, 08H	; 在变量是双精度时,使 D 成为 8。
263E:	FE23	CP	23H	; 测试变量名后面是否是#。
2640:	C8	RET	Z	; 如果是#(双精度): D=8, 则返回。
2641:	78	LD	A, B	; 重取符号的第一个字符。

说明　变量名不跟有类型后缀。用变量名的第一个字符来确定数据类型。

2642:	D641	SUB	41H	; 将它从字母变换成数字(0 到 26)。十进制数 0 到 26。
2644:	E67F	AND	7FH	; 清除可能有的符号位。
2646:	5F	LD	E, A	; E=0 (A) 到 26(Z)。
2647:	1600	LD	D, 00H	; DE=0 (A) 到 26(Z)。
2649:	E5	PUSH	HL	; 保存输入序列中的当前位置。
264A:	210141	LD	HL, 4101H	; 数据类型表的首址。

地址	目的码	源程序		注释
264D:	19	ADD	HL, DE	; 加变量名的第一个字符的值 (0=A, 1=B, …26=Z)。
264E:	56	LD	D, (HL)	; 取数据类型。
264F:	E1	POP	HL	; 恢复指针为输入序列中的当前位置。
2650:	2B	DEC	HL	; 后退一个位置。
2651:	C9	RET		; 返回。此时 D 中放有数据类型(返回到 2652H)。
2652:	7A	LD	A, D	; D=数据类型。★★★ 继续寻找变量名 ★★★★★★
2653:	32AF40	LD	(40AFH), A	; 保存数据类型标志。
2656:	D7	RST	10H	; 取变量名的下一个字符 (CALL 1D78H)。
2657:	3ADC40	LD	A, (40DCH)	; 得到 'FOR' 语句标志。
265A:	B7	OR	A	; 对它进行测试。
265B:	C26426	JP	NZ, 2664H	; 如果是处理 'FOR' 语句,则转移。
265E:	7E	LD	A, (HL)	; 从代码串中再取下一个元素。
265F:	D628	SUB	28H	; 与(比较。
2661:	CAE926	JP	Z, 26E9H	; 如是'('(下标变量),则转移。
2664:	AF	XOR	A	; 使 A 寄存器为 0。
2665:	32DC40	LD	(40DCH), A	; 作为一个非下标变量的标志。
2668:	E5	PUSH	HL	; HL=输入字符串中的当前位置。
2669:	D5	PUSH	DE	; 保存数据类型标志。
266A:	2AF940	LD	HL, (40F9H)	; HL=程序指针的结束=简单变量表的首址。
266D:	EB	EX	DE, HL	; DE=简单变量的地址。
266E:	2AFB40	LD	HL, (40FBH)	; 数组指针的首址。
2671:	DF	RST	18H	; 把下一个简单变量的地址与数组表的首址作比较。
2672:	E1	POP	HL	; HL=数据类型标志。
2673:	2819	JR	Z, 268EH	; 变量尚未作定义,转移。
2675:	1A	LD	A, (DE)	; 得到当前变量的类型。
2676:	6F	LD	L, A	; 保存在 L 中。
2677:	BC	CP	H	; 比较类型。
2678:	13	INC	DE	; DE 加 1, 指向这一项名字的第二个字符。
2679:	200B	JR	NZ, 2686H	; 类型不相符。跳到表中的下一个变量。
267B:	1A	LD	A, (DE)	; 类型相符,把 VLT 中的名字的第二个字符,
267C:	B9	CP	C	; 与 BC 中名字的第二个字符作比较。
267D:	2007	JR	NZ, 2686H	; 不一致,去 VLT 中找下一项。
267F:	13	INC	DE	; 第二个字符一致, DE 加 1 以后, 指向 VLT 中名字的第一个字符。
2680:	1A	LD	A, (DE)	; 将它与 BC 中变量名的第一个字符比较。
2681:	B8	CP	B	; 测试名字的第一个字符是否相符。
2682:	CACC26	JP	Z, 26CCH	; 已找到简单变量的地址,转移到 26CCH。
2685:	3E13	LD	A, 13H	; 2686H; INC DE——指向简单变量表中的下一项。
2687:	13	INC	DE	;
2688:	E5	PUSH	HL	; 保存数据类型,以便在 2672H 处重新取用它。
2689:	2600	LD	H, 00H	;

地址	目的码	源程序		注　释
268B:	19	ADD	HL, DE	; 指向变量表的下一项。
268C:	18DF	JR	266DH	; 继续搜索变量名。
268E:	7C	LD	A, H	; 保存数据变型。
268F:	E1	POP	HL	; 清除堆栈，HL＝输入代码串中的当前位置。
2690:	E3	EX	(SP), HL	; HL＝返回地址，堆栈＝输入代码串中的当前位置。
2691:	F5	PUSH	AF	; A＝数据类型。
2692:	D5	PUSH	DE	; DE＝数组指针的首址。
2693:	11F124	LD	DE, 24F1H	; VARPTR 定位程序的地址。
2696:	DF	RST	18H	; 是从 VARPTR 处调用的吗?
2697:	2836	JR	Z, 26CFH	; 是，转移到 26CFH。
2699:	114325	LD	DE, 2543H	; DE＝找变量地址的子程序的地址。
269C:	DF	RST	18H	; 是从搜索变量地址的子程序调用的吗?
269D:	D1	POP	DE	; 清除堆栈中的数组指针的首址。
269E:	2835	JR	Z, 26D5H	; 在计算一个下标时调用。这是第一次引用简单变量，为它下定义。
26A0:	F1	POP	AF	; 清除堆栈，A＝数据类型。
26A1:	E3	EX	(SP), HL	; HL＝输入代码串中的当前位置。
26A2:	E5	PUSH	HL	; 堆栈＝返回地址。
26A3:	C5	PUSH	BC	; 把 BC（名字的第一个字符/第二个字符）放入堆栈，后跟返回地址。
26A4:	4F	LD	C, A	; C＝类型。
26A5:	0600	LD	B, 00H	; 清除 B 寄存器，供计算用。
26A7:	C5	PUSH	BC	; 保存 BC 内容。现在要为当前变量在空闲区建立一个新项。
26A8:	03	INC	BC	; B＝0，C＝数据类型。
26A9:	03	INC	BC	; 给出的是类型加 2。
26AA:	03	INC	BC	; 给出(数据类型 ＋03) ＝ 3 字节多余开销＋变量的备用部分。
26AB:	2AFD40	LD	HL, (40FDH)	; 装入空闲存储器指针的首址。
26AE:	E5	PUSH	HL	; 保存空闲存储器指针。
26AF:	09	ADD	HL, BC	; 空闲存储器指针加类型（长度）可以得出新的空闲存储器指针。
26B0:	C1	POP	BC	; BC＝老的空闲存储器指针。
26B1:	E5	PUSH	HL	; 保存新的空闲存储器指针。
26B2:	CD5519	CALL	1955H	; 把数组表下移。把值加到简单变量表中。
26B5:	E1	POP	HL	; 弹出 HL 的内容。
26B6:	22FD40	LD	(40FDH), HL	; 存储新的空闲存储器指针(它是正式的)。
26B9:	60	LD	H, B	; HL＝老的空闲存储器指针＝新项的第一个字节。
26BA:	69	LD	L, C	; L＝老的空闲存储器指针的 LSB。
26BB:	22FB40	LD	(40FBH), HL	; 数组指针的新首址。
26BE:	2B	DEC	HL	; 使给出的新项为 0，即把新的空闲存储器指针

地址	目的码	源程序		注　释
26BF:	3600	LD	(HL), 00H	; 和数组指针首址之间的全部空间清零。
26C1:	DF	RST	18H	; 达到表的终点了吗?
26C2:	20FA	JR	NZ, 26BEH	; 没到,进行循环。
26C4:	D1	POP	DE	; 取长度(类型)。
26C5:	73	LD	(HL), E	; 作为第一个字存在新的项中。
26C6:	23	INC	HL	; HL 加 1, 指向项的下一个位置。
26C7:	D1	POP	DE	; 得到名字的第二个字符,
26C8:	73	LD	(HL), E	; 并作为项的第二个字存起来。
26C9:	23	INC	HL	; HL 加 1, 指向项的第三个字符。
26CA:	72	LD	(HL), D	; 现在得到名字的第一个字符。
26CB:	EB	EX	DE, HL	; DE=项中值的首址。
26CC:	13	INC	DE	; 把变量名的地址留在 DE 中。
26CD:	E1	POP	HL	; 在返回以前,清除堆栈。
26CE:	C9	RET		; 返回到调用它的程序。
26CF:	57	LD	D, A	; D=类型。 ★★★★★★★★★★★★★★★★★★★★
26D0:	5F	LD	E, A	; E=类型。
26D1:	F1	POP	AF	; 清除堆栈。
26D2:	F1	POP	AF	; 清除堆栈。
26D3:	E3	EX	(SP), HL	; 返回地址放到堆栈。代码串放到 HL。
26D4:	C9	RET		; 返回到 VARPTR 子程序。
26D5:	322441	LD	(4124H), A	; 把 WRAI 置成 0。★★★寻找下标变量 ★★★★★★
26D8:	C1	POP	BC	; 清除堆栈。
26D9:	67	LD	H, A	; 把 H 置成 0。
26DA:	6F	LD	L, A	; 把 L 置成 0。
26DB:	222141	LD	(4121H), HL	; 把 WRA1 中的字符串指针置成 0。
26DE:	E7	RST	20H	; 确定数据类型。
26DF:	2006	JR	NZ, 26E7H	; 如果不是字符串,则转移。
26E1:	212819	LD	HL, 1928H	; READY 信息的地址。
26E4:	222141	LD	(4121H), HL	; 放到 WRA1 中。
26E7:	E1	POP	HL	; 恢复代码串的地址。
26E8:	C9	RET		; 返回到调用它的程序。
26E9:	E5	PUSH	HL	; 保存输入代码串中的当前位置。★★★ 寻找下标变量地址 ★★★★★★★★★★★★★★★★★★★★★★

说明 入口时: D=类型,B=变量名的第一个字符,C=名字的第二个字符,HL=输入代码串中当前位置。

26EA:	2AAE40	LD	HL, (40AEH)	; HL=00 (寻找方式), <>0(建立方式)。
26ED:	E3	EX	(SP), HL	; 堆栈=(40AEH), HL=代码串地址。
26EE:	57	LD	D, A	; D 清零。
26EF:	D5	PUSH	DE	; D=0, E=第一个字符的数字值。
26F0:	C5	PUSH	BC	; BC=以 ASCII 表示的名字的第一个字符/第二个字符。
26F1:	CD451E	CALL	1E45H	; 计算每一个下标直到第一个 ')' 或 ',' 符号为止。结果

地址	目的码	源程序		注　释
				在 DE 中(整数)。
26F4:	C1	POP	BC	; BC=以 ASCII 表示的名字的第一个字符/第二个字符。
26F5:	F1	POP	AF	; A=0
26F6:	E8	EX	DE, HL	; DE=输入代码串中的当前位置，下标表示式结束。HL=下标值。
26F7:	E3	EX	(SP), HL	; 堆栈=下标值，HL=(40AEH)。
26F8:	E5	PUSH	HL	; 保存输入代码串中的当前位置。
26F9:	EB	EX	DE, HL	; HL=输入代码串中的当前位置，DE=(40AEH)。
26FA:	3C	INC	A	; 计算出的下标数加 1，
26FB:	57	LD	D, A	; 并保存在 D 中。
26FC:	7E	LD	A, (HL)	; 取终止符号。
26FD:	FE2C	CP	2CH	; 如果终止符号是一个逗号，
26FF:	28EE	JR	Z, 26EFH	; 则去计算下一个下标，
2701:	CF	RST	08H	; 否则，测试输入序列中的下一个字符是否是 ','。
2702:	29	ADD	HL, HL	; 2702H: 给出逗号的代码 29H——','
2703:	22F340	LD	(40F3H), HL	; (40F3H)=下标表达式的终止符号的地址。
2706:	E1	POP	HL	; HL=(40AEH)，下标计算以前的值。建立或寻找方式标志。
2707:	22AE40	LD	(40AEH), HL	; 并保存起来，以备今后使用。
270A:	D5	PUSH	DE	; DE=算出的下标的数。
270B:	2AFB40	LD	HL, (40FBH)	; 数组指针的首址。
270E:	3E19	LD	A, 19H	; 270FH: ADD HL, DE——计算数组的终址，寻找已命名的数组。
2710:	EB	EX	DE, HL	; DE=下一个数组的地址。
2711:	2AFD40	LD	HL, (40FDH)	; 取空闲存储器指针——为重新搜索设立上限。
2714:	EB	EX	DE, HL	; HL=数组指针。DE=空闲存储器指针。
2715:	DF	RST	18H	; 把空闲存储器指针同数组指针比较。
2716:	3AAF40	LD	A, (40AFH)	; 数据类型/长度标志。
2719:	2827	JR	Z, 2742H	; 如果名字没有找到和测试过了所有的数组，则转移。
271B:	BE	CP	(HL)	; 把数组项的数据类型与正寻找的类型作比较。
271C:	23	INC	HL	; HL 加 1，指向下一个数组。
271D:	2008	JR	NZ, 2727H	; 类型不相符。跳到下一个数组。
271F:	7E	LD	A, (HL)	; 数据类型相符。
2720:	B9	CP	C	; 现在检查名字的第二个字符是否相符。
2721:	23	INC	HL	; HL 加 1，指向下一个数组。
2722:	2004	JR	NZ, 2728H	; 第二个字符不相符，则跳到下一个数组。
2724:	7E	LD	A, (HL)	; 第二个字符相符。
2725:	B8	CP	B	; 测试第一个字符，如果一致，则 Z 标志置位。
2726:	3E23	LD	A, 23H	; 2727H: INC HL
2728:	23	INC	HL	; HL 加 1，指向数组项的下一个字节。
2729:	5E	LD	E, (HL)	; E=到下一个数组的偏移值的 LSB。

地址	目的码	源程序		注　释
272A:	23	INC	HL	; HL 加 1，指向数组项的下一个字节。
272B:	56	LD	D, (HL)	; DE＝到下一个数组的偏移值。
272C:	23	INC	HL	; 指向下标数的项。
272D:	20E0	JR	NZ, 270FH	; 没有找到给定的数组名，检查下一项。
272F:	3AAE40	LD	A, (40AEH)	; 第一个字符相符。于是我们找到了数组表中的变量的地址。
2732:	B7	OR	A	; 处于建立方式吗?
2733:	1E12	LD	E, 12H	;
2735:	C2A219	JP	NZ, 19A2H	; 是，则有错误：数组(名)被二次定义。
2738:	F1	POP	AF	; A＝算出的下标数。
2739:	96	SUB	A, (HL)	; 与 DIM 语句中指定的数比数。
273A:	CA9527	JP	Z, 2795H	; 如果下标数一致，则转移。
273D:	1E10	LD	E, 10H	; BS 错误代码。
273F:	C3A219	JP	19A2H	; 输出 BS 错误信息。
2742:	77	LD	(HL), A	; 存储类型。建立一个下标变量项。
2743:	23	INC	HL	; HL 加 1，指向名字的第二个字符。
2744:	5F	LD	E, A	; E＝每项字节数。
2745:	1600	LD	D, 00H	; D＝00。
2747:	F1	POP	AF	; A＝下标数。
2748:	71	LD	(HL), C	; 存储名字的第二个字符。
2749:	23	INC	HL	; HL 加 1，指向名字第一个字符的位置。
274A:	70	LD	(HL), B	; 存储名字的第一个字符。
274B:	23	INC	HL	; HL 加 1，指向到下一个数组的偏移值的 LSB。
274C:	4F	LD	C, A	; C＝下标数。
274D:	CD6319	CALL	1963H	; 计算 HL 和空闲存储器之间留下的字节总数。
2750:	23	INC	HL	; 跳过偏移项。
2751:	23	INC	HL	; HL＝项内下标数的位置。
2752:	22D840	LD	(40D8H), HL	; (40D8H)＝最大下标数的位置。
2755:	71	LD	(HL), C	; 将这个数组(1，2 或 3)的下标数存起来。
2756:	23	INC	HL	; HL 指向数组表中的第一个下标项。
2757:	3AAE40	LD	A, (40AEH)	; A＝建立或寻找方式的标志。
275A:	17	RLA		; 设置进位标志：0——寻找方式，1——建立方式。
275B:	79	LD	A, C	; A＝这个数组的下标数。
275C:	010B00	LD	BC, 000BH	; 如果名字没有指明下标，则缺项下标＝10＋1。
275F:	3002	JR	NC, 2763H	; 如果不可能寻找而要建立，则转移。
2761:	C1	POP	BC	; 否则，处于建立方式。得到用户指定的下标。
2762:	03	INC	BC	; BC 加 1。
2763:	71	LD	(HL), C	; 并存在数组表内。
2764:	23	INC	HL	;
2765:	70	LD	(HL), B	;
2766:	23	INC	HL	; HL 加 1，指向下一组下标。

地址	目的码	源程序		注释
2767:	F5	PUSH	AF	; 保存建立方式/寻找方式标志。
2768:	CDAA0B	CALL	0BAAH	; 下标的大小乘每项的字节数。累加放在 DE 中的乘积，
276B:	F1	POP	AF	; 最后，DE=以字节表示的数组的大小。
276C:	3D	DEC	A	; 要乘的下标数减 1。
276D:	20ED	JR	NZ, 275CH	; 如果还有下标，则转移。
276F:	F5	PUSH	AF	; 保存建立/寻找标志。
2770:	42	LD	B, D	; B=数组长度的 MSB。
2771:	4B	LD	C, E	; BC=以字节表示的数组的长度。
2772:	EB	EX	DE, HL	; DE=数组的首址——数组表中的当前地址。
2773:	19	ADD	HL, DE	; HL=数组的终址。
2774:	38C7	JR	C, 273DH	; 若超过 2^{16}，则为 BS 错误。
2776:	CD6C19	CALL	196CH	; 测试空闲区的总数，如果足够，则返回。
2779:	22FD40	LD	(40FDH), HL	; (40FDH)=数组的最低工作地址。
277C:	2B	DEC	HL	; 数组清零，从终址开始，
277D:	3600	LD	(HL), 00H	; 朝首址方向进行。
277F:	DF	RST	18H	; 到首址了吗?
2780:	20FA	JR	NZ, 277CH	; 没有，循环。
2782:	03	INC	BC	; BC=数组中字节数加 1。
2783:	57	LD	D, A	; D=0。
2784:	2AD840	LD	HL, (40D8H)	; HL=下标数的地址。
2787:	5E	LD	E, (HL)	; DE=最大的下标数。
2788:	EB	EX	DE, HL	; DE=下标数的地址。HL=最大的下标数。
2789:	29	ADD	HL, HL	; HL=2×下标数。
278A:	09	ADD	HL, BC	; HL=2×下标数+数组的大小。
278B:	EB	EX	DE, HL	; HL=下标数的地址。
278C:	2B	DEC	HL	; 后退两个字节，到偏移值地址。
278D:	2B	DEC	HL	; 第二次后退。
278E:	73	LD	(HL), E	; 把相对于数组表中下一项的偏移值存起来。
278F:	23	INC	HL	;
2790:	72	LD	(HL), D	;
2791:	23	INC	HL	; HL=下标个数项的地址。
2792:	F1	POP	AF	; 恢复建立/寻找标志。
2793:	3830	JR	C, 27C5H	; 如在建立方式，则转移。
2795:	47	LD	B, A	; 对于第一遍循环，BC=0。

说明 继续数组处理。寻找下标变量的地址，然后取它的值。以列为主的格式排列。

2796:	4F	LD	C, A	; C=0。
2797:	7E	LD	A, (HL)	; A=数组中下标的个数。
2798:	23	INC	HL	; HL 加 1，指向右下标(最大值加 1)。
2799:	16E1	LD	D, E1H	; 279AH: POP HL。下一个下标的极限值的字地址。
279B:	5E	LD	E, (HL)	; E=下标极限值的 LSB。

地址	目的码	源程序		注　释
279C:	23	INC	HL	; HL 加 1，指向 MSB 的位置。
279D:	56	LD	D, (HL)	; D=下标极限值的 MSB。
279E:	23	INC	HL	; HL=下一个下标极限值的地址。
279F:	E3	EX	(SP), HL	; HL=调用它的程序的下标值。堆栈=下一个下标极限值的地址。
27A0:	F5	PUSH	AF	; 保存下标数。
27A1:	DF	RST	18H	; 现在，把用户的下标与那个下标的极限值作比较。
27A2:	D23D27	JP	NC, 273DH	; 如果下标值大于允许值，则转移——给出 BS 错误。
27A5:	CDAA0B	CALL	0BAAH	; 前一个下标乘当前下标最大的允许值。
27A8:	19	ADD	HL, DE	; 乘积的和存储在 HL 中。
27A9:	F1	POP	AF	; A=下标数。
27AA:	3D	DEC	A	; 为刚处理过的下标计数。
27AB:	44	LD	B, H	; BC=前一个下标。
27AC:	4D	LD	C, L	; C=LSB。
27AD:	20EB	JR	NZ, 279AH	; 如果还有下标要进行处理，则转移。
27AF:	3AAF40	LD	A, (40AFH)	; A=数据类型标志。
27B2:	44	LD	B, H	; 现在，准备用每项的大小乘下标。
27B3:	4D	LD	C, L	;
27B4:	29	ADD	HL, HL	; 下标乘 2。
27B5:	D604	SUB	04H	; 测试数据类型。
27B7:	3804	JR	C, 27BDH	; 如是整数或字符串，则转移。
27B9:	29	ADD	HL, HL	; 都不是，计算下标乘 4。
27BA:	2806	JR	Z, 27C2H	; 如是单精度，则转移。
27BC:	29	ADD	HL, HL	; 下标乘 8，必定是双精度。
27BD:	B7	OR	A	; 设置奇偶状态标志。
27BE:	E2C227	JP	PO, 27C2H	; 如是整数，则转移。
27C1:	09	ADD	HL, BC	; 下标乘 3，字符串。
27C2:	C1	POP	BC	; BC=数组的开始地址。
27C3:	09	ADD	HL, BC	; 把索引加到基地址上。
27C4:	EB	EX	DE, HL	; DE=下标变量的地址。
27C5:	2AF340	LD	HL, (40F3H)	; 恢复代码串的位置。
27C8:	C9	RET		; 返回到调用它的程序。
27C9:	AF	XOR	A	; 清除 A 和状态标志。★★★ MEM 子程序 ★★★★
27CA:	E5	PUSH	HL	; 保存程序语句表中的当前位置。
27CB:	32AF40	LD	(40AFH),A	; 设定当前变量不是字符串，于是 FRE 就简化了处理。
27CE:	CDD427	CALL	27D4H	; 调用 FRE 子程序——送回空闲存储器的总数，作当前值。
27D1:	E1	POP	HL	; 恢复程序语句中的当前指针。
27D2:	D7	RST	10H	; 把下一个代号放进 A。
27D3:	C9	RET		; 返回到 BASIC。
27D4:	2AFD40	LD	HL, (40FDH)	; HL=空闲存储器的首址。★★★ FRE 子程序 ★★★

地址	目的码	源程序		注　释
27D7:	EB	EX	DE, HL	; DE=空闲存储器的首址指针。
27D8:	210000	LD	HL, 0000H	; HL 清零，然后把堆栈指针加到 HL 上，取得当前堆栈指针。
27DB:	39	ADD	HL, SP	; HL=当前堆栈指针。
27DC:	E7	RST	20H	; 测试数据类型。
27DD:	200D	JR	NZ, 27ECH	; 如果是从 MEM 调用的，则转移。因变量不是一个字符串。
27DF:	CDDA29	CALL	29DAH	; 把字符串的地址装进 HL。
27E2:	CDE628	CALL	28E6H	; 计算剩余空间的总数。

说明 剩余空间：

(1) 如果(自)变量不是字符串，则等于当前堆栈地址减去空闲存储器首址的指针。

(2) 如果(自)变量是字符串，则等于字符串区中的下一个可用单元(的地址)减去字符串区的首址。

27E5:	2AA040	LD	HL, (40A0H)	; 取字符串区的边界地址。
27E8:	EB	EX	DE, HL	; 把此极限值移到 DE。
27E9:	2AD640	LD	HL, (40D6H)	; HL=当前字符串区的指针。
27EC:	7D	LD	A, L	; A=一个地址的 LSB。
27ED:	93	SUB	A, E	; 减另一个地址的 LSB。
27EE:	6F	LD	L, A	; 重新存入 L。
27EF:	7C	LD	A, H	; H=一个地址的 MSB。
27F0:	9A	SBC	A, D	; 减另一个地址的 MSB。
27F1:	67	LD	H, A	; 再存入 H。HL=地址差值——(HL−DE)。
27F2:	C3660C	JP	0C66H	; 把差值变换成单精度，并返回。
27F5:	3AA640	LD	A, (40A6H)	; 取当前光标的位置。★★★ POS 子程序★★★★★★
27F8:	6F	LD	L, A	; 保存在 L 中。
27F9:	AF	XOR	A	; A, H 寄存器清零。
27FA:	67	LD	H, A	; HL=光标位置 (H=00, L=位置)。
27FB:	C39A0A	JP	0A9AH	; 把 HL 中的值放到 4121H, 类型标志为整数，返回 BASIC。
27FE:	CDA941	CALL	41A9H	; DOS 出口 (JP 5679H)。★★★ USR 子程序★★★★
2801:	D7	RST	10H	; 从输入串中取得下一个字符。
2802:	CD2C25	CALL	252CH	; 计算语句的剩余部分。得到 USR 的编号。
2805:	E5	PUSH	HL	; 保存代码串中的下一个元素的地址。
2806:	219008	LD	HL, 0890H	; 在返回到调用它的 BASIC 程序之前，
2809:	E5	PUSH	HL	; 把这个继续地址推入堆栈。
280A:	3AAF40	LD	A, (40AFH)	; A=当前数据类型。
280D:	F5	PUSH	AF	; 保存于堆栈。
280E:	FE03	CP	03H	; 测试是否是字符串。
2810:	CCDA29	CALL	Z, 29DAH	; 如果是一个字符串，把地址放进 HL。
2813:	F1	POP	AF	; 把数据类型标志重新放到 A 寄存器内。
2814:	EB	EX	DE, HL	; DE=字符串地址。
2815:	2A8E40	LD	HL, (408EH)	; (408EH) 包含有 USR 子程序的入口指针。

地址	目的码	源程序		注　释
2818:	E9	JP	(HL)	; 进入用户汇编语言子程序。
2819:	E5	PUSH	HL	; ★★★ 由 LET 调用把算术子程序的结果变换成合适的最终数据类型 ★★★★★★★★★★★★★★★★★★
281A:	E607	AND	07H	; A=结果的数据类型。
281C:	21A118	LD	HL, 18A1H	; 算术变换子程序的地址。
281F:	4F	LD	C, A	; 设定 B=00, C=数据类型。
2820:	0600	LD	B, 00H	; 其中: 数据类型——0(双精度), 1(整数), 2(字符串), 3(单精度)。
2822:	09	ADD	HL, BC	; 加上算术结果的偏移值。
2823:	CD8625	CALL	2586H	; 把结果变换成合适的数据类型。
2826:	E1	POP	HL	; 恢复 HL。
2827:	C9	RET		; 返回。
2828:	E5	PUSH	HL	; 保存代码串地址 ★★★ 从 INPUT 子程序调用 ★★
2829:	2AA240	LD	HL, (40A2H)	; HL=以二进制表示的当前行号。
282C:	23	INC	HL	; HL 加 1, 于是能够对直接语句作测试。
282D:	7C	LD	A, H	; 在输入阶段,行号=FFFFH。
282E:	B5	OR	L	;
282F:	E1	POP	HL	; 恢复代码串指针。
2830:	C0	RET	NZ	; 如行号不为 0(不是直接语句),则返回。
2831:	1E16	LD	E, 16H	; 否则给出一个 ID 错误。
2833:	C3A219	JP	19A2H	; 打印错误,并返回输入阶段。
2836:	CDBD0F	CALL	0FBDH	; 把当前值,即调用它的程序的参数变换成 ASCII 码。★
2839:	CD6528	CALL	2865H	; 为 ASCII 数建立一个文字串库项, 作为当前值保存起来。
283C:	CDDA29	CALL	29DAH	; 把当前值的地址放进 HL。
283F:	012B2A	LD	BC, 2A2BH	; 把 CHR$ 子程序中的继续地址放到 BC 中。
2842:	C5	PUSH	BC	; 把地址放到堆栈。
2843:	7E	LD	A, (HL)	; A=字符串的长度。
2844:	23	INC	HL	; HL 加 1, 指向字符串地址。
2845:	E5	PUSH	HL	; HL=字符串地址的指针。
2846:	CDBF28	CALL	28BFH	; 测试剩余的字符串区,以确保能容纳新的字符串。
2849:	E1	POP	HL	; 把字符串地址重新装入 HL。
284A:	4E	LD	C, (HL)	; C=字符串地址的 LSB。
284B:	23	INC	HL	; HL 加 1, 指向 MSB。
284C:	46	LD	B, (HL)	; BC=相当于用户值的 ASCII 字符串的地址。
284D:	CD5A28	CALL	285AH	; 把字符串的长度和地址存入 40D3H—40D5H。
2850:	E5	PUSH	HL	; HL=40D3H。
2851:	6F	LD	L, A	; L=字符串的长度。
2852:	CDCE29	CALL	29CEH	; 把字符串从 BC (暂存区)传送到 DE(字符串数据区)。
2855:	D1	POP	DE	; DE=40D3H。
2856:	C9	RET		; 返回到调用它的程序。

地址	目的码	源程序		注　释
2857:	CDBF28	CALL	28BFH	; 确认有空间，并把字符串区的下个可用地址放入 DE 中。★★★★★★★★★★★★★★★★★★★★★★

说明 把 A 和 DE 的内容保存在 40D3H—40D5H。

285A:	21D340	LD	HL, 4QD3H	; HL=暂存区的地址。
285D:	E5	PUSH	HL	; 把 40D3H 放到堆栈,以便能取回使用。
285E:	77	LD	(HL), A	; 保存字符串长度。
285F:	23	INC	HL	; HL 加 1，指向地址的 LSB 位置。
2860:	73	LD	(HL), E	; 保存字符串地址的 LSB。
2861:	23	INC	HL	; HL 加 1，指向地址的 MSB 位置。
2862:	72	LD	(HL), D	; 保存字符串地址的 MSB。
2863:	E1	POP	HL	; 恢复字符串控制块的首址。
2864:	C9	RET		; 返回到调用它的程序。
2865:	2B	DEC	HL	; 把输入指针退到引号,★★★引号子程序★★★★★★
2866:	0622	LD	B, 22H	; B=引号('')的 ASCII 值。
2868:	50	LD	D, B	; D=终止搜索的字符。
2869:	E5	PUSH	HL	; 保存开始引号的地址。
286A:	0EFF	LD	C, FFH	; 把计数器预置为－1。
286C:	23	INC	HL	; 跳过引号。
286D:	7E	LD	A, (HL)	; 取下一个字符。
286E:	0C	INC	C	; 处理过的字符计数加 1。
286F:	B7	OR	A	; 设置状态标志。
2870:	2806	JR	Z, 2878H	; 如是 EOS (语句结束),则转移。
2872:	BA	CP	D	; 测试是否是终止字符(通常是引号)。
2873:	2803	JR	Z, 2878H	; 如是终止字符,则转移。
2875:	B8	CP	B	; 第二次测试终止字符。
2876:	20F4	JR	NZ, 286CH	; 仍然不是终止字符,循环直到是终止字符时为止。
2878:	FE22	CP	22H	; 最后一个字符是一个引号吗?
287A:	CC781D	CALL	Z, 1D78H	; 如果是,则取下一个字符。
287D:	E3	EX	(SP), HL	; HL=开始的引号的地址。

说明 引号后第一个非空格字符的地址放入堆栈。

287E:	23	INC	HL	; HL 加 1，给出第一个字符的地址。
287F:	EB	EX	DE, HL	; 字符串的开始地址放到 DE。
2880:	79	LD	A, C	; A=字符串的长度。
2881:	CD5A28	CALL	285AH	; 把字符串的长度和地址放到 40D3H。
2884:	11D340	LD	DE, 40D3H	; (40D3H)=字符串数据区中字符串的长度和地址。

说明 把长度和地址从 40D3H 传送到当前的文字串内。文字库项由 40D3H 指出。把当前值设置为字符串类型,并把它的地址看作为当前文字串的地址 (40D3H)。

2887:	3ED5	LD	A, D5H	;
2889:	2AB340	LD	HL, (40B3H)	; HL=下一个可用字符串项的地址。

地址	目的码	源程序		注释
288C:	222141	LD	(4121H), HL	; 当前字符串值的地址=当前文字区的字符串值。
288F:	3E03	LD	A, 03H	; 当前值的类型=字符串。
2891:	32AF40	LD	(40AFH), A	; 放入类型标志字节内。
2894:	CDD309	CALL	09D3H	; 把字符串区长度传送到当前文字串库区。
2897:	11D640	LD	DE, 40D6H	; DE=文字串区结束地址。
289A:	DF	RST	18H	; 确认没有越出文字串库区。
289B:	22B340	LD	(40B3H), HL	; 修改下一个可用文字串库项的地址。
289E:	E1	POP	HL	; 恢复代码串地址。
289F:	7E	LD	A, (HL)	; A=代码串的下一个元素。
28A0:	C0	RET	NZ	; 如果没有越出字符串暂存区,则返回。
28A1:	1E1E	LD	E, 1EH	; ST 错误代码。
28A3:	C3A219	JP	19A2H	; 给出 ST 错误信息。
28A6:	23	INC	HL	; 信息输出子程序。 ★★★★★★★★★★★★★★★
28A7:	CD6528	CALL	2865H	; 建立文字串库项。
28AA:	CDDA29	CALL	29DAH	; 把当前变量地址放进 HL。
28AD:	CDC409	CALL	09C4H	; 把字符串长度放进 D。在 BC 内的起始地址减 1。
28B0:	14	INC	D	;
28B1:	15	DEC	D	; 为打印过的字符计数。
28B2:	C8	RET	Z	; 如果所有的字符都打印完了,则返回。
28B3:	0A	LD	A, (BC)	; 待打印的字符。
28B4:	CD2A03	CALL	032AH	; 把字符输出到系统的输出设备。
28B7:	FE0D	CP	0DH	; 测试是否是一个回车符。
28B9:	CC0321	CALL	Z, 2103H	; 如果字符是一个回车符,则转移。
28BC:	03	INC	BC	; BC 加 1,指向下一个字符。
28BD:	18F2	JR	28B1H	; 循环,直到遇到 CR, 或者打印出 D 个字符为止。
28BF:	B7	OR	A	; 计算字符串区内剩余空间的总数。 ★★★★★★★★
28C0:	0EF1	LD	C, F1H	; 28C1H: POP AF
28C2:	F5	PUSH	AF	; 保存串长度。
28C3:	2AA040	LD	HL, (40A0H)	; 把字符串区的首址装进 HL。
28C6:	EB	EX	DE, HL	; DE=字符串区的地址。
28C7:	2AD640	LD	HL, (40D6H)	; 把下一个可用的字符串单元的指针装进 HL。
28CA:	2F	CPL		; 字符串长度值求反。
28CB:	4F	LD	C, A	; 把求反后的值保存在 C 中。
28CC:	06FF	LD	B, FFH	; BC=负的字符串长度。
28CE:	09	ADD	HL, BC	; HL=新的当前字符串指针。
28CF:	23	INC	HL	; HL 加 1。
28D0:	DF	RST	18H	; 把新的字符串指针与极限值作比较。
28D1:	3807	JR	C, 28DAH	; 如进位标志置位,则给出 QS 错误。表示字符串区内空间不足。
28D3:	22D640	LD	(40D6H), HL	; 保存新的当前字符串指针。
28D6:	23	INC	HL	; HL 加 1。

地址	目的码	源程序		注释
28D7:	EB	EX	DE, HL	; DE=新的当前字符串指针。
28D8:	F1	POP	AF	; A=字符串长度。
28D9:	C9	RET		; 返回到调用它的程序。
28DA:	F1	POP	AF	; A=字符串长度，得到状态标志以了解是否准备重新安排。
28DB:	1E1A	LD	E, 1AH	; OS 错误代码。
28DD:	CAA219	JP	Z, 19A2H	; 如空区已重新安排，但仍旧没有地方，则有错误。
28E0:	BF	CP	A	; 设置状态标志为 0，并重新试一下。
28E1:	F5	PUSH	AF	; 保存 0。
28E2:	01C128	LD	BC, 28C1H	; 再试一下安排存储单元的继续地址。
28E5:	C5	PUSH	BC	; 推入堆栈。
28E6:	2AB140	LD	HL, (40B1H)	; HL=最高位置的存储器指针。
28E9:	22D640	LD	(40D6H), HL	; 把当前字符串指针重新设置成存储器终点。
28EC:	210000	LD	HL, 0000H	; HL 清 0。
28EF:	E5	PUSH	HL	; 并把 0 保存在堆栈。
28F0:	2AA040	LD	HL, (40A0H)	; HL=字符串数据区的边界。
28F3:	E5	PUSH	HL	; 把它也保存在堆栈。
28F4:	21B540	LD	HL, 40B5H	; HL=字符串指针区中第一项的地址。
28F7:	EB	EX	DE, HL	; 把 HL 的内容存到 DE 中。
28F8:	2AB340	LD	HL, (40B3H)	; HL=LSPT (文字串库表)中当前项的地址。
28FB:	EB	EX	DE, HL	; DE=字符串指针区中当前项的地址。
28FC:	DF	RST	18H	; 指出第一项 (40B5H) 的是 40B3H 吗?
28FD:	01F728	LD	BC, 28F7H	; 答案为否的情况下的继续地址。
2900:	C24A29	JP	NZ, 294AH	; 不是，转移到 294AH，并返回 28F7H。
2903:	2AF940	LD	HL, (40F9H)	; HL=简单变量指针。
2906:	EB	EX	DE, HL	; 把它保存在 DE 中。
2907:	2AFB40	LD	HL, (40FBH)	; HL=数组指针。
290A:	EB	EX	DE, HL	; HL=变量表指针。DE=数组指针。
290B:	DF	RST	18H	; 把它们的地址进行比较。它们相等吗?
290C:	2813	JR	Z, 2921H	; 相等，已经扫描了简单变量。
290E:	7E	LD	A, (HL)	; 取第一个简单变量的数据类型。
290F:	23	INC	HL	; HL 加 3，指向 LSB。
2910:	23	INC	HL	; 于是，加数据类型就能给出下一个变量的地址。
2911:	23	INC	HL	;
2912:	FE03	CP	03H	; 测试变量是否是一个字符串。
2914:	2004	JR	NZ, 291AH	; 如果不是一个字符串，则转移。
2916:	CD4B29	CALL	294BH	; 如果是一个字符串，则把它的地址放入 HL 中。
2919:	AF	XOR	A	; A 清零，因为 HL 已经指向下一项了。
291A:	5F	LD	E, A	; 指向下一个变量的地址。
291B:	1600	LD	D, 00H	;
291D:	19	ADD	HL, DE	; 给出表中下一个变量的地址。

地址	目的码	源程序		注释
291E:	18E6	JR	2906H	; 循环到所有的简单变量都测试过为止。
2920:	C1	POP	BC	; 清除 HL，把下面程序中推入堆栈的内容弹出来。
2921:	EB	EX	DE, HL	; DE= 指向当前数组项。
2922:	2AFD40	LD	HL, (40FDH)	; HL=下一个有效存储单元的地址。
2925:	EB	EX	DE, HL	; DE=第一个有效存储单元的地址。
2926:	DF	RST	18H	; 所有的数组项都扫描过了吗?
2927:	CA6B29	JP	Z, 296BH	; 是,转移。
292A:	7E	LD	A, (HL)	; 不是,取这一个数组的数据类型。
292B:	23	INC	HL	; 指向名字的第二个字符。
292C:	CDC209	CALL	09C2H	; 把到下一数组的偏移值装进 BC。跳过名字。
292F:	E5	PUSH	HL	; 保存下标数的地址。
2930:	09	ADD	HL, BC	; 加偏移值,以获得下一个数组项。
2931:	FE03	CP	03H	; 当前数据类型是字符串吗?
2933:	20EB	JR	NZ, 2920H	; 不是,循环继续寻找。
2935:	22D840	LD	(40D8H), HL	; 保存下一数组项的地址。
2938:	E1	POP	HL	; HL=下标数的地址。
2939:	4E	LD	C, (HL)	; C=下标数。
293A:	0600	LD	B, 00H	; 设定 B=0,
293C:	09	ADD	HL, BC	; 然后将下标数乘以 2，加当前地址，
293D:	09	ADD	HL, BC	; 以获得下标边界的端地址。
293E:	23	INC	HL	; HL=这个变量的下标的终地址。
293F:	EB	EX	DE, HL	; 把它传送到 DE。
2940:	2AD840	LD	HL, (40DBH)	; HL=下一个变量的地址。
2943:	EB	EX	DE, HL	; HL=下标边界的端地址，DE=下一个变量的地址。
2944:	DF	RST	18H	; 测试是否为空表。
2945:	28DA	JR	Z, 2921H	; 如果表是空的,则转移。
2947:	013F29	LD	BC, 293FH	; 字符串数组处理的继续地址。
294A:	C5	PUSH	BC	; 把继续地址保存在堆栈中。
294B:	AF	XOR	A	; 清除所有的状态标志。
294C:	B6	OR	(HL)	; A=字符串长度。
294D:	23	INC	HL	; HL 加 1，指向下两个字节，
294E:	5E	LD	E,(HL)	; 以获取字符串地址。
294F:	23	INC	HL	; HL 加 1，指向字符串地址的 MSB。
2950:	56	LD	D, (HL)	; DE=字符串地址。
2951:	23	INC	HL	; HL 加 1，指向字符串指针区的下一项(测试地址)。
2952:	C8	RET	Z	; 如字符串长度为 0，则返回。
2953:	44	LD	B, H	; 将 HL 的内容装进 BC。
2954:	4D	LD	C, L	; BC=下一个字符串指针的地址。
2955:	2AD640	LD	HL, (40D6H)	; HL=当前字符串区指针。
2958:	DF	RST	18H	; 字符串是在字符串数据区中吗?
2959:	60	LD	H, B	; 将 BC 的内容装进 HL。

地址	目的码	源程序		注　释
295A:	69	LD	L, C	; 把下一个文字串库项的地址重新放到 HL 中。
295B:	D8	RET	C	; 如果字符串是在字符串区中，则返回。
295C:	E1	POP	HL	; HL＝返回地址。
295D:	E3	EX	(SP), HL	; HL＝调用它的程序的测试地址。
295E:	DF	RST	18H	; 把调用它的程序的测试地址与字符串地址作比较。
295F:	E3	EX	(SP), HL	; 恢复堆栈为调用它的程序的测试地址，HL＝返回地址。
2960:	E5	PUSH	HL	; 重新把返回地址放到堆栈。
2961:	60	LD	H, B	; 把 BC 的内容装入 HL。
2962:	69	LD	L, C	; HL＝下一个文字串库项的地址。
2963:	D0	RET	NC	; 若字符串地址在调用它的程序的地址的下面，则返回。
2964:	C1	POP	BC	; BC＝返回地址。
2965:	F1	POP	AF	; 清除调用它的程序的字符串地址。
2966:	F1	POP	AF	; 调用它的程序的标志。
2967:	E5	PUSH	HL	; 保存下一个字符串区指针的地址。
2968:	D5	PUSH	DE	; 保存当前字符串地址。
2969:	C5	PUSH	BC	; 保存返回地址。
296A:	C9	RET		; 返回到调用它的程序。
296B:	D1	POP	DE	; DE＝送到字符串区的最后一个字符串的地址。 ★★★
296C:	E1	POP	HL	; HL＝下一个字符串区指针的地址。
296D:	7D	LD	A, L	; 如果 HL＝0，
296E:	B4	OR	H	; 则属于文字串库(暂存)的字符串区内没有字符串。
296F:	C8	RET	Z	; 若在重排过的字符串区内没有暂存的字符串，则返回。
2970:	2B	DEC	HL	; 为获取文字串库项的指针，后退一个地址。
2971:	46	LD	B, (HL)	; B＝字符串地址的 LSB。
2972:	2B	DEC	HL	; 向后跳到地址的下一个字节。
2973:	4E	LD	C, (HL)	; C＝字符串地址的 MSB。
2974:	E5	PUSH	HL	; 把字符串中指针的地址保存起来，这样就可在移动之后修改它。
2975:	2B	DEC	HL	; HL 减 1，以便取得字符串的长度。
2976:	6E	LD	L, (HL)	; L＝字符串的长度。
2977:	2600	LD	H, 00H	; 使 HL＝0，于是能作 16 位的算术运算。
2979:	09	ADD	HL, BC	; HL＝字符串的终址。
297A:	50	LD	D, B	; 把 BC 内容装入 DE。
297B:	59	LD	E, C	; DE＝字符串的首址。
297C:	2B	DEC	HL	; HL＝终址－1。
297D:	44	LD	B, H	; 把 HL 内容装入 BC。
297E:	4D	LD	C, L	; BC＝终址－1。
297F:	2AD640	LD	HL,(40D6H)	; HL＝当前字符串数据的指针。
2982:	CD5819	CALL	1958H	; 把字符串传送到字符串区表中的新区。
2985:	E1	POP	HL	; HL＝文字串指针的地址。
2986:	71	LD	(HL), C	; 现在，移动字符串区中的字符串地址，

地址	目的码	源程序		注　释
2987:	23	INC	HL	; 把它传送到文字库项的第二和第三字节。
2988:	70	LD	(HL), B	; 保存名字的第一个字符。
2989:	69	LD	L, C	; 然后设置 HL,
298A:	60	LD	H, B	; 这样它就指向传送到字符串区的最后一个字符串的首址。
298B:	2B	DEC	HL	; 一直循环到再也找不到文字库项时为止,
298C:	C3E928	JP	28E9H	; 这是指那些必须要传送到字符串区的文字库项。
298F:	C5	PUSH	BC	; 字符串相加。联接由表达式计算调用的两个字符串。★★★★★★★★★★★★★★★★★★★★★
2990:	E5	PUSH	HL	; 保存最后一个操作数/最后一个代号的优先值，以及代码串地址。
2991:	2A2141	LD	HL, (4121H)	; 堆栈＝字符串 1 的地址,
2994:	E3	EX	(SP), HL	; HL＝输入代码串中的当前位置。
2995:	CD9F24	CALL	249FH	; 寻找下一个变量。
2998:	E3	EX	(SP), HL	; HL=(4121H), 堆栈＝代码串的地址。
2999:	CDF40A	CALL	0AF4H	; 确认它是一个字符串。
299C:	7E	LD	A, (HL)	; A＝字符串 1 的长度。
299D:	E5	PUSH	HL	; 保存字符串 1 的地址。
299E:	2A2141	LD	HL, (4121H)	; HL＝字符串 2 的地址。
29A1:	E5	PUSH	HL	; 把字符串 2 的地址放入堆栈。
29A2:	86	ADD	A, (HL)	; A＝字符串 1 的长度＋字符串 2 的长度。
29A3:	1E1C	LD	E, 1CH	; 如果进位标志置位，则输出代码为 1CH 的错误信息(LS)。
29A5:	DAA219	JP	C, 19A2H	; 说明合并后的字符串长度已超出 256。
29A8:	CD5728	CALL	2857H	; 确认对这两个字符串有足够的空间存放。
29AB:	D1	POP	DE	; DE＝字符串 2 的地址。
29AC:	CDDE29	CALL	29DEH	; 如有必要,要修改字符串 2 用的字符串区。
29AF:	E3	EX	(SP), HL	; HL＝字符串 1 的地址。
29B0:	CDDD29	CALL	29DDH	; 如有必要,要修改字符串 1 用的字符串区。
29B3:	E5	PUSH	HL	; 保存字符串 1 的地址。
29B4:	2AD440	LD	HL, (40D4H)	; 取字符串 2 的地址。
29B7:	EB	EX	DE, HL	; DE＝第二个字符串的地址。
29B8:	CDC629	CALL	29C6H	; 把字符串 1 从堆栈移到字符串工作区。
29BB:	CDC629	CALL	29C6H	; 传送字符串 2。
29BE:	214923	LD	HL, 2349H	; 把表达式计算中的继续地址放到堆栈。
29C1:	E3	EX	(SP), HL	; 代码串的地址放到 HL。
29C2:	E5	PUSH	HL	; 保存代码串的地址。
29C3:	C38428	JP	2884H	; 把(字符串 1 + 字符串 2)作为文字库表中的项保存起来。★★★★★★★★★★★★★★★★★★★★★
29C6:	E1	POP	HL	; HL＝返回地址，堆栈＝字符串地址。★★★★★★★
29C7:	E3	EX	(SP), HL	; 堆栈＝返回地址，HL＝字符串地址。★★★用堆栈子

地址	目的码	源程序		注　释
				程序 ★★★ 来传送。在入口时,堆栈＝个数/源地址,
				DE＝目的地址 ★★★★★★★★★★★★★★★★★★
29C8:	7E	LD	A, (HL)	; A＝要传送的字符数。
29C9:	23	INC	HL	; HL 加 1, 指向地址的 LSB。
29CA:	4E	LD	C, (HL)	; C＝地址的 LSB。
29CB:	23	INC	HL	; HL 加 1, 指向地址的 MSB。
29CC:	46	LD	B, (HL)	; B＝地址的 MSB。BC＝地址。
29CD:	6F	LD	L, A	; L＝要传送的字节数。
29CE:	2C	INC	L	; 对 L 作加 1 和减 1 的操作,以便设置状态标志。
29CF:	2D	DEC	L	; 对已传送的字符进行计数。
29D0:	C8	RET	Z	; 如果所有的字符已传送完,则返回。

说明 把 L 个字符从 (DE) 传送到 (BC)。

29D1:	0A	LD	A, (BC)	; 取一个字符。
29D2:	12	LD	(DE), A	; 存一个字符。
29D3:	03	INC	BC	; BC 加 1, 指向源地址。
29D4:	13	INC	DE	; DE 加 1, 指向目的地址。
29D5:	18F8	JR	29CFH	; 循环。
29D7:	CDF40A	CALL	0AF4H	; ★★★ 继续 VAL, FRE 和 PRINT 处理 ★★★★★★
				测试当前值,看它是否为字符串,如是数字则为错误。
29DA:	2A2141	LD	HL, (4121H)	; HL＝当前字符串的地址。
29DD:	EB	EX	DE, HL	; 把地址送入 DE。
29DE:	CDF529	CALL	29F5H	; 测试:当前变量是否也是最后的字符串库项。
29E1:	EB	EX	DE, HL	;
29E2:	C0	RET	NZ	; 不是,则返回,并且 DE＝当前变量的地址。
29E3:	D5	PUSH	DE	; 是,当前变量是定义的最后一个字符串。
29E4:	50	LD	D, B	; 把字符串地址放到 DE 中,
29E5:	59	LD	E, C	;
29E6:	1B	DEC	DE	; 并保存于堆栈。
29E7:	4E	LD	C, (HL)	; C＝当前字符串中字符的个数。
29E8:	2AD640	LD	HL, (40D6H)	; HL＝当前字符串指针。
29EB:	DF	RST	18H	; 当前的字符串是字符串区中定义的最后一个吗?
29EC:	2005	JR	NZ, 29F3H	; 不是,转移。
29EE:	47	LD	B, A	; 是,修改当前字符串指针。
29EF:	09	ADD	HL, BC	; HL＝字符串地址＋长度＝新的字符串指针地址。
29F0:	22D640	LD	(40D6H), HL	; 存入新的字符串指针地址。
29F3:	E1	POP	HL	; HL＝当前字符串的地址。
29F4:	C9	RET		; 返回到调用它的程序。
29F5:	2AB340	LD	HL, (40B3H)	; HL＝下一个可用的字符串单元的地址。★★★★★
29F8:	2B	DEC	HL	; 现在,把前一字符串地址装进 BC。
29F9:	46	LD	B, (HL)	;

地址	目的码	源程序		注释
29FA:	2B	DEC	HL	;
29FB:	4E	LD	C, (HL)	;
29FC:	2B	DEC	HL	; 然后,把那一项的地址与当前变量(在DE中不论是什么
29FD:	DF	RST	18H	; 变量)地址作比较。
29FE:	C0	RET	NZ	; 如果不相等,则返回,否则重新设置指针 (40B3H) 使它指向当前(最后)的项。
29FF:	22B340	LD	(40B3H),HL	; 把指针修改成 LSPT 中的当前项。
2A02:	C9	RET		; 返回到调用它的程序。
2A03:	01F827	LD	BC, 27F8H	; 把 POS 子程序的继续地址放到堆栈。★★★ LEN 子程序 ★★★★★★★★★★★★★★★★★★★★
2A06:	C5	PUSH	BC	; 把 27FBH 推入堆栈。
2A07:	CDD729	CALL	29D7H	; 把当前字符串指针的地址放入 HL 中。
2A0A:	AF	XOR	A	; 清除状态标志,并使 A 为 0。
2A0B:	57	LD	D, A	; 使 D 为 0。
2A0C:	7E	LD	A, (HL)	; A=从字符串指针区得到的字符串长度。
2A0D:	B7	OR	A	; 为长度设置状态标志。
2A0E:	C9	RET		; 若不是由 2A07H 进入的,那末转入 POS 子程序的继续地址。
2A0F:	01F827	LD	BC, 27F8H	; 把继续地址 27F8H 放入堆栈。★★★ ASC 子程序★
2A12:	C5	PUSH	BC	; 把 BC 中的值作为当前值保存起来。
2A13:	CD072A	CALL	2A07H	; 把当前字符串指针的地址放进 HL, 长度放进 A 中。
2A16:	CA4A1E	JP	Z, 1E4AH	; 如字符串长度 =0, 则有错误。
2A19:	23	INC	HL	; 现在把字符串地址装入 DE。
2A1A:	5E	LD	E, (HL)	; E=字符串地址的 LSB。
2A1B:	23	INC	HL	; HL加 1, 指向 MSB。
2A1C:	56	LD	D, (HL)	; D=字符串地址的 MSB。
2A1D:	1A	LD	A, (DE)	; A=字符串的第一个字符。
2A1E:	C9	RET		; 返回到调用它的程序。
2A1F:	3E01	LD	A, 01H	; A=待建立字符串的长度。★★★CHR$ 子程序 ★★
2A21:	CD5728	CALL	2857H	; 把长度和字符的值保存在 40D3H 单元。
2A24:	CD1F2B	CALL	2B1FH	; 把值变换成整数,保存在 DE 中。
2A27:	2AD440	LD	HL, (40D4H)	; HL=暂存字符串的地址。
2A2A:	73	LD	(HL), E	; 把值存在字符串区中。
2A2B:	C1	POP	BC	; 清除堆栈。
2A2C:	C38428	JP	2884H	; 把字符串从文字库传送到字符串区。返回解释程序。
2A2F:	D7	RST	10H	; ★★★ STRING$ 子程序 ★★★★★★★★★★★★
2A30:	CF	RST	08H	; 测试下一个字符是否是'('。
2A31:	28CD	JR	Z, 2A00H	; 2A31H: 给出左括号的代码 28H——) '('
2A33:	1C	INC	E	; 2A32H: CALL 2B1CH; 计算表达式——得到 N。
2A34:	2B	DEC	HL	; 把代码串的指针后移。
2A35:	D5	PUSH	DE	; 保存整数值 N。

地址	目的码	源程序		注释
2A36:	CF	RST	08H	; 测试下一个字符是否是逗号。
2A37:	2C	INC	L	; 2A37H: 给出逗号的代码 2CH——','
2A38:	CD3723	CALL	2337H	; 计算表达式,得到字符的值。
2A3B:	CF	RST	08H	; 测试下一个字符是否是')'。
2A3C:	29	ADD	HL, HL	; 2A3CH: 给出右括号的代码 29H——')'
2A3D:	E3	EX	(SP), HL	; HL=整数值 N, 堆栈=下一个代码串的地址。
2A3E:	E5	PUSH	HL	; 接着把 N 也推入堆栈。
2A3F:	E7	RST	20H	; 测试当前值的数据类型。
2A40:	2805	JR	Z, 2A47H	; 如是字符串,则转移。
2A42:	CD1F2B	CALL	2B1FH	; 把值变换成整数。留在 DE 和 WRA1 中。
2A45:	1803	JR	2A4AH	; 跳过装入字符串地址和第一个字符的程序。
2A47:	CD132A	CALL	2A13H	; A=待重复的字符。
2A4A:	D1	POP	DE	; DE=STRING$ (N, 'X') 调用中的N值。
2A4B:	F5	PUSH	AF	; 保存字符。
2A4C:	F5	PUSH	AF	; 再保存一个字符。
2A4D;	7B	LD	A, E	; A=重复数。
2A4E:	CD5728	CALL	2857H	; 把N个字节分配在字符串区。把分配区地址保存于 40D4H—40D5H。
2A51:	5F	LD	E, A	; E=重复的个数。
2A52:	F1	POP	AF	; A=要重复的字符。
2A53:	1C	INC	E	; 设置状态标志。
2A54:	1D	DEC	E	; DE 减 1, 对送出的要重复的字符进行计数。
2A55:	28D4	JR	Z, 2A2BH	; 如果不再要重复了,则转移。
2A57:	2AD440	LD	HL, (40D4H)	; HL=已分配在字符串区中的地址。
2A5A:	77	LD	(HL), A	; 传送字符。
2A5B:	23	INC	HL	; 指向下一个字符串地址。
2A5C;	1D	DEC	E	; 重复次数的计数。
2A5D:	20FB	JR	NZ, 2A5AH	; 循环,直到把同一字符传送了 N 次为止。
2A5F:	18CA	JR	2A2BH	; 返回到调用它的程序。
2A61:	CDDF2A	CALL	2ADFH	; 测试是否为')'。★★★ LEFT$ 子程序 ★★★★★★

说明 设置寄存器——在入口时: HL=LEFT$ 的地址,堆栈=字符串地址,
堆栈+1=*n* DE=代码串地址。

2A64:	AF	XOR	A	; 清除 A 和状态标志。
2A65:	E3	EX	(SP), HL	; HL=n 的地址。堆栈=当前代码串地址。
2A66:	4F	LD	C, A	; C 清零。
2A67:	3EE5	LD	A, E5H	; 2A68H: LD H, L
2A69:	E5	PUSH	HL	; 保存字符串地址。
2A6A:	7E	LD	A, (HL)	; 取字符串的长度。
2A6B:	B8	CP	B	; 与要送回的字节数作比较。
2A6C:	3802	JR	C, 2A70H	; 如果需要的字节超过了字符串的大小,则转移。
2A6E:	78	LD	A, B	; 保存要送回的字节数。

地址	目的码	源程序		注　释
2A6F:	110E00	LD	DE, 000EH	; 2A70H: LD C, 00
2A72:	C5	PUSH	BC	; 保存要送回的字符串长度。
2A73:	CDBF28	CALL	28BFH	; 确认有空地方供新字符串用。在 DE 中得到下一个字符串区的地址。
2A76:	C1	POP	BC	; BC=待送回的字符串的长度。
2A77:	E1	POP	HL	; HL=字符串地址。
2A78:	E5	PUSH	HL	; 把字符串地址推入堆栈。
2A79:	23	INC	HL	; 跳过字符计数。
2A7A:	46	LD	B, (HL)	; B=字符串地址的 LSB。
2A7B:	23	INC	HL	; 跳到 MSB。
2A7C:	66	LD	H, (HL)	; H=字符串地址的 MSB。
2A7D:	68	LD	L, B	; HL=字符串地址。
2A7E:	0600	LD	B, 00H	; B=00, C=所要的字符串的长度。
2A80:	09	ADD	HL, BC	; HL=待传送的最后一个字符的结束地址。
2A81:	44	LD	B, H	; 现在,把传送结束地址放进 BC。
2A82:	4D	LD	C, L	;
2A83:	CD5A28	CALL	285AH	; 保存文字库中下一个可用单元的长度 (A) 和开始地址 (DE)。
2A86:	6F	LD	L, A	; L=要传送的字符数。
2A87:	CDCE29	CALL	29CEH	; 把 (L) 个字符从 (BC) 传送到 (DE)。
2A8A:	D1	POP	DE	; 清除堆栈。
2A8B:	CDDE29	CALL	29DEH	; 把文字库字符串的地址放入 40D3H。
2A8E:	C38428	JP	2884H	; 把字符串传送到字符串区。返回解释程序。
2A91:	CDDF2A	CALL	2ADFH	; 设置寄存器。★★★ RIGHT$ 子程序 ★★★★★★★
2A94:	D1	POP	DE	; 取回字符串地址。
2A95:	D5	PUSH	DE	; 重新存入堆栈。
2A96:	1A	LD	A, (DE)	; A=字符串中的字符数。
2A97:	90	SUB	A, B	; 减去要分离出去的字节数。
2A98:	18CB	JR	2A65H	; 使用 LEFT$ 子程序。
2A9A:	EB	EX	DE, HL	; HL=代码串地址。★★★ MID$ 子程序★★★★★★
2A9B:	7E	LD	A, (HL)	; A=终止字符。
2A9C:	CDE22A	CALL	2AE2H	; BC=位置, DE=字符串地址。
2A9F:	04	INC	B	; 设置状态标志为相应位置的值。
2AA0:	05	DEC	B	;
2AA1:	CA4A1E	JP	Z, 1E4AH	; 如起始位置为 0, 则出错。
2AA4:	C5	PUSH	BC	; 保存起始位置。
2AA5:	1EFF	LD	E, FFH	; 在不给出字节数的情况下, E=256。
2AA7:	FE29	CP	29H	; 测试在 P 后面的是否是右括号。
2AA9:	2805	JR	Z, 2AB0H	; 如果没有给出字节的个数,则转移。
2AAB:	CF	RST	08H	; 否则,测试下一个输入值是否是逗号。
2AAC:	2C	INC	L	; 2AACH: 给出逗号的代码 2CH——','

地址	目的码	源程序		注　释
2AAD:	CD1C2B	CALL	2B1CH	; 计算表达式。把字节数作为整数放入 DE。
2AB0:	CF	RST	08H	; 测试下一个字符是否是')'。
2AB1:	29	ADD	HL, HL	; 2AB1H: 给出右括号的代码 29H——')'
2AB2:	F1	POP	AF	; A=起始位置。
2AB3:	E3	EX	(SP), HL	; HL=字符串地址。堆栈=当前代码串地址。
2AB4:	01692A	LD	BC, 2A69H	; 在 LEFT$ 中 MID$ 处理的继续地址。
2AB7:	C5	PUSH	BC	; 把地址放入堆栈。
2AB8:	3D	DEC	A	; 起始位置减 1。
2AB9:	BE	CP	(HL)	; 把起始位置与字符串长度作比较。
2ABA:	0600	LD	B, 00H	; B=00。
2ABC:	D0	RET	NC	; 如果起始位置减 1 大于字符串长度，则在 2A69H 继续执行。
2ABD:	4F	LD	C, A	; C=起始位置减 1。
2ABE:	7E	LD	A, (HL)	; A=字符串的长度。
2ABF:	91	SUB	A, C	; A=P 和字符串末尾之间的字符数。
2AC0:	BB	CP	E	; 与要送回的字符数作比较。
2AC1:	47	LD	B, A	; B=送回的字符数。
2AC2:	D8	RET	C	; 如果需要的字符数大于字符串中的字符数，则在2A69H继续执行，
2AC3:	43	LD	B, E	; 否则，送回需要的字符数。
2AC4:	C9	RET		; 在 2A69H 处继续执行。
2AC5;	CD072A	CALL	2A07H	; 把字符串长度放入 A 寄存器，字符串指针地址放入 HL 中。★★★ VAL 子程序 ★★★★★★★★★★★★
2AC8:	CAF827	JP	Z, 27F8H	; 如果长度 =0，则返回。
2ACB:	5F	LD	E, A	; 把长度传送到 E 内，D=0。
2ACC:	23	INC	HL	; HL 加 1，跳过长度。
2ACD:	7E	LD	A, (HL)	; A=字符串地址的 LSB。
2ACE:	23	INC	HL	; HL 加 1，指向地址的 MSB。
2ACF:	66	LD	H, (HL)	; H=字符串地址的 MSB。
2AD0:	6F	LD	L, A	; 现在，HL=字符串地址。
2AD1:	E5	PUSH	HL	; 保存字符串地址。
2AD2:	19	ADD	HL, DE	; 然后加上长度，由此可得 HL=结束地址。
2AD3:	46	LD	B, (HL)	; 保存字符串的最后一个字符。
2AD4:	72	LD	(HL), D	; 用一个 0 来代替它。
2AD5:	E3	EX	(SP), HL	; 堆栈=字符串的结束地址，HL=字符串的开始地址。
2AD6:	C5	PUSH	BC	; 保存字符串的被替换的字符。
2AD7:	7E	LD	A, (HL)	; A=字符串的第一个字符。
2AD8:	CD650E	CALL	0E65H	; 把字符串开始处的数字，从 ASCII 变换成二进制。
2ADB:	C1	POP	BC	; B=被替换的字符。
2ADC:	E1	POP	HL	; HL=字符串的结束地址。
2ADD:	70	LD	(HL), B	; 恢复被替换的字符。

地址	目的码	源程序		注　释
2ADE:	C9	RET		; 返回到 BASIC。
2ADF:	EB	EX	DE, HL	; DE=调用子程序的地址，HL=代码串的地址。

说明 被 LEFT$, MID$ 和 RIGHT$ 所调用，用以测试结束的')'。

入口：堆栈=代码串地址。
　　字节数。
　　返回地址。

出口：HL=代码串地址。
　　DE=要分离的字节数。
　　B=字节数。

2AE0:	CF	RST	08H	; 寻找跟在参数后面的右括号。
2AE1:	29	ADD	HL, HL	; 2AE1H: 给出右括号的代码 29H——')'
2AE2:	C1	POP	BC	; 返回地址。
2AE3:	D1	POP	DE	; DE=要分离的字节数。
2AE4:	C5	PUSH	BC	; 再存返回地址。
2AE5:	43	LD	B, E	; B=字节个数。
2AE6:	C9	RET		; HL=代码串地址。
2AE7:	FE7A	CP	7AH	; 测试代号是否在范围内。★★★★★★★★★★★★★
2AE9:	C29719	JP	NZ, 1997H	; 如结果不为 0，则 SN 错误。如代号 =>FAH，则有错误。
2AEC:	C3D941	JP	41D9H	; 磁盘 BASIC 出口。让磁盘 BASIC 处理 TAB—MID$。
2AEF:	CD1F2B	CALL	2B1FH	; 把口编号放入 A 寄存器。★★★ INP 子程序 ★★★
2AF2:	329440	LD	(4094H), A	; 保存口编号。
2AF5:	CD9340	CALL	4093H	; 去执行 IN ×× 指令。返回到执行驱动程序。
2AF8:	C3F827	JP	27F8H	; 计算表达式。★★★ OUT 子程序 ★★★★★★★★
2AFB:	CD0E2B	CALL	2B0EH	; 值放到 A 寄存器。口编号放入 4094H 和 4097H。★
2AFE:	C39640	JP	4096H	; 去执行 OUT ×× 指令。返回到执行驱动程序。
2B01:	D7	RST	10H	; 把输入代码串中的下一个字符送入 A 中。★★★计算表达式，把结果作为整数留在 DE 中★★★
2B02:	CD3723	CALL	2337H	; 计算表达式。把结果放到 WRA1 中。
2B05:	E5	PUSH	HL	; 下一个代码串地址。
2B06:	CD7F0A	CALL	0A7FH	; 把结果变换成整数。把它放在 HL 中。
2B09:	EB	EX	DE, HL	; DE=结果(整数形式)。
2B0A:	E1	POP	HL	; 恢复输入代码串中的位置。
2B0B:	7A	LD	A, D	; 结果的 MSB 放到 A。
2B0C:	B7	OR	A	; 返回到调用它的程序。
2B0D:	C9	RET		; 送回结果的符号和零 (Z) 标志。
2B0E:	CD1C2B	CALL	2B1CH	; 计算表达式。得到口编号。★★★ OUT 子程序的继续 ★★★★★★★★★★★★★★★★★★★★★★★
2B11:	329440	LD	(4094H), A	; 把口编号保存在 DOS 地址 4094H 中。
2B14:	329740	LD	(4097H), A	; 把口编号保存在 DOS 地址 4097H 中。
2B17:	CF	RST	08H	; 测试下一个字符是否是单引号。
2B18:	2C	INC	L	; 2B18H: 给出单引号的代码 2CH——(')
2B19:	1801	JR	2B1CH	; 跳过 PRINT TAB 入口点。
2B1B:	D7	RST	10H	; 检查下一个字符(被 PRINT TAB 调用)。

地址	目的码	源程序		注　释
2B1C:	CD3723	CALL	2337H	; 计算表达式,得到值。
2B1F:	CD052B	CALL	2B05H	; 把指数结果变换成整数,把它装进 DE。使 A=MSB。
2B22:	C24A1E	JP	NZ, 1E4AH	; 若值 >255, 则为 FC 错误。
2B25:	2B	DEC	HL	; 输入字符串指针后退。
2B26:	D7	RST	10H	; 从输入字符串得到下一个字符 (HL 加 1 和复位各标志)。
2B27:	7B	LD	A, E	; 结果的 LSB 放到 A。
2B28:	C9	RET		; 返回到调用它的程序。
2B29:	3E01	LD	A, 01H	; 打印机的设备类型。★★★ LLIST 子程序 ★★★★★
2B2B:	329C40	LD	(409CH), A	; 设定当前输出设备为打印机。
2B2E:	C1	POP	BC	; 从堆栈中清除返回地址。★★★ LIST 子程序 ★★★
2B2F:	CD101B	CALL	1B10H	; 在返回时取得 LIST 的行号范围。 BC=第一行的地址,堆栈=最后一行的地址。
2B32:	C5	PUSH	BC	; 保存开始行指针。
2B33:	21FFFF	LD	HL, FFFFH	; 设定当前行号为-1。
2B36:	22A240	LD	(40A2H), HL	; 保存在存放当前行号的单元。
2B39:	E1	POP	HL	; HL=待列出的第一行的地址。
2B3A:	D1	POP	DE	; DE=待列出的最后一行的地址。
2B3B:	4E	LD	C, (HL)	; 现在,取指向下一行的指针。
2B3C:	23	INC	HL	; C 含有指向下一行的指针的 LSB。
2B3D:	46	LD	B, (HL)	; B=下一行指针的 MSB。
2B3E:	23	INC	HL	; HL=当前行(行号)的第一个字节的地址。
2B3F:	78	LD	A, B	; 如果指向下一行的指针为 0, 则已找到了程序的结束。
2B40:	B1	OR	C	; 检验是否是程序的结束。
2B41:	CA191A	JP	Z, 1A19H	; 如已结束,则返回到 READY 子程序。
2B44:	CDDF41	CALL	41DFH	; DOS 出口 (JP 579CH)。
2B47:	CD9B1D	CALL	1D9BH	; 测试键盘输入。如打入 SHIFT @ 键,则暂停, 当打入任意一个键时,则返回。
2B4A:	C5	PUSH	BC	; 保存待打印的下一行地址。
2B4B:	4E	LD	C, (HL)	; 取当前行行号的 LSB。
2B4C:	23	INC	HL	; HL 加 1, 指向行号的下一个字节。
2B4D:	46	LD	B, (HL)	; 取当前行号的 MSB。
2B4E:	23	INC	HL	; HL=当前行程序语句的第一个字节。
2B4F:	C5	PUSH	BC	; 把当前行的行号(二进制)保存于堆栈。
2B50:	E3	EX	(SP), HL	; 重新排列: 堆栈=程序第一字节的地址, HL=二进制行号。
2B51:	EB	EX	DE, HL	; DE=当前行的地址, HL=要列出的程序的最后一行地址。
2B52:	DF	RST	18H	; 测试一下,看看是否所有的行已经列出了。
2B53:	C1	POP	BC	; BC=当前行的第一字节的地址。
2B54:	DA181A	JP	C, 1A18H	; 如果所有的行均已列出,则返回到输入状态。

地址	目的码	源 程 序		注 释
2B57:	E3	EX	(SP), HL	; HL=待打印的最后一行的地址，堆栈=当前行的行号。
2B58:	E5	PUSH	HL	; 保存当前行的地址。
2B59:	C5	PUSH	BC	; 保存当前行的行号(二进制)。
2B5A:	EB	EX	DE, HL	; HL=当前行的地址。
2B5B:	22EC40	LD	(40ECH), HL	; 存入有错误的行号地址。
2B5E:	CDAF0F	CALL	0FAFH	; 输出一个以 ASCII 形式表示的行号。
2B61:	3E20	LD	A, 20H	; A=ASCII 形式的空格值。
2B63:	E1	POP	HL	; HL=当前行的行号。
2B64:	CD2A03	CALL	032AH	; 把一个空格送到荧光屏,磁带机或打印机。
2B67:	CD7E2B	CALL	2B7EH	; 把当前行传送到工作区 (40A7H),并把它扩展。
2B6A:	2AA740	LD	HL, (40A7H)	; HL=被扩展行的地址。
2B6D:	CD752B	CALL	2B75H	; 把缓冲区的内容送到荧光屏显示(显示当前行)。
2B70:	CDFE20	CALL	20FEH	; 用回车换行结束一行。
2B73:	18BE	JR	2B33H	; 循环到所有行都显示完为止。
2B75:	7E	LD	A, (HL)	; 由 HL 所指出的输出区。★★★★★★★★★★★★
2B76:	B7	OR	A	; 取要显示的下一个字符。
2B77:	C8	RET	Z	; 如信息结束,则返回。
2B78:	CD2A03	CALL	032AH	; 显示 (HL)。
2B7B:	23	INC	HL	; HL 加 1，指向下一个字符。
2B7C:	18F7	JR	2B75H	; 继续显示,直到 (HL)=0 为止。
2B7E:	E5	PUSH	HL	; 保存待传送的行地址。★★★ 被 LIST 和 EDIT 调用 ★★★★★★★★★★★★★★★★★★★★★★★★

说明 根据 HL，移动行指针到输入缓冲区。把每个代号扩展为它的关键字。

2B7F:	2AA740	LD	HL, (40A7H)	; HL=输入缓冲器的地址。
2B82:	44	LD	B, H	; 把这个地址放到 BC 中，
2B83:	4D	LD	C, L	; 在 BC 中这个地址将用作为被扩展行的输出缓冲区。
2B84:	E1	POP	HL	; 恢复待传送/待扩展行的地址。
2B85:	16FF	LD	D, FFH	; D=一行中最大字符数。
2B87:	1803	JR	2B8CH	; 转移到传送/扩展的代码的中间。
2B89:	03	INC	BC	; 指向打印/工作缓冲区中的下一个位置。
2B8A:	15	DEC	D	; 已传送字符的计数。
2B8B:	C8	RET	Z	; 如已传送 256 个字符,则返回。
2B8C:	7E	LD	A, (HL)	; 从程序语句表 (PST) 得到一个字符。
2B8D:	B7	OR	A	; 设定状态标志,于是能进行 EOS 或代号的测试。
2B8E:	23	INC	HL	; 指向代码串中的下一个字符。
2B8F:	02	LD	(BC), A	; 把刚得到的字符放入打印/工作缓冲区。
2B90:	C8	RET	Z	; 如是 EOS (语句结束),则返回。
2B91:	F2892B	JP	P, 2B89H	; 如果字符不是一个代号(不必进行扩展)，则转移，去取下一个字符。
2B94:	FEFB	CP	FBH	; 测试是否是单引号的代号。

地址	目的码	源程序		注　释
2B96:	2008	JR	NZ, 2BA0H	; 不是单引号的代号,则去搜索全部语法规定的保留字表
2B98:	0B	DEC	BC	; 以便找出这个代号。
2B99:	0B	DEC	BC	; 如是单引号的代号,则已扩展的缓冲器指针后移。
2B9A:	0B	DEC	BC	; 4 个字节的位置。
2B9B:	0B	DEC	BC	;
2B9C:	14	INC	D	; 然后调整缓冲区内字符计数值,共调整 4 次。
2B9D:	14	INC	D	;
2B9E:	14	INC	D	;
2B9F:	14	INC	D	;
2BA0:	FE95	CP	95H	; 测试是否是 ELSE 代号。
2BA2:	CC240B	CALL	Z,0B24H	; 如是 ELSE 代号,则后移扩展过的缓冲器指针。
2BA5:	D67F	SUB	7FH	; A=我们在保留字表中找到的那一项的编号。
2BA7:	E5	PUSH	HL	; 保存当前代码串地址。
2BA8:	5F	LD	E, A	; E=要跳过的项数。
2BA9:	215016	LD	HL, 1650H	; HL=保留字表的指针。
2BAC:	7E	LD	A, (HL)	; 从保留字表中取一个字节。
2BAD:	B7	OR	A	; 设置状态,以测试开始项。
2BAE:	23	INC	HL	; HL 加 1, 指向保留字表中的下一个字。
2BAF:	F2AC2B	JP	P, 2BACH	; 如果不是开始项,则转移。
2BB2:	1D	DEC	E	; 跳过一项,计数减 1。

说明 扫描保留字表,寻找第 *n* (E 寄存器)项。每项长度是可变的,并由符号位为 1 的字节开始,项本身是以 ASCII 表示的保留字,我们正在寻找的就是这个 ASCII 表示的保留字。

2BB3:	20F7	JR	NZ, 2BACH	; 如没有跳过足够的项,则转移。
2BB5:	E67F	AND	7FH	; 清除项的第一个字中的符号位。
2BB7:	02	LD	(BC), A	; 把保留字 (ASCII 形式)的一个字节传送到打印/工作缓冲区。
2BB8:	03	INC	BC	; 指向下一个工作缓冲器地址。
2BB9:	15	DEC	D	; 对已传送到打印缓冲器的所有字符进行计数。
2BBA:	CAD828	JP	Z, 28D8H	; 如已传送 256 个字符, 则转移。(清除在 2BA7H 处推入堆栈的内容以后,返回到调用它的程序。)
2BBD:	7E	LD	A, (HL)	; 从保留字表取下一个字。
2BBE:	23	INC	HL	; HL 加 1, 指向保留字表的下一项。
2BBF:	B7	OR	A	; 设置状态标志,于是我们能对这个字的结束作测试。
2BC0:	F2B72B	JP	P, 2BB7H	; 如果没有结束,则转移——把字符的剩余部分传送到打印/工作缓冲器。
2BC3:	E1	POP	HL	; 恢复代码串地址。
2BC4:	18C6	JR	2B8CH	; 继续扫描/传送代码串。
2BC6:	CD101B	CALL	1B10H	; 得到要删除的行号范围。★★★ DELETE 子程序★★
2BC9:	D1	POP	DE	; DE=以二进制表示的结束行号。
2BCA:	C5	PUSH	BC	; BC=在程序表区内的开始行的地址。
2BCB:	C5	PUSH	BC	; 把它存入堆栈两次。

地址	目的码	源程序		注　释
2BCC:	CD2C1B	CALL	1B2CH	; 得到要删除的结束行的地址。DE=要寻找的结束行号。
2BCF:	3005	JR	NC, 2BD6H	; 如果没有找到结束行号,则转移。
2BD1:	54	LD	D, H	; 把下一行从 HL 传送到 DE。
2BD2:	5D	LD	E, L	; (下一行指的是待删除的最后一行后面的那一行)。
2BD3:	E3	EX	(SP), HL	; 把最后一行的结束地址加 1 推入堆栈, HL= 待删除的第一行的地址。
2BD4:	E5	PUSH	HL	; 保存待删除的第一行的地址。
2BD5:	DF	RST	18H	; 确认第一行地址<=最后一行的地址。
2BD6:	D24A1E	JP	NC, 1E4AH	; 如果进位标志为 0, 则给出 FC 错误。
2BD9:	212919	LD	HL, 1929H	; HL='READY' 信息的地址。
2BDC:	CDA728	CALL	28A7H	; 把信息发送到系统的输出设备。
2BDF:	C1	POP	BC	; BC=待删除的第一行的地址。
2BE0:	21E81A	LD	HL, 1AE8H	; HL= 把待删除的最后一行后面所有的行移下去以后的继续地址。
2BE3:	E3	EX	(SP), HL	; 把返回地址保存到堆栈,于是能经 RET 返回。
2BE4:	EB	EX	DE, HL	; DE=待删除的最后一行的下一行的地址。
2BE5:	2AF940	LD	HL, (40F9H)	; HL=程序区中最后一行的结束地址。

说明 从地址在 DE 中的那一行开始,把所有的行向下移动。移动到地址在 BC 中的那一行。

2BE8:	1A	LD	A, (DE)	; 从被删除部分后面的程序行中取一个字节。
2BE9:	02	LD	(BC), A	; 把它移动到被删除部分的存储器中, BC = 当前存储地址。
2BEA:	03	INC	BC	; 存储地址加 1。
2BEB:	13	INC	DE	; 取出地址加 1。
2BEC:	DF	RST	18H	; 然后,把取出地址与程序区的结束地址作比较。
2BED:	20F9	JR	NZ, 2BE8H	; 如果不是所有的行均向下移动了,则转移。
2BEF:	60	LD	H, B	; 把程序最后一行的结束地址传送到程序的结束地址。
2BF0:	69	LD	L, C	; (即简单变量的开始地址)
2BF1:	22F940	LD	(40F9H), HL	;
2BF4:	C9	RET		; 返回到调用它的程序。
2BF5:	CD8402	CALL	0284H	; 写同步字节和其后的 A5H。★★★ CSAVE 子程序 ★
2BF8:	CD3723	CALL	2337H	; 计算 CSAVE 表达式的剩余部分。
2BFB:	E5	PUSH	HL	; 保存当前代码串地址。
2BFC:	CD132A	CALL	2A13H	; 把文件名的地址放进 DE。
2BFF:	3ED3	LD	A, D3H	; A=要写到盒式磁带上的字节。
2C01:	CD6402	CALL	0264H	; 写一个 'S', 但此 'S' 相应的 ASCII 码最高位 (符号位)为 1。
2C04:	CD6102	CALL	0261H	; 再写两个 'S'。
2C07:	1A	LD	A, (DE)	; 取待存储文件的名字。
2C08:	CD6402	CALL	0264H	; 把文件名写入盒式磁带(占一个字节)。
2C0B:	2AA440	LD	HL,(40A4H)	; HL=程序区的首址。

地址	目的码	源程序		注　释
2C0E:	EB	EX	DE, HL	; 把首址保存在 DE 中。
2C0F:	2AF940	LD	HL, (40F9H)	; HL=程序区的终址。
2C12:	1A	LD	A, (DE)	; 取机内存放的程序的一个字节。
2C13:	13	INC	DE	; DE 加 1，指向程序的下一个字节。
2C14:	CD6402	CALL	0264H	; 把当前字节写到盒式磁带上。
2C17:	DF	RST	18H	; 已经把全部程序写完了吗?
2C18:	20F8	JR	NZ, 2C12H	; 没有，循环。
2C1A:	CDF801	CALL	01F8H	; 是，关闭磁带机马达。
2C1D:	E1	POP	HL	; 恢复代码串地址。
2C1E;	C9	RET		; 返回到输入状态。
2C1F:	CD9302	CALL	0293H	; 打开马达，寻找同步码。★★★ CLOAD 子程序★★★
2C22:	7E	LD	A, (HL)	; 取跟在 CLOAD 后面的代号。
2C23:	D6B2	SUB	B2H	; 测试是否是 CLOAD?
2C25:	2802	JR	Z, 2C29H	; 如是 CLOAD?，则转移。
2C27:	AF	XOR	A	; 清除 A 和状态标志。
2C28:	012F23	LD	BC, 232FH	; 2C29H: CPL 如是 CLOAD? 则 A = −1，是 CLOAD，则 A=0。
2C2B:	F5	PUSH	AF	; 2C2AH: INC HL HL 加 1，指向文件名。
2C2C:	2B	DEC	HL	; 把代码串指针后退一个字节，因为 RST 10H 将向前跳一个字节。
2C2D:	D7	RST	10H	; 检验代码串的下一个元素。
2C2E:	3E00	LD	A, 0H	;（如没有文件名，则 A 清零。）
2C30:	2807	JR	Z, 2C39H	; 如没有指定文件名，则转移。
2C32:	CD3723	CALL	2337H	; 计算表达式，得到文件名。
2C35:	CD132A	CALL	2A13H	; 把文件名字符串的地址放到 HL 内。
2C38:	1A	LD	A, (DE)	; 得到搜索用的文件名。
2C39:	6F	LD	L, A	; 保存文件名。
2C3A:	F1	POP	AF	; 恢复 CLOAD, CLOAD? 标志。
2C3B:	B7	OR	A	; 设置 CLOAD 的类型状态。
2C3C:	67	LD	H, A	; CLOAD 类型标志作为当前值存入 WRA1 中。
2C3D:	222141	LD	(4121H), HL	; 将 CLOAD 类型标志和文件名存入 WRA1。
2C40:	CC4D1B	CALL	Z, 1B4DH	; 如是 CLOAD，则调用 NEW 子程序，以便初始化系统的变量。
2C43:	2A2141	LD	HL, (4121H)	; 恢复 CLOAD 类型标志和文件名。
2C46:	EB	EX	DE, HL	; 并保存在 D 寄存器内，E=文件名。
2C47:	0603	LD	B, 03H	; B=准备核对的字节数。
2C49:	CD3502	CALL	0235H	; 读一个字节。
2C4C:	D6D3	SUB	D3H	; 与符号位为 1 的 'S' 作比较。
2C4E:	20F7	JR	NZ, 2C47H	; 不一致，继续扫描直到找到三个 'S' 时为止。
2C50:	10F7	DJNZ	2C49H	; 循环三次。
2C52:	CD3502	CALL	0235H	; 已找到三个 'S' 就读文件名。

地址	目的码	源程序		注　释
2C55:	1C	INC	E	; 用户指定了文件名了吗?
2C56:	1D	DEC	E	; 设置状态,状态取决于文件名。
2C57:	2803	JR	Z, 2C5CH	; 如果没有给出文件名,则转移。装入第一个程序。
2C59:	BB	CP	E	; 把调用中的文件名与磁带上找到的那个文件名作比较。
2C5A:	2037	JR	NZ, 2C93H	; 如不一致,则跳到当前文件的结束。
2C5C:	2AA440	LD	HL,(40A4H)	; HL=程序区的首址。
2C5F:	0603	LD	B, 03H	; B = 要寻找的作为文件结束符的连续的 0H 的数目。
2C61:	CD3502	CALL	0235H	; 读磁带上程序的一个字节。
2C64:	5F	LD	E, A	; 为了存储到程序区需要保存起来。
2C65:	96	SUB	A, (HL)	; 与当前程序的相应字节作比较。
2C66:	A2	AND	D	; 如是 CLOAD? 则 D=FFH, 如是 CLOAD, 则为 00H。
2C67:	2021	JR	NZ, 2C8AH	; 如是 CLOAD? 并且两者不一致,则有错误。
2C69:	73	LD	(HL), E	; 否则,就是两者一致或 CLOAD。把刚读入的字节存储起来。
2C6A:	CD6C19	CALL	196CH	; 测试有无足够的存储空间。
2C6D:	7E	LD	A, (HL)	; 取刚读的字节。
2C6E:	B7	OR	A	; 测试是否为 0。
2C6F:	23	INC	HL	; 指向程序区中的下一个字。
2C70:	20ED	JR	NZ, 2C5FH	; 如果不是程序的结束或语句的结束 (EOS), 则转移。
2C72:	CD2C02	CALL	022CH	; '*' 闪亮。
2C75:	10EA	DJNZ	2C61H	; 寻找作为程序结束的 3 个 00H, 否则就是 EOS。
2C77:	22F940	LD	(40F9H), HL	; 存储程序的结束地址,给出变量的开始地址。
2C7A:	212919	LD	HL, 1929H	; HL='READY' 信息的地址。
2C7D:	CDA728	CALL	28A7H	; 显示 'READY'。
2C80:	CDF801	CALL	01F8H	; 关闭盒式磁带机。
2C83:	2AA440	LD	HL, (40A4H)	; HL=程序的开始地址。
2C86:	E5	PUSH	HL	; 保存在堆栈。
2C87:	C3E81A	JP	1AE8H	; 返回到输入状态,在新的一行输入的结束处开始执行。
2C8A:	21A52C	LD	HL, 2CA5H	; HL='BAD' 信息的地址。
2C8D:	CDA728	CALL	28A7H	; 把信息发送到系统的输出设备。
2C90:	C3181A	JP	1A18H	; 重新初始化 BASIC 解释程序,并且继续执行。
2C93:	323E3C	LD	(3C3EH), A	; 存放用于搜索的文件名。★★★ 搜索文件结束——3 字节的零 ★★★★★★★★★★★★★★★★★★★★
2C96:	0603	LD	B, 03H	; 要找的零的数目。
2C98:	CD3502	CALL	0235H	; 读一个字节。
2C9B:	B7	OR	A	; 设置状态,并测试是否为 0。
2C9C:	20F8	JR	NZ, 2C96H	; 不是 0, 取下一个字节。
2C9E:	10F8	DJNZ	2C98H	; 是 0, 寻找结束文件的 3 个 0。
2CA0:	CD9602	CALL	0296H	; 找到文件的结束,检查文件的同步码和引导码,
2CA3:	18A2	JR	2C47H	; 然后,测试前面的 'S', 核对文件名。
2CA5:	42	LD	B, D	; B ★★★ BAD 信息 ★★★★★★★★★★★★★★

地址	目的码	源程序		注释
2CA6:	41	LD	B, C	; A
2CA7:	44	LD	B, H	; D
2CA8:	0D	DEC	C	; 回车。
2CA9:	00	NOP		; 信息结束符。★★★★★★★★★★★★★★★★★★
2CAA:	CD7F0A	CALL	0A7FH	; 把要检查的存储单元的地址放进 HL。★★★ PEEK 子程序 ★★★★★★★★★★★★★★★★★★★★
2CAD:	7E	LD	A, (HL)	; 取被检查地址的值。
2CAE:	C3F827	JP	27F8H	; 作为当前值保存起来并返回到输入状态。
2CB1:	CD022B	CALL	2B02H	; 计算表达式,得到要改变的字节的地址。★★★ POKE 子程序 ★★★★★★★★★★★★★★★★★★★★
2CB4:	D5	PUSH	DE	; 保存要改变的字节的地址。
2CB5:	CF	RST	08H	; 测试后面的字节是否是逗号。
2CB6:	2C	INC	L	; 2CB6H: 给出逗号的代码 2CH——','
2CB7:	CD1C2B	CALL	2B1CH	; 计算表达式,把待存储的值放进 A 寄存器。
2CBA:	D1	POP	DE	; DE=要改变的字节的地址。
2CBB:	12	LD	(DE), A	; 存储新的字节。
2CBC:	C9	RET		; 返回到输入状态。
2CBD:	CD3823	CALL	2338H	; 计算试验表达式。★★★ PRINT USING 子程序 ★★
2CC0:	CDF40A	CALL	0AF4H	; 保证当前的数据类型是字符串。
2CC3:	CF	RST	08H	; 测试下一个字符是否是';'。
2CC4:	3B	DEC	SP	; 2CC4H: 给出分号的代码 3BH——';'。
2CC5:	EB	EX	DE, HL	; DE=下一个输入符号的地址。
2CC6:	2A2141	LD	HL, (4121H)	; HL=USING 字符串的地址。
2CC9:	1808	JR	2CD3H	; 去计算 USING 字符串。
2CCB:	3ADE40	LD	A, (40DEH)	; 取入 READ 标志。★★★★★★★★★★★★★★★★
2CCE:	B7	OR	A	; 根据标志,设置状态。
2CCF:	280C	JR	Z, 2CDDH	; 如果是与 INPUT 语句相反的 READ, 则转移。
2CD1:	D1	POP	DE	; 恢复代码串地址,并把它送到 HL 中。
2CD2:	EB	EX	DE, HL	; D=字符串的长度。
2CD3:	E5	PUSH	HL	; 保存说明格式的字符串的开始地址。
2CD4:	AF	XOR	A	; 使 A 和标志为 0。
2CD5:	32DE40	LD	(40DEH), A	; 清除 READ/INPUT 标志。继续 PRINT USING。
2CD8:	BA	CP	D	; 把字符串长度与 0 作比较。
2CD9:	F5	PUSH	AF	; 保存比较的结果。
2CDA:	D5	PUSH	DE	; 保存代码串的下一个输入符号。
2CDB:	46	LD	B, (HL)	; 把字符串长度放进 B 内。
2CDC:	B0	OR	B	; 设置标志,并确认它不是 0。
2CDD:	CA4A1E	JP	Z, 1E4AH	; 如 Z 标志置位, 则有 FC 错误。
2CE0:	23	INC	HL	; HL 加 1, 指向字符串的地址。
2CE1:	4E	LD	C, (HL)	; 把字符串地址的 LSB 放到 C 中。
2CE2:	23	INC	HL	; HL 加 1, 指向字符串地址的 MSB 的地址。

地址	目的码	源程序		注释
2CE3:	66	LD	H, (HL)	; H=字符串地址的 MSB。
2CE4:	69	LD	L, C	; HL=字符串的开始地址。
2CE5:	181C	JR	2D03H	; 去分析字段说明，B=要分析的字符数。返回到2D99H单元。
2CE7:	58	LD	E, B	; E=现在剩余的字符数。★★★ PRINT USING 的%号处理 ★★★★★★★★★★★★★★★★★★★★★★★★★
2CE8:	E5	PUSH	HL	; 保存字符串中的当前位置。
2CE9:	0E02	LD	C, 02H	; C=开头的和末尾的%的个数。
2CEB:	7E	LD	A, (HL)	; 现在，扫描字符串的剩余部分，寻找结束的%。
2CEC:	23	INC	HL	; 计数 C 中所有的空格数。
2CED:	FE25	CP	25H	; 当找到%或非空格时，则要出口。
2CEF:	CA172E	JP	Z, 2E17H	; 如是%，则转移。
2CF2:	FE20	CP	20H	; 测试是否是空格。
2CF4:	2003	JR	NZ, 2CF9H	; 如果不是空格，则转移。
2CF6:	0C	INC	C	; 计数一个空格。
2CF7:	10F2	DJNZ	2CEBH	; 循环，直到字符串结束，或遇到%，以及非空格时为止。
2CF9:	E1	POP	HL	; 我们已经处理完输入，或找到一个非空格字符。
2CFA:	43	LD	B, E	; 这时，恢复 HL 为开头的%后的第一个符号位置。
2CFB:	3E25	LD	A, 25H	; 把 B 恢复为剩下的符号数目，并继续执行。
2CFD:	CD492E	CALL	2E49H	; 在打印单个%以后打印'+'。
2D00:	CD2A03	CALL	032AH	; 打印 A 寄存器的内容。
2D03:	AF	XOR	A	; 清除 A 和标志。
2D04:	5F	LD	E, A	; 使 E 为0。
2D05:	57	LD	D, A	; 使 D 为0。(在小数点以前的#的个数。)
2D06:	CD492E	CALL	2E49H	; 如果需要，打印前面的+号。
2D09:	57	LD	D, A	; 使 D 为0。
2D0A:	7E	LD	A, (HL)	; A=取自字符串中的一个字段说明。
2D0B:	23	INC	HL	; HL 加1，指向下一个字符。
2D0C:	FE21	CP	21H	; 测试是否是!。
2D0E:	CA142E	JP	Z, 2E14H	; 如是!，则转移。
2D11:	FE23	CP	23H	; 测试是否是#号。
2D13:	2837	JR	Z, 2D4CH	; 如是#号，则转移。
2D15:	05	DEC	B	; 为处理过的字符进行计数。
2D16:	CAFE2D	JP	Z, 2DFEH	; 如字符串已处理完，则转移。
2D19:	FE2B	CP	2BH	; 测试是否是+号。
2D1B:	3E08	LD	A, 08H	; 为了使+作前导记号而设置的标志。
2D1D:	28E7	JR	Z, 2D06H	; 如是+号，则转移。
2D1F:	2B	DEC	HL	; 后退一个地址，于是能够重取当前的字符。
2D20:	7E	LD	A, (HL)	; 取当前的字符。
2D21:	23	INC	HL	; HL 加1，指向下一个字符。
2D22:	FE2E	CP	2EH	; 测试是否为小数点。

地址	目的码	源程序		注　释
2D24:	2840	JR	Z, 2D66H	; 如是小数点,则转移。
2D26:	FE25	CP	25H	; 测试是否是%。
2D28:	28BD	JR	Z, 2CE7H	; 如是%,则转移。
2D2A:	BE	CP	(HL)	; 现在,测试当前的字符是否和紧接着的字符相等。
2D2B:	20D0	JR	NZ, 2CFDH	; 如果不是,则跳过对两个连续的相同字符 $$ 的测试。
2D2D:	FE24	CP	24H	; 测试是否为 $$。
2D2F:	2814	JR	Z, 2D45H	; 如果当前字符和紧接着的字符都是 $, 则转移。
2D31:	FE2A	CP	2AH	; 不是 $$, 测试是否为**。
2D33:	20C8	JR	NZ, 2CFDH	; 不是*,则转移——继续扫描, 直到字符串处理完毕为止。
2D35:	78	LD	A, B	; A=在字符串内的剩余字符数。★★★ PRINT USING 语句的*号处理 ★★★★★★★★★★★★★★★★★★
2D36:	FE02	CP	02H	; 在字符串中至少应剩下两个字符,并且它们应是*$。
2D38:	23	INC	HL	; 指向下一个字符,它应该是一个 $。
2D39:	3803	JR	C, 2D3EH	; 如果不是剩下两个字符,则转移。
2D3B:	7E	LD	A, (HL)	; 取下一个字符。
2D3C:	FE24	CP	24H	; 测试是否是 $。
2D3E:	3E20	LD	A, 20H	; A=用于**的标志。置位 EDIT 标志中的位 5。
2D40:	2007	JR	NZ, 2D49H	; 如果不是 $, 则转移。
2D42:	05	DEC	B	; 字符串中剩下的字符个数减 1。
2D43:	1C	INC	E	; 小数点前的说明符个数加 1。
2D44:	FEAF	CP	AFH	; 2D45H: XOR A ★★★ PRINT USING 语句的 $ 处理 ★★★★★★★★★★★★★★★★★★★★★★
2D46:	C610	ADD	A, 10H	; 添加 $ 号的标志。置位 EDIT 标志中的位 4。
2D48:	23	INC	HL	; 指向输入代码串中的下一个字符。
2D49:	1C	INC	E	; 小数点前说明符的个数加 1。
2D4A:	82	ADD	A, D	; 合并 EDIT 标志。
2D4B:	57	LD	D, A	; D=保存修改过的 EDIT 标志。
2D4C:	1C	INC	E	; E=小数点前的#号的个数。★★★ PRINT USING ★★★语句的#号处理,并处理后面的 $$ ★★★★★★
2D4D:	0E00	LD	C, 00H	; 为·号或 $$ 号之后的#号的计数预置初值。
2D4F:	05	DEC	B	; 为检验过的字符串中的字符进行计数。
2D50:	2847	JR	Z, 2D99H	; 如果字符串已处理完,则转移。
2D52:	7E	LD	A, (HL)	; 取字符串中的下一个字符。
2D53:	23	INC	HL	; HL 加 1, 指向下一个字符。
2D54:	FE2E	CP	2EH	; 测试是否为小数点。
2D56:	2818	JR	Z, 2D70H	; 如是小数点,则转移。去寻找小数点后的#。
2D58:	FE23	CP	23H	; 测试是否为#号。
2D5A:	28F0	JR	Z, 2D4CH	; 如是#号,则转移。把它们的计数保留在 E 寄存器内。
2D5C:	FE2C	CP	2CH	; 测试是否是逗号。
2D5E:	201A	JR	NZ, 2D7AH	; 如果不是逗号,则转移。

地址	目的码	源程序		注　释
2D60:	7A	LD	A, D	; 取 EDIT 标志。
2D61:	F640	OR	40H	; 逗号标志位置 1。
2D63:	57	LD	D, A	; 保存修改过的 EDIT 标志。
2D64:	18E6	JR	2D4CH	; 循环，直到字符串处理完或找到小数点，#号或逗号为止。
2D66:	7E	LD	A, (HL)	; 取小数点后的说明符。★★★ PRINT USING 语句的·号处理 ★★★★★★★★★★★★★★★★★★★★★
2D67:	FE23	CP	23H	; 测试是否是#号。
2D69:	3E2E	LD	A, 2EH	; A=小数点的 ASCII 值。
2D6B:	2090	JR	NZ, 2CFDH	; 如不是#号，则转移。
2D6D:	0E01	LD	C, 01H	; C=小数点后的#号的计数。
2D6F:	23	INC	HL	; HL 加 1，指向输入串中的下一个符号。
2D70:	0C	INC	C	; C=小数点后的#号的计数。
2D71:	05	DEC	B	; 检验过的字符串中的字符计数减 1。
2D72:	2825	JR	Z, 2D99H	; 如果字符串已处理完，则转移。
2D74:	7E	LD	A, (HL)	; 从字符串中取下一个符号。
2D75:	23	INC	HL	; 指向字符串中的下一个地址。
2D76:	FE23	CP	23H	; 测试是否是#号。
2D78:	28F6	JR	Z, 2D70H	; 如是#号，则计数和循环，直到字符串处理完。
2D7A:	D5	PUSH	DE	; 保存计数。
2D7B:	11972D	LD	DE, 2D97H	; 测试指数格式[[[[(或↑↑↑↑)以后的转移地址(2D97H)。
2D7E:	D5	PUSH	DE	; DE=字符串中下一个符号的地址。
2D7F:	54	LD	D, H	; 把当前的字符串地址保存在 DE 中。
2D80:	5D	LD	E, L	;
2D81:	FE5B	CP	5BH	; 测试是否为指数符号。
2D83:	C0	RET	NZ	; 如果不是[(或↑)，则返回。
2D84:	BE	CP	(HL)	; 测试是否是[[符号。
2D85:	C0	RET	NZ	; 如果不是[[，则转向 2D97H 去执行。
2D86:	23	INC	HL	; HL 加 1，指向输入串中的下一个元素。
2D87:	BE	CP	(HL)	; 对第三个[作测试。
2D88:	C0	RET	NZ	; 如果不是[[[，则转向 2D97H 去执行。
2D89:	23	INC	HL	; HL 加 1，指向输入串中的下一个元素。
2D8A:	BE	CP	(HL)	; 对第四个[作测试。
2D8B:	C0	RET	NZ	; 如果不是[[[[，则转向 2D97H 去执行。
2D8C:	23	INC	HL	; 是#·##[[[[这样的格式。
2D8D:	78	LD	A, B	; 取字符串说明中剩下的字符个数。
2D8E:	D604	SUB	04H	; 至少剩下 4 个字符吗?
2D90:	D8	RET	C	; 不是，转向 2D97H 去执行。
2D91:	D1	POP	DE	; 是，从堆栈中清除 2D97H。
2D92:	D1	POP	DE	; 把计数和标志再放回到 DE 中。

地址	目的码	源程序		注　释
2D93:	47	LD	B, A	; B=剩余说明符的个数。
2D94:	14	INC	D	;
2D95:	23	INC	HL	; 2D97H: EX DE, HL 保存输入串中的当前位置。
2D96:	CAEBD1	JP	Z, D1EBH	; 2D98H: POP DE 恢复计数和标志。
2D99:	7A	LD	A, D	; 把+,一标志字放进 A 中。★★★★★★★★★★★

说明　完成说明符串的分析。

2D9A:	2B	DEC	HL	; 后退一个说明符。
2D9B:	1C	INC	E	; 对处理过的说明符进行计数。
2D9C:	E608	AND	08H	; 测试是否前面遇到过+号。
2D9E:	2015	JR	NZ, 2DB5H	; 是,跳过对+,一的测试。
2DA0:	1D	DEC	E	; 不是,则进行测试,
2DA1:	78	LD	A, B	; 是否还有剩余的说明符。
2DA2:	B7	OR	A	; 设置状态标志。
2DA3:	2810	JR	Z, 2DB5H	; 如果没有说明符剩下,则转移。
2DA5:	7E	LD	A, (HL)	; 取下一个说明符。
2DA6:	D62D	SUB	2DH	; 测试是否为一号。
2DA8:	2806	JR	Z, 2DB0H	; 如是一号,则把一的标志置1。
2DAA:	FEFE	CP	FEH	; 不是一号,则测试是否为+号。
2DAC:	2007	JR	NZ, 2DB5H	; 如果不是+,则转移。
2DAE:	3E08	LD	A, 08H	; 将位 3 置位(遇到了+号)。
2DB0:	C604	ADD	A, 04H	; 将位 2 置位(遇到了一号)。
2DB2:	82	ADD	A, D	; 把+和一的标志合并。
2DB3:	57	LD	D, A	; 把标志重新存到 D 寄存器。
2DB4:	05	DEC	B	; 对刚处理过的说明符进行计数。
2DB5:	E1	POP	HL	; HL=当前代码串的地址。
2DB6:	F1	POP	AF	; 恢复刚测试过的那个字符及其状态。
2DB7:	2850	JR	Z, 2E09H	; 如果字符串没有结束,则转移。
2DB9:	C5	PUSH	BC	; 保存小数点后的#的个数 (C)。
2DBA:	D5	PUSH	DE	; 保存小数点前的#的个数 (E)。
2DBB:	CD3723	CALL	2337H	; 计算表达式(得到待打印值)。
2DBE:	D1	POP	DE	; 恢复小数点前#的个数 (E)。
2DBF:	C1	POP	BC	; 恢复小数点后#的个数 (C)。
2DC0:	C5	PUSH	BC	; 保存小数点后#的个数。
2DC1:	E5	PUSH	HL	; 保存当前代码串的地址。
2DC2:	43	LD	B, E	; B=小数点前#的个数。
2DC3:	78	LD	A, B	;
2DC4:	81	ADD	A, C	; 小数点前后的#的个数相加。
2DC5:	FE19	CP	19H	; #的总数与 25 作比较。
2DC7:	D24A1E	JP	NC, 1E4AH	; 若多于 25 个#号,则为 FC 错误。
2DCA:	7A	LD	A, D	; D=$$, +, —, 逗号的标志。

地址	目的码	源程序		注　释
2DCB:	F680	OR	80H	; 设置 PRINT USING 调用的标志。
2DCD:	CDBE0F	CALL	0FBEH	; 把当前值变换成 ASCII 形式，它取决于字符串说明。
2DD0:	CDA728	CALL	28A7H	; 打印当前值。
2DD3:	E1	POP	HL	; 把 HL 恢复成代号化的输入代码串。
2DD4:	2B	DEC	HL	;
2DD5:	D7	RST	10H	; 检验代码串的下一个元素。
2DD6:	37	SCF		; 置位进位标志，在代码串结束的情况下，用于 2E04H 处的子程序(以检验代码串的结束)。
2DD7:	280D	JR	Z, 2DE6H	; 如是代码串的结束，则转移。
2DD9:	32DE40	LD	(40DEH), A	; 保存下一个元素。
2DDC:	FE3B	CP	3BH	; 测试是否是分号(;)。
2DDE:	2805	JR	Z, 2DE5H	; 如是;号，则转移——去取表中新的一项。
2DE0:	FE2C	CP	2CH	; 测试是否是逗号。
2DE2:	C29719	JP	NZ, 1997H	; 如不是逗号，则为 SN 错误。
2DE5:	D7	RST	10H	; 取代码串中跟在;号后面的元素。
2DE6:	C1	POP	BC	; B=要打印的字符数。
2DE7:	EB	EX	DE, HL	; DE=当前代码串的地址。
2DE8:	E1	POP	HL	; HL=字符串地址。
2DE9:	E5	PUSH	HL	; 保存在堆栈上。
2DEA:	F5	PUSH	AF	; 保存跟在;号后面的元素。
2DEB:	D5	PUSH	DE	; 保存当前代码串的地址。
2DEC:	7E	LD	A, (HL)	; A=字符串的长度。
2DED:	90	SUB	A, B	; 与要打印的字符数作比较。
2DEE:	23	INC	HL	; 指向字符串地址的 LSB。
2DEF:	4E	LD	C, (HL)	; C=字符串地址的 LSB。
2DF0:	23	INC	HL	; 指向字符串地址的 MSB。
2DF1:	66	LD	H, (HL)	; H=字符串地址的 MSB。
2DF2:	69	LD	L, C	; HL=字符串地址。
2DF3:	1600	LD	D, 00H	; DE=字符串长度。
2DF5:	5F	LD	E, A	; D=0, E=长度。
2DF6:	19	ADD	HL, DE	; HL=字符串的结束地址。
2DF7:	78	LD	A, B	; 现在，测试从字符串取来的待用字符的个数。
2DF8:	B7	OR	A	;
2DF9:	C2032D	JP	NZ, 2D03H	; 如果不为 0，则去检验说明打印格式的字符串。
2DFC:	1806	JR	2E04H	; 如果为 0，去取回代码串地址。
2DFE:	CD492E	CALL	2E49H	; 如果 D 不等于 0，则打印一个+。 ★★★★★★★★
2E01:	CD2A03	CALL	032AH	; 打印 A 寄存器的内容。
2E04:	E1	POP	HL	; HL=当前代码串地址。
2E05:	F1	POP	AF	; A=刚检验过的一个元素。如果代码串结束，则进位标志为 1，否则为 0。
2E06:	C2CB2C	JP	NZ, 2CCBH	; 如果代码串不结束，则转移。

地址	目的码	源程序		注释
2E09:	DCFE20	CALL	C, 20FEH	; 如果代码串结束,则跳一行。
2E0C:	E3	EX	(SP), HL	; 代码串地址放到堆栈,字符串地址放到 HL。
2E0D:	CDDD29	CALL	29DDH	; 把字符串地址放到 DE 中。
2E10;	E1	POP	HL	; HL=代码串地址。
2E11:	C36921	JP	2169H	; 返回到执行驱动程序。
2E14:	0E01	LD	C, 01H	; 根据 (PRINT USING) 后面的字符串要打印的字符个数。★★★ PRINT USING 语句的! 号处理 ★★★★
2E16:	3EF1	LD	A, F1H	; 2E17H: POP AF 清除堆栈。
2E18:	05	DEC	B	; 剩在字符串内的字符个数减 1。
2E19:	CD492E	CALL	2E49H	; 如果 D 寄存器不为 0, 则打印+号。
2E1C:	E1	POP	HL	; HL=输入代码串内的下一个代号的地址。
2E1D:	F1	POP	AF	; 弹出暂存在堆栈的格式字串的起始地址。
2E1E:	28E9	JR	Z, 2E09H	; 如果是!号结束,则转移。
2E20:	C5	PUSH	BC	; 保存'!'串的长度和要打印的字节数。
2E21:	CD3723	CALL	2337H	; 计算下一个表达式。得到要打印的字符串的地址。
2E24:	CDF40A	CALL	0AF4H	; 确认它是一个字符串,否则有错误。
2E27:	C1	POP	BC	; 恢复要打印的字符个数。
2E28:	C5	PUSH	BC	; 把计数保存起来。
2E29:	E5	PUSH	HL	; 保存代码串地址。
2E2A:	2A2141	LD	HL, (4121H)	; 取要打印的字符串地址。
2E2D:	41	LD	B, C	; B=要打印的字符数。
2E2E:	0E00	LD	C, 00H	; C=0。
2E30:	C5	PUSH	BC	; 把计数保存在堆栈。
2E31:	CD682A	CALL	2A68H	; 用 LEFT$ 处理来建立另外一个要打印的字符子串。
2E34:	CDAA28	CALL	28AAH	; 得到子串的地址,并打印它。
2E37:	2A2141	LD	HL, (4121H)	; HL=主串的地址。
2E3A:	F1	POP	AF	; A=主串中的已打印字符的个数。
2E3B:	96	SUB	A, (HL)	; A=没有打印的字符个数=空格数。
2E3C:	47	LD	B, A	; 保存在 B 中。
2E3D:	3E20	LD	A, 20H	; A=ASCII 表示的空格值。
2E3F:	04	INC	B	; 测试要打印的空格数。
2E40:	05	DEC	B	;
2E41:	CAD32D	JP	Z, 2DD3H	; 检验语句的剩余部分是否全部打印空格。
2E44:	CD2A03	CALL	032AH	; 打印空格。
2E47:	18F7	JR	2E40H	; 循环,直到打印出全部空格。
2E49:	F5	PUSH	AF	; 保存状态标志和 A 寄存器内容。 ★★★★★★★★
2E4A:	7A	LD	A, D	; 取 D 寄存器内容。
2E4B:	B7	OR	A	; 测试它是否为 0。
2E4C:	3E2B	LD	A, 2BH	; A='+' 号的 ASCII 码。
2E4E:	C42A03	CALL	NZ, 032AH	; 如果调用时 D 不为 0, 则打印+号。
2E51:	F1	POP	AF	; 恢复调用它的程序的 A 寄存器和标志。

地址	目的码	源程序		注释
2E52:	C9	RET		; 返回到调用它的程序。
2E53:	329A40	LD	(409AH), A	; 清除关于 RESUME 调用的标志。★★★★★★★★★
2E56:	2AEA40	LD	HL, (40EAH)	; 取错误发生处的行号。
2E59:	B4	OR	H	; 如是 FFFFH, 则不开始执行。
2E5A:	A5	AND	L	; 测试行号是否为 FFFFH。
2E5B:	3C	INC	A	;
2E5C:	EB	EX	DE, HL	; DE=有错的行号。
2E5D:	C8	RET	Z	; 如果行号是 FFFFH, 则返回到输入状态。
2E5E:	1804	JR	2E64H	; 否则去打印行号,并进入 EDIT 子程序。
2E60:	CD4F1E	CALL	1E4FH	; 得到第一行号。★★★ EDIT 子程序 ★★★★★★
2E63:	C0	RET	NZ	; 如果第一个行后面有东西,则为语法错误。
2E64:	E1	POP	HL	; 得到代码串地址。
2E65:	EB	EX	DE, HL	; 把它传送到 DE。行号送到 HL。
2E66:	22EC40	LD	(40ECH), HL	; 把编辑的行号传送到通讯区。
2E69:	EB	EX	DE, HL	; 重新把行号放到 DE, 于是就能搜索它。
2E6A:	CD2C1B	CALL	1B2CH	; 在程序表中搜索当前行的地址。
2E6D:	D2D91E	JP	NC, 1ED9H	; 如 C 标志不置位,则为 UL 错误。
2E70:	60	LD	H, B	; 把当前行的地址从 BC 移到 HL。
2E71:	69	LD	L, C	;
2E72:	23	INC	HL	; 跳过指向下一行的指针,
2E73:	23	INC	HL	;
2E74:	4E	LD	C, (HL)	; 并把当前的行号(二进制形式)装入 BC。
2E75:	23	INC	HL	;
2E76:	46	LD	B, (HL)	;
2E77:	23	INC	HL	; HL 加 1, 指向编辑行中的第一个位置。
2E78:	C5	PUSH	BC	; 保存行号。
2E79:	CD7E2B	CALL	2B7EH	; 把当前的行号移到打印/工作区。
2E7C:	E1	POP	HL	; 把当前行号放进 HL,
2E7D:	E5	PUSH	HL	; 并把它存入堆栈。
2E7E:	CDAF0F	CALL	0FAFH	; 把行号变换成 ASCII 码,
2E81:	3E20	LD	A, 20H	; 然后显示行号,并在行号后面放一个空格。
2E83:	CD2A03	CALL	032AH	; 显示空格。
2E86:	2AA740	LD	HL, (40A7H)	; HL=扩展后的当前行地址。
2E89:	3E0E	LD	A, 0EH	; 显示光标代码 (0EH 是光标控制代码)。
2E8B:	CD2A03	CALL	032AH	; 送到显示器。
2E8E:	E5	PUSH	HL	; 保存扩展后的行地址。
2E8F:	0EFF	LD	C, FFH	; C=要检查的字符个数。 扩展后的缓冲器内字符的个数。
2E91:	0C	INC	C	; 测试过一个字符,计数就加 1。
2E92:	7E	LD	A, (HL)	; 从扩展后的缓冲器内取一个字符。
2E93:	B7	OR	A	; 设置状态,于是能对行的结束作测试。

地址	目的码	源程序		注　释
2E94:	23	INC	HL	; 指向被扩展缓冲器中的下一个字符。
2E95:	20FA	JR	NZ, 2E91H	; 如果不是行的结束,则转移。
2E97:	E1	POP	HL	; HL=被扩展缓冲器的首址。C=缓冲器内的字符数。
2E98:	47	LD	B, A	; 使 B=0。它将放有插入的字符数。
2E99:	1600	LD	D, 00H	; 清除 D 寄存器。
2E9B:	CD8403	CALL	0384H	; 用户键入一个字符 (DOS 出口 41C4H)——调整送入的值。
2E9E:	D630	SUB	30H	; 测试字符是字母还是字母数字。
2EA0:	380E	JR	C, 2EB0H	; 若两者都不是,则去测试是否是 EDIT 命令。
2EA2:	FE0A	CP	0AH	; 测试是否是字母数字。
2EA4:	300A	JR	NC, 2EB0H	; 不是数字,则去测试是否是 EDIT 命令。
2EA6:	5F	LD	E, A	; 保存数字的二进制值。
2EA7:	7A	LD	A, D	; 变换成十进制值。构成新的数值。
2EA8:	07	RLCA		; 乘 2。
2EA9:	07	RLCA		; 乘 4。
2EAA:	82	ADD	A, D	; 再加一次,这样到目前为止乘了 5。
2EAB:	07	RLCA		; 给出乘 10 的结果。
2EAC:	83	ADD	A, E	; 加新的数字。
2EAD:	57	LD	D, A	; 作为当前值保存在 D 中。
2EAE:	18EB	JR	2E9BH	; 循环,直到结束命令为止。
2EB0:	E5	PUSH	HL	; 保存被扩展缓冲器的当前地址。
				★★★ 寻找 EDIT 子命令 ★★★★★★★★★★
2EB1:	21992E	LD	HL, 2E99H	; 把 2E99H 作为继续地址保存在堆栈内。
2EB4:	E3	EX	(SP), HL	; HL=被扩展缓冲器地址(当前位置)。
2EB5:	15	DEC	D	; 测试子命令前面是否带有数值。
2EB6:	14	INC	D	; 设置状态标志。
2EB7:	C2BB2E	JP	NZ, 2EBBH	; 如果数值在于命令的前面,则转移。
2EBA:	14	INC	D	; D=1。
2EBB:	FED8	CP	D8H	; 测试是否用户打入了退格键(←)。
2EBD:	CAD22F	JP	Z, 2FD2H	; 如打入了一个退格键,则转移。
2EC0:	FEDD	CP	DDH	; 测试是否是回车键 (CR)。
2EC2:	CAE02F	JP	Z, 2FE0H	; 如用户打入了回车,则转移。
2EC5:	FEF0	CP	F0H	; 测试是否是空格。
2EC7:	2841	JR	Z, 2F0AH	; 如打入了空格,则转移。
2EC9:	FE31	CP	31H	; 测试是否为小写字母。
2ECB:	3802	JR	C, 2ECFH	; 如果不是小写字母,则转移。
2ECD:	D620	SUB	20H	; 把小写字母变换成大写字母。
2ECF:	FE21	CP	21H	; 测试是否为 Q。
2ED1:	CAF62F	JP	Z, 2FF6H	; 若是,则转 QUIT 命令。
2ED4:	FE1C	CP	1CH	; 测试是否为 L。
2ED6:	CA402F	JP	Z, 2F40H	; 若是,则转 LIST 命令。

地址	目的码	源程序		注　释
2ED9:	FE23	CP	23H	; 测试是否为 S。
2EDB:	283F	JR	Z, 2F1CH	; 若是,则转 SEARCH 命令。
2EDD:	FE19	CP	19H	; 测试是否为 I。
2EDF:	CA7D2F	JR	Z, 2F7DH	; 若是,则转 INSERT 命令。
2EE2:	FE14	CP	14H	; 测试是否为 D。
2EE4:	CA4A2F	JR	Z, 2F4AH	; 若是,则转 DELETE 命令。
2EE7:	FE13	CP	13H	; 测试是否为 C。
2EE9:	CA652F	JP	Z, 2F65H	; 若是,则转 CHANGE 命令。
2EEC:	FE15	CP	15H	; 测试是否为 E。
2EEE:	CAE32F	JP	Z, 2FE3H	; 若是,则转 END 命令。
2EF1:	FE28	CP	28H	; 测试是否为 X。
2EF3:	CA782F	JP	Z, 2F78H	; 若是,则转 X 命令。
2EF6:	FE1B	CP	1BH	; 测试是否为 K。
2EF8:	281C	JR	Z, 2F16H	; 若是,则转 KILL 命令。
2EFA:	FE18	CP	18H	; 测试是否为 H。
2EFC:	CA752F	JP	Z, 2F75H	; 若是,则转 HACK 命令。
2EFF:	FE11	CP	11H	; 测试是否为 A。
2F01:	C0	RET	NZ	; 如不是 A, 则退出 EDIT。
2F02:	C1	POP	BC	; 清除堆栈。 ★★★ 取消和恢复 ★★★★★★★★★
2F03:	D1	POP	DE	; 装入二进制格式的当前行号。
2F04:	CDFE20	CALL	20FEH	; 荧光屏显示跳到下一行。
2F07:	C3652E	JP	2E65H	; 重新进入 EDIT 子程序。
2F0A:	7E	LD	A, (HL)	; 从工作区取当前字节。 ★★★★★★★★★★★★★
2F0B:	B7	OR	A	; 设置状态标志,于是我们可对行结束作测试。
2F0C:	C8	RET	Z	; 如是行结束,则返回。
2F0D:	04	INC	B	; (进入)工作缓冲区的索引加 1。
2F0E:	CD2A03	CALL	032AH	; 显示当前的字符。

说明 显示当前行(扩展型式)中的字符,字符个数在 D 寄存器中,当显示遇到行的结束符要终止。每显示一个字符,(进入)工作区的索引 (B 寄存器)就加 1。

2F11:	23	INC	HL	; 指向工作缓冲区的下一个字符。
2F12:	15	DEC	D	; 要显示的字符数减 1。
2F13:	20F5	JR	NZ, 2F0AH	; 如果还有需要显示的字符没有显示,则转移。
2F15:	C9	RET		; 返回。HL=行的结束, B=索引。
2F16:	E5	PUSH	HL	; 保存工作缓冲区中的当前位置。 ★★★ KILL 子命令 ★★★★★★★★★★★★★★★★★★★★★★★★★
2F17:	215F2F	LD	HL, 2F5FH	; 把继续地址 2F5FH (显示最后的!)放入堆栈。
2F1A:	E3	EX	(SP), HL	; 把缓冲区地址重新放入 HL。
2F1B:	37	SCF		; 进位标志置位表示是 KILL, 反之为 SEARCH。
2F1C:	F5	PUSH	AF	; 保存 KILL/SEARCH 标志。
2F1D:	CD8403	CALL	0384H	; 扫描键盘,得到要搜索的字符。

地址	目的码	源程序		注释
2F20:	5F	LD	E, A	; 保存搜索字符。
2F21:	F1	POP	AF	; 装入 KILL/SEARCH 标志。
2F22:	F5	PUSH	AF	; 再把 KILL/SEARCH 标志推入堆栈。
2F23:	DC5F2F	CALL	C, 2F5FH	; 如需要显示前导的!,则转移。(KILL 子命令)
2F26:	7E	LD	A, (HL)	; 取当前的字符。
2F27:	B7	OR	A	; 设置状态标志。
2F28:	CA3E2F	JP	Z, 2F3EH	; 如找到行的结束,则转移。
2F2B:	CD2A03	CALL	032AH	; 显示待删除/检验的字符。
2F2E:	F1	POP	AF	: 装入 KILL/SEARCH 标志。
2F2F:	F5	PUSH	AF	; 保存标志字。
2F30:	DCA12F	CALL	C, 2FA1H	; 如是 KILL, 则把工作缓冲区的剩余部分往下移一个字符。
2F33:	3802	JR	C, 2F37H	; 如是 KILL 子命令,则转移。
2F35:	23	INC	HL	; 对于 SEARCH——指向下一个字符。
2F36:	04	INC	B	; 对于 SEARCH——计数刚显示的字符。
2F37:	7E	LD	A, (HL)	; 为 KILL/SEARCH 取下一个字符。
2F38:	BB	CP	E	; 测试是否同搜索的字符一致。
2F39:	20EB	JR	NZ, 2F26H	; 不一致,则循环。
2F3B:	15	DEC	D	; 所有需要出现的搜索字符是否都已找到。
3F3C:	20E8	JR	NZ, 2F26H	; 没有,则循环。
2F3E:	F1	POP	AF	; 是,清除 KILL/SEARCH 标志。
2F3F:	C9	RET		; 退出编辑子命令。
2F40:	CD752B	CALL	2B75H	; 显示当前的行(由 EDIT 扩展的)。★★★ LIST 子程序 ★★★★★★★★★★★★★★★★★★★★★★
2F43:	CDFE20	CALL	20FEH	; 跳到下一行,输出一个回车符。
2F46:	C1	POP	BC	; 恢复当前的行号。
2F47:	C37C2E	JP	2E7CH	; 显示当前的行号和等待下一个 EDIT 命令。
2F4A:	7E	LD	A, (HL)	; 从工作缓冲区取得当前的字符。★★★ DELETE 子程序 ★★★★★★★★★★★★★★★★★★★★★★
2F4B:	B7	OR	A	; 设定状态标志,于是能对行的结束作测试。
2F4C:	C8	RET	Z	; 如是行的结束,则返回。
2F4D:	3E21	LD	A, 21H	; A='!' 的 ASCII 代码。
2F4F:	CD2A03	CALL	032AH	; 显示'!'作为删除区开始的标记。
2F52:	7E	LD	A, (HL)	; 取当前的字符。
2F53:	B7	OR	A	; 测试是否是行的结束。
2F54:	2809	JR	Z, 2F5FH	; 如果在 D 内的数减完以前遇到了行的结束, 则转移。
2F56:	CD2A03	CALL	032AH	; 显示待删除的字符。
2F59:	CDA12F	CALL	2FA1H	; 从工作缓冲区删去字符。
2F5C:	15	DEC	D	; 删去了一个字符,计数减 1。
2F5D:	20F3	JR	NZ, 2F52H	; 如果没有删去 D 个字符,则循环。
2F5F:	3E21	LD	A, 21H	; 显示'!',作为删除区结束的标记。

地址	目的码	源程序		注　释
2F61:	CD2A03	CALL	032AH	; 显示'!'。
2F64:	C9	RET		; 从删除子命令退出。
2F65:	7E	LD	A, (HL)	; 取得待改变的字符。★★★ CHANGE 子命令 ★★★
2F66:	B7	OR	A	; 测试是否是行的结束。
2F67:	C8	RET	Z	; 如果是行的结束,则从 CHANGE 子命令退出。
2F68:	CD8403	CALL	0384H	; 得到从键盘来的一个字符以代替当前的字符。
2F6B:	77	LD	(HL), A	; 代替工作缓冲区中的当前字符。
2F6C:	CD2A03	CALL	032AH	; 显示新的字符。
2F6F:	23	INC	HL	; 指向工作缓冲区的下一个位置。
2F70:	04	INC	B	; 改变了一个字符,计数加 1。
2F71:	15	DEC	D	; 已改变字符的计数要减 1。
2F72:	20F1	JR	NZ, 2F65H	; 如果还有字符要改变,则循环。
2F74:	C9	RET		; 从子命令退出。
2F75:	3600	LD	(HL), 00H	; 结束当前的行。 ★★★ HACK/INSERT 和 X 子命令 ★★★★★★★★★★★★★★★★★★★★★★★
2F77:	48	LD	C, B	; 把行的数目放在 C 中。
2F78:	16FF	LD	D, FFH	; 设定显示的字节数为 255。
2F7A:	CD0A2F	CALL	2F0AH	; 显示 255 个字节,或者直到行的结束。显示当前行。
2F7D:	CD8403	CALL	0384H	; 调用键盘扫描程序。在按下一个键时就返回。 ★★★ INSERT 子命令 ★★★★★★★★★★★★★
2F80:	B7	OR	A	; 测试是否是非零字符。
2F81:	CA7D2F	JP	Z, 2F7DH	; 这个测试并非必需, 因 384H 单元的程序作同样的测试。
2F84:	FE08	CP	08H	; 测试是否是退格键(←)。
2F86:	280A	JR	Z, 2F92H	; 如是退格,则转移——把光标左移一个字符。
2F88:	FE0D	CP	0DH	; 测试是否是回车键。
2F8A:	CAE02F	JP	Z, 2FE0H	; 如打了回车键,则显示一行,并把该行加到当前的程序中。
2F8D:	FE1B	CP	1BH	; 测试是否是 ESC 键。
2F8F:	C8	RET	Z	; 如是 ESC, 则从 EDIT 方式退出。
2F90:	201E	JR	NZ, 2FB0H	; 无条件转移,在当前行中加一个新字符。
2F92:	3E08	LD	A, 08H	; A=光标左移的代码。★★★ BACKSPACK CURSOR 子命令 ★★★★★★★★★★★★★★★★★★★★★★
2F94:	05	DEC	B	; 在左移以前,
2F95:	04	INC	B	; 测试当前行中的字符数。
2F96:	281F	JR	Z, 2FB7H	; 如果是 0, 则处于行的起点。执行 INSERT 命令。
2F98:	CD2A03	CALL	032AH	; 把光标左移命令送到视频显示器。
2F9B:	2B	DEC	HL	; 工作缓冲区的指针后退一个位置。
2F9C:	05	DEC	B	; 当前行中的字符个数减 1。
2F9D:	117D2F	LD	DE, 2F7DH	; 把继续地址 2F7DH (INSERT) 推入堆栈。

说明　从工作缓冲区删去一个字符。把它后面的所有字符下移一个字节。

地址	目的码	源程序		注　释
2FA0:	D5	PUSH	DE	;
2FA1:	E5	PUSH	HL	; 保存工作缓冲区中的当前地址。
2FA2:	0D	DEC	C	; 缓冲区中的字符数减 1。
2FA3:	7E	LD	A, (HL)	; 取将被覆盖的下一个字符。
2FA4:	B7	OR	A	; 为测试行的结束而设置状态标志。
2FA5:	37	SCF		; 进位标志置 1，表示是个删去的字符。
2FA6:	CA9008	JP	Z, 0890H	; 如果所有的字符都向下移动了一个字符，则退出。
2FA9:	23	INC	HL	; HL 加 1。
2FAA:	7E	LD	A, (HL)	; 把字符 *n* 放进 A 寄存器。
2FAB:	2B	DEC	HL	; 指针减1，指向字符 *n*−1。
2FAC:	77	LD	(HL), A	; 把字符 *n* 存放到字符 *n*−1 的地方。
2FAD:	23	INC	HL	; 重新定位缓冲器地址于字符 *n*。
2FAE:	18F3	JR	2FA3H	; 循环，直到工作缓冲区内所有的字符均下移一个字节。
2FB0:	F5	PUSH	AF	; 保存待加进去的字符。在当前行中添加一个字符。 ★
2FB1:	79	LD	A, C	; 取当前行中字符的个数。
2FB2:	FEFF	CP	FFH	; 测试是否已达最大的行容量。
2FB4:	3803	JR	C, 2FB9H	; 如果该行长度尚未达到 255 字节，则转移，
2FB6:	F1	POP	AF	; 否则，要重打键入的最后一个字符——因它将不被认可。
2FB7:	18C4	JR	2F7DH	; 返回到 INSERT（插入子命令）。循环，直到键入退格键（←），回车键（CR）或 ESC（↑）键为止。
2FB9:	90	SUB	A, B	; 给出缓冲器中当前字节的位置。 ★★★★★★★★★
2FBA:	0C	INC	C	; 当前行中的字符个数加 1。
2FBB:	04	INC	B	; 添加的字符个数加 1。
2FBC:	C5	PUSH	BC	; 保存添加的字符个数和当前行的字符数。
2FBD:	EB	EX	DE, HL	; DE＝当前行的开始地址。
2FBE:	6F	LD	L, A	; 把当前的字符索引传送到 HL 中。
2FBF:	2600	LD	H, 00H	; 使高八位为 0，于是能用 16 位算术运算，
2FC1:	19	ADD	HL, DE	; 把索引加开始地址(缓冲器)以得到当前字符的地址。
2FC2:	44	LD	B, H	; 把当前字符的地址放到 BC 中。
2FC3:	4D	LD	C, L	;
2FC4:	23	INC	HL	; HL＝下一个可用字符位置的地址。
2FC5:	CD5819	CALL	1958H	; 为插入字符，把带有空位的新行传送到工作缓冲区。
2FC8:	C1	POP	BC	; 把添加的字符个数/行内的字符个数重新放到 BC 中。
2FC9:	F1	POP	AF	; 把要添加的字符重新放到当前行内。
2FCA:	77	LD	(HL), A	; 把新的字符插入行内。
2FCB:	CD2A03	CALL	032AH	; 显示已插入的字符。
2FCE:	23	INC	HL	; 指向工作缓冲区内的下一个位置。
2FCF:	C37D2F	JP	2F7DH	; 等待下一个字符或者回车键，ESC 键或退格键。
2FD2:	78	LD	A, B	; B＝要左移的字符数。 ★★★★★★★★★★★★★★
2FD3:	B7	OR	A	; 测试是否是 0。
2FD4:	C8	RET	Z	; 如果已完成了左移，则返回到 2E99H 单元。

地址	目的码	源程序		注释
2FD5:	05	DEC	B	; 为左移了一个字符而计数。
2FD6:	2B	DEC	HL	; EDIT 缓冲区的指针减 1。
2FD7:	3E08	LD	A, 08H	; 光标左移命令。
2FD9:	CD2A03	CALL	032AH	; 荧光屏显示光标左移。
2FDC:	15	DEC	D	; 左移过的字符个数。
2FDD:	20F3	JR	NZ, 2FD2H	; 循环,直到 D 内记有的字符数左移完为止。
2FDF:	C9	RET		; 返回到 2E99H。
2FE0:	CD752B	CALL	2B75H	; 显示当前行的剩余字符。★★★ 在插入和命令输入方式中, END (结束) 和 CR (回车) ★★★★★★★★
2FE3:	CDFE20	CALL	20FEH	; 在荧光屏上跳到下一行。
2FE6:	C1	POP	BC	; 清除堆栈。
2FE7:	D1	POP	DE	; 取回当前行的二进制行号。
2FE8:	7A	LD	A, D	; 把行号的 LSB 和 MSB 合并。
2FE9:	A3	AND	E	;
2FEA:	3C	INC	A	; 指向下一个行号。
2FEB:	2AA740	LD	HL, (40A7H)	; HL=工作缓冲区的首址。
2FEE:	2B	DEC	HL	; 工作缓冲区的首址减 1。
2FEF:	C8	RET	Z	; 如果尚未开始执行 BASIC, 则退出。
2FF0:	37	SCF		; 设置进位标志; 以表示一个 BASIC 程序语句。它将在 1AA4H 处测试。
2FF1:	23	INC	HL	; 指向工作缓冲区的首址。
2FF2:	F5	PUSH	AF	; 保存语句与命令输入的标志。
2FF3:	C3981A	JP	1A98H	; 在程序中加入一个新的行。
2FF6:	C1	POP	BC	; 清除堆栈。★★★ QUIT 子命令 ★★★★★★★★
2FF7:	D1	POP	DE	; DE=当前的行号。
2FF8:	C3191A	JP	1A19H	; 返回到 BASIC 的 'READY' 子程序。
2FFB:	00	NOP		; 0000H=程序的入口地址。
2FFC:	00	NOP		;
2FFD:	00	NOP		;
2FFE:	00	NOP		;
2FFF:	00	NOP		;
3000:	C34232	JP	3242H	;
3003:	C3DA32	JP	32DAH	;
3006:	C35C33	JP	335CH	;
3009:	C36D33	JP	336DH	;
300C:	C38233	JP	3382H	;
300F:	C37F34	JP	347FH	;
3012:	C38734	JP	3487H	;
3015:	2AE640	LD	HL, (40E6H)	;
3018:	C31E1D	JP	1D1EH	;
301B:	C36534	JP	3465H	;

地址	目的码	源程序		注释
301E:	C32A33	JP	332AH	;
3021:	C36E33	JP	336EH	;
3024:	C35F32	JP	325FH	;
3027:	C36433	JP	3364H	;
302A:	C39A34	JP	349AH	;

附　　录

一、LEVEL II 子程序入口地址索引

地址 (Hex.)	说　　明
0000	开机时 IPL 处理的入口。
0013	键盘子程序入口 (B = 入口代码 01H→JR 0046H→JP 03C2H)。
001B	显示器和打印机子程序入口 (B = 入口代码 02H→JR 0046H→JP 03C2H)。
002B★	扫描键盘一次,键入字符送 A。
0033★	荧光屏显示A中的字符。
003B★	打印机打印A中的字符。
0040	(JP 05D9H), 见 05D9 说明。
0046	(JP 03C2H), 见 03C2 说明。
0049★	循环扫描键盘,等待键入一个字符。
0060	软件延时 (BC = 延时计数)。
0066	复位 (RESET) 处理。
0075	没有磁盘时复位处理。
00B2	清除荧光屏,显示 "MEMORY SIZE?" 并等待输入。
00FC	显示 "RADIO SHACK LEVEL II BASIC" 信息和 "READY" (JP 1A19H)。
0105	"MEMORY SIZE" 信息首址。
0111	"RADIO SHACK LEVEL II BASIC" 信息首址。
0132	POINT/SET/RESET 语句入口 (A=0 为 POINT, A=−1 SET, A = 1 RESET)。
0135	SET 语句。
0138	RESET 语句。
019D	INKEY$ 函数。
01C9★	清除荧光屏 (CLS 语句)。
01D3★	RANDOM 语句。
01D9	往磁带上写一位(一个时钟脉冲)。
01F8	关磁带机马达。
0212★	开磁带机马达 (A = 设备号)。
022C★	闪烁"*"号一次。
0235★	从磁带读入一个字节。
0241	从磁带读入一位。
0264★	往磁带上写一个字节。
0284★	开磁带机马达,并往磁带上写引导码。
0287	往磁带上写引导码
0293	开磁带机马达,并读磁带寻找引导码。

地址 (Hex.)	说　　明
0296★	读磁带寻找引导码。
02A9	从磁带上读入两个字节的汇编程序执行地址。
02B2	SYSTEM 命令。
02CE	从磁带上读入汇编程序。
031D	执行汇编程序 (40DFH 中为执行地址)。
032A	将 (A) 输出到荧光屏、打印机或磁带机((409CH) = −1 为磁带机，0 为显示器，1 为打印机)。
0361	等待键盘输入子程序。
0384	循环扫描键盘(与 0049 相同)。
039C	打印机打印 A 的内容。
03C2	通用驱动程序入口 (B = 入口代码，DE = DCB 地址)。
03E3	键盘驱动程序。
0458	显示器驱动程序。
04C0	光标复原、显示器为每行 64 字符。
058D	打印机驱动程序。
05D1★	取打印机状态 (Z 标志置位为 READY)。
05D9★	等待由键盘输入一行。
0674	IPL 程序。
0708	单精度加子程序 (0.5 + WRA1→WRA1)。
070B	单精度加(以 HL 为指针的单精度值 + WRA1→WRA1)。
0710	单精度减(以 HL 为指针的单精度值−WRA1→WRA1)。
0713★	单精度减 (BCDE − WRA1→WRA1)
0716★	单精度加 (BCDE + WRA1→WRA1)
0754	以 HL 为指针的单精度值 +(−BCDE)→BCDE
07C3	四字节负整数求补。
07D7	分解一个单精度数。
0809★	自然对数 (LOG 函数)。
0847★	单精度乘 (BCDE＊WRA1→WRA1)。
0897	WRA1 中单精度数除以 10 (WRA1/10→WRA1)。
08A2★	单精度除 (BCDE/WRA1→WRA1)。
093E	WRA1 中单精度数乘以 10 (WRA1＊10→WRA1)。
0955	测试 WRA1 中单精度数的符号。
0964	把整数转换成单精度数。
0977★	取绝对值 (ABS 函数)。
0982★	改变 WRA1 中值的符号。
098A	求自变量的符号 (SGN 函数)。
09A4★	WRA1→堆栈。
09B1★	HL 为指针的单精度数→BCDE→WRA1。
09B4★	BCDE→WRA1。
09BF	WRA1→BCDE。
09C2★	HL 为指针的单精度数→BCDE。

地址（Hex.）	说　　明
09CB	WRA1→(HL)（送入以 HL 为指针的内存单元）。
09D2	变量传送：(HL)→(DE)，(40AFH) = 传送的字节数(变量类型码)。
09D3	变量传送：(DE)→(HL)。
09D6	通用传送子程序：(DE)→(HL)，A = 传送的字节数。
09D7★	通用传送子程序：(HL)→(DE)，B = 传送的字节数。
09F4	WRA2→WRA1。
09FC	WRA1→WRA2。
0A0C★	比较单精度数:BCDE:WRA1。
0A39★	比较整数:HL:DE。
0A49	比较双精度数：WRA1:(DE)，((DE) 表示以 DE 为指针的双精度数)。
0A4F	比较双精度数：WRA1:WRA2。
0A78★	(CALL 0A4FH)，同上。
0A7F★	取整数（CINT 函数）。
0A9A	HL 中的整数→WRA1。
0AB1★	CSNG 函数(转换成单精度数)。
0ACC	WRA1 中的整数转换成单精度数。
0ADB★	CDBL 函数(转换成双精度数)。
0AF4	测试数据类型是否字符串，若是数值，则为 TM 错误。
0B26★	FIX 函数。
0B37★	INT 函数。
0B59	双精度数转换成整数。
0BAA	16 位二进制数(整数)乘：BC＊DE→DE。
0BC7★	整数减：DE − HL→HL（以及 WRA1 中）。
0BD2★	整数加：DE + HL→HL（以及 WRA1 中）。
0BF2★	整数乘：DE＊HL→HL（以及 WRA1 中）。
0C70★	双精度减:WRA1 − WRA2→WRA1。
0C77★	双精度加：WRA1 + WRA2→WRA1。
0C92	交换工作寄存器区（WRA1↔WRA2）。
0D57	WRA1 中双精度数求反。
0DA1★	双精度乘（WRA1＊WRA2→WRA1）。
0DDC	WRA1 中双精度数除以 10（WRA1/10→WRA1）。
0DE5★	双精度除（WRA1/WRA2→WRA1）。
0E4D	WRA1 中双精度数乘以 10（WRA1＊10→WRA1）。
0E65★	ASCII 转换成双精度数。
0E6C★	ASCII 转换成二进制值。
0FAF★	HL 中值转换成 ASCII。
0FBE★	二进制转换成 ASCII。
103D	单(或双)精度值转换成 ASCII。
109A	PRINT USING 语句。
132F★	整数转换成 ASCII。
1364	10 的乘幂表。

地址（Hex.）	说　　明
13E7★	求平方根（SQR 函数）。
13F2★	求乘方：X^Y。
1439★	EXP 函数(求 e^x)。
149A	求级数和的通用子程序。
14C9★	RND（X）函数。
14F0	RND（0）函数。
1541★	COS 函数。
1547★	SIN 函数。
15A8★	TAN 函数。
15BD★	ATN 函数。
1608	内部函数地址表。
1650	保留字内部代号表。
1822	工作子程序向量地址表。
189A	算符优先值表。
18A1	算术子程序地址表。
18C9	错误代码表。
18F7	除法支援子程序。
191D	''ERROR'' 信息。
1924	''IN'' 信息。
1929	''READY'' 信息。
1930	''BREAK'' 信息。
1955	将字符串变量送入指定区域。
1963	计算 HL 为指针的内存单元到 FFC6H 之间的内存数量。
19A2	显示错误信息子程序（E = 错误代码）。
1A19	BASIC READY 程序
1A39	自动行号输入((40E1H) = AUTO 标志:非 0 为 AUTO, 00 为非 AUTO)。
1A76	不用 AUTO 输入行号。
1AFC	修改新行后边的所有行的行指针。
1B2C★	寻找相同的行号。
1B49	NEW 命令。
1B5D	RUN 命令。
1B64	RUN ×× 命令。
1BB3	显示''?''后等待键盘输入。
1BC0	将输入缓冲区内容代号化。
1C90★	RST 18H
1C96★	RST 08H
1CA1	FOR 语句。
1D5A	执行阶段。
1D78★	RST 10H
1D91	RESTORE 语句。
1D9B	扫描键盘一次,测试是否按了 SHIFT ⓐ 或 BREAK 键,以作不同处理。

地址 (Hex.)	说　　明
1DA9	STOP 语句。
1DAE	END 语句。
1DE4	CONT 命令。
1DF7★	TRON 命令。
1DF8★	TROFF 命令。
1E00	DEFSTR 语句。
1E03	DEFINT 语句。
1E06	DEFSNG 语句。
1E09	DEFDBL 语句。
1E3D	测试 (HL) 是否字母。
1E5A★	ASCII (以 HL 为指针)转换成二进制整数(放 DE 中)。
1E7A	CLEAR 语句。
1EA3	RUN 子程序人口。
1EB1★	GOSUB 语句。
1EC2	GOTO 语句。
1EDE★	RETURN 语句。
1F05	DATA 语句。
1F07	REM 语句和 ELSE 语句。
1F21	LET 语句。
1F6C	ON 子程序人口。
1F70	ON ERROR GOTO 语句。
1F95	ON *n* GOTO (或 GOSUB) 语句。
1FAF	RESUME 语句。
1FF4	ERROR 语句。
2008	AUTO 命令。
2039	IF 语句。
2067	LPRINT 语句。
206F	PRINT 语句。
2178	"? REDO" 信息。
219A	INPUT 语句。
21EF	READ 语句。
22B6	NEXT 语句。
23EC	关系代号子程序。
2490★	整数除 (DE/HL→WRA1)。
24E7	VARPTR 函数。
2532	二进制减法子程序。
258C	两个字符串进行比较。
25B9	比较两个逻辑量。
25C4	NOT 子程序。
25D9★	RST 20H
2608	DIM 语句。

地址（Hex.）	说　　明
260D★	寻找变量的地址。
26E9	寻找下标变量的地址。
27C9★	MEM 函数。
27D4	FRE 函数。
27F5	POS 函数。
27FE	USR 函数。
2836	STR$ 函数。
28A7★	显示（HL）信息。
29C8★	字符串传送：(HL)→(DE)。
2A03	LEN 函数。
2A0F	ASC 函数。
2A1F	CHR$ 函数。
2A2F	STRING$ 函数。
2A61	LEFT$ 函数。
2A91	RIGHT$ 函数。
2A9A	MID$ 函数。
2AC5	VAL 函数。
2AEF	INP 函数。
2AFB	OUT 语句。
2B29	LLIST 命令。
2B2E	LIST 命令。
2B75★	显示以 HL 为指针的信息(不用文字串库区)。
2BC6	DELETE 命令。
2BF5	CSAVE 命令。
2C1F	CLOAD 命令。
2CA5	"BAD" 信息。
2CAA	PEEK 函数。
2CB1	POKE 语句。
2E60	EDIT 命令。
2EB0	寻找 EDIT 子命令。
2FFF	LEVEL II ROM 终址。

注　凡上角标有★号的地址，在第二章内已有详细说明，在使用其他入口地址时应首先查看程序清单，了解入口和出口条件。

二、ASCII 代码表

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

三、英汉词汇对照表

ancillary support routine	辅助支援程序
assembler	汇编程序
code string	代码串
communication region	通讯区
compiler	编译程序
Compressed format	压缩格式
CP (clock pulse)	时钟脉冲
CPU (central processing unit)	中央处理器
CR (carige return)	回车
CSP (current stack pointer)	当前堆栈指针

current line	当前行
DCB (device control block)	设备控制块
default	缺项
descriptor	说明符
devide support routine	除法支援程序
direct statement	直接语句
DOS (disk operating system)	磁盘操作系统
D. P. (double precision)	双精度
ECSP (end of CSP)	CSP 结束
entry	入口,目录登记项
EOS (end of statement)	语句结束
execution driver	执行驱动程序
execution phase	执行阶段
exit	出口
exponent	指数
EVLT (end of VLT)	VLT 结束
FCB (file control block)	文件控制块
field	字段
FP (floating point)	浮点
FSL (free space list)	空闲区
GAP (granul assignment pair)	gr. 分配对
GAT (granul assignment table)	gr. 分配表
granul	gr. (=5 个扇区)
header	引导码(磁带)
HIT (hash index table)	散列检索表
input phase	输入阶段
internal subroutine	内部子程序
interpreter	解释程序
interrupt handler	中断处理(服务)程序
IPL (initial program loader)	初始程序的装入程序
linking loader	连接装入程序
literal string pool table	文字串库表
LSB (least segnificent byte)	最低有效字节
LSP (literal string pool)	文字串库
mode flag (或 type flag)	类型标志
module	模块
mapping	映象
MSB (most segnificent byte)	最高有效字节
name extention	(文件)名扩展
NEWDOS	TRS-80 操作系统版本中的一种
NMSB (next most segnificent byte)	次高有效字节
object time routine	目的码运行程序
operand	操作数,运算量

operator	操作符
overflow entry	扩充(目录)项
PBUF (print buffer)	打印(显示)缓冲区
precedence value	优先级值
primary entry	主(目录)项
PST (program statement table)	程序语句表
RAM (random access memory)	随机存储器
reserved word	保留字
ROM (read only memory)	只读存储器
scale	按比例换算
sector	(磁盘)扇区
S. P. (single precision)	单精度
stack area	堆栈区
string list area	字符串表区
task scheduler	任务调度程序
token	代号
tokenize	代号化
TRSDOS	TRS-80 操作系统版本中的一种
unit	设备
utility	实用程序
variable type	变量类型
vector	向量
verb action routine	工作子程序
VLT (variable list table)	变量表
WRA (work register area)	工作寄存器区