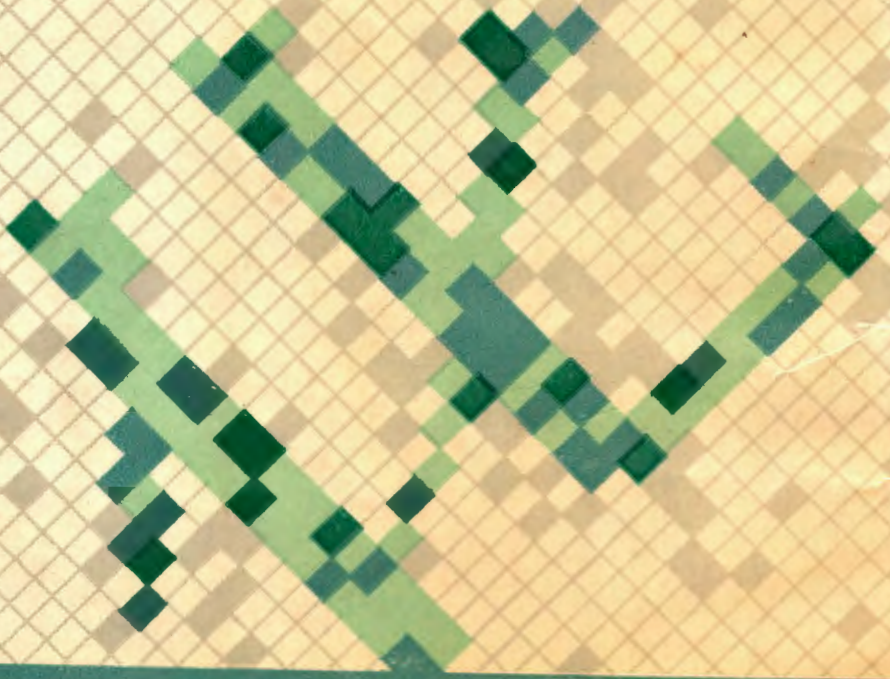
软件的汉化技术

李亦何　编著



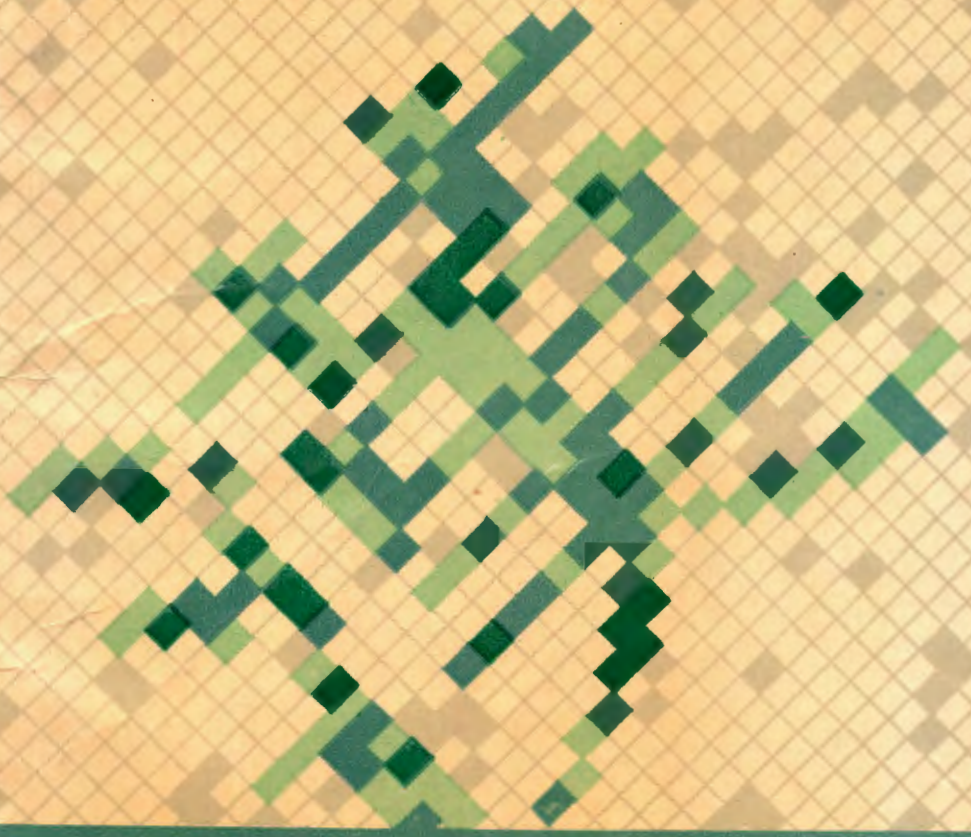
```
.286CPU
SEG_A       SEGMENT
            ASSUME      CS:SEG_A,  DS:SEG_A
            ORG         100H
DISP        PROC    FAR
START:

            JMP         SHORTLOC_1      ;  (010F)
```

ISBN 7-5608-0774-7/TF·69　　定价：3.90元

软件的汉化技术

李亦何，编著



```
CS:AAC0   MOV    CX,0020          ;每个汉字32个字节
CS:AAC3   MUL    CX               ;算出字模首地址
CS:AAC5   PUSH   ES
CS:AAC6   PUSH   CS
CS:AAC7   POP    ES
CS:AAC8   ADD    DL,10            ;AX:DX=
CS:AACB   MOV    CS:[AAFF],DL     ;初始化(
CS:AAD0   MOV    CS:[AAFD],AX
```

软件的汉化技术

李亦何　编著

同济大学出版社

内 容 提 要

本书以IBM-PC/XT、80286/386等系列的微机环境为背景，系统地叙述了软件的汉化技术、汉化方法和汉字系统的移植要点。全书主要分三个部分，即基础部分、汉化部分和移植部分。本书由浅入深，从汉化工作必须掌握的基本知识和技能开始，分别对汉字系统的功能、运行环境、汉化工作的步骤和方法进行了深入的讨论，对汉字系统的功能扩充、造字、汉字打印和在各种CRT显示终端上的移植技术均有新颖而详细的描述。为了使读者更好地理解和掌握汉化技术与系统移植原理，书中还给出了许多程序实例供读者学习和参考。

本书可作为高等院校计算机专业高年级学生的参考教材，也可作为从事计算机科学、计算机工程、计算机应用和汉字系统及汉字软件研制、生产、开发的科技人员的参考用书。

责任编辑　郁　峰
封面设计　陈益平

软件的汉化技术

李亦何　编著
同济大学出版社出版
（上海四平路1239号）
新华书店上海发行所发行
浙江诸暨印刷厂印刷
开本：850×1092　1/32　印张：9.375　字数：263千字
1991年7月第1版　1991年7月第1次印刷
印数：10000　定价：3.90元
ISBN 7-5608-0774-7/TP·69

前　言

电子计算机的发明，对于人类的科学文化生活，具有划时代的意义。当代社会是充满信息的社会，由于信息量的日趋庞大，信息结构也愈来愈复杂，用计算机实现对信息的处理和管理，已势在必行。但是，电子计算机诞生在西方，其硬件设备，操作系统和大量丰富的软件适用于西文的处理，西文拼音字的字符少，结构简单，处理起来比较方便。而在我国，所使用的文字是汉字，计算机所处理的信息大部分是汉字信息。由于汉字的数量很多，字形结构也较复杂，因此，用计算机处理汉字信息就受到一定程度的限制。为使我国科学技术迅速发展，早日实现现代化，推广计算机的应用是十分必要的。因此，研究和开发汉字信息处理技术具有重大的现实意义。

近年来，我国的计算机工作者作了大量的开发工作，克服了许多技术难关，推出了不少适应处理汉字的硬件设备和汉字操作系统。这些基础工作，为我国计算机应用技术提供了良好的应用环境。因此，在我国的各个领域中，计算机的应用范围日益扩大。譬如书刊、情报资料管理系统，文字处理、编辑排版系统，宾馆、医疗卫生管理系统，金融、企业的信息管理和办公自动化等，都离不开计算机管理。除此之外，还有更多的项目迫切地等待着我们去开发和应用，社会需要是技术进步的强大动力，计算机的推广应用不仅是计算机事业发展的目的，也是科学技术水平发展的标志。

在各种计算机应用领域中，均需要有不同的系统软件和应用软件的支持。要在我国使用计算机，只有能够处理汉字信息的应用软件才具有生命力。但是，有许多优秀软件却并不具备处理汉字信息的功能，这就给我们提出了一个问题，就是要把西文软件汉字化。或者说，对于用户，要有一个中西文兼容的应用环境。

虽然，我国自行开发了许多好的应用软件，但是，这些还远远不能满足微机应用的社会需求。对于许多国外著名的软件，若不经过汉化处理，就无法在汉字操作系统环境下使用。许多计算机用户往往被动地等待着汉化软件的推出，然后才去予以使用，这样，不仅耽误了软件应用的时间，而且由于一些软件不是自己开发和汉化的，因此，在运行的过程中出了一点小问题，往往就会一筹莫展，这些问题给用户带来了许多不便。因此，汉化技术是目前计算机领域中一个比较重要的课题。可以说，几乎每个从事计算机软件开发和使用的人都会碰到这个问题。从计算机应用的发展眼光来看，不仅系统开发人员应该掌握这一技术，而且应用者也应该了解和掌握汉化技术，这样将会大大减少软件开发的周期和使用故障，并且扩大了用户选择软件环境的余地。由于各种软件之间的结构差异很大，因此，至今没有系统地研究和讨论其规范，也没有出版过这方面的论著。但曾有不少专家提出，应把软件的汉化作为一项软件工程来研究，作者认为是很有必要的。

本书是作者在长期的软件开发工作中积累的资料和多次为开办系统级培训班编写的讲义稿的基础上整理而成的。其旨意在于为读者、尤其是为想在系统作二次开发的初学者提供一些了解系统的思路和软件汉化的技巧。对于高级程序员来说，本书可作为一本编程手册，以便提高系统开发的能力和较快地掌握软件汉化的技术。

书中的第1～3章介绍了汉化技术需要掌握的基础知识，第4、5章描述了汉化工作的具体步骤和关键技术，第6～8章讨论了汉字系统下各种功能的扩充和移植问题，希望能给读者一些有益的启发和帮助，更欢迎计算机同行们共同探讨软件的汉化技术，推动我国计算机事业迅速发展。

在编写本书的过程中，刘杰先生给于了许多热情的帮助，并得到了许多专家的大力支持和鼓励，尤其是在编辑出版时，同济大学出版社的郁峰先生提出了许多诚恳的建议和具体的指导，在此表示衷心感谢。此外，对出版社的有关编辑、审稿、制图人员

的辛勤工作一并表示深切的谢意。

由于汉字系统和汉化技术涉及面广、其对应的软件种类繁多，再加上作者才学浅陋，时间仓促，书中难免会有不少缺点和错误，敬请读者不吝指教。

作　者

1990 年 7 月

于上海

目　录

第1章 概 论

1.1 引言

目前，电子计算机的应用已经深入到各个领域。我国是使用汉字的国家，计算机技术无论应用在哪个领域，几乎全离不开汉字信息的处理。因此，为使计算机技术在我国能得到更广泛的应用和更迅速的发展，就必须开发大量能够处理汉字的系统软件和应用软件。如何使计算机软件能够处理中文信息呢？从语言文字学的角度来看，需要一种专门的中文信息处理系统。若从世界科技文化的共性来看，则需要通用的信息处理系统。由此说来，我们应该采用“中西文兼容”的方法，在世界范围内交流信息。怎样实现这种“兼容”技术？“兼容”到何种程度？这是我国广大计算机专业人员非常关注的问题。本书就是针对这一问题展开讨论。

多年来，许多计算机专业人员致力于研究计算机的汉字处理方法，根据计算机的硬件设备以及系统软件和应用软件的使用条件和环境进行了大量的二次开发工作，使我国计算机的应用得以广泛地普及和推广。譬如：CC-DOS 汉字操作系统、桌面字处理、轻印刷排版系统和适应于各个层次的汉字输入编码等等，都说明了计算机在处理汉字信息方面取得了许多成果。然而，系统软件仅仅给广大用户建立了一个汉字系统的环境。在实际应用中，绝大多数用户需要各种类型的应用软件包，如各种高级语言或数据库等软件，来开发和设计适应于本系统或本部门的生产、科研、教学和事务处理等方面的应用系统。尤其像 IBM-PC/XT、80286/80386 等系列的微机，拥有丰富的应用软件包，世界最著名的几家软件公司，为这一系列的微机开发出许许多多的软件。

但是，这么多的软件中，几乎没有一套软件能够比较全面地处理含有汉字的数据信息。它们必须经过不同程度的“汉字化”技术处理后，才能处理汉字信息。因此，在我国，汉化软件已经是计算机应用、推广和普及过程中的一种社会需求，也是现阶段我国计算机应用领域中的一个重要课题。

1.2 汉化软件的概况

从我国目前的微机应用状况来看，的确需要快速地组织计算机专业高级人员来开发出大批量的、能够处理汉字信息的实用软件产品并投入市场，以适应或满足汉字软件的社会需求。例如，我国自行开发的OFFICE，AutodBASE等软件包，虽然有一定的市场，但从开发周期、数量和使用范围的角度看，还是远远不能满足我国现阶段计算机应用的进程。因为开发一套系统级的软件，需要花费大量的时间、人力和财力。这种方法似乎可行，但却不符合实际。

当然，也有一条捷径：将一些优秀和著名的软件引进后进行汉字化，也就是说，对其进行二次开发。这样，可使软件开发的工作量大大减少，既实用、经济，又行之有效。这种做法加快了软件产品推出的周期，能够较好地满足我国计算机迅速发展的社会需求。

在丰富的应用软件集中，有许多是世界公认的优秀产品，例如：Ashton-Tate公司的dBASE、Borland公司的Turbo系列和Microsoft软件公司的各种高级语言、商口软件和字处理软件等等。这些著名的软件不仅具有一定的代表性，而且，从人类思维的共性来看，其中绝大部分功能是我们能够接受的。这些软件凝聚了许多专家的高技术思想结晶和优良的程序设计结构，直接影响着世界计算机应用水平的发展。虽然处理西文信息与中文信息在某种程度上存在着差异，但就这些软件的最终功能来说，它们是完全能够融合于我国计算机应用系统之中的。

由于计算机软硬件不断地发展和迅速地更新换代。汉化技术至今仍没有一定的标准模式。开发者大都凭借自己的经验，在各种类型的系统环境下进行不同程度的实施。例如屏幕显示卡，从早期IBM-PC机的CGA的中分辨率，发展到MDA、Color400显示卡和高分辨多功能EGA/VGA显示卡。软件方面也从行编辑、全屏幕编辑发展到各种下拉式菜单的多窗口软件。在软硬件迅速更新换代的过程中，许多专业人员作了大量的工作，结合具体环境，对各种系统软件和应用软件进行了汉化工作，以满足各行各业计算机应用的需要。这对于推动我国计算机应用水平的提高和发展起着很大的促进作用。但是，纵观我国整个计算机界软件的汉化情况，尚有许多不足之处，主要有如下几个方面：

1. 缺乏软件汉化的理论指导；
2. 汉化技术未普及，用户较被动；
3. 从软件工程角度来说，缺乏对汉化软件的统一评价标准；
4. 许多汉化后的软件整体结构有缺陷；
5. 没有真正地达到中西文兼容。

上述不足之处受多种因素的限制和影响。因为，无论是系统软件还是应用软件，它们的种类是多种多样的，所使用的环境亦各有差异，各种软件中有关阻碍汉字信息处理的模块不同，其逻辑地址也不同。另一方面，由于所要汉化的软件大都是在目标代码下进行分析和改造的，汉化者基本上是根据经验和本身的技术素质来作二次系统开发工作，各人对计算机系统和指令系统掌握的深浅以及对原西文软件分析的粗细，均直接影响汉化后软件产品的质量。

汉化一个软件，从原理上讲，并不十分复杂，就是把原西文软件或某些代码文件作适当的调整和部分修改，使之能够处理汉字信息。但是具体实施起来，则并非易事。这要求专业人员掌握、了解许多计算机系统软硬件的基础知识、汉字操作系统的功能和工具软件的使用等等。对于初学者来说，一时很难下手。他们应该在对计算机的内部结构、外设接口和汉字操作系统等了解的同

时，多读一些源程序，收集有关的资料，这对于下一步的汉化工作将会有很大裨益。本书对于汉化技术的基础知识也作了一些介绍，并由浅入深地讨论了汉化软件的环境、过程及其方法。

1.3 软件汉化的要求

软件的汉化与软件的分析不同，前者主要是寻找软件中与汉字信息处理有关的部分，对其进行流程分析，针对不同的问题作出不同的修改方案。后者是全方位系统地分析软件的设计思想、结构及其内部逻辑关系。

对于汉化软件这项软件工程来说，我们的最终目的应该是：西文软件能够运行的环境，汉化后的软件同样可以运行，汉字系统的工作环境应该是包含着西文软件同样的工作环境。汉化后的应用软件应不受输入输出系统和操作系统版本的限制，换句话说，就是有较强的兼容性和通用性，在此基础上，达到汉化软件的最高境地。通常，对于一个汉化后的软件的评测标准和要求有如下几项：

1. 对原西文软件的代码变动应尽可能的少，在不丢失原西文软件的功能条件下，取得最佳的汉化效果；

2. 对西文软件内的有关模块实行功能强化，使其兼顾到汉字处理的特点；

3. 对西文软件内的有关子程序实行功能转化，使其从单一处理西文字母转化为对中西文均能处理；

4. 人机界面的信息汉字化。

前三个项目作为一个汉化级别，意味着汉化后的软件已经具备了识别汉字信息的功能，扩展了程序对于字符处理的定义域。第四项作为二级汉化，是针对中西文系统环境下软件所具备的不同的信息吞吐量间的匹配而进行的，并且使人机之间的交流更接近日常语言。

任何一个软件，无论是版本的差异或是功能的差异，其汉化

的原理和目标都是相同的，最终的结果也是一致的。但是，通过分析研究，我们可以看到，各种软件需要修改的程序模块以及与汉字信息有关的部分，它们在内存中运行时的逻辑地址均不相同，现阶段还没有一个非常有效的智能型的汉化软件的工具，绝大多数都是采用 Debug 或 PCTOOLS 等一些动态调试程序针对具体的软件进行汉化工作，经过反复的修改调试，最终成为可以处理汉字信息的软件产品，其方式如图 1-1 所示。

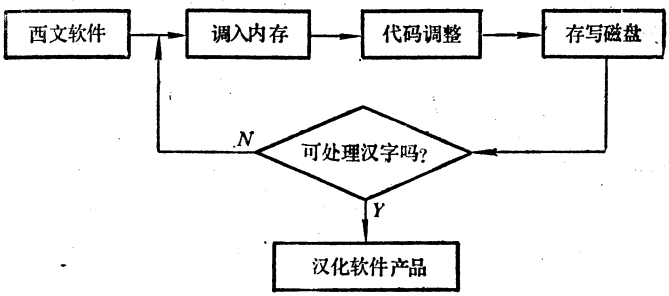


图 1-1　软件的汉化流程

西文软件大多数是在显示器字符方式下运行，尤其是一些字处理软件，所显示的字符由显示卡的字符发生器产生并显示。而汉字系统，如 CC-DOS，是在图形方式下处理汉字信息的。仅从 CRT 屏幕显示这一例子，就说明它无法运行。即使有些汉字系统的环境能够接受字符方式正常显示，但是，其定义的汉字内码也将与原西文软件中的 ASCII 码的定义发生冲突。因此，如果一个西文软件要在汉字系统环境下正常处理汉字信息的话，就必须进行汉化处理，这是唯一的前提。

1.4 二次开发的过程

设计一个软件，是将解决问题的思维过程定义翻译为一种软件表示的过程，而汉化一个软件，则是对该软件表示过程的扩充。因此，在标准软件的规范下完成评测软件汉化的优良程度，

也需要一些原则，才能顺利通过二次开发的进程，这些原则如下所列：

1. 分析原软件的层次组织，以便弄清软件元素之间的控制关系；

2. 在原软件的部分逻辑上，实现处理中文信息的子功能；

3. 从总体上逐级求精，用不同的手段进行各个层次的测试，最终完善为软件产品；

4. 建立二次开发的文档和资料，便于今后的系统维护。

图 1-2 所示说明了二次开发的过程。

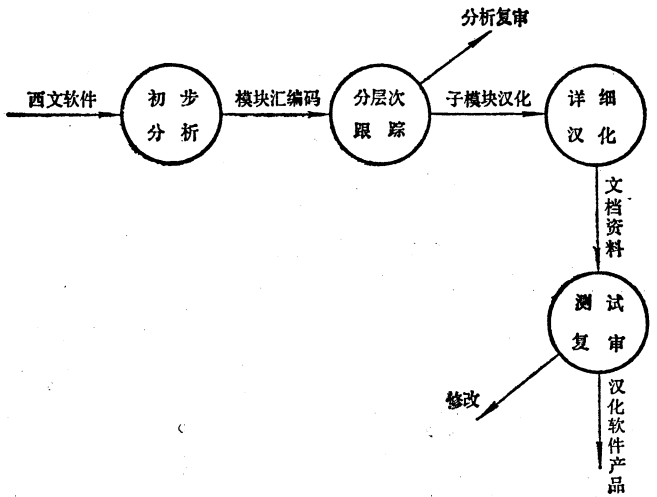


图 1-2 二次开发的过程

在当今世界计算机技术迅速发展的形势下，无论是系统程序员还是软件应用者，都应该尽快熟练地掌握软件的汉化技术。这将有助于系统的升级和软件版本的不断更新，有力地推动我国计算机应用事业的迅速发展。

第 2 章　软件汉化的基础

2.1 中西文软件的开发环境

经过汉化的软件，一般是在汉字操作系统的环境中运行，汉字信息处理的结果和界面都是通过输入输出设备表现出来的，因此，掌握计算机 I/O 接口的技术，对于汉化工作是至关重要的。如果对指令系统、汉字系统的功能调用和反汇编的方法和技巧等有所了解的话，要想分析、改造和移植某一指定的软件就比较容易了。

譬如，程序中与汉字信息处理有关的是键盘输入、屏幕显示、打印输出和信息通信等，纯西文系统与汉字系统在以上几个方面的处理过程是不尽相同的。我们所熟悉的 CC-DOS 汉字操作系统，是在原计算机内基本输入输出系统(BIOS)的基础上开发的，实际上应该称作 CC-BIOS，是系统一级的环境软件，它为我国计算机的普及和使用打下了坚实的基础。CC-BIOS 对原有 ROM-BIOS 中的几个主要服务模块的功能进行了适当的扩充，诸如显示功能 (INT 10H)、键盘功能(INT 16H)、打印功能(INT 17H)和屏幕拷贝(INT 5)等功能，都是以软中断的方式提供给调用者，使得调用程序在此环境下具备了处理汉字的条件。

有了上述条件，只能说 CC-BIOS 初步建立了一个汉字系统的基本运行环境，并不是所有的软件都可以处理汉字信息了。如果要想有效地处理汉字信息，则必须使得那些在 CC-BIOS 环境中运行的软件，毫无阻碍地调用 CC-BIOS 中那些扩充过的服务功能，才能确保正确有效地输入和输出汉字信息。

2.2 汉字系统的基本结构

2.2.1 系统层次及功能

CC-DOS 的设计目的是解决使 IBM 系列的微机具有汉字信息处理环境的问题。汉字信息处理系统(Chinese Information Processing System)的关键在于计算机对汉字代码的数据处理使得人们在计算机上使用汉字和使用西文一样的方便和容易。在计算机系统内部，西文和中文只不过被人为地定义了不同的符号，用不同的信息表示而已。

从信息处理的角度来看，汉字信息的处理和西文信息的处理并没有本质上的区别，两者均是非数值处理。这要求计算机系统具有特定的资源，并得到信息共享、数据库、数据通讯和高级语言等功能的支持。由此，在 CC-DOS 的环境中就可以开发各类中西文兼容的软件。系统的各个层次如图 2-1 所示：

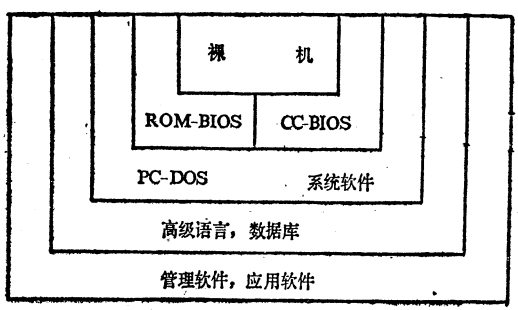


图 2-1　系统层次结构

CC-BIOS 包括如下几个主要的模块和数据：

1. 工作区和数据区；
2. 显示器控制程序；
3. ASCII 字符字模数据；
4. 键盘输入控制程序；

5. 汉字输入码与机内码对照表；

6. 汉字字模点阵数据。

汉字系统的结构是比较清晰的。汉字打印驱动程序是独立的模块，有些系统是根据不同的打印机型号装入不同的汉字打印驱动程序的。

工作区和数据区供系统程序运行中数据加工、传送和设置标志时使用。例如，汉字点阵的加工处理、外码和内码之间的转换等等。标志单元一般用于控制程序的走向，使程序完成预定的运行目标。

显示器控制程序的主要作用是：通过汉字内码，产生相应的汉字字型，根据 CRT 初始化后的不同的显示器类型（如 CGA、EGA、VGA 等）和模式，通过对汉字字模点阵的检索，以图形方式在屏幕上指定的位置显示汉字的点阵信息。

字符数据分为 ASCII 码字模和汉字字模库，一个标准的 ASCII 码由 8×8 或 8×14（EGA、VGA）点阵组成，由点位信息表示。常驻内存的汉字字模，每个汉字由 16×16 点阵 32 个字节表示，该字模库主要是提供给显示输出用的。

汉字输入码——机内码对照表，亦称为汉字扫描表，通常是指某种输入编码方案（如拼音法、五笔字型等）的码表，输入码与码表之间检索和比较的结果产生相应的机内码，尔后，根据这个机内码来定位该汉字点阵字模的首地址。

键盘输入模块可以根据功能键的切换分别或同时输入中文或西文信息，能把用户输入的外码进行有机地变换去检索码表，以达到汉字输入的目的。

汉字打印驱动程序主要解决汉字信息的打印输出。由不同的机内码检索到相应的汉字点阵字模，并在针式打印机或激光打印机上，根据不同的打印控制命令打印出大小不同、字型相异的汉字信息来。

整个汉字系统的功能结构如图 2-2 所示。

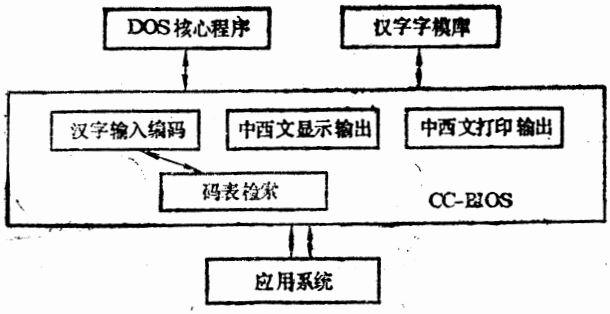


图 2-2 汉字系统功能结构

2.2.2 CC-BIOS 代码系统

作为一个完善的汉字操作系统，必须具备输入汉字、处理汉字和输出汉字三大功能。就汉字系统而言，无论是西文信息还是中文信息，其信息均是以代码的形式流通和传送的。在 CC-BIOS 汉字系统中，为了使各个功能模块处理信息更为方便，汉字的代码在各模块中的定义和名称均不相同，从而在汉字信息处理系统中存在着好几种汉字代码，这些代码构成了一个汉字代码体系。为了适应汉字系统内各组成部分对汉字信息进行处理的不同要求，在 CC-BIOS 内部各个功能模块之间，把这些代码进行了适当的转换和处理。如图 2-3 中的代码交换流程所示。

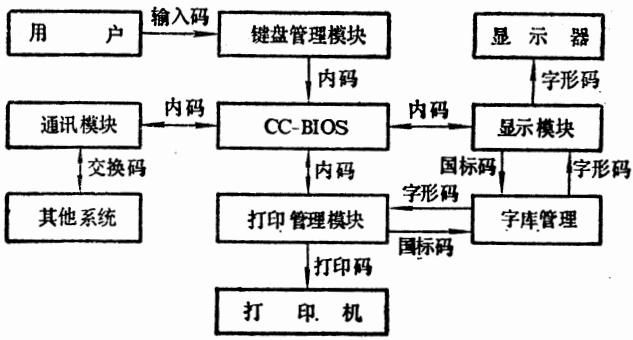


图 2-3 系统代码交换流程

1. 汉字输入码

汉字输入码是为了方便计算机用户把汉字信息输入到计算机内而定义的代码。这种代码必须通过操作员利用键盘等输入设备进行输入，因此说，输入码是位于人机界面之间的代码。相对计算机内部来说，又把输入码称为外码。所采用的输入外码方式，根据编码方案的不同而各有差异。例如，拼音码是直接把拼音字母或数字作为外部编码，根据汉字的发音进行汉字的输入。形码是根据汉字的字根元素从键盘拼形输入或以整字从大键盘直接输入。另外还有流水码，这种码是将数字按某种顺序编码输入，如区位码、电报码等。现阶段已发明了近千种汉字输入法，如纯拼、首尾、大众、五笔、电报及线码等等。众多的输入方法中，用户在输入一个汉字时需要敲键的个数及排列组合是不同的，一般需要2—4键。由于用户的层次不同，用途有别，故一个汉字系统中常配有数种汉字输入方式，以使其能广泛地适应各种用户。

2. 汉字机内码

汉字机内码（亦称内码）是计算机内部对汉字信息进行各种加工、处理的代码，不同的汉字系统，将选用不同的机内码。在一般情况下，不同输入方式，其汉字输入码的长度不等。要使计算机又能够接收，又便于存贮和传送，把多字节不等长的输入码按照一定算法变换为与国标码有一定对应关系的代码是十分必要的。CC-BIOS中采用的是以异形国标码作为其汉字机内码，有关机内码的设计思想将在下一节中作介绍。

3. 汉字地址码

汉字地址码是指存贮汉字字模信息的逻辑地址，汉字是以点阵字节表示的数字信息，它可以存贮在计算机的只读存贮器(ROM，EPROM)中，如一些微机上配置的汉卡等，也可以存贮在磁盘上。当然，也可以装载到随机存贮器的RAM中。汉字字模库存贮的介质不同，汉字信息所在的逻辑地址也不同，因此，地址码的表示方式也就不尽相同，如果汉字装入内存RAM，用内存地址数来表示。若汉字是在汉卡上(EPROM)，则由选片

控制来产生地址码；若汉字库放在磁盘上，其地址码通常由磁盘中磁道某一扇区的数字码表示。由于CC-BIOS的汉字字模库中的汉字字形信息排列序列的规则，使得CC-BIOS的汉字地址码与汉字机内码之间形成一种简单的函数关系，所以很容易实现两者间的转换。

4. 汉字字形码

目前，方块汉字都是以点阵字节方式数字化表示，所以，汉字的字形码是指汉字字形点阵的数字代码。汉字字形的大小和形状不同，其字形码亦不同，这些规格不同的字形码是根据不同的系统设置和输出要求而选定的。目前，我国已颁布了16×16、24×24、32×32和48×48点阵的字模标准。

5. 标准码

1981年，国家标准总局制定并颁布了GB2312信息交换用汉字编码字符集基本集，通称标准码或交换码。交换码是用于汉字信息处理系统之间或者与通讯系统之间进行信息交换的汉字码。这要求汉字交换码必须采用统一的形式。目前，国内计算机系统所采用的标准信息处理交换码是根据有关国际标准制定的，即GB198《信息处理交换用的七位编码字符集》和相应的代码扩充标准GB2311《信息处理交换用七位编码字符集的扩充方法》。标准中规定了汉字信息交换用的基本图形字符及其二进制编码，这些有标准的交换码，称为国标码。

国标码中共收集了7445个汉字和图形符号。其中汉字有6763个，分为两个级别。一级字库为常用汉字3755个，是按汉语拼音字母顺序排列，对于同音字，再按笔划顺序列出；二级字库有汉字3008个，按部首顺序排列，此外，收集有一般符号202个，包括间隔符号、标点符号、运算符号、单位符号和制表符号；序号有60个；数字有22个；英文字母大小写各有26个，共52个；日文假名169个；希腊字母大小写各有24个，共48个；俄文字母大小写各有33个，共66个；汉语拼音符号26个和汉字注音字母37个。

国标码的任何一个符号、汉字和图形都是用两个7位的字节来表示的，一般计算机中，每个字节为8位，因此，最高位以0来表示。国标码有一张代码表，表中的纵坐标表示区，编号为1—94，每个区有94位。表中的横坐标表示位，编号也是1—94，每个区位的焦点表示一个汉字的代码。例如汉字“啊”的区位码是1601（16区01位），其国标码为3021H，把它展开为二进制码的话，是00110000和00100001两个字节。

2.2.3 汉字机内码的设计思想

汉字机内码是汉字系统内部处理和传送汉字使用的代码。汉字机内码的表示方式很多，有两字节法、三字节法和四字节法等。为了节省汉字代码所占的存贮空间和传送时间，CC-BIOS汉字系统是采用两字节法表示的。众所周知，西文字符的机内码均是采用一个字节表示的ASCII码。一般只使用了前7位来表示128个ASCII字符，而高位则或作为奇偶校验或不用，如图2-4(*a*)所示。国标GB2312规定，一个汉字用两个字节表示，每个字节仅使用7位。

为保障汉字系统中西文兼容，在处理汉字机内码时，必须保障西文机内码（即ASCII码）的使用，但也要允许国标码的使用。显然，在一个系统中同时存在国标码和ASCII码，这将会产生二义性。例如：有两个字节机内码的内容分别为30H和21H，它既可以表示汉字“啊”的国标码，又可以表示西文“O”和“!”的ASCII码。这就产生了二义性。因此，在汉字国标码的两个字节最高位分别置1，如图2-4(*b*)所示，用此方法来认识汉字，作为汉字的机内码。这种方法实际是变了形的国标码，称之为异形国标码。例如汉字“啊”的国标码为00110000，00100001，高8位置1后所对应的机内码为10110000，10100001，即B0A1。这样处理，既解决了西文机内码与汉字机内码的二义性问题，又保证了汉字机内码与国标码之间非常简单的对应关系。CC-BIOS就是采用这种类型的异形国标码作为机内码的。

无论采用哪种类型的机内码，其设计思想应满足以下几个原则：

1） 信息的位数少，尽量减少信息的冗余度；
2） 与国标交换码有较简单的对应关系；
3） 排除汉字机内码与西文机内码间的二义性；
4） 便于汉字信息处理中的基本操作和运算；
5） 和原西文系统的兼容性要高；
6） 便于纳入各种程序设计语言中的字符类型中；
7） 尽量便于扩充字符集。

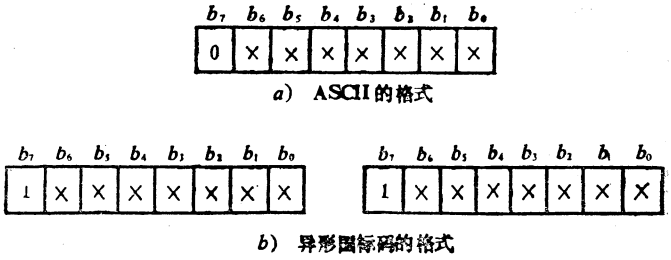


图 2-4

2.2.4 代码转换

在CC-BIOS汉字操作系统中，从输入到输出，每一个汉字或字符都要经过几个层次的转换，如图2-5所示。除了西文ASCII码以外，任何一种输入方式下键入的汉字外码，都要先被转换为区位码，由区位码产生统一的机内码后，再传送到各有关的程序模块中，以进一步得到相应的处理。

区位码和国标码属于计算类的输入编码，其外码（输入码）到机内码的转换是通过计算来完成的。

由于汉字点阵字库是按区位码顺序排列的，因此，无论是拼形还是拼音或其他各类输入法，均是由外码对码表进行扫描后，得到一个内存的逻辑地址，该地址减去码表的首地址的值，就是

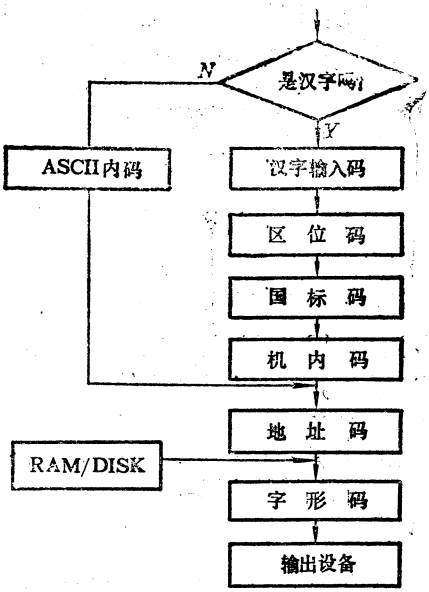


图 2-5 代码转换流程

区位码。这些工作都是在输入模块单元中完成的。下面介绍从输入码到字形码的转换过程及其步骤。

1. 区位码到国标码的转换

例如：输入的区位码是 1601，即 16 区第 1 位，在标准集中是“啊”的代码。区位码转换为国标码的步骤是：先将十进制的区码和位码转换成十六进制的区码和位码，然后再将这个代码的第一字节和第二字节分别加 20H，即可得到国标码，如下面公式所示：

$$\text{国标码} = \underbrace{\left\{\left(\frac{\text{区码}}{16}\right)_H + 20\text{H}\right\}}_{\text{第一字节}}\underbrace{\left\{\left(\frac{\text{位码}}{16}\right)_H + 20\text{H}\right\}}_{\text{第二字节}}$$

$$(\text{啊})_{\text{国标码}} = 1001\text{H} + 2020\text{H} = 3021\text{H}$$

2. 国标码转换为机内码

国标码转换为机内码的方法比较简单，就是把该代码的两个字节的最高位分别置1，即与8080H相“或”即可。

（啊）$_{机内码}$ = 3021H + 8080H = B0A1H

当代码转换为机内码之后，系统就可以方便地对其进行传送、存贮和输出等处理了。

3. 机内码转换为地址码

前面已述，汉字点阵信息是按照区位码的对应坐标排列的。因此，在计算汉字点阵信息的地址时，首先把机内码转换为国标码再转换为区位码，使可以通过算法计算地址码了。CC-BIOS系统中16×16点阵字库是用于显示的，其中，每一个汉字由32个字节组成。为了节省内存，在字库的空白处压缩了8个区。下面介绍用区位码计算汉字地址码的算法，这种算法的结果是计算出某一区位码所对应的汉字在内存中的段地址，它可以用于显示、打印和造字等程序中。

① 区号小于8

地址码 = {(区号 − 1) × 94 + (位号 − 1)} × 2 + (DS值$_H$ + 16)

② 区号大于等于16

地址码 = {(区号 − 9) × 94 + (位号 − 1)} × 2 + (DS 值$_H$ + 16)

例如：当CC-BIOS把汉字库当前的首地址装载到DS = 900H处，输入码转换为区位码是0421时，其地址码的结果是：

地址码 = {(4 − 1) × 94 + (21 − 1)} × 2 + (900$_H$ + 16)

= 25CH + 910H = B6CH

实际地址码为0B6C:0000，即段地址为0B6C，从编移地址为0的起始地址处，顺序取出32个字节信息，便得到了区位码为0421的汉字字形码。

上述地址码的算法用程序实现的话，尤如下面汇编语言子模块所示。

```
CS:276C 81E27F7F    AND     DX,7F7F     ;内码转为国标码
CS:2770 50          PUSH    AX
CS:2771 80FE29      CMP     DH,29
```

```
CS:2774 7502        JNZ     2778
CS:2776 B626        MOV     DH,26       ;6<-->9区变换
CS:2778 80FE30      CMP     DH,30
CS:277B 7203        JB      2780
CS:277D 80EE08      SUB     DH,08
CS:2780 80EE21      SUB     DH,21       ;国标码转为区位码
CS:2783 B05E        MOV     AL,5E       ;AL=94
CS:2785 F6E6        MUL     DH          ;区号乘以94
CS:2787 80EA21      SUB     DL,21
CS:278A 32F6        XOR     DH,DH
CS:278C 03C2        ADD     AX,DX       ;区号加位号
CS:278E D1E0        SHL     AX,1        ;乘以2
CS:2790 03067500    ADD     AX,[0075]   ;汉字库首址
CS:2794 8BD0        MOV     DX,AX       ;DX得出地址码
CS:2796 58          POP     AX
CS:2797 C3          RET
```

2.2.5 汉字编码表

目前，输入编码的方案不下几百种，除了计算类编码外，绝大多数的汉字输入法都是需要首先建立一个码表，即输入码-机内码的对照表，然后，用查表的方式输入汉字。当用户输入一个外码后，系统把接受到的外码，根据层次，分别在该表中进行检索。例如，CC-BIOS V2.1 中拼音和首尾码表就是这样，不过，为节省内存，其拼音和首尾使用的是同一张表，每个汉字 4 个字节，这 4 个字节作为一个表项或者说一个字段，每个表项的内容就对应着该汉字的输入码。表项的顺序号是按其对应的汉字在字库中的次序确定的。对照表的表项结构如图 2-6 所示。

由于首尾码和拼音码的码长均为 3 字符，且有一个字符是公用的，故每个表项只含 5 个字符的内容，可设这些字符均由 a — z 26 个小写字母组成，显然，可以用 00001—11010 的次序（即 5 位）来表示 a — z，因此，每个表项需要 25 位来表示 8 个输入码字符，也就是说，每个表项用 4 个字节。

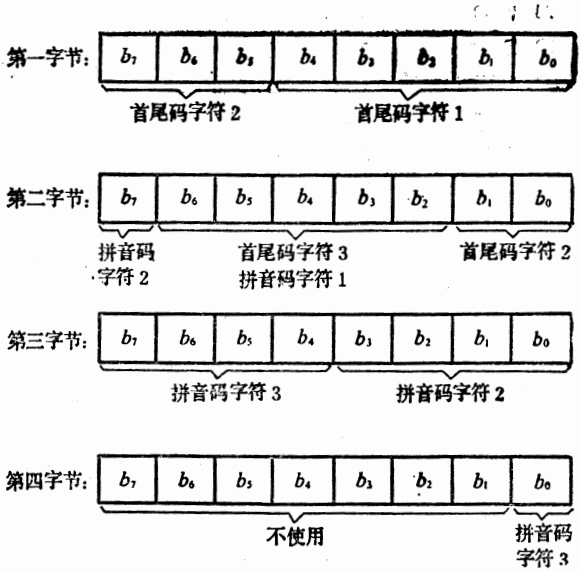


图 2-6　拼音/首尾码表结构

在查对照表时，对于首尾码，是从表项中第一字节开始部分查（根据当前输入码缓冲区中的字符数确定何处结束），而对于拼音码，在查表前，要先对输入码缓冲区中的内容进行处理，把其转换成表项内容的对应形式，然后再与表项内容进行对照比较。显然，码表扫描的方式随输入码缓冲区中的字符数而定。当输入码缓冲区中的字符数分别为1、2和3时，其对应的查找方法分别为1字符查找重码法、2字符查找重码法和3字符查找重码法，向用户提示选择。

例如：汉字“通”的码表表项内容是 F6 D1 B9 01（4个十六进制的字节），把该表项用二进制展开为：

11110110　11010001　10111001　00000001

可按照图2-6的结构进行划分，其结果应该对应的输入码为：

首尾码：vot

拼音码：ts

快速码：vots

2.2.6 汉字库与字符点阵

对于西文字母和数字来说，由于其数量少、结构简单，故点阵信息也较少。一般都使用8×8点阵，在汉字环境下，西文字母点阵信息的首地址，由中断1FH指定。目前推出的80286/386等微机，配置了高分辨率的显示器，该机所采用的常规字符点阵大小为8×14点阵。因此，一些汉字系统也相应作了调整，使得在中西文方式下其字符显示的形状不变。

汉字库的主要作用是向汉字输出设备(如显示器、打印机等)提供汉字字形数据。从汉字信息在系统内部流程的角度看，汉字库是一个处于汉字信息输出结点上的代码转换器，它吸收的是待输出的机内码，吐出的是字形码。汉字库的设计应该满足三个指标：字形质量高、读取速度快和系统开销小。

字形库点阵数愈大，汉字的质量就愈高，如48×48、64×64、128×128以及更大的点阵，但是大点阵字库的存贮介质的占用量是非常惊人的。因此，字库压缩技术以及汉字点阵的无级变倍和平滑处理技术是当前汉字信息处理领域中的一项重要课题。由于目前的微机所配置的内存和DOS操作系统管理的局限性，对于用户来说，内存是有限的。汉字字形库装入内存后，限制了许多大型软件的使用。因此，推出了许多变通技术，例如，利用汉卡把字形库放置或压缩在EPROM芯片中，使其不占用DOS管理的实际内存。或者把部分或全部字形字库放在硬盘上，供系统读取。自80286/80386微机推出后，可以采用保护方式把汉字库装载到扩展内存中。为用户执行大型软件提供了方便。

CC-BIOS汉字系统使用的16×16点阵字形库主要用于显示，为了显示(向屏幕传送点阵)处理的方便，字形库内的信息是横向排列的。而24×24点阵或更高一些的点阵字库，通常是为打印输出提供的，为了和打印机针头对应，其字模信息是纵向排列的。有关汉字点阵的数据结构，将在汉字打印输出一章中详细介绍。

2.3 CC-BIOS 基本输入输出功能

CC-BIOS 汉字系统是在原有的 ROM-BIOS 基础上进行了扩充，使得系统具有汉字信息处理的功能。由于各种 CC-DOS 不尽相同，版本很多，不能一一列举。本文针对我国目前使用得最广泛的 CC-BIOS V2.1/V4.0 的功能进行讨论，便于读者在汉化软件时作为参考。

2.3.1 显示系统的功能

显示模块的功能主要是提供向屏幕上显示信息的服务，其中断类型为 INT 10H，由近 20 个子模块组成。CC-BIOS 支持文本和汉字的显示，但是，在汉字环境下，屏幕是工作在图形方式下的。根据微机上配置的 CRT 类型的不同，将会有不同的显示程序进行服务。因此，各种显示卡的控制地址口，显示分辨率和视频缓冲区 RAM 大小的不同，必须经过设置、判别以至装入相应的模块，才能保障程序的正常运行。例如，CGA 显示卡的分辨率为 640×200，其视频段地址在 B800 处。MDA 单色卡的分辨率为 720×350，其视频段地址在 B000 处。而 EGA/VGA 卡有各种分辨率，从 320×200，640×350，640×480 到 1024×768 等近 30 几种，其视频段地址为 A000，如果汉字以 16×16 点阵字模显示的话，分别可以设计出 11 行、21 行、25行和 30 行等汉字显示程序。

CC-BIOS 汉字系统分为两种屏幕方法来处理汉字显示，即实屏和虚屏两种屏幕概念。实屏是真正的屏幕，它面向显示器控制模块。虚屏是面向用户和应用程序的。因为实屏是真正的物理屏幕，所以实屏上显示内容被存放在视频缓冲区内，它所存贮的是点阵图形信息。而虚屏是虚设的非物理的存贮单元，系统在这个虚设的存贮单元中存放对应实屏内容的代码和属性信息。也就是说存放的是 ASCII 码和汉字机内码及其属性。系统用这种方法

以便于处理点阵信息，其显示流程如图 2-7 所示。

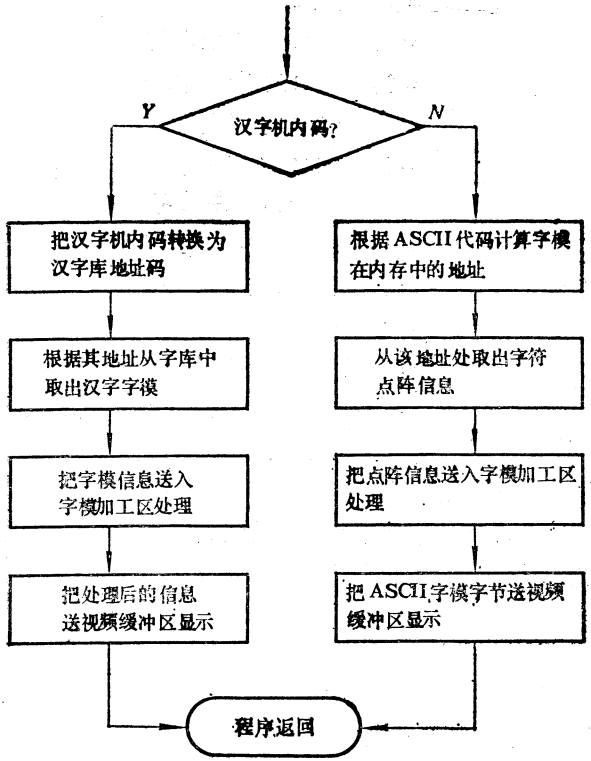


图 2-7　汉字字符显示流程图

2.3.2 显示模块的功能调用

显示器控制模由 INT 10H 中断处理程序提供给用户使用，其中有汉字和字符的显示、屏幕滚动、光标定位等近 20 个子功能。这些功能的调用方法是将功能号放在寄存器 AH 中，在其他寄存器中设置指定的入口参数，然后向 CPU 发送 INT 10H 指令，即可在屏幕或出口参数寄存器中得到调用后的状态和结果。下面列出了 CC-BIOS 显示模块中各功能的调用方法。

1. 设置显示方式及屏幕初始化

输入参数：[AH]=00 ；功能号

[AL]=显示方式

=00　40×25 黑白字符

=01　40×25 彩色字符

=02　80×25 黑白字符

=03　80×25 彩色字符

=04　320×200 彩色图形

=05　320×200 黑白图形

=06　640×200 黑白图形

=07　80×25 单色字符

=10H　640×350 彩色图形(EGA)

=12H　640×480 彩色图形(VGA)

该模块功能主要是对屏幕 CRT 控制器进行初始化操作，建立以 AL 寄存器所指定的屏幕显示方式。其内容包括写模式寄存器端口，系统以一组对应的参数通过索引寄存器寻址后送入数据寄存器中，形成当前屏幕的光栅状态后，则把有关控制器的一些参数复制到内存低端的数据区去。同时，该功能将清除全部屏幕信息（包括汉字提示行）。

2. 设置光标类型

输入参数：[AH]=01 ；功能号

[CH]=光标的起始扫描线

[CL]=光标的结尾扫描线

该功能仅在文本显示方式下才有效，它用来改变当前屏幕上的光标大小。系统将调用者送入 CX 寄存器中的光标始尾扫描线号去设置 M6845 控制卡。实际上将 CH 值写入 M6845 的 R_{10} 寄存器端口，CL 值写入 R_{11} 寄存器端口，然后再把这两个参数映像到数据区中光标类型单元(0040:0060)处。

3. 设置光标位置

输入参数：[AH]=02 ；功能号

[DH]=行号

[DL]=列号

[BH]=页面号

该功能是把光标定位到DX指定的行、列和BH指定的那个有效页面上，在图形方式下，BH=0，当调用返回时，光标在新指定的屏幕位置上闪烁。

4. 读当前光标位置和类型

输入参数：[AH]=03 ，功能号

[BH]=页面号

输出参数：[DH]=当前光标所在行

[DL]=当前光标所在列

[CX]=光标类型号

该功能仅把ROM-BIOS数据区中光标类型单元读入CX，并将BH所指定的那个显示页面的光标位置信息读入到DX寄存器中。须注意的是：BH指定的页面值必须有效。40×25字符方式有0—7个页面，80×25字符方式有0—4个页面，而汉字图形方式下只有0号页面有效。

5. 读光笔

输入参数：[AH]=04 ，功能号

CC-BIOS保留原ROM-BIOS功能。

6. 选择当前有效页面

输入参数：[AH]=05 ，功能号

[AL]=新页面号

该功能仅在文本模式才能生效。系统进入调用后，首先把AL寄存器中的参数写入BIOS数据区的页面单元(0040:0062)，经过页面调整后再写入显示缓冲区首址单元(0040:004E)。取出对应的光标位置值，转换为字符地址，再送入M6845芯片的R_{14}、R_{15}寄存器端口，将产生一个新的页面和光标位置，屏幕转换为新的有效显示页。

7. 上滚当前显示页

输入参数：[AH]=06 ，功能号

[AL]=滚动的行数，AL=0时，清除窗口
[BH]=字符方式为窗口填充属性
[CH]=窗口左上角行
[CL]=窗口左上角列
[DH]=窗口右下角行
[DL]=窗口右下角列

该模块的主要功能是使得屏幕信息在CX和DX所指定的窗口内向上滚动AL中参数所指定的行。当输入参数的右下角小于左上角时，将不会发生任何操作。若参数超出屏幕限制时，将会自动调整参数。如果CX和DX的坐标不在屏幕的始末位置，那么，窗口之外的信息不滚动。在屏幕窗口上滚后，窗口的底部有AL行的空白区，其色彩由BH属性值决定，因而，窗口可以设置为反示、闪烁和各种色彩等。当AL=0时，是清除窗口内容的操作，也可用它作为清屏之用。

8. 下滚当前显示页

输入参数：[AH]=07 ，功能号
[AL]=滚动的行数，AL=0时，清除窗口
[BH]=窗口填充属性
[CH]=窗口左上角行
[CL]=窗口左上角列
[DH]=窗口右下角行
[DL]=窗口右下角列

该模块功能与06号功能相同，只是在调用之后，屏幕将下滚AL行，下滚后的窗口顶部清为AL行的空白区。以上这两种屏幕滚动功能，在汉字状态下，最大窗口将不会清除提示行的信息。

9. 读当前光标处的字符和属性

输入参数：[AH]=08 ，功能号
[BH]=页面号

输出参数：[AL]=当前光标处的ASCII代码值

[AH]=该 ASCII 码的属性

该功能能够把当前屏幕上光标所在位置上的字符和属性字节取出来。在文本方式下，系统将直接到视频缓冲区中取出，若在汉字图形方式下，则是把虚屏中对应地址的字符与属性取出，返回给AX寄存器。其中，高位是字符属性，低位是 ASCII 代码。

10. 在当前光标位置显示字符和属性

输入参数：[AH]=09　　；功能号

[AL]=要显示的字符代码和汉字机内码。

[BH]=页面号

[BL]=字符和汉字的属性

[CX]字符的个数，CX≠0

该模块根据当前光标的位置计算出视频 RAM 内的偏移地址，并在该地址处写入 CX 个字符和属性，显示字符后，光标位置不变。在汉字状态下，显示一个汉字需调用两次该模块，并且 CX 值无效。汉字的机内码为 A1—FE。根据 CRT 不同的配置，可显示出各种前景和背景色彩以及反示的汉字信息。

11. 在当前光标位置显示字符

输入参数：[AH]=0AH　　；功能号

[AL]=要显示的 ASCII 代码和汉字机内码

[CX]=字符的个数

[BH]=页面号

该功能与 09 号功能基本相同，只是不必定义属性，即无显示属性功能。

12. 设置彩色调色板

输入参数：[AH]=0BH　　；功能号

[BH]=所置调色板 ID 号

[BL]=和 ID 号一起使用的彩色值

该功能需要根据具体的 CRT 的配置来设置 BX 寄存器的值。通常是在 CGA 卡的 320×200 的图形方式下有效，即：

当 BH=0 时，BL 包含背景色彩(0—15)

当 BH = 1 时，BL 选择调色板(00—01)

若在 Color400，EGA/VGA 的图形方式下，则要根据其随机手册中的方法进行设置。

13. 在图形方式下显示一个点

输入参数：[AH] = 0CH ，功能号

[AL] = 点素值，高 8 位为 1 时，其值与原地址值异或

[CX] = 点的横坐标位置

[DX] = 点的纵坐标位置

该模块的功能是在屏幕上由 CX 和 DX 指定的位置(X，Y)显示一个 AL 给定的彩色点素。其中 X 坐标最大位置为 640，Y 方向的坐标界限根据不同的 CRT 卡纵向扫描线决定。当位置确定后，系统将测试 AL 值的最高位，若为 0 时，将写入一个点，若为 1 时，该值则与原来地址处的点素值逻辑异或后显示一点。

14. 在指定的位置读一个点

输入参数：[AH] = 0DH ，功能号

[CX] = 横坐标位置

[DX] = 纵坐标位置

输出参数：[AL] = 读取的点素值

该模块功能是在 CX 和 DX 指定的屏幕坐标位置处，从视频缓冲区对应的地址取出一个点素值送入 AL 寄存器。它也是根据不同的 CRT 卡，如 CGA、MDA、CGE400 和 EGA/VGA 等，由不同的 BIOS 程序读出。在一些屏幕硬拷贝程序中，使用该功能的情况比较多。

15. TTY 方式显示字符

输入参数：[AH] = 0EH ，功能号

[AL] = 要显示的 ASCII 代码和汉字机内码

[BH] = 页面号

该模块功能较强，它能自动识别 ASCII 控制码，如响铃、退格、回车和换行等，并且采取相应的方式进行处理。当显示完

一个字符或一个汉字后，光标将自动移位到下一个位置。当信息的显示超过最末一行时，系统可以自动滚行，且光标置在新的空行的行首。以后整个新行的显示属性取自上一行最后一个字符。

16. 取当前显示方式和有效页面

输入参数：[AH]=0FH　　；功能号

输出参数：[AL]=显示方式号

[AH]=当前屏幕显示列数

[BH]=当前所在的有效页面号

该模块仅仅是到BIOS数据区取出几个有关的数据信息，返回给调用者。因为，每当用0号功能进行屏幕初始化时，总要把初始化的结果送给数据区，以便将来使用它们。如在0040:0049单元中就保存着当前屏幕显示方式号。

17. 汉字提示行显示

输入参数：[AH]=10H　　；功能号

[DL]=要显示的字符和汉字机内码

[CX]=要显示的字符数

[AL]=子功能号

=00 清除提示行

=01 在提示行当前光标位置显示字符

=02 设置提示行光标

=03 TTY方式显示提示行上字符

该功能模块作为汉字系统的提示行管理，分为4个子功能。0号子功能是把提示行的实屏全部清除，光标设置在提示行的行首，并且显示一根提示行分隔线。1号子功能是清除原来的光标，调用提示行字符显示功能，实现字符和汉字的显示。2号子功能是把原光标位置的参数转换成与提示行对应的视频缓冲区地址，并清除原光标。把新光标位置再转换为提示行对应的视频缓冲区地址，并显示光标及保存当前光标位置的参数。3号子功能与0EH号功能相似，不过，该子功能仅处理提示行的显示。如退格、报警和显示字符，每显示一个字符，光标自动后移一个字符。

需要注要的是：10H 号功能与 EGA/VGA 或 Color400 显示卡中的 ROM-BIOS 的 10H 号功能有所冲突，在汉化软件时，要作适当的调整。

18. 写汉字字模

输入参数：[AH]=11H ，功能号

[DX]汉字机内码

[BP:BX]=点阵字模在内存缓冲区的首地址

该功能是把汉字机内码所对应的 32 个汉字点阵信息传送到字库（在内存中）相应的地址去。大多用于造字软件中的字模传送。

19. 读取汉字字模

输入参数：[AH]=12H ，功能号

[DX]=汉字机内码

[BP:BX]=获取汉字字模的缓冲区首址

输出参数：[BP:BX]=所读取的 32 个点阵数据

该功能按DX寄存器中给定的汉字机内码，可在 BP 段地址、BX 偏移地址处，得到对于该汉字机内码的 32 个点阵信息。该功能常用于打印和造字软件中，取汉字点阵信息时，不再需要计算汉字的地址码了。

2.3.3 输入系统的功能

Intel 系列的微机控制外部设备的方式均是以向 CPU 发出中断指令来实现的。键盘输入的控制过程是：系统通过 9 类一级硬中断把从键盘键入字符的 ASCII 码及其扫描码存写入键盘数据缓冲区中，由 16H 类中断服务程序读取，最后把读取到的代码回送给调用程序，并且随时调整键盘缓冲区的首尾指针。实际上，9 类中断程序与 16H 类中断程序是通过键盘数据缓冲区和有关状态单元来进行信息传递的。

键盘输入系统的主要功能是：对系统程序和用户程序要求从键盘输入字符的请求进行处理。它是 CC-BIOS 键盘管理模块的

主体，具有对汉字代码进行处理的功能，如图 2-8 中管理流程所示。

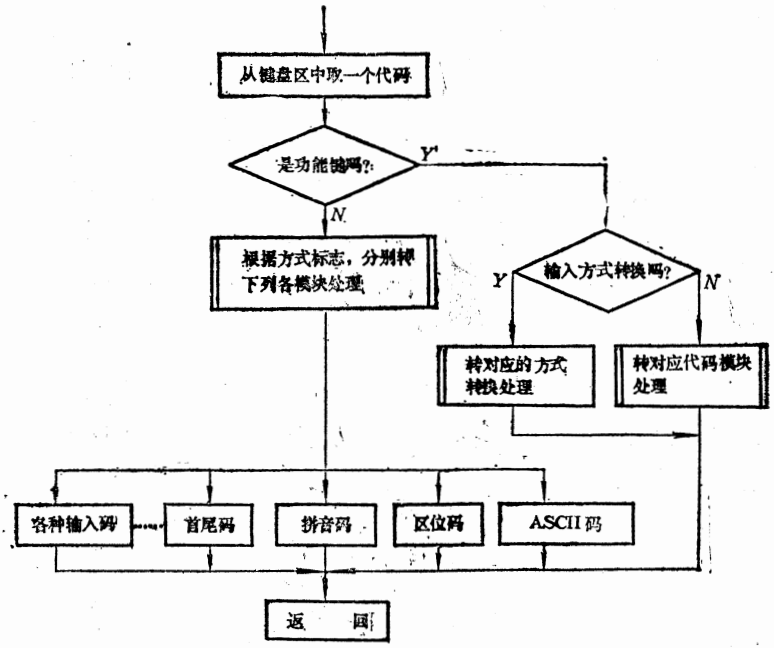


图 2-8　汉字输入管理流程

CC-BIOS 汉字系统具有几种常用的输入方式，并通过 Alt-Fn 功能键进行切换。在指定的某种方式下，系统把用户输入的外码进行转换后，根据计算或查表方式，找到所对应的汉字。若该汉字有重码的话，系统将在提示行上显示重码汉字，以供用户选择。

CC-BIOS 汉字系统主要分为两个层次，一层与具体的物理设备无关，称为高层，另一层与具体的物理设备相关，称为低层。高层软件通常是公共的，即通用部分。低层是具体的、专用的程序。这里所指的专用只是随具体设备的参数变化而变化。

CC-BIOS V4.0 汉字系统在原来的汉字系统基础上作了较大的改进，它支持十几种汉字编码的输入方式、各类显示器和打

印机。在实际应用中，不同领域中不同层次的操作人员会对汉字输入方式有不同的要求。CC-BIOS V4.0 汉字系统给用户提供了一个较好的选择余地。

当键盘有输入发生时，产生9类中断，其所接收的扫描码通常送入AX寄存器的高位AH中，而相应的ASCII码由译码表转换后，送入低位的AL寄存器中。如果输入的是扩展码，即功能键、组合键等，那么，AH中仍存放其扫描码，AL寄存器却置为0，最后把这两个代码字节存入键盘缓冲区，供INT 16H读取。这两个代码字节（有时称“字节对”）通常有以下三种组合：

1. 0000H 高位与低位均为0

当字节对的高位与低位均为0时，表示按下的键没有定义或者是非法组合键，BIOS服务程序可用响铃提示或直接返回，不作任何处理。

2. ××00H　　高位××=扫描码；低位=0时

当字节对的高位字节=01—FFH和低位=0时，系统将在译码表中搜索。如果所按下的键在表中有定义，则表示输入的是一个扩展功能键。

3. ××××H　　高位××=扫描码；低位××=ASCII代码

当字节对的高位字节=01—FFH和低位字节=01—FFH时，BIOS在内部进行译码查询，若该字节对已有定义，则将取出当前键入的有效的ASCII代码和扫描码，说明输入的是常规ASCII键。

无论是西文输入还是汉字输入，键盘缓冲区的信息均是通过ROM-BIOS的INT9写入的。系统的键盘缓冲区KB-BUFFER占用内存32个字节单元，其地址在0040:001E—003E处。因为每个键都是由两个字节的代码组成，因而该缓冲区内可盛纳15个键的有效代码，这足够保存一个块速打字员输入的字符。每当写入键盘缓冲区一对键盘码后，系统就调整缓冲区的尾指针

BUFFER-TAIL，使其尾指针的地址值加2，再与缓冲区末端地址值BUFFER-END比较，直到两值相等时，则重新将缓冲区指针调整到起始地址BUFFER-START处。由此可以看出，键盘缓冲区的末端和始端相连，是循环使用的。键盘缓冲区及其指针定位情况如图2-9所示。

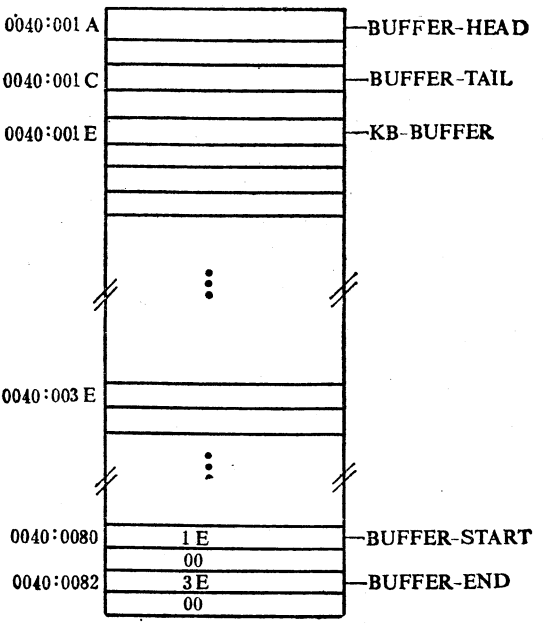


图 2-9 键盘缓冲区分布

读取键盘缓冲区的代码数据是通过INT 16H键盘I/O功能实现的。该模块的0号功能把当前键盘缓冲区的首指针BUFFER-HEAD处的两个字节送入AX寄存器，然后把首指针值加2后返回，如果有汉字内码标志时，需要调用两次，使调用程序得到一个汉字两个字节的机内码。

键盘缓冲区的尾指针总是指向将要写入的位置，而首指针总是指向要读出的位置。一旦两个指针值相等，则表明缓冲区已空，即停止数据的读取，程序将处于循环等待输入的状态。

2.3.4 键盘模块的功能调用

不同版本的汉字系统，其16类中断程序由不同的功能子模块组成，但是有3个最主要的功能。通常，功能号用AH寄存器指定，在对应的寄存器中送入指定的入口参数后，发INT 16H指令即可完成功能调用。

1. 从键盘缓冲区读取一个字符代码

输入参数：[AH]=0　　；功能号

输出参数：[AL]=输入键代码和汉字机内码

　　[AH]=输入键的扫描码

该模块功能是在AX中返回一对输入的扫描码和ASCII代码，如果在汉字输入方式下，AL寄存器将返回一个汉字机内码字节，并且自动调整键盘缓冲区的首指针。

2. 读当前键盘状态

输入参数：[AH]=01　　；功能号

输出参数：ZF标志位状态

　　ZF=1 键盘有输入

　　=0 键盘无输入，缓冲区空

不管键盘缓冲区状态如何，这个功能调用都会立即返回，但它不会改变键盘缓冲区中任何指针状态和数据。它仅仅测试当前缓冲区是否为空，用标志寄存器的Z标志返回给调用者。词组输入和读取汉字机内码程序中常用此标志来判别CC-BIOS键盘缓冲区的机内码读取的状态，以致按一个选择键可以输出一个或多个汉字。

3. 读取当前特殊功能键状态

输入参数：[AH]=02　　；功能号

输出参数：[AL]=当前特殊键状态

```
7 6 5 4 3 2 1 0
. . . . . . . 1    右边Shift键按下
. . . . . . 1 .    左边Shift键按下
```

7	6	5	4	3	2	1	0	
·	·	·	·	·	1	·	·	Ctrl 键按下
·	·	·	·	1	·	·	·	Alt 键按下
·	·	·	1	·	·	·	·	Scroll Lock 键按下
·	·	1	·	·	·	·	·	Num Lock 键按下
·	1	·	·	·	·	·	·	Caps Lock 键按下
1	·	·	·	·	·	·	·	Ins 键触发

该功能实际上仅把数据区 0040:0017 单元的内容取到 AL 寄存器中，返回给调用程序，以说明当前的键盘各类换码的状态。在调用程序中可用 TEST 和 AND 等指令按位测试 AL 寄存器各位所指定的换码键状态。

4. 改变或重置汉字输入码表

输入参数：[AH]=03　　；功能号

[BP:DX]=新的汉字输入码表的地址

该模块是一个数据块传送功能模块，利用该功能可以把其他新的输入码表(不包括程序)传送到 CC-BIOS 的码表区。但是，系统用的是覆盖技术，当装入其他输入码表后，将会把原有的拼音和首尾输入码表覆盖，当要再使用它们时，必须重新装入。用此功能传送新码表时，其码表的数据结构一定要与原拼音和首尾码表的结构相同，否则，系统程序将无法对其检索。

2.3.5 增加新输入编码的方法

CC-BIOS 汉字系统虽然给广大计算机用户提供了良好的人机界面，但由于某些限制，至今还没有真正解决汉字输入问题。因此，许多编码人员正不断地研制各种汉字输入的编码方案。怎样把一个设计好的汉字编码方案装入系统，这是广大编码设计人员非常关心的问题。这里将讨论一种“装挂”式汉字编码输入模块的程序设计思想、步骤和实现的方法。由于汉字编码方案的不同，其编码检索部分亦不相同，因此，检索部分的程序将由编码方案而定。但是程序装载和键盘输入的控制部分，大体上是相同的，在此，将着重讨论。

这种编程方案主要有三个特点：①不需要对原 CC-BIOS 系

统作深入的分析和了解；②对原汉字系统不做任何改动；③支持各种不同版本(V2.0—V4.0)的 CC-BIOS 汉字系统。

1. 编码输入程序的设计思想

在 CC-BIOS 汉字系统中，每个汉字均由两个高 8 位为“1”的ASCII码表示。对于屏幕显示或打印输出等外部设备一次调用系统的控制程序，仅接收 AL 寄存器内的一个 ASCII 码。因此，对于一个汉字来说，键盘控制程序必须返回两次，才能得到它的一对ASCII 扩展码。其原理是：第一次先取出该汉字的高位内码送 AL 后，程序便中断返回，第二次进入中断并不执行键盘处理，而是从内码缓冲区中取出该汉字的低位内码至 AL 寄存器返回，显示程序会把键盘中断返回的这对汉字机内码有机地安排后，在屏幕上显示该汉字的点阵信息。根据上述分析，倘若要在原 CC-BIOS 系统中装接新的汉字编码输入程序时，需要考虑以下几个编程要点：

(1) 新编码功能选择键及其标志单元的处理；

(2) 键盘缓冲区中读取输入码和调整首尾指针处理；

(3) 重新设置16H类中断向量并且保存原 CC-BIOS 的16H类中断向量地址；

(4) 测试键盘状态及取汉字第二码功能；

(5) 原 CC-BIOS 中断 16H 接口处理；

(6) 键盘缓冲区首尾指针的反向调整；

(7) 汉字输入码和非法键的处理；

(8) 回车、退格及空格键等的特殊处理；

(9) 由计算法、查表法和其他方式进行汉字机内码的识别；

(10) 提示行的信息提示和词组方式处理。

概括地说，汉字输入处理系统一般可分为 3 个软件层次：①程序控制部分，它包括设置标志参量及引导程序的走向；②汉字识别部分，包括外码/内码转换、计算检索和查表等功能；③提示信息和机内码输出返回功能。对于外码与内码的转换方式、码本的检索过程，需要根据具体的汉字编码方案来确定其程序的实现，

而控制部分一般都大同小异。其方式和基本原理如图2-10的流程图所示。。

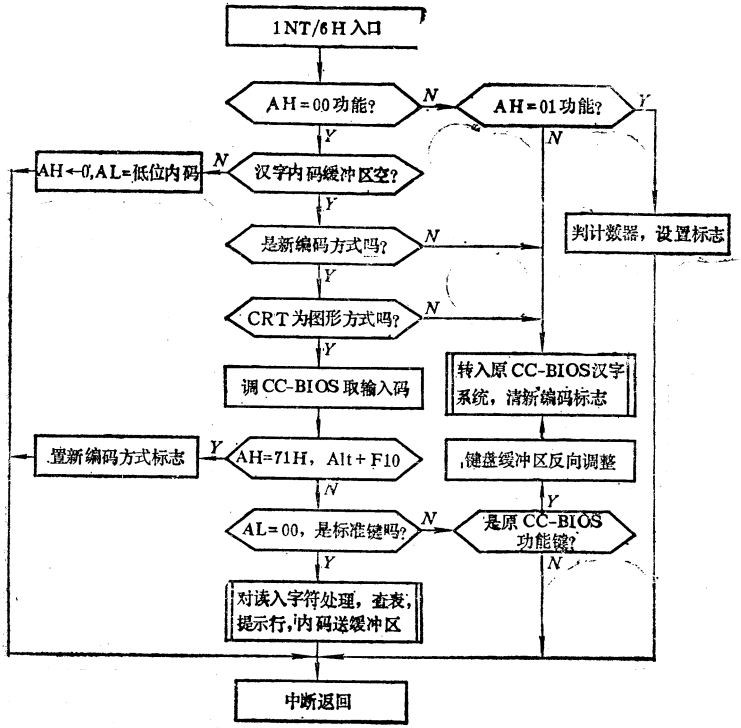


图 2-10　装接新编码输入程序流程图

在着手设计一个新的汉字编码输入程序时，先要确定一个功能键。譬如：用 Alt-F_{10} 组合键来选择新加入的输入方式，若该编码方案中有词组功能，还需定义一个词组转换功能键。除此之外，在程序数据区中要定义相应的标志单元和一些数据缓冲区。标志单元用来设置判别标志字节的控制程序的处理目标。数据缓冲区用来保存外码、内码、重码、计数值和有关的地址指针等。对于不同的编码方案，可根据需要另外定义一些必要的单元，如码本区、字模缓冲区和分层检索标志等。

2. 编码输入系统的实现

程序可用宏汇编语言编程，其速度快、效率高，是开发系统软件的有效工具。各模块的原理和功能实现如下所述：

(1) 在实现新编码输入方式的同时，必须保留原 CC-BIOS 键盘中断的所有功能，并且不修改其任何指令。也就是说，把多种输入方式均作为一个并行系统。因而，当装入新编码程序时，先用 DOS 的功能调用把原 CC-BIOS 的中断 16H 的向量指针取出，保存在内存单元中，并把新编码的输入程序首地址，重新设置到 16H 类中断向量处。当系统需要使用原 CC-BIOS 的功能时，可利用段间跳转或长调用指令转入执行。

(2) 为了有效地从键盘缓冲区内读取输入字符代码，应编制一个读键盘模块。另外，占用中断向量的方法是不可取的，这样会降低系统的兼容性。实际上，这个模块类的 ROM-BIOS 的 16H 中断读代码功能，即读一个输入代码和扫描码到 AX 寄存器，随之调整键盘缓冲区的首尾指针。

(3) 功能键切换汉字输入方式是调用原 CC-BIOS 键盘模块完成的，在此仅把它作为一个子程序调用。如果程序测试到是新编码输入方式的功能键 Alt-F_{10}，返回之前在标志单元中置好标志，并调用 10H 类中断的提示行 TTY 显示功能，向用户提示当前输入方式的提示符。

(4) 程序设计时，还要考虑一个键盘缓冲区指针反向调整模块，其作用是把缓冲区的首指针逆向推进两个字节。例如，当前是在新编码的输入方式下操作，当按下功能键 Alt-F_3 要转入原 CC-BIOS 拼音码的输入方式时，系统从键盘缓冲区取出对应的扫描码后，即刻调整了首尾指针。若再转入原 CC-BIOS，此时，键盘缓冲区的首尾指针已相同，系统无法再次读取到刚才键入的代码，就不会作相应的处理。因此，在程序转入原 CC-BIOS 之前，必须把键盘缓冲区的首指针反向调整两个字节，使得指针仍然对准 Alt-F_3 的输入码，并置 AX = 0，清除 Alt-F_{10} 的标志单元后再进入原 CC-BIOS 执行。

（5）当输入的字符不是扩展功能键时，便作为外码处理，处理的方式视汉字编码方案而定。但要考虑回车键、退格键和非法键均要作特殊处理。而且，值得注意的是：无论汉字编码方案是采用计算法还是查表法，有重码或无重码，在没有换算出指定汉字的机内码之前，程序不能中断返回。应该在"等待键盘输入"子模块中循环等待。直到操作者输入了最后一个码或选中了指定的汉字后，程序才在AL寄存器中带着该汉字的高位内码中断返回。当系统再次调用键盘中断时，再取出低位内码并清内码计数单元返回。这样才能把一个完整的汉字机内码送给相应的输出设备。

（6）提示行操作方式如下：在装接的新编码输入程序中，若需要提示行操作功能时，可调用CC-BIOS中断10H中的10H号功能。只要把提示行要显示的字符码送入DL寄存器，CX为显示字符计数，AL指定操作类型，发INT 10H指令即可实现。

要在CC-BIOS汉字系统下装接新的汉字编码的输入方式，无论编码方案如何复杂，其控制部分的逻辑均可套用表2-1所列出的宏汇编程序。该设计方法系统兼容性强，无须修改原系统程序，这将会给研制汉字输入编码人员节省大量工作，其兼容性也将为推广或普及各种汉字输入方式提供了良好的条件和环境。

控制部分程序清单　　表 2-1

```
CODE    SEGMENT PARA
        CRTMODE          EQU   049H    ;CRT模式指针
        ASSUME  CS:CODE, DS:CODE, ES:CODE, SS:CODE
        INT160FF         DW    0       ;INT 16H偏移地址
        INT16SEG         DW    0       ;INT 16H段地址
        KEYCOUT          DB    0       ;内码计数器
        ALTF10           DB    0       ;Alt+F10标识单元
        COUNT_1          DB    0       ;外码计数器
        INTCODE          DW    0       ;内码缓冲区
           :               :
           :               :           ;[有关数据单元]
```

续表 2-1

```
INT16_INT:  CLD                         ;中断 16H始址
            STI
            PUSH    ES                  ;保存寄存器原值
            PUSH    DS
            PUSH    BX
            PUSH    SI
            PUSH    DI
            PUSH    CX
            PUSH    BP
            PUSH    DX
            PUSH    CS
            POP     DS
            OR      AH, AH              ;AH=0 等待输入
            JZ      INT16_S1
            CMP     AH, 01              ;AH=1 测试键盘状态
            JZ      INT16_T5
INT16_T5:   JMP     INT16_S2            ;转原CC-BIOS 处理
            JMP     TEST_KEY            ;测状态或取第二码
INT16_S1:   CMP     KEYCOUT,0           ;内码计数器=0?
            JZ      HAVE_W              ;是，转移
            MOV     BX, 1
            XOR     AH, AH              ;AH=0, AL=取第
                                        二个内码
            MOV     AL, BYTE PTR [BX+INTCODE]
            DEC     KEYCOUT             ;内码计数器清零
            JMP     INT16_RET           ;返回
HAVE_W:
            CMP     ALTE10, 0           ;是新编码方式吗?
            JNZ     INT16_T1            ;是，转移
            MOV     BX, 40H             ;否
            MOV     ES, BX              ;ES=BIOS 数据段
            MOV     SI, CRTMODE         ;测屏幕模式
            CMP     BYTE PTR ES:[SI], 6
            JNZ     INT16_S2            ;非图形模式转 CC-
```

续表 2-1

```
                                          BIOS
        CALL    INT16_CED       ;调 CC-BIOS 取输
                                  入码
        CMP     AX, 7100H       ;是 Alt+F10?
        JNZ     INT16_T4        ;否，转移
        MOV     ALTF10, 0FFH    ;是，置新编码标志
        CALL    DISPTS          ;提示行显示
        XOR     AL, AL
        JMP     INT16_RET
INT16_T1:
        XOR     AX, AX
        CALL    INT16_S3        ;取输入码
        OR      AL, AL          ;是标准键盘吗?
        JNZ     ASCII_N         ;是，转外码处理
        CMP     COUNT_1, 0      ;外码计数器=0?
        JZ      INT16_T4
        MOV     DL, 7           ;非法键报警
        CALL    DISPTTY
        JMP     HAVE_W
INT16_T4:
        CMP     AH, 68H         ;是原 CC-BIOS 功
                                  能键吗?
        JB      OTHER_R         ;否，转出
        CMP     AH, 6EH
        JG      OTHER_R
        JMP     INT16_T3        ;转键盘指针反向调
                                  整
OTHER_R:
        JMP     INT16_RET       ;其他键均返回
ASCII_N:
        JMP     ASCII_IN        ;转输入码处理
INT16_RET:
        STI
        POP     DX              ;恢复各寄存器原值
        POP     BP
        POP     CX
        POP     DI
        POP     SI
```

续表 2-1

```
                POP     BX
                POP     DS
                POP     ES
                IRET                        ;中断返回
INT16_T3:
                CALL    ADJKEY              ;键盘缓冲指针调整
                MOV     CS:COUNT_1, 0       ;外码计数器置零
                MOV     CS:ALTF10, 0        ;清消 Alt+F10 标
                                            志
                XOR     AX, AX              ;AH=0
INT16_S2:       STI                         ;转原 CC-BIOS 处
                                            理模块
                POP     DX
                POP     BP
                POP     CX
                POP     DI
                POP     SI
                POP     BX
                POP     DS
                POP     ES
                JMP     DWORD PTR CS:INT160FF
INT16_CED:
                PUSHF                       ;调原 CC-BIOS 功
                                            能
                CALL    DWORD PTR CS:INT160FF
                RET
INT16_S3:                                   ;INT16H, AH=0
                                            功能模块
                PUSH    DS
                MOV     AX, 040H
                MOV     DS, AX              ;置数据段
IN_LOOP:        CLI
                MOV     BX, 01AH            ;BX=键盘缓冲区
                                            首指针
                MOV     SI, [BX]            ;取出一个字
```

续表 2-1

```
                CMP     SI, [BX+2]      ;等于键盘缓冲区尾指针吗?
                LODSW                   ;取一对输入代码->AX
                STI
                JZ      IN_LOOP         ;若相等，循环等待
                CLI
                MOV     SI, [BX]
                ADD     SI, 2           ;否则，指针加2
                MOV     DI,082H
                CMP     SI,DS:[DI]
                JNZ     RT_LOOP
                MOV     SI,DS:[DI-2]
RT_LOOP         MOV     [BX], SI        ;保存调整后的指针
                STI
                POP     DS
                RET
ADJKEY:
                PUSH    ES              ;整盘指针反向调整
                MOV     BX, 040H
                MOV     ES, BX          ;ES=BIOS整数据段
                SUB     WORD PTR ES:[01AH], 2
                CMP     WORD PTR ES:[01AH],1CH
                JNZ     KEYRET
                MOV     WORD PTR ES:[01AH],003CH
KEYRET:         POP     ES
                RET
TEST_KEY:               ;-------AH=01〈测试键盘状态〉
                CMP     KEYCOUT,0       ;内码计数器=0?
                JZ      INT16_S2        ;是，转原CC-BIOS
```

续表 2-1

```
                MOV     BL, KEYCOUT
                XOR     BH. BH
                XOR     AH, AH          ;AH=0, AL=
                                        第二个内码
                MOV     AL,BYTE PTR[BX+INTCODE]
                CMP     BX, 0           ;内码计数=0?
                POP     DX
                POP     BP              ;还原栈数据
                POP     CX
                POP     DI
                POP     SI
                POP     BX
                POP     DS
                POP     ES
TEST_KPT        PROC    FAR
                RET     0002            ;放弃标志返回
TEST_KPT        ENDP
CHINA_1:                                ;返回汉字高位
                                        内码
                MOV     COUNT_1, 0      ;计数器置0
                MOV     KEYCOUT,2       ;内码计数 器置
                                        2
                XOR     AH, AH          ;AH=0, AL=
                                        内码
                MOV     AL, BYTE PTR [INTCODE]
                DEC     KEYCOUT         ;内码计数器－1
                JMP     INT16_RET       ;程序返回
ASCII_IN:       :       :               ;[外码-内码变
                                        换程序]
                :       :               ;[和各 级 码 表
                                        检索程序]
                :       :
START           PROC NEAR               ;程序装载 模块
                                        ‹重 置 INT16›
                PUSH    CS
```

续表 2-1

```
        POP     DS              ;DS=CS
        MOV     AX, 3516H
        INT     21H             ;取原中断16H
                                 地址向量
        MOV     INT160FF, BX    ;保存其偏移量
        MOV     INT16SEG, ES    ;保存其段地址
        LEA     DX, INT16_INT   ;指向本程序首
                                 址
        MOV     AX, 2516H
        INT     21H             ;重新设置INT
                                 16H中断
        LEA     DX, START       ;指向该程序结
                                 束指针
        ADD     DX, 100H
        INT     27H             ;装载并驻留在
                                 内存中
START   ENDP                    ;程序结束
CODE    ENDS
        END     START
```

2.3.6 打印系统的功能

通常的微机系统能够处理三个并行设备(LPT1—3)，串行打印机与并行打印机的控制方式是相同的，只是在向打印机发送数据时，由于接口不同，其数据传送的方式也不同而已。当每个并行设备要与CPU总线相联时，必须有一个相应的适配器。对适配器的操作主是要通过对其中的三个I/O寄存器的读写实现的。每个I/O端口寄存器都有自己的口地址。在ROM-BIOS数据区中分配了两个字节存放适配器的基地址。基地址，也就是指这三个寄存器中的最低地址。存放在0040:0008—0040:0009存贮单元中的是设备LPT1的口地址，LPT2的基地址存放在0040:000A—0040:000B存贮单元中，依次类推。一个适配器分配哪个并行口是可以选择的。如表2-2所示。因此，编程时要访问某个并行

口，首先要检查一下该适配器所占用的端口地址。当并行口上安装有适配器时，系统将该基地址初始化为零。

各类适配器端口地址表 **表 2-2**

适配器类型	数据寄存器	状态寄存器	控制寄存器
单显卡(XT/AT)	3BCH	3BDH	3BEH
彩显卡(PC-XT)	378H	379H	37AH
AT串/并卡(LPT1)	378H	379H	37AH
AT串/并卡)(LPT2)	278H	279H	27AH

上表列出了各适配器相应的寄存器端口地址，其长度均为一个字节。要打印的数据必须发送给数据寄存器。状态寄存器报告有关打印机当前的各种信息，CPU可以连续地监视其状态，以便正确地发送数据。状态寄存器还将返回打印机的出错情况，以便告知调用程序作出相应的处理。控制寄存器的功能包括适配器的初始化和控制数据的输出，它可以用以设定该并行口为中断操作方式，即当该口就绪时向CPU发出中断请求信号。这种方式可以使CPU在发送一个数据后转去处理其他事情，而不是单纯地等待。下面给出各端口寄存器各位的意义：

数据寄存器

位$_0$= 0 正常设置，= 1 数据输出选通

位$_1$= 0 正常设置，= 1 自动回车换行

位$_2$= 0 初始化打印口，= 1 正常设置

位$_3$= 0 使打印机脱机，= 1 正常设置

位$_4$= 0 禁止打印机中断，= 1 允许打印机中断

位$_5$—位$_7$ 不用

状态寄放器

位$_0$= 0 正常，= 1 超时

位$_1$—位$_2$ 不用

位$_3$= 0 打印机出错，= 1 打印机正常

位$_4$= 0 打印机脱机，= 1 打印机联机

位$_5$= 0 打印机有纸，= 1 打印机缺纸

位$_6$= 0 打印机接收到确认信号，= 1 正常

位$_7$= 0 打印机忙，= 1 打印机不忙

一般，行式打印机在接收到回车、换行字节或者接收到的字符和数据等于打印机一行允许的最大列数之前，是不发生打印操作的。其数据信息暂时存放在打印机的数据缓冲区内。直到收到换行或满行信息时，才把一行的数据全部打印输出。

CC-BIOS 汉字系统对于打印模块进行了扩充。也就是说，当进入汉字系统之后，还要执行一个汉字打印驱动程序，使得点阵式打印机或者其他类型的打印机，能够打印输出汉字信息来。

汉字打印驱动程序是汉字系统中的一个独立的模块，它的主要功能是利用系统建立在内存、磁盘或汉卡等存贮介质上的汉字点阵字模库来驱动各种针式打印机，打印出各种字型和各种大小的汉字信息。汉字打印程序仅修改了 INT 17H 中断向量地址，但保存了 BIOS 中原西文打印模块的入口地址。因为大多数汉字驱动程序仅扩充了中断 17H 的 0 号功能，其他功能仍然是调用原西文打印模块。打印驱动程序的主控流程如图 2-11 所示。

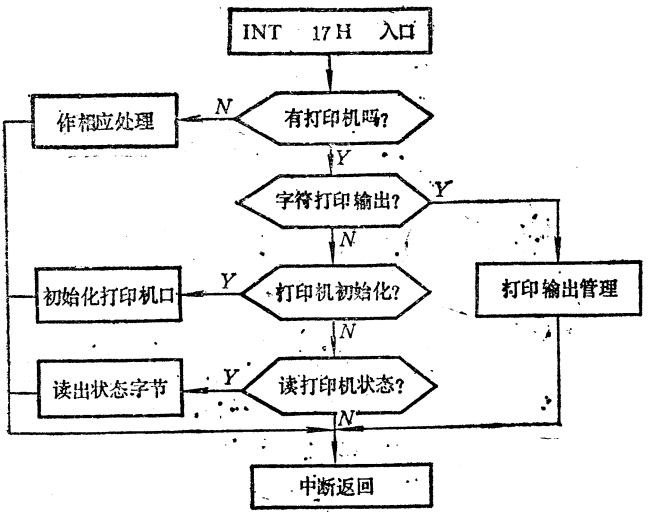


图 2-11 打印模块主控程序

2.3.7 打印模块的功能调用

在高级语言中，向打印机发送数据是轻而易举的。但是要直接向打印机端口发送数据，则要复杂得多，虽然，这样作可以更灵活，功能更强，但其设备的兼容性比较差，因此，可以根据系统ROM-BIOS或CC-BIOS提供的打印功能调用，用汇编语言编制程序要方便得多，即指定参数后发INT 17H指令。

1. 向打印机输出一个字节

输入参数：[AH]=00　　；功能号
　　[AL]=要打印的字节
　　[DX]=打印机号(0—2)

输出参数：[AH]=打印机当前状态

通常，向打印机输出信息都使用该功能，AL寄存器内是要打印的ASCII代码或汉字机内码。DX是当前所设定的打印机号。系统默认是LPT1=0号，改变这个基地址号，则可以实现选择其他打印机的目的。

例如：向打印机送一个ASCII码：

```
        MOV  AH, 0          ; 功能号
        MOV  AL, 'A'        ; 送一 "A"
        XOR  DX, DX         ; LPT1 口
        INT  17H            ; 调 BIOS 打印输出
        TEST AH, 1          ; 超时吗?
        JNZ  ERROR          ; 是，转出错处理
          ⋮
ERROR
          ⋮
```

2. 初始化打印机，并返回状态

输入参数：[AH]=01　　，功能号
　　[DX]=打印机号

输出参数：[AH]=打印机当前状态

位$_7$ = 打印机忙
位$_6$ = 确认
位$_5$ = 缺纸
位$_4$ = 选择打印机
位$_3$ = I/O 出错
位$_2$，位$_1$ 不用
位$_0$ = 超时

3. 返回指定的并行打印机当前状态

输入参数：[AH] = 02　　；功能号
[DX] = 打印机号

输出参数：[AH] = 打印机当前状态
位$_7$ = 打印机忙
位$_6$ = 确认
位$_5$ = 缺纸
位$_4$ = 选择打印机
位$_3$ = I/O 出错
位$_2$，位$_1$ 不用
位$_0$ = 超时

当使用这些功能调用时，要测试其返回的 AH 值，取出程序需要判测的那一位，以作相应的处理。在汉字驱动程序中，由于一个汉字是由 2 个扩展 ASCII 代码作为汉字机内码的，因此，必须向打印机送两个扩展 ASCII 码以后，驱动程序将根据得到的汉字机内码，转换为该汉字在内存、磁盘和汉卡等处的地址码，取出相应的汉字点阵字模，把打印机设置为图形打印方式，由字型、字体和方式标志再向打印机送出相应的汉字字形信息。其具体的编程技术将在汉字打印驱动程序一章中作详细的描述。

2.4 汇编语言指令系统

2.4.1 寄存器和标志位

8088/8086处理器中可以作为操作数参加运算的寄存器有：通用寄存器、索引寄存器、段寄存器和指针寄存器。标志寄存器不能作为独立的操作数使用，但大多数指令执行后会改变标志寄存器中的某些标志位。各寄存器描述如下：

AH	1字节	累加器的高字节
AL	1字节	累加器的低字节
BH	1字节	基地址寄存器的高字节
BL	1字节	基地址寄存器的低字节
CH	1字节	计数寄存器的高字节
CL	1字节	计数寄存器的低字节
DH	1字节	数据寄存器的高字节
DL	1字节	数据寄存器的低字节
AX	2字节	累加器（整字）
BX	2字节	基址寄存器（整字）
CX	2字节	计数寄存器（整字）
DX	2字节	数据寄存器（整字）
BP	2字节	基址指针
SP	2字节	栈指针
SI	2字节	源变址寄存器
DI	2字节	目的变址寄存器
CS	2字节	代码段寄存器
SS	2字节	栈段寄存器
DS	2字节	数据段寄存器
ES	2字节	附加段寄存器

8088/8086有一个16位的标志寄存器，其各标志位的功能与描述如图2-12所示。

1. 进位标志 C（Carry Flag）

在算术运算的操作中，如果结果的最高位（字节的位7或字的位15）产生一个进位或错位时，则 $C=1$，否则为0。这个标志主要用于多字节数的加、减法运算，移位和循环指令也能把存

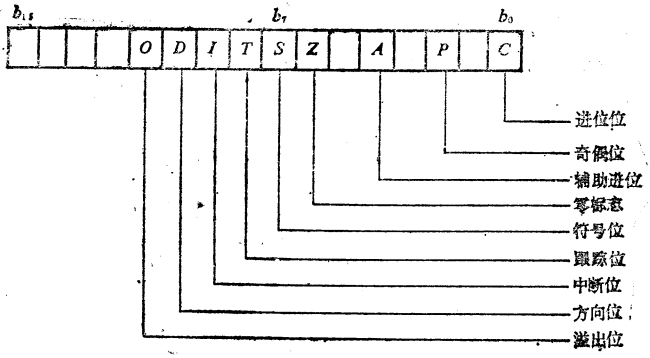


图 2-12 标志寄存器各位

贮器或寄存器中的最高位（左移）或最低位（右移）放入标志位 C 中。

2. 奇偶标志 P（Parity Flag）

当任一数据操作结果的低8位中“1”的个数为偶数时，则 $P=1$，否则 $P=0$。这个标志可用于检查在数据传送过程中是否发生错误。

3. 辅助进位标志 A（Auxitiary Carry Flag）

在字节操作时，则由低半字节（一个字节的低4位）向高半字节有进位或错位时，在字操作时，低位字节向高位字节有进位或错位，则 $A=1$，否则为0。这个标志用于十进制算术运算指令中。

4. 零标志 Z（Zero Flag）

当各类运算的结果为0时，$Z=1$，否则 $Z=0$。

5. 符号标志 S（Sign Flag）

该标志的值与运算结果的最高位相同，即结果的最高位为1，则 $S=1$，否则 $S=0$。

6. 追踪标志 T（Trap Flag）

Trap状态是一种特殊的调试手段。若置 T 标志，使处理器进入单步方式，以便于调试程序，在 T 置位方式下，CPU在每

条指令执行以后，产生一个内部的中断，允许程序在每条指令执行以后进行观察。

7. 中断允许标志 I（Interrupt-enable Flag）

若指令中置 $I=1$，则允许 CPU 去接收外部的可屏蔽的中断请求；若使 $I=0$，则屏蔽上述的中断请求，对 CPU 内部产生的中断不起作用。

8. 方向标志 D（Direction Flag）

D 标志的状态决定在字符串操作时，变址寄存器中的内容是自动增加还是自动减少。若 $D=1$，SI 和 DI 寄存器的内容就自动递减，就是说字符串将从存贮器高地址开始向存贮器低地址进行存取。若 $D=0$，SI 和 DI 寄存器的内容就自动递增，字符串将从低地址向高地址存取。

9. 溢出标志 O（Overflow Flag）

在算术运算中，带符号数的运算结果超出了 8 位或 16 位的带符号数所能表达的范围，将由此标志指明。在字节运算时，其结果＞+127，或结果＜-128，16 位字运算时，其结果＞+32767，或结果＜-32768，此标志置位。反之，则标志复位。一个任选的溢出中断指令，在溢出情况下能够产生中断。

2.4.2 8088/8086 指令系统

本书所列出的指令，仅说明了它们的用途，在汉化软件的过程中作为参考之用。有关指令的详细介绍请参考一些汇编语言的书籍，在此不再赘述。

AAA　加法 ASCII 调整

该指令用来对两个非压缩型十进制数相加的结果(在 AL 中)进行调整，以获得一个非压缩型的十进制和。

AAD　ASCII 除法调整

在两个非压缩型十进制数相除之前，用该指令对 AL 中的被除数进行调整，以获得一个非压缩型的十进制商。

AAM　ASCII 乘法调整

对两个非压缩型十进制数相乘的结果（在AX中）进行调整，以获得非压缩型的十进制积。

AAS　ASCII减法调整

对两个非压缩型十进制数相减的结果（在AL中）进行调整，以获得非压缩型十进制数差。

ADC　目的数，源数　带进位位的加法指令

该指令实现源操作数和目的操作数的相加，还要加上进位位CF的值，最后把结果送回目的操作数中去。

ADD　目的数，源数　加法指令

该指令使源操作数与目的操作数相加，并将结果送回目的操作数中去。

AND　目的数，源数　逻辑“与”

使源操作数和目的操作数进行按位相“与”，两操作数对应位均为1，结果位则为1，否则结果位为0，结果送回目的操作数。常用来屏蔽某位之用。

CALL　目标　调一个子过程

该指令的功能是调用一个子程序，分为段内调用和段间调用。

CBW　把字节转换为字

该指令对AL寄存器中有符号数进行符号扩展，使其符号填满AH寄存器中的各位，扩展成为AX满字。

CLC　清进位标志

该指令用来把进位标志CF置0。

CLD　清方向标志

该指令用来将方向标志位DF清0，使串操作中的源串和目的串指针（SI、DI）为自动增量。

CLI　清中断标志

清除中断允许标志IF，从而禁止了出现在处理器INTR线上的“可屏蔽的外部中断”。在处理器上的NMI线上“不可屏蔽中断”不能禁止。

CMC　进位标志取反

该指令使得进位标志 CF 取反，若 CF = 1，则 CF←0，反之亦然。

CMP　目的数，源数　比较两个操作数

该指令执行源操作数与目的操作数减法运算，但所得的结果不回送给目的数，仅仅影响标志位。

CMPS　目的串，源串　串与串比较

该指令从由SI寻址的源串元素(字节/字)减去由DI寻址的目的串元素（字节/字），以此方法进行比较，结果不回送给目的串，只影响标志位。

CWD　把字转换为双字

该指令用来将 AX 寄存器中的内容的符号位扩展到 DX 中去，常用于除法指令之前。

DAA　十进制加法调整

用来对两个压缩型十进制数相加的结果（在 AL 中）进行调整，以产生一个压缩型的十进制和。

DAS　十进制减法调整

用来对两个压缩型十进制数相减的结果（在 AL 中）进行调整，以获得压缩型的十进制差。

DEC　目的数　目的操作数减 1

该指令将目的操作数减 1，并将结果回送给目的操作数。

DIV　源数　不带符号除法

该指令完成累加器 AL/AX（16 位扩展 AH/DX）中的无符号二进制数被源操作数除的操作，商在 AL/AX 中，余数在 AH/DX 中。

ESC　外部码，源数　换码指令

该指令使其他的处理器可以从 8088/8086 CPU 指令流中接收它们的指令。

HLT　停机指令

该指令使 8088/8086 CPU 进入暂停状态，这个状态可被一

个允许的“外部中断”或“复位”清除掉。

IDIV　源数　带符号的整数除

该指令完成累加器 AL/AX（16 位扩展 AH/DX）中的有符号二进制整数被源操作数除的操作。

IMUL　源数　带符号的整数乘

该指令执行源操作数与累加器中有符号数的乘法，8 位运算乘积送 AX 中，16 位运算乘积送 DX 和 AX 中。

IN　累加器，通道　输入字节或字

该指令从输入端口传送一个字节或字到 AL 或 AX 累加器中，外设通道端口可由立即数或 DX 寄存器间接寻址两种方式来指定。

INC　目的数　目的操作数加 1

该指令将目的操作数加 1，并将结果回送到目的操作数中去。

INT　类型号　中断指令

该指令用来启动由“中断类型”操作数指定的中断服务程序，将程序控制转送给中断向量表中指定的一个中断入口地址。

INTO　溢出中断指令

若溢出标志 OF 置位，则该指令将产生一个中断，去启动一个溢出中断处理程序，其类型号为 4。

IRET　中断返回指令

该指令使控制返回到由前面中断操作所保存的返回地址，并恢复保存在堆栈中的标志寄存器的初值。

JA/JNBE　短标号　高于/不低于/不等于转移

该指令用于比较两个无符号数之间的关系后，当“高于”或“不低于/不等于”时，将控制转给标号指定的地址处，寻址范围为±127 字节。

JAE/JNB　短标号　高于或等于/不低于转移

该指令用于比较两个无符号数之间的关系后，当“高于/等于”或“不低于”时，将控制转给标号指定的地址处，寻址范围为±127 字节。

JB/JNAE/JC　短标号　低于或不高于或有进位转移

当比较两个无符号数关系后，其关系为“不高于/不等于”或“低于”或进位标志为1时，将控制转给标号指定的地址处，寻址范围为±127字节。

JBE/JNA　短标号　低于/等于或不高于转移

当比较两个无符号数后，其关系为“低于/等于”或“不高于”时，将控制转给标号所指定的地址处，寻址范围为±127字节。

JCXZ　短标号　当CX为零时转移

若CX寄存器的内容为0，JCXZ指令将把控制转给标号所指定的地址处，寻址范围为±127字节。

JE/JZ　短标号　等于/为零时转移

在该指令之前最后一条两数比较指令(或两数相加减指令)的结果为0时，将控制转给标号指定的地址处，寻址范围为±127字节。

JG/JNLE　短标号　大于或不小于/不等于转移

对两个有符号数进行比较时，当“大于”或“不少于/不等于”时，该指令将控制转给标号指定的地址处，寻址范围为±127字节。

JL/JNGE　短标号　小于或不大于/不等于转移

对两个有符号数进行比较时，当“小于”或“不大于/不等于”时，该指令将控制转给短标号指定的地址处，寻址范围为±127字节

JLE/JNG　短标号　小于/等于或不大于转移

对两个有符号数进行比较时，当“小于/等于”或“不大于”时，该指令将控制转给标号指定的地址处，寻址范围为±127字节。

JMP　目标　无条件转移

该指令无条件地将控制转给目标操作数。

JNC　短标号　进位标志为零时转移

若进位标志 CF 等于 0，则把短标号的相对位移量加入到指令指示器 IP 中，产生转移操作。

JNE/JNZ　短标号　不等于或不为零转移

在该指令之前最后一条指令相比较时，其结果不等于 0，则将控制转移到标号指定的地址处。反之，若 ZF = 1，则不转移而顺序执行下一条指令，寻址范围为±127 字节。

JNO　短标号　当不溢出时转移

若溢出标志 OF = 1，不发生转移。当溢出标志 OF = 0 时，则把控制转移到标号指定的地址处，寻址范围为±127 字节。

JNP/JPO　短标号　无奇偶性/奇偶性为奇转移

若奇偶标志为 1，即影响 PF 的最后运算将奇偶标志 PF 置位偶性时，不产生转移，若 PF = 0，则转移到标号处，寻址范围为 ±127 字节。

JNS　短标号　无符号或正数时转移

当无符号时，即标志 SF = 0，该指令把控制转移到标号指定的地址处。若 SF = 1 时，不发生转移，寻址范围为 ±127 字节。

JO　短标号　当溢出时转移

当溢出标志 OF = 1 时，该指令把控制转移到标号指定的地址处，寻址范围为 ±127 字节。

JP/JPE　短标号　有奇偶性/奇偶性为偶转移

当奇偶标志 PF = 1 时，该指令将控制转移到标号指定的地址处，寻址范围为 ±127 字节。

JS　短标号　负数时/符号标志为 1 时转移

若符号标志 PF = 1 时，该指令将控制转移到标号指定的地址处。若 SF = 0 时，不发生转移，寻址范围为±127 字节。

LAHF　将标志装入 AH 寄存器

该指令将标志寄存器中的 SF、ZF、AF、PF 和 CF 位传送给 AH 寄存器中的指定位（7、6、4、2、0）中。

LDS　目的数，源数　将指示器装入 DS

该指令用来将“指示器——目标”传送到一对目标寄存器

中，其段地址传送到 DS 中，偏移地址传送到任何一个十六位寄存器中。它指定了要传送的“指示器——目标”放在存贮器中的何处。

LEA　目的数，源数　装入有效地址

该指令将源操作数的偏移地址传送到目的数指定的 AX、BX CX、DX、BP、SP、SI 和 DI 任何一个寄存器中去。

LES　目的数、源数　将指示器装入 ES

该指令将源操作数“指示器——目标”传送到一对目标寄存器中，其段地址传送到 ES 中，偏移地址传送到任何一个十六位寄存器、指示器中。

LOCK　总线锁定

该指令使得 8088/8086 最大系统在有 LOCK 前缀指令执行期间保持“总线锁定”信号 LOCK 有效。

LODS/LODSB/LODSW　源串　串传送指令

该指令将由 SI 指定的串元素（字节/字）传送到 AL/AX 寄存器中去，并自动调整 SI 寄存器的指针。

LOOP　短标号　循环控制指令

CX 预置循环的次数，执行一次 LOOP，则将计数寄存器 CX 内容自动减 1，若 CX 不等于 0，控制再转移到标号指定的地址处，直到 CX = 0 为止。寻址范围为 ±127 字节。

LOOPNE/LOOPNZ　短标号　不相等/非零循环

该指令将 CX 计数寄存器内容减 1，若 CX 不等于 0 或 ZF 标志置 0，则将控制转移到标号指定的地址处。否则顺序执行下一条指令，寻址范围为 ±127 字节。

LOOPE/LOOPZ　短标号　相等/等于零循环

该指令将 CX 计数寄存器减 1，若 CX 新值等于 0 或 ZF 标志置 1，则转至标号处执行。否则停止循环，寻址范围为 ±127 字节。

MOV　目的数，源数　传送指令

该指令将源操作数传送到目的操作数中去，共有七种类型的

传送方式。

MOVS/MOVSB/MOVSW　串传送指令

该指令将由SI寻址的源串中的字节或字传送到由ES:DI寻址的目的串中去，并修正SI和DI寄存器的指针。由于该指令可用前缀方式重复操作，常用于数据块在存贮器内的传送。

MUL　源数　不带符号乘法

该指令将AL或AX累加器中的无符号数与源操作数进行乘法运算，把双倍字长的积送回AH和AL或DX和AX中，对于8位运算，乘积在AH和AL中，对于16位的运算，其乘积在DX和AX中。

NEG　目的数　求补码/求反

该指令执行从0中减去目的操作数的运算，并且加1后再将结果回送到操作数中，指定的操作数则变成二进制补码。

NOP　空操作

该指令使CPU执行空操作，无任何结果，但每条NOP指令则需要用3个时钟。

NOT　目的数　逻辑“非”

该指令建立目的操作数的反码，并将结果送回目的操作数。

OR　目的数，源数　逻辑“或”

该指令用来执行两个操作数的按位逻辑“或”，并将结果送回目的操作数中。

OUT　通道，累加器　输出字节或字

该指令将AL或AX累加器输出一个字节或一个字到外部通道端口去，通道由一个字节的立即数定义，允许范围为0—255，由DX寄存器间接寻址，允许范围为64KB。

POP　目的数　从堆栈弹出字

该指令将SP指向的栈单元中的一个字传送到目的操作数中，然后将SP作加工修正。通常，使用POP指令将中间变量以及现场数据送到寄存器或存贮器中去。

POPF　从堆栈中弹出标志

该指令将SP指向的栈单元中的内容（16位）传送到标志寄存器中去，然后SP寄存器作加2修正。

PUSH　源数　把字推入堆栈

该指令使堆栈指针SP减2，然后把源操作数中的字送到SP当前指向的堆栈单元中去。

PUSHF　把标志推入堆栈

该指令将标志寄存器内容保存入栈。通常，在子程序中用PUSHF和POPF指令来保存和恢复标志寄存器内容。

RCL　目的数，计数　通过进位循环左移

该指令将目的操作数连同进位标志CF一起循环左移若干位（由计数确定），可用CL存放一个数来指定左移的次数。

RCR　目的数，计数　通过进位循环右移

该指令将目的操作数连同进位标志CF一起循环右移若干位（由计数确定），可用CL存放一个数来指定右移的次数。

REP/REPE/REPZ/REPNZ/REPNE　重复串操作前缀

这些前缀指令放在基本串操作指令之前，它们能使后面的基本串操作指令重复执行。每执行一次，CX计数器内容减1，根据使用的重复前缀的不同，来确定程序执行的性质。

RET　从过程返回

RET指令将控制转给由前面CALL指令保护进栈的返回地址处的指令。

ROL　目的数，计数　循环左移

该指令把目的操作数循环左移若干位（由计数确定），目的操作数最高位移入标志位CF，而进位位CF的原始值被取代，最低位由新的CF值填入，直到计数值为0。

ROR　目的数，计数　循环右移

该指令把目的操作数循环右移若干位（由计数确定），目的操作数最低位移入标志位CF中，而CF原始值被取代，最高位

由 CF 值填入，直到计数值为 0 。

SAHF　AH 存入标志

该指令将 AH 寄存器中指定的位送入标志寄存器的 SF、ZF、AF、PF 和 CF 标志位中。

SAL/SHL　目的数，计数　算术左移/逻辑左移

将指定的目的操作数根据计数值左移若干位，目的操作数的最高位移入进位位 CF 中去，取代原来 CF 值，而最低位由 0 填入。

SAR　目的数，计数　算术右移

将目的操作数根据计数值向右移若干位，每次右移一位后，往目的操作数左端补入原最高位(符号)的值以保持最初符号值。

SBB　目的数，源数　带借位的减法

从目的操作数中减去源操作数，若在相减时使进位标志 CF 置 1，则将结果减 1 后送回目的操作数。

SCAS/SCASB/SCASW　串扫描

由 DI 寄存器寻址的存贮器单元的内容和 AL (8 位)或 AX (16 位) 寄存器的内容，利用减法进行比较，并由结果来设置标志位，存贮器单元和 AX 内容均不受影响。

SHR　目的数，计数　逻辑右移

将指定的目的操作数根据计数值向右移若干位，每次右移一位后，目的操作数的左端 (高位) 补入 0 。

STC　进位标志置位

该指令将进位标志 CF 置 1 。

STD　方向标志置位

该指令用来将方向标志 DF 置 1，使串操作指令中“源串、目的串”的指针 (SI、DI) 为自动递减。

STI　中断允许

该指令使中断标志 IF 置 1，允许 CPU 响应来自可屏蔽中断线上发来的中断请求。

STOS/STOSB/STOSW　存入串指令

该指令将 AL 或 AX 中的一个字节或一个字传送到 ES:DI 指定的存贮单元中去，并根据方向标志 DF 自动修正 DI 指针的地址。

SUB　目的数，源数　减法指令

该指令执行目的操作数减去源操作数的运算，并将其结果送回到目的操作数中。

TEST　目的数，源数　测试指令

该指令将两个操作数按位进行逻辑“与”，根据“与”的结果去影响标志，而对操作数不发生影响，并且，检测标志、清进位标志和溢出标志。

WAIT　等待指令

如果引脚上的TEST信号没有维持低电平，该指令使得CPU进入空转状态，也就是该处理器进入一个等待状态。

XCHG　目的数，源数　交换指令

该指令可将源操作数和目的操作数中的字节或字进行交换，即把目的操作数内容放入源操作数中，把源操作数中的内容放入目的操作数中。但该指令不能用段寄存器作为操作数。

XLAT　源表　查表字节传送

该指令用来实现一个查表找字节的换码操作，AL 寄存器用来作为进入一个基地址在 BX 中的表索引，即表的基地址在 BX 中，表中换码字节所在的偏移地址在 AL 中。

XOR　目的数，源数　逻辑“异或”

对两个操作数进行按位“异或”，若对应的位均为 0 或均为 1，则结果位为 0，若对应位不相等，则结果位为 1，结果送回到目的操作数。

2.4.3 80186/80286 新指令

PUSH　立即数

这条指令使立即数据能被推入处理器堆栈，立即数据可以是立即字节或立即字。若该数据是一个字节，则它在被推入堆栈

以前，将被符号扩展。

PUSHA　把所有寄存器推入堆栈

该指令能使CPU所有通用寄存器保护到堆栈中，推入堆栈的SP的值是第一个入栈寄存器以前的值，PUSHA指令并不保护任何段寄存器（CS、DS、SS、ES）、指令指示器IP、标志寄存器或任何集成的外围寄存器。

POPA　从堆栈中弹出所有寄存器

该指令弹出PUSHA压入栈中的所有通用寄存器值，弹出的SP寄存器的值将被忽略。

IMUL　立即数　带符号整数乘立即数

这条指令使一个值能乘以一个立即数，这种操作的结果是16位的。立即数可以是一个字节或一个字，但若是一个字节，则要进行符号扩展。这种乘法的结果，可以放在任何一个通用或指示寄存器中。

INS　输入字节或字串

该指令执行从一个I/O端口到存贮器的块输入。I/O地址由DX寄存器指定，而存贮器单元则由DI寄存器指示。在操作执行后，DI寄存器被调整1（字节）或2（字）。当指令之前有REP前缀时，该指令能把数据块从一个I/O地址传送到一个存贮器块。但这种操作不修改DX寄存器中的I/O地址值。

OUTS　输出字节或字串

该指令执行从存贮器到一个输出端口的块输出。I/O地址由DX寄存器指定，存贮器单元由SI寄存器指定，在执行操作以后，SI寄存器被调整1（字节）或2（字），由方向标志决定递增还是递减调整。当该指令之前有REP前缀时，这条指令能把在一个存贮器块中的数据块传送到一个I/O地址中去。同样，这种操作也不修改在DX寄存器中的I/O地址。

BOUND　检测超出范围值

CPU提供了一条BOUND指令，使数组的边界检查更加方便。使用这条指令时，由计算机得到进入一个数组的索引，被放

在CPU的一个通用寄存器中，而该数组的索引的上界和下界，则放在存贮器相连的两个字单元中。BOUND指令把寄存器的内容和存贮器单元中的内容进行比较，若寄存器中的值并不介于存贮器单元中的两个值之间，就会发生中断类型为5的中断，所执行的比较是Signed比较。等于上界或下界的寄存器值将不会引起一个中断。

ENTER/LEAVE　进入/离开过程

这两条指令用来建立/放弃高级、块结构语言的堆栈框架的指令。该指令不保护通用寄存器的内容。若它们必须被保护，则要用另外的指令来实现。此外，LEAVE指令也不执行从一个子程序中返回的功能，若需这种功能的话，则在此指令之后必须直接跟一条RET指令。

第三章　汉化软件的工具及其应用

要进行软件的二次开发，必须具备适当的工具。换句话说，就是需要一些行之有效的调试程序。近年来，在微机上开发了许多功能齐全的调试工具软件，有些是随机软件，有些是单独出售的。最常用的则是 Microsoft 软件公司的 Debug 程序，它是一个相当可靠的调试程序，具有程序跟踪、反汇编目标代码、搜索、转存和磁盘读写等功能，其具体用法将在下一节讨论。但是 Debug 有限的和低级的屏幕支持限制了某些程序员的使用。对于形式多样的调试程序，其功能均各有千秋，下面介绍一些具有特色的调试工具软件。这些软件都可以作为汉化软件的工具，但要根据系统的硬件环境、使用者的习惯和熟悉程度以及汉化工作的特殊要求进行选择使用。

3.1 工具软件的性能

3.1.1 动态调试程序

作为 Microsoft Macro Assemble 软件包的一部分，Microsoft 为用户提供了一个较灵活有效的调试工具：Symdeb 或 SDebug。这些软件可读入连接程序生成的 MAP 文件，显示高级语言程序及其相应的目标代码，分离出调试程序的显示屏幕与其跟踪程序的屏幕显示信息。Symdeb 与 Microsoft 所有编译程序均兼容，如 Assemble、Pascal、C 和 Fortran 一些高级语言等。相对 Debug 调试程序来说，它增加了许多新的命令和功能，但是 Symdeb 的命令却与 Debug 的命令向下兼容，其命令如表 3-1 所示。

该调试程序可以设置10个以上的断点，非常便于程序的动态

Symdeb 命令表　　表 3-1

```
A [<address>] - assemble
BC[<bp>] - clear breakpoint(s)
BD[<bp>] - disable breakpoint(s)
BE[<bp>] - enable breakpoint(s)
BL[<bp>] - list breakpoint(s)
BP [bp] <address> - set breakpoint
C <range> <address> - compare
D[type][<range>] - dump memory
E[type] <address> [<list>]-enter
F <range> <list> - fill
G [=<address> [<address>...]]-go
H <value> <value> - hexadd
I <value> - input from port
K [<value>] - stack trace
L [<addr> [<drive><rec><rec>]]-load

M <range> <address>-move
N <filename> [<filename>...]-name
O <value> <byte> - output to port
P [=<address>] [<value>]-program step
Q - quit
R [<reg>] [[=] <value>]-register
S <range> <list> - search
S {-|&|+} - source level debugging
T [=<address>] [<value>]-trace
U [<range>] - unassemble
V [<range>] - view source lines
W[<address> [<drive><rec><rec>]]-write
X [?] <symbol> - examine symbols(s)
XO<symbol> - open map/segment
Z<symbol> <value>

? <expr> - display expression
! [dos command] - shell escape
. - display current source line
\ - screen flip

> } <device/file> - Redirect output
< { <device/file> - Redirect input
= ~ <device/file> - Redirect both
* <string> - comment

<expr> ops: + - * / : not seg off by wo dw poi port wport mod and xor or
<type> : Byte, Word, Doubleword, Asciz, Shortreal, Longreal, Tenbytereal
```

跟踪。另外，对于内存中的数操作，可以字节、字、双字、短实数、长实数、ASCII 字符和十进制数的方式进行。例如，用双字类型的方式显示中断向量的数据，就非常直观，如下所示。

```
-DD 0000:0000
0000:0000  02C1:5186  0070:0C67  0DD7:2C1B  0070:0C67
0000:0010  0070:0C67  F000:FF54  F000:EB52  F000:EAA6
0000:0020  0070:022C  0DD7:2BAD  0070:0325  0070:039F
0000:0030  0070:0419  0070:0493  0070:050D  0070:0C67
0000:0040  C000:08F8  F000:F84D  F000:F841  0070:237D
0000:0050  F000:E739  F000:F859  F000:E82E  F000:EFD2
0000:0060  F000:E000  09A5:008E  F000:FE6E  0070:0C81
0000:0070  F000:FF53  F000:F0A4  0000:0522  C000:3AD0
```

从上面的表述中，可以直接读出中断向量的段地址：偏移地址。另外，在Symdeb的提示符下，可以方便地进行各种逻辑运算，运算的结果由十六进制、实数、十进制和ASCII 字符等表达式显示出来，例如下列的“异或”运算：

```
-? 35 XOR 11 ↵
0024H 00000024 (36) "$"
```

3.1.2 全屏幕调试程序

Phoenix Software 公司推出的 Pfix86-Plus 是一个全屏幕的调试软件。它可以在屏幕上滚动代码文本，标出作为临时断点的指令，而固定断点是在菜单中用一个专门的命令处理的。如表3-2所示 Pfix86 全屏幕菜单。

Pfix86 全屏幕菜单　　表 3-2

```
Breakpoint Disk Evaluate Format I/O Memory Program Quit Screen Tracing Window
Clear,Disable,Enable,Initialize,Set,Temporary
┌STACK──────────┐┌DATA1──────────────────────────────────────────┐
 30ED:0012  0000 │ 30DD:0000  CD 20 C0 9F 00 9A F0 FE   . . . . . .
 30ED:0010  0000 │ 30DD:0008  1D F0 5B 00 51 1D 51 00   . . . . . .
 30ED:000E  0000 │ 30DD:0010  51 1D EB 04 F1 0C 41 1D   . . . . . .
 30ED:000C  0000 ├FILE──────────────────────────────────────────┤
 30ED:000A  0000 │ C:\COSMAT.COM
 30ED:0008  0000 ├PROGRAM1─────────────────────────────────────┤
 30ED:0006  0000 │ 30ED:0100  E97201        JMP    0275
 30ED:0004  0000 │ 30ED:0103  1E            PUSH   DS
 30ED:0002  0000 │ 30ED:0104  53            PUSH   BX
 30ED:0000  0000 │ 30ED:0105  BB4000        MOV    BX,0040
 30ED:FFFE  0000 │ 30ED:0108  8EDB          MOV    DS,BX
┌CPU────────────┐│ 30ED:010A  BB4900        MOV    BX,0049
 AX=0000  BX=0000 │ 30ED:010D  803F04        CMP    BYTE PTR [BX],04
 CX=0288  DX=0000 │ 30ED:0110  7207          JB     0119
 SP=FFFE  BP=0000 │ 30ED:0112  803F07        CMP    BYTE PTR [BX],07
 SI=0000  DI=0000 │ 30ED:0115  7402          JE     0119
 CS=30ED  DS=30DD │ 30ED:0117  EB22          JMP    013B
 SS=30ED  ES=30DD │ 30ED:0119  56            PUSH   SI
 IP=0100  T:0     │ 30ED:011A  33DB          XOR    BX,BX
 A:0 C:0 P:0 S:0  ├BREAKPOINTS─────────────────────────────────┤
 Z:0 D:0 I:1 O:0  │ 00:
└─────────────────┘└──────────────────────────────────────────┘
```

Pfix86-Plus 在屏幕上分为几个窗口，每个窗口映射一个项目状态，如 CPU 各寄存器状态、堆栈的情况等等。每个窗口均可用 Window 命令改变其在屏幕上的尺寸，最大可以调整到整幅屏幕，数据在窗口内滚动。还可以通过功能键把光标驻留在指定的窗口内进行修改和调整。Pfix86 允许用户在表达式中使用22个操作符（算术、逻辑和位运算）来进行相当复杂的测试，以至确定某一个寄存器在何时达到某一值和某一个存贮单元的值是何时改变。该程序还允许用户规定一个在断点处执行的动作，也就是说允许在到达一个地址或某一条件断点时选择几种可能的动作。此时，用户可选择停止执行，调用一个子程序或者使下一个断点有效或无效等。

3.1.3 特殊功能调试软件

有些调试软件功能很强，但是价格昂贵，一般作为特殊功能使用。

Morgan Computing 公司的 Advanced Trace-86。该软件有许多特点，它可以将断点设置为当出现某一命令代码时，或者当寄存器具有某一特定内容时中止程序。并且还可以设置条件断点，跟踪被修改的内存内容，可以记录下20步以内的程序执行情况和加标记到反汇编出的程序中。Advanced Trace-86 的 UNDO 功能突破了向后追踪的技术，这个命令不仅保存机器先前的状态，它还能逐步向后退，即反向运行代码，一次一条指令，一共可以后退20条指令。由于有些类型的指令不能退回去（如中断或写磁盘等），因此在向后退时有一定的限制。但是在许多场合下，向后追踪的功能是十分有用的。

Visual Age 公司的 Code Smith86 调试软件。它特别有助于反汇编，给目标码加标记和注释，并将其结果写入磁盘。Code Smith86 可以使源代码插入屏幕上的指令串中，但不是真正嵌入程序当中，然后由屏幕写到磁盘上，以便进行重新汇编。新的指令放在存贮器中别的地方，并用 INT3 取代，在屏幕上可以看到“插入”代码的实际 CS:IP 地址。这种插入只是为了临时性的修改，对新代码的访问较慢，命令语法也比较繁琐复杂。

此外，还有 IBM 公司的 Resident Debug TooL，Digital Research 公司的 SID86，Answer Software 公司推出的 PPT-PC，Puttkammer 公司的 FDebug 和 Atron 公司推出的 PC-PROBE 等调试程序，都是性能优良，功能强大的调试工具。

3.1.4 反汇编器

通讯公司推出的 Sourcer 反汇编器受到了许多用户的青睐。该软件可以把各种类型的目标代码，反汇编为带符号的宏汇编 .ASM 源文件和 .LST 列表文件。这给程序分析和汉化工作提

供了方便的条件。Sourcer 的功能如表 3-3 中大写字母选择项所示。

反汇编器功能菜单　　表 3-3

```
                                                    V COMMUNICATIONS
            SOURCER                                 Copyright (c) 1988
                                                    V1.87  s/n B106636

F1 - help      File format, list(.lst)/source(.asm)      Analysis flags menu

 Output filename      Header title                 Xref (cross reference) OFF
                                                   Press return when done
  Temp.Asm                                                        Page 3
 5102:0234  32 D2          xor   dl,dl             ; Zero register
 5102:0236  CD 11          int   11h               ; Put equip bits in ax
 5102:0238  0E A1E4        mov   cx,data_6         ; (6033:A1E4=3C1h)

  Segment display    Word case style    Label type      Remarks - all
      on/off                            (decimal)
          Tabs inserted on/off (ON)                    Press "Q" to Quit

 Input filename  none          Label Counts     Drive c: used for output
 Beginning addr  0000:0000     Data      0       14,567,424 bytes available
 Ending address  0000:0000     Sub       0
 Math  off   uP  286 normal    Loc       0     Code fragment      Passes 2

 Select input file or address range, and any highlighted options desired.
```

该程序可以用 File 命令选择要进行反汇编的目标文件，又可以用 Begin/End 命令选择反汇编内存中指定地址的程序块。分析标志中设置了许多选择项，以助于汇编出的源文件的分析。Code 命令是提示用户选择何种类型的文件，它可以指定如下几种选择：

.EXE 指定的可执行文件

.COM 指定的命令文件

.OVL 多模块软件包的覆盖文件

.SYS 设备驱动程序

该程序在反汇编过种中，对指定的目标代码程序扫描 2—5 遍。边扫描边排序，把数据段、代码段、子程序以及过程标号有序地存写到文本文件中。形成了宏汇编源程序，这一宏汇编程序可以反复地汇编或反汇编。

除此之外，有些服务软件对汉化工作也有很大的帮助。虽然服务软件的主要功能是用于磁盘和文件的维护，但也可用于修改

和汉化程序。如 Peter Norton 公司的 Norton Utilities，Central Point Software 公司的 PCTOOLS 和该公司最近推出的 Ver5.0 版本——PCSHELL 等。这些文件对于目标文件的全屏幕编辑功能很强，又具有良好的用户界面，是用户常备的工具软件之一。

3.2 调试程序的运行状态

一个汇编语言的调试程序至少要提供两个主要功能：第一，它能一次运行用户程序的一条指令，执行每一步后停下来，使用户可以检查程序中每一条语句的运行结果。第二，调试程序还可以使用户程序连续运行下去，直到某一条特定的指令。这些功能称为单步跟踪和断点操作。

由于调试程序的需要，Intel 在设计 CPU 处理器时，加入了支持跟踪和断点的特殊性能，这使得编制一个简单的调试程序十分方便。

3.2.1 中断方式

为了使读者理解 CPU 结构具体是怎样支持调试程序的，先描述一下中断。由于 CPU 在程序运行时需要处理中断，但它的中断机制必须使得处理器能够保存中断时它正在做的工作，转向产生中断的更为重要的处理，然后再恢复它原来的工作。中断可以由硬设备产生，例如键盘和时钟，当然也可以由程序中的指令产生。

8088/80286 的中断可以有各种不同的原因。各硬件设备或中断指令都有它自己的中断类型，中断类型号从 0 编号到 255。这个编号将通知 CPU 引起了什么中断，并被用来计算中断处理程序的入口地址。对每一种中断类型，存贮器中有一个程序进行相应的处理，这个程序称为中断处理程序。

Intel 系列的微机都约定在存贮器地址0开始有一个地址表，

表中的每个地址占4个字节，每个地址的前两个字节是偏移量，后两个字节是段地址。这个地址表称为中断向量表。表中的第一个地址告知CPU，当发生类型为0的中断程序时，转向哪里进行处理，其后的4个字节包含处理中断类型1的程序的入口地址，以此类推。

当发生中断时，CPU把当前的标志压入堆栈，便于以后恢复。然后清除IF（中断标志）和TF（跟踪标志）。IF决定处理器是否要响应中断，一旦清除IF之后，CPU就不能响应其他中断，直到用STI（置中断允许标志）命令置IF后才能响应。这个机制保证了中断处理程序本身不会被中断。接着把CS（代码段）寄存器和IP（指令指针）寄存器的当前内容压入堆栈，这使得CPU能够在中断处理完毕之后接着运行被中断的程序。最后，CPU按照中断类型号在中断向量表中找到对应的中断处理程序的入口地址，把它们装入CS和IP。

中断处理程序通过IRET（中断返回）指令把控制返回被中断的程序。这条指令从堆栈中取出对应的值，恢复CS、IP寄存器和标志寄存器。注意，没有对其他寄存器作自动保存和恢复，因此，中断处理程序应该保存和恢复它所涉及的寄存器。

3.2.2 单步跟踪

一般情况下，CPU是顺序执行指令，除非在程序中遇到循环、中断、返回、转移或调用语句。但是，当设置了TF标志时，处理器每执行一条指令便产生一个INT1中断。如果调试程序把它自己的地址放入中断向量表中对应1号中断的项内，那么，每执行一条指令后，控制将转移到调试程序。因为INT指令在保存标志后将清除TF标志，所以，调试程序的指令不会再进一步产生INT1。

处理INT1的调试程序保存被调试程序的显示信息，显示寄存器和存贮器中的内容和处理用户的命令。当把控制返回给被调试程序时，调试程序恢复其显示信息，然后执行IRET指令。这

样，就把控制转移给用户程序，并从堆栈中退出保存的标志，从而恢复了 TF 标志。此时，用户的程序仍处于单步方式。接着将重复以上的过程。

然而，调试程序是怎样开始单步跟踪的呢？如果它在设置 TF 标志之前就把控制转到了用户程序，那么，就不可能产生单步中断。但是如果它先设置了 TF 标志，在把控制转移给用户程序之前，它将对自己的下一条指令进行单步跟踪。

解决的办法是需要有一条指令可以同时转移控制并且置标志。利用 IRET 指令，可以达到这个目的。如果调试程序是第一次把控制传送给用户程序，那么，它可以在执行 IRET 指令之前把标志以及那个程序的 CS:IP 压入堆栈，这样再执行 IRET 就可以同时转移控制并置标志了。

3.2.3 断点方式

调试程序还可以使用户程序连续地运行，直至某个指定的地址处停止。它是通过把该地址处的指令换成一条 INT3 指令来达到这个目的的。在把控制转移给用户程序之前，调试程序必须把它自己的地址放在中断向量表中对应于该中断类型的地方，使得执行该中断指令时，它可以重新获得控制。另外，调试程序还必须把被替换的指令保存起来，以便寄存器的内容不被破坏，可再次恢复。

通常，大多数中断指令都是两个字节长的，其中一个字节标志该指令是中断指令，第二个字节是该中断指令的类型号。但是，如果要设置断点的话，必须有一条单字节中断指令，否则，调试程序在设置断点时就必须替换两个字节。对于在双字节或三个字节指令处设置断点，还是可以的，如果要在单字节指令处设置断点，则不允许。因为，如果程序中有一条转跳指令，转跳到设置断点的单字节指令之后的话，那么，双字节中断指令的后一字节就会被当成指令码来执行。INT3 是一条单字节的中断指令，调试程序就是用它来实现断点操作的。

有些调试程序允许一次设置多个断点，如Symdeb、Pfix86等。在这种情况下，调试程序的INT3处理程序可以通过检查堆栈中的内容来辨别，当前执行程序的逻辑地址到达了哪个断点，一旦认定后，程序便停止运行。

总而言之，一个简单的调试程序由一个主体程序和两个常驻中断处理程序组成。主体程序必须把INT1和INT3处理程序的首地址填入中断向量表，调用SETBLOCK（DOS功能调用4AH），释放存贮区自由空间，为被调试程序让出内存；建立ASCII字符串（以0结尾的字符串）和文件控制块，为EXEC调用（DOS功能调用4BH）准备一个参数块；然后用EXEC功能调用和启动被调试的程序。两个常驻中断处理程序将显示寄存器的内容和处理调试程序的命令。

3.3 调试程序的应用

调试程序使用的好坏，将会直接影响汉化软件的周期和质量。因此，要想作好汉化软件的工作，熟练掌握调试软件的各种命令用法，是十分必要的。尤其是对于初学者，往往是调试程序的命令使用不当，长时间也找不到要汉化的软件关键模块，失去了信心，以致放弃了这项有意义的探索。下面主要针对Debug程序常用命令的使用方法，介绍其功能及实际操作。

3.3.1 Debug 程序的调用

在系统提示符下用下列命令格式把要调试的程序或Debug调入内存。

C>DEBUG [d:][path][filename[.ext]][parm]

在命令行中，DEBUG是调试程序的文件名；其后所跟的一系列内容都是要调试程序的文件标识符和参数。如果在命令中规定了文件标识符，则在DOS把Debug程序装入内存后，Debug程序还把要调试的程序（根据文件标识符）调入内存。若在命令

中没有规定文件标识符，则 Debug 程序自身与内存中的内容打交道；或者通过 Debug 的命令，从磁盘上调入要调试的程序和磁盘上指定扇区的数据。命令行中的 parm 参数项，是指 Debug要调试的程序所接收的数据参数。例如 WardStar 字处理程序可直接装入被编辑的文本文件，命令格式为：

C>WS BSF.TXT

如果要想用 Debug 程序观察 WordStar 把被编辑的文本文件 BSF.TXT 装入内存或在内存中传送和编辑的情况，则可用代参数的方式用 Debug 调入内存进行跟踪观察。

C>DEBUG WS.COM BSF.TXT

这样，DOS 将会把文件后面所指定的参数带入内存，放置在程序段前缀中未格式化参数区，其地址在 DS:0080H 处，该地址处也作为默认的磁盘数据传送区。

当 Debug 程序调入内存后，将显示一个提示符“—”。说明当前系统在 Debug 程序的控制之下，所有的 Debug 命令，也只有在出现该提示符之后才有效。

进入 Debug 程序后，Debug 将对寄存器和标志位进行初始化。它把各个寄存器和标志位设置成如下状态：

(1) 各段寄存器 (CS、DS、ES 和 SS) 置于自由存贮空间的底部，也就是 Debug 程序结束处后面的第一个段。

(2) 指令指针 (IP)置 0100H，若是 EXE 文件，指针值将由文件的定位参数给出。

(3) 堆栈指针置到段的结尾处，或者是装入程序的临时底部，取决于哪一个更低。

(4) 仅调入 Debug 时，所有通用寄存器都被设置为 0，若调用 Debug，其后包含一个要调试的程序的文件标识符时，则 CX 中包含以字节表示的文件长度。如果文件大于 64KB，进位到 BX 中，文件长度包含在 BX 和 CX 寄存器中(BX 是高位)。

(5) 标志寄存器的 8 个标志位全部置为清除状态。

(6) 磁盘传送缓冲区默认在数据段 DS 的 0080H 指针处。

如果 Debug 所调入程序是具有 EXE 扩展名的执行文件，则 Debug 必须根据该文件的重定位表进行再分配，把各寄存器内容和指针等内容设置为文件所规定的值。

3.3.2 Debug 程序使用要点

Debug 是一个最底层的调试程序，一旦进入它的控制之下，计算机的实际内存全部交给用户，到 00000H 到 FFFFFH 共 1 MB 的寻址范围。使用者可观看内存中任何一处的数据单元和程序代码，毫无任何限制。在使用 Debug 程序调试程序时，有如下一些要领是需要注意的：

(1) Debug 命令通常都是一个字母，后面跟有一个或多个参数；

(2) 命令和参数允许用大写或小写字母及其混合方式输入。

(3) 命令和参数之间，可以用空格和定界符分隔，然而，定界符只是在两个邻接的十六进制值之间才是必需的。下列命令是等效的：

```
dcs:100 120
d cs:100 120
d,cs:100,120
```

(4) 可以用按Ctrl-Break 键来停止一个命令执行，返回 Debug 的提示符。

(5) 每一个命令，只有在按了 Enter（回车）之后才有效，并开始执行。

(6) 若一个命令产生相当大数量的输出行时，在屏幕上卷一行以前，为了能读清楚它，可以在显示过程中按 Ctrl-NumLock 组合键，暂停其上卷，然后按任何一键再重新滚动显示。

(7) 若 Debug 检测出一个命令行的语法错误，则给出提示和指示错误的所在，例如：

```
d DS:100 cs :120
          ^
        Error
```

(8) Debug程序不像Symdeb那样具有设置断点的命令，但是可以用修改代码的方式来设置，即用一个中断类型3的指令(INT 3,操作码为CCH)来代替被调试程序在断点地址处的指令操作码。当程序执行到一个断点地址时，便停止运行，并且显示CPU内部所有寄存器的内容及标志位状态。然后，再把断点处的原程序恢复，即可继续运行。

3.3.3 Debug命令

1. 汇编命令 (Assemble Command)

用途：把8088/8086的汇编语言语句直接汇编到内存中。

格式：A [地址]

若在调试程序时发现程序中的某一部分需要改写，或要增补一段程序等，就可以直接在Debug下输入、汇编、运行和调试这一段程序。尤其是要汉化的软件，是要经常使用这条命令的，这比每一次修改都要经过编辑、汇编、连接等过程要简便多了，特别适用于编制较小的命令文件。

从命令后面指定的地址开始，可以输入汇编语言的语句，A命令把它们汇编成机器码，从指定的地址单元开始连续存放。

如果在命令中没有指定地址，则接着前面汇编命令的最后一个单元开始。当输入的格式及汇编语句出错时，Debug将显示错误提示信息，并且重新显示当前的汇编地址，等待输入。

输入给汇编命令的所有数值都是十六进制的1—4个字符。前缀助记符必须在操作码之前输入，也可以分行输入。最基本的段助记符是CS:、DS:、ES:和SS:，远程返回的指令用RETF。字符串操作助记符必须明确指定字符串的长度。例如：MOVSW是传送字字符串，而MOVSB是传送字节字符串。根据到目标地址的字节的多少，将自动汇编短的、近的或远程的各种转移(Jumps)和调用(Calls)指令，可用NEAR或FAR前缀来取代这些目标地址。

Debug不能确定某些操作数涉及的是字存贮单元还是字节存

贮单元。在这种情况下，必须用前缀“Word Ptr”（可缩写为“WO”）或“Byte Ptr”（可缩写为“BY”）来指明数据的类型。例如：

```
NOT BYTE PTR[300]
DEC WO [SI]
```

Debug 也不能确定一个操作数是立即数还是存贮单元的地址。所以，要把存贮单元的地址放在方括号内，来表明存贮单元的操作数。例如：

```
MOV AX,21; 把 21H 送至 AX
MOV AX,[21]; 把地址为 21H 及 22H 们存贮单元的内容
                送至 AX
```

汇编还包括两个一般的伪指令，DB 指令将把字节汇编到内存单元中，DW 把字值直接汇编到内存中。例如：

```
DB 1, 2, 3, "THIS IS AN EXAMPLE"
DW 1000,2000,3000,"BACH"
```

Debug 还支持所有形式的寄存器间接寻址命令。例如：

```
ADD BX, 34[BP+2],[SI-1]
POP [BP+DI]
```

A 命令的使用如下所示

```
-A 100
1F51:0100 DB    41, 42, 43
1F51:0103 MOV   AX,0600
1F51:0106 MOV   CX,0000
1F51:0109 MOV   DX,184F
1F51:010C MOV   BX,0000
1F51:010F INT   10
1F51:0111
```

最后一行不输入内容，直接按“回车”便可结束汇编输入，返回 Debug。

2. 比较命令（Compare Command）

用途：比较两个内存块的内容，第一块由范围指定，另一块由地址指定起始处，大小和第一块相同。

格式：C［地址1］［范围］［地址2］

地址1是第一个内存块的首地址，地址2是第二个内存块的首地址，范围可用长度L指定，也可用第一个内存块的尾地址指定。例如

-C 100 L30 500

或 -C 100 1FF 300

如果两个内存区域的内容完全相同，则命令不返回任何信息，并且直接返回Debug提示符。若有不一致的数据时，则将显示出地址和两处数据不相同的内容。如；

IF 51:0105 06 00 IF 51:0302

IF 51:010F CD 25 IF51:030 C

3. 显示命令(Dump Command)

用途：用于显示内存中指定范围的内容。

格式：D［地址］［范围］

如果在D命令中指定了地址或范围，该区域中的内容将被显示出来，满屏幕后自动滚行。若D命令不带参数，则显示开始地址DS:100H或是前一个D命令所显示的地址之后的128个字节。下面的例子作为从起始地址100H开始显示到14FH处的信息：

```
-D100 14F
19F9:0100   0C 83 7C 0A 00 75 1C 8A-44 08 30 E4 25 04 00 75   ..|..u..D.0.%..u
19F9:0110   12 56 E8 C3 02 8B E5 85-C0 74 08 B8 FF FF 83 C4   .V.......t......
19F9:0120   08 5D C3 8B 76 0C 8A 44-08 30 E4 25 04 00 74 07   .]..v..D.0.%..t.
19F9:0130   C7 46 06 01 00 EB 29 8B-76 0C 8A 44 08 30 E4 25   .F....).v..D.0.%
19F9:0140   02 00 74 08 B8 FF FF 83-C4 08 5D C3 8B 76 0C 8A   ..t.......]..v..
```

被显示的内容分为三部分；左边内容是十六进制的地址值，它以节为单位指定或默认段地址和偏移量。当中部分是以十六进制值表示的ASCII代码，每一行(节)显示16个字节，在第8个和第9个字节之间有一短线“-”。每行始于16个字节的边界，每次显示都是按页分开的。最右边部分是可显示的ASCII字符，

其可显示的范围为21H—7EH，控制码或扩展ASCII码用圆点（·）表示。倘若Debug本身已经过汉化的话，右边字符部分将可以把汉字信息和扩展ASCII字符显示出来。

D命令还可以指定段地址在某一个范围内进行显示，例如：

```
-DCS:210 L30
1600:0210  00 43 4F 4D 53 50 45 43-3D 43 3A 43 4F 4D 4D 41  .COMSPEC=C:COMMA
1600:0220  4E 44 2E 43 4F 4D 00 50-52 4F 4D 50 54 3D 24 70  ND.COM.PROMPT=$p
1600:0230  24 67 00 00 01 00 5A 3A-5C 50 55 42 4C 49 43 5C  $g....Z:\PUBLIC\
```

4. 放入命令（Enter Command）

用途：修改存贮单元的内容或将字节值放入内存中指定的地址处。

格式：E［地址］［“内容表”］

如果使用带有选择项“内容表”的E命令，可以自动地替换字节值，若出现命令错误，将不会有字节值被修改。如果不选择“内容表”，Debug则显示该地址及其内容，并在光标处显示一个“.”等待输入或修改，只要在当前的数值之后输入要修改的数值即可。若输入的是非法的十六进制数值，或包含两个以上的数字，则非法或多余的字符不作回送。

输入完一个字节，只有按空格键才能使光标推进到下一个字节。如果要改变其内容，只须按照上面讨论的方式打入一个新值即可。当空格键移动超过8个字节的边界时，Debug将会自动转到新的一行显示。若输入一个减号“-”，可以退回到上一个字节的地址处，在任何字节位置都可以用回车键结束E命令。例如，打入下列命令

E CS:120

假定在Debug下显示下面的内容

0F9C:120 EB._

为了将EB改为41，则在光标处输入

0F9C:120 EB.41_

按空格键则可以依次显示和修改后面的字节，直到回车为止。

使用选择项“内容表”的方式修改数据也非常方便，其命令和信息如下所示：

```
-D100 L10
19F9:0100    46 69 6C 65 20 4E 6F 74-20 46 69 6E 64 2C 20 50  File Not Find, P
-E100 'WHAT IS A FILE'
-D100 L10
19F9:0100    57 48 41 54 20 49 53 20-41 20 46 49 4C 45 20 50  WHATIS A FILE P
```

5. 填充命令(Fill Command)

用途：把给定的值填入指定的内存区域中去。

格式：F [地址] [范围] [“内容表”]

如果内容表中所含的字节数比范围小，那么，内容表将被重复使用，直至指定的范围被填满。若内容表含有的字节数比范围大，多余的表内容将被忽略。例如：

```
-F200 L30 31 33 35
-D200 L30
19F9:0200  31 33 35 31 33 35 31 33-35 31 33 35 31 33 35 31  1351351351351351
19F9:0210  33 35 31 33 35 31 33 35-31 33 35 31 33 35 31 33  3513513513513513
19F9:0220  35 31 33 35 31 33 35 31-33 35 31 33 35 31 33 35  5135135135135135
```

以上命令是从偏移地址 0200 处开始用 3 个 ASCII 字符内容填充 30H 个存贮单元。如果在文本显示下，向视频缓冲区填充，则会在屏幕上显示填充的信息。若在图形显示模式下，则会把填充的数据以点阵方式显示出来。例如：

—F B800:0 L1000 35 07

6. 执行命令(Go Command)

用途：执行正在调试的程序

格式：G [=地址1] [地址2]

第一个参数=地址 1 规定了程序执行的起始地址；以 CS 中的内容作为段地址，以等号后面的地址值作为地址偏移量。如果 G 命令之后不带参数，则从当前的 CS:IP 所定位的地址处开始执行。后面的地址值是执行程序的结束地址，也就是说是Debug的断点地址，程序执行到这里停止运行，控制返回 Debug 的提示符。

当程序一旦执行完毕，Debug 显示信息“Program terminated normally”，即程序正常结束，若还要执行此程序的话，必须退出 Debug，重新把程序调入内存。

在命令中的地址参数项中所指定的地址单元，必须包含有效的指令码，否则将会出现不可预料的结果。

7. 运算命令(Hex Command)

用途：完成两个指定参数之间的十六进制算术运算。

格式：H [数值1] [数值2]

这条命令把两个十六进制数相加，然后用第一个值减去第二个值。在同一行上显示它们的和与差，这条命令主要用于在内存中的地址变换时作参考。例如，打入以下的命令：

—H 28F 30D

Debug 完成运算后显示以下结果。

059C FF82

8. 输入命令(Input Command)

用途：从指定的端口输入和显示一个字节内容。

格式：I [端口值]

这条命令允许一个十六进制的端口地址值，在输入命令再回车后，Debug 将从指定的端口读入一个字节并且显示出来，如

—I 3FB

IF

9. 装入命令(Load Command)

用途：将文件或磁盘数据装入内存。

格式：L [地址 驱动器号 扇区号 扇区数]

装入命令分为带参数和不带参数两种基本功能：

(1) 带参数命令是指磁盘上指定区域的内容，装入到内存指定区域中，参数的意义是：地址是指数据装入内存的起始地址，驱动器号是指磁盘的逻辑号，如A:=0，B:=1，C:=2…。扇区号是指从磁盘的哪个逻辑扇区开始读数据。扇区数指需要读入几个扇区的数据，最大不超过80H。当用 L 命令装入数据时，

没有指定内存段地址，则默认段地址是当前的CS段值。例如：从C盘的0号扇区共有8个扇区的数读入内存地址190F:100处，可输入命令：

-L 190F:100 2 0 8

用此种方法观察磁盘上某一个逻辑扇区的数据是很方便的。

(2) 不带参数的L命令是用于把指定的文件装入内存中。通常先用N命令指定一个文件名：

-NFILE.COM

然后再简单打入L命令，即可实现。此命令装入已在CS:5C地址处格式化的文件控制块所指定的文件。所以，必须用N命令来认定。

如果命令中没有规定地址，则文件装入到CS:100开始的内存区域中；若命令中指定了地址，将会把文件内容装入指定的区域中。但若是具有扩展名.COM或.EXE的文件，则始终是装入到CS:100的区域中，即使在命令中指定了地址，这些地址也将被忽略。

在BX和CX中包含所装入文件的大小；如果所装入的文件是.EXE扩展名的执行文件，则BX和CX中包含实际的程序长度。

10. 传送命令(Move Command)

用途：把由范围规定的内存单元的内容传送到所指定的地址区域中去。

格式：M [地址1] [范围] [地址2]

这种操作是映射式的传送。把地址1的内容按范围的大小传送到地址2去后，地址1的数据不会丢失。对于从较高地址向较低地址的传送，其顺序过程是先传送最低端的数据，然后向最高端进行；对于从较低地址向较高地址的传送，其顺序是先传送最高端的数据，然后向最低端进行。

如果被传送的块内空间没有新的数据写入时，传送之前的数据将被保留。M命令按所讨论的顺序将数据从一个区域复制到另

一个区域，并写在新的地址上。如：

-M 190 A:300 L 120 7000:500

Debug 把 190A:300 处的 120H 个字节的内容传送到 7000:500 的地址空间去。为了观察传送结果，可以执行 D 命令来显示传送的内容。

11. 命名命令(Name Command)

用途：给文件命名。

格式：N [文件名$_1$] [文件名$_2$]

N 命令有两个功能；第一，为正在内存中汇编和调试的程序命名、提供文件名参数。第二，是为 L 和 W 命令提供读写的文件名。如果在进入 Debug 程序时没有把文件调入内存，用 L 命令装入文件之前必须先使用 N 命令。例如

—N file1.exe

—L

—G

下面的命令也非常有用，尤其在调试或跟踪多模块的文件程序时，非常方便。

—N file$_1$.exe

—L

—N file$_2$.dat file$_3$.dat

—G

在上面的例子中，N 命令设置的 file$_1$.exe 文件名是为后面的 L 命令调入 file$_1$.exe 文件到内存作准备的。随后 N 命令再使用一次，是指定 file$_1$.exe 所使用的参数，当用 G 命令运行时，file$_1$.exe 被执行，这就类似在系统提示下输入 file$_1$、file$_2$.dat file$_3$.dat 一样。

文件前缀有以下几个内存区域受 N 命令的影响：

CS:5C file$_1$ 的文件控制块

CS:6C file$_2$ 的文件控制块

CS:80H 文件名参数的长度字节

CS:81H 二次命名的参数内容

在 CS:5C、CS:6C处是文件控制块(FCB)，系统是根据该地址的信息对文件进行操作的。在 CS:81 处，放置文件名后面的参数，其内容与任何在 MS-DOS 命令级下输入的相同、对任何命令都是合法使用的开关与分界符。如：

C>Debug Drawe。COM Point,80,100

划线部分则为文件的参数内容。

12. 输出命令 (Output Command)

用途：把字节参数规定的内容送到指定的输出端口。

格式：O 〈字节值〉

对于一个有效端口，允许指定十六位的地址。下列命令是向 M6845 彩色寄存器端口送一个参数，可改变屏幕色彩：

—O 3D9 09

13. 过程命令(Process Command)

用途：执行一个过程或子程序。

格式：P [=〈地址〉] [〈数值〉]

P 命令可以连续执行一段过程，像 LOOP 一类的循环指令，可用 P 不带参数使其执行到 CX 等于 0。若指定了具体的数值，Debug 将在过程结束后继续执行数值规定的若干条指令。

该命令的另外一个功能是执行 CALL 指令后面跟的地址处的子程序，在执行完该子程序后，将自动返回 CALL 指令紧跟着的一条指令地址处终断执行，并且显示各寄存器内容以及标志位状态。这一功能对于大程序的跟踪分析比较实用。

14. 退出命令(Quit Command)

用途：结束 Debug 运行，退回到系统。

格式：Q

Q 命令不带任何参数，它不保存正在内存中运行的文件而退出 Debug，并立即返回到 DOS 系统的提示符下。

15. 寄存器命令(Register Command)

用途：显示或修改寄存器和标志位的信息内容。

格式：R [寄存器名]

R 命令包含下述三种功能：

(1) 能显示 CPU 内部的所有寄存器内容以及全部标志位的状态。

例如：在 Debug 提示符下输入不带寄存器名 的 R 命令，将显示如下信息：

```
-r
AX=0000 BX=0002 CX=0800 DX=0000
SP=0055 BP=0000 SI=0000 DI=0000
DS=1F56 ES=1F56 SS=3C75 CS=3CE5
IP=02FA NV UP EI PL NZ NA PO NC
3CE5:02FA FA          CLI
```

前四行显示了所有 CPU 内部寄存器的内容和全部标志位的状态，第五行显示了当前 CS:IP 所 定 位的地址及该地址处指令的机器码和反汇编符号，这也就是下一条即将要执行的指令。

(2) 可以显示和修改指定寄存器的内容，例如修改AX内容：

```
-RAX
AX 0000
:—
```

若不需修改，按回车键即可。

若要修改，则可以输入1—4个十六进制的字符值，再按回车键。

(3) 可显示和修改标志位状态。

在 8088/8086 中，共有 9 个标志位，其中，追踪标志 T，不允许直接修改。其他 8 个标志均可显示和修改，8 个标志位的显示次序和符号如下：

标志位	置位	清除
溢　出	OV（是）	NV（否）
方　向	DN（递减）	UP（递增）
中　断	EI（能）	DI（不能）

符号	NG（负）	PL（正）
零标志	ZR（是）	NZ（否）
辅助进位	AC（是）	NA（否）
奇偶校验	PE（偶）	PO（奇）
进位	CY（是）	NC（否）

当输入RF命令时，将在一行内显示所有标志位，光标停留在短线处，等待修改：

```
-RF
NV UP DI PL NZ NA PO NC _
```

这时，可用两个字母与上述显示相反的标志输入，若连续修改几个标志，可紧接着输入，当中不必留有空格。若输入错误，Debug将会提示，标志位不变化。修改标志的方法如下：

```
—RF
NV UP DI PL NZ NA PO NC_OVEIPE
```

16. 搜索命令(Search Command)

用途：在指定的范围内搜索指定的字节或信息。

格式：S ［地址］［范围］［“内容表”］

内容表中可以使用一个或多个任何指定的ASCII代码，在指定的范围中若没有搜索到，将不显示任何信息，返回Debug提示符。如果搜索到了，则显示含有内容表的所有地址。例如：

```
-S CS:100 L500 41
```

或

```
-S CS:100 L500 “A”
05BF:023F
05BF:0485
```

上面的两条命令是等效的，若内容表的信息用引号括起来，就必须使用ASCII字符，用此命令来查找字符串将非常方便。假如Debug调试程序是经过汉化的，S命令与E命令配合使用，可作为汉化任何软件提示信息的好工具。例如：

```
—S 500 2000“File Not Find”
- 059F:15B0
```

—E 059F:15B0 “文件没找到”

17. 跟踪命令(Trace Command)

用途：一条指令一条指令地执行。

格式：T [=地址] [数值]

如果不带参数时，则执行一条指令，CPU将产生一个硬件单步中断，使程序停下来，显示CPU的所有寄存器内容和标志位的状态，并返回Debug。这可以用来观察程序动态运行的中间结果，并作必要的修改。

如果T命令后跟着地址参数，其被调试程序将从指定的地址单步执行。

如果T命令后跟有数值，将说明T命令连续执行几条指令。不过，每条指令执行后均显示寄存器和标志位信息。例如：

```
—T2
AX=00CD  BX=0002 CX=0800 DX=0000 SP=0053  BP=0000
SI=0001    DI=0000  DS=1F56 ES=1F56 SS=3C75 CS=3CE5
IP=03D9 NV UP EI PL ZR NA PE NC
3CE5:03D9 0AC0                    OR        AL,AL
—T
AX=00CD BX=0002 CX=0800 DX=0000 SP=0053 BP=0000
  SI=0001 DI=0000 DS=1F56 ES=1F56 SS=3C75 CS=3CE5
  IP=03DB NV UP EI NG NZ NA PO NC
3CE5:03DB 740D                    JZ      03EA
—T
AX=00CD BX=0002 CX=0800 DX=0000 SP=0053 BP=0000
  SI=0001 DI=0000 DS=1F56 ES=1F56 SS=3C75 CS=3CE5
  IP=03DD NV UP EI NG NZ NA PO NC
3CE5:03DD 3C20                          CMP   AL,20
```

18. 反汇编命令(Unassemble Command)

用途：把机器码翻译为助记符汇编语言，并显示与之相对应的地址、机器码和汇编语句。

格式：U [地址] [范围]

若在内存的某一区域中，已存在某一程序的目标码，为了解和分析此程序的内容，就希望能把目标程序反汇编为源程序，则可用U命令来实现。如果U命令后面没有参数，U命令则从当前的CS:IP或上一次U命令结束地址处开始反汇编。例如：

```
—u
3CE5:030F 8EC7        MOV   ES,DI
3CE5:0311 56          PUSH  SI
3CE5:0312 51          PUSH  CX
3CE5:0312 50          PUSH  AX
3CE5:0314 1E          PUSH  DS
3CE5:0315 06          PUSH  ES
3CE:50316 2E8C068502  MOV   CS:[0285],ES
3CE5:031B B82D3D      MOV   AX,3D2D
```

如果输入的U命令带有参数，Debug将把指定地址范围内的所有代码全部反汇编出来。当一个屏幕显示满后，会自动作滚行处理。下面列出的两条反汇编命令是把某一范围的代码进行反汇编。

—U 16F2:500 7000

—U 16F2:500 L7000

当反汇编显示结束后，控制将自动返回Debug提示符。

19. 写盘命令（Write Command）

用途：把内存中的数据和程序文件存写到磁盘上。

格式： W [地址 驱动器号 扇区号 扇区数]

如果使用不带参数的W命令，必须由N命令指定文件名，并且在BX和CX寄存器中设置好将要写入的字节数。W命令默认存盘的起始地址是CS:100，若指定了地址值，Debug将从指定的地址处开始，把数据存写到盘上。假如一个被调试的程序，曾经使用过G和T等命令，再用不带参数的W命令写盘时，必须重新调整BX:CX寄存器的内容，否则，容易造成数据和程序的丢失。

对于.EXE含有重定位信息的文件，不允许用W命令存盘，

只有改名后才可以使用。

当使用含有带参数的W命令时，写盘的起始地址由地址参量决定，驱动器名定为0=A盘；1=B盘…，并指定存入哪个扇区，存写多少个扇区均由参量指定。写盘命令与L装载命令是相对应的，一次最多写入80H扇区。例如：

-W CS:100 1 40 20

把内存中CS:100开始的内容写入B盘的40H号扇区，共写入20H个扇区数据。

3.4 程序反汇编的要点

通常，在着手汉化一个软件之前，应该对该软件进行分析研究工作。这不仅是对要修改的模块具体分析，而且也要对需要汉化的软件的总体结构及其设计思想作一些大致的了解。由于目标文件均是经过编译后的机器代码，不易于阅读，因此，有必要对有关的程序进行反汇编，反汇编的目的就是把机器码再翻译成为具有助记符的汇编指令源程序。

3.4.1 软件反汇编

根据不同的软件，可以采用不同的方法，譬如，一些较小的目标文件，一般指几KB或十几KB的.EXE或.COM文件，可使用 Sourcer 反汇编器进行反汇编。在Sourcer程序的主菜单下，把要反汇编的目标程序调入内存，并指定源程序结果分析情况。

这种反汇编法允许用户指定反汇编的源程序是以.ASM为后缀的宏汇编文件，也可以指定为后缀为 .LST 的列表源文件。其反汇编的源程序还自动地把有关指令加上常规的注释或内存单元的地址。对于某些数据区会自动地跟踪排序，用宏指令DB和DW加以归类，有些单元的地址代码，则用EQU语句指明。

以下是通过Sourcer对Disp.COM程序的反汇编结果：

```
PAGE 01, 132
;=============================================
;==                DISP.COM                  ==
;==      Created:  11-Jun-90                 ==
;==      Passes:  5      Analysis Flags on: HIL  ==
;=============================================
.286CPU
SEG_A     SEGMENT
          ASSUMECS:SEG_A,DS:SEG_A
          ORG  100h
disp        PROC FAR
start
          JMP SHORT LOC_1      ;(010F)
DATA_1    DB      'How are you $'
LOC_1:
          MOV DX,OFFSET DATA_1;(6850:0102=48H)
          MOV AH,9  ;DOS Services AH=Function 09H
          INT 21H    ;Display char string at ds:dx
          RET
disp      ENDP
SEG_A     ENDS
          END START
```

利用 Sourcer 反汇编的文本，是一个可读性很好的宏汇编源程序文本。用打印机把清单打印成册，对源程序的分析研究是非常方便的。当修改软件的方案确定之后，就可以用 EDLIN、WordStar 以及 PE 等文本编辑对源程序进行修改或重新编辑。最后，把修改过的源程序再用 MASM 宏汇编软件进行编译，用 LINK 软件进行联接后则可以产生汉化后的目标代码文件。如果没有满足要求，可重复以上操作，直到成功为止。

3.4.2 Debug 直接反汇编

这种方法用于局部模块的反汇编或者读取某一内存块的数据，也是高级程序员最常用的方法。即用 U 或 D 命令把反汇编

的结果直接送到打印机输出。

例如，对 Drawe.com 文件局部反汇编，先用下列命令把该文件调入内存：

C>Debug Drawe.com（回车）

此时，系统已进入 Debug 状态，打开打印机电源并装好打印纸，按下 Ctrl-P 键之后，再用 U 命令或 D 命令把指定地址的程序和数据送往打印机，打印机的输出结果与屏幕显示的反汇编信息一致。操作结果如下列程序所示：

```
-^P                    PUSH    DX
-U149 169              MOV     AX,[BX+06]
1A10:0149 52           CALL    015C
1A10:014A 8B4706       POP     DX
1A10:014D E80C00       SUB     AL,DL
1A10:0150 5A           XCHG    DL,AL
1A10:0151 28D0         MOV     AL,[BX+09]
1A10:0153 86D0         AND     AL,0F
1A10:0155 8A4709       JMP     017A
1A10:0158 240F         AND     AX,0F0F
1A10:015A EB1E         MOV     DH,AH
1A10:015C 250F0F       AND     CH,CH
1A10:015F 88E6         MOV     CL,0A
1A10:0161 30ED         XOR     CL
1A10:0163 B10A         MOV     AL,DH
1A10:0165 F6E1         MUL
1A10:0167 00F0         ADD
1A10:0169 C3           RET
```

若是 .EXE 文件或大于 64KB 的各类目标程序，在反汇编时，需要指定段地址。

数据区或存贮单元的内容，使用 D 命令，用同样的方法把各种需要的数据内容，送往打印机。

3.4.3 DOS 重定向反汇编

MS-DOS 标准输入/输出设备重定向功能允许程序不是从键

盘（标准输入设备）的其他设备上输入，或者除了向屏幕（标准输出）输出外，还能把输出重定向给非屏幕的其他设备。譬如，把输出送往一个文件、打印机或通讯口等。使用这种方法有利于生成或打印出连续的分段反汇编程序。通常是把反汇编的源程序先输出到一个文件中，经过 WordStar 一类的字处理软件整理后，再把源程序清单打印出来，当然，也可以直接把反汇编程序重定向到打印机，直接得到程序清单。

以下是重定向的基本命令：

C>dir>[d:][Path]filemane

C>dir≫[d:][Path]filemane

第一条命令是指把 dir 显示目录的结果写到所指定的驱动器和路径下的文件中。第二条命令的结果是把 dir 显示目录的结果追加到指定的文件之后。

根据 DOS 系统传送标准输入/输出功能，来建立反汇编源程序的生成环境。这种传送可把几个带有自动重定向标准输入/输出的程序连接起来，在命令行上要用竖杠符"|"分隔开。

以下为重定向反汇编的操作步骤：

(1) 用 EDLIN 或 WordStar 编辑一个文件，取名为 UASMF。文件中是根据反汇编所指定的 Debug 命令，必须是一行一条命令，如：

```
1:U 10E 135
2:D 4D0 L20
3:U 38C 39D
4:Q
```

(2) 用重定向传送方式操作。该功能可直接在系统提示符下键入命令，也可以用批文件处理方式进行。如果要把 ESTS.COM 文件的反汇编结果按 UASMF 文本命令行的方式写到一个文件中，则用下面命令操作即可：

```
C>TYPE UASMF | DEBUG ESTS.COM>ASM1.TXT
```

这条命令的执行结果将会按照UASMF文件的命令行逐条把ESTS.COM文件中指定地址的目标机器码反汇编成为ASM1.TXT源程序文本。通过字处理程序的编辑处理后，即可把程序清单打印出来。

（3）如果要把反汇编的结果直接打印出来，则用下列命令即可：

C>TYPE UASMF ¦ DEBUG ESTS.COM>PRN

执行的结果如下所示：

```
-U 10E 135
3FD9:010E  FC            CLD
3FD9:010F  31C0          XOR   AX,AX
3FD9:0111  BFF004        MOV   DI,04F0
3FD9:0114  B9E002        MOV   CX,02E0
3FD9:0117  F3            REPZ
3FD9:0118  AB            STOSW
3FD9:0119  1E            PUSH  DS
3FD9:011A  8ED8          MOV   DS,AX
3FD9:011C  8E1E4200      MOV   DS,[0042]
3FD9:0120  2E            CS:
3FD9:0121  8C1E2404      MOV   [0424],DS
3FD9:0125  A17500        MOV   AX,[0075]
3FD9:0128  1F            POP   DS
3FD9:0129  A32604        MOV   [0426],AX
3FD9:012C  BF8100        MOV   DI,0081
3FD9:012F  E88E01        CALL  02C0
3FD9:0132  7256          JB    018A
3FD9:0134  F3            REPZ
3FD9:0135  AE            SCASB
-D 4D0 L20
3FD9:04D0   00 00 00 00 00 00 00 00-00  00 00 00 00 00 00 00  ................
3FD9:04E0   0C BE 18 7B E8 68 01 E8-FC 00 E9 AC FD 8A C2 A8  ...{.h..........
-U 38C 39D
3FD9:038C  07            POP   ES
3FD9:038D  BE000A        MOV   SI,0A00
3FD9:0390  89F7          MOV   DI,SI
3FD9:0392  B90205        MOV   CX,0502
3FD9:0395  F8            CLC
3FD9:0396  FD            STD
3FD9:0397  AC            LODSB
3FD9:0398  D0D0          RCL   AL,1
3FD9:039A  AA            STOSB
3FD9:039B  E2FA          LOOP  0397
3FD9:039D  07            POP   ES
-Q
```

许多需要汉化的软件，都分为好几个甚至是十几个模块，但是最主要的是一个核心程序。因为核心程序往往作为输入输出的控制和管理。尤其是 Microsoft 公司推出的软件包，它们的基本数据结构均大同小异。一般分为分配内存进行装载、数据单元、覆盖程序和核心程序代码等几大模块。如图 3-1 所示。

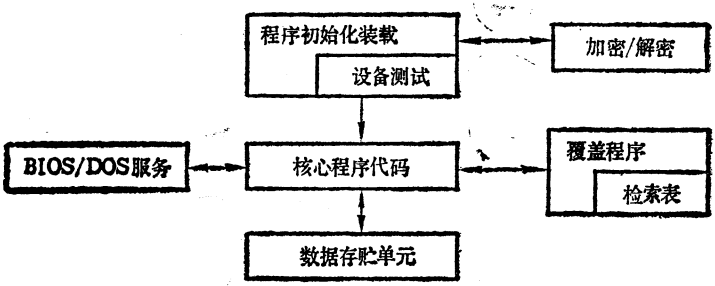


图 3-1 程序运行环境

但是，在 Debug 状态下用 U 命令观察反汇编程序时，有些是无效的汇编助记符，也就是说，是一些毫无逻辑、杂乱无章的代码。类似这种情况，即使反汇编出来，也将无法阅读。这有两种可能：一种情况是用 U 命令所指定的地址范围内不是代码段的程序信息，而是数据段或者堆栈段内的数据信息。有些程序也会在代码段中存贮一些标志信息，对于这些数据信息，Debug 当然是不可能把它们反汇编和汇编源程序的。如果是这一种情况，应该用 D 命令来读取其内容。

另外，有些软件，因为商业的需要，其内部的某些程序进行了加密处理。只有当程序运行到某一状态时，通过一些指定单元中的密钥参数对程序进行解密。软件的加密不外乎三种情况：一是硬加密，即在磁盘介质上作有标记。如激光加密方式，在磁盘上打有小孔，并作有所谓指纹标记作为测试条件。二是软加密，即用程序的方法根据某种算法把程序转换为不可识别的代码。当该软件运行时，再根据其指定的算法和关键字节逐步和全部解释为标准的目标代码。三是软硬件结合方式加密，即在介质上作有密

钥又在程序上进行了变换。这对软件的二次开发增加了分析程序、改造程序的难度。一般，汉化和改造这类软件需要有极大的耐心，逐步地读出其反汇编程序。即使是再困难的加密软件，也都能找到其攻击方法，有了解密，就可使得加密技术不断地发展，这是一对在矛盾中发展的技术，在分析或反汇编目标程序时应该加以注意。

第四章　软件汉化的步骤与方法

对于国际上许多著名的优秀软件，绝大多数都是以西文为背景的，不能直接有效地处理中文信息。在汉语国家和地区，这种情况，在很大程度上限制了计算机的应用和推广。解决这一问题的方法是，开发大量的汉字软件或者利用软件汉化技术，对原西文软件进行二次开发，使其不但能够处理西文信息，同时也能够在汉字系统下有效地处理汉字。具体的选择有如下几种：

1．直接使用西文软件。

这种方法要求使用计算机人员的技术和文化水平比较高。尤其是对基层的操作员，要求有一定的计算机知识和外语水平，这要通过长期的培训才行。目前很难达到这一标准，且计算机的输入/输出信息也很难与一些现有的事务处理票据相一致，因此，这种方法仅限于在一些科研部门使用。

2．完全抛开原系统，建立纯中文系统，开发纯中文的操作系统、高级语言和一些系统级的软件包。

有人建议开发具有汉语语词的高级语言软件，但是在信息时代，国际上的信息千变万化，在与世界各国的信息交往和通信时，就会遇到许多不便和限制。再者，如果重新开发一套软件（系统级的），需要花费大量的人力、财力和时间。虽然国内已有少量的自行开发的中西文系统，但是要适应和满足广大用户在各个领域中的应用还远远不够。因此，这个办法似乎可行，却又不太实际。

3．在原系统的基础上增加汉字信息处理功能。

有些系统从硬件上加以改造，或者对原系统不作任何修改，增加相应的汉字处理功能模块，使该软件能够处理中西文信息。硬件上的实现，大多是工作在文本字符方式下，限制了一些图形

软件的兼容性。对于那些仅用于文本数据处理的用户来说，基本上可以满足需要。

4. 对原系统进行局部汉化改造。

这种方法不需要重新编程，仅对软件的内部作一些适当的调整，打通汉字输入/输出通道，使之既保留了处理西文的功能，又能够处理汉字信息。这样，大大减少了开发经费、工作量和开发周期，是一种实用、经济又行之有效的方法，已得到许多专家的肯定。

4.1 汉化的基本要点

汉化软件的过程，是一个分析问题、发现问题和解决问题的过程。根据指定软件的层次，从分析理解、跟踪研究、修改调试到最后的产品鉴定，是一项复杂的工程。因此，应该有步骤地进行，大致如下：

(1) 首先要了解和掌握将要汉化的软件的基本结构，也就是指，对指定的软件运行、功能测试和初步的跟踪分析。在系统的内存中，先大致了解该软件的代码、数据的内部形式，并且摸清其数据传送流程和各个功能模块之间的内在联系，以便找出与汉字处理有关的部分。

(2) 采用“灰盒子”方法，在分析系统整体结构的基础上，重点分析系统的 I/O 流，找出限制汉字机内码流向的原因和解决办法。

(3) 把指定软件中与汉字处理有关的部分代码进行反汇编处理，程序部分可以文本文件的形式打印出源程序清单，数据信息部分可按其存贮结构生成文本，或者直接把数据区的信息打印出来。

(4) 仔细阅读反汇编出来的源程序清单，并结合在计算机上用调试程序对该软件进行局部模块的跟踪观察。在每次运行的断点处，记录下有关寄存器的结果、标志位的状态和程序在不同情

况下运行走向的地址值。

（5）根据程序跟踪的结果，参照源程序有关模块功能，绘制出原西文软件有关模块的程序流程图，作为汉化程序的基本依据。

（6）确定汉化方案，对原西文软件进行修改、调试和试运行。测试该软件原有的西文处理功能和经汉化后的汉字处理功能，应该在最大限度下保持兼容。

（7）整理和保存汉化过程中的资料文档，同时申请有关部门的技术鉴定，生成软件产品。

上述前5个步骤对于软件的有效汉化是非常重要的。这些步骤虽然还没有开始真正进行汉化的实验工作，但是对下一步的汉化工作来说，起着举足轻重的作用，它将直接影响汉化软件的技术性能和指标，因此，必须认真地进行软件系统的分析研究工作。一般的汉化流程如图4-1所示，当然，可以根据各人掌握系统的知识程度和汉化经验的不同，采用更直接的步骤进行。

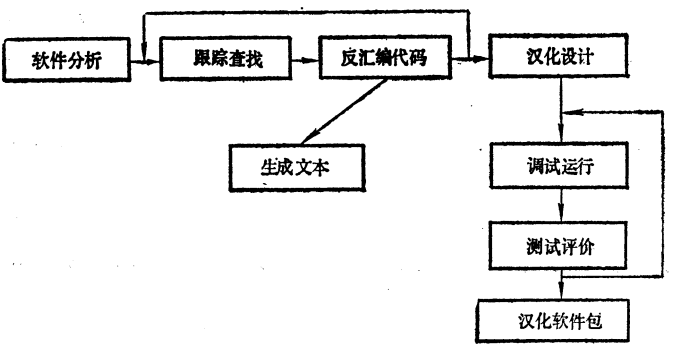


图 4-1 软件汉化的步骤

汉化一个软件，很大一部分工作是对原西文软件的跟踪、观察和分析，找出其与汉字信息处理有关的部分程序模块，进行相应的调整。那么，哪些程序模块与汉字信息处理相关呢？它们的处理方式与西文的处理方式有何区别呢？以下分几个要点加以说明。

1. 屏幕显示汉化问题

在IBM-PC/XT、AT等微机上所使用的CC-DOS汉字系统，是工作在图形显示方式的。汉字系统通过10H类中断的显示字符功能，把汉字或ASCII代码以点阵方式送到屏幕进行图形显示。而未经汉化的软件，为了提高屏幕的显示速度，往往避开BIOS视频中断的调用，把要显示的字符码和属性码从数据区中取出后，直接传送到视频缓冲区去，以达到快速显示的目的。这种数据传送的方法仅用于CRT字符模式的情况下，如果处在图形显示模式下，系统将会把传送到视频缓冲区中的字符码和属性码也作为图像点阵字节显示出来。例如41H是ASCII字符"A"的代码，在图形方式下将会把这个代码的01000001B两个小亮点显示出来。这样，势必造成屏幕信息的杂乱无章，使人难以辨认。因此，必须首先解决汉字的显示的功能汉化问题，这也是软件与用户之间最直接的界面。

2. 输入词法汉化问题

在一般软件系统中，对于用户输入的数据和语句，均是以一定的代码被程序识别的。尤其是一些应用软件包，在词法分析程序中，仅仅识别一定范围内的输入串，如果输入的汉字机内码在其所能识别的范围之外，在词法分析时，没有识别出来，系统给出错误提示，不予处理。因此，必须设法扩大其代码识别的范围，让中西文代码均能顺利通过词法分析程序，并给出相应的定义名，才能在该软件环境下处理汉字的输入信息。

3. 全屏幕编辑功能的汉化问题

许多功能齐全的软件自成系统，除了有语句解释和程序编译功能外，还具有全屏幕的字处理编辑功能和数据全屏幕操作功能。譬如；Turbo集成软件系列、dBASE、FoxBase和Windows等软件，就有良好的全屏幕编辑功能。还有一类软件，像Lotus 1-2-3、Supercalc和Mutiplan等等，属于电子表处理软件，也具有全屏幕表操作功能。这些软件，通常不支持图形汉字系统，

因此，需要进行修改。有时，全屏幕编辑模块的汉化，可能是某一软件汉化工作量最大的功能模块。

4. 字符处理及其定义域扩展

在许多西文软件中，对于数据的标识部分，如变量名、字段变量和文件名等的属性域，均按传统的方法定义在ANSI（American National Standards Institute）标准范围之内。实际上，是在ASCII码的标准级（前128个）范围之内。若超出此范围，将由屏蔽方式给与"滤除"，不予进一步处理。但是，在CC-BIOS汉字系统中，汉字的机内码是由扩展的ASCII码（160之后）表示，且每两个扩展ASCII码组成一个汉字机内码。因此，需要在此基础上扩充处理字符机内码的范围，使该软件输入输出字符代码的定义域能够识别并处理用于表示汉字机内码的扩展ASCII码。

5. 屏幕显示重新布局

针对各种类型的显示器环境，要对汉化过的软件在屏幕上的显示行数和有关信息的位置有所调整。因为CC-BIOS汉字系统是在图形方式下工作，通常，所显示的汉字是16×16点阵的图形字模。因此，显示器的分辨率，即垂直扫描线数决定了系统能够显示的行数。例如，分辨率为640×400的图形显示器，其纵向有400根扫描线，汉字纵向为16排点，那么，其最大显示行数为400/16=25行。再如，PC/XT机上配置的CGA彩色显示器，其分辨率为640×200，要显示16点阵汉字，并且每行之间有两根扫描线作为行间隔，那么，最大可显示行数为200/(16+2)≈11行。当然，由于环境和系统之异，还有16行、20行和30行的汉字显示系统。原西文软件绝大多数是以80×25字符方式显示的，如果汉化软件所要运行的环境不是25行显示的话，势必会产生滚行和闪烁不稳定的现象。尤其是具有全屏幕编辑功能的汉化软件，应该对其扫描的行限界单元、光标最末一行的定位数据或一些满屏幕显示的信息进行合理的适当调整。

6. 人机交互信息汉字化问题

面向用户的软件，为了便于操作，通常通过屏幕上显示的信息和选择菜单来帮助用户正确操作，如果操作失误或输入的命令不正确时，将会显示一些出错提示，通知用户，用户再根据提示操作下去。这一界面，实际上是人机信息交互过程。但是西文软件的信息提示都是用英文给出的，因此，阻碍了推广普及。

不过，对于有一定英文基础的计算机编程人员来说，也许并非希望把所有的显示信息全部改成中文信息，尤其是想要在中/西文显示方式下兼容使用的话，可以省去这项汉化工作。实际上，前5个步骤解决好了，其软件就已经可以在汉字系统下使用了，因此，把信息数据英译中，可以作为二级汉化，或者根据用户的要求作进一步开发。

当然，为了在国内普及和推广某一个汉化的软件产品，把软件中某些重要的信息或者出错提示译成中文也是有一定意义的，这可使其尽可能地接近人们的日常用语，较大程度地消除人机界面上的隔阂。

4.2 原西文软件的分析方法

大多数计算机系统的分析过程可分为四步完成：①理解当前的现实系统及环境，获得人工系统或者原始系统的具体模型；②从当前系统的具体模型抽象出当前系统的逻辑模型；③分析目标系统与当前系统逻辑上的差别，建立目标系统的逻辑模型；④为目标系统的逻辑模型作补充。

在软件汉化的初级阶段，对软件要进行深入细致的研究分析工作。通常采用以下几种方法进行。

4.2.1 静态分析法

所谓静态分析法，就是把与汉字处理有关的功能模块代码，利用反汇编的手段把其源程序清单打印出来。通过仔细研读程序，按其实现的目标和逻辑功能，绘出程序流程图。弄清楚在程

序中数据传送的来龙去脉、地址定位的算法和字符处理的方式等。其分析步骤如图 4-2 所示。

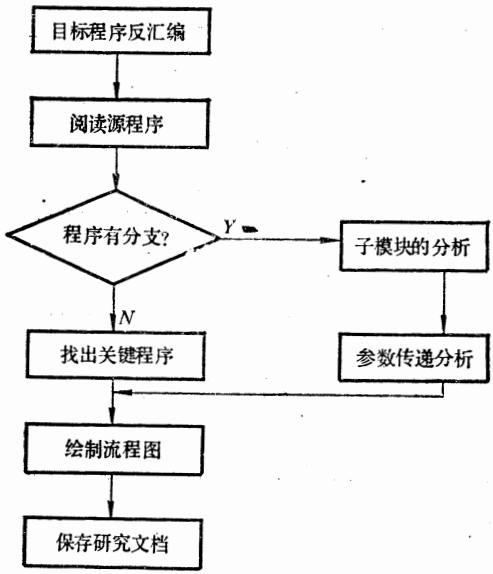


图 4-2 静态分析框图

例如，Pfix. EXE 是 Phoenix Software 公司推出的全屏幕工具软件包，其中有关屏幕显示模块的源程序如下所示：

```
CS:C96B   PUSH    EP
CS:C96C   MOV     BP,SP
CS:C96E   PUSH    ES
CS:C96F   XOR     DI,DI
CS:C971   MOV     AX,[BP+06]
CS:C974   MOV     ES,AX                    ;视频段地址
CS:C976   CMP     WORD PTR[3563],B800
CS:C97C   JNZ     C998
CS:C97E   PUSH    DS
CS:C97F   MOV     AX,0040
CS:C982   MOV     DS,AX                    ;BIOS数据区
CS:C984   MOV     BX,0049
```

```
CS:C987   MOV      AL,[BX]
CS:C989   POP      DS

CS:C98A   XOR      AH,AH
CS:C98C   MOV      SI,AX
CS:C98E   MOV      AL,[SI+3580]
CS:C992   AND      AL,F7
CS:C994   MOV      DX,03D8
CS:C997   OUT      DX,AL                    关闭屏幕
CS:C998   XOR      SI,SI
CS:C99A   MOV      CX,07D0
CS:C99D   PUSH     DS
CS:C99E   MOV      AX,[BP+04]
CS:C9A1   MOV      DS,AX
CS:C9A3   CLD
CS:C9A4   REPZ
CS:C9A5   MOVSW                             ;向屏幕送数据
CS:C9A6   POP      DS
CS:C9A7   CMP      WORD PTR [3563],B800
CS:C9AD   JNZ      C9CA
CS:C9AF   PUSH     DS
CS:C9B0   MOV      AX,0040
CS:B9B3   MOV      DS,AX
CS:C9B5   MOV      BX,0049
CS:C9B8   MOV      AL,[BX]
CS:C9BA POP        DS
CS:C9BB XOR        AH,AH
CS:C9BD MOV        SI,AX
CS:C9BF CALL       C954
CS:C9C2   MOV      AL,[SI+3580]
CS:C9C6   MOV      DX,03D8
CS:C9C9   OUT      DX,AL                    ;打开屏幕显示
CS:C9CA STI
CS:C9CB POP        ES
CS:C9CC POP        BP
CS:C9CD RET
```

在系统中，这个模块是供其他程序调用的子程序，其参数是通过堆栈进行传送的。通过仔细阅读与分析，按照程序指令的走向，绘制出图 4-3 所示的分析流程。

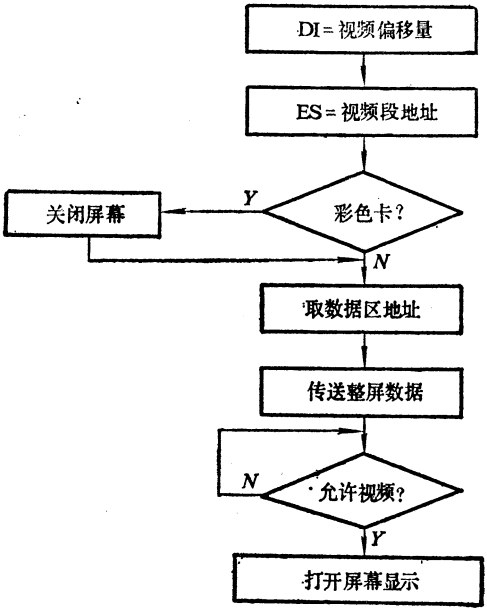


图 4-3 Pfix 显示模块分析

4.2.2 动态分析法

动态分析法是指利用动态调试工具，把需要汉化的程序调入内存，对其进行反复的跟踪、搜索，并且找到与汉字处理有关的功能子模块。可以利用设置断点或单步追踪命令、随时观察 CPU 各个寄存器的内容、各转移语句的条件和数据单元的数据信息。尤其是对一些覆盖文件，要注意它在内存中相互覆盖和被调用的情况，记录下程序走向的关键地址，便于在下一步汉化工作中使用。图 4-4 是动态分析的步骤流程。

例如，上述的 Pfix 中的显示模块，在动态环境下跟踪至此，

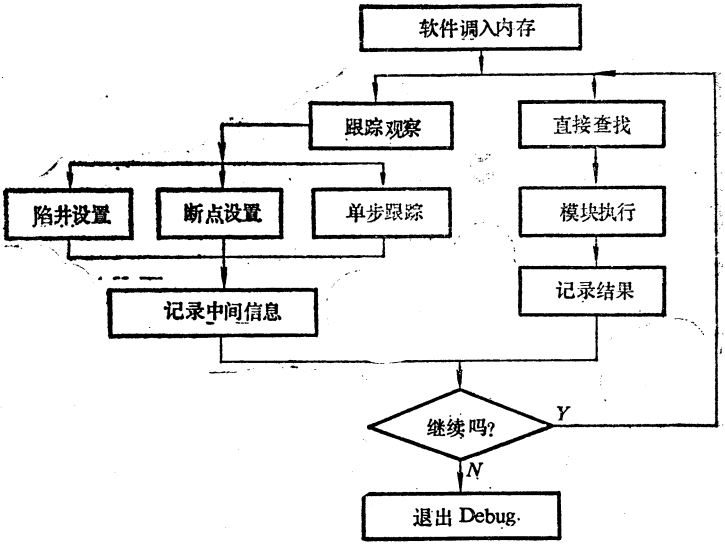


图 4-4 动态分析步骤

观察其从堆栈中取出的参数是什么，有关的内存单元(如[3563])中存放的是何种数据等。如果子模块不很大，可用单步命令T逐步执行一遍，以便观察其中间参数和运行结果，掌握程序的基本思想和运行规律。由此来绘出程序流程图。

当然，软件的开发与分析过程是复杂的，不能完全用工具来代替，许多工作还需要人的创造能力，软件的开发或汉化全部自动化也是很困难的。计算机的工具软件只能起到辅助作用，软件分析方法和软件工具有着紧密的联系。方法是主导，工具是辅助，而工具的实现又将促进方法的发展。

4.3 目标程序的跟踪技巧

在IBM-PC/XT等兼容机上，可利用各种动态调试软件对要汉化的软件进行跟踪。而最简单、最有效又最实用的工具软件就是Debug。

软件的汉化，首先应该确定该软件与汉字处理有关的一些子程序的地址。这个地址是指程序在内存中运行时的逻辑地址。但在实际工作中，对于一个几十KB甚至几百KB的程序，很难一下子就能找到有关子程序的具体地址，要想设置一个好的断点来观察其执行的情况也比较困难。因此，对于某一功能子程序的地址定位是一个技术性很高的工作，对于初学者，应该了解和掌握一些动态跟踪的技巧，以提高项目开发的效率。否则，会花费许多不必要的时间和精力。

这里讨论几种常用的跟踪查找的方法和原理，读者可以根据要汉化的具体软件选择使用。

(1) 地址的变换。通常，把后缀为COM或EXE的文件调入内存后，则根据当前的首地址就可以进行跟踪了，COM文件的首地址，即偏移量为100H。这类文件通常都是少于64KB，仅在一个段内运行。如果有必要修改的子模块可以直接用汇编指令修改，则把修改后的程序用W命令把修改好的程序存盘。

EXE文件其首地址是由文件的重定位信息来定位的。因为这类文件的结构是各段地址不同，例如，代码段和数据段不在一个段内，因此，不允许对EXE文件直接用W命令存盘。要想修改EXE文件，必须先把文件名改换为非EXE文件。如：

C>REN dBASE.EXE dBASE

然后再调入内存进行修改，随后存盘。但是，对于大多数改名后的非EXE文件，在内存中的逻辑地址却比原EXE文件的地址高出300H个字节，这是由于原EXE文件的文件头前缀所致。所以，在EXE文件中跟踪到的地址，在改为非EXE文件后，其地址将相对增加300H。如果原地址为153A，改名后的地址则在183A。当修改程序结束，用W命令存到磁盘上以后，再把文件改为带EXE后缀的文件名。

(2) 根据现象判断。这是观察一些功能模块的指令，大体确定问题的性质和关键程序的范围。利用一些经验，先搜索一些与显示有关的代码。例如以下的代码和指令：

代码	指令
BA D8 03	MOV DX,03D8
B8 00 B3	MOV AX,B300
CD 10	INT 10

可以用 S 命令分段查找

```
- S100 F000 CD 10
```

如果能够搜索到类似的指令代码，说明这一段程序与屏幕显示有关，然后在这个地址范围内使用单步和反汇编命令，观察这段程序走向和执行情况，记录有关地址数据。通过栈回溯和指针设置，即可把有关的子程序定位。

(3) 字符处理查找。对于字符信息处理的子程序，也可以借助于一些启发性的指令，在程序中搜索。这些指令往往是在词法分析模块中。例如

代 码	指 令
24 7F	AND AL, 7F
3C 7A	CMP AL, 7A

用 S 命令搜索到后，要对该程序段作具体分析，有些程序不一定与字符处理有关，这是值得注意的。

(4) 输入模块的跟踪。对于绝大多数软件来说，接收键盘输入的信息，均是调用 BIOS 的中断 16H服务程序来实现的。可以用 S 命令搜索其二进制机器码 CD16来定位它的地址。如果在INT16指令之前有一条 XOR AH,AH 或者 MOV AH,00 的指令，可以把这样的指令改为 INT3 或者用 SDebug 调试程序在此指令的地址处设置一个断点。例如：

```
1765:130A    XOR AH,AH
1765:130C    INT 16
-BP 1765:130A
-G
```

然后用 G命令开始执行程序。当程序运行到断点地址1765:130A处时，便自动停止，对于 INT 16H 指令，需要用过程命令 P 进

入，或者用 G 命令执行到 INT 16H 的下一条指令的地址处。这时，系统将调用 BIOS 的中断 16H 功能，等待键盘输入信息。实际上，此时，整个软件及调试系统的控制权均已掌握在调试者的手中。可以有目的地输入一个字符，用单步 T 命令观察程序的走向。在此过程中，对于输入的信息，通常有以下几种跟踪要领：

第一种方式是先输入一个 ASCII 字符或都是功能控制键信息，用 T 命令观察一段程序是如何运行的，各个寄存器和内存单元有何变化，输入这一信息又是被怎样接收和处理的。使程序正常返回。

第二种方式是输入一个高 8 位为“1”的扩展 ASCII 代码，即十六进制大于 A0H 的数据，可用 Alt＋〈数字键〉输入，这表明输入了一个汉字内码。再用单步命令跟踪一段程序，观察各寄存器和内存单元的变化，看看程序的走向与输入标准 ASCII 码有何不同，在什么地方分支，记录下程序分支的地址。在此范围内，就可找到需要汉化的关键所在。

第三种方式就是输入一个完整的语句或者建立的一个变量。例如 dBASE、FoxBase 一类软件。可以先用西文方式进行跟踪，再用汉字作为变量输入，使程序进入语词分析模块。逐步跟踪到中西文语词处理的分支地址处，使所要汉化修改的程序段：偏移地址定位。

（5）逐渐逼近法。这是一种最常用的跟踪方法。其步骤是：利用 Debug 中的 G＜地址＞命令，使程序执行到最近的某个转移指令处，用单步 T 命令跟踪一步，让程序正常转移。如果遇到 CALL＜地址＞或 LOOP ＜地址＞这类指令时，可用 P 命令执行一次。P 命令的功能是执行完 CALL 后面所调用的那个子程序后，返回到紧接着 CALL 下面的一条指令地址处。在 LOOP 指令处使用 P 命令时，将使程序在其循环范围内连续执行到 CX＝0 为止，这样便加快了跟踪的速度。

如果在CALL指令处，执行了P命令之后，程序的控制权不再返回Debug提示符，那么，就把该地址记录下来，然后用Q命令返回DOS系统提示符。再重新把该文件调入内存，这次用G命令直接执行到刚才记录下来的地址处。不过，这一次对CALL指令，不再用P命令直接执行跳过，而是用单步命令跟入到CALL所调用的那个子程序中去，继续用上述的方法反复跟踪，一个层次一个层次地深入，并且随时观察各个寄存器的变化和程序所开辟的数据缓冲区内的数据处理情况。因为，对于要处理的数据信息，首先被程序送入数据缓冲区中，尔后的各种加工处理，都是直接与数据缓冲区打交道。例如dBASEⅢ字符处理模块，在跟踪的过程中，可以观察到输入的信息、要显示的字符串和控制码在缓冲区中被加工处理的变化情况。再如Lotus 1-2-3、Supercalc等一些表处理软件，其工作表中数据移动的情况，均可以通过观察程序执行中开辟的数据缓冲区来判定字符识别的规范。

因而，有必要记录下一些重要的地址和信息。有时还要及时地记录一些反汇编源程序和某地址处寄存器和标志位的状态。所以，在跟踪过程中，最好使打印机联机，以便随时打印出指定的程序段文本。

实践证明，用逐步求精来渐渐逼近目标是一种行之有效的跟踪方法。它不仅可以准确迅速地找到需要汉化的关键所在，而且对了解程序运行的情况和分析软件的结构亦大有裨益。

(6) 人为指定程序的走向。这是指，对于某个被调试的程序，使其按照调试者的意图运行。这是比较灵活实用的跟踪方法，对于局部的模块分析非常有用。例如，在程序中往往遇到一些如下所示的测试、比较或转跳等指令：

```
CMP AX, 007A
TEST BYTE PTR [145A],80
JNZ 145F
```

可用R命令来改变寄存器和IP指针，用E命令改写[145A]内存单元的数据，使得程序按照指定的步骤执行下去。例如

```
-RAX
:55
-E145A
DS:145A 30 80
```

在实际的汉化过程中，根据环境和具体的软件来选择一种或几种相结合的跟踪方法，以找到问题的关键作为最终目标。

还可以根据自己的经验，以各种方式在程序中插入一些小程序，或者自己有目的地编一些调试小程序，以中断方式驻入内存，在所要跟踪的软件某一指定的地址处，用中断的方式进行截取，导致程序在执行某段代码时进入预先设置的陷井，以此来控制和观察程序执行的情况和结果。

总之，一些技巧运用得适当与否，取决于对中西文操作系统的掌握情况和对程序设计中一些规范的熟悉程度。软件的跟踪和分析，是一项十分枯燥又易疲乏的工作，因比，要有耐心，切勿急躁，否则，就会前功尽弃。只有在汉化过程中慢慢地积累经验，无论要汉化的软件多么大，多么复杂，都能够迅速而有条不紊地找到问题的关键所在。

4.4 屏幕显示汉字化

无论哪一种自成系统的软件包，当它进入系统显示信息之前，都要进行一次清除屏幕处理，然后再显示有关的信息或操作菜单等。其作用，一是为了清除屏幕原有的、不需要的旧信息，二是为了进入该软件所指定的显示方式。这虽然是一个简单的清屏操作，但不同的软件却采用不同的方法来实现。有些清屏方法，对于在汉字操作系统环境下的执行，有着极大的影响。由于屏幕是计算机与人之间直接的界面，因比，汉化工作必须从这里着手。

汉字操作系统是在图形显示方式下工作的。它把所需要显示的汉字和 ASCII 字符的点阵信息从字模库中取出后，按其点阵

排列，送入视频缓冲区指定的地址偏移量显示。因此，无论何种软件，要想在汉字系统下使用，必须保障该软件进入系统后的处于图形显示方式。就是说 CRT 的模式不能够改变，这也是汉字显示的前提。

4.4.1 屏幕初始化

有些软件，一旦被调入内存运行，首先对显示器进行初始化处理。任何显示控制器的初始化功能都包含视频刷新操作，即清除屏幕功能。所以，可以说是一举两得。一般，初始化的目的是把屏幕的显示方式设置为该软件指定的显示环境。例如，下列指令即为屏幕初始化指令：

```
MOV  AX,0003
INT  10H
```

这一指令的意思是指通过中断 10H 调用，把当前的显示方式设置为 80×25 彩色字符方式。

常规的屏幕显示方式有如下几种：

AL = 00	40×25	黑白/字符	CGA
= 01	40×25	彩色/字符	CGA
= 02	80×25	黑白/字符	CGA
= 03	80×25	彩色/字符	CGA
= 04	320×200	单色/图形	CGA
= 05	320×200	彩色/图形	CGA
= 06	640×200	单色/图形	CGA
= 07	80×25	单色/字符	MDA
= 10H	640×350	彩色/图形	EGA
= 12H	640×480	彩色/图形	VGA
= 42H	640×400	彩色/图形	Color400

当然，对于不同的显示器卡，尤其是 EGA/VGA 等增强型的显示卡，有的具有模拟跟踪功能，分别可变换几十种显示模式，要视具体环境而作具体分析。

类似上述设置显示方式的指令，可以根据系统配置的CC-DOS环境，选择修改AX寄存器的初值。譬如，若是CGA显示卡，可置AX=0006，发出中断10H指令。若是EGA显示卡，则可设置AX=0010H等，使系统初始化后仍保留图形的显示状态。

但是，直接修改初始化功能的入口参数的方法，无论系统的当前环境是在文本方式还是在图形方式，都将置为图形方式，这对于在各种环境下的兼容使用，就不太灵活方便。对于屏幕初始化模块，其汉化和调整方法有如下几种：

(1) 插入取方式指令

为了得到较好的中西文兼容性，可以在设置屏幕初始化的指令前面，插入一个取当前CRT方式的功能调用。这意思是指，取出当前显示方式参数，并且仍用该参数进行初始化，就保障了清屏后屏幕模式不变，如下所示。

```
MOV  AH,0FH
INT  10H          ; 屏幕模式读入AL
XOR  AH,AH
INT  10H          ; 初初化屏幕
```

(2) 利用滚屏功能

若软件进入系统纯粹是为了刷屏的话，为了保持当前屏幕模式不变，可以利用CC-BIOS的上滚或下滚功能进行清屏操作。把原初始化程序修改为以下的程序即可。

```
MOV   AH,06H     ; 上滚功能号
MOV   AL,00      ; 表示清窗口
MOV   CX,0000    ; 左上角0行0列
MOV   DX,184FH   ; 右下角25行80列
XOR   BX,BX
INT   10H        ; 刷屏
```

其中DX寄存器的值为屏幕窗口的右下角坐标参数。该值可以根据CC-BIOS的环境(如10行、20行和25行等)设置不同的

行列值。

(3) 去除屏幕初始化指令

倘若该指令仅仅为了屏幕初始化后把显示器设置为80×25的文本方式，并在此之后，还有专门的刷屏功能被调用的话，则可以去掉这条指令。即用A命令把INT 10H指令用两个NOP（空操作）来代替，使得程序运行至此，不发生初始化的操作，以保持当前CRT状态。在分析程序时可以发现，有为数不少的软件是属此种情况。

4.4.2 中/西文刷屏

有些软件，为了提高显示速度，其刷屏的方式未采用BIOS的INT 10H功能调用，而是用直接往视频缓冲区传送满屏幕的空格字符及其黑色属性。如以下程序所示：

```
        PUSH  DS
        MOV   AX,40H
        MOV   DS,AX          ;BIOS数据段
        MOV   AX,[0010]      ;取设备标志字
        AND   AX,0030H       ;屏蔽多余位
        CMP   AX,0030H       ;是单色显示卡?
        JNZ   COLOR
        MOV   AX,B000        ;是
        JMP   R1
COLOR:
        MOV   AX,B800        ;彩色卡
R1:     MOV   ES,AX          ;ES=视频段地址
        XOR   DI,DI          ;偏移指针
        XOR   AH,AH          ;黑色属性
        MOV   AL,20H         ;ASCII空格符
        MOV   CX,07D0        ;CX=2000
        CLD
        REPZ
        STOSW                ;刷屏幕
        POP   DS
```

上述程序虽然刷屏速度很快，但是，若在汉字图形方式执行时，屏幕上将产生一片“麻点”。这是因为在图形方式下，显示控制器把空格字符20H以二进制00100000点阵方式送往视频缓冲区，这显然不能清屏幕。再者，诸如此类的清屏子程序大多数仅仅判别是CGA显示卡还是MDA显示卡。如果是EGA/VGA显示卡的话，系统在图形方式下，其视频缓冲区的段地址无法得到，因此，信息无法显示，清屏也不起作用。

对于这类清屏幕程序的汉化，采用前面所述的取当前屏幕方式或者屏幕上滚下滚法进行修改均可。这样做不但方法简单，而且修改的程序地址范围也足够，不必增加新的程序模块。假如上述程序中CX计数器不是常量，而是变量，也就是说，不是刷整幅屏幕，而只是清局部窗口的话。那么，可以由DI寄存器的偏移值进行运算，求出所清屏幕窗口的左上角和右下角坐标，用上滚下滚方式可以做到有效的刷屏或清除窗口。

4.4.3 批量数据显示的汉化

有些西文软件，不是直接把数据缓冲区内的显示数据一个一个地送往屏幕，而是在内存中开辟一个模拟屏幕显示区。该区与视频缓冲区具有相同的结构和容量，程序把要显示的数据，先在模拟区内加工好。通常，都是以整幅屏幕信息进行加工和处理，最后把模拟区的数据加工为与视频缓冲区一模一样的格式，再以视频缓冲区首址为目标，作为块（批量）传送，其显示速度非常之快，整幅屏幕的信息只在一瞬间就显示完毕。

例如，下列美国Peter公司的Norton Utilities磁盘工具软件的显示部分，就是采用这类块传送的方式来显示信息的。

```
0CEB:8B8E PUSH  BP
0CEB:8B8F MOV   BP,SP
0CEB:8B91 MOV   ES:[BP+04]  ; 视屏段址 B800
0CEB:8B94 MOV   DI,[BP+06]  ; 显示偏移地址
0CEB:8B97 MOV   SI,[BP+08]  ; 数据区首址
```

```
0CEB:8B9A   MOV     CX,[BP+0A]；显示字节数
0CEB:8B9D   CLD
0CEB:8B9E   REPZ
0CEB:8B9F   MOVSB                ；把数据区的数
0CEB:8BA0   POP     BP           ；据传送到屏幕
0CEB:8BA1   RET
```

程序中的各项参数都是通过堆栈传送的，利用 Debug 程序，通过跟踪分析，搞清了各级堆栈带入该显示模块的参数内容和类型之后，即可作出汉化的修改方法及程序模块。经过汉化的程序如下：

```
0CEB:8B8E   PUSH  BP
0CEB:8B8F   MOV   BP,SP
0CEB:8B91   JMP   BCC0          ;转汉化程序
0CEB:8BA0   POP   BP
0CEB:8BA1   RET
    ⋮        ⋮
0CEB:BCC0   PUSH  AX
0CEB:BCC1   PUSH  DX
0CEB:BCC2   PUSH  BX
0CEB:BCC3   MOV   DX,[BP+06]    ;取当前行列值
0CEB:BCC6   MOV   AH,02         ;光标定位功能
0CEB:BCC8   XOR   BH,BH
0CEB:BCCA   INT   10
0CEB:BCCC   MOV   BX,[BP+08]    ;数据区首址
0CEB:BCCF   MOV   CX,[BP+0A]
0CEB:BCD2   SAR   CX,1          ;字节数除2
0CEB:BCD4   MOV   AX,[BX]       ;取一个字符
0CEB:BCD6   PUSH  BX
0CEB:BCD7   MOV   BL,AH         ;显示属性
0CEB:BCD9   MOV   AH,0E         ;TTY显示功能
0CEB:BCDB   XOR   BH,BH
0CEB:BCDD   INT   10            ;调用BIOS
0CEB:BCDF   POP   BX
0CEB:BCE0   INC   BX
0CEB:BCE1   INC   BX            ;数据区地址递增
0CEB:BCE2   LOOP  BCD4          ;显示CX个字节
```

```
0CEB:BCE4  POP   BX
0CEB:BCE5  POP   DX
0CEB:BCE6  POP   AX
0CEB:BCE7  JMP   8BA0
```

这个程序仅给出一个汉化的思想，所以用CC-BIOS中断10H的0EH功能，即TTY方式显示字符。如果不是整幅屏幕数据的批传送，那么，DX的光标位置不能直接取出，而是由[BP+06]的栈参数，把偏移地址换算为光标位置。

由于原程序的地址范围容纳不下汉化过的程序，因此，把汉化的程序段加到程序末端，由跳转指令联接，如何增添新的程序，后面有一节专门论述。

4.4.4 屏幕显示模式的定义

有些表处理软件，在表格数据编辑时，工作在字符显示方式下。若要用图形来表达电子表处理的数据的统计结果时，显示方式将从字符切换为图形方式。除了用屏幕初始化功能外，还采用了直接设置显示器模式方法。换句话说，通过端口指令，直接向M6845显示控制器模式寄存器端口发送参数，以达到改变屏幕状态的目的。其指令如下：

```
MOV     AL,29H      ; 模式参数
MOV     DX,03D8     ; 端口地址
OUT     DX,AL       ; 设置模式
```

3D8是M6845显示控制器中的一个模式寄存器端口地址，该寄存器的各位意义如表4-1所示。从表中可以看出，若模式参数为29H，那么，说明对3D8寄存器的b_0、b_3和b_5同时置"1"，这种状态把显示模式设置为80×25的彩色字符状态，并且允许视频信号选通和字符闪烁。

在汉字系统的环境下，一旦执行了上述几条指令，屏幕将出现一片混乱。这是因为显示模式虽已改变，但行扫描和帧扫描的时间、频宽等参数没有改变，即数据寄存器的内容还保持图形参

3D8 模式寄存器各位意义　　表 4-1

位　号	=1 的 状 态	=0 的 状 态
b_0	80×25模式	40×25模式
b_1	320×200模式	字母/数字模式
b_2	选为黑白模式	选为彩色模式
b_3	允许视频信号	不允许视频信号
b_4	640×200模式	非高分辨率
b_5	允许字符闪烁	不[illegible]
b_6b_7	不用	

数，导致无法正常显示。因此，为了达到汉字化或中西文兼容的目的，有必要找这一类指令，一是可以用相应的模式参数修改寄存器值，二是用 NOP 或 JMP 等指令跳过这段程序不用。

4.5 全屏幕功能的汉化

具有全屏幕编辑功能的软件很多，如 Turbo 系列的集成软件、dBASE、PE Ⅱ 和 FoxBase 等。其全屏幕功能用于程序文本的编辑或数据的编辑。由于全屏幕的窗口是面向用户的，所以，必须进行合理的改造，使其可以在 CC-DOS 汉学系统下正常地运行和有效地对汉字信息进行处理。

4.5.1 西文字符显示原理

前面已讲过，西文软件为了提高显示速度。往往直接向视频缓冲区传送数据。绝大多数软件的显示程序是采用从 DS:SI 至 ES:DI 两个地址之间的串传送。其原理并不十分复杂，如图4-5所示。但是具体的程序实现方法各不相同。以下介绍几种比较典型也是最常碰到的字符显示方式。

(1) 传送数据到 AL = 0

这种方法通常用在传送不定长的显示字符串程序中。它利用

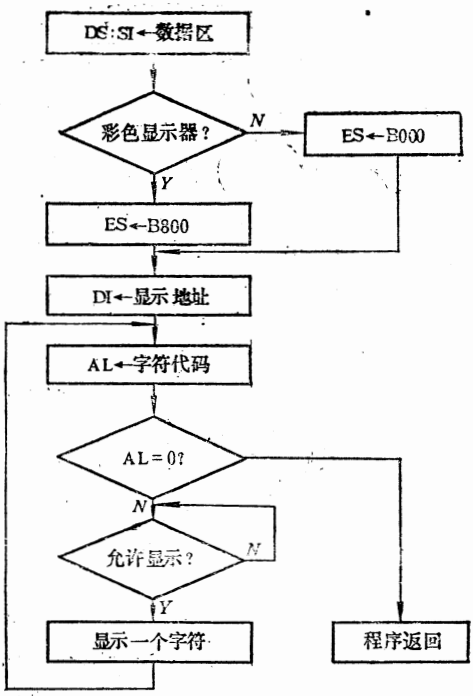


图 4-5 西文字符显示流程

“或”指令的逻辑判别，对作为传送数据用的 AL 的寄存器进行测试。若 AL 取到的字符代码为 0（NULL）时，则立即停止传送而退出，程序如下：

```
        CLD
        MOV     SI,2000      ;数据区首址
        MOV     DI,[1004]    ;视频偏移地址
R1:     LODSW                ;DS:[SI]的内容送 AX
        OR      AL,AL        ;字符等于 0 吗?
        JZ      R2           ;是，结束
        STOSW                ;AX 内容送 ES:[DI]
        LOOP    R1           ;循环显示
        ⋮
R2:     RET                  ;返回
```

(2) 传送 CX 个字符

这种显示传送的方法也称作块传送。其显示 CX 字符后即可返回，程序如下：

```
        MOV     SI,2000      ;数据区首地址
        MOV     CX,[1004]    ;需要传送的字节数
        MOV     DI,[1002]    ;显示区偏移地址
        MOV     AX,[1000]
        MOV     DS,AX
        MOV     AX,B800
        MOV     ES,AX        ;ES=视频段地址
        CLD
        REPZ
        MOVSW                ;传送字符/属性
        ⋮
        RET                  ;返回
```

(3) 测状态显示

这种方法常用于用不同彩色显示不同长度的变量。即每从数据缓冲区取一字符后，首先测试一下 CRT 是否选通，当允许显示时，就送字符显示，否则，就不断地测试状态。这样，可以保障显示信息时，屏幕不会出现异常。程序如下：

```
        MOV     SI,2000      ;数据区首址
        LEA     DI,[BP-2]    ;视频偏移地址
        MOV     DX,03DA      ;状态寄存器口
R1:     LODSB                ;AL取一个字节
        MOV     BX,AX        ;保存
R2:     IN      AL,DX        ;读显示器状态
        TEST    AL,01        ;测状态字节
        JNZ     R2
        CLI
        MOV     AX,BX
        STOSW                ;显示字符
        SIT
        LOOP    R1           ;循环显示
        ⋮
        RET
```

上述几个典型的西文软件显示方式，均是根据数据块的传送原理显示的。即把DS:SI指向将要显示的源数据缓冲区首地址，ES:DI 指向 CRT 视频缓冲区目的首地址，通过传送指令来实现快速显示。视频缓冲区的地址 ES:DI 虽然是从某一个存贮单元取得的，实际上是根据当前屏幕上的光标位置换算出来的参数事先置入了某一个单元。例如，PCTOOLS V4.11 软件中视频缓冲区偏移地址的运算方法模块程序如下：

```
CS:0A01   PUSH   BX
CS:0A02   MOV    BX,AX              ; 取当前光标行列值
CS:0A04   XOR    AX,AX
CS:0A06   MOV    AL,BH              ; AL=当前行
CS:0A08   MUL    BYTE PTR[0532]     ; 乘上80(一行80列)
CS:0A0D   MOV    BH,0
CS:0A0F   ADD    AX,BX              ; 加上当前列数
CS:0A11   SHL    AX,1               ; 乘以2（包括属性）
CS:0A13   MOV    DI,AX              ; DI=视频偏移地址
CS:0A15   MOV    ES,SS:[034A]       ; 取视频段地址
CS:0A1A   POP    BX
CS:0A1B   RET
```

程序在执行显示功能之前，先要调用该运算子程序，在 DI 寄存器中得到要显示信息的首地址后返回。在 SI 和 DI 之间数据传送的过程中，它们的指针地址的递增或递减是由方向标志决定的，而需要传送的字节数则是由 CX 寄存器值的大小决定。

4.5.2 调用CC-BIOS 功能的汉化法

为使西文软件能在汉字操作系统下正常地显示汉字信息，应该充分利用 CC-BIOS 的功能调用，对西文软件进行修改。也就是说，编制一段以 CC-BIOS中断调用的显示程序来代换原西文软件的显示模块。

这种方法，虽然其显示速度受到一定的影响，但是，经汉化

过的软件，其硬件兼容性很强。对于微机上配置的 CGA、MDA 或 EGA 等不同的显示卡，只要系统进入了 CC-BIOS 汉字系统，汉化后的软件均可使用，不受硬件设备和系统的限制。目前国内大部分汉化软件，都是采用这种方法实现的。

在 CC-BIOS 汉字系统的 10H 类中断服务程序中，如 2 号功能（光标定位）、9 号功能（显示字符和属性）和 0EH 号功能（TTY 方式显示字符）是汉化软件时最常用的几个功能调用。9 号和 0EH 号功能虽然都是显示字符，但需要视具体的实现要求来选择使用。这是因为，9 号功能是在当前光标位置显示一个字符及其属性后，光标位置保持不变，要连续显示信息，必须在每调用一次后，光标要重新定位处理才行。而 0EH 号功能就类似电传打字机一样，在当前光标处显示一个字符后，光标自动前移，且遇到回车换行码时，亦将自动处理。不过，早期的 CC-DOS 版本中，该功能不支持属性的显示（即字符的色彩，反示等）。

无论选择哪种显示字符的功能调用，在显示第一个字符之前，都必须先在屏幕上设置当前光标的位置，在此称为首标定位。原西文软件在显示之前，先通过运算产生出视频缓冲区的偏移地址。虽然这个偏移地址与光标位置之间有一定关系，但是其值并不相同。

在汉化时，如果有可能，就用 JMP 指令跳过偏移地址的运算程序，直接把光标的行列值分别送入 DH 和 DL 寄存器，用 2 号功能调用进行首光标定位。若这段运算程序与其他许多有关程序模块有着不可分割的联系而不易跳过的话。那么，可以通过视频缓冲区的偏移地址 DI 的值逆运算产生当前首光标值。其算法如图 4-6 所示。

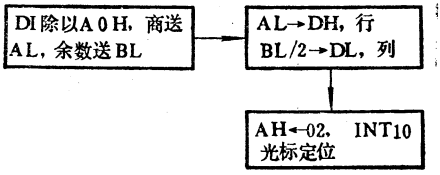


图 4-6　光标定位算法

有了算法，就不难实现首光标定位，下面是用汇编语言编写一段程序，是由 DI 的偏移地址运算出首光标位置，再通过调用实现当前屏幕光标的定位。

```
MOV   AX,DI      ;取视频偏移地址
MOV   BX,00A0
IDIV  BX         ;AX除以A0
SHR   DX,1       ;列值除2
MOV   BX,DX
MOV   DH,AL      ;当前行
MOV   DL,BL      ;当前列
XOR   BX,BX
MOV   AH,02      ;光标定位
INT   10
```

下面例举一个调用 CC-BIOS 功能的汉化软件的范例。该软件是一个具有数据处理和统计图形的软件—dGRAPH。其西文显示模块如下：

```
0CFB:78DD  PUSH  BP            ;保护栈指针
0CFB:78DE  MOV   BP,SP
0CFB:78E0  PUSH  ES
0CFB:78E1  XOR   AX,AX
0CFB:78E3  PUSH  AX
0CFB:78E4  POP   ES
0CFB:78E5  ES:
0CFB:78E6  MOV   DL,[0410]     ;取设备字节
0CFB:78EA  AND   DL,30         ;屏蔽无关位
0CFB:78ED  CMP   DL,30
0CFB:78F0  JZ    78F7          ;单色CRT
0CFB:78F2  MOV   AX,B800       ;彩色CRT段址
0CFB:78F5  JMP   78FA
0CFB:78F7  MOV   AX,B000       ;单色CRT段址
0CFB:78FA  PUSH  AX
0CFB:78FB  POP   ES            ;ES=视频段
0CFB:78FC  MOV   AX,0000
0CFB:78FF  MOV   DX,0050
0CFB:7902  MOV   CX,[BP+04]    ;取出行值
```

```
0CFB:7905  ADD     AX,CX          ;光标行
0CFB:7907  DEC     DX
0CFB:7908  JNZ     7905
0CFB:790A  SHL     AX,1
0CFB:790C  MOV     BX,[BP+06]     ;取出列值
0CFB:790F  SHL     BX,1           ;光标列
0CFB:7911  ADD     AX,BX
0CFB:7913  MOV     DI,AX          ;DI=视频偏移值
0CFB:7915  MOV     CX,[BP+08]     ;要显示的字符数
0CFB:7918  MOV     SI,[BP+0A]     ;数据区首址
0CFB:791B  MOV     BX,0000
0CFB:791E  MOVSB                  ;传送显示
0CFB:791F  INC     DI             ;地址递增
0CFB:7920  CMP     BL,[SI]
0CFB:7922  JZ      792A           ;BL=0结束
0CFB:7924  DEC     CX
0CFB:7925  CMP     CX,+00
0CFB:7928  JNZ     791E           ;显示CX个字节
0CFB:792A  NOP
0CFB:792B  POP     ES
0CFB:792C  MOV     SP,BP
0CFB:792E  POP     BP
0CFB:792F  RET
```

该软件可以把dBASE等系统建立的数据作出各种统计图形，因此，受到许多用户的欢迎，只可借不能处理汉字。根据对上述西文显示模块的分析，可用 CC-BIOS 的功能调用进行汉化，其结果如下列程序所示。请读者注意每条指令后面的注释，就不难理解程序的实现方法。

```
0CFB:78DD  PUSH    BP
0CFB:78DE  MOV     BP,SP
0CFB:78E0  XOR     BX,BX
0CFB:78E3  MOV     DH,[BP+04]     ;光标行值
0CFB:78E6  MOV     DL,[BP+06]     ;光标列值
0CFB:78E9  MOV     AH,02          ;光标定位
```

```
0CFB:78EB   INT     10              ;BIOS功能
0CFB:78ED   MOV     CX,[BP+03]      ;显示字节数
0CFB:78F0   MOV     SI,[BP+0A]      ;数据区首址
0CFB:78F3   LODSB                   ;取一字符
0CFB:78F4   OR      AL,AL
0CFB:78F6   JZ      792C            ;AL=0结束
0CFB:78F8   MOV     AH,0E           ;TTY显示功能
0CFB:78FA   INT     10
0CFB:78FC   LOOP    78F3            ;显示CX个字节
0CFB:78FE   JMP     792C            ;返回
      ⋮       ⋮
0CFB:792C   MOV     SP,BP
0CFB:792E   POP     BP
0CFB:792F   RET
```

4.5.3 直接显示汉化法

不通过中断调用的服务，直接把显示数据送往视频缓冲区的方法，将会大大提高显示速度。但是这种方法对原西文软件的修改量比较大，需要增加许多指令，并且兼容性不太好。在此仅提出一种方法，供读者参考。

以 CGA 10 行的 CC-DOS 系统为环境背景，把要显示的信息直接送入 B800 段的视频区。其功能流程如图 4-7 所示。

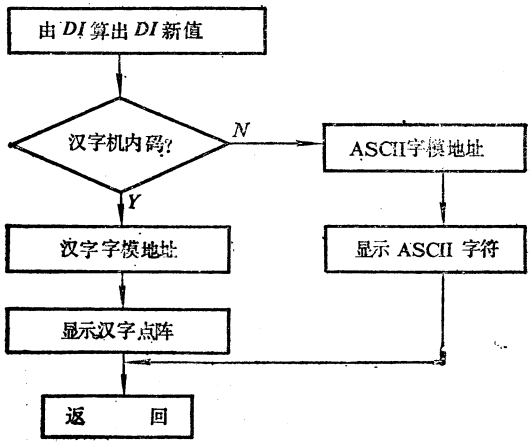


图 4-7　直接显示点阵流程

由于汉字是在640×200分辨率的图形方式下显示的，因此，其视频缓冲区偏移地址与在字符方式下的地址不同。如果已得出当前光标位置的行列(n,m)值，可通过下列公式算出对应的偏移地址：

$$Offset = 2D0H \times n + m$$

在CC-BIOS系统下，ASCII字符占一个字节，而汉字占用两个字节。从显示形状上来看，汉字比字符宽一倍。因此，显示汉字与ASCII码要有不同的处理方式。可用判别代码是否大于A0H来确定是ASCII字符还是汉字。在图形方式下，即使要显示的是ASCII字符，也必须由图形点阵的方式显示，其点阵字符的首地址由中断1FH的向量指针给出，根据8×8或8×14点阵等形式取得不同字节。

在显示汉字时，需要通过汉字机内码转换为区位码，再由区位码算出汉字点阵字模的段首址，取出字模后，最后向视频地址DI指针处送显示信息。程序处理的方法如图4-8所示。

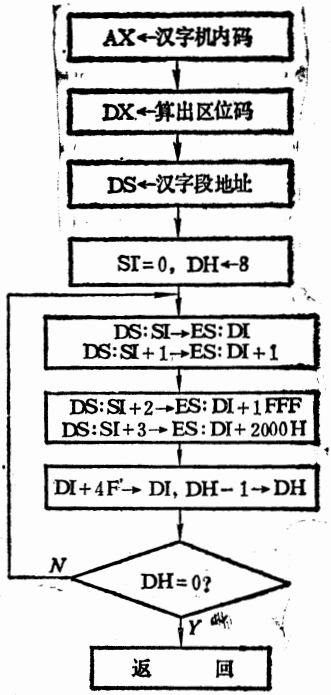


图 4-8 汉字直接显示

但是，在汉字显示过程中，当汉字出现在屏幕边缘时，有可能出现只显示半个汉字而另半个汉字显示在下一行的首端的情况。若屏幕分为左右两个窗口时，半个汉字超出窗口边缘进入另一窗口。所以，在显示汉字之前，需判断其是否到了边缘，若是，就用一个空格来代替，在进行屏幕左右移动时再显示出来。

4.5.4 下拉式菜单的汉化

下拉式菜单的软件的应用愈

来愈广泛。因为它层次分明、用户界面好，许多较大型的软件中的辅助功能都采用这种方法。

下拉式(POP DOWN)显示方式，实际上是一种允许相互覆盖的窗口管理系统。屏幕上可允许几个层次的窗口信息相互覆盖。如图4-9所示。

每打开一个窗口，则将覆盖旧窗口的信息，作为新的一个层面来显示信息。如在图4-9中，*C*层覆盖*B*层，*B*层覆盖*A*层等。当退出一个窗口时，系统将恢复被该窗口信息覆盖的窗口信息。

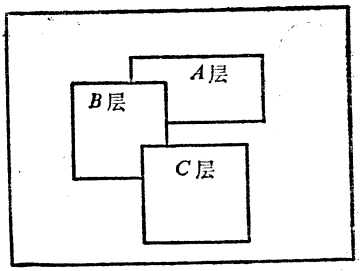


图 4-9 分层窗口

实际上，每个窗口都带有一个缓冲区，它的目的是记住窗口所显示的内容，以便将来用它恢复窗口的内容。它还带有光标和窗口尺寸等参数，记住下次输出信息的位置和信息量。

例如FoxBase V2.10和dBASE Ⅳ等软件，就是采用了这类下拉式菜单方式，作为它辅助编程和管理的功能。FoxBase V2.10采用的是缓冲区存贮窗口的方法，而dBASE Ⅳ则是由缓冲区与TOP View方式相结合实现的。

TOP View功能调用有两个子功能，也是通过INT 10H的服务模块实现的。其功能的主要作用是赋予过程--个投影缓冲区，作为从程序接收显示输出，当请求修改时，投影缓冲区的一部分或全部内容被复制到当前进程的屏幕窗口。此功能提供的是直接修改显示缓冲的功能。其调用方法如下：

① 取任务显示缓冲区地址

输入参数：[AH]=FEH ;功能号

[ES:DI]=指定显示缓冲区的段：偏移地址

输出参数：[ES:DI]=当前进程的实际显示缓冲区的段：偏移地址

② 复制修改投影显示缓冲区

输入参数：[AH]=FFH　　；功能号
[CX]=已被修改的字符数
[ES]=投影显示缓冲区段
[DI]=在投影显示缓冲区已被修改的首字符的偏移值。

遗憾的是，这两个功能不支持图形显示，这给汉化工作带来许多困难。因为要保存被覆盖的信息，必须在内存中开辟一块相应的缓冲区。一幅图形的信息量要比相应的字符代码信息量大几个数量级，因此，不可能开辟如此庞大的内存缓冲区。

一种汉化方法，是根据要覆盖信息在屏幕上的首地址，想办法保存其ASCII代码信息，当高层窗口退出时，调用CC-BIOS功能显示恢复，其速度将会降低。

另外一种汉化方法，是把要覆盖的屏幕信息，换算出左上角和右下角的坐标值，用文件的方式保存到磁盘上去。文件头是坐标和容量参数，其数据部分为对应的屏幕映像信息。当高层窗口退出时，再从磁盘读入内存，根据参数量，再回显到原来的位置。由于程序长而复杂，在此仅说明一些原理和方法，具体程序不再列出。

4.6 输入词法逻辑的汉化

对于西文软件的汉字输入，尤其是建立汉字变量名等功能的汉化并不复杂。其要领是在INT 16H键盘输入控制处，设置适当的断点，跟踪到键盘控制的子程序处，分别用ASCII字符和汉字代码输入，来找出其不同路程之处。把该程序段中识别ASCII代码的范围加以扩展，另外再修改一些标识单元即可。

例如FoxBaseV 1.1中设置字段变量名的输入代码控制部分。在未汉化之前，其输入的字符代码限制在大于等于61H(a)、小于7AH(z)的范围内。由于汉字机内码从A0H到FEH，显示在该程序识别的范围之外，所以，它不予受理。如下列程序所示：

```
CS:08CC    PUSH   BP
CS:08CD    MOV    BP,SP
  :          :
CS:08FA    CMP    BYTE PTR [BP+06],61
CS:08FE    JB     0911
CS:0900    CMP    BYTE PTR[BP+06],7A
CS:0904    JA     0911
CS:0906    MOV    AL,[BP+06]
CS:0909    SUB    AH,AH
CS:090B    SUB    AX,[06DC]
CS:090F    JMP    0913
CS:0911    XOR    AX,AX
CS:0913    MOV    SP,BP
CS:0915    POP    BP
CS:0916    RETF
```

在上述程序 CS:0900 地址处，是一条比较指令。汉化时，只要把它与 7A 的比较改为与 FF 比较，或者把 CS:0904 的转跳指令用 NOP 指令代换即可。当然，还要修改相应的对照表的信息，把 0 改为 1，以确保汉字变量名可以被程序识别和接受。

还有一种类型的输入码判别程序，不但限制扩展 ASCII 码或汉字机内码的处理，而且把所接受的输入码一律作变换处理。即把输入的命令和程序语句中的大写字符变换为小写字符，小写字符变换为大写字符。例如，AL 寄存器中是键盘输入的代码，用 ADD AL，20H 的指令把大写字符转换为小写字符，用 AND AL，DFH 或 SUB AL，20H 把小写字符转换为大写字符。

像这一类的判别程序，汉化时就不能单纯地扩充其输入识别的范围。要考虑到，如果输入码为汉字机内码，就必须跳过大小写字符转换的程序。在程序返回时，需要在其返回标志寄存器或单元中，说明该输入码有效。因为汉字机内码被以上的指令处理后，就不会是原输入的汉字机内码或已不是原输入的汉字机内码了，所以，必定会引起执行出错，这一点必须注意，这种情况的汉化将在 dBASE-ⅢPLUS 软件的汉化一节中讲述。

4.7 字符处理及其定义域的扩展

西文软件之所以不接受汉字信息，主要是因为在输入输出处理途经上有一些“滤除”操作指令对数据流进行“过滤”，使得汉字机内码无法正常通过。

4.7.1 制表符的汉化

通常，在显示模块中无此滤除操作，因为许多软件在屏幕上显示的制表符多数是扩展的 ASCII 代码。这些制表符代码在汉化软件中还需要修改，不然，在汉字系统下，若有两个制表符在一起显示，CC-BIOS 就会把这两个扩展 ASCII 码作为其对应的汉字显示出来。一般的汉化软件把这些扩展 ASCII 代码的制表符改为标准 ASCII 码所对应的虚线或星号等。也可以改为标准汉字库中对应的汉字制表符。

修改扩展 ASCII 码的制表符，要视具体的软件结构而定。有些软件把整个程序所要使用的制表符都放在某一个模块中固定的存贮区域中。无论程序执行到哪里，只要是需要制表符代码，其程序都会自动地到该区域中调出后送去显示。像 dBASE-Ⅲ Plus 或 dBASE Ⅳ 都属于这一类操作形式。譬如，dBASE-Ⅲ Plus 软件中有个名为 dBASEINL.OVL 覆盖程序，其中就有一段区域用来存贮制表符：

```
-d DS:59D0
0CD4:59D0 CD C4 BA 00 20 B3 B3 B3-C4 D9 BF B4 C4 C0 DA C3
0CD4:59E0 C4 C1 C2 C5 20 BA BA BA-CD BC B3 B9 CD C8 C9 CC
0CD4:59F0 CD CA CB CE 00 00 00 00-00 00 00 00 00 00 00 00
```

对于上述存贮单元的制表符代码，在西文字符显示环境下运行很正常，程序中每次显示菜单框和表格均到此区域中取代码。代码 CD 表示一个横线字符，B3 表示一个竖线字符，其他还有各方向的拐角和连接线等。如果在汉字系统下运行的话，当有两个制表符连续显示时，就会显示一些使人莫明其妙的汉字。因

此，修改这些制表符代码，也是汉化软件的一项必要的工作。

这些制表符代码用何种代码代换比较合适？这要根据两方面的要求来定：①不能与汉字机内码有二义性冲突，也就是说，最好用键盘有定义的标准 ASCII 代码。如果要用汉字制表符时，必须注意所显示的字符个数是单数还是双数，尤其是竖线必须保证每次显示两个字节才能形成一个汉字；②修改的同时，要注意屏幕的清晰美观，反复调试几遍，使用户感觉有一个良好的屏幕交互的环境。下面的数据代码就是针对以上所论述的情况，用 ASCII 标准代码进行汉化调整的。

```
-d DS:59D0
0CD4:59D0 3D 2D 7C 00 20 7C 7C 7C-2D 2A 2A 2A 2D 2A 2A 7C
0CD4:59E0 2D 2D 2D 2B 20 7C 7C 7C-3D 2A 2A 7C 3D 2A 2A 7C
0CD4:59F0 3D 3D 3D 2B 00 00 00 00-00 00 00 00 00 00 00 00
```

其中的代码 3D 是一个等于号“=”，而 7C 是“|”的ASCII 代码，一切拐角均用星号“*”来代换。首先，这些代码不会与汉字机内码发生冲突，其次，用一些用户习惯使用的代码修改即可。

还有一种情况，就是扩展 ASCII 代码的制表符分布在程序的各个部位。在显示之前向寄存器内发送一个代码，这些代码不固定在数据区，而是在代码程序区内。诸如这样的程序，修改代码的工作量就比较大。这需要仔细地跟踪，摸清程序显示的规律，只要找到了这些制表符代码，用 Debug 程序对其作相应的修改即可。

4.7.2 字符域的扩展

所谓字符定义域的扩展，就是把在数据处理程序中对ASCII 代码的识别范围扩大，使得大于 A0H 的汉字机内码可以顺利地通过。

在前面基础部分已讲过，汉字机内码是由两个高 8 位为 1 的异形国标码组成的。但是，大部分西文软件都不允许高 8 位为 1 的代码通过。就是允许通过的代码，也是某些软件作为控制码或判别码之用。往往在字串传送等数据流中用屏蔽高 8 位的方法对

于信息代码进行“过滤”，或者用测试指令和比较指令对流过的扩展数据代码作无效处理。因此阻障了汉字信息的处理。CC-DOS 汉字系统中汉字机内码的最小代码为 A1A1H，其对应的区位码为 0101H。譬如，一个“大”字，它的汉字机内码为B4F3H。在数据流处理过程中，一旦碰到“滤除”操作，有关的程序把其高 8 位的 1 屏蔽掉了，就变为两个标准 ASCII 代码 3473H，其扩展为二进制形式如下：

	十六进制	二进制
高 8 位为 1：	B4F3H	10110100 11110011
高 8 位为 0：	3473H	00110100 01110011

显然，数据代码被“滤除”之后，就是一般的 ASCII 代码了，CC-BIOS 汉字系统是不会把它们作为汉字处理的。一般，常见的“滤除”指令有如下几种：

```
247F     AND  AL,7F
257F00   AND  AX,007F
3C7F     CMP  AL,7F
A880     TEST AL,80
A98000   TEST AX,0080
```

这些指令代码可以在 Debug 调试程序中用下列的搜索命令来查找：

```
-SDS:100  FFF0  24  7F
19CA:2875
19CA:28BB
```

如果有此指令的话，Debug 立刻就能定位。修改的原理非常简单，对于“与”指令，可以修改为 AND AL,FF，或者把该指令用两个 NOP 空操作指令代替。对于 CMP 一类的比较指令，在修改时必须注意标志寄存器进位标志的变化情况，不能单纯地修改源操作数 7F 的比较值，有时还要修改跳转指令。对于 TEST 指令来说，因为源操作数在执行之后不变化，仅反映在标志位上。因此，只要对其后面跳转指令的地址作适当的修改，使

其代码作为有效码就可以了。

但是，在有些软件中，如 WordStar、Multiplan 等，其程序中有些“滤除”指令是有效的，系统用它来判定一些控制码和高亮度显示，尤如 WordStar 中的软回车换行控制码，是用的 8D，8A。因此，简单地去掉滤除指令也是不妥当的，有时会引起其他功能的丢失或执行出错。这需要用逐步逼进的方法进行跟踪，找到与汉字处理有关的“滤除”指令后加以修改，而与汉字处理无关的屏蔽和测试指令保留不变。

4.8 屏幕显示重新布局

标准的西文软件都是采用 80×25 的字符方式显示屏幕信息的。但汉字系统并非都能显示25行信息，这就需要对汉化过的软件屏幕显示的行数作些调整。

4.8.1 光标定位的调整

一些具有全屏幕编辑功能的软件，例如 dBASE、FoxBase、WordStar 和 Lotus 1-2-3 等，在显示或文字编辑时，当光标超过系统约定的最大行数时，屏幕信息就会自动滚行处理。这一约定的最大行数值，通常是在系统安装或配置等初始化时，由选择或其他方式定义好后存放在程序的某一个单元中，以便作为屏幕显示的限界之用。有些程序是在显示信息之前，与国定的25行值进行比较，一旦超出 25 行，就转去作滚行处理，经过分析均可得以确认。

例如 WordStar、Lotus 1-2-3 等软件中，在设置光标模块中，往往把当前屏幕光标的行/列值放在 DH 和 DL 寄存器内。DH 存放行值，DL 存放列值。在执行过程中，该行列值总是要与某个屏幕限界单元进行比较，程序如下：

```
CMP  DH,BYTE PTR [0243] ; 行等于限界值吗?
JGE  SCROL              ; 是，转上滚处理
INC  DH                 ; 否，行数加 1
```

```
        ⋮
SCROL                    ；上滚处理子程序
        ⋮
```

在一些限界单元中，存放了两个字节19 50或4F 18，其值为十六进制数，表示80列25行。光标值的增量大于等于这一值时，程序将会作上滚操作。

碰到此类软件，只须把限界单元的行值修改为汉字系统对应的行数即可。10行的CC-DOS环境，行限界值改为0AH，20行的汉字系统，行限界值改为13H等等。例如，在WordStar字处理软件中，其行限界单元在DS:0248地址处，行限界值为18H。可用Debug程序把其值改为0AH，就可在10行的CC-DOS下正常使用了。

还有一些软件，是采用直接比较法。每当光标定位或行值增加时，都要和一个固定的常数进行比较，如：

```
        CMP  DH,19     ；等于25行吗?
        JE   SCROL     ；是，转上滚处理。
        INC  DH        ；否，行值加1
          ⋮
SCROL:
          ⋮
```

行限界值与常数比较，在程序中往往不止一处，需要在多处进行修改。

当然，如果汉字操作系统的屏幕显示大于等于25行，那就没有必要修改此值了。

诸如dBASE、FoxBase一类的数据库软件，有一个圆点光标位，也就是该软件的提示符。该提示符光标通常都是定位在屏幕的最末一行。在西文环境下，多数在第25行上（当状态行关闭后）。若在10行CC-DOS下运行。就会发生跳闪，屏幕十分不稳定。这个限界行值，实际上是专门为光标定位的参数值，肯定是存放在存贮单元内。与显示用的行限界单元有所区别，汉

化时要加以注意。例如dBASE-Ⅲ PlusV1.1系统，在dBASEINL.OVL 模块的 DS:67A2 单元中，有一个光标行值19H，如果把它改为 0BH，就可以把圆点光标定位在第 10 行上。

4.8.2 屏幕显示信息的调整

当汉字操作系统的环境小于25行时，原软件一屏的信息就不能全部显示出来。如果是在 10 行的 CC-DOS 系统下，那么，三分之二的信息将被虚屏“吃掉”，而不能显示出来。或者是滚动地显示完，稳定后只剩下 10 行信息。这就应该适当地调整屏幕的格式或者对显示的信息重新布局。

通常，采用两种方法：一种是去除一些不重要的显示内容，使屏幕的显示格式尽可能地紧凑一些。由于每显示一行信息，总要经过光标定位程序的处理，每次光标定位总要送给寄存器一对具体的行列值。那么，为了压缩显示，可以对这些具体的行列值参数作适当的调整，以达到显示全部信息的目的。

另一种方法是把原来的一屏信息分为两屏显示。这种方法用于批数据显示比较合适，但不适合菜单选择屏幕。其汉化思想是：当显示信息满一屏幕后，调用一个键盘输入模块，等待用户输入任何一键后，再继续显示第二屏的信息。在显示第二屏信息之前，或者作一次清屏处理，或者使屏幕滚动显示均可。例如，在数据显示程序中插入一段下列的小程序：

```
        CMP  DH,0AH      ; 等于 10 行吗?
        JNZ  DISP        ; 否，转显示
        XOR  AH,AH       ; 是，等待输入
        INT  16H
        CALL CLEAR       ; 调清屏
        MOV  DH,0        ; 行位置 0
          ⋮
DISP:                    ; 显示子程序
```

4.9 人机交互信息的汉化

人机交互信息主要是指原西文软件在运行过程中，在屏幕上显示的软件名称、菜单选择、帮助操作信息和语句或操作出错后的提示等。大多数软件都是以英文来描述的，如果一个汉化的软件，要想普及推广，便于更多的用户接收，把这些以英文提出的信息改译为通俗简练的汉语，也是十分必要的。

这项汉化工作的方法和原理都比较简单，就是要求把原来的西文信息，通过正确的翻译，用汉语来代替即可。一般来说，这些菜单选择、屏幕提示和出错信息均放在某些扩展名为 HLP、MSG 或 OVL 等软件包的辅助程序模块中。个别的也有在主控文件中，但信息较少。

要翻译汉化一个较大的软件包中的信息数据的话，其工作量是很大的。不过，只要掌握了方法，利用一些软件工具，正确地掌握其技术规范，可以达到一次成功的境地，不必反复调试，将可节省不少时间。

汉化信息的方法有很多，一般要看数据量的大小和各人的喜好来选择。但是，所使用的工具软件最好是经过汉化的，这对处理汉字信息要方便得多。通常，所采用的工具软件如C-WordStar CDebug 或 PCTOOLS 等，都可以作为信息汉化的工具。

4.9.1 查找信息汉化法

这种方法对信息量较小的软件进行汉化很实用。也就是说，在屏幕上出现了一条提示信息，记录下这条西文信息的内容，用 Debug 或其他工具软件，把有关的数据文件调入内存，再利用搜索命令，根据记录下来的信息找到它的逻辑地址。然后把这条信息翻译并修改为中文，存盘后即可显示中文提示信息。

例如，在 dBASE Ⅱ 系统下，当打开一个磁盘未曾建立过的文件名时，dBASE Ⅱ 将会给出一条出错提示：File does not exist。汉化时，把其信息模块（可在任何一个文件中查找）用

CDebug 调入内存修改：

```
C>CDEBUG dBASE.MSG
—S 100 5000"File does not exist"
34DA:0361
—E 34DA:0361 "该文件不存在"，0
—W
—Q
```

这样修改之后，当再遇到此提示时，就会在屏幕上显示中文出错提示信息。

需要注意的是：通常所修改的中文信息的字节数不要超过原西文信息的字节长度。在汉化之前，必须弄清原西文信息串的结束符是什么字节码，大多数软件在一条信息串之后，即信息串的最后一个字节，用 0(NULL)字符表示结束。也有个别软件用"$"为结束符。了解了是何种结束码后，在修改的中文信息之后，也作要补入一个相同的结束符，否则，将会出现一些错误信息。

4.9.2 批数据信息汉化法

有些软件，把大量的菜单信息和提示信息集中在数据文件的一个区域中，并且，这些数据信息除了每一句当中有一个结束符分隔外，数据都是连续的。例如：FoxBaseV2.10 软件系统中，有一个名为 FoxPlus.RSE 的数据模块就是如此。其中有些提示信息如下：

```
DS:3770  20 65 72 72 6F 72 2E 00-54  6F 6F 20 6D 61 6E 79   error..Too  many
DS:3780  20 66 69 6C 65 73 20 6F-70  65 6E 2E 00 43 61 3E   files  open..Can
DS:3790  6E 6F 74 20 6F 70 65 6E-20  66 69 6C 65 2E 00 43   not  open  file..C
DS:37A0  61 6E 6E 6F 74 20 63 72-65  61 74 65 20 66 69 6C  annot  create  fil
DS:37B0  65 2E 00 49 6C 6C 65 67-61  6C 20 73 65 65 6B 20  e..Illegal  seek
DS:37C0  6F 66 66 73 65 74 2E 00-46  69 6C 65 20 72 65 61  offset..File  rea
DS:37D0  64 20 65 72 72 6F 72 2E-00  46 69 6C 65 20 77 72  d error..File  wr
DS:37E0  69 74 65 20 65 72 72 6F-72  2E 00 49 6E 76 61 6C  ite  error..Inval
```

像这样连续的信息，有两种方法进行汉化。一种就是用Debug直接修改。把其调入内存后，用 D 命令观察有关的存贮区域，以

确定修改的区域。譬如，修改 DS:3778 的信息，原西文内容是“Too many files open”。

—E DS:3778 “打开的文件太多”，0

因为该系统的信息结束符为 0，因此，在中文信息后面也补零。接着再修改下一条信息，从 DS:378D 开始，逐句修改，直到结束，用W命令存盘即可。用此方法虽然很有效，但是工作的进展很慢。尤其是当数据量较多时，效率明显降低。

另外一种方法是把这些连续的信息，从其首地址到结束地址的这一范围的数据重定向或者以读写文件的方式转存到另外一个文本文件中去。用 WordStar、Edlin 等文字编辑软件对该文本文件进行汉化修改。把汉化后的文本文件调入内存，用覆盖的方式，把汉化后的数据覆盖到原西文信息的同一地址范围中。其步骤如图 4-10 所示。用重定向存贮的文本文件，修改时与在 Debuy 下基本相同，但是用的行编辑或全屏幕编辑，不再需要进行地址计算，显然方便得多。

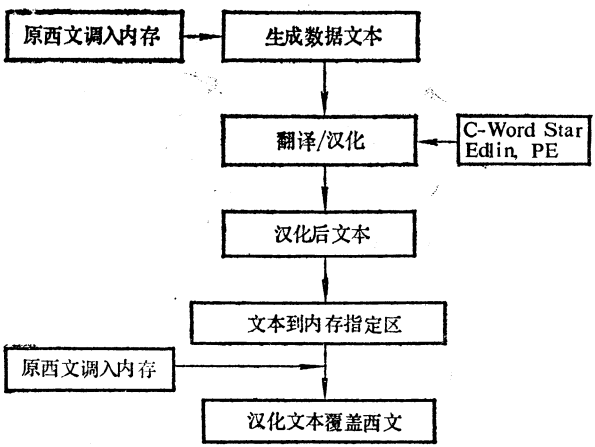


图 4-10 信息汉化步骤

利用 DOS 读写文件功能编制一个小程序，给这个程序输入几个指定的参数，如要汉化的数据文件名、偏移首地址、一共多少个字节、转存到什么文件中等。程序自动生成指定的文本或数

据文件，再用编辑软件对其进行翻译汉化，最后再把汉化后的文件调入内存，先移至高段地址，例如7000:0000，再把原西文数据文件调入内存，然后用Debug中的M命令，把高段地址的汉化文本移到西文数据所在段的指定地址上进行覆盖，用W命令存盘即可。

4.9.3 数据解释汉化法

还有一些软件，它的一些提示信息在文件中用常规的搜索方法是找不到的。因为这类软件的数据信息是根据某种算法进行了变换后存贮在文件中的。每当要显示某一级菜单或辅助提示时，系统首先把这一块信息串在内存中进行解释后，再逐个送往屏幕显示。如dBASE Ⅳ中的Help帮助提示文件就是这一类软件的典型例子。

汉化此类文件，需要在Debug调试软件中进行反复的跟踪分析：第一步要弄清其解释程序的算法，第二步要弄清其所解释的数据信息所在的首地址是哪里，最后还要分析出该系统的解释信息是分块解释还是一次运行后把所有的信息全部解释出来，倘若是分块解释，每一块的解释程序的算法是否相同。当这些问题都搞清楚后，再着手汉化工作就比较顺利了。

例如，下列程序是dBASE-Ⅲ Plus对其版权名称的解释算法程序：

```
0CEB:0A50 55       PUSH   BP
0CEB:0A51 8BEC     MOV    BP,SP
0CEB:0A53 C47E06   LES    DI,[BP+06]  ;指针装入附加段的DI
0CEB:0A56 FC       CLD                ;置传送的增量方向
0CEB:0A57 8B1C00   MOV    BX,001C     ;取字符缓冲区首地址
0CEB:0A5A 2E       CS:
0CEB:0A5B 8A07     MOV    AL,[BX]     ;从缓冲区取一个字符
0CEB:0A5D 0AC0     OR     AL,AL
0CEB:0A5F 7406     JZ     0A67        ;0字符表示结束，跳出
0CEB:0A61 34A5     XOR    AL,A5       ;字符码和A5异或
0CEB:0A63 AA       STOSB              ;把AL中新字符存入DI处
```

```
0CEB:0A64 43      INC     BX              ;字符缓冲区地址加1
0CEB:0A65 EBF3    JMP     0A5A            ;循环取存
0CEB:0A67 AA      STOSB                   ;把0存入DI表示结束
0CEB:0A68 8BE5    MOV     SP,BP
0CEB:0A6A 5D      POP     BP
0CEB:0A6B CB      RETF                    ;段间返回
    :               :
```

其中，BX 指向原数据区，ES:DI 指向解释后的数据区。这个程序算法比较简单，可以明显地看出，把原来的数据与 AS 进行异或处理后，存入字符缓冲区中，当程序返回时就可把该字符缓冲区的数据信息拿去显示了。

所以说，汉化这类软件中的信息，一旦了解了它的解释还原的算法，可以编制一段临时用的小程序，其算法和语句基本上与原解释程序相同。不过，源地址、目的地址可以自己指定。在内存中，用这个临时程序把原数据文件中的数据解释还原后存盘。这项工作完成后，就可以用上两节介绍的汉化方法，把其英文信息翻译成中文，再编制一个临时程序，用与解释程序相反的逆运算，把其恢复原状。然后再用内存覆盖法，用汉化过的数据代替原来的西文数据，存盘即可。但是要注意，两者的地址一定要对准，不能有偏差。

总之，数据信息的汉化方法有许多种，这里仅介绍了几种最常用的方法。给读者提供一些思路，以便在汉化过程中，不断探索出更多更好的方法。

第五章　软件汉化的实施

根据上一章介绍的软件汉化方法，本章将列举几个例子，以此说明汉化一个软件的实施情况，便于读者更好地掌握这一技术。

着手汉化一个软件系统之前，并不一定要急于找到有关汉字处理模块，马上就着手修改。应该在西文状态下反复地运行，了解该软件的各种功能，从外部，也就是说从人机界面上观察其运行的环境和屏幕的转换方式。然后进一步了解各个文件模块之间有何联系，用动态调试程序跟踪一部分程序。一旦对该软件有了一些了解后，再具体地实施汉化工作。

5.1　dBASE-ⅢPlus 的汉化

dBASE-ⅢPlus (V 1.1) 是美国 Ashton-Tate 公司推出的最新关系数据库管理软件。它不仅对 dBASEⅢ 进行了功能增强与扩充，而且针对近年来迅猛发展的局部网络环境做了新的开发工作，从而以多用户管理系统的面貌展现在用户的面前。

这套软件经过多年改进和完善，已经发展成一个既考虑用户交互操作的需要、又考虑程序员编写程序需要的数据库管理系统。它兼顾了系统的非过程性和过程性，是一个集此两大特性于一体的软件包。

在此基础上对该软件进行汉化，使其能够处理中文数据信息，这将无疑为我国信息管理和办公室自动化工作的发展提供了一个良好环境。由于系统较庞大，在此仅以单用户主控模块的二次开发为例，描述其输入/输出部分的汉化方法。

5.1.1　显示输出的汉化

dBASE-ⅢPlus 原西文系统仅工作在 80×25 彩色字符方式

下，不支持图形方式。因此，在 CC-DOS 汉字系统下将无法使用。dBASE 在程序装载时，对微机的设备字节进行了测试，以便确认当前 CRT 视频缓冲区的段地址是在 B000H 还是 B800H，换句话说，判别出是单色显示卡还是彩色显示卡，然后在某一个单元中置入一个标志。

当需要显示信息时，根据这一标志，系统把数据信息直接传送到相应的视频缓冲区。其格式为低位是字符的，高位是属性的。dBASE-Ⅲ Plus 显示模块的程序如下：

```
CS:0996   PUSH  AX                      ; 保存字符属性
CS:0997   MOV   AH,00
CS:0999   MOV   AL,DH                   ; 取当前行值
CS:099B   MOV   CX,00A0                 ; 一行宽度
CS:099E   MUL   CL                      ; AL 乘 CL 得出行坐标
CS:09A0   MOV   CL,DL                   ; 取当前列值
CS:09A2   ADD   AX,CX                   ; 行标加列标
CS:09A4   ADD   AX,CX                   ; 得出视频首址
CS:09A6   POP   DI                      ; 弹出属性
CS:09A7   PUSH  AX                      ; 保存视频首址
CS:09A8   PUSH  DI
CS:09A9   MOV   DI,AX
CS:09AB   PUSH  DS
CS:09AC   MOV   AX,2AEE                 ; 重定位
CS:09AF   MOV   DS,AX
CS:09B1   CMP   BYTE PTR [6A34],01
CS:09B6   JNZ   09D0                    ; 非图形显示器转
CS:09B8   CMP   WORD PTR[6A00], B800
CS:09BE   JNZ   09D0
CS:09C0   PUSH  DX
CS:09C1   MOV   DX,03DA                 ; 取 CRT 状态口址
CS:09C4   IN    AL,DX
CS:09C5   TEST  AL,08                   ; 视频允许吗?
CS:09C7   JZ    09C4
CS:09C9   MOV   DX 03D8                 ; 模式寄存器口
CS:09CC   MOV   AL,25
```

```
CS:09CE  OUT   DX,AL              ; 禁止视频显示
CS:09CF  POP   DX
CS:09D0  POP   DS
CS:09D1  POP   AX
CS:09D2  XOR   CX,CX
CS:09D4  MOV   AL,[BX]           ; 从数据区取字符
CS:09D6  OR    AL,AL
CS:9D8   JZ    09F1              ; 字符为 0 退出
CS:09DA  CALL  0A36
CS:09DD  ES:
CS:09DE  MOV   [DI],AX           ; 字符/属性送视频
CS:09E0  INC   CX                ; 计数加 1
CS:09E1  INC   BX                ; 数据区地址递增
CS:09E2  INC   DI
CS:09E3  INC   DI                ; 缓冲区地址递增
CS:09E4  INC   DL                ; 光标列值加 1
CS:09E6  CMP   DL,50             ; 满一行吗?
CS:09E9  JB    09D4              ; 继续送数据
CS:09EB  INC   DH                ; 行数加 1
CS:09ED  MOV   DL,00             ; 列数置 0
CS:09EF  JMP   09D4
CS:09F1  PUSH  D
CS:09F2  MOV   AX,5FD8
CS:09F5  MOV   DS,AX
CS:09F7  CMP   BTYE PTR [6A34],01
CS:09FC  POP   DS
CS:09FD  JNZ   0A07
CS:09FF  PUSH  DX
CS:0A00  MOV   DX,03D8
CS:0A03  MOV   AL,2D             ; 模式寄存器口
CS:0A05  OUT   DX,AL             ; 允许视频并显示
CS:0A06  POP   DX                ; 传送参数
```

上述程序是原西文的显示子程序，DH 和 DL 寄存器中的内容是当前屏幕的光标位置参数，由 CS:999～CS:9A9的程序段，

计算出光标所对应的当前视频缓冲区偏移地址。而 DS:BX 所指向的地址，是要往屏幕显示的数据缓冲区首地址。

在送数据之前，程序先判别一下是单色显示卡还是彩色显示卡，如果是彩色显示卡（根据初始化装载的标志单元判定），就关闭屏幕显示，以免在写数据时发生屏幕干扰。直到数据传送完毕之后，再打开屏幕。因此，从感觉上看，其显示的速度非常快，实际上是开/关屏幕的缘故。在开/关显示屏幕的操作中，程序指令向 3D8 模程寄存器端口设置了一个参数，该 8 位参数的第 4 位作为屏幕禁止/选通信号的控制位。这里的参数是指把 CRT 设置为 80×25 彩色字符模式。从 DS:BX 单元中取出一个字符，显示一个字符及属性，直到所取出的字符代码为 0，程序退出，屏幕信息显示完毕。

利用静态或动态分析方法，掌握了该系统数据信息显示的原理之后，即可以着手编汉化修改程序了。这个汉化方案是采用调 CC-BIOS 功能的设计思想，也就是说，利用显示中断 INT 10H 的服务功能，来实现汉字显示。

该汉化方案有三个步骤：①根据取得的光标位置参数在屏幕上设置当前光标；②用 9 号功能显示字符和属性；③取当前光标值后返回。其程序的实现方式如下：

```
CS:0996  PUSH  AX              ; 保存字符属性
CS:0997  MOV   AX,DX           ; 当前行/列值送 AX
CS:0999  POP   DI
CS:099A  NOP
CS:099B  PUSH  DI
CS:099C  MOV   DX,AX           ; DX=光标位置
CS:099E  MOV   AX,DI           ; 字符属性送 AX
CS:09A0  PUSH  AX
CS:09A1  PUSH  DX
CS:09A2  PUSH  BX
CS:09A3  MOV   BH 00
CS:09A5  MOV   AH,02           ; 光标定位功能
CS:09A7  INT   10              ; 调 BIOS 显示模块
```

```
CS:09A9  POP   BX
CS:09AA  JMP   09B3
          :
CS:09B3  POP   DX
CS:09B4  POP   AX
CS:09B5  MOV   AL,[BX]          ; 从数据取字符
CS:09B7  OR    AL,AL
CS:09B9  JZ    09E9             ; 字符为0退出
CS:09BB  PUSH  DX
CS:09BC  PUSH  BX
CS:09BD  MOV   BH,00
CS:09BF  MOV   BL,AH            ; BL=显示属性
CS:09C1  MOV   CX,0001
CS:09C4  MOV   AH,09            ; 显示字符功能
CS:09C6  INT   10               ; 调 BIOS 显示模块
CS:09C8  POP   BX
CS:09C9  POP   DX
CS:09CA  INC   BX               ; 数据区地址递增
CS:09CB  INC   DL               ; 光标列值加 1
CS:09CD  CMP   DL,50
CS:09D0  JB    099E             ; 小于一行
CS:09D2  INC   DH               ; 行值加 1
CS:09D4  MOV   DL,00
CS:09D6  JMP   099E             ; 继续显示
          :
CS:09E9  PUSH  CX
CS:09EA  MOV   BH,00
CS:09EC  MOV   AH,03            ; 取光标位置
CS:09EE  INT   10               ; BIOS取出光标至 DX
CS:09F0  POP   CX
CS:09F1  JMP   0A07
```

程序一旦改好后，就可以中西文兼容使用了。当然，这只是指在显示方式上，无论是西文环境还是汉字环境，都可以实现了。

在汉化过程中特别要注意的是：有重定位指针的指令不能随

意改动，因为修改程序是在非 EXE 扩展名的文件中修改的。当修改完毕后，再把文件改为具有 EXE 扩展名的文件后，其程序中所有的重定位指针全部由 PSP 文件前缀约定的值变化，这样，就会把修改过的程序冲毁。显示模块是在主控文件 dBASE.EXE 中，汉化时有类似的情况，应该用 JMP 命令跳过。

下面一段程序是原 dBASE-Ⅱ Plus 显示菜单框的子程序，即每调用一次，显示一个或一行制表符扩展 ASCII 码。读者可参照上述汉化方法，自己考虑一下如何汉化。

```
CS:0567  PUSH  AX
CS:0568  MOV   [6A02],DX
CS:056C  MOV   AX,[6A00]
CS:056F  MOV   ES,AX
CS:0571  MOV   AH,00
CS:0573  MOV   AL,DH
CS:0575  MOV   CX;00A0
CS:0578  MUL   CL
CS:057A  MOV   CL,DL
CS:057C  ADD   AX,CX
CS:057E  ADD   AX,CX
CS:0580  MOV   DI,AX
CS:0582  XCHG  AX,[BP-04]                ;取显示的制表字符
CS:0585  PUSH  AX
CS:0586  CMP   BYTE PTR [6A34],01
CS:058B  JNZ   05A5
CS:058D  CMP   WORD PTR[6A00],B800
CS:0593  JNZ   05A5
CS:0595  PUSH  DX
CS:0596  MOV   DX,03DA
CS:0599  IN    AL,DX
CS:059A  TEST  AL,08
SC:059C  JZ    0599
CS:059E  MOV   DX,03D8
```

```
CS:05A1  MOV   AL,25
CS:05A3  OUT   DX,AL
CS:05A4  POP   DX
CS:05A5  POP   AX
CS:05A6  MOV   CL,AH                    ; 显示字节计数
CS:05A8  PUSH  DX
CS:05A9  PUSH  AX
CS:05AA  MOV   AX,[69F8]                ; 取属性码
CS:05AD  CALL  0946
CS:05B0  MOV   DL,AH
CS:05B2  POP   AX
CS:05B3  CALL  0A36
CS:05B6  MOV   AH,DL
CS:05B8  POP   DX
CS:05B9  OR    CL,CL
CS:05BB  JZ    05D0
CS:05BD  ES:
CS:05BE  MOV   [DI],AX                  ; 写屏幕
CS:05C0  INC   BX
CS:05C1  INC   DI
CS:05C2  INC   DI
CS:05C3  INC   DL
CS:05C5  DEC   CL
CS:05C7  CMP   DL,50
CS:05CA  JB    05B9
CS:05CC  INC   DH
CS:05CE  MOV   DL;00
CS:05D0  CMP   BYTE PTR [6A34]; 01
CS:05D5  JNZ   05DF
CS:05D7  PUSH  DX
CS:05D8  MOV   DX;05D3
CS:05DB  MOV   AL,2D
CS:05DD  OUT   DX,AL
CS:05DE  POP   DX
CS:05DF  MOV   [6A02];DX
```

5.1.2 变量名及输入部分的汉化

当在系统建立文件时，首先生成一个数据结构，然后赋于每个字段一个名称。或者在dBASE状态下，给一个暂存变量赋初值，也需要一个变量名。例如：

·Name＝“王志虹”

但是，原dBASE系统只能接受字符型变量名，换句话说，仅允许用户赋于标准的ASCII字符的变量名，不接受中文的变量名，未经汉化之前，若设置一个变量：

·姓名＝“王志虹”

＊＊＊Unrecognized command verb.

系统将会给出一条出错提示，意思是无法识别的命令或变量名。这说明在输入的语词分析过程中，汉字代码已超出了系统的识别范围。因此，必须对输入识别模块进行汉化，扩展它的识别范围。

通过在键盘中断INT 16H处建立断点，用Debug调试程序对dBASE.EXE文件跟踪，原西文系统控制键盘输入码的程序模块如下：

```
CS:36F4  PUSH  BP
CS:36F5  PUSH  DS
CS:36F6  MOV   BS,SP
CS:36F8  MOV   CX,0000              ; 输入字符计数
CS:36FB  CLD
CS:36FC  LDS   SI,[BP+08]           ; 键盘缓冲区首址
CS:36FF  LES   DI,[BP+0C]           ; 数据缓冲区首址
CS:3702  MOV   AL,[SI]              ; 取一输入码
CS:3704  CMP   AL,41
CS:3706  JL    3740                 ; 小于 'A' 退出
CS:3708  CMP   AL,5B
CS:370A  JL    3716                 ; 小于 'Z' 转移
CS:370C  CMP   AL,61
CS:370E  JL    3740                 ; 小于 'a' 退出
```

```
CS:3710  CMP   AL,7B
CS:3712  JNB   3740          ; 大于'Z'退出
CS:3714  SUB   AL,20         ; 小写换大写
CS:3716  INC   SI            ; 地址加1
CS:3717  INC   CX
CS:3718  CMP   CX,[BP+10]
CS:371B  JGE   3740          ; 变量名超过10个
CS:371D  STOSB               ; 存入输入码
CS:371E  MOV   AL,[SI]
CS:3720  CMP   AL,30
CS:3722  JL    3740          ; 小于'0'退出
CS:3724  CMP   AL,3A
CS:3726  JL    3716
CS:3728  CMP   AL,41
CS:372A  JL    3740
CS:372C  CMP   AL,5B
CS:372E  JL    3716
CS:3730  CMP   AL,61
CS:3732  JL    373C
CS:3734  CMP   AL,7B
CS:3736  JNB   3740
CS:3738  SUB   AL,20
CS:373A  JMP   3716
CS:373C  CMP   AL,5F
CS:373E  JZ    3716
CS:3740  XOR   AL,AL         ; 无效输入码标志
CS:3742  STOSB
CS:3743  MOV   DX,DS
CS:3745  MOV   AX,SI
CS:3747  POP   DS
CS:3748  POP   BP
CS:3749  RETF
```

dBASE约定变量名最多不超过10个ASCII字符，即不超过5个汉字。在程序中，CX寄存器作为计数判别。DS:SI是指

向从键盘缓冲区取得的输入字符地址，ES:DI 指向加工好的有效输入码的地址。从上述程序中可以看出，对于变量名的 ASCII 代码，不可小于“0”或大于“z”，如果输入的是小写英文字符，通过 SUB AL,20H 指令，全部转换为大写字母。因此，当输入的汉字机内码大于 7BH 时，系统作为无效码处理。

汉化这类软件，不能单纯地把比较指令 CMP AL，7B 改为 CMP AL,FF。因为 JL 跳转指令是根据方向标志转移的，有时条件不满足但标志已置位的状态下，也会发生转移动作。再者，当输入的是汉字机内码时，绝不能作大小写字母的转换操作，而应作为有效输入码返回。对应其汉化程序如下列清单所示。

```
CS:36F4   PUSH   BP
CS:36F5   PUSH   DS
CS:36F6   MOV    BP,SP
CS:36F8   MOV    CX,0000          ; 输入字符计数
CS:36FB   CLD
CS:36FC   LDS    SI,[BP+08]       ; 键盘缓冲区首址
CS:36FF   LES    DI,[BP+0C]       ; 数据缓冲区首址
CS:3702   MOV    AL,[SL]          ; 取一个输入码
CS:3704   CMP    AL,41
CS:3706   JB     3740             ; 小于'A'退出
CS:8708   CMP    AL,61
CS:370A   JB     3718             ; 小于'a'转移
CS:370C   CMP    AL,7B
CS:370E   JB     3716             ; 小于'{'去转换
CS:3710   CMP    AL,FF            ; A0—FF为汉字内码
CS:3712   JNB    3740
CS:3714   JMP    3718
CS:3716   SUB    AL,20            ; 小写转大写
CS:3718   INC    SI               ; 地址加1
CS:3719   INC    CX
CS:371A   CMP    CX,[BP+10]
CS:371D   JGE    3740             ; 变量名超出10个
CS:371F   STOSB                   ; 存入输入码
```

```
CS:3720   MOV    AL,[SI]
CS:3722   CMP    AL,30
CS:3724   JB     3740          ; 小于'0'退出
CS:3726   CMP    AL,3A
CS:3728   JB     3718
CS:372A   CMP    AL,41
CS:372C   JB     3740
CS:372E   CMP    AL,61
CS:3730   JB     3718
CS:3732   CMP    AL,7B
CS:3734   JB     373C
CS:3736   CMP    AL,FF
CS:3738   JNB    3740
CS:373A   JMP    3718          ; 汉字内码转移
CS:373C   SUB    AL,20         ; 小写转大写
CS:373E   JMP    3718
CS:3740   XOR    AL,AL         ; 无效码标志
CS:3742   STOSB
CS:3743   MOV    DX,DS
CS:3745   MOV    AX,SI
CS:3747   POP    DS
CS:3748   POP    BP
CS:3749   RETF
```

按照这个程序修改后的dBASE-Ⅲ Plus输入模块，就可以在任何条件下接受汉字输入码了。再设置一个变量：

·姓名＝“林国强”

dBASE就会认为该变量有效。同样，在数据文件结构中，使用汉字作字段名，也不会出错了。

5.1.3 “滤除符”和提示信息的汉化

在dBASE.OVL覆盖文件中，有几处屏幕字符码高8位的操作。可用Debug把该文件调入内存，以便进行查找并修改。查找的范围可以分段进行，如果在当前段0—FFFEH之间没找到，

那么，再把当前的段地址加上1000H，仍然在0—FFFEH之间查找，以此类推，直到搜索出来为止。

dBASE-Ⅲ Plus中的“滤除”指令是AND AX，007F，其机器码为25 7F 00三字节，用S命令查：

-S 0 FFFE 25 7F 00

当找到其地址后，把7F改为FF或者把该指令用3个NOP指令代替即可。这时再用MODI COMM命令编辑程序或者作文字处理就可以接受汉字信息了。

此外，dBASE-Ⅲ Plus软件的提示信息，大多数都存贮在一个名为dBASE.MSG的文件中，利用汉化过的CDebug或其他工具软件，对提示信息文件中的英文提示以及一些菜单窗口进行翻译汉化，用E命令逐条修改即可实现。

如果在10行CC-DOS下运行，还须修改限界值，前面已介绍过，这里不再赘述。

5.2 Turbo-Pascal的汉化

Turbo-Pascal V3.0是一种交互式的软件，它面向高级语言，将程序编辑、编译、连结、运行等各种操作融为一体，其灵活的操作方法和友善的用户接口，是微机上较受欢迎的软件之一。

对于该软件汉化的分析步骤、跟踪方式等，前面已有介绍，这里仅作为一个实例，说明其具体修改的程序，便于读者更好地掌握汉化软件的技术。

原西文Turbo-Pascal不支持图形显示，除了进入系统后以80×25字符显示方式进行初始化屏幕外，主要有两个重要的子程序，需要作适当的修改，才能在汉字系统下正常运行。

这两个子程序分为计算视频缓冲区的偏移地址和屏幕信息传送。其程序如下：

```
0CEB:50DD   PUSH    DS
0CEB:50DE   POP     ES
```

```
0CEB:50DF   MOV    DI,04EE                ; 数据区首址
0CEB:50E2   CLD
0CEB:50E3   XOR    AH,AH
0CEB:50E5   MOV    CX,AX
0CEB:50E7   MOV    SI,AX
0CEB:50E9   ADD    SI,SI                  ; 列数值
0CEB:50EB   CS:
0CEB:50EC   MOV    AL,[016A]
0CEB:50EF   ADD    AX,AX                  ; AX=A0H
0CEB:50F1   MOV    BP,AX
0CEB:50F3   SUB    BP,SI
0CEB:50F5   MUL    BYTE PTR [0592]
0CEB:50F9   ADD    BP,AX                  BP=视频偏移值
0CEB:50FB   RET
      :          :
0CEB:5193   MOV    CX,SI
0CEB:5195   SHR    CX,1                   ; 显示的字节数
0CEB:5197   MOV    DI,BP                  ; 视频偏移值
0CEB:5199   MOV    ES,[0590]              ; ES=B800 段址
0CEB:519D   MOV    SI,04EE                ; 数据区首址
0CEB:51A0   CMP    BYTE PTR [0593],FF
0CEB:51A5   JZ     51AA                   ; 彩色卡
0CEB:51A7   REPZ
0CEB:51A8   MOVSW
0CEB:51A9   RET
0CEB:51AA   MOV    DX,03DA                ; 状态寄存器口
0CEB:51AD   LODSW                         ; 取字符和属性
0CEB:51AE   MOV    BP,AX                  ; 暂存
0CEB:51B0   IN     AL,DX
0CEB:51B1   RCR    AL,1                   ; 测屏幕选通
0CEB:51B3   JB     51B0
0CEB:51B5   CLI
0CEB:51B6   IN     AL,DX
0CEB:51B7   RCR    AL,1
0CEB:51B9   JNB    51B6
```

```
0CE3:51BB    XCHG   BP,AX          ; AX=字符和属性
0CEB:51BC    STOSW                 ; 送视频区显示
0CEB:51BD    STI
0CEB:51BE    LOOP   51AD           ; 显示 CL 个字节
0CEB:51C0    RET
```

用 Debug 程序对 Pascal 进行动态跟踪，就可以看到，DS:04EE 地址处是将要向屏幕显示的数据区，该区域内存放着被处理过的显示信息，在[0592]单元存放着当前的行数值，而[0593]

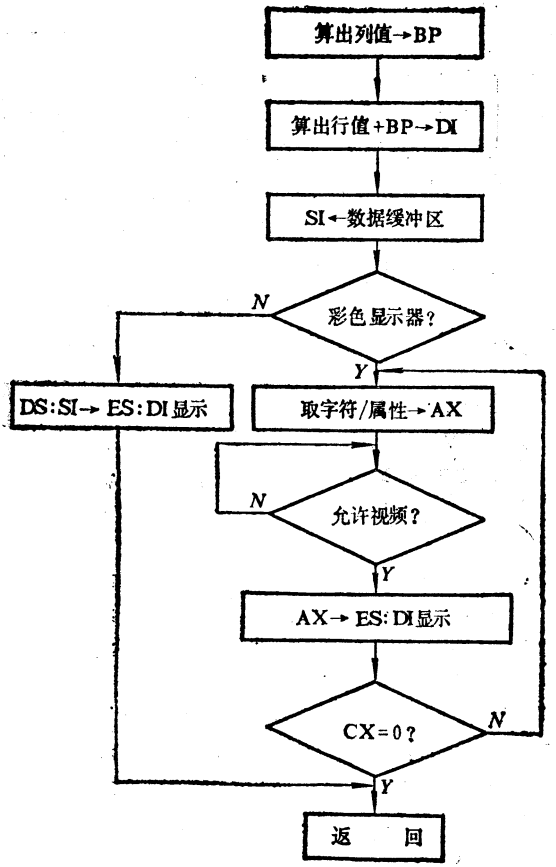


图 5-1　西文显示模块流程

是作显示设备判别的标志，即单色/彩色显示卡的标志。其整个显示过程如图 5-1 所示。

汉化这一类的软件，要掌握两个关键问题，即数据从哪里取出，又送往哪里；在汉字系统下用哪种功能可以代替这种读写方式。从流程图上来分析，即刻就能作出肯定的回答。

用 CC-BIOS 的视频中断 10H 中的 9 号功能作为数据显示时，首先要进行光标定位。以后每显示一个字符，光标都要重新定位一次。在汉化的程序中，BL 寄存器作为显示属性字节送给 BIOS。程序中设有回车换行处理，说明每调用一次该子程序，仅显示一行的信息。至于行限界，也是存贮在一个字节单元中，每显示一行信息后，均要与该单元的限界字节进行比较，直到当前的行坐标值大于或等于该字节时，转至屏幕上滚处理。经过汉化的显示模块程序如下列清单所示，请读者对照西文程序来分析和理解。

```
0CEB:50DD  PUSH   DS
0CEB:50DE  POP    ES
0CEB:50DF  MOV    DI,04EE          ; 数据区首址
0CEB:50E2  CLD
0CEB:50E3  XOR    AH,AH
0CEB:50E5  MOV    CX,AX
0CEB:50E7  PUSH   BX
0CEB:50E8  MOV    SI,AX
0CEB:50EA  MOV    DL,50
0CEB:50EC  SUB    DL,AL            ; 光标列数
0CEB:50EE  MOV    DH,[0592]        ; 行数
0CEB:50F2  MOV    AH,02            ; 光标定位
0CEB:50F4  XOR    BH,BH
0CEB:50F6  PUSH   DX
0CEB:50F7  INT    10               ; BIOS 功能
0CEB:50F9  POP    BP               ; 取光标
0CEB:50FA  POP    BX
0CEB:50FB  RET
    :      :
```

```
0CEB:5193   PUSH    BX
0CEB:5194   MOV     CX,SI              ; CL=字节数
0CEB:5196   XOR     CH,CH
0CEB:5198   MOV     SI 04EE            ; 数据区首址
0CEB:519B   PUSH    DX
0CEB:519C   MOV     DX,BP               光标位置
0CEB:519E   LODSW                      ;取字节和属性
0CEB:519F   MOV     BL,AH              ; BL=显示属性
0CEB:51A1   XOR     BH,BH
0CEB:51A3   PUSH    CX
0CEB:51A4   MOV     CX,0001
0CEB:51A7   MOV     AH,09              ; 显示一字符
0CEB:51A9   INT     10                 ; BIOS功能
0CEB:51AB   INC     DL                 ; 列数加1
0CEB:51AD   POP     CX
0CEB:51AE   CMP     DL,50              ; 满80列吗?
0CEB:51B1   JNB     51B9
0CEB:51B3   MOV     AH,02              ; 光标再定位
0CEB:51B5   INT     10
0CEB:51B7   LOOP    519E               ; 显示CL个字节
0CEB:51B9   POP     DX
0CEB:51BA   POP     EX
0CEB:51BB   RET
```

5.3 Lotus 1-2-3 显示模块的汉化

Lotus 公司推出的 Lotus 1-2-3是一种集成表处理软件包。由于该软件的强大数据表处理功能，很快就被广大用户接受。国内也拥有大量的用户，早期的 Lotus 1-2-3 V 1.1 版，已有汉化版本，广泛地应用于各个领域。因为 Ver 1.1 版在汉字环境下使用，其内存容量受到一定的限制，使它无法处理较大的表数据。

最近，Lotus 公司推出了 Lotus 1-2-3 Ver 2.0 版，它突破了 640KB 内存容量的限制，根据 EMS 规范，最大可以支持

4 MB 内存的使用。尤其是那些配置了 1 MB 以上扩展内存的 80286/80386 等微机，可以处理一张巨大的活动电子数据表。

Lotus 软件包有一个特点，就是它的各种功能的实现均是由相应的独立的程序文件来完成的。譬如说 V1.1 版本，其显示模块是在 TD.DRV 的驱动文件中，而 V2.0 版，却放置在 123.SET 文件中。这给汉化工作带来许多便利条件。因为这些独立的模块一般都较小，阅读和查找有关的子程序比较容易。

下面以 Lotus 1-2-3 V 2.0 的显示模块为例，列举出它的汉化程序。V 2.0 版不但支持扩展内存，而且在设备配置上兼容性很广。在初次使用时，用户必须按要求执行 Install 安装程序。在安装过程中要求用户定义当前系统显示卡（CGA、MDA 及 EGA 等）类型，打印机型号等。一切都设置好后，系统将产生一个 123.SET 文件。它把用户定义的信息贮存起来，以后每次起动 Lotus，都会调用该文件中的设备信息来控制程序的执行。下列的汉化程序就是放置在 123.SET 显示配置文件中的。

```
CS:3DE7  XOR   DX,DX
CS:3DE9  MOV   CX,00A0
CS:3DEC  DIV   CX                ; AX 除以 A0H
CS:3DEE  SHR   DL,1              ; 列数除 2
CS:3DF0  JMP   42F8
   :      :
CS:42CF  PUSH  DS
CS:42D0  MOV   AX,CS
CS:42D2  MOV   DS,AX
CS:42D4  MOV   ES,AX
CS:42D6  MOV   CX,007D
CS:42D9  LEA   DI,[012D]         ; 缓冲区地址
CS:42DD  XOR   AL,AL
CS:42DF  STD
CS:42E0  REPZ
CS:42E1  SCASB                   ; 搜索
CS:42E2  JZ 4325
```

```
CS:42E8  PUSH  CX
CS:42E9  MOV   DI,CX
CS:42EB  MOV   CL,05
CS:42ED  SHL   DI,CL
CS:42EF  LEA   SI,[DI+0620]      ; 数据区首址
CS:42F3  MOV   AX,DI
CS:42F5  JMP   3DE7              ; 计算光标
CS:42F8  MOV   DH,AL             ; 行值
CS:42FA  MOV   CX,0010
CS:42FD  PUSH  CX
CS:42FE  MOV   AH,02             ; 光标定位
CS:4300  INT   10
CS:4302  MOV   AX,[SI]           ; 取字符/属性
CS:4304  MOV   BL,AH             ; BL=属性字节
CS:4306  MOV   CL,01
CS:4308  MOV   AH,09             ; 显示一个字符
CS:430A  INT   10
CS:430C  INC   DL                ; 列数加 1
CS:430E  INC   SI
CS:430F  INC   SI                ; 数据地址增加
CS:4310  POP   CX
CS:4311  LOOP  42FD              ; 循环显示
CS:4313  POP   CX
CS:4314  POP   DI
CS:4315  MOV   AX,CS
CS:4317  MOV   ES,AX
CS:4319  XOR   AL,AL
CS:431B  STD
CS:431C  REPZ
CS:431D  SCASB
CS:431E  JNZ   42E7
CS:4320  NOP
CS:4321  CLD
```

```
CS:4322    CALL   3EDC
CS:4325    POP    DS
CS:4326    CLD
CS:4327    RET
```

Lotus 1-2-3 的显示方式与其他程序软件不同，它是从屏幕的右下角逆向显示到左上角的，在汉化时应予注意。尤其是设置方向指令 STD 不可丢失。该程序的显示过程是按表项进行的。显示一个表项，逆方向再搜索，如果搜索到下一表项，再继续显示，直到结束返回。

该程序段的地址是根据安装配置时用户指定的配置信息而生成的逻辑地址，配置参数不同，123.SET 文件大小或某些指令不尽相同。所以，其地址也会略有差别。不过，显示方法是基本相同的，可以按此程序进行汉化。经过这样修改的程序，就可在主控程序中处理汉字信息了。Lotus 1-2-3 的输入模块，不限制汉字机内码的通过，因此，可不必修改。

5.4 增添程序模块的方法

微型计算机拥有丰富的软件，促使计算机的应用推广到各个领域。因为各类软件的应用具有不同的要求和功能，所以软件的设计思想以及数据结构也有很大的差异，汉化软件也要针对各种不同的模块、地址和代码进行修改。显然，在汉化过程中，有些软件修改的程序量比较小，在原来的代码区对某些关键的指令修改就可以了，并且地址范围也容得下汉化过的程序段。但是，有些软件，修改的程序量比较大，而且在原来的有关代码区范围内也不够放置汉化的程序。由于目标程序的逻辑地址是连续的，移动程序块插入汉化的程序，就会造成一些地址错位，需要把所有的跳转指令和调用指令的地址都要作相应的调整。如果程序较大或者有重定位表存在，势必要花去大量的不必要的时间。这也是汉化工作中的一个矛盾。

解决这个矛盾的办法是在原目标程序的后面增加一段程序或者由一个软中断方法设置新加的程序入口，程序运行到有关汉字信息处理模块时，转到该程序内执行。这样可以尽量保留原西文系统的功能不丢失，又缩短了二次开发的周期。

本文提出的几种程序追加的方法，不仅仅是针对汉化的软件，对于各类目标程序，也均能使用。

5.4.1 命令文件的扩充

命令文件是指具有.COM 后缀名的程序文件。COM 文件是一种纯二进制代码文件，其结构比较紧凑。通常，它的程序长度不超过 64KB，仅在一个段内运行。这种文件被系统装入内存运行时，CS、DS、ES 和 SS 段寄存器均被定义为相同的起始段初值。因此 COM 文件相对其他文件来说，增加新程序比较方便。

当用 Debug 程序把一个 COM 文件调入内存后，可以看到其起始指令的地址为100H，而文件的总长度等于当前 CX 寄存器的值加 100H，即文件尾 = CX + 100H。增加程序模块的步骤为：

(1) 用 Debug 把要增加修改的 COM 文件调入内存。

(2) 在文件尾，即在地址大于或等于 CX + 100H 处，用汇编命令（A 命令）输入汉化程序的指令或修改程序。

(3) 在需要汉化和修改的模块中，用 JMP 或 CALL 指令转入新增加程序的首地址，并通过 JMP 或 RET 指令返回。

(4) 用 R 命令修改 CX 寄存器值，该值的大小等于所增加程序模块的尾地址减去 100H。十六进制运算。

(5) 用 W 命令把修改过的程序存盘。

COM 文件运行时，在 DS 的最低端建立一个相应的程序段前缀 PSP (Program Segment Prefix)，其地址则占用了00H—100H。所以，该文件加载后的第一条指令是从 100H 开始，并且是一条可执行的代码指令。PSP 包含许多项目，其中有文件名参数区 (FCB) 和命令行参数。命令行的参数放在从 81H 开始的命令行缓冲区，80H 单元指出命令行的长度。譬如，执行的 widn 是

一个 COM 命令文件：

C>widn 10

那么，当程序被加载到内存后，在 81H 开始的缓冲区内被代进来 10 这两个 ASCII 代码，而 80 单元却是 03(包括回车码)。这样就可以从该缓冲区中取得命令所代入的参数。

```
MOV SI, 81H
MOV AX, [SI]
```

在程序中，根据取得的参数，就可以进行程序控制和相应的处理了。

5.4.2 执行文件的扩充

执行文件是以 .EXE 为后缀的、可装入内存运行的程序文件。其结构要比 COM 复杂得多，因此增加新的程序，也要相应地麻烦一些。

EXE 文件可以由若干个模块连接而成，并且不限制在 64KB 范围之内。它可以在同一个段内，也可以在不同的段内，由该文件中的重定位表项决定。为了能够有效地增加修改的程序，应先了解 EXE 文件的结构。

EXE 文件是由文件首部和装入模块两部分组成。文件首部包括 14 项共 28 个字节的格式化区（如表 5-1 所示）和最多 97 项共 388 个字节的重定位表单元，该首部存放着系统对文件的控制和重定位信息。重定位表项对应于装入模块中作为段定位的内容。每个重定位项占用 4 个字节。实际上，这 4 个字节表示一个地址值，其中高位 2 字节指定某个需作段重定位字的段值，低位 2 字节指定需作段重定位的字的偏移地址。在文件首部偏移地址 06—07 的两字节给出了重定位表所含有效重定位项，首部偏移地址 18H 处的两个字节单元记录着重定位表在文件首部的偏移地址。

装入模块即程序正文，有若干个连续或不连续的程序段。在它的代码段前还有一个装入区，其中的装入字与重定位表中重定

EXE 文件首部结构　　表 5-1

项	偏移地址	项内容及意义
1	00—01	LINK 产生的 EXE 文件标志(4D 5A)
2	02—03	文件在末扇区中实际使用的字节数
3	04—05	文件总共占用的扇区整数 (512 字节 1 扇区)
4	06—07	重定位表的项目数
5	08—09	以节 (1 节=16 字节) 表示的文件首部长度
6	0A—0BH	装入程序代码末尾前要求的最小字节数
7	0C—0DH	装入程序代码末尾前要求的最大字节数
8	0E—0FH	被装入模块中堆栈段的偏移值
9	10—11H	被装入程序入口时，SP 寄存器的初值
10	12—13H	字检查和，文件中所有字的负和，忽略溢出
11	14—15H	程序装入内存时，IP 寄存器的初值
12	16—17H	被装入模块的代码段偏移值
13	18—19H	第一个重定位项的偏移地址
14	1A—1BH	覆盖号 (0 表示本程序常驻内存)

位项一一对应，装入过程如下：

(1) 将文件首部格式化区前 26 个字节读入内存，根据文件标志，确认是否为 EXE 文件。

(2) 根据格式化区有关项约定的文件长度(扇区数)、文件首部长度 (节数)、装入程序代码末尾之前要求的最小和最大节数，计算装入模块的长度和确定程序的起始段值。然后把装入模块读到从起始段开始的内存自由空间中。

(3) 在格式化区有关项指出的装入模块获得控制时，SP、IP 寄存器应有的初值存入内存指定单元，将程序起始段值加上装入模块中堆栈段、代码段的偏移值所得的 SS、CS 寄存器的初值存入指定单元。

(4) 由格式化区第一个重定位项的偏移地址，移动读写指针，将重定位表读到工作区中。根据重定位表的项数，逐项进行程序段的重定位，重定位项中的段值加上程序起始段值作段地址，结合重定位项中偏移地址得到对应的装入字，再把程序起始段地址加上该装入字。这时的装入字即为各程序段运行时的段地址。

从上述分析的原理看出，要想在EXE文件中增加新程序，就不能用修改CX寄存器文件长度的简单的办法了。尤其是对较大的程序，即大于64KB或者几百KB的文件，就更是如此。

修改或在EXE文件中增加程序，首先应该把EXE扩展名改成别的扩展名或者暂时去掉扩展名。因为系统不允许Debug把带有EXE扩展名的文件写盘。如：

C>REN CCCC.EXE CCCC

改名后再用Debug把它调入内存，其地址从100H开始，实际上，100H—300H是原EXE文件的首部及重定位表项内容。因此，原来的逻辑地址，也就是未改名之前的偏移地址，就需要相应地加上300H。

在程序的最后追加程序，一般有两种情况，一种是非常驻程序，这类程序需要避开紧接着原来程序之后的数据加工区。另一种情况是常驻内存的，这类程序应该注意，当装入程序运行结束后，必须保障把追加的新程序驻留在内存中再返回系统。

若追加的程序大于512个字节时，必须改写文件首部第3项内容。该项的内容是文件尾数除以512的商，余数放在首部的第2项中。否则，即使追加的程序能够存写到磁盘上，加载时也不会把它读到内存中来。除此之外，再修改CX寄存器的文件长度值。如果有必要的话，还要指定重定位项和项值。

EXE文件中第一条指令的地址是编程时指定的，经编译后，系统把第一条指令的逻辑地址值存放在文件首部的第11项中，作为程序装入时IP指针寄存器的初值，以此定位该程序在内存中的地址偏移量。如果要想在该地址之前增加新程序（尤其是那些要驻留在内存的EXE文件，经常会碰到这类情况），可在Debug下把这个地址开始的程序代码用M命令向后移动，然后在原来的位置上编写新的程序。在要写盘之前，除了以上讲的EXE文件追加程序的方法外，还需要把文件首部第11项的内容，修改为原装入时第一条指令被移动的新地址值。这样，当程序装入时，虽然IP初值不同了，但其操作执行的指令仍然不变。

5.4.3 用中断方法扩充程序

有些软件在汉化时，需要修改的程序比原来的西文程序大得多，挤不出适当的地址区域供这些程序编址。可以采用另外编写一个小程序，也可以仅针对一个功能或根据代入的功能号完成几个功能调用。

上述方法的基本思想是：按操作功能的要求编一个程序，该程序运行后便常驻内存，该程序的首地址，通过一个中断向量表来指定。在需要汉化的软件或需要增加功能的程序运行之前，先把这个小程序驻留在内存中，假如主程序运行过程中，需要调用该小程序才能完成操作时，只须设置几个入口参数，发一个相应的中断指令，即可进入指定的陷井，完成我们所需要的任何功能。实际上，这是一种程序截取方法，当然，也可以嵌套在 BIOS 的中断程序中。这种方法不仅用在汉化软件中，对于任何程序类似功能增加模块，均是一种简捷有效的方法。

下面列出的程序是一个汉字显示模块：

```
CODE  SEGMENT  'CODE'
      ASSUME CS:CODE,DS:CODE
DISP  PROCFAR
      STI
      PUSH   DS
      PUSH   ES                    ; 寄存器入栈
      PUSH   AX
      PUSE   BX
      PUSH   BP
      MOV    ES, BP                ; BP 数据段地址
Jsdisp:
      P、SH  CX
      MOV    AH, 2                 ; DX 代入光标参数
      INT    10H                   ; 光标定位
      MOV    AX,ES:[SI]            ; 取字符及属性
      OR     AL,AL                 ; AL=0 结束
```

```
        JZ      Dispret
        MOV     BL, AH                  ; 取属性码
        MOV     AH, 09                  ; 显示功能
        MOV     CX, 1
        INT     10H
        INC     DL                      ; 光标列加 1
        CMP     DL, 50H                 ; 满一行吗?
        JB      Coust
        INC     DH                      ; 是, 行值加 1
        MOV     DL, 0                   ; 列值回 0
Coust:
        ADD     SI,2                    ; 数据指针加 2
        POP     CX
        LOOP    Jsdisp
Dispret:
        POP     BP                      ; 弹出
        POP     BX
        POP     AX
        POP     ES
        POP     DS
        IRET                            ; 中断返回
DISP    ENDP
START   PROC    FAR
        PUSH    CS
        POP     DS
        MOV     AX, 256AH               ; 置中断 6AH
        LEA     DX,Disp
        INT     21H
        LEA     DX, START
        ADD     DX, 100H
        INT     27H                     ; 贮留内存退出
START   ENDP
CODE    ENDS
        END     START
```

该程序是调用 BIOS 的显示中断的 9 号功能。程序被装在 INT 6AH 中，这个中断号是可以由用户自己指定的。但要注意的是，中断类型号尽量不要与原 BIOS、DOS 和网络服务的一些中断类型发生冲突，以免丢失原系统功能。该程序只要求代入几个有关的参数，发送一个 INT 6AH 指令，即可完成操作。其入口参数为：

[DH]= 当前光标的行值
[DL]= 当前光标的列值
[CX]= 要显示的字节数
[BP:SI]= 数据缓冲区首址

譬如，在其他软件中，只要用如下几条指令，即可完成汉字或字符的显示：

```
FUSH BP
MOV BP,DS          ; 数据段地址
MOV CX,20H         ; 显示 20H 个字符
MOV DX,050AH       ; 第 5 行 10 列
LEA SI,[320A]      ; 数据缓冲区
INT 6AH            ; 显示
POP BP
```

当然，CX、DX 的值可以用变量或内存单元的值代入。总之，在自编的小程序中完全可以根据自己的要求来实现各种功能，不受地址范围的限制。另外，如果要在一个程序中完成许多功能，还可以用 AH 或其他寄存器作为功能号，进入中断后，再分支处理。

第六章　汉字库及造字原理

在汉字系统的环境下，汉化软件应用在信息管理、办公自动化等各个领域，均离不开汉字库。汉字库的信息量非常大，字库使用得好不好，对于各种汉字系统下软件的运行，有着直接的影响。而字库的质量好不好，则将会影响汉字输出的效果。本章将论述汉字库的使用、结构、字形维护以及造字等方面的原理及其设计。

6.1　扩展内存读写汉字库的技术

目前，广泛推出的80286/80386微机大都配置了1MB以上的扩展内存，若不使用XENIX系统，而由MS-DOS作为操作系统管理的话，系统的扩展内存仍然无法直接访问。因为DOS只管理640KB的存内空间，对扩展内存无法访问和利用。实际上，286/386的BIOS有管理扩展内存的功能，即利用保护方式进行存贮器的寻址。如果把庞大的汉字库装入扩展内存，将为用户空出550KB以上的实际内存空间，这一技术要比配置汉卡、读取硬盘汉字库等方法更为先进简便。在汉字环境下，保障了各种大型软件的使用。并且也为海量词库和大型图形软件的开发提供了良好的环境。下面就这一技术，详细介绍其原理、程序设计及汉字库管理的方法。

6.1.1　保护方式寻址原理及功能

80286/386微机都装有1MB以上的扩展内存(Expanded Memory)，而像这一类的处理器，除提高了系统的运行速度外，还具有虚拟存贮和存贮保护功能。因此说，286/386具有两种存贮器操作方式：一种是实际方式，这种寻址方式与8088/8086相

同，是通过段地址左移4位与偏移地址相加后，形成了20位的物理地址，在主内存 1^{MB} 字节范围内寻址。另一种方式称为保护方式，该方式与实际方式截然不同。该方式所形成的物理地址为24位，因此，可以在存贮器中 $2^{24}=16^{MB}$ 的地址范围内寻址。而在保护方式中，根本不使用实际方式的段地址和偏移地址。

在保护方式下，一个虚拟的地址由2个16位字组成，它们是一个段选择器和一个偏移地址。80286使用段选择器中的两位作为存贮保护，因此，一个程序可访问虚拟存贮器的 $2^{16-2}=2^{14}$ 个段。如果每个段含有 $2^{16}=64$KB字节，那么，虚拟存贮器可以提供 $2^{14}\times 2^{16}=2^{30}=1$ 千兆字节的寻址空间，也就是 10^9 个字节。实际方式与保护方式的关系式如图6-1所示。

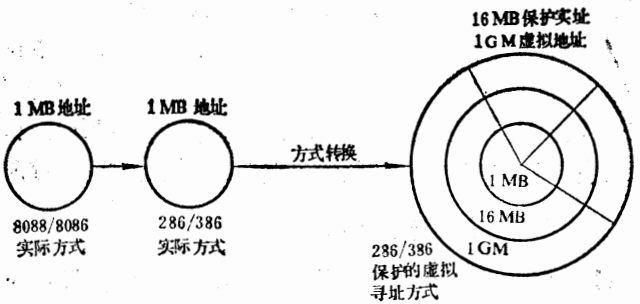


图 6-1　实址与虚址的关系

保护方式本质上是实际方式的超集合，利用内存管理部件启动，使保护方式提供了与实际方式相同的处理性能水平。而且，保护方式中有更强的超指令集与地址空间，用来做虚拟内存管理、任务管理和保护。

在286/386的BIOS服务程序中，有一个类型号为15H的中断模块。它提供了诸如事件等待、操纵杆信号、转虚拟方式和移动内存块等十几项服务功能。其中有一个功能就是在实际内存和1MB以上的扩展内存之间进行数据传送，亦称为内存块移动。其功能描述如下：

输入参数：[AH]=87H　；功能号

[CX]=每次要传送的字数

[ES:SI]=进入保护方式后使用的全局描述表的段：偏移地址

输出参数：[AH]=状态码

=00 传送成功

=01 RAM 奇偶错

=02 无效的全局描述表

=03 地址线 20 位失效

ZF=1 表示调用成功

CY=1 表示调用失败

另外，置功能 AH=88H，执行中断 15H 后，可以测试扩展内存的容量，即中断返回时，AX 寄存器以 KB 为单位表示系统具有扩展内存的大小，以此来保障 AH=87H 功能的有效操作。从输入参数上来看，实现该功能并不复杂，但是要按照约定的方式建立一个全局描述器表(Global Discriptor Table)，必须了解该表的各单元内容。并且按规定来初始化该表，这是关键所在。

6.1.2 全局描述器表(GDT)

Intel 80286/386 有两个专用的寄存器，称之为全局和局部描述寄存器，GDTR 和 LDTR。它们的作用是帮助 CPU 执行存贮转换的。在保护方式操作过程中，寄存器将查找表中所设置的信息，以计算物理地址和判测操作权限。整个 GDT 表共由 48 个字节构成，除了一些单元作为系统保留外，每个请求单元占用 8 个字节。而每个单元是由 4 个系统规定的字段组成。其格式如图 6-2 所示：

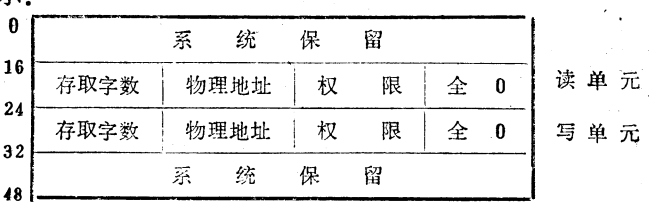


图 6-2 全局描述器表格式

全局描述器表各主要字段的意义如下：

(1) 存取字数字段：该字段占用表中第 16、17 两字节。该字段的内容限定每次传送的最大字数，通常设置为十六进制的 3FFFH，表示一次最多传送的字节≤32KB。

(2) 物理地址字段：该字段在 18—20 三个字节范围，用于指定 0—16MB 内存中 24 位实际的段地址。譬如，在 1MB 之内的地址为 5452:0000，应把它变换为 20 45 05 三个字节。若是 1MB 以上的扩展内存的地址为 10345:0000，则应变换为 50 34 10 三个字节后，分别按照低、中、高位写入该字段中，作为保护方式下 CPU 寻址的依据。

(3) 权限字段：第 21 个字节单元存放访问权限内容，说明 CPU 此次访问的类型。在保护方式下，286/386 的访问有多种权限类型，这里仅作为扩展内存的访问。在操作执行前，应把权限字段设置为 93H，其各位的意义如图 6-3 所示：

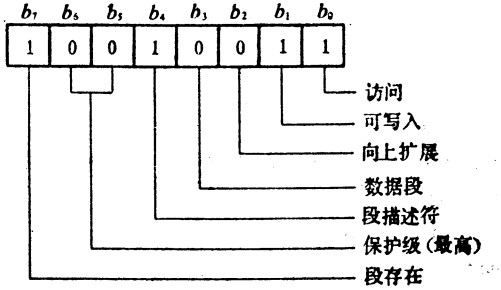


图 6-3 权限字节的各位意义

GDT 表内其他字节单元是系统保留给多任务、多用户和以后扩充用的。读和写这两个单元，除了物理地址字段的设置不同外，其他字段的参数均相同。此外，读单元和写单元中物理地址字段的参数设置，将决定实际内存与扩展内存之间的数据传送方向。

(1) 中断 15H 的 87H 功能调用包含有两种类型的操作，通过对 GDT 的读写单元设置，指定其当前的调用是读状态还是写

状态，因此，根据调用的要求来设置这两种状态参数。

（1）写状态：我们把从1MB以内的实际内存向1MB以上的扩展内存传送数据定义为写状态。为实现这一功能，在写单元的物理地址字段中，先设置为扩展内存的24位地址字节。而在读单元的物理地址字段置为经过变换的实际内存24位地址值。

（2）读状态：表示把1—16MB扩展内存中指定地址的数据读取到实际内存中来。在发出中断15H指令之前，应把写单元的物理地址字段初始化为实际内存的24位地址值，读单元中该字段的三个字节，则置为扩展内存的24位物理地址值。

需要注意的是，在系统每次发INT 15H中断指令前，必须初始化GDT表，即设置指定值。下列Pascal程序说明了保护方式的调用及初始化GDT表的方法：

```
function at-movel (const source-address:ad3mem;
                   const target-address:ad3mem;
                   length       ; word)
type
              GDT-index=0…47;
var
  i:GDT-index;
  GDT:array [GDT-index] of byte;
  registers:reglist;
  begin{at-movel}
    if length=0
      then
        at-movel:=0
      else if length>32768 then
        at-movel:=#ff
      else
        begin
          for i:=0 to 15 do
            GDT[i]:=0;
            GDT[16]:=lobyte(2*length);
            GDT[17]:=hibyte(2*length);
```

```
            GDT[18]:=source-address. lo;
            GDT[19]:=source-address. md;
            GDT[20]=source-address. hi;
            GDT[21]:=#93;
            GDT[22]:=0;
            GDT[23]:=0;
            GDT[24]:=GDT[16];
            GDT[25]:=GDT[17];
            GDT[26]:=target-address. lo;
            GDT[27]:=target-address. md;
            GDT[28]:=target address. hi;
            GDT[29]:=#93;
            GDT[30]:=0;
            GDT[31]:=0;
            for i:=32 to 47 do
                GDT[i]:=0;
                with registers do
                begin
                  ax:=#8700;
                    cx:=length;
                    es:=(ads GDT). s;
                    si:=(ads GDT). r;
                end;
                intrp(#15, registers, registers);
                at-movel:=registers. ax
        end;
end;{at-movel}
```

6.1.3 保护方式汉字模块读写的实现

根据上两节论述的原理，通过对一些汉字操作系统作一点改进，即可以实现。这里针对大家比较熟悉的 CC-DOS V2.1 为蓝本，进行如下几方面的修改：

(1) 在CC-BIOS 系统软件中(如 CCCC.EXE 文件)，开辟一个48字节的描述器表单元，在内存中初始化为图 6-4 所示的数

据格式，其起始地址为 DS:AAEB。

```
DS:AAE0                                          00  00   00 00 00
DS:AAF0 00  00 00 FF FF 00 00 00-00 00 00 FF 3F  00 00 10
DS:AB00  93  00 00 FF 3F  00 00 10-93 00 00 FF FF 00 00 0F
DS:AB10  9B 00 00 FF FF 30 00 00-93 00 00
```

图 6-4 全局描述符表(GDT)设置

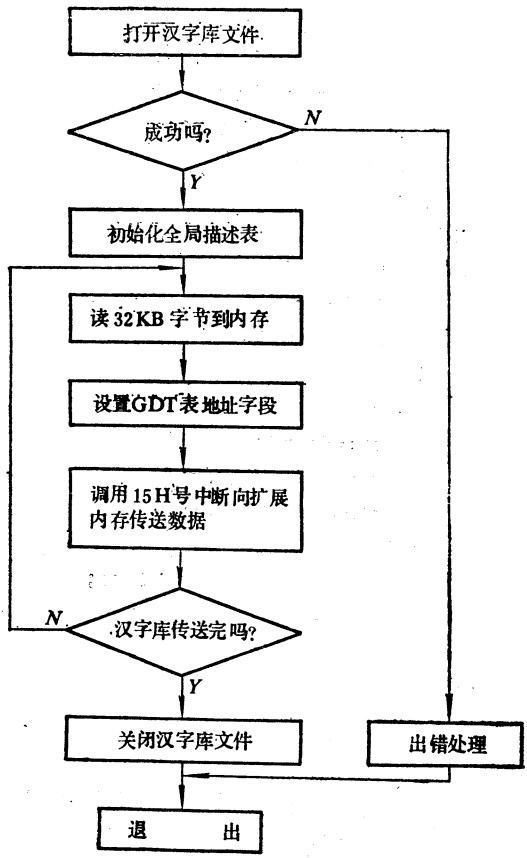


图 6-5 将汉字库装入扩展内存流程

(2) 为了从扩展内存向实际内存传送汉字点阵，还应该开设一个大于等于 32 个字节的字模缓冲区。当汉字库全部装入扩展

内存之后，必须把该缓冲区的首地址变换为24位存入GDT表中写单元的物理地址字段内。

（3）汉字库装载模块可以按照图6-5的流程设计，对原CCDOS把汉字装入实际内存的程序进行改写。

其方法是把汉字库的点阵信息先读到实际内存中的自由空间，再通过保护方式功能调用，把读入实际内存的数据传送到扩展内存中去。在文件的循环读写过程中，GDT表中的物理地址字段内容按4000H（即32KB）递增。如下列程序清单所示：

```
CS:AB56   LEA   DX,[AA79]      ;指向"CCLIB"文件名
CS:AB5A   MOV   AX,3D00
CS:AB5D   INT   21             ;打开字库文件
CS:AB5F   JNB   AB64
CS:AB61   JMP   AC02
CS:AB64   PUSH  AX             ;保存文件句柄
CS:AB65   MOV   AX,CS
CS:AB67   MOV   CL,04
CS:AB69   ADD   AX,0AC1        ;段址+偏移地址(AC10)
CS:AB6C   PUSH  AX
CS:AB6D   ROL   AX,CL          ;循环左移4位
CS:AB6F   AND   AL,0F          ;算出24位物理地址高位
CS:AB71   MOV   [AAFF],AL      ;存入GDT表
CS:AB74   POP   AX
CS:AB75   SHL   AX,CL          ;算出24位地址低、中位
CS:AB77   MOV   [AAFD],AX      ;存入GDT表
CS:AB7A   POP   BX             ;弹出文件句柄
CS:AB7B   PUSH  CS
CS:AB7C   POP   ES
CS:AB7D   MOV   CX,4000
CS:AB80   MOV   DX,AC10        ;读文件缓冲区地址
CS:AB83   MOV   AH,3F
CS:AB85   INT   21             ;一次读入32KB字节
CS:AB87   OR    AX,AX
CS:AB89   JZ    ABA3           ;AX=0读结束
```

```
CS:AB8B   MOV   SI,AAEB              ; ES:SI 指向 GDT 首址
CS:AB8E   MOV   CX,2000              ; 存入虚址 32KB 字节
CS:AB91   MOV   AX,8700
CS:AB94   INT   15                   ; 调保护方式传送
CS:AB96   ADD   Byte Ptr [AB06],40   ; GDT 表虚址中位增量
CS:AB9B   JNB   AB7D
CS:AB9D   INC   Byte Ptr [AB07]      ; GDT 表虚址高位增量
CS:ABA1   JMP   AB7D
CS:ABA3   MOV   AX,CS
CS:ABA5   MOV   CL,04
CS:ABA7   ADD   AX,0AB2              ; 段址+字模缓冲区指针
CS:ABAA   PUSH  AX
CS:ABAB   ROL   AX,CL
CS:ABAD   AND   AL,0F                ; 算出字模缓冲区24位高位
CS:ABAF   MOV   [HB37],AL            ; 初始化 GDT 表
CS:ABB2   POP   AX
CS:ABB3   SHL   AX,CL
CS:ABB5   MOV   [AB05],AX            ; 字模缓冲区地址中、低位
CS:ABB8   MOV   Byte Ptr [AAFF],10
CS:ABBD   MOV   AH,3E
CS:ABBF   INT   21                   ; 关闭字库文件
          ⋮
```

(4) 在读取汉字点阵模块中，即在原程序计算汉字模的地址处，插入了一段功能调用程序，如下列程序所示：

```
CS:276C   AND   DX,7F7F              ; DX=汉字内码
CS:2770   PUSH  AX
CS:2771   CMP   DH,29
CS:2774   JNZ   2778
CS:2776   MOV   DH,26
CS:2778   CMP   DH,30
CS:277B   JB    2780
CS:277D   SUB   DH,08
CS:2780   SUB   DH,21                ; 算出区码
CS:2783   MOV   AL,5E
```

```
CS:2785  MUL   DH              ; 区码X94位
CS:2787  SUB   DL,21
CS:278A  XOR   DH,DH
CS:278C  ADD   AX,DX           ; 计算出区位码
CS:278E  PUSH  CX
CS:278F  CALL  AAC0            ; 读取汉字字模
CS:2792  POP   CX
CS:2793  POP   AX
CS:2794  RET
; …
CS:AAC0  MOV   CX,0020         ; 每个汉字32个字节
CS:AAC3  MUL   CX              ; 算出字模首地址
CS:AAC5  PUSH  ES
CS:AAC6  PUSH  CS
CS:AAC7  POP   ES
CS:AAC8  ADD   DL,10           ; AX:DX=24位地址
CS:AACB  MOV   CS:[AAFF],DL    ; 初始化GDT表
CS:AAD0  MOV   CS:[AAFD],AX
CS:AAD4  MOV   CX,0010         ; 读16个双字节
CS:AAD7  MOV   SI,AAEB         ; ES:SI=GDT首地址
CS:AADA  MOV   AX,8700         ; 功能号
CS:AADD  INT   15              ; 调保护方式传送
CS:AADF  MOV   DX,CS           ; DX指向读出的字模
CS:AAE1  ADD   DX,0AB2         ; 缓冲区实际段地址
CS:AAE5  POP   ES
CS:AAE6  RET
```

实际运行中，把系统发来的汉字机内码先转换为相应的区位码，在该区位码的基指针上乘以32（一个汉字32个字节），计算出指定汉字字模的首地址，还需要把读单元的物理地址的高位字节增加10H，表明是在1MB以上的扩展内存中寻址。每读取一次汉字字模，则要设置一次物理地址字段，以实现有效的24位地址定位。其实现流程如图6-6所示。

这里所介绍的技术，可以在任何版本的汉字系统中实现。该

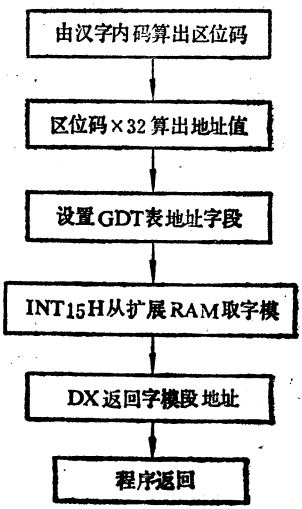


图 6-6 从扩展内存取字模流程

技术摆脱了 EMS 页面操作和虚拟磁盘管理的局限，为用户最大限度地贡献出实际内存空间。

6.2 汉字字形及其结构

国家标准《信息交换用汉字编码字符集——基本集》共收集汉字 6763 个。其中，第一级汉字 3755 个，第二级汉字 3008 个。这一标准中所收的汉字都是“正字”，也就是说，没有包括异体字、旧体字等“非正字”。所以，所收集的 6763 个汉字也是不完全的，还有许多字没有收集进来。有一些生僻字或专业符号，使用的频率很低，但只要用到它，就一定要有这个汉字的字形。这就需要对基本集进行合理的维护，因此，了解汉字库的数据结构，对于使用汉字信息的人员来说也是很重要的。

6.2.1 汉字库的存储

汉字由于笔划繁多，至少要 15×16 点阵以上，才能满足基

本字形要求。否则许多汉字，即使采用压缩变通简化处理技术，也很难正确表示。汉字字形点阵数越大，字形失真就越小，但字形库的容量也就越大。我国目前已颁布了许多种汉字点阵库的标准。各种不同点阵的字库，其使用的场合不同，因而它们的规范及存储容量亦不同。

15×16 点阵通常采用单线体，也有个别的系统采用宋体。由于汉字中很多字的横笔和竖笔均超过八笔，而且又有笔锋，故 15×16 点阵的字形不太美观。相对其他点阵来说，其容量不算很大，大多 15×16 点阵的字库被调入内存作为显示汉字的字模。

24×24 点阵，由于其点阵稍大，点阵字形也基本可以做到笔画齐全，分宋、黑、楷、仿宋和繁体等，适合于普通针式打印机输出使用。

32×32、48×48 点阵，可以更好地描绘宋、黑、楷、仿宋等多种风格的汉字。也可以比较满意地放大字形，用于高质量的打印输出。

64×64 以上的点阵字形，一般来说，因为所占存储容量较大，均采用压缩还原技术，可用于电子照排系统或激光打印输出等。

汉字点阵信息都是由二进制表示的数字代码组成，其点阵的规格越大，字库的容量就越大，表 6-1 列出了每种规格的汉字库所占存贮器容量的基本情况，它是根据国标字库 87 个区的汉字量计算出来的。

汉字点阵库容量表　　表 6-1

点阵类型	每汉字字节数	每汉字点位数	汉字数	字库容量
16×16	32	256	7426	237.63KB
24×24	72	576	8176	588.82KB
32×32	128	1024	8176	1.05MB
40×40	200	1600	8176	1.64MB
48×48	288	2308	8176	2.36MB
64×64	512	4096	8176	4.19MB
128×128	2048	16384	8176	16.75MB

6.2.2 汉字字形结构

不论汉字字形的笔划多少，都可以写在同样大小的方块中，从而可以把这一方块划分为许多小方格，组成一个“点阵”。每一个小方格是汉字点阵中的一个点（即组成字模笔划的最小单位）。例如，对于一个16×16点阵，是把一个方块横向分成16格，纵向也分16格，从而形成有256个小方格的网格图。也就是说，该矩阵有256个点。点阵中每个点可以有黑、白两种颜色，用这样的点阵就可以描绘出汉字的字形，因此，称作汉字点阵字形。

在这一网格中，很容易用二进制数字来表示点阵。如果用二进制的“1”表示黑点，用二进制的“0”表示白点，那么，一个16×16的点阵字形，就可以用一串二进制数（即256位二进制数）来表示。如图6-7中的“合”字的点阵就可以用256位二进制数来表示，由左向右，从上到下逐点描绘。

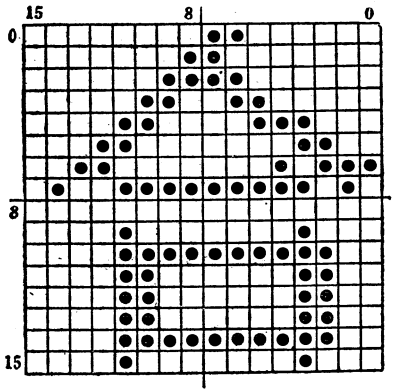


图 6-7　汉字点阵字形

因为二进制表达不方便，通常用十六进制数来表示。实际上，一位十六进制数就是4位二进制数，两位十六进制数就是8位一个字节，以上面的“合”字为例，它的点阵信息可以用以下一串十六进制数表示：00，C0，01，80，03，C0，06，60，0C，

38，18，0C，30，17，4F，FA，00，00，08，08，0F，FC，0C，18，0C，18，0C，18，0F，F8，08，10。

一个字节是8位二进制，所以16×16的字形点阵共有32个字节，则需要用256位二进制数来表示。

计算机在往屏幕传送字模的字节信息时，就是按照图6-7的字节位置，两个字节一排，共传送16排。因此，16×16的字模库排列是为了屏幕显示安排的横排式字模库。

6.2.3 高点阵字形结构

因为16×16点阵的汉字不能完整地描述某些汉字的笔划，因此质量较低，一般仅用于屏幕显示。若要增强汉字的质量，就必须要扩展字模的点阵信息。24×24点阵的字模，其每个汉字为72个字节，它的排列大多数是按照打印机的针头顺序排列的。即一列有3个字节，每个字节8个点位，共有24点，如图6-8的字模格式。

d_1	d_4	…	d_{70}
d_2	d_5	…	d_{71}
d_3	d_6	…	d_{72}

图 6-8　24×24点阵格式

因为考虑到汉字库的使用情况不同，故对字模数据的表示形式亦要求不同。例如，在屏幕上显示汉字时，其排列格式是以横排为主，即汉字点阵是从左到右，从上到下，因此，要求字模数据表示形式以行式（或称横式）为佳。针式点阵打印机打印汉字时，由于打印机针头是纵向排列的，因此，要求字模数据表示形式以列式（或竖式）为佳，而激光印字机，要求字模数据表示形式以行式最好。

各种不同点阵字模库的数据结构均不相同，它们在汉字库中的存放格式是多种多样的。了解这些字模点阵的结构，于对编写输出类的程序，或对字形库的维护，都会有很大帮助。以下是一

些常用的高点阵字模的格式。

32×32 点阵字模，每个汉字由 128 个字节组成，竖式按列的顺序存放。每列 4 个字节，共有 32 列。如图 6-9 所示：

d_1	d_5	…	d_{125}
d_2	d_6	…	d_{126}
d_3	d_7	…	d_{127}
d_4	d_8	…	d_{128}

图 6-9　32×32 点阵格式

40×40 点阵字模，每个汉字由 200 个字节组成，也是竖式顺序存放，其每列有 5 个字节，共有 40 列。如图 6-10 所示。

d_1	d_6	…	d_{196}
d_2	d_7	…	d_{197}
d_3	d_8	…	d_{198}
d_4	d_9	…	d_{199}
d_5	d_{10}	…	d_{200}

图 6-10　40×40 点阵格式

48×48 点阵字模的数据排列顺序与上述几种字库的点阵有所不同，它分为上下两部分存放。如图 6-11 所示。

d_1	d_4	…	d_{142}
d_2	d_5	…	d_{143}
d_3	d_6	…	d_{144}
d_{145}	d_{148}	…	d_{286}
d_{146}	d_{149}	…	d_{287}
d_{147}	d_{150}	…	d_{288}

图 6-11　48×48 点阵格式

其数据排列顺序是：先排上一半，从左到右，一列有3个十六进制数据，一列排完再排第二列，上半部排完后再以同样的方式接着排下半部分。这样的排列对于24针打印机分两行打印完整的汉字是极为方便的。通常，对48×48点阵的字模都是分别分两部分处理的。

6.3 造字系统的功能

虽然汉字库中有6000多个汉字字模，但是碰到一些异体字，或专业符号等，就无法调出，这时，可以通过造字软件，对原字库进行修改、添加等维护操作，以实现使用自己所造的汉字的目的。

通常，造字软件要具有以下几个主要功能：

(1) 显示各区的汉字；

(2) 造字；

(3) 改码；

(4) 删除；

(5) 帮助提示。

6.3.1 高点阵的汉字显示

显示各区的汉字是指：按用户键入的区码，把整个区内的汉字从01位到94位依次送到屏幕上去，并按横式排列显示出来。这种显示，为用户造字、改码等操作提供了一个非常直观的信息窗口，让用户自由地选择在哪个区造字，或者改码。

16×16点阵、24×24点阵的字模相对较小，一个屏幕可以把整个区中94个汉字全部显示出来。如果是32×32点阵以上的字模，就必须考虑分屏显示。当显示出一屏汉字后，应提供用户几个选择功能，就是上一区翻页、下一区翻页和重新指定某一个区显示等。显示完毕后，或想停止显示时，即可退回主菜单。

显示模块的功能流程如图6-12所示。在此模块中，必须注意各种不同点阵的汉字在屏幕的地址的计算方法。

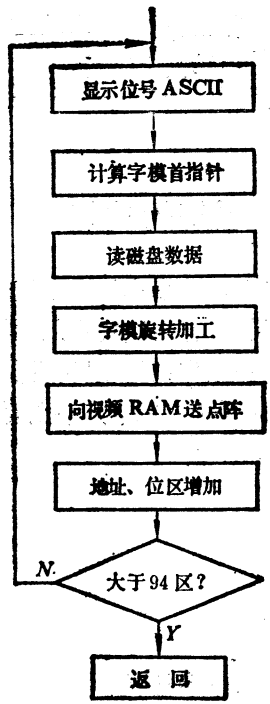


图 6-12 分区汉字点阵显示流程

显示每个汉字点阵字节之前，用ASCII 字符显示其位号，再通过这一区位码，计算出该汉字所在字库中的偏移量，16 点阵为区位号乘以 32，24 点阵偏移量等于区位号乘以 72，以各种字模库中每个汉字所占的单元字节为计算依据即可。在程序中还需要开辟两个字模缓冲区，一个是作为从磁盘读入汉字点阵字模的数据缓冲区，另一个作为各种加工之用。例如，点阵排列旋转、造字与改码，在造字网格内相应的各种变形操作等，都是对应于该字模加工区而进行的。

除了 16×16 点阵是横向排列外，其他高点阵汉字字模的排列大多都是竖式的。必须通过旋转处理，如下列程序：

```
WORD_R1:                          ; 24×24点阵竖向转横向
        MOV     DX,8
        LEA     SI,DOT_DTA        ; 原字模缓冲区
        LEA     DI,DOT_DTB        ; 目的缓冲区
        MOV     CX,3              ; 3×8=24列
WORD_R2: PUSH   CX
        PUSH    DI
        CALL    WORD_R3
        POP     DI
        POP     CX
        INC     DI
        LOOP    WORD_R2
        RET
WORD_R3:
        MOV     CX,8
WORD_R4: PUSH   CX
        PUSH    SI
        PUSH    DI
        CALL    WORD_R6
        POP     DI
        POP     SI
        POP     CX
        DEC     DL
        CMP     DL,0
        JNZ     WORD_R5
        MOV     DL,8              ; 处理8位
WORD_R5 ADD     SI,3
        LOOP    WORD_R4
        RET
WORD_R6
        MOV     CX,3
WORD_R7 PUSH    CX
        PUSH    DI
        CALL    ROTUN
        POP     DI
```

```
                POP     CX
                ADD     DI,24
                INC     SI
                LOOP    WORD_R7
                RET
ROTUN:
                MOV     CL,8
ROTU1: PUSH     CX
                MOV     AL,[SI]             ;取一个字节
                SHR     AL,CL               ;右移一位
                MOV     CL,DL
                RCL     AH,CL               ;移位8次
                OR      BYTE PTR [DI],AH;和原点位相或
                XOR     AH,AH
                ADD     DI,3                ;地址加3
                POP     CX
                LOOP    ROTU1
                RET
```

以上的程序是以24×24点阵字模处理的快速旋转模块，若是不同的点阵，只要把循环的次数修改一下即可通用。SI是指向原始点阵字模的，程序运行完后，则在DI的地址处，就是已经加工好的高点阵的显示点阵字模。这个模块中关键的子程序是在ROTUN标号指定的地方。在SI处取出一个字节处理8次，向纵向变换后，DI的地址加3。因为24点阵一行有3个字节，如果是32点阵，则加4，如果是40点阵则加5，等等。通过这种点位的移位操作，便可以实现各种不同排列的点阵变换。

6.3.2 造字的原理和设计

实际上，造字的过程，就是把用键盘对屏幕网格图像的操作对应于字模缓冲区中的点位操作。每一个图形符号或汉字的点阵字模由显示屏幕上的行×列的网格内的若干小点来确定。

该网格由若干垂直线和若干水平线交叉形成，对于24×24点阵来说，其横向有24格，竖向有24格，这种网格仅对构成字模

的点进行定位。所以，不同的点阵将对应有不同的网格。

对于造字模块来说，也同样包含了许多操作功能。其中有写/抹网格，光标的上移、下移、左移、右移，4个对角的控制，另外还有汉字旋转、上下左右搬动整字、造好的汉字存盘和汉字的前翻后翻等功能操作。当然，为了用户的方便，还可以加入更多的功能，如行插入、列插入等。

当用户输入一个完整的区位码后，程序首先把该区位码转换为该汉字在磁盘文件的起始指针，然后把指定点阵的汉字字模数据读到数据缓冲区来。如果是高点阵字库，必须进行竖转横处理，然后，首先在屏幕指定的位置按照汉字库的点阵大小把它显示出来。最后把加工过的汉字点阵按比例和网格一同向屏幕写入。最后把光标定位在指定的网格上，利用键盘中断 INT 16H 的1号功能测试键盘的状态，等待输入功能的选择。

这时，如果没有输入，程序将循环测试状态，并且按一定的周期（可用一个计数器）把显示的光标用 NOT 指令取反。使得在等待输入的状态下网格内的光标不停地闪烁。各种功能的具体实现如下：

(1) 光标的移动写点。在程序中需定义一个键，作为写点/抹点的开关。例如用小键盘的光标键移动光标，可用〈Num Lock〉键把键盘由光标状态切换为数字状态。在程序中，虽然这两种状态所读取的扫描码相同，但 AL 中返回的 ASCII 码却不同，若小键盘在光标控制状态，只在 AH 寄存器返回扫描码，而在数字键状态，AL 中却返回数字的 ASCII 代码，以此来判断写点和抹点。

在屏幕上写一个网格，有时需一两个字节，但是对应的汉字点阵字模却只增加了一个点位，所以说，无论在屏幕的网格上写点还是抹点，光标每移动一格，程序均要作点位处理，使其与点阵字模时刻对应变化，其程序处理如下：

```
MOV  BL,80H          ; 最高位置1
SHR  BL,CL           ; 右移到当前位
```

```
        MOV  AL,[SI]          ; 对应地址取字节
        CMP  BP,0             ; 写点/抹点
        JZ   Null             ; BP=0 抹点
        OR   AL,BL            ; 当前位与原字节相或
        JMP  SHORT Dots
Null:   NOT  BL               ; 当前位取反
        AND  AL,BL            ; 屏蔽当前位
Dots:   MOV  [SI],AL          ; 再写入原地址
        CALL Disp             ; 显示一个小字
        RET
```

程序中的CL表示当前屏幕位置字节是8位中的第几位，如果是写点，就把移到该位的这一点（1状态）和对应位置上的字节“或”处理后再写回原位置，说明该汉字在这个字节中已增加笔划内容。若是在抹点状态，就把经移位处理的字节取反，这意味着指定的那一位原来是1，取反后是0，而其他位取反后则都为1，然后再和原来的字节相“与”，这样“与”的结果，使得这一点总为0，而其他位不变，当写入缓冲区后，该汉字的这一点位就被抹去了。最后调用一个显示小字(按标准点阵)的程序，在屏幕上可以立即反映出被修改或新造的汉字的实际字形来。

上面这个过程，实际上是整个造字系统的核心程序。要设计一个成功的造字软件，不管是什么类型的点阵，只要掌握这个基本原理，再增加一些辅助功能，就可以适应任何复杂的造字程序。

光标移动过程中，必须对光标在视频缓冲区的扫描线位置进行重新计算定位，并且把这一地址保存在内存变量中，以便下一次光标移动时用来计算相对地址。程序设计时，要考虑上下左右和4个对角的边界限制，不允许光标移出网格。如图6-13所示的造字网格。虽然是24×24点阵的，但是对32×32、48×48等类型的高点阵网格与字模缓冲区的对应处理，其原理都是一样的，只不过要根据屏幕的分辨率来定义其网格中每小格的大小。

(2) 汉字图形的移动。这里指的是在网格内的汉字图形，通

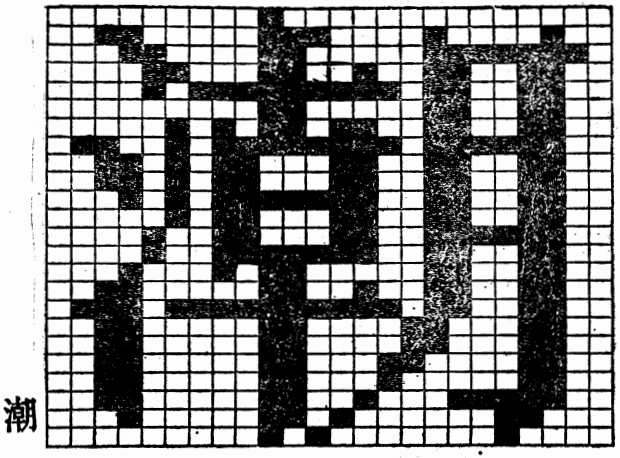


图 6-13　24×24 点阵造字网格

过功能键的控制，使其在网格内上下左右地自由移动，甚至旋转、四面循环等。

屏幕上所有的操作都是对应着字模缓冲区进行的。那么，这些功能的实现，也是通过对缓冲区内的字模数据进行变换后调用一次写网格信息的子程序即可实现。

例如，24×24 点阵字库中，一个汉字需要 72 个字节，那么，把缓冲区定义为 75 个字节单元，并且认为该缓冲区首尾相联，是一个类似于键盘缓冲区一样的循环缓冲区。

当进行上移功能操作时，就把网格最上面一行（即 3 个字节）移到字模缓冲区尾，也就是对应网格的最下面一行，其他字节全部向前移动 3 个字节指针，然后，以此字模格式重新写网格信息即可显示出向上移动了一行的汉字图形。

当向下移动时，则把缓冲区的 72 个字节向后移动 3 个字节的位置，然后，把最后 3 个字节传送到缓冲区首部，也就是移到首地址处，然后重新写网格信息。这样处理的速度是相当快的，给用户只有一个网格刷新的感觉。

汉字的左移右移，相对上下移动处理起来要复杂一些，这里

给出的是一段向右移的程序：

```
        LEA SI,Buffer                ; 字模缓冲区
        MOV CX,24                    ; 共24列
Right:  PUSH CX
        PUSH SI
        MOV CX,3                     ; 每列3个字节
Right2: RCR BYTE PTR [SI],1          ; 循环右移
        INC SI                       ; 地址递增
        LOOP Right2
        CLC
        POP SI
        POP CX
        ADD SI,3                     ; 地址递增
        LOOP Right
        RET
```

实际上，该程序的功能是把字模缓冲区内的所有字节按列均向右移了一位，然后再重写屏幕。如果考虑到循环显示的话，可以从第二个字节向右移，最后一个字节的低位内容移进标志寄存器，然后通过对第一个字节移位，把标志寄存器的内容移到该字节的最高位即可。

向左移动原理相同，只是从第三个字节（一行内最后一个字节）开始向左移位即可实现。

若要增加在光标处横向或纵向插入或删除一排点阵的功能，只要先定位光标当前所对应字模缓冲区中那个字节的第几位地址后，用相同的原理就可以实现该功能。

有些造字软件允许汉字图形按90°进行旋转，可根据上一节介绍的字模旋转程序来实现。

6.3.3 造字网格的显示

显示网格一般分两种形式，一种是显示一个空网格，然后再

显示汉字点阵。另一种，则是根据字模缓冲区内的汉字点阵经变换后和网格信息一同送往屏幕。当然，编程方法有多种，有的是按点显示的，有的是按字节显示的。按字节显示，相对来说要快得多。这要看屏幕上对点阵的行数是否安排得下而定。

如何把字模缓冲区的点阵字节，在屏幕上以大网格的形式显示出来呢？第一步应该把纵向排列的点阵字节，按照屏幕显示的格式旋转为横向排列。然后，再按一个字节 8 位的处理方法处理每一位，每一位对应着屏幕网格中的一个小方格。程序的流程如图 6-14 所示，这里仅画出了显示一位点阵的流程图，根据点阵的大小，将调用该子程序若干次。

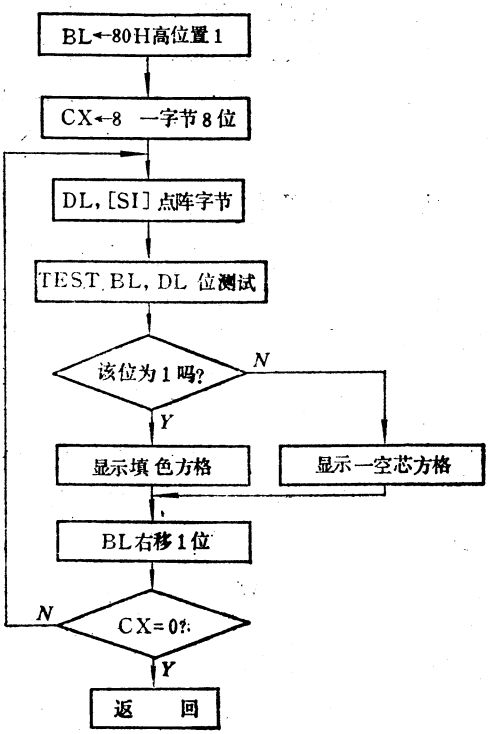


图 6-14 点位方格的显示过程

程序中显示填色/空芯方格的处理可以按点、字节和双字节进行，这里指的是横向显示。纵向显示应根据显示器不同的扫描方式，以逐行、隔行或隔双行的方法重复显示，尽量使得小方格接近正方形。按图 6-13 显示的方格，是按双字填色和置空芯的，以下的程序是显示一根线的过程，BP 寄存器为 0 表示送空芯加上方格左边的竖线：

```
        CMP  BP,0              ; 空芯吗?
        JNZ  Full              ; 否
        MOV  AX,0080H
        STOSW                  ; 送空芯字
        JMP  SHORT RTS
Full:   MOV  AX,0FFFFH         ; 全1字
        STOSW                  ; 送填色字
RTS:    RET
```

6.3.4 改码及磁盘读写

改码的意思是把原来某一个汉字的区位码用新定义的区位来代替。实际处理过程也可以把原区位码对应的汉字点阵复制到新定义的区位码处，这对于造字提供了方便。即可把与要造的新字类似的汉字取过来进行适当的修改，或者使用原来汉字现成的边旁笔划等，节省了造字的时间。

当进入改码子程序时，屏幕提示要求输入要改码的区位码，一旦区位码输入完毕（4 个 ASCII 字符），程序将根据这个区位码从磁盘中把对应的汉字点阵读到内存来，然后在屏幕网格上显示出来。接着将提示你把该码要改为其他的何种区位码，这时，可以输入一个新的区位码，作为对应该汉字的代码。一旦程序接受到这一新的区位码，通过运算，得到该区位码的汉字字模在磁盘文件库中的首指针，最后通过写盘功能，把第一次读出的汉字字模点阵，从缓冲区中按第二次输入区位码换算的指针写到磁盘字库文件中去，作为永久保存，这就是改码的全过程。

无论是从磁盘的字库读，还是写入磁盘字库，都要先进行字模在字库中的指针换算。在读写操作时，需要两个入口参数：一个是打开字库文件时，DOS为该文件约定的句柄，也就是字库文件的检查字。这是在打开文件（用AX=3D02H功能）后，BX取得的检索字，保存在内存单元中。另一个参数就是该汉字的机内码，由该机内码来换算对应的字库文件中指针位置。下列程序即为读/写汉字点阵过程。

```
Read:   MOV BX,[CODE]      ; 取出汉字机内码
        CALL Compt         ; 去换算指针
        LEA DX, Buffer     ; 字模缓冲区首址
        MOV AH,3FH         ; 读文件功能
        INT 21H            ; 读CX个字节
        RET
Write:  MOV BX,[CODE]      ; 取汉字机内码
        CALL Compt         ; 换算指针
        LEA DX,Buffer      ; 字模缓冲区首址
        MOV AH,40H         ; 写文件功能
        INT 21H            ; DOS功能调用
        RET                ; 写CX字节后返回
Compt:  AND BX,7F7FH       ; 屏幕高8位
        SUB  BX,2121H      ; 转为国标码
        MOV AL,94
        MUL BH             ; 乘94区
        XOR BH,BH
        ADD AX,BX          ; 换算为区位码
        MOV DX,72          ; 一个汉字72个字节
        MUL DX
        MOV CX,AX
        XCHG CX,DX         ; DX:CX=字模指针
```

```
MOV BX,[Handle]          ; 取出文件句柄
MOV AX,4200H             ; 文件指针定位
INT 21H                  , DOS 功能
MOV CX,72                ; 读/写 72 个字节
RET
```

如果是 32×32 点阵字库，将以 128 个字节作为操作单位，如果是 48×48 点阵字库，则需要 288 字节。除了处理字节不同外，高点阵字库的其他读写操作基本相同。

6.4 汉卡系统造字的考虑

由于把汉字库全部调入内存，占用了系统许多可贵的内存空间，致使许多大型软件在汉字系统下没有足够的内存，不能运行。因此，大量的汉卡系统问世，就是为了满足这种社会需求。在汉字系统下，调用 ROM 汉字点阵，空出实际 RAM 内存空间，提供和保障在汉字系统下进入网络和大型软件等正常使用。

虽然汉卡系统节省了大量的内存，但是，一般固化在 ROM 芯片上的汉字点阵信息，只允许读，不允许写。也就是 Read Only Memory (ROM) 只读存贮器。因此，汉卡上的数据无法改写，若要用到二级字库中没有的汉字和专业符号时，就无法调出。因此，根据这种要求，必须考虑一种变通的方法来解决。

汉卡内的字模通常是按照国标区位码的位置顺序存放的，无论汉卡中固化的是精密字模点阵还是压缩过的点阵信息，都可以根据汉字机内码，通过运算或者还原等方式在内存中指定的字模缓冲区中得到该汉字的点阵字模。

国标区位码的范围为 01—94 区，实际上，88 区以后是空白区，作为保留。那么，可以把 88—94 区这 7 个区作为造字的扩展区，每个区 94 个汉字，共有近 600 个造字空间，基本可以满足用户的使用要求，同时也避免了与汉卡内的汉字冲突，而且，88—94 区的代码，汉字系统亦可识别。

汉卡系统的造字主要从以下三个方面考虑：

第一步：首先建立一个字库文件，用于存放88区以后新造的汉字点阵数据。采用纯数据文件结构也可以，这样还需要编制一个装载和寻址程序。如果直接用Debug编制一个字库文件，由于装载和寻址程序很短，用COM文件作定义是很方便的。其结构分三个部分：①字库的装载部分，即在系统下直接运行字库文件，通过装载程序，把字库装入内存或是指定的位置，或以中断方式进入，然后驻留内存退出；②字模地址运算程序，这部分程序的功能，是根据给定的汉字机内码（必须大于等于88区），计算出该汉字的点阵字模在内存中的段地址，这部分程序包含一些内存标志单元；③汉字点阵数据，这部分内容是用户自己造的汉字字模，数据从指定的偏移地址处开始存贮，在计算出某个汉字的地址后，要加上该偏移地址，才是该汉字字模的实际地址。

第二步：造字的实现。汉卡系统下的造字，除了磁盘读写之外，其他功能都和软字库方式相同。

在读写磁盘的子程序中，如果是88区以前的汉字机内码，可采用调CC-BIOS取汉字点阵的功能，这样，就不用再重新编写读取汉卡内字模的程序。大部分汉卡读取字模程序都是通过某个中断程序来实现的。因此，利用这一陷井，也可以达到预期的目的。

如果读取的汉字机内码大于等于88区，那么，就打开自己建立的字库文件，以88区为基地址（其指令可用CMP DH,F8H来判别汉字机内码的高位），计算出该汉字字模在文件中的指针并加上指定的偏移量，然后把字模数据读到内存来。

写字模数据时，应保障88区以前的汉字不回写，因为把数据写入ROM汉卡的操作无意义。把88区以后所造好的汉字点阵数据，用DOS提供的写文件功能，存入字库文件相应的地址。在写文件的同时，还要往字库文件的一个内存单元中写入文件的长度，以保障在字库文件装入时，所造的汉字字模数据都能装入。

第三步：汉字系统字模的读取。如果在汉卡系统下使用应用

软件，并需要调用新造的汉字时，应该在进入 CC-DOS 之后再把 88 区之后的字库文件调入内存。利用中断截取的方法或者程序覆盖的方法均可，关键是把原系统只有调取汉卡的功能截取一个 88 区以后的汉字读取分支。

例如，原汉卡系统读取汉字模是用中断方式，那么，当新造的字库装入时，就把其中断向量取到内存的某个单元中，然后再重置该中断，作为汉字读取的入口。当系统需要读取汉字点阵字模时，首先判别一下该汉字机内码是否大于等于 88 区，若不是，则转到原系统中去读汉卡信息。若是 88 区以后的汉字，利用字库中计算字模地址的程序，算出该字模在内存中的地址指针，再把汉字点阵信息按 32 个字节（16 点阵）或 72 个字节（24 点阵）传送给调用程序。如图 6-15 流程所示：

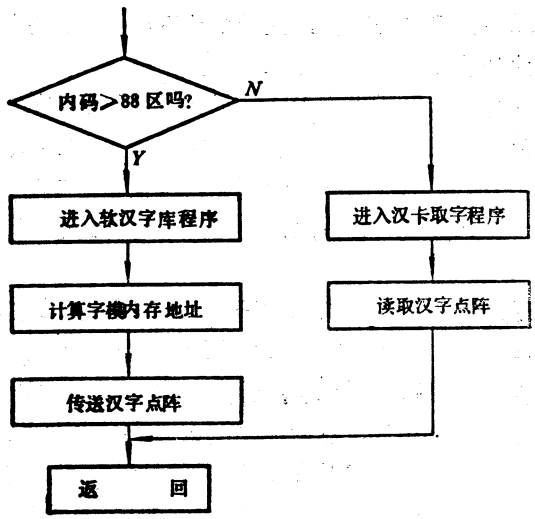


图 6-15　读取汉字点阵

随着计算机硬件设备的迅速发展，使用汉字的环境将会越长越好，但使用汉卡毕竟也是一个好的方法，配合点阵的压缩技术，可以把 64×64 点阵、甚至 128×128 或更高的点阵压缩在几个

存储器芯片中，对于高质量的打印输出提供了良好的基础，像这一类的汉字系统，用上述介绍的造字方法来补偿 ROM 芯片中的汉字“非正字”，可以得到较好的使用效果。

第七章　汉字打印驱动程序的设计

汉字信息的打印输出，在计算机汉字应用领域中占有举足轻重的地位。随着计算机应用技术的发展，常常要求打印输出的汉字能具有多种字型和字体，以提高输出文本和报表的质量。这类变换，虽然用硬件或软件的方法都可以实现，但一般用户均采用软件方法对指定的汉字字模通过一些算法进行变换。由于国内目前流行的几种打印输出的字型，远不能满足用户的要求，在此，作者提出一种较新颖的汉字打印驱动程序的设计思想，解决了目前一些驱动程序只能把汉字的点阵字节按倍数扩展的不足。

该设计思想的特点是兼容性强，可使用各种型号的24针打印机。它以24×24点阵汉字字模为基础，在12×12—96×96点阵或更大范围内，对每个字节的点位进行横向或纵向的任意缩放。可输出的字型种类有$S=(X_{max}-X_{min})(Y_{max}-Y_{min})$之多。如果定义的最大字型可扩展到96×96点阵的话，那么，可以输出的字型有$S=(96-12)(96-12)=7056$种。除此之外，还有转向90°、网点背景、下划线、上下标、斜体、黑底白字、行间距调整和宋、楷、黑、仿宋等几种字体的变化，可以说变化无穷。

这种按点位缩放的汉字打印输出，基本上可以对应各种印刷字号。如果再采用平滑补偿技术，在硬件良好环境的支持下，不但可以满足一般用户的要求，还可以作为某些计算机轻印刷系统的辅助输出驱动程序。

7.1　打印机汉字打印原理

一般的24针打印机均有图形点位打印功能，实际可以独立地控制每一根打印针的击出。而打印头有两排12根撞针，交错地布

置，可以打印在同一横坐标上，产生一排24点的纵列。重复移动打印头的位置（横坐标），或适当地移动行距，便可以打印出任意的图像来。在汉字系统下，汉字点阵字模每个字节的各个点位，对应着打印机的针头。图 7-1 就是一种最典型的字节与针头排列的对应图。如常用的M2024、LQ-1500、TH-3070和NEC-P7 等打印机，它们的针头与系统送来的点阵字节的关系就是如此。

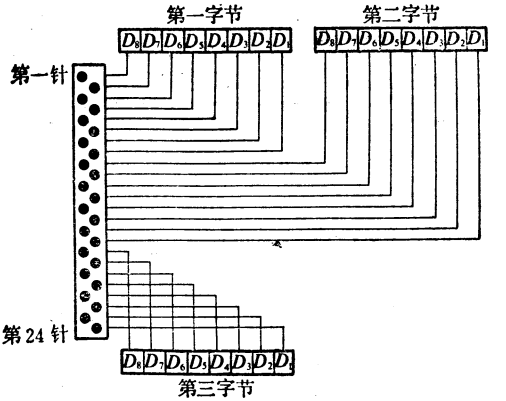


图 7-1　点阵字节与针头的对应关系

从图中可以看出，打印机的一列 24 根针可对应 3 个点阵字节。因此，打印机一行打印的高度是24点，如果输出16×16点阵汉字或24×24点阵汉字，均可以一次打印出来，如果打印的汉字其高度超过了24点，则需要两行才能完成，以此类推。

在横向方面，以LQ-1500打印机为例，图形打印方式下满行共有2448列，如图7-2所示，可装载的点素有3×2448＝7344个。

按照24×24点阵汉字打印的话，每个汉字占用24列，字与字之间无间隔，一行最多印打 102 个汉字。由于目前计算机与打印机接口中只有 8 位数据线，因此，把一个汉字点阵数据的24列，按高、中、低三个字节送到打印机的一列中，一列中先送入的放在最上方的 8 根针，第二个字节送到当中的 8 根针，第三个字节送

到下面的8根针。如图7-1的方式，再送第四个字节时，就送到下一列最上方的8根针，如此继续送入，直到一行汉字全部点阵数据填满，或遇到回车换行指令时为止。

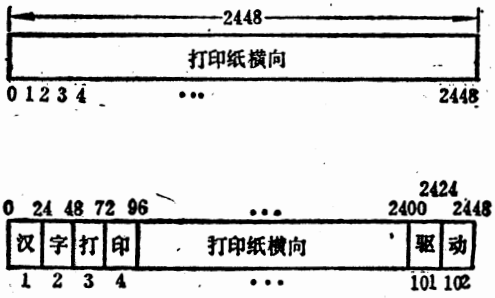


图 7-2 打印机横向排列

7.2 汉字打印变倍方法

汉字字形的变倍，通常有逻辑方程插入法、曲面插入法、分式变换法和直接放大法等。在以微机为基础的汉字信息处理系统中，对汉字库中存储的字形常采用直接放大法，放大的字形比较粗糙。

这种直接放大法的基本思想是根据汉字库中存储的基本字形和要放大的倍数（一般是整数倍），直接交换出放大后的字形数据，然后供显示或打印输出。

例如下面的一段简单程序，就是直接放大法的核心程序。

```
        MOV  AH, [SI]     ; 取一个点阵字节
        MOV  CX,  8       ; 循环计数
Lops:   RCL  AH,  1       ; 高位移进CY标志
        RCL  DX,  1       ; CY内容移入DX
        MOV  BX,  DX      ;
        RCR  BX,  1       ; BX低位移入CY
        RCL  DX,  1       ; DX取得2位
```

```
        LOOP  Lops          ；8次共得16位
        MOV   [DI]，DX      ；存入目的缓冲区
```

该程序的主要功能是把原缓冲区DS:SI处取出的一个点阵字节，按位扩展一倍，在DX中形成两个字节后，放入目的缓冲区DS:DI。

放大扩展时，可先在内存中开辟16×16和32×32两个缓冲区，或者其他点阵形式的缓冲区。前一个用来存放基本点阵字模，后一个用来存放扩展放大后的点阵字模，按行（或列）读取基本字模缓冲区中的数据，放大后依次写到放大缓冲区。譬如，AH寄存器原字节为25H，通过上述程序放大处理后，DX寄存器中将得到放大了一倍的点阵字节0C33H，如下图所示。

25H　○○●○○●○●

0C33H　○○○○●●○○○○●●○○●●

7.2.1 变换运算无级变倍法

这种变换法可以对字形点阵行列元素作任意倍数的变换。假定把原来的$n\times n$点阵字形变换成新的$m\times m$点阵字形。原始字形的点阵用$\{A(i,j)\}$来表示，变换后新字形的点阵用$\{B(k,l)\}$表示，其中$i,j=1,2,\cdots,n$，而$k,l=1,2,\cdots,m$。

若元素$A(i,j)=1$或$B(k,l)=1$，表示点阵字形中相应的点为“1”；若元素$A(i,j)=0$或$B(k,l)=0$，则表示点阵字形中相应的点为“0”。结果为“1”，表示打印机针头动作，为“0”时针头不出打。

在这种变换中，由两种情况来定义变换后的字形是放大还是缩小。若$n<m$，其变换结果为放大，若$n>m$，则为缩小。

(1) $n<m$的情况

用$[a]$表示a的整数部分，用$[a/b]$表示a/b的余数部分；

设：$\left[\frac{nk}{m}\right]=p$，　$\left[\frac{nl}{m}\right]=q$

$$\begin{cases} A(p,q) = A_1 \\ A(p+1,q) = A_2 \\ A(p,q+1) = A_3 \\ A(p+1,q+1) = A_4 \end{cases}$$

$$f_k = \min(n, m-(nk)m)$$

$$f_e = \min(n, m-(nl)m)$$

使得：

$$b(k,l) = f_k \cdot f_1 \cdot A_1 + (n-f_k)\cdot f_1 \cdot A_2 + f_k(n-k)\cdot A_3 + (n-f_k)(n-f_1)\cdot A_4$$

这样一来，$b(k,l)$就表征了新的点阵字形中与$B(k,l)$相应的点阵S的黑白程度，换句话说，$b(k,l)$的值表示了与$B(k,l)$相应的点阵S对“1”的隶属程序。由于$A_i(i=1,2,3,4)=0$或1，显然，不等式$0 \leqslant b(k,l) \leqslant n^2$成立。这样便得到了一个$m \times n$的矩阵$\{b(k,l)\}$，其中所有元素的值都在$0-n^2$之间，若选取一个适当的阈值，则可以得到字形点阵$\{B(k,l)\}$。

(2) $n>m$的情况

在这种情况下，不能简单地直接使用$n<m$时的方法，因为，此时$\{B(k,l)\}$中的9个元素的值可能和$\{A(i,j)\}$中的每个元素值有关，从而将易于使原来分离开的笔划被粘连。为了改善这种情况，先引入一个定义。

定义：当$n>m$时，在点阵字形中与元素$\{B(k,l)\}$相应的点阵S中划一个小方格S_1，使S_1的中心与S的中心重合，S_1的面积等于S面积的$(m/n)^2$倍，那么，S_1叫做S的特征方格，如图7-3所示。

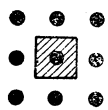


图 7-3 点阵元素

设：$\left[\frac{nk}{m}+\frac{n-m}{2m}\right]=p$，$\left[\frac{nl}{m}+\frac{n-m}{2m}\right]=q$

那么，$A_i(i=1,2,3,4)$的定义如前。

$$f_k = m-\left(nk+\frac{n-m}{2}\right)m$$

$$f_l = m - \left(nl + \frac{n-m}{2}\right)m$$

使得：

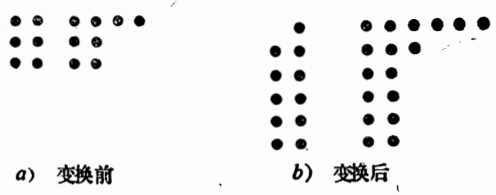
$$b(k,l) = f_k \cdot f_l \cdot A_1 + (m - f_k) \cdot f_l \cdot A_2 + f_k(m - f_k) \cdot A_3 + (m - f_k) \cdot (m - f_l) \cdot A_4$$

式中，$b(k,l)(0 \leqslant b(k,l) \leqslant m^2)$ 的值就表征了与 $B(k,l)$ 相应的小方格 S 的特征方格 S_1 的黑白程度。和 $n<m$ 的情况相似，$b(k,l)$ 的值在 0—m^2 之间，如果再选择一个适当的阈值，就可以求得点阵 $m \times m$ 中的全部字形点阵 $\{B(k,l)\}$。显然，在这种情况下的阈值，相对说来，总是小于 $n<m$ 情况下所选取的阈值。

7.2.2 变换字形的整形和补偿

直接由上述的变换方法所得到的点阵字形，有可能在直线段的端部和交叉处产生变形，因此需要整形。

这种变形如图 7-4 所示，是指在直线段的端部本来应该为“1”的元素变成了“0”，而在直线交叉处本来应该为“0”的元素变成了“1”。以下可采用局部逻辑判断的办法进行整形。



a) 变换前　　b) 变换后

图 7-4　变换的线段产生的变形

当满足一定的逻辑条件时，使 $b(k,l)$ 的值加上一个修正项后，将能消除这种在直线段端部和交叉处产生的变形。

(1) $n<m$ 的情况

当

$$\begin{cases} (nk)m > m - n \\ (nl)m > m - n \end{cases}$$

若

$$A_1\cdot\bar{A}_2\cdot\bar{A}_3\cdot\bar{A}_4\cdot\bar{A}(p-1,q+1)\cdot\bar{A}(p+1,q-1)=1 \quad (1)$$

或 $A_1,\bar{A}_2,\bar{A}_3,\bar{A}_4,\bar{A}(p,q+2),\bar{A}(p+2,q)=1$

则

$$\Delta_1=\min[(n-m+(nk)_m)(m-(nl)_m)\cdot(m-(nk)m)(m-(nl)_m)]$$

若

$$\left.\begin{aligned}&\bar{A}_1\cdot\bar{A}_2\cdot\bar{A}_3\cdot\bar{A}_4\cdot\bar{A}(p,q-1)\cdot\bar{A}(p+2,q+1)=1\\ \text{或}\quad&\bar{A}_1\cdot\bar{A}_2\cdot\bar{A}_3\cdot\bar{A}_4\cdot\bar{A}(p-1,q)\cdot\bar{A}(p+1,q+2)=1\end{aligned}\right\} \quad (2)$$

则

$$\Delta_2=\min[(n-m+(nk)_m)(n-m+(nl)_m)\cdot(m-(nk)_m)(m-(nl)_m)]$$

修正项 Δ_1、Δ_2 将使与直线端部相应的那些 $b(k,l)$的值增加，从而阻止本应该为“1”的元素变为“0”。

如果

$$\left.\begin{aligned}&\bar{A}_1\cdot A_2\cdot A_3\cdot A_4[A(p-1,q+1)\cdot A(p+1,q-1)\\&+A(p-1,q+1)\cdot\bar{A}(p+2,q)+A(p+1,q-1)\\&\cdot\bar{A}(p,q+2)]=1\\ \text{或}\quad&A_1\cdot A_2\cdot A_3\cdot\bar{A}_4[A(p,q+2)\cdot A(p+2,q)+A(p,q+2)\\&\cdot\bar{A}(p+1,q\ \ 1)+A(p+2,q)\cdot\bar{A}(p-1,q+1)]=1\end{aligned}\right\} \quad (3)$$

则有 $$\Delta_3=-\Delta_1$$

如果

$$\left.\begin{aligned}&A_1\cdot\bar{A}_2\cdot A_3\cdot A_4[A(p,q-1)\cdot A(p+2,q+1)+A(p,q-1)\\&\cdot\bar{A}(p+1,q+2)+A(p+2,q+1)\cdot\bar{A}(p-1,q)]=1\\ \text{或}\quad&A_1\cdot A_2\cdot\bar{A}_3\cdot A_4[A(p-1,q)\cdot A(p+1,q+2)+A(p-1,q)\\&\cdot\bar{A}(p+2,q+1)+A(p+1,q+2)\cdot\bar{A}(p,q-1)]=1\end{aligned}\right\} \quad (4)$$

则有 $$\Delta_4=-\Delta_2$$

修正项 Δ_3、Δ_4 将使与直线段交叉处相应的那些 $b(k,l)$的值减小，从而阻止本来应该为“0”的元素变成为“1”。

(2) $n>m$ 的情况

此时，只要把 $n<m$ 的情况的计算公式加以适当改变即可运用。当然，相应条件

$$\begin{cases}(nk)m>m-n\\(nl)m>m-n\end{cases}$$

就不再需要了。

如果式(1)成立，则

$$\Delta_5=\min\left\{\left(nk+\frac{n-m}{2}\right)m\cdot\left[m-\left(nl+\frac{n-m}{2}\right)m\right],\right.$$
$$\left.\left[m-\left(nk+\frac{n-m}{2}\right)m\right]\left(nl+\frac{n-m}{2}\right)m\right\}$$

如果式(2)成立，则

$$\Delta_6=\min\left\{\left[m-\left(nk+\frac{n-m}{2}\right)m\right]\cdot\left[m-\left(nl+\frac{n-m}{2}\right)m\right],\right.$$
$$\left.\left(nk+\frac{n-m}{2}\right)m\cdot\left(nl+\frac{n-m}{2}\right)m\right\}$$

如果式(3)成立，则 $\Delta_7=-\Delta_5$；

如果式(4)成立，则 $\Delta_8=-\Delta_6$。

由此可见，判别整形条件和进行修正量 Δ 的计算是比较费事的。因此，在实际计算时，必须考虑到机器的执行时间，特别在 m 比 n 大得较多的情况下所需要执行的时间。同时满足条件 $(nk)m>m$ 和 $(nl)m>m-n$ 的 $b(k,l)$ 只占全部 $b(k,l)$ 的一小部分。再注意到只有在 $\sum_{i=1}^{4}A_i=1$，而 $b(k,l)$ 的值尚小于阈值，或 $\sum_{i=1}^{4}A_i=3$，而在 $b(k,l)$ 的值已大于阈值的情况下，才需要进行整形条件的判别。整形后的字形，可以作到局部修补的效果。

7.2.3 笔划平滑放大法

当汉字的数据点阵被放大后，由于是数字信息，因此很容易产生锯齿效应。这可以采用平滑补偿或对原汉字点阵从左至右、从上至下逐点进行判断的方法进行修正。

这种笔划平滑的基本思想是：如果该点为一笔划的起点、落点、拐点（亦称为边际点），即进行与之相应的边际点矩阵转换，倘若该点为非边际点（例如横、竖中间的点），则只须用相应的全“1”点矩阵进行转换。实际上，对汉字字模进行分析、筛选，可选取15种相对位置的边际或拐角等字模的补偿点阵。把这15种补偿点阵作在处理程序中，用它们作为被放大后的汉字边际或拐角以及引起失真的笔划贴补之用。这种方法作为倍数放大的平滑处理是比较简单的，若作为无级变倍则处理的速度比较慢。

下面以24×24点阵汉字为基础，给出一些算法，供读者参考。该算法平均可以覆盖一个汉字90%左右的相对位置状态。

此法简单分为以下4个步骤：

（1）取一个汉字中一个字节信息，将其转换为点阵信息，以每个元素进行处理，经循环判别，完成对一个完整汉字的处理操作。下面的Pascal程序段，是以24×24点阵的汉字字模为基础的：

```
begin
  for J: = 0 to 23 do
    for I: = 0 to 2 do
      for N: = 0 to 7 do
        if Ord(CCdata[J * 3+I]) and Mask[K]>0 then
          OldDot[I * 8+k, J]:= 1;
        else
          OldDot[I * 8+k, J]:= 0;
end;
```

在程序中：CCdata数组用于存放原汉字的72个字节信息，OldDot数组为原汉字的点阵元素，也就是说24×24点阵的“0”和“1”矩阵，而Mask数组存放了相应的屏蔽字，其值分别是128、64、32、16、8、4、2、1。

（2）对原始汉字点阵，以放大倍数为比例因子，逐点进行放大转换，这一步的核心就是逐点转换的判断过程。

（3）将放大矩阵转换为字节信息，以转换成供打印机输出的

字节数据为例，其Pascal语言的表述形式为：

```
M: = 0;
for K: = 1 to NewLine div 24 do
  for J: = 1 to Newcol do
    for I: = 1 to 3 do
    begin
      M: = ((K-1)*3+I-1)*8;
      for L: = 0 to 7 do
        ChgByte[K,I,J]: = ChgByte[K,I,J]*2+NewDot[M+L,J];
    end;
```

其中，NewLine和Newcol分别是放大汉字点阵形成的矩阵行、列长度，ChgByte数组用于存放转换后的字节信息，NewDot数组就是被放大了的汉字点矩阵。

（4）最后，可将ChgByte数组中存放的字节信息传送到指定的汉字字模缓存区中保存，以此信息可直接送到屏幕显示、或打印机输出这一被放大了的信息。

7.2.4 表检索无级变倍法

无论是汉字点阵，还是打印机的针头排列映像，都可以视为一个二维矩阵。尤其对打印机的数据缓冲区，虽然横向的宽度是有限的，但是其纵向可趋向无限。从行与行之间来看，其基本单元为3个字节24根针。如果以打印机点位来进纸，可以看作以点位为基本单元。因此，可以把一行的打印机针头排列映像为一个数组阵列，如果考虑在纵向可任意扩展的话，这个矩阵在纵向是连续的。根据汉字变化的大小而延伸，那么，对于这个矩阵，可由以下表达式定义：

$$D=\begin{bmatrix} u_{11} & u_{12} & \cdots & u_{1m} \\ u_{21} & u_{22} & \cdots & u_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ u_{n1} & u_{n2} & \cdots & u_{nm} \end{bmatrix}$$

其算式为：

$$D=\sum_{i=1}^{n}\sum_{j=1}^{m}u_{ij}$$

其中　　$u_{ij}\in[0,1],(i=1,2,\cdots,n \quad j=1,2,\cdots,m)$

相对于汉字点阵来说，其字模信息横向和纵向缩放的最终结果均应该满足上述表达式，也就是说，它们之间有着一一对应的关系。

倘若汉字点阵需要纵向的扩展处理时，该数组的列数不变，行数将以倍数形式增加。一行可以打印多少个汉字，由定义的汉字当前点阵和该矩阵的最大列数来限定。如果指定汉字横向缩小为12点的话，一行最多可以打印 2448/12 = 204 个汉字，ASCII字符的输出横向为汉字的一半，可打印 408 个。

针对上述打印机的数组排列，可在驱动程序中设计两个纵横向变换表，把X定义为横向缩放表，Y定义为纵向缩放表，其矩阵表达式如下：

$$X=\begin{bmatrix} x_{11} & x_{12} & \cdots & x_{1m} \\ x_{21} & x_{22} & \cdots & x_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ x_{n1} & x_{n2} & \cdots & x_{nm} \end{bmatrix}$$

$$Y=\begin{bmatrix} y_{11} & y_{12} & \cdots & y_{1m} \\ y_{21} & y_{22} & \cdots & y_{2m} \\ \cdots & \cdots & \cdots & \cdots \\ y_{n1} & y_{n2} & \cdots & y_{nm} \end{bmatrix}$$

X、Y这两个表与D数组形成一种不确定的关系，换句话说，是一种动态的不稳定的对应关系，其表征D数组与字形变换的关系为：

$$D=(c+a)\cdot(X\blacktriangle Y)$$

其中，c为汉字系数，a为ASCII码系数，▲为模糊运算，字形变换的基本算法如下描述：

横向放大：$X_{ml}=N\cdot(x_{ij}\blacktriangle y_{ij})$

纵向放大：$Y_{nd}=(y_{ij}\vee b_n\vee\vec{y}_{ij})\blacktriangle x_{ij}$　　$(n=1,2,\cdots,8)$

横向缩小：$X_{mx}=(x_{ij}\vee x_{i+1,j})\blacktriangle y_{ij}$

纵向缩小：$Y_{i*}=(\vec{y}_{ij}\vee\vec{y}_{ij+1})\blacktriangle x_{ij}$

式中，$\rightarrow$表示移位运算，$\vee$为逻辑“或”，b_n则表示对应放大的那一点位。

两组缩放表的数组元素是由程序员定义的。实际上是由若干个判别码或者操作代码所组成的。通过应用程序中所指定的字形变换控制码，由程序内部两个方向元素的索引程序，来转换和定位缩放表的行、列数组指针以及表单元地址，并用以确定两表中的x_{ij}、y_{ij}的交并逻辑。然后，根据表项内容，决定字符或汉字的缩或放以及缩放的点阵形式。最后，按点位由上述的算法来实现汉字字形的变换。

程序的基本设计思想是串行的按点位操作。针对打印机阵列的24点位（3个字节），则是以其上下左右对称的方式进行压缩和扩展，这样，可以把失真效应降到最低程度。图7-5简单地描述了这种以点位（1字节）方式缩放的形式。

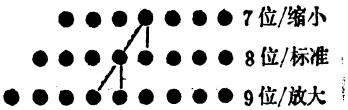


图 7-5　点位缩放方式

如前所述，每种汉字字形是由水平点数和垂直点数决定的，并且通过算法，把对应的点阵字节顺序按打印机针头的排列送入打印缓冲区或者是指定的数据区内。以24×24点阵的处理为例，以3个字节为单元，取出一个字节后，根据当前控制码标志状态字节到两个缩放表中检索数组表元素。先是以纵向定位处理为基准，其实现的过程是：把一个字节循环处理8次，8个点位，即3个字节共需循环查表24次。当某一个字节在循环中遇到放大标志，也就是说纵向大于24点阵，就在字模加工区内对表指针所指的那一点位进行左移位运算，以致把那一点处理两次扩展为两点或更多的点，并把移入标志位的那一点移至下一个字节的低位。

在查表过程中，如果遇到字形控制码标志是缩小标志，就把对应表指针的那一点位进行右移运算，并和下一位相邻的那一点

位相“或”，同时，把下一个字节的低位移入该字节的高位。以后的处理都必须根据这个缩小标志，对每一个点位都要进行移位操作，直到3个字节循环处理完毕。这时，在缓冲区中就形成了纵向变换过的点阵列，然后再根据横向缩放标志进行处理。若缓冲区对应打印机是满行阵列，便可以满足字底对齐的打印，即一行中有纵向放大也有纵向缩小的情况，且需要分行打印。不过，这种方式需要较大的内存开销。如果按一个汉字开辟缓冲区时，必须保存一行的所有汉字机内码，也可做到分行输出。纵向实际缩放表由下列汇编语言的表格式给出。

```
FONT_2  LABEL WORD          ;纵向变倍表

DW  OFFSET  FONT_2_14
DW  OFFSET  FONT_2_15

DW  OFFSET  FONT_2_23
DW  OFFSET  FONT_2_24
DW  OFFSET  FONT_2_25
DW  OFFSET  FONT_2_26
DW  OFFSET  FONT_2_27

DW  OFFSET  FONT_2_39
DW  OFFSET  FONT_2_40
DW  OFFSET  FONT_2_41

FONT_2_14   DB  0,0,1,0,0,1,0,0,1,0,0,1,0,0
FONT_2_15   DB  1,1,0,0,1,0,0,1,0,0,1,0,0,1,0

FONT_2_23   DB  1,1,1,1,1,1,1,1,1,1,1,1,0,1,1,1,1,1,1,1,1,1,1
FONT_2_24   DB  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1
FONT_2_25   DB  1,1,1,1,1,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,1,1,1
FONT_2_26   DB  1,1,1,1,1,1,1,2,1,1,1,1,1,1,1,2,1,1,1,1,1,1,1
FONT_2_27   DB  1,1,1,1,1,2,1,1,1,1,1,1,2,1,1,1,1,1,2,1,1,1,1

FONT_2_39   DB  1,2,2,1,2,2,1,2,2,1,2,1,2,1,2,1,2,2,1,2,2,1,2,2
FONT_2_40   DB  1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2
FONT_2_41   DB  2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,1,2,2,2
```

从该表中就很容易理解程序的处理过程，表中OFFSET是数组表项指针。因为篇幅有限，这里仅列出了几种点阵的元素，

以便说明问题。

FONT_2_24 是24×24点阵标准处理方式，表元素都是1，表明程序对点位进行一次处理。如果指针定位在25点，说明是放大操作，在其表元素中遇到2时，就指定程序对该点位进行2次处理，即扩展为2点。当有缩小标志时，程序对于表元素为0的，作与相邻点位的“或”处理。这类处理，用汇编语言是不难解决的，若是用C语言就更容易实现了。

该算法的横向缩放，原理与纵向基本类似，只不过不像纵向那样有固定的3字节长度单元。因此，横向表的数组元素不参与程序运算，仅作为字形变换的处理标志。如下列的汇编语言程序表所示：

```
FONT_1  LABEL WORD          ;横向变倍表(4为结束码)
 DW  OFFSET FONT_1_12       ;12点
 DW  OFFSET FONT_1_13

 DW  OFFSET FONT_1_23
 DW  OFFSET FONT_1_24       ;24点(标准)
 DW  OFFSET FONT_1_25
 DW  OFFSET FONT_1_26

 DW  OFFSET FONT_1_47       ;47点
 DW  OFFSET FONT_1_48

FONT_1_12   DB  3,3,3,3,0,0,0,0,3,7,0,0,0,0,7,0,0,0,0,4
FONT_1_13   DB  3,3,3,3,1,0,1,0,0,3,7,0,0,0,0,7,0,0,0,0,4

FONT_1_23   DB  3,3,3,1,1,1,1,1,1,1,3,1,7,1,1,1,1,0,1,1,3,7,6,4
FONT_1_24   DB  3,3,3,6,6,6,4
FONT_1_25   DB  3,3,1,3,1,1,1,1,1,1,1,7,1,3,1,1,1,2,1,1,1,3,7,6,4
FONT_1_26   DB  3,3,1,1,3,1,1,1,1,1,2,7,1,3,1,1,1,1,1,1,1,7,5,1,1,1,1,1,1,1,4

FONT_1_47   DB  2,2,2,5,2,2,2,5,7,2,2,2,1,3,2,2,2,2,3,7,2,2,2,2,3,2,2,2,2,4
FONT_1_48   DB  2,2,2,2,3,2,2,2,2,3,7,2,2,2,2,3,2,2,2,2,3,7,2,2,2,2,3,2,2,2,2,4
```

在循环检索表项过程的同时，把加工缓冲区的一列点阵信息送入打印缓冲区或者是指定的缓冲区。若是标准字形，就只须传送一次即可。如果有缩小标志被获得，就把当前的一列点阵和随后的一列点阵按位相“或”后再送入打印缓冲区。若有放大标志

时，则相应的点列由程序按位向打印缓冲区内送入两次。

由于在一行内可打印任意大小的汉字字型，因此，处理每一个汉字时都需要对数组的表项进行检索，其方式不是静态的，亦非对称和固定的，而是模糊的，不确定的，通过上述给出的一些表达式逻辑算法来定义。

7.3 汉字打印驱动程序

汉字打印驱动程序，相对于整个汉字系统而言，是一个独立的模块，作为BIOS的扩充和升级来与内存、磁盘和打印机端口发生联系。应用程序是通对17H类中断的调用，以图形方式打印出汉字信息的。

汉字打印的过程是把指定的汉字文本从外存调入内存的某一区域，其内容为汉字机内码和ASCII代码的集合。当打印输出时，从这个内存区域内逐个取出汉字内码或ASCII代码，由打印驱动程序把这些内码转换为地址码或文件指针，然后再从汉字库中取出汉字的点阵字模。若是从16×16点阵汉字库中取出的字模，其点阵排列是横向信息，因此，还需要通过字模转换程序把它转换为对应打印机针头排列的纵向点阵信息。若是从24×24点阵汉字库中取出的字模，当控制命令未指定其旋转时，其点阵信息本身就是纵向排列的。32×32点阵以上的高点阵汉字，就需要分行进行打印。通过驱动程序的有关功能，把汉字的纵向点阵字节信息送入打印数据缓冲区，待该缓冲区送满了，或者程序收到了回车换行命令，则向打印机发出图形打印命令，并把打印数据缓冲区中的内容通过端口送向打印机。这时，打印机是在图形方式下，根据指定的打印控制命令，在打印纸上输出各种字体、各种字形的汉字或ASCII字符来。

打印机输出汉字的另一途径是屏幕硬拷贝，即把当前屏幕上所显示的内容在打印机上输出。屏幕硬拷贝的过程是把视频缓冲区中的横向图形点阵信息由程序转换为打印机对应格式的数据，

然后向打印机发出图形打印命令，再把转换过的信息送往打印机，即可实现屏幕信息的硬拷贝。

7.3.1 环境设备的兼容性

虽然计算机的硬件设备种类繁多，但在软件设计时，应该尽量考虑它们的兼容性，这样，可以避免重复劳动和由过多的同种类型的程序模块占用存储资源。

早期的汉字系统配备的改字打印程序，大多是针对一种微机或一种指定的打印机的。随着汉字处理技术的发展，都采用通用性较强的系统。若要尽量做到系统的兼容，可以从以下几个方面考虑。

（1）对于高点阵的打印驱动程序来说，其汉字点阵字模是从磁盘上读取的，因此，要做到不依赖于 CC-DOS 的版本，也不依赖任何应用程序。只要系统是 Intel 系列的 CPU，字节高8位为“1”的两码汉字系统，均可运行。这就要求编制打印程序时，避免调用CC-BIOS 的功能，而是各种功能由打印程序独立完成。否则，更换一个不同版本的 CC-BIOS，将会失去原有的调用功能，致使用户缩小了系统的选择范围。

但是，当从磁盘上读取汉字点阵字模时，如果用文件读取的方法，将存在一个与 DOS 系统嵌套调用的问题，解决得不好，在打印过程中将会引起系统死锁，使得系统无法继续运行。这个问题将在下一节详细讨论。

（2）打印口的兼容。汉字打印程序仅把汉字信息处理为适应打印机图形打印的字节，为简化程序，向打印机发送字节信息和控制代码，仍然需要调用原 ROM-BIOS 的 17 H 类中断模块。INT 17H 的入口地址在中断向量表中是由 0:005C—0:005F 4个字节给出的。0:005C、005D指向BIOS 打印程序偏移首地址，005E、005F 指向它的段地址。但是，IBM 系列的微机与一些异种机（如5550、North Star 和网络服务器等）的中断入口地址是不相同的，若把该入口地址以常量给出，那么，在其他微机上

就不能使用。

为了能够兼容使用，可以用下面的一段程序来解决该问题。

```
PRINT PROC NEAR
      XOR       AH,AH
PO    PROC      NEAR
      PUSH      DX
      XOR       DX,DX
      PUSHF
      CALL      DWORD PTR CS:INT17OFF
      MOV       CS:P_STATS,AX
      POP       DX
      RET
PO    ENDP
  PRINT ENDP
```

该程序的功能是在装载打印程序时，把BIOS中断17H的段地址和偏移地址取出，存放在内存单元INT 17 OFF中，当需要向打印机发送信息时，用一个长调用指令指向该内存单元即可实现。

(3) 打印机的通用。由于生产单位不同，各种型号的打印机，其控制命令均不统一，为使得各种针式打印机都能通用一个程序，可以建立一个打印机控制命令的参数文件。在该文件中，预先根据各种打印机手册中给定的控制命令，把打印机的名称、写点方式（即 d_0—d_7 写点或 d_7— d_0 写点、3×8 排列还是 4×6 排列、9 针或16针等）和一些与汉字打印有关的控制码，存入该文件指定的区域。该文件用程序编辑写入参数也可、用 Debug PCTOOLS 直接写入也可；要给出将来扩充的方式。一般，与汉字打印有关的控制命令包括：设置图形打印方式、行距、单/双向打印和一行的最大点阵列极限等。当然，控制命令越多，功能越强。但要注意，控制命令太多，以后再追加时，用户容易厌烦，所以说要作到恰如其分。如图 7-6 所示。

图 7-6 中，每个打印机类型，占用了50个字节的参数单元。

```
DS:100  4C 51 31 35 30 30 00 00-00 00 20 1B 2A 27 24 00  LQ1500.... .*'$.
DS:110  00 00 00 1B 33 18 24 00-00 00 00 1B 55 01 24 00  ....j.$.....U.$.
DS:120  00 00 00 1B 55 00 24 00-00 00 00 01 00 12 90 09  ....U.$.........
DS:130  90 09 4D 33 30 37 30 00-00 00 00 00 10 1B 49 24  ..M3070.......I$
DS:140  00 00 00 00 00 1B 57 30-30 31 38 24 00 1B 3C 24  ......W0018$..<$
DS:150  00 00 00 00 00 1B 3E 24-00 00 00 00 00 01 00 12  ......>$........
DS:160  90 09 90 09                                      ....
```

图 7-6 打印机参数文件表

对应这50个字节单元，在打印驱动程序中，也相应地列出一组参数单元，其字段分配完全与该表相同，如下列汇编程序的定义：

```
PRINT_PRAM:
    Print_model   DB   "LQ1500",0,0,0                ; 打印机类型
    Image_mode    DB   20H                           ; 针排列标志
    Bit_image     DB   1BH,2AH,27H,24H,0,0,0,0       ; 图形打印码
    Line_space    DB   1BH,33H, 18H,24H,0,0,0,0      ; 行距控制码
    Bi_print      DB   1BH,55H, 01H,24H,0,0,0,0      ; 单向打印码
    Uni_print     DB   1BH,55H, 00H,24H,0,0,0,0      ; 双向打印码
                  DB   2    DUP(0)                   ; 标志码
    Asc_len       DB   12H
    Lmax          DW   0990H                         ; 一行最大点列
                  DW   0990H
```

在这组单元中，每个字段约8个字节，因为各种打印机控制码的长度不同，因而定义的单元充足一些。在控制码之后，用24H（即$字符的代码）作为结束码，以便于程序的控制。当然，也可以用别的代码作为结束码。例如，在发送某一控制码之前，先由DS:SI指向其字段首地址，LEA SI,Bi_Print（单向打印），然后调用下列 Pcode 子程序，该子程序把装入AL的控制码顺序送入打印机（调 Print 模块），直到AL等于24H，便停止发送。

```
Pcode:
        LODSB
        CMP     AL,'$'
        JE      Pret
```

```
        CALL    Print
        JMP     SHORT Pcode
Pret:
        RET
```

当装载打印驱动程序时，可在文件名之后输入指定的打印机名称，装载程序将根据该名称到打印机参数表文件中去查找，若输入的名称在该文件中未找到，程序退出，不予装载，或者默认以支持某种型号的打印机。如果找到了，就把从该地址开始的50个字节的参数，传送到内存参数单元去，再去作其他装载处理。因此，无论将该参数表文件扩充到多大，最终装载到内存的内容只有这50个字节。

当然，这种方式还可以进一步优化，在汉字打印过程中即可有效地控制各种型号的打印机，克服了一个程序仅支持一种打印机的缺点。

7.3.2 数据结构和编程

不同的设计方案，将会有不同的数据结构。对于汉字打印驱动程序来说，其数据结构的安排主要分4个部分：①磁盘传输缓冲区(DTA)。用于放置从磁盘读入的高点阵汉字字模信息。该缓冲区的大小，可以根据不同的汉字库所定义的一个汉字点阵字节数的大小而定；②点位变换操作区，亦称为点位加工区。该区域视最大扩展极限以3个字节（打印机的一列针头）的倍数来定义。把DTA区内的字模点阵取出，在加工区内按点位元素进行缩放变换后，顺序送入打印缓冲区；③打印缓冲区。这个缓冲区的结构是和打印机针头的排列相一致的。当程序遇到回车换行等控制码时，就把在打印缓冲区内的点位字节送入打印机输出；④内码缓冲区和标志单元。存放一些代码和控制程序走向的标志。

打印程序的设计需要考虑的细节很多，这里仅以主要功能提出一些步骤和方法。

1. 主程序

通常，当程序进入打印模块后，根据入口参数转到不同的子功能处理。最主要的功能是0号功能，它完成字符和汉字的打印输出，其他功能均可以调用原ROM-BIOS中INT 17H的服务模块来完成。主控程序的流程如图7-7所示。

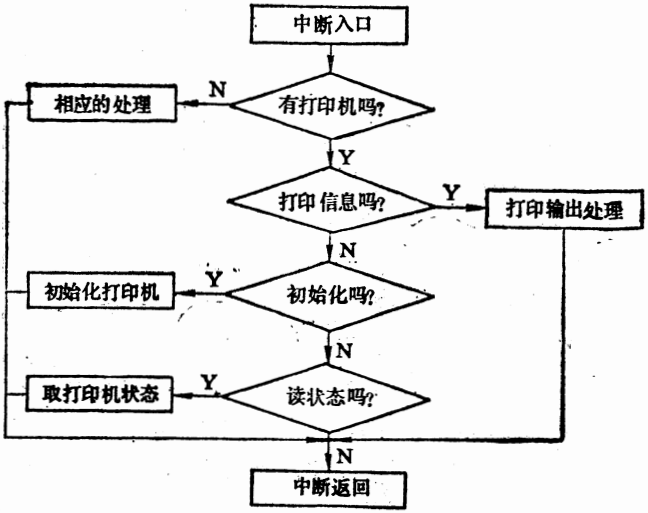


图 7-7　打印主控流程

0号功能也就是打印信息功能，是17H类中断处理程序中最重要的模块，为了使打印机输出汉字，汉字驱动程序主要是扩充了该模块的功能。由DX寄存器指定打印机号，AL寄存器为输出字符或汉字机内码，然后发INT 17H指令即可。

在该功能模块中，根据要求，分为许多子程序。例如：回车换行的处理、ASCII码和汉字内码的分别处理、打印机控制命令的处理等。一般，在打印机接收到换行指令(0AH)时，才把一行信息打印。同样，设计汉字打印驱动程序时，当接收到换行指令时，才向打印机发出传送数据的操作。在其他方式下进入程序后，一旦处理完毕，必须保障程序顺利地中断返回。

汉字与ASCII码的判别，一般以代码大于A0H为分支。小

于A0H的代码或小于80H的代码（有的程序把80H—A0H范围内的代码也定义为程序的控制标志）作为ASCII码处理，根据要求，可以在磁盘上装入不同字体的ASCII字符的字模库，如标准、粗体、斜体等。由控制命令来指定。也可以直接调用打印机内部ROM发生器内的ASCII字符，视情况作不同的处理。

代码大于A0H的，作为汉字机内码处理，通常，需要接收到两个机内码才作为汉字处理的依据。由控制命令指定所要输出的汉字字体，分别从磁盘上的宋体、楷体、黑体、仿宋体和繁体字库文件中读取汉字点阵信息，进行处理。

无论是ASCII字符还是汉字，除了用户指定外，一般有一种默认的字形，若在应用程序中指定了打印输出字形的大小，就要通过倍数变换或者无级缩放，对点阵信息进行相应的处理，然后送入打印缓冲区。有时，打印的汉字的放大倍数较大，就可能需要分几行才能把完整的信息打印出来。这要求每行之间需插入一个回车换行命令。但是，送回车换行或其他的控制命令时，打印机必须是在打印字符状态。否则，打印机会把送给它的控制命令作为图形信息打印出来，而控制码却失效。那么，怎样才能使打印机正确地接收点阵信息，以图形方式打印，而控制却以字符方式打印呢？下面以LQ-1500打印机为例加以说明。这种24针打印机有好几种图形打印功能，通常选择3倍密度的方式，其控制命令为：

$$ESC * m\ n_1\ n_2\ d_1\ d_2\ \cdots d_i$$

前面三个参数是图形选择码，十六进制为1BH 2AH 27H，这3个码发送给打印机后，就会设置成3倍密度的图形打印方式。n_1、n_2这两个参数用来指定横向所要打印点的数目。d_1—d_i是要打印的点阵信息。横向打点的数目为$i=3(n_1+n_2\times 256)$，n_1是横向点以256被除的余数，而n_2的值则是256被除的商

$$n_1 = （横向点）\ MOD\ 256$$

$$n_2 = INT（横向点/256）$$

为了说明问题，下面举两个例子。

(1) 送一个24×24点阵的汉字，其图形控制命令的设置为：

横向24点：$n_1 = 24 \ \text{MOD} \ 256 = 24 = 18\text{H}$

$n_2 = \text{INT}(24/256) = 0$ （取整）

命令为：1BH,2AH,27H,18H,00

(2) 若送一个图形，其横向点为300点，其设置应是：

横向300点：$n_1 = 300 \ \text{MOD} \ 256 = 44 = 2\text{CH}$

$n_2 = \text{INT}(300/256) = 1$

命令为：1BH,2AH,27H,2CH,01H

一旦设置了图形打印状态，随后，所有送入打印机的数据均被当作图形数据处理，直到接收完指定的点数目列的数据为止，打印机就会恢复正常打印字符的状态。如上述第1个例子，如果送完24列点阵数据后，再往第25列送数据时，打印机就会作为ASCII字符码处理了。

2. 控制命令与标志

在打印程序中需要设置一些标志单元，用以控制程序的走向和各种功能的实现。在打印输出的处理过程中，程序要识别字体、汉字的放大与缩小、打印背景、字形旋转和行间距等控制状态。这些功能的操作均将取决于一些标志单元的置位与否作为功能判别来实现。

在编程时，要考虑到，除了处理汉字所约定的控制命令外，还应该保留打印机原有的控制命令生效，否则，就谈不上中西文兼容了。如果在一行当中发送原来控制西文状态的命令比较难处理，起码在行首是可以解决这个问题的。有些程序是按照单个汉字处理后送入打印机的，其控制码不受图形打印方式限制，在一行的任何地方都可以发送控制码。若是按整行处理好后再送往打印机输出的设计方案，就必须解决好图形和字符打印状态的识别和转换。例如，设控制字形变换的命令是ESC，b，n_1，n_2，要输出横向为30点、纵向为45点的汉字（即30×45点阵），在应用程序中使用 Chr(27)+"b3045" 指令即可。而在文本文件中，如用WordStar，Edlin 编辑的文件，用 ^b3045 命令，随后打印出

的字符和汉字均为30×45点阵，直到发出变换其他字形的指令时为止。

当程序接收到ESC等控制码时，首先把该码保存在内存单元中再返回。如果随后接收到的第二码是本程序使用的汉字处理控制码，就置第二码标志作为汉字控制码处理。否则，就将ESC和第一码之后送来的代码一起送入打印机，用这种方法则可以保留打印机的所有命令。

3. 汉字的辅助处理

除了字体、字形的变换外，为了美化汉字打印输出的效果，有时需要一些辅助处理功能。例如旋转打印、网点背景、黑底白字、斜体、下划线和上下标等功能。质量好的汉字打印驱动程序，还具有对笔划按点加粗减细和平滑补偿功能。当然，功能越全，程序越大，所占用的内存资源亦越多，这要视应用的领域不同来权衡其利弊。

旋转90°处理，通常也称为竖行打印字。这种旋转又分为左旋和右旋，这一功能是在字形的缩放之前进行的。由于各种点阵的字模点阵的安排都不同，因此，虽然有字形旋转的标志，也要根据点阵排列的方式不同进行不同的处理。就24×24点阵为例，其汉字点阵信息在字库中的实际排列如图7-8所示。这种排列是对应着打印机针头的，即一行3个字节对应送到打印机一列24根针，黑点击针，白点不击针，将会自动打印出标准的汉字。

如果要旋转的话，就把该汉字或字符的点阵按从首行的3个字节到末行的3个字节的顺序，以3个字节为一个单元，将每行的各点位以相反次序送入预先设置的缓冲区或者堆栈中，再作其他字形变换等处理。实现这种旋转的程序在造字一节中有过描述，在此不再赘述。

对于下划线、斜体、网点背景和反白输出等功能，通常是在字形变换之后进行的。

当把变换后的字形点阵送入打印缓冲区之前测有下划线标志时，即可把一列点阵的最下面一个字节的最低位和01相“或”，

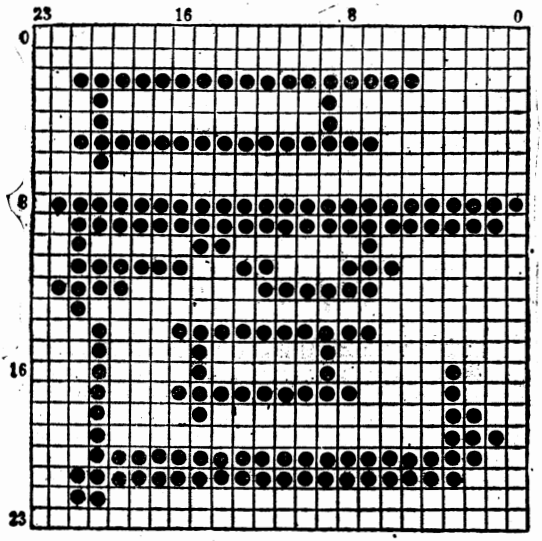


图 7-8　24×24点阵字模排列

每一列点阵均作同样处理，直到该标志清零。也可以在打印完一行信息之后，把打印机的行距只按点调整，再使打印机走一行，这样将使得下划线不会覆盖汉字的笔划，但打印的速度要比前者慢一倍。

网点背景的处理有几种方法，有的用算法实现，一般都有给定的几种不同疏密的网点字节，把每个送入打印缓冲区的字节按位相“或”，或者是网点字节移位处理后再与汉字点阵相“或”，前者打印出的汉字为斑点背景，而后者却可以获得背景为斜线或网格的效果。

反白处理较为简单，即把送入打印缓冲区的每个字节进行逻辑非：NOT AL，使得一个字节中各个点位的“1”变为“0”，“0”变为“1”，有效地实现了黑底白字的打印输出。

ASCII 码若用图形点阵方式打印的话，与汉字的比例为2:1，也可以通过变换打印出 1:1 的 ASCII 码。有时，为了提高打印速

度，通过命令转换直接打印出打印机ROM字符发生器中的ASCII码。由于打印机内部有一个列印缓冲区，其当前指针随着送入的数据不断推进，在向打印机送每个ASCII码之前，先送一个调整相对点位的控制码，来设定当前打印机列印指针，即可调整汉字与ASCII码打印的比例对齐，使得它们输出的效果相一致。

7.4 汉字打印系统死锁及其解决方法

由于高点阵的字库容量较大，无法装入有限的内存，一般，都把汉字库以文件的形式存贮在磁盘上。当打印程序需要取汉字字模时，就到磁盘的字库文件中去访问，把指定的点阵字模读取到内存中来。这个读文件的功能嵌套在BIOS中，给程序的使用带来许多隐患。

例如，有些高级打印程序，仅在应用程序中直接调用INT 17H时，才可以正常运行。但在DOS系统下用Ctrl+P键盘命令联机打印文本等操作时，就会引起系统死锁。

经过对系统的分析，Ctrl+P命令是在DOS级下的IO.SYS控制中，若要把信息由DOS系统去调用BIOS中的INT 17H打印模块输出信息，当向INT 17H发出调用之前，系统需要把当前的指针、设备字节和一些关键的参数等压入DOS内部的堆栈区内。进入BIOS的INT 17H之后，再要嵌套调用DOS的文件管理功能，系统又把当前的指针、设备字节和一些与这一次调用有关的参数再一次压入与第一次调用使用的同一个堆栈中，把第一次调用压入的内容全部覆盖。当程序从DOS返回BIOS，再从BIOS返回系统时，将寻找不到原压入栈的内容，程序的走向将无法预料，致使系统瓦解或死锁。

这一问题是高点阵汉字打印输出的关键技术之一，通常有几种办法来解决。

(1) 把汉字库存贮在硬盘的一块连续的空间位置上，用INT

13H磁盘读写功能来实现。这种方法是根据汉字内码直接计算出该汉字点阵在磁盘上的物理地址。计算方法简便，编程也比较容易。但是需要先格式化硬盘，因此，很容易把用户在建立此系统前的一些有用的信息破坏掉，且一旦系统有点问题，需要重新安装时，还得再次格式化硬盘，因此，给系统安装和使用带来极大的不便。这种方法虽然可以解决系统死锁问题，但是很少有人使用。

(2) 把字库以文件的形式由系统的拷贝命令拷入硬盘，在实现汉字点阵读取时，通过文件目录区和文件分配表(FAT)来计算出汉字点阵的物理地址(即簇号、磁道号和扇区号)，然后，再通过INT 13H中断功能来读取汉字点阵信息。这种方法给安装和使用带来很大的方便，有不少汉字系统都采用这种方法。如电子部六所推出的CC-DOS V4.0就是如此，已有不少的书籍和资料中作了详细介绍，不在此重复了。不过，采用这种方法，需要对操作系统有深入的了解，编程复杂冗长，几乎是原DOS的INT 25H的全部程序，还要考虑一个汉字点阵跨两个扇区的处理。再者，这种方法是针对个人机或单用户微机设计的。如果是网络服务器或多用户共享磁盘数据的话，因其磁盘的BPB表的格式，即0号磁道上的信息格式不同，致使无法使用。例如在3COM、Novell网络上就会发生此现象。

(3) 移动DOS栈指针的方法。这种方法编程简单，仅在原来的读取字模点阵的程序中插入十几条指令，安装使用也方便，并且也可在网络服务器中只安装一套打印系统，所有工作站的用户均可共享。

从原理上讲，在INT 17H调用DOS文件管理功能之前，首先把DOS堆栈中的内容保存起来，待调用返回后，再恢复该栈的内容。但是，若要保存堆栈的内容，除了要搞清楚DOS堆栈地址外，内存还要增加许多开销。再者，每调一个汉字点阵，都要额外地作传输几百字节的操作，也大量占用CPU周期，明显地影响了打印速度。

通过对各版本DOS系统的分析研究，操作系统通过INT 21H中的功能调用来对文件进行打开、读写和关闭操作，若是读文件操作，其最终要进入INT 25H进行磁盘操作。INT 25H的主要功能是计算文件数据在磁盘的地址，提供给底层的INT 13H使用。在INT 21H和INT 25H功能模块中，均有一个相同的设置堆栈指针的指令。当然，不同的DOS版本，该指令的形式和地址都有些不同，如表7-1所示。但有一点，DOS的版本是有限的，即使新的版本，由上述原理也可很快地找到它们。

几种 DOS 版本读磁盘入栈地址表　　表 7-1

DOS 版本	名称			
	版本标志字	入栈指令	INT 21H栈址	INT 25H栈址
Ver 2.00	AX=0002H	MOV SP,0B30	CS: 0C9F	CS: 14E2
Ver 2.10	AX=0A02H	MOV SP,0A30	CS: 0BAB	CS: 13EE
Ver 3.00	AX=0003H	MOV SP,0BA8	CS: 152D	CS: 15C6
Ver 3.10	AX=0A03H	MOV SP,0866	CS: 13A3	CS: 1469
Ver 3.20	AX=1403H	MOV SP,0886	CS: 1451	CS: 1517
Ver 3.30	AX=1E03H	MOV SP,0886	CS: 1524	CS: 15EA
Ver 4.00	AX=0004H	MOV SP,0920	CS: 17E6	CS: 18BD

在表中的入栈指令送入SP寄存器的地址值是指向栈底的。当进入MS-DOS有关的功能操作时，系统将从栈底顺序把一些必要的信息压入栈。该堆栈区域的容量较大，一般约有1KB左右，而一个调用仅压入或占用几十个字节的栈单元。因此，留有较大的空栈区可以利用。

用文件方式读取汉字点阵的方法，就是利用系统的空栈区作为基本思想。在调用DOS功能之前，把原来入栈指令所指的地址降低几十个字节，以与原压入的信息不发生冲突为宗旨。换句话说，就是使DOS再次入栈的指针指向空栈区。调用返回后，再用同样的方法抬高栈指针，恢复原来的栈地址，至此原来堆栈中所保存的信息内容没有受到任何破坏，使得程序顺利地返回操作系统。堆栈地址图如图7-9所示。

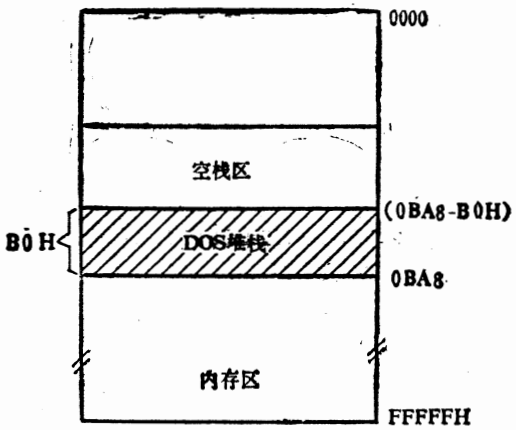


图 7-9 MS-DOS堆栈区变位图

以 MS-DOS V3.00为例，其入栈指令是 MOV SP，0BA8，即栈底地址是指向 0BA8。在读字模之前把空栈区的指针指向 0BA8-B0H 的地方即可。然后，进行读取磁盘信息的操作，如下列程序所示：

```
BEAD:
        MOV     CS:CHINA,AX             ;保存汉字内码
        MOV     CS:INTES,ES             ;保存段地址
        MOV     AX,INT25SEG
        MOV     ES,AX                   ;指向INT25段地址
        MOV     AX,00B0H
        MOV     BX,INT21OFF             ;DOS入栈指针
        SUB     ES:[BX],AX              ;降低B0H字节
        MOV     BX,INT25OFF             ;INT25栈指针
        SUB     ES:[BX],AX              ;降低B0H字节
;——读磁盘字库
        MOV     AH,1AH
        LEA     DX,DDTA                 ;置磁盘传送地址
        INT     21H
        MOV     AX,CS:CHINA             ;取出汉字内码
        MOV     DX,AX
```

```
        AND     DX,7F7FH
        SUB     DX,2121H                ;内码变换
        MOV     AL,94
        MUL     DH
        XOR     DH,DH
        ADD     AX,DX                   ;算出区位码
        MOV     SI,CLIB_NAME            ;字库FCB表首址
        MOV     WORD PTR [SI+33],AX
        MOV     DX,SI
        MOV     CX,1                    ;读一个记录(72字节)
        MOV     AH,27H
        INT     21H                     ;读点阵字模至内存
;——恢复DOS栈指针
        MOV     AX,00B0H
        MOV     BX,INT25OFF             ;INT 25栈指针
        ADD     ES:[BX],AX              ;增高B0H字节
        MOV     BX,INT12OFF             ;DOS栈指针
        ADD     ES:[BX],AX              ;增高B0H字节
        MOV     ES,CS:INTES
        LEA     SI,DDTA                 ;SI指向字模缓冲区
        RET
```

程序中给定的段地址，应该指向 INT 25H 的段地址，用堆栈的方法编程时，尤其是程序装载模块，可采用 DOS 测版本号的功能来设置空栈地址单元。这些单元用来作为读取汉字时的地址变换数据，其驱动程序的装载模块部分有关程序如下所示：

```
LOAD:
        MOV     AX,3517H            ;取原INT 17向量
        INT     21H
        MOV     INT17OFF,BX         ;保存向量地址
        MOV     INT17SEG,ES
        MOV     AX,3525H            ;取INT 25向量
        INT     21H
        MOV     INT25SEG,ES         ;保存INT 25段址
        MOV     AH,30H
```

```
        INT     21H             ; 测试MS-DOS版本
        CMP     AX,0003H
        JNZ     VER2_1          ; Ver 3.0 ?
        MOV     AX,152DH
        MOV     INT21OFF,AX     ; 存INT 21入栈指针
        MOV     AX,15C6H
        MOV     INT25OFF,AX     ; 存INT 25入栈指针
        JMP     VERTU
VER2_1:                         ; 其他版本测试
VERTU:
        INT     27H             ; 贮留退出
```

也许装载程序判别指令较多，但这些指令并不驻留在内存中，所以，装载程序再长，也只是执行完就退出，与内存开销无关，这种方法要比前两种简便得多。该方法不仅用于打印程序，对于 INT 16H 汉字输入程序的话字库、海量词组输入等均可借鉴。

第八章　汉字系统的环境及其移植

8.1　显示器概述

IBM个人计算机系统的发展，经历了从IBM-PC、PC/XT、PC/AT和各种386系统三个阶段。它们以采用8088、80286和80386作主CPU为主要特征。在显示系统方面，IBM公司也不断推出新的显示标准，以适应主机性能的升级。因此，与微机相配套的显示系统，也经历了几个发展阶段。目前有些显示系统的技术指标已被工业界所采纳，形成了公认的标准。

CGA(Color Graphics Adapter)是早期IBM公司推出的彩色图形/字符显示卡，其字符点阵为7×7或5×7，显示的质量不太高，其图形方式的最大分辨率为640×200，汉字按全点阵显示，最多显示11行。

MDA(Monochrome Display Adapter)是IBM公司推出的单色字符显示卡。早期作为高质量的字符工作站使用，以后Hercules公司为其扩充了图形功能，使其成为单色图形/字符显示卡。它采用9×14点阵的字符窗口，其图形分辨率为720×348。按标准可以显示21行汉字。

CGE400(Color Graphics Enhance)亦称为Color400，其分辨率比CGA增加了一倍。虽然不是主流产品，也未形成标准，但在国内拥有许多用户。该显示卡采用9×14点阵字符窗口，最大分辨率为640×400，可以显示25行汉字。

EGA/VGA (Enhanced Graphics Adapter)/(Video Graphics Array)增强型彩色图形卡。IBM-EGA标准分辨率为640×350，字符窗口为8×14点阵，16种色彩。超过该分辨率标准的，统称EGA⁺，这类兼容卡的图形分辨率可以达到640×

480、800×600甚至1024×768。

IBM-VGA 标准分辨率为640×480，字符窗口采用 9×16点阵。其主要特点是采用了256K 种颜色的调色板和用模拟量输出，使得显示的颜色更为逼真。有些兼容卡的分辨率可达800×600、60× 720和1024×768，统称之为VGA⁺。对于EGA/VGA显示卡来说，由于它们的分辨率高，可以显示的汉字为20—30行，甚至可以达到37行显示。目前，大多配置在 286/386 微机系统上，使用得非常广泛。

由于计算机硬件设备发展迅速，这对于汉字系统的兼容性要求也愈来愈高。这些年来，汉字操作系统一直被不断地改进，其版本不断地升级。从早期的仅支持单一的物理设备 的 CC-BIOS V2.1，发展到支持各种配置的系统。

因此，汉字系统应该是面对用户的，根据用户的需要选择软件环境。例如，输入方法可从十几种甚至几十种的编码方案中选择自己最方便、最好用的一种，并且，程序作为“摘挂”式相互覆盖，不多占用内存的开销。显示模块根据用户微机的配置，装

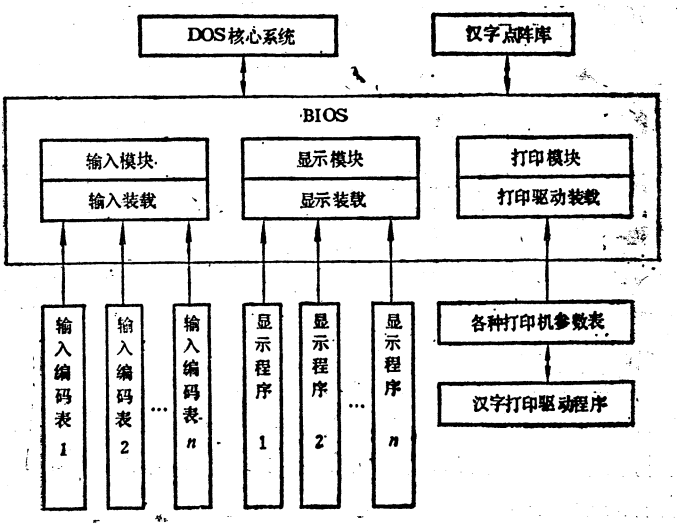


图 8-1 模块组合式汉字系统

入不同的程序模块。打印机驱动程序也不局限一两种类型的打印机，而是根据各种类型的打印机的控制参数，建立一个或数个参数表，由一个通用的程序或者是由用户指定的打印机类型生成一个汉字打印驱动程序。

这样，从整体结构上看，汉字系统由几个大类功能模块组成。如图 8-1 所示。

基于上述思想，就必须对原有的 CC-BIOS系统进行移植，才能支持各种物理环境。本章主要针对 CC-DOS 在各种显示卡上的移植方法，并针对各种常用的显示卡，介绍显示器的基本结构、编程技术和显示汉字的原理。由于是以汉字化问题来讨论的，因此，着重论述各类显示卡的图形功能。

8.2 CGA 显示卡

早期配置在IBM-PC/XT微机上的显示器大部分是CGA 彩色/图形适配器，由于其分辨率的限制，只能使用 11 行显示的 CC-DOS 汉字系统。当然，该显示器的用途很广，具有许多开发潜力，在此，对其原理及其编程有针对性地作一简要介绍。

8.2.1 CGA卡的基本结构

CGA 卡是 PC/XT 系统和彩色显示器的接口，它通过一个槽口和系统总线（I/O 通道）相连接，其背面以 9 针D型插座和视频显示器连接。它的作用是将PC/XT 微机系统发出的一系列显示信息转换为视频信号去控制显示器内电子束的强度变化，从而达到显示字符或图形的目的。

CGA 可以工作在两种方式下，一种是字符方式：即 40×25 或80×25字符方式。在这两种分辨率下，把字符定义在一个8×8 的字符框中。小写由7×5点阵组成，大写由7×7点阵组成，并具有各种显示属性。

另一种方式是图形方式，显示器屏幕上的每个像素均可由软

件选择其色彩。图形方式下有三种不同的分辨率：

（1）高分辨率：每帧200线，每线640点，每点只能取黑白两种颜色。

（2）中分辨率：每帧200线，每线320点，每点可以有4种不同颜色。

（3）低分辨率：每帧100线，每线160点，而每点则有16种不同颜色。

但是，BIOS功能不支持低分辨率，一般由程序员自行开发。

视频缓冲区（亦称显示缓冲器）是存放屏幕要显示的信息的，它所存贮的内容和显示屏画面一一对应。系统由程序预先把要显示的数据存入视频缓冲区，然后由CRT控制器产生视频缓冲区地址码，以读出其中的显示信息。

视频缓冲区采用8片动态RAM芯片组成一个容量为16KB的随机读写存贮器。在系统内存地址空间中，它占据的位置是B8000H—BBFFFH。对于字符方式，由于每个显示字符需两个存贮单元（一个代码，一个属性码），因此，当每屏显示80×25个字符时，显示缓冲区可容纳4个页面的信息。在图形方式下，屏幕上每个像素对应于视频缓冲区中的每字节的一位和二位；若采用640×200图形方式时，一个像素只占一位，一个字节8个像素。所以满屏的图形需要640×200/8＝16000byte的存贮器。若采用320×200图形方式时，一个像素由两位组成，一个字节可置4个像素，因此满屏的图形也需要16KB的缓冲容量。

CRT控制器是根据显示时钟的频率来对视频缓冲区进行扫描刷新的。因此，它所采用的是隔行扫描方式。即奇数扫描线和偶数扫描线分两个区域扫描。在图形方式下，CGA的视频缓冲区的地址分配如图8-2所示。

系统把16KB的缓冲区分为两个区域，偶数行送入第一分区，奇数行送入第二个分区。虽然两个缓冲区的地址相差2000H，但是在屏幕上的奇偶扫描线是连续的。

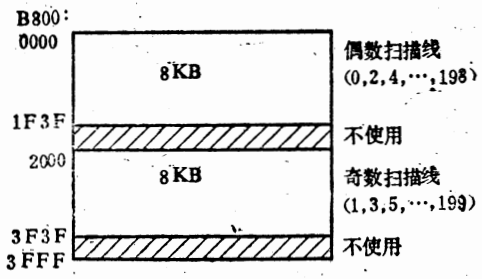


图 8-2 CGA视频缓冲区

CGA 的最高分辨率为 640×200，其缓冲区中的每一位对应一个像素，该位直接对应于屏幕上一个点的亮与暗。寻址显示时，水平位移X坐标的范围为0—639，垂直位移Y坐标的范围为0—199，如图8-3所示。

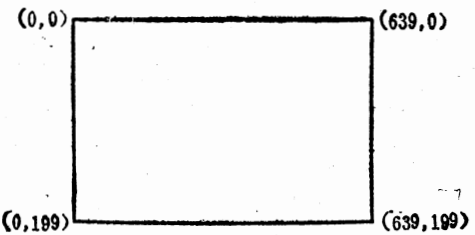


图 8-3 640×200图形寻址

其中，每连续的一组80个字节（640位）对应一根水平扫描行。一个字节内，每一位对应一个点素，最高有效位对应于连续8个像素的左端点位，最低有效位对应于连续8个像素的右端点位。内存映像作为隔行扫描，在屏幕上某一个(x,y)坐标的字节地址，可由下列公式算出：

Offset = {(Y AND 1)×2000H} + (Y/2×50H) + (X/8)

上述算法可由下列汇编程序实现：

```
;     AX=Y,BX=X
      SHR       BX,1
      SHR       BX,1
      SHR       BX,1          ;X除以8
      PUSH      AX            ;保存Y值
      MOV       CX,50H        ;(Y/2)*50H
      MUL       CX
      ADD       BX,AX         ;乘积加(X/8)
      POP       AX            ;恢复Y值
      AND       AX,1          ;加(Y AND 1)*2000H
      JZ        Secd
      ADD       BX,2000H
Secd:
      :                       ;BX=视频缓冲区偏移地址
      RET
```

8.2.2 CGA编程要点

CGA 显示器其显示功能是由一个 M6845 芯片控制，也称作可编程 CRT 控制器。控制器内部有18 个寄存器，用来定义和控制 CRT 的光栅扫描，另外还有一个索引寄存器，实际上是用作为18个内部寄存器的指针。这些寄存器是只写寄存器，通过 CPU 执行一条输出指令而把参数写入的。

此外，还有几个寄存器用来选择工作模式，设置色彩以及控制状态等，最常用的寄存器端口地址如表 8-1 所示：

CGA 寄存器端口地址　　表 8-1

CGA 寄存器口	寄存器名称
3D 4	基址索引寄存器
3D 5	M6845数据寄存器
3D 8	模式寄存器
3D 9	彩色控制寄存器
3DA	状态寄存器

为了对M6845内部寄存器设置参数，首先通过索引寄存器口3D4，指明寄存器号 R_n，然后由OUT输出指令，向数据寄存器口3D5写入要设置的数据。通过向M6845寄存器写入不同的参数，可以把屏幕设置为不同的显示方式。数据寄存器3D5的参数值如表8-2所示：

M6845 数码寄存器参数　　表 8-2

寄存器号	名称意义	单位	40×25(A/N)	80×25(A/N)	图形方式
R_0	水平总时间	字符	38	71	38
R_1	水平显示时间	字符	28	50	28
R_2	水平同步位置	字符	2D	5A	2D
R_3	水平同步宽度	字符	0A	0A	0A
R_4	垂直总时间	字符行	1F	1F	7F
R_5	垂直总时间校正	扫描线	06	06	06
R_6	垂直显示时间	字符行	19	19	64
R_7	垂直同步位置	字符行	1C	1C	70
R_8	隔行扫描方式	——	02	02	02
R_9	最大扫描线地址	扫描线	07	07	01
R_{10}	光标起始线	扫描线	06	06	06
R_{11}	光标结束线	扫描线	07	07	07
R_{12}	起始地址	——	00	00	00

下面的汇编语言程序，就是为设置CRT内部寄存器而编写的。其中BX指向表8-2中某一组寄存器参数表，通过输出指令，使得M6845设置为指定的屏幕显示方式。

```
;       BX=视频参数表地址
        MOV     CX,10H
        XOR     AH,AH
        MOV     DX,3D4H     ;索引寄存器
Setd:
        MOV     AL,AH       ;索引号Rn
        OUT     DX,AL
        INC     DX          ;数据寄存器
        INC     AH
        MOV     AL,[BX]     ;参数表
```

```
OUT     DX,AL      ; 发送数据
INC     BX         ; 表地址加1
DEC     DX
LOOP    Setd
```

模式寄存器3D8：适配器中模式寄存器口是一个8位的只写寄存器。它的作用是选择屏幕显示模式，CGA的显示器操作模式共有8种，除单色外，其余7种模式之一均可以被彩色显示器选定。这里的选定指的是向3D8寄存器设置不同的控制字节。寄存器各位的意义如下：

$b_0=0$ 40×25字符模式；$=1$ 80×25字符模式

$b_1=0$ 字符模式；$=1$ 图形方式

$b_2=0$ 彩色方式；$=1$ 黑白方式

$b_3=0$ 禁止视频输出；$=1$ 允许视频输出

$b_4=0$ 其他方式；$=1$ 640×200图形方式

$b_5=0$ 不允许闪烁；$=1$ 允许字符属性闪烁

b_6，b_7不使用。

在彩色适配器中，3D8模式选择寄存器，除参与闪烁和颜色控制外，还用于控制生成时序信号和同步信号。其中，包括字符时钟、点位时钟，并/串转换器的输入信号和颜色编码器的控制信号等。但是，3D8各位的组合并不都是任意的，常用的组合如表8-3所示。

模式寄存器选择　　表 8-3

编号	操作模式	控制字	b_5	b_4	b_3	b_2	b_1	b_0
0	40×25 黑白字符	2CH	1	0	1	1	0	0
1	40×25 彩色字符	28H	1	0	1	0	0	0
2	80×25 黑白字符	2DH	1	0	1	1	0	1
3	80×25 彩色字符	29H	1	0	1	0	0	1
4	320×200 彩色图形	2AH	1	0	1	0	1	0
5	320×200 黑白图形	2EH	1	0	1	1	1	0
6	640×200 黑白图形	1EH	0	1	1	1	1	0

彩色控制寄存器3D9：该寄存器是一个只用6位的只写寄存器，其控制屏幕色彩变化要根据当前的显示方式来决定，在不同的显示方式下，某一位的设置将影响不同的场景。3D9寄存器的各位意义及控制场景如下：

b_0——设置蓝色，字符方式为边框，320×200方式为背景，640×200方式为前景；

b_1——设置绿色，场景控制同上；

b_2——设置红色，场景控制同上；

b_3——设置高亮度，场景控制同上；

b_4——在图形方式下选择相同的、高亮度色彩组，在字符方式下，为选择背景色彩；

b_5——在320×200图形方式下，选择常用的色彩组；

b_6，b_7不使用。

通过高亮度位与蓝、绿、红位组合，可在8种基本色彩的基础上，演变为16种色彩。

状态寄存器3DA：该寄存器是一个只使用4位的只读寄存器，利用该端口来测试屏幕状态。3DA寄存器各位的意义如下：

b_0——禁止显示；

b_1——光笔触发器置位；

b_2——光笔开关接通；

b_4——垂直同步；

b_4～b_7不使用。

当b_0置1时，允许显示，此时访问视频缓冲区不致于干扰屏幕显示。当b_3处于1状态时，说明此瞬间光栅处于垂直回扫状态，也是进行视频缓冲区刷新的好时机。

8.2.3 汉字显示改进

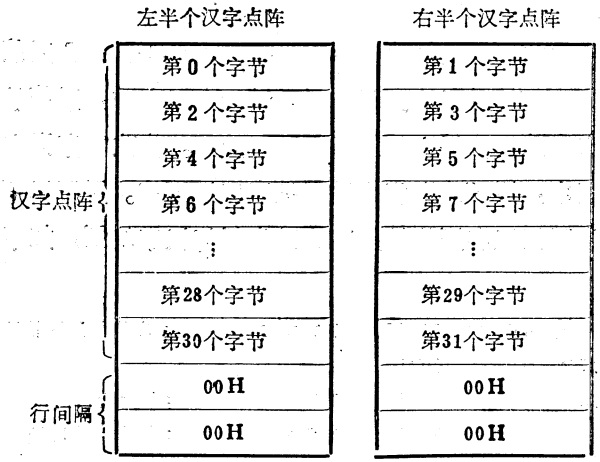
在CGA显示卡的环境下，由于最高分辨率为640×200，对于16×16点阵的显示字模来说，只能显示10行信息外加1行提示

窗口。信息吞吐量远远不能满足用户的要求，汉化后的软件也只能显示10行信息，使得原西文25行屏幕信息面目全非。

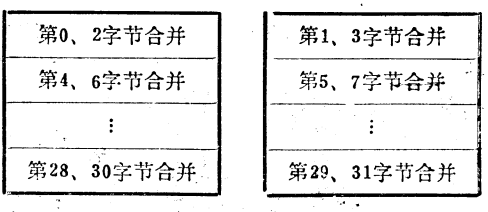
这里，提出了一种方案，就是把CC-BIOS汉字系统的显示模块作如下几方面的修改，使之在CGA 640×200的显示分辨率下可以显示25行的信息。

(1) 修改末行光标尾指针，把0AH(10行)改为18H(24行)，最后一行留作提示窗口。

(2) 修改滚行程序，把原来按一组18根扫描线（16点阵加2线行间隔）滚行改为按8根扫描线一组的形式滚行。为了在纵向200根扫描线上显示25行，每行汉字只能占用8根扫描。ASCII



(a) 10行显示的汉字点阵排列



(b) 25行显示汉字点阵压缩

图 8-4 显示汉字点阵排列

字符为8×8的字符框，汉字为8×16的字符框。

(3) 把送往提示行的字符代码，其BL寄存器内的属性字节高7位置1，以致提示窗口的重码汉字和信息用反示效果显示出来。这是为了当去掉提示行分隔线后，屏幕信息与提示行窗口有所区别，为用户提供了视觉上的方便。

(4) 屏幕写点的修改，这是比较重要的一环。因为把原有的16×16汉字点阵字模，压缩为8×16点阵进行显示，必须在字模加工时作适当的处理。这种处理方式不干涉原有的汉字库内的点阵数据，仅当显示的过程把前后两个字节作“或”的逻辑运算，把两个字节合并为一个字节。原显示点阵排列如图8-4(*a*)所示，经压缩处理后的点阵排列如图8-4(*b*)所示。

字库内的汉字点阵数据排列，是和原CC-BIOS系统的字模加工区的排列相对应的。每当要显示一个汉字时，系统总要把该汉字的点阵数据先送入加工区内进行处理。修改程序嵌在字模加工的过程中，用“或”指令进行纵向字模压缩。用逻辑“或”的思想是为了把汉字压缩并且不会丢失任何汉字笔划，如下列程序所示。

```
CS:224E BF7800  MOV   DI,0078     ; 字模加工区首址
CS:2251 07      POP   ES
CS:2252 06      PUSH  ES          ; ES=DS
CS:2253 8EDE    MOV   DS,SI       ; 汉字点阵段址
CS:2255 33F6    XOR   SI,SI
CS:2257 B91000  MOV   CX,0010     ; 共32个字节
CS:225A AD      LODSW             ; 取2个字节
CS:225B 0B04    OR    AX,[SI]     ; 和下2个字节“或”
CS:225D 46      INC   SI
CS:225E 46      INC   SI          ; 跳过2字节
CS:225F 49      DEC   CX
CS:2260 F6C370  TEST  BL,70       ; 反示吗 ?
CS:2263 7402    JZ    2267
CS:2265 F7D0    NOT   AX          ; 字取反
```

```
CS:2267  AA       STOSB                ;低位存入加工区
CS:2268  26       ES:
CS:2269  886511   MOV    [DI+11],AH    ;高位存入加工区
CS:226C  E2EC     LOOP   225A          ;循环至CX=0
```

程序中DS:SI指向汉字库指定的汉字点阵首地址，当用LODSW指令把左半边和右半边的两个字节装入AX寄存器后，SI的地址指针将自动加2，指向下两个点阵字节，然后再进行“或”运算即可实现。

把在字模加工区加工好的汉字点阵送往屏幕显示时，按照奇偶扫描线分两个区域传送，无论是ASCII字符点阵，还是汉字点阵，均写入8个字节。其实现过程如下列程序所示。

```
CS:2319  53       PUSH   BX
CS:231A  B604     MOV    DH,04         ;共显示8个字节
CS:231C  AC       LODSB                ;取一个字节
CS:231D  F6C380   TEST   BL,80
CS:2320  7518     JNZ    233A
CS:2322  AA       STOSB                ;显示一个字节
CS:2323  AC       LODSB                ;再取一字节
CS:2324  26       ES:
CS:2325  8885FF1F MOV    [DI+1FFF],AL  ;隔行显示一字节
CS:2329  83C74F   ADD    DI,+4F        ;视频地址递增
CS:232C  FECE     DEC    DH
CS:232E  75EC     JNZ    231C          ;循环至DH=0
CS:2330  5B       POP    BX
```

程序中DS:SI指向字模加工区地址，ES:SI指向视频缓冲区地址，如果BL高8位为1，作为“异或”操作，其传送方式相同。

虽然用这种方法所显示的汉字略有失真变形，但是，所显示的信息量比原10行系统增加了1.5倍。并且，许多25行的汉化软件都可以毫无阻碍地得以运行，大大提高了系统的兼容性。

8.3 MDA单色显示卡

在许多微机工作站上的仿真终端或低档个人机上，都配置了MDA单色显示器。早期的单色适配卡没有图形显示功能，只能以字符方式显示质量比较高的字符。后经Hercules公司对该显示卡作了一些修改，增加了该显示卡的图形功能。从此以后，渐渐被用户们接受，随之形成了一个单色显示卡的标准，国内也拥有大量的用户。

MDA显示卡除了没有色彩外，其价格便宜，兼容性强，而且开发的潜力很大。在此介绍一些有关单色卡（Hercules卡）的结构原理和汉字系统移值的方法，便于读者掌握MDA图形功能的使用。

8.3.1 MDA卡的基本结构

MDA显示适配器插在系统板的任一扩展槽内，与系统的I/O通道相连接。其主要性能如下：

(1) 在字符方式下，可显示25行，每行80个字符，即80×25黑白字符模式。

(2) 每个字符为9×14点阵的字符框，其中，字体为7×9点阵。于是，水平分辨率为80×9=720，垂直分辨率为25×14=350。

(3) 水平扫描频率为18.43kHz，垂直扫描频率（刷新）为50Hz。

(4) 使用P-39荧光粉，有高保真特性。

(5) 采用M6845 CRT控制器，具有8KB的ROM和256个字符的字符发生器。

(6) 在黑白字符方式下，视频缓冲区的段地址从B000H开始。若为Hercules卡，具有8个显示页面。

(7) MDA卡若为Hercules扩充型（目前大多数微机均为

此配置），其视频缓冲区的静态 RAM 具有 64KB 之大，就是在高分辨率的图形方式下，也可以分为两个显示页。

MDA 卡在图形方式下，一般标准分辨率为 720×350，使用第一个显示页面，即视频缓冲区的首地址为 B000H，关闭第二页面，即首地址为 B800H 的视频区域。在一个图形页面中。其刷新扫描的方式又分为 4 个区域，通常称作隔双行扫描方式。如若调整M6845内部寄存器的垂直扫描时间，可以把其分辨率最高调整到640×408。若按常规640×400分辨率方式显示的话，屏幕与视频缓冲区正好对应 32KB RAM，视频缓冲区映像和分配情况如图8-5所示。

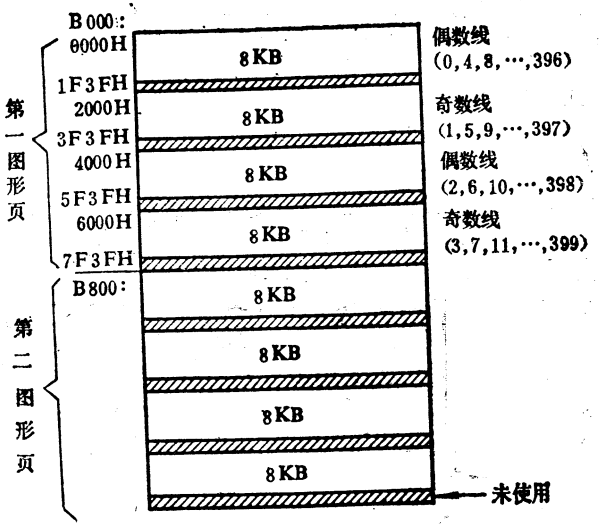


图 8-5 MDA视频缓冲区分配

图中每个分区占 8KB RAM 地址，可以扫描100根线，其视频偏移量首址分别为：0、2000、4000、6000等。

两个图形页面可以通过一个寄存器端口，在初始化时进行切换。打开的页面，对应显示屏幕信息。而关闭的页面就如同一般的计算机内存贮器。可通过程序对其进行直接寻址，储存任何数

据。当屏幕初始化且又不切换图形页面时，该页面的数据不会丢失。

在720×350图形方式下，可以显示21行汉字，但是每行显示45个汉字，不太规范。

一般，用MDA显示卡作为终端的数据录入，总是以标准的25行汉字显示较好，而且软件的兼容性比较强。如果要显示25行的汉字，其分辨率至少要设置到640×400。由于单色显示卡没有色彩，在屏幕每一位只对应一个点的亮和暗。水平位移X坐标的范围为0—639，垂直位移Y坐标的范围为0—399，其某一字节在屏幕上显示的偏移地址的算法为：

$$Offset=(X\ AND\ 3)*2000H+(X/4*50H)+Y/8$$

根据上述公式，可用下列汇编语言来实现地址定位：

```
; CX=Y,DX=X
  MOV     AX,DX
  MOV     BL,50H
  SHR     AX,1
  SHR     AX,1          ; X/4
  MUL     BL            ; (X/4*80)
  MOV     SI,AX
  AND     DX,0003H      ; DX=X AND 3
  MOV     AX,2000H
  MUL     DX            ; AX=(X AND 3)*2000H
  ADD     SI,AX         ; SI=(X AND 3)*2000H+(X/4*8)
  MOV     DX,CX
  MOV     CX,0003H
  AND     CH,DL
  SHR     DX,CL         ; DX=Y/8
  ADD     SI,DX         ; 视频缓冲区地址
  RET
```

8.3.2 MDA显示卡编程要点

由于MDA显示卡也是通过M6845控制器来控制屏幕显示及

光栅扫描的。其地址端口与CGA相对应但又不同，如表8-4所示。

MDA 显示卡寄存器端口　　表 8-4

寄存器地址	功　能　名　称
3B4	基址索引寄存器
3B5	M6845数据寄存器
3B8	CRT控制端口
3BA	状态寄存器
3BF	Hercules 卡寄存器

M6845内部寄存器是用“输出”指令来写入设置的，通过3B4索引寄存器口，选择3B5中某一个寄存器的指针，然后发送数据参数。

3B5数据寄存器的控制参数值，如表8-5所示。

MDA 卡 M6845 数据寄存器参数　　表 8-5

编号	寄存器名称意义	单　位	80×25	720×348	644×401
R_0	水平总时间	字　符	61H	35H	35H
R_1	水平显示时间	字　符	50H	2DH	28H
R_2	水平同步位置	字　符	52H	2EH	2DH
R_3	水平同步宽度	字　符	0FH	07H	07H
R_4	垂直总时间	字符行	19H	5BH	67H
R_5	垂直总时间校正	扫描线	06H	02H	0DH
R_6	垂直显示时间	字符行	19H	57H	66H
R_7	垂直同步位置	字符行	19H	57H	66H
R_8	隔行扫描方式	——	02H	02H	02H
R_9	最大扫描线地址	扫描线	0DH	03H	03H
R_{10}	光标起始线号	扫描线	0BH	00H	0BH
R_{11}	光标结束线号	扫描线	0CH	00H	0CH

在移植的CC-DOS汉字系统下，把单色显示卡的分辨率设置为640×401，25行作为显示汉字信息，多出一根线作为屏幕信息与提示行的分隔线之用。绝大多数的Hercules单色卡都可以按照表8-5的参数对CRT进行初始化操作，如果有些差异的话，可以调整一下R_4～R_7，控制垂直扫描的寄存器参数。

在黑白字符方式下，虽然设有色彩，但字符属性码可以设置显示字符闪烁、高亮度及反示等。其属性代码的意义如图 8-6 所示，并按其译码前景与背景和高亮度、闪烁位进行组合，增加属性功能，如表8-6所示。

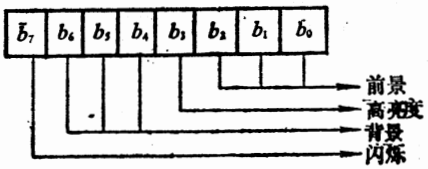


图 8-6 属性字节各位意义

属性码组合功能 **表 8-6**

背景			前景			背能
b_6	b_5	b_4	b_2	b_1	b_0	
0	0	0	0	0	0	不显示
0	0	1	0	0	0	下划线
1	1	1	0	0	0	黑底白字
0	0	0	1	1	1	反像显示

3B8是CRT控制端口，也可作为模式寄存器使用。其各位功能如下：

b_0——高分辨率字符方式；

b_1——图形显示；

b_2——未使用；

b_3——视频允许位；

b_4～b_6——未使用；

b_7——选择图形页面，=0 为 B000H，=1 为 B800H。

3BA是状态端口，该口的寄存器只允许读取，其各位意义如下：

b_0——水平驱动；

b_1、b_2——保留；

b_3——黑白视频；

$b_4 \sim b_7$——未使用。

3BF寄存器的b_0、b_1位可决定打开Hercules图形卡，若要使得MDA卡工作在图形方式下，必须把这两位设置为1，否则无法打开图形功能，通常，在初始化之前，作这项操作。

8.3.3 MDA汉字显示模块的移值

目前，越来越多的多用户系统的工作站都采用了价廉物美的MDA显示终端。为了能够录入汉字信息，就必须适当地配置汉字系统软件。在能够使用汉字显示系统的同时，还应该发挥MDA卡高分辨率的图形功能，把其分辨率设置为640×401。这个工作是把原CC-BIOS程序中M6845寄存器参数改为表8-5中所示的参数，并打开Hercules图形卡功能，初始化的移植工作就完成了。

下一步，把CC-BIOS光标尾指针单元改为18H，即有效屏幕信息为24行，保留一行提示行。

滚行的修改原则是按照4个分区循环4次为一行的滚行处理。即16线对应汉字纵向16点阵，当中没有行间隔。

向视频缓冲区传送字模点阵信息的方式，也是按4个分区交叉写入，如下列程序所示。

```
CS:15BD  52          PUSH   DX
CS:15BE  B604        MOV    DH,04          ;各区送4次
CS:15C0  F6C380      TEST   BL,80          ;异或标志
CS:15C3  751B        JNZ    15E0
CS:15C5  AD          LODSW                 ;取2个点阵字节
CS:15C6  AA          STOSB                 ;送第一区
CS:15C7  26          ES:
CS:15C8  88A5FF1F    MOV    [DI+1FFF],AH   ;送第二区
CS:15CC  AD          LODSW                 ;再取2个字节
CS:15CD  26          ES:
CS:15CE  8885FF3F    MOV    [DI+3FFF],AL   ;送第三区
CS:15D2  26          ES:
CS:15D3  88A5FF5F    MOV    [DI+5FFF],AH   ;送第四区
```

```
CS:15D7  83C74F      ADD     DI,+4F          ;加4线地址
CS:15DA  FECE        DEC     DH
CS:15DC  75E7        JNZ     15C5            ;继续显示共16线
CS:15DE  5A          POP     DX
CS:15DF  C3          RET
CS:15E0  AD          LODSW
CS:15E1  26          ES:                     ;异或操作同上
CS:15E2  3005        XOR     [DI],AL
CS:15E4  26          ES:
CS:15E5  30A50020    XOR     [DI+2000],AH
CS:15E9  AD          LODSW
CS:15EA  26          ES:
CS:15EB  30850040    XOR     [DI+4000],AL
CS:15EF  26          ES:
CS:15F0  30A50060    XOR     [DI+6000],AH
CS:15F4  83C750      ADD     DI,+50
CS:15F7  FECE        DEC     DH
CS:15F9  75E5        JNZ     15E0
CS:15FB  EBE1        JMP     15DE            ;返回
```

程序中DS:SI指向字模加工区，而ES:DI指向视频缓冲区的偏移地址，视频段地址根据所选择的图形页，分别设置ES=B000H或B800H。程序中地址CS:15C6—CS:15D3表明了分别向4个分区对应的地址处写数据，使得屏幕上的图形像素纵向连续，一共循环DH=4次。若是ASCII码，将调用一次该程序，如果是汉字，则调用两次，显示左右半个汉字字模以形成完整的汉字。

对于ASCII码的点阵字模来说，如果仍以7×8点阵显示，就显得比较小，与汉字比较失调。如有可能，可把ROM中9×14点阵的字模用Debug的文件形式调出来，然后把该字模库的首地址设置在INT 1FH处。代换原来的7×8点阵字模。在CC-BIOS计算ASCII码地址的程序中，把原来乘以8的运算基数，改为乘以0EH。这样修改后的CC-BIOS显示程序，无论是在西文

方式还是在中文方式下，其ASCII字符的大小一致。不但看起来美观，而且与汉字的高低非常协调。

8.3.4 MDA仿真CGA显示

由于许多用户开发的汉字应用软件是在CGA彩色显示卡10行CC-DOS下进行的，还有一些汉字操作系统，都是针对CGA卡环境开发的，致使这些用户不能在MDA单色显示器上使用，若要重新移植，则需一定的技术、资料和时间。另外，有许多低档个人机，大多配置的是单色卡，使得丰富的游戏软件不能运行使用，由此，提出了一个系统兼容的问题。

可喜的是，通过对MDA显示卡的仔细分析，发现它能够模拟CGA显示卡的图形显示功能。虽然没有色彩，但可由不同的灰度来反映不同的色差。不但可以仿真640×200分辨率方式，还可以仿真320×200，使得所有原CGA上使用的汉字软件、作图软件和游戏等软件都可以顺利使用。

仿真CGA的编程要点如下：

- 首先，打开Hercules图形卡；
- 修改BIOS数据区的设备字节；
- 对M6845内部寄存器的R$_9$指定隔一行扫描方式；
- 初始化后清屏幕；
- 打开图形第二页，使得视频缓冲区对应B800H。

实现仿真CGA的程序如下：

```
DATA    SEGMENT
 Dmd            DB 35H,28H,2DH,0AH,7FH,06H,
            DB 64H,70H,02H,01H,06H,07H
DATA    ENDS
CODE    SEGMENT
        ASSUME  CS:CODE,DS:DATA
START PROC      FAR
                PUSH    DS
                XOR     AX,AX
```

```
        PUSH    AX
        MOV     AX,40H          ; ROM数据区
        MOV     DS,AX
        MOV     AX,6DH
        MOV     DS:[0010H],AX   ; 置设备字节
        MOV     AX,6            ; AX=6,仿真640×200
        INT     10H             ; AX=4,仿真320×200
        MOV     AX,DATA
        MOV     DS,AX
        MOV     AL,3
        MOV     DX,3BFH
        OUT     DX,AL           ; 打开Hercules卡
        MOV     AL,2
        PUSH    AX
        MOV     DX,3B8H
        OUT     DX,AL           ; 显式模式
        PUSH    DS
        POP     ES
        LEA     SI,Dmd          ; 指向视频参数表
        MOV     DX,3B4H         ; M6845基址
        MOV     CX,12
        XOR     AH,AH
PRARX:
        MOV     AL,AH
        OUT     DX,AL
        INC     DX              ; 数据寄存器口
        LODSB
        OUT     DX,AL           ; 送初始化参数
        INC     AH
        DEC     DX
        LOOP    PRARX
        MOV     CX,8000H
        MOV     AX,0B800H       ; 视频区段地址
        CLD
        MOV     ES,AX
```

```
                XOR     DI,DI
                XOR     AX,AX
                REPZ    STOSW           ; 清屏幕
                MOV     DX,3B8H         ; 模式寄存器口
                POP     AX
                OR      AX,10001000B    ; B800段址参数
                OUT     DX,AL           ; 允许隔行扫描
                RET
START   ENDP
CODE            ENDS
                END     START
```

以上程序经过汇编连接，就可形成后缀为EXE的执行文件。在系统提示符下执行该程序之后，显示缓存区将工作在第二个图形页，并仅仅使用了两个分区，共16KB的存贮器，所以，屏幕信息比较紧凑。这时，就可以执行任何在CGA显示卡上使用的图形软件了。

8.4 CGE400显示卡

CGE400显示卡亦称为Color400卡，这种高分辨率彩色图形扩展卡是为IBM PC/XT显示器的软硬件能最大程度地相兼容而设计的。Color400显示卡在M6845控制芯片基础上，可提供640×400分辨率的彩色图形及同CGA兼容的中分辨率和文本方式。由于显示时钟是由主CPU控制，所以，利用Intel 80286处理器来控制操作，大大提高了显示速度和运行效果。

8.4.1 Color400卡的基本结构

在文本方式下，Color400显示卡与EGA显示卡区别不太大。当工作在640×400高分辨率彩色图形方式下时，其视频缓冲区中每个点位均可以寻址，每个像素可以设置为16种彩色中的一种。像素的16种色彩需用二进制4个点位(bit)来表示，因而，

其显示缓冲区容量可由下式得出：

全屏幕像素：M = 640 × 400 = 256000

全屏幕点位：B = M × 4 = 1024000Bits

全屏幕RAM：R = B/8 = 128000Bytes

由此可以看出，高分辨彩色图形的显示，视频缓冲区RAM将需要128KB存贮器。计算器的屏幕显示，实际上是通过CPU发送指令对视频缓冲RAM读写的过程。视频缓冲区中相应地址的点位信息映像出屏幕上相应位置的内容。所以说，视频缓冲区使用得好坏直接影响CRT的显示效果。

Color400显示卡虽然有128KB的显示RAM，但在系统1MB字节的地址空间中，随机读写存贮器仅占B8000H—BFFFFH之间的32KB空间。实际上，系统把128KB的显示RAM分为4个平面板，如图8-7所示。

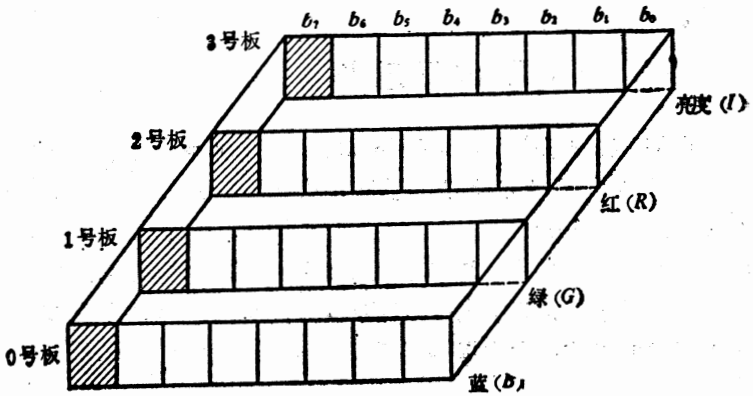


图 8-7　视频缓冲区分平面图

其中，每个平面板占用32KB的字节空间，系统在每一时刻只能对一个平面读写。4个平面板分别表示4种基本色根(B、G、R、I)。0号板为蓝色；1号板为绿色；2号板为红色；3号板为

高亮度。以 2DEH 调色板寄存器 I/O12 地址的低 2 位为开关，来切换所要寻址的平面。一旦选择了某一块平面板为当前的视频缓冲区，系统允许CPU在B800:0000—B800:8000H 地址之间任何一点位寻址。若在4 个平面板的同一位写入相应的像素值，它们的关系是“或”的逻辑组合，可在屏幕上相应的位置组合出16种不同的色彩。

这种彩色组合方式，并没有前景和背景的概念，实际上是一种色彩叠加的中和技术，映像到屏幕上，给人以不同色彩的效果。调色板的 C_0、C_1、C_2和C_3 分别表示蓝、绿、红和亮度，其 16 种组合的彩色编码如表 8-7 所示。

Color400 彩色编码表　　表 8-7

C_3	C_2	C_1	C_0	颜色
0	0	0	0	黑色
0	0	0	1	蓝色
0	0	1	0	绿色
0	0	1	1	青色
0	1	0	0	红色
0	1	0	1	洋红
0	1	1	0	棕色
0	1	1	1	淡灰
1	0	0	0	深灰
1	0	0	1	天蓝
1	0	1	0	浅绿
1	0	1	1	淡青
1	1	0	0	淡红
1	1	0	1	品红
1	1	1	0	黄色
1	1	1	1	白色

C_0—C_3 作为调色寄存器的索引选择配色寄存器中的值，把它们作为CRT 的数字驱动信号。

这些调色寄存器是4 位寄存器，它们分别驱动 CRT的 I、R、

G、B信号线。譬如，在C_0板上的局部位置送入全像点字节FFH屏幕将显示一个局部的蓝色块，再选择另外一个彩色板，或者把当前彩色板从0号切换到2号板，这时，若写入任何字符或像素值后，屏幕将会显示蓝色背景和品红的前景。若再打开亮度板，将组合成天蓝色背景和浅红色前景，以此类推。若要清屏的话，必须对4个彩色板全部写入00H，才能使整个屏幕为黑色。所以说，当屏幕上有好几种彩色信息时，若要有滚屏或刷屏操作的话，必须要对4个平面板分别作相同的操作。

下列程序是向屏幕指定的位置写彩色值的过程，简单说明了在640×400图形方式下彩色显示的基本方法。

```
; BH=色彩0，  CH=色彩2
; BL=色彩1，  CL=色彩3
Putbyte  PROC   NEAR
         MOV    AX,0B800H
         MOV    ES,AX        ; ES:DI=视频地址
         MOV    DX,2DEH      ; 平面选择寄存器
         XOR    AL,AL
         OUT    DX,AL        ; 0号平面
         MOV    ES:[DI],BH
         INC    AL
         OUT    DX,AL        ; 1号平面
         MOV    ES:[DI],BL
         INC    AL
         OUT    EX,AL        ; 2号平面
         MOV    ES:[DI],CH
         INC    AL
         OUT    DX,AL        ; 3号平面
         MOV    ES:[DI],CL
         RET
Putbyte  ENDP
```

Color400在当前的32KB平面显示时，其扫描方式与MDA图形方式下基本相同。即把当前随机访问的视频缓冲区分为4个

区域。视频段地址从B800H 开始。它的扫描方式如图8-8所示的奇/偶线在4 个区域中隔行交叉扫描。每个区域最大扫描行数为100线。

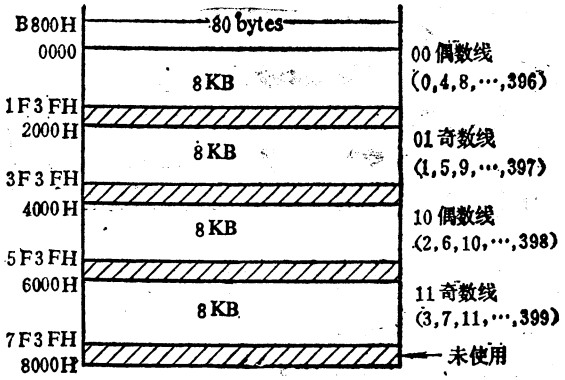


图 8-8 Color400 视频分区域图

屏幕水平位移X坐标的范围为0—639，垂直位移Y坐标的范围为0－399，每连续的一组80个字节对应一根水平扫描线。

Color400视频缓冲区偏移地址的算法如下：

Offset=(X AND 3)*2000H+(X/4*50H)+Y/8

但是在该地址画任一彩色点，其算法与别的显示卡有很大差别，最主要的是平面板的选择，以下是计算点位地址的模拟算法程序：

```
        0≤irgb≤15（彩色码）
bitORmask=2∧[7-(Y MOD 8)]
bitANDmask=NOT bitORmask
FOR plane=0 TO 3              （4个平面）
    testmask=2∧plane
写I/O(2DE)=plane              （选平面）
    data=[B800:Offset]
if irgb AND testmask <> 0
```

```
    THEN [B800:Offset]=data OR bitORmask
    ELSE [B800:Offset]=data AND bitANDmask
NEXT plane
```

这些算法利用汇编语言或者C语言都能够方便地实现。

8.4.2 Color400卡的编程要点

CPU控制CRT显示是通过IN/OUT指令进行的。Color400占用了16个I/O端口地址，即2D0H—2DFH。而端口地址3D0H—3DFH仍然保留给标准IBM-PC/XT与CGA兼容使用，2D0H是Color400控制寄存器的基地址，对寄存器编程的一般步骤如下：

(1) 选择操作模式，设置初始化参数。

(2) 设置彩色平面板。

(3) 写模式、调色寄存器。

(4) 刷新视频缓存区。

以下列出了Color400显示卡各寄存器的端口地址，并对其各位的控制及编程的要点作了扼要说明。

2D0H　M6845　索引寄存器（只写）

2D1H　M6845　数据寄存器（只写）

这两个寄存器通常是在屏幕初始化时使用的，对应其数据表在ROMBIOS的CC00:00F3—CC00:0182处。一般只有在640×400图形方式下使用。

2D2H—2D7H　奇地址重复2D0H，偶地址重复2D1H。

2D8H　Color400　模式寄存器（只写）

该寄存器是一个8位寄存器，用来设置CRT的显示模式，其中各位的功能如下：

$b_0=0$　仅用于40×25方式；$=1$　其他方式；

$b_1=0$　字符模式显示；$=1$　图形模式显示；

b_2　未使用；

$b_3=0$　禁止视频；$=1$　选通视频信号；

$b_4=0$ 中、低分辨率模式；$=1$ 图形为640点列，字符为80列；

$b_5=0$ 禁止闪烁；$=1$ 允许闪烁；

$b_6=0$ 交叉扫描方式；$=1$ 隔行扫描方式；

$b_7=0$ 在80×50方式下8×8点阵字符；$=1$ 选择8×16点阵字符；

2D9H 彩色控制寄存器（只写）

该寄存器作为2DEH的控制寄存器来指定显示操作的方式，其各位功能如下：

$b_0=0$ 选择标准显示操作；$=1$ 定义2DEH寄存器是地址值还是数据值；

$b_1=0$ 关闭信号；$=1$ 打开写彩色图像RAM信号；

$b_2=0$ 预置光笔闩锁器；

$b_3=0$ 清除光笔闩锁器；

b_4 未使用；

$b_5=0$ 禁止非屏蔽中断(NMI)；$=1$ 允许NMI

b_6 未使用；

$b_7=0$ 光标在一对字符的首字符；$=1$ 光标在一对字符的第二字符。

2DAH 状态寄存器（只读）

当进行屏幕操作时，可以读该寄存器的指定位来得到当前屏幕的扫描状态，其各位功能如下：

b_0 点位时钟；

b_1 光笔触发置位；

b_2 光笔开关；

b_3 同b_0；

b_4 垂直同步脉冲输出；

b_5—b_7 同2D9H相应位。

2DEH 彩色板与调色寄存器（只写）

该寄存器用于位平面的选择，通过改变数据位和地址位的

值，指定当前所显示的色彩，其各位功能如下：

b_0—b_3 配色数据位；

b_4—b_7 彩色平面板地址位。

对于调色寄存器的设置，是通过2D9H和2DEH端口写操作来完成的。在Color400高分辨率下，读/写像素都是它们的彩色码，这些彩色码存放在视频缓冲区中各个平面上。在屏幕上真正的显示颜色是由调色寄存器实现的。在BIOS初始调色寄存器时，它们的内容等于编码表的索引号，所以，根据彩色编码表中不同的色彩码，将产生不同的颜色。因为这些寄存器的内容是可以修改的，所以，不同的调色寄存器又可以设置相同的内容，也就是说，可通过修改调色寄存器而使不同的彩色码具有相同的显示色彩。这一功能可在不改变视频缓冲区内容的情况下，改变屏幕上某些图形块的显示色彩。

下面的一段汇编语言程序，说明了修改调色寄存器内容的方法。

```
; Defco=默认面, Orig=原值
MOV    AL,01          ; 选择写配色寄存器
MOV    DX,2D9
OUT    DX,AL          ; 指定2DE端口
MOV    AL,Color
MOV    DX,2DE
OUT    DX,AL          ; 写数据地址
MOV    DX,2D9
MOV    AL,03
OUT    DX,AL          ; 修改配色寄存器
AND    AL,FD
OUT    DX,AL
MOV    DX,2DE
MOV    AL,Defco
OUT    DX,AL          ; 恢复原平面
MOV    AL,Orig
MOV    DX,2D9
```

```
OUT     DX,AL           ; 恢复2D9原值
```

程序中 Color 单元的值高 4 位是指定要修改的寄存器号，低 4 位则表示要写入寄存器的内容。

在 Color400 显示卡上，对应其色彩共有16种颜色的值可以组合。因此，可以对原有的调色寄存器用指定的色彩值进行重置和修改。

例如下列程序，就是把要修改的色彩值顺序存放在内存某一单元中，即 ES:BX 指向的地址，作为一组参数表。其长度为 16 个字节，由程序来实现修改全部调色寄存器。

```
; ES:BX=要修改的表值
        MOV     DX,2DA
LX:
        IN      AL,DX
        TEST    AL,10H      ; 状态测试
        JE      LX
        AND     AL,E0       ; 取b5—b7位状态
        OR      AL,01       ; 色彩地址值
        MOV     DX,2D9
        OUT     DX,AL       ; 选择写配色寄存器
        MOV     AH,AL
        XOR     CX,CX
NX:
        MOV     AL,ES:[BX] ; 取出属性字节
        AND     AL,0F
        INC     BX
        NOT     AL
        AND     AL,0F       ; 合并寄存器的地址
        OR      AL,CL       ; 以及要写入的值
        MOV     DX,2DE
        OUT     DX,AL
        MOV     AL,AH
        OR      AL,03       ; 接通写配色的信号
        MOV     DX,2D9
```

```
        OUT     DX,AL       ; 写入配色寄存器
        CMP     CL,0F       ; 共16色
        JE      FIN
        AND     AL,F0       ; 关闭写配色信号
        OUT     DX,AL       ; 写寄存器端口
        MOV     AH,AL
        ADD     CL,01       ; 寄存器号加1
        JMP     NX
FIN:
        MOV     AL,Orig     ; 恢复原值
        MOV     DX,2D9
        OUT     DX,AL
        MOV     AL,Defco
        MOV     DX,2DE
        OUT     DX,AL       ; 恢复默认平面
```

在Color400显示卡上，除了有与CGA相同的ROM-BIOS之外，还有一块ROM板用于高分辨率的扩充功能。其默认地址为CC000H—CDFFFH，占8KB存贮空间，该地址可以通过DIP开关重新设置。前6KB是Color400视频控制程序，后面2KB是装载ASCII码的字符点阵。

Color400的视频BIOS有许多功能，其中大部分与CGA卡相同，但也有几个功能与其他显示卡有区别，这些特性在系统二次开发时经常用到，在此作些介绍。

(1) 设置当前屏幕显示方式

输入参数：[AH]=00H　　；功能号

[AL]=显示方式参数，

AL=00—07　与CGA相同；

=40H　80×30字符方式；

=41H　80×50字符方式；

=42H　640×400彩色图形方式。

(2) 设置调色寄存器

输入参数：[AH]=10H　　，功能号

[AL]=00H　置调色寄存器；
=01H　置边框色彩；
=02H　修改所有的调色寄存器；
[BL]=要设置的调色寄存器(0—0FH)；
[BH]=要设置的彩色值；
[ES:DX]=指向一个17字节的色彩参数表。

(3) 取调色寄存器值；

输入参数：[AH]=10H　；功能号
[AL]=40H　取调色寄存器；
=41H　取边框调色寄存器；
=42H　取所有调色寄存器。

输出参数：[BL]=取出的调色寄存器；
[BH]=返回色彩值；
[ES:DX]=指向一个17字节的彩色参数表。

(4) 取CGE400信息

输入参数：[AH]=8FH　；功能号

输出参数：[AX]=ROM-BIOS版本号（用二进制表示）；
[BX]=视频缓冲区地址（一般为B800H）；
[CX]=CGE400显示卡ROM地址。

8.4.3 汉字显示模块移植要点

Color400显示器与IBM-PC286微机一同推出，拥有不少用户。因此，把CC-DOS移值到该系统上，使其在高分辨彩色图形方式下，能够保证汉化的系统软件和应用软件正常运行，这就需要对原CC-BIOS汉字系统的显示模块作二次开发，修改有关的功能模块，才能实现这一目标。

在640×400图形方式下，CC-BIOS可以显示25行的汉字（包括提示行）。其视频缓冲区的扫描刷新方式与MDA显示卡640×401一样，是隔4线分别在4个视频RAM区域内扫描的。由于该卡彩色平面的控制方法比较复杂，因此，修改的程序也比

较多，在此仅描述汉字系统移植的方法及要点，了解了显示卡的控制原理和移植方法，便可以方便地实现系统移植的目标。

(1) 屏幕初始化

由于 Color400 显示卡控制寄存器的端口地址和视频初始化参数与 CGA 不同，必须进行重新编程。但是较好的办法是调用 ROM-BIOS 功能，这样，就不需要去化时间摸索其视频参数的控制内容。例如，当初始化模块程序在有关单元内设置了一系列参数后，最后调用下列子程序，便可完成屏幕初始化的操作。

```
        CMP     AL,04                   ; 字符显式方式
        JB      Unit
        MOV     AL,42H                  ; 置640×400方式参数
Unit:
        XOR     AH,AH
        PUSHF
        CALL    DWORD PTR Orig_Bios     ; 调原BIOS视频中断
        RET
Orig_Bios DW    0,0
Load:
        MOV     AX,3510H                ; 取中断向量功能
        INT     21H                     ; 取出Color400中断
        MOV     Orig_Bios,BX            ; 保存偏移地址
        MOV     Orig_Bios+2,ES          ; 保存段地址
```

这个程序的原理是：当汉字系统装载时，把原 ROM-BIOS 视频中断的入口地址取出，分别保存在指定的单元内。当系统有屏幕初始化操作时，则利用一条长调用指令即可实现。其调用的地址是保存在指定单元中原视频中断地址。这种方法要比其他设置寄存器等方法简便得多，还可以根据系统环境，设置入口参数转换为各种显示方式。

(2) 上滚下滚功能

由于CPU在某一时刻，只能访问 128KB 视频缓冲区的某一个 32KB 平面，因此，在一个平面传送扫描信息时，其他平面中的数据不受影响。例如，当红色板上的信息移动时，绿色、蓝色

字符的在屏幕原处显示，保持不变。因此，对于滚屏程序模块来说，应修改为同时滚动4个平面的视频数据。通过2D9H和2DEH平面选择寄存器的切换来实现平面的指定，然后根据隔行扫描的方式，一根线一根线地传送，滚动的速度将会受到一定影响。

(3) 汉字字符显示功能

汉字字符在图形方式下是以点阵的方式显示的，从一个平面的角度来看，把向屏幕传送点阵的子程序，照MDA卡上汉字字符传送方法修改即可，程序在此不再列出。这仅是一种色彩，Color400一旦初始化为640×400的图形方式，其默认是在1号平面板上，即绿色的显示字符。若不考虑色彩控制的话，在当前屏幕状态下，汉字始终以绿色前景显示。

如果要满足各种色彩的汉字字符显示效果，必须在向视频缓冲区传送点阵信息之前，根据应用程序传送过来的属性参数，设置或修改调色寄存器的值，或者选择指定彩色平面板，在指定平面板的相应地址写完点阵信息之后，再返回默认的状态。每一行汉字的第一根扫描线的地址以140H作为被乘数来运算。

因为对于640×400分辨率正好显示25行汉字，屏幕信息与提示行窗口之间无法再显示分隔线。不过，可以把向提示送出的信息用反示的方法来解决，这样，既美观，又可以分清屏幕与提示信息的内容。

(4) 前景/背景色彩控制功能

Color400的色彩控制与CGA完全不同，因此，原CC-BIOS中的0BH功能和Ctrl-F6已不起作用。修改应从CC-BIOS的0BH号功能着手，其方法是：当进入0BH号功能子模块后，根据从BL寄存器代入的参数值，调用原Color400 ROM-BIOS的设置调色寄存器功能即可。

在640×400图形方式下，调用BIOS的10H功能02号子模块，即可方便地改变前景或背景的色彩。例如，下面一段简单的程序，是改变前景色彩的控制方法，还可以进一步改进或增加一些功能来改变背景和边框色彩。

```
        BL=色彩值
        PUSH    ES
        MOV     CS:Color+2,BL          ; 色彩值存入表
        PUSH    CS
        POP     ES
        LEA     DX,Color               ; ES:DX=参数表
        MOV     AX,1002H               ; 修改调色寄存器
        PUSHF
        CALL    DWORD PTR Orig_Bios    ; 调原BIOS功能
        POP     ES
        RET                            ; 前景已改变
Color DB        17 DUP(?)
```

程序中在内存开辟了一个彩色参数表缓冲区，对应有17个字节，由ES:DX 指向该表的首地址。改变有关的表参数，将修改不同的调色寄存器。譬如，第1个字节为背景色彩，第3个字节作为改变前景等，当设置好表参数之后，用长调指令调用原BIOS功能即可。

8.5 EGA/VGA显示卡

EGA/VGA 显示卡是高级图形显示设备，国外已把它们作为高分辨率彩色图形的标准。其显示模式、随机访问存储器和监视器接口方面有着很强的可变性和兼容性。它们与 CGA、MDA 或 CGE400 等其他图形显示卡相比，在技术上和应用方面要优越得多，编程功能亦更加完善。已被广泛地用作为高档微机的显示终端。如 AST、COMPAQ 等 286/386 微型计算机就是配置这类显示卡。下面以 EGA 为主介绍其结构、寄存器控制及编程或汉字处理等方面的应用知识。

8.5.1 结构与控制原理

EGA 卡包括52 个超大规模的集成电路和一组非常复杂的寄

存器结构，这组寄存器之间有着相互作用，并可以进行各种组合控制，使得CRT产生多种多样的显示效果。EGA卡没有采用M6845芯片作为CRT控制器，而是采用TTL门阵列作为CRT的控制逻辑。在其显示器的控制卡内部装有一个准16位的CPU处理器，以有效地控制各种屏幕操作。在80286/386主CPU的控制下，其I/O端口寄存器可接收16位的机器指令。

EGA/VGA显示卡若配置了多功能的监视器，将可以设置多种显示方式。在文本方式下，除了兼容CGA、MDA的视频模式外，还可显示高达132×43或100×70的字符模式。图形方式也兼容CGA、MDA，若是EGA⁺或VGA⁺显示卡，其图形分辨率为640×350、640×480，甚至高达1024×768点阵。这种增强型的显示适配器，能够模拟各种显示卡的显示方式，根据不同的显示方式，其随机访问的视频缓冲区首地址亦不同，通常有如下几种。

B0000H　模拟单色显示器(MDA)

B8000H　模拟彩色显示器(CGA)

A0000H　高分辨率彩色图形显示器(EGA/VGA)

这三种视频RAM的起始地址是由CRT初始化程序进行切换和定义的，也可以通过寄存器控制来实现。但在指定的模式下，只能在其相应的视频RAM中读写，其他未定义的视频地址不能作为存储器使用。

显示控制卡上装有一个8KB的ROM芯片，其中，2KB为各种类型的ASCII字符点阵字模，6KB为视频BIOS控制程序。视频BIOS起始地址在C0000H，该段内还含有约30几种CRT模式所需要的视频参数，每一种显示模式由64个字节的参数组成。这些参数分别用于显示方式、光标定位、色彩和扫描频率等控制，并且兼容于系统配置的各类监视器。表8-8列出了EGA/VGA常用的图形显示方式，供读者参考，有些兼容卡功能号可能有些不同，表中有“*”号的，作为参考标识，这些方式均可通过视频BIOS的0号初始化功能进行转换。

EGA/VGA对屏幕扫描方式仍采用隔行扫描技术，但与其

EGA/VGA 图形模式表　　表 8-8

BIOS 功能号	方　式	图形分辨率	字符	行/列数	视频RAM地址
AX=0004H	CGA	320×200	8×8	40×25	B800:0—3EEF
AX=0006H	CGA	640×200	8×8	80×25	B800:0—3EEF
AX=000DH	EGA	320×200	8×8	40×25	A000:0—1F3F
AX=000EH	EGA	640×200	8×8	80×25	A000:0—3E7F
AX=0010H	EGA	640×350	8×14	80×25	A000:0—6D5F
AX=0012H	VGA	640×480	8×14	80×30	A000:0—95FF
*AX=005BH	EGA/VGA	800×600	8×8	100×42	A000:0—E9FD
*AX=0052H	EGA/VGA	1024×480	8×8	128×84	A000:0—EFFF

他显示卡不同的是：奇数场的扫描是嵌在偶数场的扫描线之间。控制逻辑配合高速的31.5kHz光栅扫描，有足够的带宽以较高的速度刷新像点信息，消除了非隔行扫描所产生的黑带现象，从而提高了图像的分辨率和清晰度。

在图形方式下，对于视频缓冲区，却是以逐行扫描方式进行的。即从左至右，从上至下，不分奇行、偶行，依次存放，顺序读写。通常，EGA/VGA在高档微机上使用，配有256KB的视频RAM，分为4个彩色平面板，如图8-9所示。每个平面板占64KB，分别表示I、R、G、B4种颜色，4位二进制码在屏幕上组合出16种色彩。

进一步说，每个像点需要占4个像位，8个像位一个字节，若在1024×480高分辨率图形方式下，至少需要M=1024×480×4/8=246KB字节的RAM存贮区来显示全屏幕的图像。虽然这种彩色平面分层次的存贮器结构类似Color400，但是，存储器随机访问的方式却不同。当读写视频缓冲区时，EGA是通过控制器内部CPU处理后间接进行的。倘若对指定I/O寄存器设置某种写模式，则可以有效地选择各种随机访问视频RAM的方式，可变

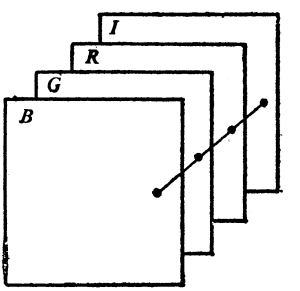


图 8-9　彩色平面图

的位图存贮器访问模式，为优化底层点位的执行提供了多种手段。其中，处理器的数据模式允许单独读写每一个彩色平面板，还提供了用特定的颜色同时写所有彩色平面板的方式，但不能直接读取所有平面板的像位，只有通过对寄存器的设置来切换4个彩色平面，分别读出每个平面板在同一地址的像位，组合成一个最终的彩色像点字节。

8.5.2 EGA寄存器设置

EGA显示卡占用了3C0H—3CFH16个端口地址，作为其寄存器寻址。其中，3C0H、3C1H为彩色信号寄存器，3C4H、3C5H为平面板编址寄存器，3CCH、3CAH为图形位置寄存器，3CEH、3CFH内包含9个图形控制寄存器。随着计算机硬件技术和设备的发展，势必有越来越多的诸如此类的显示适配器产生，并且其功能也必然会有所增强，其控制参数信息，对于用户和高级程序员来说尤为重要。在此介绍EGA常用寄存器控制方法，这对后面讨论的编程问题和汉字移植是很有帮助的。

1. 彩色板信号线控制

EGA显示卡除了红、绿、蓝三种基本色根外，又增加了三种辅助色彩(Secondary Colors)。所谓辅助色彩，是指硬件调节处理出与原色根浓度不同的色彩，这是在原色根的基础上产生的辅助红(R′)、绿(G′)、蓝(B′)，这样，就形成了6位彩色二进制编码RGBR′G′B′。其中RGB表示三分之二亮度的红、绿、蓝色，R′G′B′表示三分之一亮度的红、绿、蓝色。在显示卡与监视器之间有6根彩色信号线，分别对应这6个彩色码，信号线的接通与断开是由寄存器的置位与变位来决定的。

3C0H是彩色信号控制寄存器，它被用来选择由3C1H信号线寄存器改变当前屏幕的前景、背景或者边框的颜色，并含有一个视频选通位。3C0H寄存器是以组合置位的方式控制屏幕景色的，其组合的方式有如下几种：① b_0、b_1、b_2置位设置前景色彩；②b_0、b_4置位设置边框色彩；③全字节变位可以控制背景色

彩的变换；④b_5位作为选通位，该位置1表示信号线已联机，允许显示。3C1H是彩色信号的设置，其中各位的意义如图8-10所示。

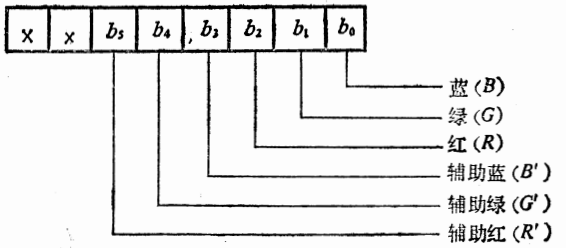


图 8-10 3C1H彩色线色彩位

这6个色码位可以进行任意组合，通过3C0H寄存器的景色控制，在CRT屏幕上，前景、背景和边框分别由表8-9所示的64种颜色显示出来。

寄存器彩色编码表　　表 8-9

R'G'B' \ RGB	000	001	010	011	100	101	110	111
000	黑	蓝	绿	青	红	紫	草绿	浅灰
001	深蓝	纯蓝	深青绿	青蓝	偏红紫	紫蓝	灰黄	淡蓝
010	深绿	深青蓝	纯绿	青绿	棕黄	灰紫	偏黄绿	淡绿
011	深青	偏青蓝	偏青绿	天蓝	红灰	浅紫蓝	浅黄绿	淡天蓝
100	深红	蓝紫	偏黄绿	浅青灰	纯红	偏粉红	菊黄	淡红
101	深紫	偏紫蓝	浅灰绿	浅青蓝	偏紫红	纯粉红	浅菊黄	淡粉红
110	深草绿	蓝灰	黄绿	浅青绿	菊红	浅紫红	纯黄	淡黄
111	深灰	浅蓝	浅绿	浅天蓝	浅红	浅粉红	浅黄	白

2. 图屏蔽寄存器(Map Mask Register)

图屏蔽寄存器口地址是3C5H，它是一个只用了低4位的只写寄存器。通过3C4H寄存器的b_1位作为索引，向3C5H发送数据之前，应先向3C4H写入02，便打开了4个彩色平面板的开关。3C5H寄存器的b_0—b_3对应于0—3彩色平面，并且，b_0—b_3位可以在屏幕上组合出16种不同颜色。采用不同的写模式，便可以

获得不同的显示效果。其寄存器的控制方法如下：

```
MOV   DX,3C4H        ；索引口
MOV   AL,02          ；打开图屏蔽
OUT   DX,AL
INC   DX             ；指向3C5H口
MOV   AL,04          ；红色图平面
OUT   DX,AL
  ⋮
```

当这段程序执行后，紧接着向视频缓冲区送任何数据都将以红色显示。

3. 图形控制寄存器

当4个字节数据处于视频RAM同一地址处，怎样才能对4个不同字节执行写操作呢？这就需要对图形控制寄存器进行设置，选择适当的写模式来实现。因为4个字节数据并非顺序地被送到该地址处，当接收到CPU发来的一字节数据时，几种写模式的任意一种均能改变全部4个字节。CPU数据的作用由几个寄存器的设置状态决定。其中，位屏蔽和图屏蔽寄存器决定了哪位和哪个平面将被修改。

若要掌握这些寄存器的工作原理，应该了解锁存寄存器，锁存寄存器内为最后访问的存贮器位置处的位平面（位平面即指视频缓冲区的全部内容，也指锁存寄存器中临时保存的缓冲区——字节副本）。当CPU向一特定的地址送数据时，这些数据就将改变成完全取代锁存寄存器的原有数据，然后，锁存寄存器中的数据就被写入视频缓冲区。根据写模式和其他寄存器建立的状态不同，锁存寄存器受CPU数据的影响也不同。当读视频缓冲区的某地址时，锁存寄存器就被该地址处4个位平面中的4个字节所填充。由于锁存寄存器易于控制，因此，其内容可以被进行布尔逻辑（AND、OR、XOR）和循环等操作，这样就大大加强了特技图形和滚行的功能。

图形控制寄存器以3CEH作为索引功能来选择9个图形寄存

器号，然后向3CFH数据寄存器中发送控制信息。这里，*n是指用3CEH选定的图形寄存器号，其功能由3CFH的各位控制，如下所述：

*0 置位/复位寄存器。仅用于写模式0，3CFH寄存器的0—3位控制位平面0—3，如果b_0—b_3的某一位置1，由功能*1使能，某一位将被收入由*5功能设置的彩色平面。该功能设置是由#5功能写的颜色。

*1 置位/复位使能寄存器。即3CEH的b_0位索引，3CFH寄存器的b_0—b_3位对应4个位平面。若被禁止，则禁止位平面的颜色将不受写操作而改变。即使一个彩色值由功能*0选择，也输出0FH到该寄存器。

*2 彩色比较寄存器。当3CEH b_1置位后，可用来从位平面读彩色值，用3CFHb_0—b_3位指定彩色值，若该彩色值与地址读出值比较后，相同的位为1，不相同的位为0。在视频RAM给定的地址寻找指定的色彩时，常用此功能。

*3 逻辑/循环寄存器。当3CEH的b_0、b_1置位后，3CFH寄存器的b_0—b_2位含有一个0—7的循环计数，它表示来自CPU的数据向左旋转的位数（是在*5功能写，执行*0操作时）。为改变一个单个像素，可以写一个0X01到显示地址，旋转它到一个字节内，即8个像素位置之一。该寄存器还含有以下几种逻辑运算来选择写模式，其控制位由3CFH的b_3、b_4位的置位状态决定。

00 未修改的数据；

01 和锁存器的内容相“AND”；

10 和锁存器的内容相“OR”；

11 和锁存器的内容相“XOR”；

*4 平面选择寄存器。用于读模式0，该寄存器的b_0—b_1，以选择所要读的4个平面板。

*5 模式寄存器。当3CEH b_0、b_2位置位后，将索引指针指向写模式选择寄存器，三种写模式可由3CFH的b_0、b_1位选择指定。

b_1、b_0 这两位所选定的参数与写模式的状态如下所示：

00 写模式0：默认状态。来自 *3 功能的数，表示处理器数据被旋转，根据图形控制器、锁存器或 *0 功能设置的值，8个位屏蔽设置被用来设置在所有位平面中的相应位，位平面使用 *1 功能使能。

01 写模式1：由读操作被装入锁存器的值不修改，被写到相应的位平面。用于快速读写，把显示内容从一个区域复制到另一个区域。

10 写模式2：写入的数据包含b_0—b_3的色彩值，该功能设置所有给定颜色选择的位平面中的所有8位。若用 *8 功能屏蔽，则可选择哪些位由锁存器提供，哪些位由写入的值修改。用于设置8个像素，送到一个指定的色彩上。

11 未使用（VGA为写模式3）

该寄存器的b_2可指定一个测试条件，一般情况下此位置0。

该寄存器b_3指定读模式。在从视频缓冲区内读一个字节时，若置$b_3=0$为读模式0，将从由 *4 功能所选择的平面读数据。若置$b_3=1$为读模式1，用 *2 功能指定的颜色值决定哪些位被设置。

b_4指定奇/偶或顺序寻址，$b_4=0$使用奇/偶寻址，$b_4=0$，使用顺序寻址。

b_5用来产生一个带有4种颜色的CGA兼容模式。每个像元有两个邻接的位。为了便于理解，在图8-11中列出了#5寄存器各位功能的意义。

*6 映射寄存器：当3CEH寄存器b_1、b_2置1后，即索引到 *6 功能寄存器，这时，若3CFH数据位b_0为1，字符发生器的锁存器被禁止，若b_0为0，字母、数据图形被选择。

如果$b_1=1$，奇映射被链接到偶映射的后面，即逐行扫描方式。否则，就同在CGA方式一样，映射2接着映射0，映射3接着映射1，以此类推，形成隔行扫描方式。换句话说，用这一位选择逐行扫描或者隔行扫描方式。

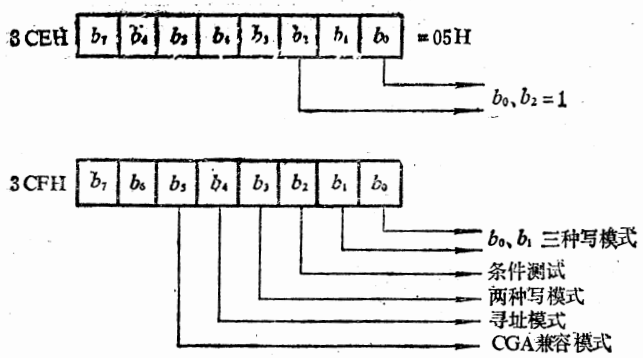


图 8-11 模式寄存器 #5 各位意义

该寄存器的 b_2、b_3 位选择视频缓冲区映射基地址，这两位的置位所对应的视频缓冲区的基地址如下：

00 A000H是高分辨率128—256KB 基地址

01 A000H是高分辨率图形64KB 基地址

10 B000H是单色32KB 文本基地址

11 B800H是CGA兼容的 32KB 图形基地址

#7 颜色无关寄存器。当 3CEH b_0、b_1、b_2 置为 1 时，在读模式 1 下，寄存器中 b_0—b_3 被置位的对应平面将不被使用，即比较操作舍去相应的位平面。

#8 位屏蔽寄存器。当索引寄存器 3CEH 的 b_3=1(08H) 时，被置为 1 的位被数据位修改后写回缓冲区。而被置 0 的位将不被修改由锁存器中再写入缓冲区。常用于保护背景色彩，首先读视频缓冲区指定的地址值，以便装入锁存器中，根据指定的修改值再写回该地址。

8.5.3 EGA显示器编程要点

通常，对EGA/VGA 显示器的编程步骤为：

①求出读/写像点的视频地址；

②对相关的图形寄存器置位；

③读/写视频缓冲区地址；

④恢复默认状态。

无论 EGA 显示卡内的寄存器编程如何复杂，最终不外乎读和写这两种操作，只是，通过寄存器设置，可以得到不同的读写效果。只要掌握各种读写模式的要点及用途，在编程时就会运用自如。以下通过简单的实例，来说明几种读写模式的编程方法。

(1) 写模式 0。写模式 0 的作用是把 CPU 发送到视频缓冲区的数据同时写入 4 个平面中，如下列程序所示。

```
;写模式0:
;ES:DI=视频地址，BL=色彩值
    MOV     DX,3CE
    MOV     AL,05
    OUT     DX,AL
    INC     DX
    MOV     AL,00
    OUT     DX,AL       ;置写模式0
    MOV     DX,3CE
    MOV     AL,08
    OUT     DX,AL       ;选择位屏蔽寄存器
    INC     DX
    MOV     AL,80H      ;选择屏蔽位
    OUT     DX,AL
    MOV     AL,ES:[DI]  ;读入锁存字节
    MOV     AL,00
    MOV     ES:[DI],AL  ;清原色彩
    MOV     DX,3CE
    MOV     AL,02
    OUT     DX,AL       ;选择映像寄存器
    INC     DX
    MOV     AL,BL       ;取彩色值
    OUT     DX,AL
    MOV     ES:[DI],FF  ;写入该像素
    RET
```

当所有位及位平面都被使能时（默认状态），将FFH送入视频缓冲区，4个位平面的每位都置1，这样，每个对应像素的位组合格式都是1111B。这意味着相应的8个像素都以彩色码15（白色）显示。

（2）写模式1。该模式一般作为特殊用途，在此模式下，锁存寄存器的当前位被写入指定的地址，如下面程序所示。

```
;写模式1:
        MOV     DX,3CE
        MOV     AL,05
        OUT     DX,AL
        INC     DX
        MOV     AL,01
        OUT     DX,AL                   ;置写模式1
        MOV     CX,50H                  ;一行字节数
LP:
        MOV     AL,ES:[DI]              ;取一字节
        MOV     ES:[DI+50H],AL          ;写入下一行线
        INC     DI
        LOOP    LP
        RET
```

因为锁存器的内容是由读操作来获得的，因此，在执行屏幕滚行操作需要快速传送数据时，这个模式非常有用。在图形或者汉字系统下，可用该模式配合256KB大缓冲区作为图形窗口漫游和汉字菜单快速翻页等功能，其效果极好。而在写模式1状态下，位屏蔽寄存器和图屏蔽寄存器均不起作用，并且，CPU向指定缓冲区地址送何值也无关紧要，这是因为锁存器中的内容未加修改就被转存了。

（3）写模式2。写模式2提供了一种可供选择的设置单个像素点的方法。CPU发送来的彩色值中，只有低4位是有效的。而且，这4位信息作为彩色代码，这意味着位组合格式被插入4个位平面。如果位屏蔽寄存器没被设置成防止修改某些像素的状态，那么，位组合格式就将被复制到视频RAM相应地址的全部

8个像素位置处。图屏蔽寄存器是当前有效的。虽然，CPU必须将整个字节送到缓冲区地址处，但只有低4位是有效色彩值，如下面程序所示。

```
;写模式2
;ES:DI=视频地址，Color=色彩值
        MOV     DX,3CE
        MOV     AL,05
        OUT     DX,AL
        INC     DX              ;指向3CF口
        MOV     AL,02
        OUT     DX,AL           ;置写模式2
        MOV     DX,3CE
        MOV     AL,08
        OUT     DX,AL           ;选择位屏蔽寄存器
        INC     DX
        MOV     AL,80H
        OUT     DX,AL           ;屏蔽b0—b6位
        MOV     AL,ES:[DI]      ;锁存当前色彩
        MOV     AL,Color
        MOV     ES:[DI],AL      ;写入彩色值
        RET
;与背景色彩异或
        MOV     DX,3CE
        MOV     AL,03
        OUT     DX,AL
        INC     DX
        MOV     AL,18H
        OUT     DX,AL           ;置"异或"逻辑
        DEC     DX
        MOV     AL,00
        OUT     DX,AL           ;置位/复位口
        INC     DX
        MOV     AL,Color
        OUT     DX,AL           ;置写入色彩值
```

```
        DEC     DX
        MOV     AL,08
        OUT     DX,AL           ; 置位屏蔽寄存器
        MOV     AL,80H
        OUT     DX,AL           ; 异或高8位像素
        MOV     AL,ES:[DI]      ; 锁存当前值
        MOV     ES:[DI],FF      ; 异或写入
        RET
```

(4) 读模式0。该模式所读的内容将返回指定缓冲区地址处任一位平面的字节，要求首先设置图平面选择寄存器，来指定要读的位平面。读模式0至少需要4次读操作，如下列程序所示。

```
; 读模式0:
; ES:DI=视频地址，CX=色彩
        MOV     AH,3            ; 4个平面
Red1:
        MOV     DX,3CE
        MOV     AL,4
        OUT     DX,AL           ; 选择位平面寄存器
        INC     DX
        XCHG    AH,AL
        OUT     DX,AL
        MOV     AL,ES:[DI]      ; 读指定平面像元值
        SHL     CL,1
        AND     AL,CH           ; 屏蔽无关位
        JZ      Red2
        OR      CL,1
Red2:
        DEC     AH              ; 位平面号减1
        JGE     Red1            ; 继续读
        AND     CX,0FH          ; 清除无用位
        RET
```

(5) 读模式1。该模式具有寻找指定彩色码像素值的功能。首先将要寻找的彩色码的位组合格式送入彩色比较寄存器，此代

码只置位于该寄存器的低4位，而高4位无效。当读指定的地址时，就会返回一个字节的数据，如下列程序所示。

```
;读模式1:
;ES:DI=视频地址，CX=色彩值
        PUSH    CX
        MOV     DX,3CE
        MOV     AL,5
        OUT     DX,AL
        INC     DX
        MOV     AL,8
        OUT     DX,AL           ;置读模式1
        MOV     DX,3CE
        MOV     AL,2
        OUT     DX,AL           ;选择比较寄存器
        INC     DX
        POP     AX
        OUT     DX,AL           ;置色彩比较值
        MOV     AL,ES:[DI]      ;锁存比较像素
        AND     AL,CH           ;与屏蔽字节比较
        JZ      Rea1
        MOV     CX,1            ;相同，CX置1
        JMP     Rea2
Rea1:
        XOR     CX,CX           ;不相同，CX置0
Rea2:
        MOV     DX,3CE
        MOV     AL,5
        OUT     DX,AL
        INC     DX
        MOV     AL,0
        OUT     DX,AL           ;恢复默认状态
        MOV     DX,3CE
        MOV     BL,8
        OUT     DX,AL
        INC     DX
```

```
MOV     AL,0FFH
OUT     DX,AL                    ;恢复屏蔽字节
RET
```

如果该字节中某像位与给定的彩色相符，则对应位为1。但若使用彩色无关寄存器，进行比较时，就可以舍弃彩色代码的一或几位。通常，该寄存器的低4位为1，若将这4位中的某位置0，相应的位平面内容就会被舍弃，即相应该位的位平面的内容不会影响比较结果。

8.5.4 EGA/VGA汉字显示模块的移植

因为EGA/VGA显示器具有各种图形显示方式，因此，原来在CGA显示器上使用的汉字操作系统仅作一些修改，便可以移植到EGA/VGA显示卡上。

一般，移植CC-DOS显示模块通过以下几个方面来实现：

①CRT初始化视频控制参数的调整；

②上滚和下滚功能的修改；

③显示彩色汉字点阵信息；

④ASCII字符点阵字模代换。

EGA/VGA视频参数的地址保存在BIOS数据区0040:00A8开始的双字节单元中，该单元的内容指向ROM-BIOS中的一个指针区，指针区内头4个字节所指的实际地址就是视频参数区的首地址。

当在程序中要设置指定的屏幕显示方式时，根据其功能号，BIOS以该首地址为基地址乘以40H（因为每种方式有64个显示参数），计算出相应的参数块的偏移量后，由定位的这组参数对有关寄存器进行一系列操作，达到实现初始化的目的。通常，用10H功能把其初始化为640×350的图形方式，遗憾的是只能显示21行汉字，使得许多应用软件（25行）不能很好地使用。

为了能够显示25行汉字，至少要把屏幕分辨率设置为640×400（即16点×25行=400线）。表8-10所示的一组参数，就是用

10H号功能把屏幕分辨率置为640×401点阵。400线用于显示25行汉字信息，余一线作为提示行的分隔线。

EGA 置设640×401分辨率的视频参数　　表 8-10

```
DS:0520  50  18  10  00 80 01 0F 00-06  A7 5B 4F 53 37 52 00
DS:0530  A0  1F  00  00 00 00  00 00-00  00 93 2B 92 28 0F 94
DS:0540  00  E3  FF  00 01 02  03 04-05  14 07 38 39 3A 3B 3C
DS:0550  3D  3E  3F  01 00 0F  00 00-00  00 00 00 00 05 0F FF
```

表中有下划线的参数是在原参数基础上进行了调整。实际上，调整了显示器垂直扫描的参数，实现了增加纵向扫描线的功能。这些参数可用Debug程序通过中断10H跟踪取出，通过对有关参数的调整后，再写入CC-BIOS某一个区域内。由于ROM区内的参数及程序不允许修改，可采用RAM参数代换的方法来实现。编程思想为：把原ROM-BIOS内有关几组的参数驻留在内存RAM某个区域，再把表8-10所用的参数覆盖对应功能的视频参数，最后把0040:00A8的地址指针指向RAM中的该参数表。当用10H功能初始化屏幕时，就可以显示640×401分辨率，建立良好的汉字运行环境。

屏幕滚动方式是把下面一行的内容按扫描行一一重写到上面一行去，最末一行填写全零字节。若是全屏幕滚动，上述的处理要循环执行24行。在EGA/VGA显示卡上，如果不加以控制，滚动信息的各种色彩将会全都变为当前色彩，失去了彩色显示的效果。

为了保障屏幕滚动时原有信息的色彩保持不变，在滚动之前，把寄存器设置为写模式1，这样，读出的内容放置在锁存器中不作任何修改再写入指定的目标地址去。等滚动传送结束后，再重新置为写模式0，使得锁存寄存器解锁，用此方法实现了彩色画面的滚动，如下面两段程序所示。

```
; ……锁存色彩位
CS:2347          MOV     DX,03CE
CS:234A          MOV     AL,05
```

```
CS:234C        OUT     DX,AL
CS:234D        INC     DX
CS:234E        MOV     AL,01          ; 写模式1
CS:2350        OUT     DX,AL
CS:2451        RET
; ……解锁存
CS:2356        MOV     DX,03CE
CS:2359        MOV     AL,05
CS:235B        OUT     DX,AL
CS:235C        INC     DX
CS:235D        MOV     AL,00          ; 写模式0
CS:235F        OUT     DX,AL
CS:2360        RET
```

在屏幕上显示汉字或ASCII字符点阵时，若考虑其显示属性的话，则需要对有关寄存器进行置位。在原系统向视频RAM写数据模块之前插一段程序，来实现彩色平面选择等功能。EGA/VGA视频段地址在A000H，并且，视频RAM的刷新是采用的逐行扫描方式。因此，每一行汉字以500H的地址增量为单元，来定位行坐标。下列程序说明了在EGA/VGA显示卡上向屏幕送彩色汉字信息的技术，其中BL是用户在应用系统中指定的彩色值，ES:DI为指向视频缓冲区的地址。

当显示汉字时，需要调用两次该程序，一旦显示完毕，还必须把寄存器状态设置为默认状态。这样将可以在同一屏幕上显示16种前景和16种背景色彩的汉字信息。

```
CS:A768  E818FF     CALL    A683        ; 置色彩和屏蔽位
CS:A76B  57         PUSH    DI
CS:A76C  56         PUSH    SI
CS:A76D  B611       MOV     DH,10       ; 共送16个字节
CS:A76F  AC         LODSB               ; 取1个点阵字节
CS:A770  26         ES:
CS:A771  8A25       MOV     AH,[DI]     ; 锁存当前色彩值
CS:A773  AA         STOSB               ; 写1个字节
CS:A774  033E4A00   ADD     DI,[004A]   ; 地址减50H
```

```
CS:A778  4F          DEC     DI
CS:A779  FECE        DEC     DH
CS:A77B  75F2        JNZ     A76F        ; 写屏至DH=0
CS:A77D  5E          POP     SI
CS:A77E  5F          POP     DI
   :                   :
CS:A67A  86C4        XCHG    AL,AH       ; 设置寄存器
CS:A67C  EE          OUT     DX,AL
CS:A67D  42          INC     DX
CS:A67E  86C4        XCHG    AL,AH
CS:A680  EE          OUT     DX,AL
CS:A681  4A          DEC     DX
CS:A682  C3          RET
CS:A683  B603        MOV     DH,03
CS:A685  F6C380      TEST    BL,80       ; 异或标志
CS:A688  740A        JZ      A694
CS:A68A  B2CE        MOV     DL,CE       ; DX=3CE
CS:A68C  B82003      MOV     AX,0320     ; 和背景相或
CS:A68F  E8E8FF      CALL    A67A
CS:A692  EB19        JMP     A6AD
CS:A694  B2C4        MOV     DL,C4       ; DX=3C4
CS:A696  B80F02      MOV     AX,020F     ; 写位平面
CS:A699  E8DEFF      CALL    A67A
CS:A69C  2BC0        SUB     AX,AX
CS:A69E  57          PUSH    DI
CS:A69F              PUSH    CX
CS:A6A0  B91100      MOV     CX,0010
CS:A6A3  AA          STOSB               ; 清字符窗口
CS:A6A4  033E4A00    ADD     DI,[004A]
CS:A6A8  4F          DEC     DI
CS:A6A9  E2F8        LOOP    A6A3
CS:A6AB  59          POP     CX
CS:A6AC  5F          POP     DI
CS:A6AD  B2C4        MOV     DL,C4
CS:A6AF  B402        MOV     AH,02       ; 写平面
```

```
CS:A6B1  8AC3        MOV     AL,BL     ; BL=彩色平面值
CS:A6B3  E8C4FF      CALL    A67A      ; 置彩色平面
CS:A6B6  C3          RET
```

最后，是把8×14点阵的ASCII点阵字模代换原CC-DOS内的8×8点阵字模。在ROM-BIOS中找到该点阵字模存贮区域，加入CC-DOS，或用其他方法用中断1FH的向量指向字模的首地址，还可以BIOS的功能调用该字模等，把原系统以8为字符代码的地址增量改为14，在向字模加工区传送时，先送两个字节的0，表示两根空白线，然后再把14个字节顺序送出。当在汉字系统下运行时，ASCII字符的质量与在西文方式下运行时相同，与汉字的比例也比较匀称。

从用户界面及汉字和字符的显示质量来看，通过这样移植的汉字系统，比把字库信息进行压缩、抽线等方法实现的25行显示的效果要好得多。

附录 I　二、十、十六进制数转换表

十进数	二进数	十六进数	十进数	二进数	十六进数
0	00000000	00	40	00101000	28
1	00000001	01	41	00101001	29
2	00000010	02	42	00101010	2A
3	00000011	03	43	00101011	2B
4	00000100	04	44	00101100	2C
5	00000101	05	45	00101101	2D
6	00000110	06	46	00101110	2E
7	00000111	07	47	00101111	2F
8	00001000	07	48	00110000	30
9	00001001	09	49	00110001	31
10	00001010	0A	50	00110010	32
11	00001011	0B	51	00110011	33
12	00001100	0C	52	00110100	34
13	00001101	0D	53	00110101	35
14	00001110	0E	54	00110110	36
15	00001111	0F	55	00110111	37
16	00010000	10	56	00111000	38
17	00010001	11	57	00111001	39
18	00010010	12	58	00111010	3A
19	00010011	13	59	00111011	3B
20	00010100	14	60	00111100	3C
21	00010101	15	61	00111101	3D
22	00010110	16	62	00111110	3E
23	00010111	17	63	00111111	3F
24	00011000	18	64	01000000	40
25	0001 001	19	65	01000001	41
26	00011010	1A	66	01000010	42
27	00011011	1B	67	01000011	43
28	00011100	1C	68	01000100	44
29	00011101	1D	69	01000101	45
30	00011110	1E	70	01000110	46
31	0001 11	1F	71	01000111	47
32	00100000	20	72	01001000	48
33	0010.001	21	73	0100.001	49
34	00100010	22	74	01001010	4A
35	00100011	23	75	01001011	4B
36	00100100	24	76	01001100	4C
37	00100101	25	77	01001101	4D
38	00100110	26	78	0 001110	4E
39	00100111	27	79	01001111	4F

（续附录Ⅰ表）

十进数	二进数	十六进数	十进数	二进数	十六进数
80	01010000	50	125	01111101	7D
81	01010001	51	126	01111110	7E
82	01010010	52	127	01111111	7F
83	01010011	53	128	10000000	80
84	01010100	54	129	10000001	81
85	01010101	55	130	10000010	82
86	01010110	56	131	10000011	83
87	01010111	57	132	10000100	84
88	01011000	58	133	10000101	85
89	01011001	59	134	10000110	86
90	01011010	5A	135	10000111	87
91	01011011	5B	136	10001000	88
92	01011100	5C	137	10001001	89
93	01011101	5D	138	10001010	8A
94	01011110	5E	139	10001011	8B
95	01011111	5F	140	10001100	8C
96	01100000	60	141	10001101	8D
97	01100001	61	142	10001110	8E
98	01100010	62	143	10001111	8F
99	01100011	63	144	10010000	90
100	01100100	64	145	10010001	91
101	01100101	65	146	10010010	92
102	01100110	66	147	10010011	93
103	01100111	67	148	10010100	94
104	01101000	68	149	10010101	95
105	01101001	69	150	10010110	96
106	01101010	6A	151	10010111	97
107	01101011	6B	152	10011000	98
108	01101100	6C	153	10011001	99
109	01101101	6D	154	10011010	9A
110	01101110	6E	155	10011011	9B
111	01101111	6F	156	10011100	9C
112	01110000	70	147	10011101	9D
113	01110001	71	158	10011110	9E
114	01110010	72	159	10011111	9F
115	01110011	73	160	10100000	A0
116	01110100	74	161	10100001	A1
117	01110101	75	162	10100010	A2
118	01110110	76	163	10100011	A3
119	01110111	77	164	10100100	A4
120	01111000	78	165	10100101	A5
121	01111001	79	166	10100110	A_6
122	01111010	7A	167	10100111	A7
123	01111011	7B	168	10101000	A_8
124	01111100	7C	169	10101001	A9

十进数	二进数	十六进数	十进数	二进数	十六进数
170	10101010	AA	215	11010111	D7
171	10101011	AB	216	11011000	D8
172	10101100	AC	217	11011001	D9
173	10101101	AD	218	11011010	DA
174	10101110	AE	219	11011011	DB
175	10101111	AF	220	11011100	DC
176	10110000	B0	221	11011101	DD
177	10110001	B1	222	11011110	DE
178	10110010	B2	223	11011111	DF
179	10110011	B3	224	11100000	E0
180	10110100	B4	225	11100001	E1
181	10110101	B5	226	11100010	E2
182	10110110	B6	227	11100011	E3
183	10110111	B7	228	11100100	E4
184	10111000	B8	229	11100101	E5
185	10111001	B9	230	11100110	E6
186	10111010	BA	231	11100111	E7
187	10111011	BB	232	11101000	E8
188	10111100	EC	233	11101001	E9
189	10111101	BD	234	11101010	EA
190	10111110	BE	235	11101011	EB
191	10111111	BF	236	11101100	EC
182	11000000	C0	237	11101101	ED
193	11000001	C1	238	11101110	EE
194	11000010	C2	239	11101111	EF
195	11000011	C3	240	11110000	F0
196	11000100	C4	241	11110001	F1
197	11000101	C5	242	11110010	F2
198	11000110	C6	243	11110011	F3
199	11000111	C7	244	11110100	F4
200	11001000	C8	245	11110101	F5
201	11001001	C9	246	11110110	F6
202	11001010	CA	247	11110111	F7
203	11001011	CB	248	11111000	F8
204	11001100	CC	249	11111001	F9
205	11001101	CD	250	11111010	FA
201	11001110	CE	251	11111011	FB
207	11001111	CF	252	11111100	FC
208	11010000	D0	253	11111101	FD
209	11010001	D1	254	11111110	FE
210	11010010	D2	255	11111111	FF
211	11010011	D3			
212	11010100	D4			
213	11010101	D5			
214	11010110	D6			

附录Ⅱ　功能扩展键扫描码表

扩展键	扫描码（十进制）	扫描码（十六进制）
F1	59D	0; 3BH
F2	60D	0; 3CH
F3	61D	0; 3DH
F4	62D	0; 3EH
F5	63D	0; 3FH
F6	64D	0; 40H
F7	65D	0; 41H
F8	66D	0; 42H
F9	67D	0; 43H
F10	68D	0; 44H
F11	133D	0; 85H
F12	134D	0; 86H
Ctr-F1	94D	0; 5EH
Ctr-F2	95D	0; 5FH
Ctr-F3	96D	0; 60H
Ctr-F4	97D	0; 61H
Ctr-F5	98D	0; 62H
Ctr-F6	99D	0; 63H
Ctr-F7	100D	0; 64H
Ctr-F8	101D	0; 65H
Ctr-F9	102D	0; 66H
Ctr-F10	103D	0; 67H
Ctr-F11	137D	0; 89H
Ctr-F12	138D	0; 8AH
Alt-F1	104D	0; 68H
Alt-F2	105D	0; 69H
Alt-F3	106D	0; 6AH
Alt-F4	107D	0; 6BH
Alt-F5	108D	0; 6CH
Alt-F6	109D	0; 6DH
Alt-F7	110D	0; 6EH
Alt-F8	111D	0; 6FH
Alt-F9	112D	0; 70H
Alt-F10	113D	0; 71H

（续附录Ⅱ表）

扩展键	扫描码（十进制）	扫描码（十六进制）
Alt-F11	139D	0; 8BH
Alt-F12	140D	0; 8CH
Home	71D	0; 47H
Up	72D	0; 48H
Pg Up	73D	0; 49H
Left	75D	0; 4BH
Right	77D	0; 4DH
End	79D	0; 4FH
Down	80D	0; 50H
PgDn	81D	0; 51H
Ins	82D	0; 52H
Del	83D	0; 53H
Alt-A	30D	0; 1EH
Alt-B	48D	0; 30H
Alt C	46D	0; 2EH
Alt D	32D	0; 20H
Alt E	18D	0; 12H
Alt F	33D	0; 21H
Alt-G	34D	0; 22H
Alt-H	35D	0; 23H
Alt-I	23D	0; 17H
Alt-J	36D	0; 24H
Alt K	37D	0; 25H
Alt-L	38D	0; 26H
Alt-M	50D	0; 32H
Alt-N	49D	0; 31H
Alt-O	24D	0; 18H
Alt-P	25D	0; 19H
Alt-Q	16D	0; 10H
Alt-R	19D	0; 13H
Alt-S	31D	0; 1FH
Alt-T	20D	0; 14H
Alt-U	22D	0; 16H
Alt-V	47D	0; 2FH
Alt-W	17D	0; 11H
Alt-X	45D	0; 2DH
Alt-Y	21D	0; 15H
Alt-Z	44D	0; 2CH

（续附录Ⅰ表）

扩展键	扫描码（十进制）	扫描码（十六进制）
Shift-F1	84D	0; 54H
Shift-F2	85D	0; 55H
Shift-F3	86D	0; 56H
Shift-F4	87D	0; 57H
Shift-F5	88D	0; 58H
Shift-F6	89D	0; 59H
Shift-F7	90D	0; 5AH
Shift-F8	91D	0; 5BH
Shift-F9	92D	0; 5CH
Shift-F10	93D	0; 5DH
Shift-F11	135D	0; 87H
Shift-F12	136D	0; 88H
Alt-1	120D	0; 78H
Alt-2	121D	0; 79H
Alt-3	122D	0; 7AH
Alr-4	123D	0; 7BH
Alt-5	124D	0; 7CH
Alt-6	125D	0; 7DH
Alt-7	126D	0; 7EH
Alt-8	127D	0; 7FH
Alt-9	128D	0; 80H
Alt-0	129D	0; 81H
Alt-—	130D	0; 82H
Alt-=	131D	0; 83H
Ctr-Left	115D	0; 73H
Ctr-Right	116D	0; 74H
Ctr-End	117D	0; 75H
Ctr-PgDn	118D	0; 76H
Ctr-Home	119D	0; 77H
Ctr-PgUp	132D	0; 84H
Ctr-Up	141D	0; 8DH
Ctr--（小）	142D	0; 8EH
Ctr-5（小）	143D	0; 8FH
Ctr-+（小）	144D	0; 90H
Ctr-Down	145D	0; 91H
Ctr-Ins	146D	0; 92H
Ctr-Del	147D	0; 93H
Ctr-/（小）	149D	0; 95H
Alt-/（小）	164D	0; 0A4H

（续附录Ⅱ表）

扩展键	扫描码（十进制）	扫描码（十六进制）
Alt-Enter（小）	166D	0; 0A6H
Alt-Home（扩）	151D	0; 97H
Alt-Up（扩）	152D	0; 98H
Alt-PgUp（扩）	153D	0; 99H
Alt-Left（扩）	155D	0; 9BH
Alt-Right（扩）	157D	0; 9DH
Alt-End（扩）	159D	0; 9FH
Alt-Down（扩）	160D	0; 0A0H
Alt-PgDn（扩）	161D	0; 0A1H
Alt-Ins（扩）	162D	0; 0A2H
Alt-Del（扩）	163D	0; 0A3H
Alt-Tab	165D	0; 0A5H
Ctr-Tab	148D	0; 94H
Shilf-Tab	15D	0; 0FH
Ctr-2	3D	0; 03H

注：（小）——表示83键小键盘

（扩）——表示101扩展键盘中扩展盘

参 考 文 献

[1] 王敏生，浅谈建立汉化软件的理论，计算机信息报，1989.3。

[2] 李亦何，软件的汉化方法，计算机与自动化，1988.3。

[3] 何克清，计算机软件工程学，武汉大学出版社，1984。

[4] 赵珀璋、徐力，计算机中文信息处理（上册），宇航出版社，1987.12。

[5] 周明德等，微型计算机IBMPC系统原理及应用，清华大学出版社，1985.9。

[6] Leo Scanlon，IBMPC *and* XT *Assemble Language A Guide for Programmers*，1985。

[7] 张福炎等，微型计算机IBMPC的原理与应用，南京大学出版社，1985.9。

[8] *The IBM Personal Computer* AT *Technical Reference Manual*，IBM Corp.，1984.

[9] 杜毅仁等，十六位微型计算机，上海交通大学出版社，1984。

[10] 薛行译，IBMPC调试程序综述，微型计算机，1987.1。

[11] 钱培德，IBM-PC汉字信息处理系统，陕西电子编辑部，1988。

[12] 李亦何，在CC-BIOS系统中装接汉字输入编码的编程方案，微计算机应用，1989.5。

[13] 郭平欣、张淞芝，汉字信息处理技术，国防工业出版社，1985.12。

[14] 费翔林等，Turbo集成开发环境的分析和汉化，小型

微型计算机系统，1989，Vol.10，No. 7。
[15] 余娟芬，INFORMIX 的汉字化，中国计算机用户，1987.8。
[16] 冯劲，软件的汉化，微型机与应用，1988.4。
[17] 叶乃菶、张炘中。汉字微型计算机与汉字识别，机械工业出版社，1989.9。
[18] 李亦何，在 80286/386 扩展内存读取汉字的技术，计算机应用，1990.3。
[19] LQ-1500 *Operating Manul*，EPSON Corp.，1984。
[20] 李亦何，24针打印机无级变倍汉字驱动程序的设计，小型微型计算机系统，1988，Vol. 9，No. 11。
[21] 杨士强，IBMPC 系列机显示系统的发展及性能比较，计算机信息报，1990.7。
[22] 李亦何，CGE400彩色图形显示器的程序设计，微型机与应用，1988.4。
[23] 谢进一，点阵汉字放大的笔划平滑方案，中文信息，1990.2。
[24] 鲍明忠，高档微机开发指南，中国科学院希望公司，1990.3。
[25] 李亦何，EGA1024 显示器彩色图形编程方法，小型微型计算机系统，1989，Vol. 10，No. 3。
[26] Richard Wilton，*Programming the Enbanced Graphics Adapter*，BYTE，1985，Vol. 10，No. 11。