钱培德　陆建明　编著

DOS

汉字系统

高级技术

DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术

DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术
DOS 汉字系统高级技术

ISBN 7-5308-1533-4
TP·48 定价：8.75元

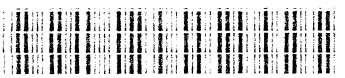
DOS 汉字系统高级技术

钱培德 陆建明 编著

天津科学技术出版社

DOS 汉字系统高级技术

钱培德 陆建明 编著



天津科学技术出版社

内 容 简 介

目前使用的计算机汉字系统绝大部分是DOS汉字系统，本书全面和深入地介绍DOS汉字系统中的高级技术，这些技术能够有效地增强系统功能、提高系统效率和改善系统性能。

本书主要内容包括：系统概述和汉字代码体系，DOS汉字系统总体设计，DOS内存管理技术及其优化，词输入处理技术和多级词库结构，联想输入处理技术和智能型汉字输入技术，多级汉字库结构设计，字形变换技术，假脱机打印子系统设计，汉字处理数学模型，汉字造字程序设计，字模压缩和还原技术，汉字编码辅助技术和保护方式编程技术。

本书突出先进性、技术性和实用性。本书可供计算机开发和应用人员参考，亦可作为大专院校计算机专业的教学参考书。

前　言

DOS 操作系统是 PC 系列微型计算机的主操作系统。由于 PC 系列机的广泛应用，以及 DOS 自身的短小精悍，再加上 DOS 拥有极为丰富的应用程序，因而使 DOS 自 80 年代至今一直是拥有最多用户的操作系统。

在我国，DOS 同样获得了非常广泛地应用，是使用最广泛的操作系统。我国是使用汉语的国家，因此，汉字系统成为我国计算机应用的支撑基础。由于 DOS 使用的普遍性，所以我们使用的汉字系统基本上是基于 DOS 这一汉字操作系统的。最典型的汉字系统要数 CC-DOS，另外还有 UC-DOS、2.13 系统、SP-DOS 等，它们均是基于 DOS，在 DOS 基础上扩充了处理汉字的功能，所以我们把它们称为 DOS 的汉化版本，也称为 DOS 汉字系统。

CC-DOS 是一个典型的 DOS 汉字系统，它采用一系列技术实现 DOS 的汉化，我们称这些技术为 DOS 汉字系统的基本技术。CC-DOS 推出后，很快获得了用户的青睐，并且引起了越来越多的人对它的兴趣。随着对 CC-DOS 剖析的深入，DOS 汉字系统的基本技术已被越来越多的人掌握，目前在 DOS 用户中已相当普及。

近年来，我国的计算机汉字信息处理技术获得了飞速发展，DOS 汉字系统也有较大的发展。在此过程中，DOS 汉字系统采用了一系列新技术，这些技术是基本技术的升级，故称为 DOS 汉字系统高级技术。这些技术的应用，使得 DOS 汉字系统的性能更加优越，用户使用更加方便。

本书全面和深入地介绍 DOS 汉字系统高级技术，以帮助读者对它们有深入的了解，使这些技术得到普及，把我国的计算机汉字信息处理技术不断推向前进。全书共分四个部分，每个部分主要内

容如下：

第一部分（第一章至第二章）是基础部分，主要介绍汉字信息处理的概况，DOS 操作系统及其汉化原理，汉字代码体系和通用字符编码的国际新标准。

第二部分（第三章至第四章）是总体部分，主要介绍 DOS 汉字系统的总体设计，全汉字集处理系统的总体设计、内码设计、输入输出处理和系统实现方法，以及 DOS 内存管理技术。

第三部分（第五章至第七章）是核心部分，主要介绍词输入处理和联想输入处理技术，多级词库的结构与性能分析，输入系统的智能化，键盘模块的标准化，VGA 显示模块的设计，汉字库结构设计与性能分析，多级汉字库设计技术，字形变换技术，假脱机打印子系统设计，汉字处理数学模型的建立。

第四部分（第八章）是辅助技术部分，主要介绍汉字造字程序设计，字模压缩和还原技术，汉字编码辅助技术，保护方式程序设计技术。

本书注意突出先进性、技术性和实用性，并且力求作到概念清楚和通俗易懂。本书适合于广大从事计算机研究、应用和开发的科技人员参考，也可以作为大专院校计算机专业的教材和教学参考书。

本书由钱培德和陆建明执笔，并由钱培德主持编写和最后修改定稿。另外，杨季文、吕强、方奕、鲁言民等参加了本书编写过程中的部分工作。

在写作本书的过程中，我们始终得到天津科学技术出版社的支持，特别是徐彤编辑为本书的出版付出了大量的劳动。我们谨在此表示由衷的感谢。

最后，希望使用这本书的同行们能对本书提出宝贵意见和建议，以便今后进行修正和充实，我们对此表示谢意。

作者

1993.6

目　录

第一章 绪 论

第一节 汉字信息处理概况

一、汉字信息处理的重要性

信息的含义十分广泛，它是人们认识的一个基本要素，是对客观世界的直接描述，也是在人们之间进行传递的一些汉字知识。当今社会的一切活动都离不开信息的收集、组织、存贮、加工、传输、再生等等。它和物质、能量一起构成客观世界的三大要素。

信息处理是对信息的接收、存贮、转化、传递等操作。由于信息量的日趋庞大，使我们面临一个“信息爆炸”的社会。然而，人脑虽然在信息的识别、分析、综合、推理等高智能方面有着不可比拟的能力，但在存贮记忆、数值计算、检索能力及处理速度等方面却不够理想。因此，迫切需要人们用现代化的先进设备、技术来处理信息。

在信息处理中，对文字信息的处理称之为文字信息处理，其中对汉字信息的处理称之为汉字信息处理。我国是文明古国，是汉字的发源地，使用汉字已有数千年的历史，目前世界上使用汉字的人愈来愈多，汉语也做为联合国所采用的六种语言之一。由于汉字是一种表意文字，所以其字形复杂，而且字体繁多。传统的汉字信息处理基本上还是以手工抄写或机械处理，需消耗大量的时间和精力，因此远远不能满足现代社会的需要。

汉字信息处理的现代含义是用现代化的技术设备——计算机

去处理汉字信息，如存贮、分类、统计、检索、转换、传输等。主要处理汉字的文本信息、图像信息和语音信息。随着现代科学技术的迅速发展，用计算机对汉字信息进行处理已势在必行，如果不解决计算机的汉字信息处理，那么计算机的使用在我国将会受到极大的限制，影响了国内及国际间的信息交流，因此研究和开发汉字信息处理技术具有十分重要的意义。

二、汉字信息处理的原理

汉字信息处理是一门跨学科、多学科的综合性学科，包括自动检索、人机对话、自动翻译、语音识别、文字识别、自动分词、人工编码输入、自然语言处理等一系列课题的研究，其基本原理和西文信息处理的原理类似，它先将汉字转化为数字信息代码，而后经计算机进行代码处理，最后再在输出设备上将处理过的数字信息代码恢复为字形信息输出。

1. 汉字转化为数字信息代码

计算机无论是以数值计算为主，还是以非数值性的数据处理为主，都以代码信息的方式处理，目前，计算机一般均以 ASCII 码（USA Standard Code For Information Interchange）为内码，但也有的是以 EBCDIC 码（Extended Binary — Coded — Decimal Interchange Code）。由于 ASCII 码字符集较小，所以每个字符可全部集中在西文小键盘上表示，但是，当计算机处理汉字时，由于汉字的数量多，致使计算机处理汉字比处理西文字符更为复杂。从汉字输入的角度来说，说是要设法抽取方块汉字的特征信息，给每个汉字设计合理的信息编码输入计算机，这个编码称为汉字的输入码，它实际上是将汉字转化为数字信息代码。

例如，王永民先生的“五笔”输入法，它是以汉字的笔划和构件（部件、字根）为基础的字形编码。如“植”字，根据“植”字的特征信息，它的输入码为“SFHG”。

就目前而言，汉字编码方案不下六百多种，而且新的方案还在

不断涌现，大有“万码奔腾”之势，其主要原因在于汉字输入速度的问题，也可以说是重码率、击键次数以及用户记忆量之间的矛盾未彻底解决。所以，如何使汉字转化为合理的数字信息代码，即给每一个汉字编码，这是汉字编码方案研究中的一个重要课题。

2. 计算机内汉字代码的处理

任何用于处理汉字信息的计算机，均必须能同时处理西文字符信息，以保证中西文兼容，不然，许多先进的西文软件无法直接得到应用。为了确保计算机能自动区分西文代码和汉字代码，首要的问题是设计合理汉字代码。计算机内最重要的代码是汉字内码，有关汉字内码的详情将在第二章中作详细的论述。目前国内主要是采用与 GB2312-80 基本相吻合的变形国标码，用两个字节代表一个汉字，且每个字节的最高位为 1，这样作使用简便，易于区分中西文代码。但也有的汉字操作系统采用其它的汉字内码。

这样，计算机内汉字代码的处理首先是将键盘接收到的输入码转换为汉字内码，然后将汉字内码进行传输、存贮等处理转换，在输出时，根据汉字内码转换成汉字地址码，根据汉字地址码取出汉字字模供输出设备显示、打印。

3. 汉字的数字信息代码转换为汉字字形

将汉字的数字信息代码转换为汉字字形是汉字信息处理中的重要组成部分，是汉字输出的一个重要环节。一般汉字字形用点阵或矢量表示，对于汉字字形点阵而言，每个汉字都是由“0”、“1”按一定规则排列而成的信息代码，在输出时，“0”表示空白点、“1”表示汉字笔划上的点，这样将“0”、“1”组成的信息代码经一定转换，在屏幕或打印纸上，就可看到一个汉字的字形。对于用矢量表示的汉字字形而言，它的转换大体上分两步：第一步是将汉字字形的骨架和轮廓确定，第二步是对汉字字形的骨架和轮廓作必要的修饰，一般这两步在计算机内信息处理时完成。用户在屏幕或打印纸上所看到的是一个完整的汉字字形。用上述两种方式所表示的汉字，可有各种字体，如宋体、仿宋体、楷体、黑体等等。汉字字形的质量

和表示汉字字形的信息有关，也和外部输出设备有关。

三、汉字信息处理系统的结构

汉字信息处理系统由四部分组成：汉字输入、汉字信息加工处理、汉字库和汉字输出。这四个部分之间的关系如图 1-1 所示。

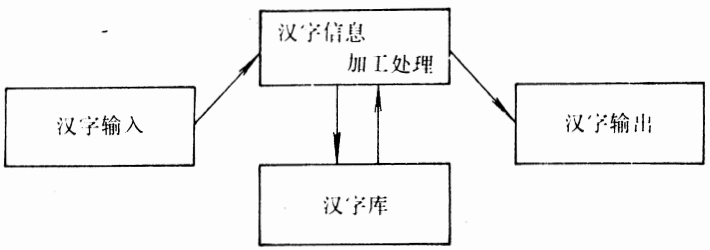


图 1-1　汉字信息处理系统结构

1. 汉字输入

把汉字输入计算机就是将汉字的信息代码输入计算机。目前汉字的输入方法有三种，它们是汉字键盘输入法、汉字语音识别输入法和汉字字形识别输入法。其中西文小键盘的汉字输入法是当前汉字输入法中用得最为广泛的一种，印刷体的汉字识别已经走入市场，开始流行。但是语音识别输入和手写体汉字的字形识别输入尚未完全实用化，所以汉字输入仍是今后相当一段时间内需要研究的课题。

2. 汉字信息加工处理

汉字信息加工与所用的机器有关，从系统硬件角度来看，我们应当设计出各种汉字信息处理系统以适合各种不同机器本身的特点，如设计出专用的、通用的、多用户的、局部网络型的或分布式的汉字信息处理系统，除了专用的汉字信息处理系统之外，主要是采用对现有计算机系统进行改造，使其保持原有西文系统的所有特

性，同时具有处理汉字信息的能力。从系统软件的角度来说，要考虑到充分利用现有的许多优秀的西文软件，对它们进行必要的“汉化”处理，使之也能处理汉字。总之，就是要保持中西文兼容，保持西文软件的良好性能，充分利用中西文信息处理各自的特点，提高计算机汉字信息加工处理的性能。

3. 汉字库

要使系统能够输出汉字，必须事先在系统中建立一个汉字字形库，简称汉字库。

由于汉字共有六万多个，每个汉字的使用频度差别很大，有的汉字使用频度很高，如“的”字，其使用频率为 4%。有的汉字为死字，极少被用到。为此我国制定了《信息交换用汉字编码字符集——基本集》(即 GB2312-80)，其中规定了 6763 个汉字。汉字字形质量与字形点阵有关。例如对于 108×108 的字形点阵，每个汉字需占据 1458 个字节的存贮容量，若需要不同字体、字号，则所占用的存贮容量，甚至可达到数千兆字节。而内存资源是十分宝贵的，外存资源也是有限的。因此，需要研究和确定汉字字库的设计方法，采用诸如字模压缩、多级存贮等技术处理。

4. 汉字输出

汉字的输出设备有多种，一般是指汉字的显示设备及打印设备。通常的西文计算机系统所配的显示器及打印机是不能直接显示和打印汉字的，这就需要对原有的输出软件进行改造或者研制出专用的汉字输出设备。另外，还需要对输出的字形进行无级放大、缩小和旋转等技术处理。

从汉字信息处理系统的结构分析可以看出，它的每一个组成部分都有待进一步的发展，对汉字信息处理的研究需要经过长期艰苦地努力。

四、汉字信息处理的发展史

我国在 1958 年研制成功第一台大型电子管计算机，差不多与

此同时，开展由俄语到汉语自动翻译的研究工作，开始了汉字信息处理的研究。经过三十多年的理论研究和实用开发，从各个方面推动了计算机在我国的应用和推广。

1. 系统理论研究的发展

50年代对俄汉机器翻译的研究，虽然受当时的计算机技术水平的限制，未能取得理想的结果，但是在我国的汉字信息处理研究史上写下了光辉的一页。60年代后期，随着计算机技术水平的提高，逐步对汉字信息处理技术进行探索和研究。在70年代中期开始系统地研究和开发这项技术。

70年代伊始，在国家有关部、委的领导与资助下，于1974年8月制订和组织开展了我国第一个大型的汉字信息处理工程项目的研究，即"748"工程。从此，开始了汉字印刷领域的一场革命。为了科学地解决汉字集问题，经过对2亿1千万左右个字的资料统计，形成了第一本《汉字频度表》。在1980年，经过对搜集到的《汉字频度表》和其它一系列字的分析统计，汉字集处理方面的第一个国家标准《信息交换用汉字编码字符集—基本集》(GB2312-80)正式颁布了，从此，汉字编码技术、汉字码本、汉字库、汉字点阵在字量方面均以此为准。

在1981年，中国中文信息学会宣告成立，在学会之下又分别设立了各专业委员会(下设专业学组)。如汉字编码专业委员会、基础理论专业委员会、系统专业委员会、设备专业委员会等。于是，基础理论的研究受到高度重视，一大批基础研究成果陆续在80年代中期以前推出。在汉语语料库、汉字属性两项重要语言工程项目上取得了重大进展，特别是在汉字输入技术、编码字符集、汉字内码等方面取得了长足的进步。

(1)汉字输入技术方面　汉字输入方法可分为汉字键盘人工输入、汉字的自动识别和汉语的语音识别。

①汉字键盘方面：在我国汉字键盘输入技术方面的发展，可以归纳为如下几个方面：首先从汉字本身形、音、义的特征出发的编

码方案发展到利用计算机软件支撑的汉字键盘输入技术研究的发展。其次，从人工设计编码方案发展到利用计算机辅助设计提高编码方案的质量。再次，从汉字编码发展到汉语词语编码，进而朝着自然语言处理的方向发展。最后一个方面是从单纯注重面向专职操作人员的编码方案设计，转向同时注重面向一般使用人员（非专职输入人员）的编码方案研究。总之，以字为基础、词为主导、智能化输入是键盘输入总的发展趋势。

②汉字识别方面：自 70 年代末期，汉字识别技术研究在我国开展以来，特别是从 1986 年后取得很大进展。

联机手写汉字识别已经商品化，识别字数为 6763～12000 左右，识别率初次使用时为 80%左右，经常使用可达到 95%以上，识别速度基本上能跟上人书写的速度。印刷体是我国汉字识别的主流，识别字数为一般 1～2 级汉字（3755～6763 个），识别速度已达到 9～14 个字/秒（用 286 微机）、20 个字/秒（用 386 微机），对样张识别率能达到 99.9%的高指标。脱机手写印刷体汉字的识别最近几年也进行了很多研究，并建立了几个实验性系统。

汉字识别的重点在提高性能，加强实用化研究，改善人机综合环境，版面分析和识别结果的后处理等方面 。

③语音识别方面：汉语语音识别主要沿着两条途经展开：孤立词的模式匹配识别和有限词汇的连续识别。1978 年中国科学院声学所成功研制了通用实时语音识别系统。1984 年清华大学成功研制出 3000 个孤立词的语音识别系统。语音识别系统的发展相当迅速，1988 年，中科院声学所研制的具有 2000 孤立词的实时语音识别系统，在西欧高技术展览上，获国际大奖。

(2)编码字符集方面的发展　随着基础理论研究工作的深入开展和汉字编码键盘输入技术的发展，在国务院电子振兴办公室、机电部、国家技术监督局和国家语委推动下，先后制定和颁布了一系列有关的国家标准：

GB1988-80　　《信息处理交换用的七位编码字符集》

GB2312-80	《信息处理交换用汉字编码集基本集》
GB5007.1-85	《信息交换用汉字 24×24 点阵字模集》
GB5007.2-85	《信息交换用汉字 24×24 点阵字模数据集》
GB5119.1-85	《信息交换用汉字 15×16 点阵字模集》
GB5119.2-85	《信息交换用汉字 15×16 点阵字模数据集》
GB6364.1-86	《信息交换用汉字 32×32 点阵字模集》
GB6364.2-86	《信息交换用汉字 32×32 点阵字模数据集》

除此之外，还有许多点阵规格的字模集，如 GB12034-89～GB12044－89，GBT13443-92～GBT13446-92。而且新的标准也在不断制定，这些标准的制定有助于汉字信息处理的标准化和统一化，加速汉字信息处理的发展。

(3)汉字内码方面的发展　在开展汉字内码与数据类型标准化的课题研究方面，国内组织了文字专家、计算机专家、标准化专家的跨部门协作。在深入研究文字、字符、代码、图形符的关系的基础上，提出了 HCC 方案，有效地参与、影响了 ISO 的活动，深刻反映了汉字信息处理的特点和要求。并得到 ISO 各成员和包括 IBM 公司在内的跨国公司的重视。

2. 汉字信息处理应用系统的发展

自 70 年代以来，我国在中文计算机情报检索、电子印刷排版系统、机器翻译系统、汉语理解和人机接口等应用系统方面的研究取得了重大成果。

(1)中文计算机情报检索　长期以来，我国的科技情报管理几乎完全靠人工实现，在 80 年代，中文计算机情报检索取得了长足的进步，并走向实用化的过程。

①中文数据库建设：数据库建设是开展计算机情报检索的基础和关键因素之一，我国已建立各种数据库 300 个以上，在不同层

次和范围内使用。

②中文情报检索的研制：中文情报检索的研制主要有两个方面：一是自行研制，另一是在国外检索软件基础上进行汉化和二次开发。特别是在汉语主题表的编制、汉语自动标引、汉语全文检索、汉语智能接口等课题都取得了不同程度的进展。

(2)电子印刷排版系统　中文电子印刷排版系统在中文信息处理应用领域中卓有成效。1985 年初，华光Ⅱ型系统投入生产性使用。1985 年夏天，我国首次使用微机完成排版出清样的工作。1986 年研制成我国第一套台式出版系统，同年，完成了国内第一个实用的高性能科技排版软件。1987 年夏天，又推出国内第一个微型机上实用批处理科技排版软件，同年秋天，国内第一个交互式书刊排版软件应运而生。1988 年春，专用芯片支持的华光Ⅳ型系统投放市场。而且性能更强的系统正在逐步走入市场。

(3)　机器翻译系统　机器翻译早在 50 年代就列入我国科研工作的发展规划，其后走了一条曲折道路，直到 70 年代末期，开始重建研究队伍，进入了我国机器翻译的再发展阶段。

目前，国内已有四个系统通过了部级鉴定。这四个系统是："英汉科技文献题录机器翻译系统"，"科译 1 号英汉机译系统"，"ISTIC－1 型英汉冶金题录机译系统"，"人名翻译和题录翻译系统"。国内已推出的商品化系统有三个："译星英汉机译系统"，"IECM 英汉机译系统"和"英汉机器翻译系统 Marcopolo"。

(4)汉语理解和人机接口　我国自然语言理解方面的研究工作起步较晚，进入 80 年代以后，自然语言理解开始得到重视。80 年代中，"自然语言理解和人机接口"列入重点研究课题，我国自然语言理解的研究和国外有较大的差距，主要表现在：研究力量薄弱、水平较低、重复多、深度和广度不够。

总的来说，近年来微型机汉字信息处理系统研究开发取得了可喜的进步，不论是在理论研究方面，还是在实用技术、实用系统方面，一大批适合我国国情的，具有中国特点的多层次性能强的汉

字信息系统应运而生，推动了我国计算机事业的蓬勃发展。

第二节　DOS 操作系统

一、DOS 总述

1. DOS **概述**

DOS 是目前广泛使用于 PC 系列微型计算机上的一个操作系统，它和建立在它之上的其它软件一起，使微型计算机系统充分发挥作用，完成用户指定的各项任务。

DOS 的功能是为了使用户有一个理想的工作环境，随着版本的不断升级，其功能逐步增加。它主要有如下四个功能：

①管理计算机系统的全部系统资源；

②驱动计算机系统的外部设备；

③建立有效的文件系统；

④为用户使用计算机系统提供有效的途经。

用户可以采用键入命令或选择命令的办法来使用 DOS 系统，DOS 使用方式主要有两种：一种是在 DOS SHELL 模式下使用，另一种是在命令提示符下键入命令。

DOS 有两种类型：一种是 Microsoft 公司的 MS－DOS，另一种是 IBM 公司的 PC－DOS。实际上，对于版本号相同的这两种 DOS 系统，对普通用户而言，它们几乎没有区别。在通常情况下，可以不再仔细区分它们，而统称为 DOS。

2. DOS **发展过程**

1980 年 10 月，IBM 公司为其正在设计的个人计算机系统向一些大的软件研制公司寻求配套的操作系统。当时，Microsoft 公司向 Seattle Computer Products 公司购买了 86－DOS 的专利权，并对其作了较大的改进，将此改进了的系统命名为 MS－DOS。IBM 公司于 1981 年 10 月推出 PC 系列机基本型——IBM－PC

机时，选定了 MS－DOS 为该机的基本操作系统，并改名为 PC－DOS1.0。于是，以后由 Microsoft 公司推出的 DOS 称为 MS－DOS，由 IBM 公司推出的 DOS 称为 PC－DOS。下面简单地谈谈各 DOS 版本及其主要性能。

DOS1.0 版本是磁盘操作系统的第一版本，此版本以单面软盘为基础，对磁盘文件只使用单级目录的管理办法，文件操作较为简单。

DOS1.1 版本，支持双面软盘并可实现错误定位。

DOS2.0 版本，支持带硬盘的 PC/XT 机，在结构上作了重大变动，引入了多级目录系统，增加了输入输出重定向的管道功能，有类似 UNIX 系统的许多特色。

DOS2.11 版本，改进了国际支持，并对错误精确定位。

DOS3.0 版本，支持以 80286 为 CPU 的 PC/AT 机，提供了 1.2M 软盘和大于 20M 的硬盘服务，可使用虚拟磁盘，进一步扩充了文件管理功能。

DOS3.1 版本，支持网络的功能，并扩展了错误检测功能。

DOS3.2 版本，支持 8.89cm(3.5 英寸)软盘，并且盘的格式化功能固化在盘的驱动器中。

DOS3.3 版本，支持更大容量的硬盘，配备了一些新的设备驱动程序。

DOS4.0 版本，具有了多任务功能。

DOS5.0 版本，支持更大、更多的软、硬盘，能贯穿多级目录检索文件，增强了数据的安全保护等功能。

DOS6.0 版本，支持反病毒和磁盘空间压缩，并对系统命令进行了扩充。

DOS 版本的不断升级，内部和外部命令的个数不断增加，且功能也进一步完善，高版本的 DOS 保持着向下的兼容性。目前国内大多数用户使用的是 DOS3.1、3.3 或 5.0 等版本。

3. DOS **系统结构**

DOS 系统结构，从层次上分，可分为三层：

①内层为设备驱动层，为基本输入/输出系统；

②中间层为文件系统层，为文件管理和系统调用；

③外层为命令解释层，为用户输入的 DOS 命令进行解释并执行。

设备驱动层，依赖于机器硬件，控制和管理外部设备，它由两部分组成：系统初始化程序 SYSINT 和标准字符及块设备驱动。它扩充了 ROM－BIOS 与文件系统层 DOS－Kernel 的接口，处理同全体外设的通讯。

文件系统层是 DOS 的核心，不依赖于机器硬件，它是真正的操作系统，它由如下两部组成：内核初始化和系统功能调用程序。它实现有关文件和外存管理方面的全部工作，如处理终 端、打印机、键盘的输入/输出。

命令解释层，它是操作系统和用户间的接口，负责接受用户的键盘命令，并通过文件系统层和设备驱动层，实现用户的命令控制和管理外部设备进行基本的输入/输出。它由如下三部分组成：①常驻部分 CCPR；②SHELL 初始化程序；③暂驻部分 CCPT。它含有 COPY，DEL，TYPE 等内部命令处理程序、控制批处理文件，进行批处理、加载程序等处理。

无论是 PC－DOS，还是 MS－DOS，每一层次上都有一个系统文件，它们的文件名略有不同 ，分别规定如下：

	MS－DOS	PC－DOS
内层系统文件	IO. SYS	IBMBIOS. COM
中间层系统文件	MSDOS. SYS	IBMDOS. COM
外层系统文件	COM-MAND. COM	COM-MAND. COM

二、DOS 文件

1. 文件名

每个文件都有一个文件名，它由两部分组成：第一部分称为本名，第二部分称为扩展名。本名和扩展名之间用一个句点分隔。扩展名可以省缺，如果省缺，则将它们之间的分隔符也省缺。

扩展名常用于说明文件的内容，以便按内容区分文件。DOS 和用户之间关于扩展名有如下约定：

COM	命令文件
EXE	可执行文件
BAT	批处理文件
SYS	系统专用文件
BAK	备份文件
OBJ	目标程序文件
LIB	库文件
LST	列表文件
BAS	BASIC 语言源程序文件
FOR	FORTRAN 语言源程序文件
PAS	PASCAL 语言源程序文件
ASM	汇编语言源程序文件
C	C 语言源程序文件
PRG	dBASE 命令文件
DBF	dBASE 库文件

2. 文件属性

DOS 中文件可具有下列属性：

①只读：表示文件只能够读，不能够写，也不能删除，但可被复制。

②隐藏：表示文件在普通列目录操作时，不显示，也不能删除

和复制。

③系统：表示该文件为系统专用。

④档案：表示该文件应该后备的，一般文件都具有这属性。

⑤卷标：记载文件卷的卷标，这种文件只存在于根目录中。

⑥子目录：具有该属性的文件为子目录文件，它记载子目录，这类文件的长度一般为零。

三、DOS 目录

1. 目录树

DOS 采用树型目录结构，由根目录和子目录组成。每个盘至少有一个目录，当格式化一个软盘或硬盘时，DOS 就会建立一个目录，可以把所有其它的文件和目录放在这个目录中，这个目录被称为根目录，一般磁盘上根目录的大小是固定的，与磁盘的容量有关。子目录是由用户根据需要建立的，DOS 不能自动建立子目录，子目录作为其父目录的一个文件，登记在其父目录中，每一个子目录的大小不是固定的，但实际上受到磁盘空间的限制。DOS 提供专门的子目录操作命令，用于建立和删除子目录。子目录的命名规则同文件的命名规则相同。

2. 路径

路径是指表示文件在目录树中的位置，它由目录名连接而成，目录名与目录名间必须用反斜线符号“\”。

路径有绝对路径和相对路径之分。所谓绝对路径是指从根目录开始的路径，路径的第一个字符为反斜线符号“\”。利用绝对路径可以定位到任意一个目录下的文件。相对路径是以当前目录为前提的，一般是以当前目录的下级子目录开始的路径，路径的开始部分为当前目录的下级子目录名。

DOS 所能识别的路径最多不能超过 66 个字符（包括驱动器字母和冒号）。

四、DOS 命令的类型

1. 内部命令

内部命令存贮在 COMMAND.COM 文件中，随 DOS 的启动而被装入内存。它是 COMMAND.COM 的一部分，因此在列目录表上看不到它们的名字。这些命令常驻内存，需要时可直接执行。如 DIR，COPY，DATE 等命令。

2. 外部命令

外部命令以一个独立的文件存放在磁盘上，在文件名与目录名的列表上能看到它们的名字，任何一个外部命令的文件名都具有 .COM、.EXE 或 .BAT 扩展名，这些命令执行时，先到磁盘上把对应命令的解释执行程序装入内存，然后再执行。一般，外部命令与 DOS 版本要配套，常见的外部命令如 backup，exe2bin，format 等。

3. 批处理命令

批处理命令是一些内部命令。用户能使用批处理命令引导批处理程序怎样运行。如 call，for，if，rem 等命令。

4. CONFIG.SYS

CONFIG.SYS(配置文件)命令是一些用来定制用户系统的命令，这些命令对于某些任务，例如对于安装设备驱动程序，设定对文件和缓冲区的限制以及在 CONFIG.SYS 处理过程期间执行 DOS 命令等来说是十分有用的。如 break，shell，buffers 等命令。

5. 网络

并非所有的命令被设计成能在网络上使用。如果某命令的类别标示出“网络”二字，则表明该命令能够用于网络。

第三节　DOS 汉化原理与结构

一、DOS 汉化原理

1. DOS 汉化的意义

计算机已逐步深入到我国的各个领域，然而由于目前几乎所有的系统软件和应用软件都是以西文状态出现，这对使用汉字的我国受到了极大的限制，为了使计算机技术在我国能得到广泛的应用，就必须要大力开发能够处理汉字的系统软件和应用软件，特别是系统软件的开发使用。如果没有一个好的系统软件，就不可能产生较好的应用软件，因此对西文 DOS 系统的汉化具有十分重要的意义。

2. DOS 汉化的基本要求

DOS 汉化的目的是使西文计算机系统能成为一个汉字信息处理系统，关键是在计算机对汉字的数据处理的基础上，使计算机使用汉字如同使用西文字符一样方便和容易。通常，汉化后的 DOS 应具有如下几项要求：

(1)应具有中西文兼容性　汉化后的 DOS 系统应该与原西文的 DOS 系统兼容，因为原西文的 DOS 系统已配有丰富的西文软件，并且已为广大的用户所接受。如果不能保持兼容，那么就导致人力、财力和时间的浪费。因此汉化后的 DOS 系统只有保证中西文兼容，才可以直接用他人开发出来的软件财富，吸取他人先进经验，加快我国计算机事业的发展，且有利于系统的推广应用，也有利于国际间的互相交流。

汉化后的 DOS 还应具有原西文 DOS 系统的全部功能和使用方法，具有原西文 DOS 系统所不具有的汉字处理功能，使用户感到在计算机系统内，中西文之间只不过是用不同的信息表示而已。

(2)把汉字看成是一种特殊的西文字符　从信息角度来看，汉

字信息处理系统与西文信息处理系统之间没有本质的区别。两者均为非数值处理，都要求计算机系统具有资源、信息共享、高级语言、数据库和数据通讯等功能的支持。另外，从计算机信息处理而言，表示汉字的字符代码，必须不同于普通的西文字符代码，否则会产生中西文的混淆。同时必须在西文系统中有意义，不然就很难做到中西文之间的兼容，所以可以把汉字看成是一种特殊的西文字符。

(3)具有汉字信息的加工处理能力　汉化后的 DOS 系统是一个汉字操作系统，它可直接对汉字信息进行加工处理，至少应包括键盘输入汉字处理程序，CRT 显示汉字处理程序，打印汉字处理程序。

其中键盘输入汉字处理程序就是能够接收给定的若干种汉字输入方案，并进行代码处理，使得每输入一个汉字的编码，都可转换成唯一的汉字内码。

CRT 显示汉字处理程序就是根据汉字内码，通过一系列的代码变换，获得相应的汉字字形，并在 CRT 上显示出来，且还要解决汉字字形点阵的检索。

打印汉字处理程序就是根据汉字内码，通过一系列的代码变换，获取相应的汉字字形，并在打印机上打印出来，且还要解决各种规格字型的产生，汉字点阵的旋转和字库点阵的检索等等。

此外，系统应具有良好的可靠性、实用性和方便性，包括汉字库的建立与维护、汉字输入码表的自动生成、中文编辑程序等等。应以实用程序的形式向用户提供方便的软件工具。

3. DOS **汉化的原理**

DOS 的汉化，从原理上并不十分复杂，它是把原西文 DOS 系统中的某些代码文件作适当的修改、调整，使之能够处理汉字信息。

根据原西文 DOS 系统的分析可知，汉化后的 DOS 应具有和处理西文字符类似的汉字输入/输出处理，即需要修改软件中断调

用 ROM－BIOS 中的有关驱动程序。主要有下列四个软件中断：

①INT16H，键盘输入驱动程序。

②INT10H，显示器驱动程序。

③INT5H，屏幕打印驱动程序。

④INT17H，打印机输出驱动程序。

因此，DOS 汉化的本质就是修改原西文 DOS 系统中有关输入/输出的软件中断。

二、DOS 汉化版本的基本结构

为了方便叙述，下面以 DOS 汉化为 CC－DOS 为例作一个结构上的描述。

1. CC－DOS 的内部结构

CC－DOS 是在 DOS 的基础上，对其中文件管理系统(IBMDOS. COM)和基本输入输出系统(BIOS)扩充了汉字处理功能，所以 CC－DOS 的层次结构，如图 1-2 所示。

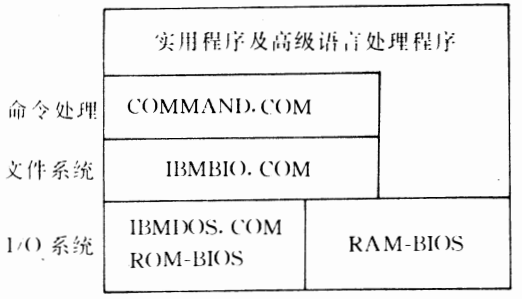


图 1-2 CC-DOS 的内部结构

CC－DOS 中的 COMMAND. COM 是对 DOS 的 COMMAND. COM 内的提示信息进行汉化而成的；CC－DOS 中的 IBMDOS. COM 也是对 DOS 的 IBMDOS. COM 内的过滤符进行修

改而成的；CC－DOS 中的 IBMBIO.COM 与 DOS 的 IBMBIO.-COM 是一样的，其中的 ROM－BIOS 存放在主机板上的 EPROM 中，控制着系统所需的主要外部设备的工作。用户软件频繁地调用 ROM－BIOS 的各个设备驱动模块，为了响应对汉字外部设备的调动要求，就必须在原来的 ROM－BIOS 之外，再添加用以进行汉字键盘输入、显示和打印等操作的驱动模块。这部分软件不存放在 ROM 中，而是由系统自举时调入 RAM 中，然后常驻内存，故称为 RAM－BIOS。所以，CC－DOS 的基本输入输出部分（BIOS）应包括 ROM－BIOS 和 RAM－BIOS 两部分，总称为 CC－BIOS。

2. 内码的选择

内码是计算机系统中用来表示西文或汉字信息的代码。在原西文系统中，每个西文字符的内码是采用一个字节表示的 ASCII 码（或 EBCDIC 码等）。且一般只使用前 7 位来表示 128 个字符，而最高位作为奇偶校验或不用。因此汉字系统中的内码选择必须考虑到以下几个因素。

①不能产生二义性，即西文内码和汉字内码应严格区分。

②和原西文系统的兼容性要高。

③信息的位数尽可能少，尽量减少信息的冗余度。

④便于汉字信息处理中的基本操作和运算。

⑤便于扩充字符集。

⑥应与 GB2312-80 汉字字符集有尽量简单的对应关系。

据此，为了保障汉字系统的中西文兼容，内码的选择是十分重要的，我们在第二章中将作较为详细的论述。

3. CC－DOS 的内存分配

内存资源在计算机系统中是非常有限的，因此十分宝贵，对内存空间的分配是否合理，关系到整个操作系统的性能优劣。CC－DOS 被引入内存和完成装配后，其内存空间的信息布局如图 1-3 所示。

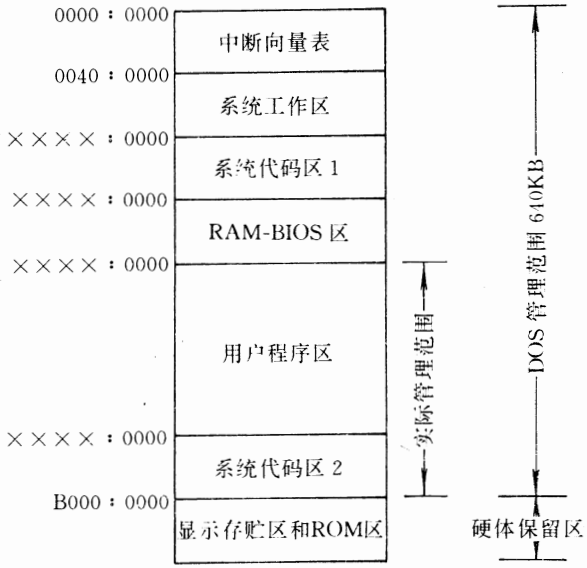


图 1-3 CC-DOS 的内存空间情况

(1)中断向量表 用于存放 256 个中断向量,每个中断向量占 4 个字节,这些中断向量中,用户只能使用一部分,另外一部分为系统占用。

(2)系统工作区 用作 CC－DOS 内核运行时的工作单元和它与 BIOS 间的通讯单元。另外,还用作 BIOS 和 BASIC 解释程序的工作单元和数据区。

(3)系统代码区 1 用作 CC－DOS 的核心部分,包括 IBMDOS.COM 和 IBMBIO.COM 的代码和 COMMAND.COM 中的常驻内存部分。这个区域是系统的重要部分,该空间是不允许用户使用的。这部分内存空间的大小是不固定的,它随系统的配置的变动而变化。

(4)RAM－BIOS 区 这部分内存空间为 CC－BIOS 中的 RAM－BIOS 占用。这包括它的代码区、工作区、汉字库和词库等

内容。这些内容在系统自举时被引入内存，然后利用系统提供的“驻留内存，退出运行”功能而常驻在这个空间区域内。

(5)用户程序区　为用户所用，系统的外部命令和实用程序也使用这个空间，它的大小与系统的内存配置有关。

(6)系统代码区 2　用于存放 COMMAND.COM 的过渡部分，在必要时可被其它内容覆盖掉，然后再从系统盘上的副本中重新恢复这部分内容。

(7)显示存贮区和 ROM 区　为显示适配器和 ROM 使用。

三、DOS 汉化的步骤与方法

对于国际上许多优秀软件，进行汉化是十分重要的。汉化的过程，是一个分析问题、发现问题和解决问题的过程。要求操作人员具有一定的计算机知识和外语水平，具有熟练的反汇编跟踪技能，从对指定软件层次的分析理解、跟踪研究、修改调试，到最后的产品鉴定，是一项非常复杂的过程，稍有差错，就会产生意想不到的后果。所以，大致有如下几个步骤。

第一，分析问题，即对将要汉化的软件的基本结构要了解和掌握。

要对软件进行汉化，先要了解软件的代码、数据结构的内部形式，弄清其数据传送流程和各个功能模块之间的内在联系。否则，就难于着手处理，实现软件的汉化。

对西文软件的分析过程可分为三步完成：

①理解软件的系统及环境，获得系统的具体模型；

②从当前系统的具体模型抽象出其逻辑模型，并建立目标系统的逻辑模型；

③为目标系统的逻辑模型的修改补充作准备。这部分工作最为重要，需要深入细致的研究分析，会直接影响下一步工作的进行。

分析的方法大致有两种：一种是利用反汇编手段把其源程序

清单打印出来，通过仔细阅读，绘出程序流程图，弄清楚在程序中数据传送的来龙去脉、地址定位和字符处理的方式等，以便找到与汉字信息处理有关的功能模块。另一种利用动态调试工具，把需要汉化的程序调入内存，对其进行反复的跟踪、搜索、弄清程序中各个功能模块之间的内在联系，以便找到与汉字处理有关的功能模块。

第二，发现问题，即找出与汉字处理有关的程序代码。

在了解和掌握汉化软件的基本结构之后，重点分析系统的输入输出流，找出限制汉字内码流向的原因和解决办法，这部分工作非常重要，直接影响到软件汉化的成功与否。

找出与汉字处理有关的程序代码并非易事，通常采用既简单有效、又很实用的工具软件 Debug，通过反复的跟踪、查找，才能找出，这是一项技术性很高的工作，带有一定的技巧，在此不再作详细论述。

第三，解决问题，即修改或调整原程序代码，使之能够处理汉字。

通过上述的分析研究，确定汉化方案，对原西文软件进行修改、调试和测试运行。在修改的过程中应尽量使中西文兼容，以及汉化后的软件保持原来西文软件的各个功能。

第四，整理和保存汉化过程中的文档材料，申请有关部门的技术鉴定，形成最终的汉化产品。

上述几个步骤对于软件的汉化是非常重要的，一般情况下，汉化所采用方法和步骤与各个人的知识程度和汉化经验有关，不一定完全相同，但其本质是一样的。

第二章　汉字代码体系

第一节　汉　字　输　入　码

一、概述

汉字“输入码”也称之为“外部码”，它是指为了把汉字输入计算机而编制的信息代码。当前，将汉字输入计算机的方法有以下三种：

①键盘输入汉字法。它又可分大键盘输入法、中键盘输入法和小键盘输入法三种，其实质是用键盘等作为设备装置，通过按键操作，产生一串（包括一个）表示汉字特征的代码信息，以此信息作为汉字输入的信息代码。

②语音识别输入法 。它是用声感器等设备装置，通过信号转换，产生一串表示汉字特性的声音信息，以此信息作为汉字输入的信息代码。

③文字识别输入法。它是由感应板或光笔或电视摄象镜头等，通过信号转换，产生一串表示汉字特性的图形信息，以此信息作为汉字输入的信息代码。

目前，汉字输入主要是通过西文小键盘完成的，因而汉字输入码设计的好坏，会直接影响到汉字输入的速度以及系统的性能，它在汉字信息处理系统中占有极其重要的地位和作用。然而，由于研究的角度和使用的场合不同，对于汉字输入码的评价也不尽相同，但是总的要求是：规则简单、易于记忆、操作方便、重码率低和码长

短。显然，这些要求之间有的是互相制约、互相矛盾的。

二、汉字输入码类

关于汉字输入码，国内外许多编码设计者们通过了辛勤的劳动，研制出数百种的编码方案，可归纳为六种类型：即流水码类、字形码类、字音码类、形音码类、音形码类和词汇码类。

1. 流水码类

这类编码方案是将所有被编码的汉字按某种顺序排列，并依次给以编号，其编号即为该汉字的输入码。例如国标码是这类编码的典型，它是按照国家标准交换字符集区位号编码。流水码的特点是整齐、容易实现、没有重码、码长适中。主要缺点是，由于码元的取值大多和汉字的音、形、义等特征之间没有直接的映射关系，且难于记忆，故这类编码一般用户难以掌握使用，可作为辅助汉字输入方式。

2. 音码类

这类编码方案是根据汉字的读音特征，用汉字的拼音作为该汉字的输入码。这类编码方案的优点是编码规则与音有关，只要有一定的汉语拼音基础，很容易掌握使用。但其主要缺点是重码率高，码长长短不一，对生僻字无法处理，此外，由于受到各地方方言和口语的限制，使得音码类的普及使用受到一定的影响。音码类最典型的例子是全拼输入法，以及在此基础上改进而得到的简拼、双拼等汉字编码方案。

3. 形码类

这类编码方案是根据汉字的字形的特征，如汉字的笔画、汉字的字元（组成汉字的基本构件）等基本字形特征来进行汉字编码。形码类的特点是形象、直观、编码的字形特征明显，用户能见字定码，且重码率较低。属于这类编码的方案很多，如“札”字输入法，它是将字形结构化为最小组成单元——笔画来表示，然后再按书写顺序依次取笔画进行编码，这种输入方法的编码规则简单、易于记

忆，但是编码的码长不一，效率较低。再如“郑码”，它是以字根为核心，以字元为主体的形码。它有二十六个主字根，一百多个副字根，按横、竖、撇、捺、弯、拐六个笔画确定次序。主字根按序映射键盘上的二十六个英文字母键。副字根按各自的首笔和次笔构形特征装入国标 6763 个简体汉字及第一辅助集的全部对应繁体汉字，每个汉字的代码不超过 4 个英文字符，它具有形码的一切优点，但是因汉字的组字规律并不严谨和单一，编码规则也较为复杂，故用户不易于一下子接受。

4. 形音码类

这类编码方案目的是为了吸取形码类和音码类的长处。形码的特点是形象、直观、易为用户使用等；音码有易于记忆，实施简单等特点，如果以取拼形（笔画成字元）为主，拼音（字符）为辅的方法编码，这样的编码就称之为形音码。例如，王晓武等的“傻瓜码”（后称简易码），它的编码规则是第一码取汉字拼音首字符（A～Z 除 I，U，V 之外，不认识的字用？代替），第二、三码分别取起、次笔划名称的拼音字符，第四码取汉字末笔划名称 拼音首字符。而且还将笔划归类为八种：横（H）、竖（S）、撇（P）、点（D）、捺（N）、提（T）、折（Z）和勾（G）。显然，这种编码方案编码规则简单，方便记忆、重码率少，码长适中，具备形码和音码的优点。

5. 音形码类

音形码也是一种目的为了吸取形码和音码两者优点的编码方案，它和形 音码类似，不过，它是采用以拼音为主，以取形为辅的原则。例如，前面谈到过的拼音码，它的主要缺点是重码太多，如果添加一个取汉字末笔划名称的数字代码，则显然可以降低重码率，若笔划归类为以下八种：横（0）、竖（1）、撇（2）、点（3）、捺（4）、提（5）、折（6）和勾（7）。很明显，这样的编码方案具有比音码更多的优点。

6. 词汇码类

以词为主、以字为基础是提高汉字输入速度的有效方法。词汇

码就是以二字词为主体，同时适当收入三字以上词，组成一个拥有数万词条的词库，可采用声韵双拼、一母一键的输入方法，对音节易混、卷舌与干舌难分等问题，可作一些技术性处理。用它处理成篇成段的文章，基本上可以实现以词为单位的输入，提高输入效率2～3倍。同时还可对GB2312-80的一、二级汉字分别编码，做到字词混用、不用切换。为了方便用户，词汇码可设有以词取字的功能。用户输入单字汉字，可先输入一个包含该字的词，然后再从中选出所需单字。这种方法特别适用于输入那些不好拆分的字。此外，由于词汇码很难收全，所以可设置动态管理功能，用户可在输入过程中不必退出系统，随时进行插入、修改、删除的操作，非常方便。

总之，汉字的编码方法很多，它们之间主要的差别在于码元、码长以及抽取汉字特征的信息不同，形成了不同的汉字输入码，它们之间各有自身的优缺点。一般来说，一个汉字系统中往往配有多个，多类的汉字输入方式，用户可以根据自身的特点，选取适合自己的输入方式来输入汉字，以不断提高自己的熟练程度，且在实践过程中可以对原有的汉字输入码作必要的改进。

第二节 汉 字 内 码

汉字内码亦称汉字内部码，它是计算机系统内部处理和存贮汉字而使用的代码。它决定计算机汉字信息处理系统的性能，是一种重要的代码，在此对它作比较详细的讨论。

一、西文字符的内码

在目前的计算机系统中，一般称8位二进制数为一个字节，计算机在处理西文字符过程中，不是直接使用西文字符的字形结构表示，而是将所有可能出现的不同西文字符字形排列成一张表，给每个字符一个序号，然后用序号代表这个西文字符的字形。这个序号称为西文字符的内码，通常采用ASCII码或EBCDIC码等作为

西文字符的内码。它只用一个字节来表示，而且该字节的最高位不用或作为奇偶校验位。

二、汉字内码的设计目标

汉字字形在计算机中以图形方式表示，而汉字的图形一般通过数字量化的过程。例如，16×16 点阵是最简单的汉字字形，每个汉字必须用 32 个字节才能完整表示。由于汉字的数目繁多，字体多样。因此，只能模仿西文字符的处理方式，对每个汉字确定一个内码(或称序号)。汉字内码的设计质量至关重要，它直接影响系统的中西文兼容性、系统资源的开销、运行的可靠性和处理汉字信息的能力等系统性能指标。于是，十分有必要提出汉字内码的设计目标和标准：

①码长尽可能短且要表达尽可能多的汉字；

②避免与西文字符内码的混淆，不能产生汉字内码的二义性；

③能支持系统和应用程序作中西文高度兼容的信息处理；

④易于建立汉字序值关系，与 GB2312 中的国标码有简单的映射关系。

由于上述四个目标之间不是相容的，因此要求它们同时达到最佳状态是不可能的，在设计汉字内码时，应兼顾多个方面，尽可能地使内码质量提高。

三、汉字内码方案

目前，汉字内码的国家标准尚未制订出来。国内外的汉字信息处理系统使用了多种汉字内码方案，相互之间有着明显的差别。从使用的字节数来看，有二字节方案、三字节方案和四字节方案等。下面分别对这三种方案作一些讨论。

1. 二字节方案

这种方案用两个字节表示一个汉字。根据这两个字节最高位的取值情况，又可分为四种格式："00"格式、"01"格式、"10"格式和

“11”格式。

“00”格式是指两个字节的最高位均为“0”，这种格式与GB2312完全对应，但也与ASCII码完全相同，因此系统无法区分汉字和西文，故不能直接使用。

“01”格式是第一个字节的最高位为“0”，第二个字节的最高位为“1”。系统根据最高位为“1”的字节与前一个字节逆向搭配成汉字内码，这样把汉字和西文区分开来。使用这种格式时，对汉字代码不易产生二义性，但对西文代码易产生二义性，因此，这种格式不很理想。

“10”格式是指第一个字节的最高位为“1”，第二个字节的最高位为“0”。系统可以根据最高位为“1”的字节与后一字节（最高位为“0”）搭配成汉字内码，从而把汉字与西文区分开来。显然这种格式亦能较好地避免汉字代码的二义性，但有时对西文代码也会产生二义性。

“11”格式是指两个字节的最高位均为“1”。系统只要根据最高位就很容易把汉字代码与西文代码区分开来，不会产生中西文混淆。这种内码称之为变形国标码。虽然它能基本符合内码的设计目标，但是有时它也会产生汉字代码的二义性。CC－DOS就采用这种格式作为它的汉字内码。

采用二字节方案优点是它的码长最短，冗余量最小，表达简单，能与西文字符区分开来，并能直接进行一些基本的字符串运算和一般性的输入输出。另外，由于汉字在屏幕上显示的最合适大小为两个字符的显示位置，故采用这种方案能够非常方便和直接地完成中西文兼容屏幕编辑功能。它的缺点是：它需要占用字节的最高位，在对最高位作特殊定义或屏蔽的西文系统和高层软件中，采用这种方案则会发生混乱现象。此外，这种标识法所能容纳的汉字限于8836（94×94）个，若在国标汉字基本集外还需增加一个或两个汉字辅助集，则需重新设置汉字内码，导致相互之间不能兼容。

2. **三字节方案**

这种方案用三个字节表示一个汉字。它可以分为两大类：一类是带标识符的三字节内码；另一类是不带标识符的独立三字节内码。

带标识符的三字节内码采用〈标识符〉〈ASCII 符〉〈ASCII 符〉结构，其中首字节为汉字标识符，一般采用不用的(或几乎不用的)ASCII 符来担任，后面两个字节为 ASCII 符(不包括用作标识符的 ASCII 符)。系统根据扫描到的标识符，就把紧随其后的两个字节作为汉字代码解释，从而把汉字与西文区分开来。中华学习机 CEC－I 系统中就采用此汉字内码方案，它把 7FH 作为汉字标识符。显然，用这种方法能容纳更多的汉字，但是其局限性也是很明显的，如果系统中用到已被作为标识符的 ASCII 符，则会产生令人难于想象的后果。

独立的三字节内码常采用〈字母〉〈数字〉〈字母〉结构，以避免与西文字符串发生冲突。系统扫描到编码规定的特殊三字节序列时，就作为汉字代码解释，以此来把汉字与西文字符区分开来。例如，BL 系列汉卡就采用这种内码方案。显然用三字节表示汉字代码，其编码空间会有所增加，但是在中西文兼容的系统中，很难保证系统中不会出现〈字母〉〈数字〉〈字母〉的西文字符串，就会产生二义现象，故这种内码有一定的局限性。

三字节方案的最大优点是，其编码均由 ASCII 符组成，故它们能够在西文系统和高层软件中存在和处理，大大减少了系统"汉化"的工作量，且编码空间远远大于二字节内码的编码空间。它的缺点是，由于用三个字节表一个汉字，空间冗余量大，增加系统的存贮开销。另外，由于用 ASCII 符的三字节表示汉字，故汉字信息与西文信息发生混淆的可能性较大。此外，用三个字节表示一个汉字会使中西文兼容屏幕编辑发生困难。如果在屏幕上让汉字占用三个西文字符的显示位置，则会显得不协调，影响人机界面的友好性，故这种方案不很理想。

3. 四字节方案

这种方案用四个字节表示一个汉字，它和三字节方案类似，可分为两大类：一类是带标识符的四字节内码；另一类是不带标识符的独立的四字节内码。

带标识符的四字节内码采用〈标识符〉〈字母〉〈数字〉〈字母数字〉结构，或采用〈标识符〉〈标识符〉〈ASCII 符〉〈ASCII 符〉结构。其目的是为了把汉字与西文区分开来，尽可能地减少汉字信息与西文信息发生混淆现象，其效果定肯定比相应的三字节方案好，但是仍有可能产生中西信息混淆，且冗余较大。

独立的四字节内码常采用〈字母〉〈数字〉〈字母〉〈字母数字〉结构，以避免与西文字符串发生冲突，它用四个字节表示汉字编码，可容纳全部的六万多汉字。系统根据扫描到编码规定的特殊四字节序列，将它们作为汉字代码解释，借此把汉字与西文区分开来。这样作的目的也是为了尽可能地减少汉字信息与西文信息的混淆。例如，需要较大内码编码空间的中国古文信息处理系统 CAW—DOS，其内码采用 NICC 码，就是属于这种独立的四字节内码。

四字节方案的主要优点是，其编码均由 ASCII 符组成，故它们能够在西文系统和高层软件中存在和处理，同样可以大大减少系统汉化的工作量，且具有较大的编码空间，提高了系统支持的汉字量，与三字节方案相比，大大减少了汉字信息与西文信息混淆的可能性。它的缺点是，由于用四个字节表示一个汉字，从而增加了系统的信息存贮开销，为二字节方案的两倍，由于用 ASCII 符的四字节序列表示汉字，故仍存在着汉字信息与西文信息发生混淆的潜在因素。另外，用四个字节表示一个汉字会给中西文兼容的屏幕编辑带来一些问题，如果在屏幕上让汉字占四个西文字符的显示位置，则会显得很不协调，故必须在显示器驱动程序中设法进行调整，使一个汉字对应于两个西文字符的位置。

以上介绍的是以每个汉字为单位的汉字内码方案，这些方案各自有自己的优缺点。因此迫切需要有一个规范或标准的汉字内

码。此外还有一种汉字内码方案值得一提。

用首尾两个标识字节括住一串汉字的方法。因为在中西文交替出现时，绝大多数情况下，它们都是成串出现，在汉字串首尾用专用的控制符或系统内不使用的图形字符作为标识字节，以此来区分中西文。这种方案的优点是节省信息量，冗余小，其编码均由ASCII 符组成，故能够在西文系统和高层软件中存在和处理。同样可以大大减少系统汉化的工作量，且具有较大的编码空间，提高了系统支持的汉字量，若用多个字节标识，则可以减少汉字信息与西文信息混淆的可能，它的缺点是有可能导致标识符的丢失，或汉字代码的二义性。

除了上面介绍的几种方案之外，还有其它种类的方案，它们用得较少，在此不一一列举。为了使计算机推广应用，对汉字内码标准化的统一是十分必要的，其意义也是十分重要的。

第三节　汉字的其它代码

汉字系统中要用到多种汉字代码，这些代码除了前面两节中已介绍的汉字输入码和汉字内码之外，还包括汉字交换码、汉字地址码、汉字字形码和汉字控制功能码。本节将分别介绍这些汉字代码。

一、汉字交换码

1. 总述

汉字交换码是汉字系统之间或与通信系统之间进行信息传输时，对每个汉字所规定的统一编码。使用交换码可以达到系统设备之间或记录媒体之间的信息交换的目的。显然，汉字交换码位于一台机器和另一台机器（包括输出设备和记录设备）之间。汉字交换码本身的作用就意味着它必须具有统一的标准格式。GB1988（ISO646）为西文的交换码作了规定，这是一种七位代码。由于汉

字的数量远远超过西文字符的数量，所以不可能用七位编码来标识汉字。我国以 GB1988 为基础，根据 GB2311 规定的扩充方法，制定了汉字交换码的统一标准—GB2312，这是一种双七位编码，它为基本集中的 6763 个汉字确定了交换码。以后又制定了 GB7589 和 GB7590，分别为辅助二集和辅助四集中的 16000 多个汉字确定了交换码。

日本和我国台湾省也制定了汉字交换码。日本的《工业标准信息交换用汉字编码字符集》为 6353 个汉字确定了交换码。台湾省推出了汉字交换码的两个版本。1981 年推出的 CCCII 编码为 21994 个汉字确定了交换码，1983 年推出的《通用汉字标准交换码》为 13053 个汉字确定了交换码。

2. GB2312 **代码**

GB2312 的名称为《信息交换用汉字编码字符集—基本集》，它是根据 GB2311 的代码扩充方法制定的汉字交换码国家标准。GB2312 能与 GB1988 兼容，故可以使通用计算机系统方便地扩充汉字处理功能和进行汉字信息的交换。

GB2312 规定用两个字节来标识一个汉字或非汉字图形符号，但是每个字节仍为七位二进制码，其取值范围为 GB1988 的图形字符集的 94 个七位二进制代码。显然，GB2312 可以标识 94×94＝8836 个汉字和非汉字图形符号。目前，GB2312 共收了进行一般汉字信息处理交换用的 6763 个汉字和 682 个非汉字图形符号，留下的位置可供今后进一步扩充或标准化用。图 2-1 给出了 GB2312 代码构成。

GB2312 的代码通常称为国标码，代码的第一字节可以表示 94 行，每一行又称为一个区；代码的第二字节可以表示 94 列，每一列又称为一个位，用它区分出每个区内的 94 个位置。所以，第一字节又称为区号，第二字节又称为位号。一旦确定了区号和位号，就确定了唯一的一个汉字或符号。国标码是用十六进制数表示的。如果用十进制数来表示区号和位号，我们称之为区位码。由此可

见，国标码与区位码无本质的区别，仅是表示的数制有所不同而已。

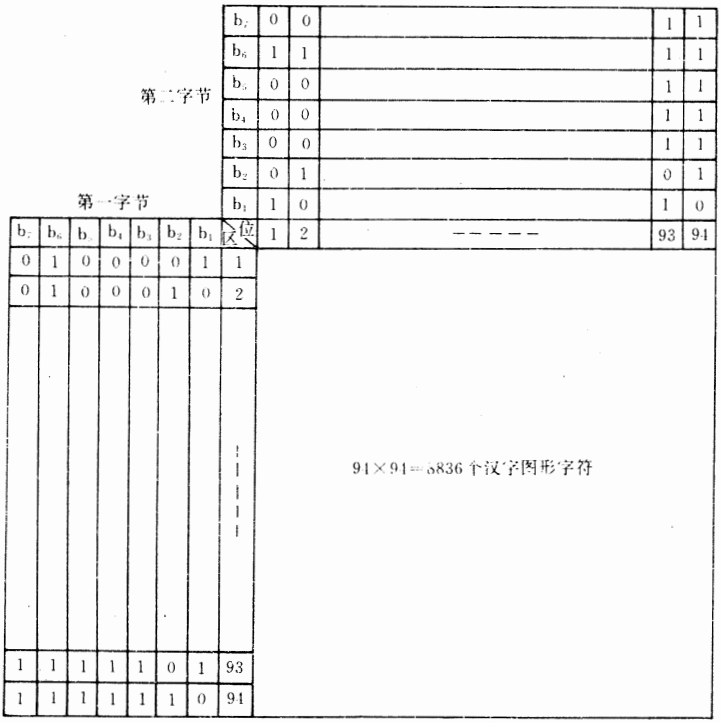


图 2-1 GB2312 代码构成

GB2312 共有 94 个区，每区有 94 个位。其中，1～9 区为非汉字图形符号，共收 682 个，第 2、4、5、6、7、8、9 区中还有 164 个空位。这些非汉字图形符号中包括以下几个部分：一般符号（间隔、标点、运算和制表符）202 个，序号 60 个，数字（0～9 和 Ⅰ～ Ⅻ）22 个，希腊字母（大写和小写）48 个，日文假名（平假名和片假名）169 个，汉语拼音符号 26 个，拉丁字母（大写和小写）52 个，俄文字母（大写和小写）66 个，汉语注音字母 37 个。

GB2312 的 10～15 区全为空位，88～94 区也全为空位。

GB2312 的 16～87 区为汉字，其中 16～55 区为一级汉字 3755 个，55 区中有 5 个空位；56～87 区为二级汉字 3008 个。一级汉字主要依据音序法来排列，即按汉字的汉语拼音字母的顺序排列。对于同音字，兼用了形序法的笔形顺序。在同音字组内，以起笔笔形为横、竖、撇、点、折的顺序排列。若起笔相同，则再按第二笔，以此类推。二级汉字依据形序法的部首顺序排列，共用部首 186 个。部首次序按部首的笔画数排列，若笔画数相同，则再按部首起笔的笔形顺序排列，起笔相同，再按第二笔，依此类推。

GB2312 中的 6763 个汉字的选择基础是中华人民共和国文化部与中国文字改革委员会于 1965 年联合发布的《印刷通用汉字字形表》，其中共有 6196 个汉字。另外又根据汉字信息处理的实际需要增加了 500 多个科技、地名和姓名用字。GB2312 中的这 6000 多个汉字已能满足一般的使用要求，因此，通常的汉字系统只要能支持这 6763 个汉字，就能成为一个实用系统。

二、汉字地址码

是用来指出汉字字形信息在汉字库中存放的逻辑地址的编码。当要向输出设备输出汉字时，必须通过汉字地址码才能从汉字库中取到所需之汉字字形信息，然后在输出设备上获得汉字字形的输出。地址码的形式随汉字库驻留的存贮介质不同而不同。如果汉字库驻留内存，则汉字地址码为内存地址值；如果汉字库驻留磁盘，则汉字地址码为磁盘空间地址值。汉字地址码的设计要考虑与汉字内码有一个简单的对应关系，以便输出字时，可以根据汉字内码简捷地从汉字库获得其字形信息。通常，汉字库内的汉字字形信息是按一定顺序连续存放的，故汉字地址码是连续有序的。

三、汉字字形码

汉字字形码是表示汉字字形信息的编码。目前汉字系统中使

用的汉字字形大多是数字式的，即以点阵的方式形成汉字。所以汉字字形码就是确定一个汉字字形点阵的代码，是汉字字形点阵的数字表示形式。

点阵表示法是表示汉字字形的最基本方法。对于一个16×16的点阵，是把一个方块横向和纵向均分成16格，以致形成256个小方格，即该矩阵有256个“点”。其中的每个点可以有黑、白两种颜色。用这样的点阵覆盖到汉字上，凡与笔划重叠之点，使之成为黑色；不与笔划重叠之点，则使之成为白色。这样形成的点阵就能描出该汉字的字形。我们可以用二进制数来表示这种点阵。用二进制数的“1”和“0”分别表示黑色点和白色点，一个点阵就可以用一串二进制数来表示。这种方法称为点阵的数字化。

根据以上所述可知，全点阵的字形中的每一点用一个二进制位来表示，随着字形点阵的不同，它们所需要的二进制位数也不同，即所需的字节数也不同。例如，16×16的字形点阵，每字共需32个字节，24×24的字形点阵，每字共需72个字节。与每个汉字对应的这一串字节就是汉字的字形码，它存放在汉字库中。当要求输出字形质量较高时，汉字点阵的点数会随之增大。例如，供印刷用的128×128字形点阵所对应的字形码要占2048个字节。由此可见，这样的汉字库的容量是十分巨大的，故有必要对这样的全点阵字形码进行压缩，经压缩后的字形码称为压缩型字形码。另外，汉字字形码的设计不仅是一个计算机技术问题，同时又涉及到文字学和书法艺术，这是一个技术和艺术相结合的问题，故具有相当的难度。现在，16×16、24×24、32×32等汉字字形点阵已经有了统一的国家标准。

四、汉字控制功能码

以上介绍的各种汉字代码都与汉字本身有联系，而汉字控制功能码并不表示汉字本身，它只是对汉字的处理过程产生影响。汉字控制功能码和汉字数据代码组成汉字数据流。并贯穿于汉字输

入、机内处理和汉字输出等系统处理流程中，它对汉字数据的格式处理、传送控制和解释执行起控制作用。

汉字控制功能码的设计包括两个方面：控制功能的编码表示和控制功能含义的确定。这两点在实际应用中往往是互相制约的，对于通用型汉字系统更为突出。这是因为其汉字控制功能码的表示必须按照有关的标准进行设计，自行选择的余地很小。在GB1988中已经规定了32个控制功能符，但是它们主要用于非汉字字符的处理，不能满足汉字处理的需要，故有必要在基本控制功能基础上加以扩充。为汉字处理需要所增加的控制功能尚未有统一的标准，因此处理汉字用的系统和设备都自行设计和定义了所需的功能。汉字控制功能码既与处理用设备和系统密切相关，也与汉字和汉语密切相关，因此，应当逐步形成统一的规范。目前汉字控制功能码的标准化工作正在进行之中。

第四节　通用字符编码国际新标准

一、制订新标准的必要性

1. ASCII 和 ISO646

多年来，计算机普遍采用美国信息交换用标准代码(American Standard Code for Information Interchange)来表示字符，通常把这种代码简称为ASCII码。这是一种七位代码，故其可以表示 $2^7=128$ 种字符，这些字符可以是字母、数字、标点符号和控制符。用这种编码来表示英文内的字符是不成问题的。

后来，国际标准化组织(ISO)通过了一个与ASCII几乎完全一样的代码集，称为ISO646，实际上这是承认ASCII作为国际标准。以后，国际标准化组织对ISO646作了扩充，把原来的七位代码扩充成八位代码，使其可以表示 $2^8=256$ 种字符。这种代码不仅可以表示英文中的字符，还可以表示包括全欧洲文字的所有字符，

比如法文、德文和西班牙文的字母。显然，ISO646 采用一个字节表示一个字符。

ISO646 中定义了一些控制符，这些控制符用于控制外部设备的工作，如回车、换页等。这些控制符的编码范围为 00H～1FH、80H～9FH，我们把这两个区域分别称为 C0 区和 C1 区。

2. 扩充方法 ISO2022

ISO646 仅能表示 256 种字符，若要表示更多的字符，则必须对八位编码进行扩充。为此，国际标准化组织制订了 ISO2022，它规定了八位编码字符集的扩充方法。按照 ISO2022 可以较大幅度地扩充字符编码，并使扩充的结果与 ISO646 兼容。

尽管 ISO2022 提供了一整套代码扩充的机制，但是它未能在计算机工业界获得广泛的应用。其原因主要是它的控制和切换机制太复杂，不易为人们理解。

3. 存在的问题

东方国家的文字与西文国家的文字有明显的差别，西方国家采用拼音文字，其字符集较小，东方国家则采用象形文字，其字符集较大，可达数千个，甚至数万个文字。其中最典型的是中文、日文和韩文，它们均有一个巨大的字符集。若要用 ISO646 来表示这些文字，那显然是不可能的。

为了要使计算机处理大字符集文字，一些东方国家和地区，在 ISO646 和 ISO2022 的基础上，制订了各自的字符编码标准，例如我国的 GB2312、台湾地区的 CNS11643、日本的 JIS0208 和韩国的 KSC5601 等。还有一些计算机厂商还制订了自己的字符编码标准，比如 IBM 公司确定了一个用于图形文字的双八位扩充 ASCII 编码集。

如此众多的字符编码标准造成了代码的混乱，影响了各种系统间的兼容性，妨碍了系统间和国家间的信息交换。因此，必须制订一种新的字符编码国际标准，它必须能把全世界各种文字使用的字符统一到一个字符编码集内。这项工作必须尽快开展。

二、ISO10646 的拟定

1. 二字节方案

国际标准化组织于 1983 年开始着手制订一种新的字符编码国际标准，并取名为 ISO10646。ISO10646 最早的方案是一种二字节字符编码集，其编码个数为 $2^{16}=65536$。美国的 ANSI(American National Standards Institute)竭力推崇二字节编码标准，它希望 ISO10646 能与 ASCII 兼容，即开头的 128 个字符应是 ASCII 符，其后才是其它字符集。

为了要保持新标准对 ISO646 的兼容(当然也对 ASCII)兼容)，则必须保留 ISO646 中的控制符代码，即 C0 区和 C1 区内的代码不能用于定义非控制符。据统计，含有 C0 区和 C1 区内代码的编码，约占二字节编码总数(65535)的 40%，所以用于定义非控制字符的编码数不足 4 万个。

为了使二字节编码能够覆盖所有文种的字符，故应对字符集作精心选择。中文、日文和韩文是世界上拥有字符数最多的文种，在这三种文字中均使用了大量的汉字，其中有许多汉字在这三个文种内是一样的。例如，“水”字在中文、日文和韩文内具有相同的字形和意义。因此，ISO10646 最早的方案是建立一个统一的汉字集，中文、日文和韩文都使用这个汉字集内的汉字，即每个汉字在 ISO10646 中只有唯一的一个编码。这样可以节省数千个汉字的编码，就有可能用二字节编码来表示所有国家的字符。

鉴于上述情况，我国建议将中文、日文和韩文中的汉字结合在一起，构成一个大约有 25000 个汉字的大汉字字符集。我国于 1989 年 7 月向国际标准化组织提交了 HCC(Han Character Collection)的提案，并于同年 10 月在 ISO－IEC JTC1/SC2/WG2 北京会议上，与台湾和香港地区的代表取得一致，支持 HCC 的意见。但是，在此会议上，日本和韩国强烈反对 HCC，他们认为日文和韩文中的汉字与中文中的汉字是有差别的，不能把三者等同起来。从

而，这次会议未能达成一致性协议。这样就使 ISO10646 的二字节方案被搁置起来了。

2. 多字节方案

国际标准化组织无法使 ISO10646 的二字节方案为所有国家和地区接受，其焦点在于某些国家拒绝接受统一的大汉字集，因此，国际标准化组织决定对这一方案进行调整。首先决定把所有国家的文字字符集全部收入，这样就会收入数千个重复的汉字。很明，若仍用两个字节来为这些字符集编码，那是不可能的。于是，从 1989 年开始，国际标准化组织开始致力于制订一个 ISO10646 的多字节方案。经过一年时间的仔细研究，最终决定建立四字节编码集，即每个字符用四个字节来标识。这样的编码空间无疑是“海量”的，可以容纳 20 亿个字符编码，足以包含世界上各国文字的所有字符。

ISO10646 的四字节方案把整个编码空间分为四个级别：组(Group)、平面(Plane)、行(Rlow)、单元(Cell)。每个组由一系列平面组成，每个平面含若干行，每行含若干个单元。ISO10646 编码的四个字节的意义为：第一个字节表示组号，第二个字节表示平面号，第三个字节表示行号，第四个字节表示单元号。值得注意的是，ISO10646 保留了 ISO646 中的控制符代码，以实现与 ISO646 的兼容。所以，非控制符的编码之四个字节均不能取控制符的代码值。从而，组号、平面号、行号和单元号都不是连续的，其中缺乏 C0 区和 C1 区内代码之值。

按照上述规定形成的 ISO10646 编码空间实在太大了，故当前仅需集中考虑其中一个平面组的分配使用就足够了，这个组就是第一组，即 32 号组。由于 0～31(00H～1FH)均为控制符代码，所以 32 号组就成为第一组了。因为仅考虑一个平面组，所以可以省略组号字节(默认组号为 32)，用三个字节(平面号、行号、单元号)标识一个字符，从而 ISO10646 就成了三字节编码。在 1990 年 6 月举行的 ISO－IEC JTC1/SC2/WG2 札幌会议上，定下了

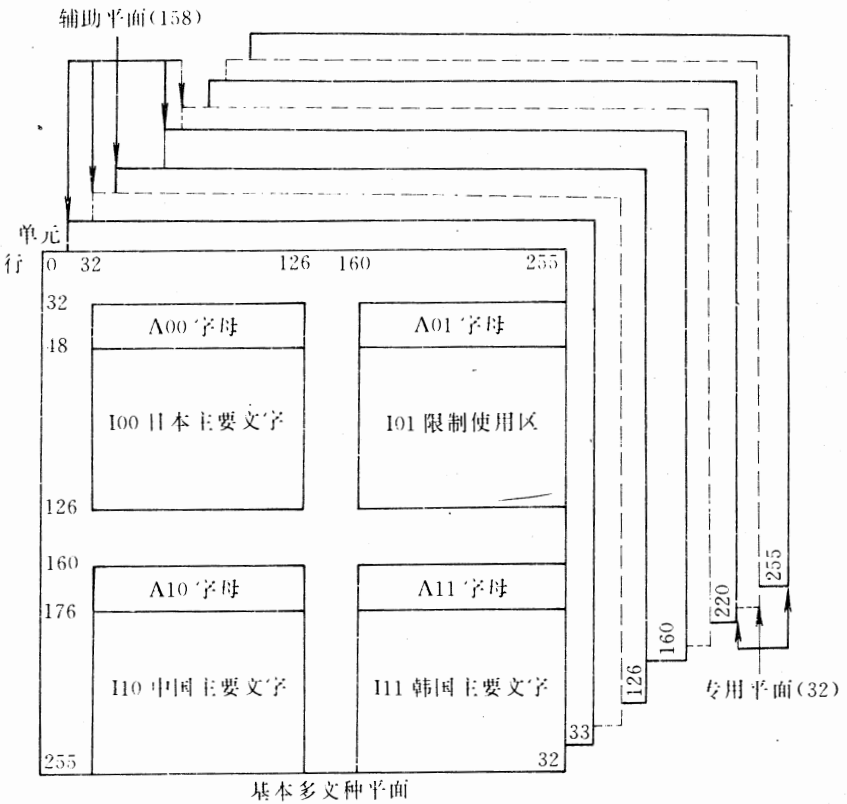


图 2-2　32 号组的多八位编码字符集

ISO10646 草案，图 2-2 给出了该草案对 32 号组内平面的使用规定，即 32 号组的多八位编码字符集。

我们从图 2-2 中可以看到，32 号组内的平面是这样分配使用的：32 号平面为基本多文种平面，33 号至 223 号平面为辅助平面，224 号至 255 号平面（即最后的 32 个平面）为专用平面。32 号平面分为八个字符区，即 A00 区、A01 区、A10 区、A11 区、I00 区、I01 区、I10 区、I11 区。其中，A00 区、A01 区、A10 区、A11 区用于西文字符，I00 区用于日文主要文字，I01 区为限制使用区，I10 区用于

中文主要文字，I11 区用于韩文主要文字。另外，辅助平面中的 40 号平面、48 号平面和 56 号平面分别用于中、日、韩文的其它文字。

尽管中国、日本和韩国均同意 ISO10646 的这一草案，但是它并没有得到计算机工业界的热烈响应。有不少计算机软、硬件厂商反对这一方案，他们认为这个方案实施太复杂，而且将引起存贮开销和通信时间开销的急剧增大。其根源在于编码不紧凑，冗余量较大。这是由于设计者过于强调与 ISO646 和 ISO2022 的兼容性，而忽略了新编码的效率问题所造成的。因此，作为新标准的 ISO10646 又背上了 ISO2022 的大多数复杂性的包袱。计算机工业界所期望的是一个高效的、统一的和易于实现的标准，然而，ISO10646 的草案与此要求相距甚远。一个国际标准若得不到工业界的支持和应用，则只能成为一个虚标准。

在拟定 ISO10646 的过程中，在计算机工业界中有人提出了一种称为 Unicode 的字符编码方案，它具有很强的竞争力，所以使 ISO10646 方案的进一步制订拖延下来了。

三、Unicode 字符编码方案

1. Unicode 的提出

在 1987 年，Xerox Palo Alto 研究中心的 Joe Becker 和 Lee Collins，以及 Apple 公司的 Mark Davis 试图研制一种适用于多文种处理的字符编码，并在 Xerox Star 计算机上进行了开发工作，使这种机器成功地成为多文种处理系统。后来，他们受到制订 ISO10646 的影响，决定研究一种比 ISO10646 更简单和实用的通用字符编码标准。他们把将要推出的编码标准命名为 Unicode，其意思是“唯一的、通用的和统一的字符编码”。他们把设计多文种处理代码的经验与对新标准的设想综合起来，并向有关厂商通报，获得了一些具有强大实力的公司的支持，这些公司有 Microsoft、Sun、Next 和 Novell 等。这些公司都派代表参加 Unicode 研制组，使 Unicode 的研制具有较快的进展。

Unicode 研制组于 1990 年正式推出了 Unicode1.0。不久，由强有力的跨国计算机公司组成的技术集团—Unicode Consortium 宣告成立，IBM、Apple、DEC、Lotus、Microsoft、Sun、Novell 和 Xerox 等公司都是这一集团的成员。另外，Unicode 又得到 ANSI 的支持，有人预言过，Unicode 将成为新一代的 ASCII。由于 Unicode 集团的成员都是世界上主要的计算机软、硬件厂商，所以 Unicode 完全有可能成为一个事实上的(de-facto)工业标准。

2. Unicode **的设计目标**

Unicode 是一种定长的二字节多文种字符集编码，它试图覆盖现有的有关国家和地区的标准，包括 GB2312、CNS11643、JIS0208 和 KSC5601 等。这些因素决定了 Unicode 的设计目标，其设计目标如下：

①完整性：Unicode 要覆盖文本中使用的所有字符；

②简单性：Unicode 代码是等长的(16 位)，每两个字节标识一个字符；

③单义性：每个代码明确地表示一个字符；

④正确性：每个被编码的字符都是由语言文字专家认可的真实字符；

⑤保真性：在进行与已有的字符编码标准间的转换时，不应该丢失任何原文数据。

以上设计目标使得 Unicode 有许多独特的地方。Unicode 大胆地开放了 C0 区和 C1 区，从而获得了较充足的编码空间，提高了编码效率。另外，Unicode 建立了通用汉字子集 UniHan，把中、日、韩文中用的汉字统一归并起来，并赋于唯一的编码。这一策略与 ISO10646 的初期方案类似。只有这样作后，才能使二字节编码空间能够容纳下多文种字符。UniHan 内收集了中、日、韩文中的 27000 个汉字，覆盖了现有的东亚汉字字符集。

3. **代码空间分配**

Unicode 的代码空间共分六个区，图 2-3 给出了 Unicode 代码

空间的分配情况。

图 2-3 Unicode 代码空间的分配

从图 2-3 中可以看到，Unicode 代码空间的第一区(0～8191)共 8192 个代码，这些代码用于标准的字母和符号，该区最低端的代码定义与 ASCII 的定义一致，该区高端部分用于古代字符；第二区(8192～12287)共 4096 个代码，这些代码用于标点符号、数学运算符、技术符号和一些新创建的符号；第三区(12288～16383)共 4096 个代码，这些代码用于中、日、韩文的字母(如中文的注音符号、日文的假名等)和标点符号；第四区(16384～59391)共 43008 个代码，其中 27000 个代码用于统一的汉字子集(UniHan)，其它代码用于中、日、韩文内的另外一些文字，以及用于今后扩充的字

符；第五区(59392～65023)共 5632 个代码，这些代码为用户专用，用户可以用这些代码自定义一些所要用的字符；第六区(65024～65535)共 512 个代码，这个区称为兼容性区域，用于把其它代码转换成 Unicode 代码。

四、ISO10646 和 Unicode 之间的关系

1. 对立与竞争

ISO10646 和 Unicode 是两种多文种字符编码方案，它们之间有着明显的差别。ISO10646 以四个字节标识一个字符，即使在一个组内，也要用三个字节标识一个字符。Unicode 则总是用两个字节标识一个字符。显然，使用 Unicode 需要的存贮开销和传输数据的时间开销都较少。ISO10646 采取对中、日、韩文内的汉字分别编码，所以会发生“一字多码”的现象。Unicode 则强调对汉字的统一编码，每个汉字在字符集内只能出现一次，保证作到“一字一码”。显然，Unicode 的编码空间较紧凑。ISO10646 强调对 ISO646 和 ISO2022 的兼容性，特别注意对控制符代码的兼容，故封锁了 C0 区和 C1 区。Unicode 只注意与 ASCII 的兼容，而不考虑对 ISO2022 的兼容，它开放了 C0 区和 C1 区。因此，Unicode 的编码空间利用率较高。

从以上的分析来看，似乎 Unicode 在各方面均要强于 ISO10646，其实并非如此。在不少方面 ISO10646 要强于 Unicode。例如，在兼容性方面，Unicode 肯定远不如 ISO10646，特别在控制符的兼容性方面更是如此。又如，在字符集的完整性方面，Unicode 也不如 ISO10646。就拿汉字来说吧，总共有 6 万多个。ISO10646 由于有巨大的编码空间，故完全可以把所有汉字全都收入。然而，Unicode 是绝对不可能这样作的，因为它的总编码字符数才 65536 个。

ISO10646 和 Unicode 的设计者都希望自己的方案成为国际标准，几年来它们相互竞争。从它们的背景来看，ISO10646 得到国

际标准化组织的支持，Unicode 得到 ANSI 和计算机工业界的支持。两者的后台各有千秋，所以它们能够相互对峙多年。

2. 协调与统一

ISO10646 和 Unicode 对立和竞争的同时，也表露出两者寻求统一的迹象。双方的设计者都不希望出现两个国际字符编码标准，都希望只有一个标准。Unicode 集团成员都原则上支持两种编码归并的建议，并希望把 Unicode 作为 ISO10646 中的基本多文种平面。Unicode 集团的某权威人士表示，Unicode 的崛起是为了影响 ISO10646，使其向合理的方向发展。这些就足以说明了 Unicode 一方寻求统一的意向。

然而，国际标准化组织在制订和讨论 ISO10646 方案时，也把 Unicode 的合理成分吸收进去，对 ISO10646 草案作了较大的调整。在 1991 年 8 月举行的 ISO－IEC JTC1/SC2/WG2 日内瓦会议上，一致通过了释放 C0 区和 C1 区，以及固定汉字区的重要决议，使 ISO10646 与 Unicode 基本趋于一致。实际上，按照此决议形成的 ISO10646 中的基本多文种平面与 Unicode 已经基本上一致了。这样，中、日、韩文内的汉字统一编码的大局已定。会议汇总了对我国提交的统一汉字字符集 HCS(Han Character Set)的技术意见，并决定由我国负责新字符集(草案)1.0 版(Unified Ideographic CJK Character Repertoire and Ordering V1.0 Draft)的修订。

由此看来，ISO10646 和 Unicode 的主要区别已变成双方汉字集(即 HCS 和 UniHan)的不同。HCS 与 Unicode 的差别较大，为了形成一个国际标准，应该使 HCS 和 UniHan 尽可能接近，最终统一起来。于是，双方达成了促成 HCS 与 UniHan 相互对齐的协议。通过双方分别对汉字字符集的调整，实现 HCS 和 UniHan 在以下几个方面的一致：

①在源字集上取得一致；

②采用相同的汉字认同规则；

③采用一致的分级方式；

④采用一致的汉字排序方式。

在国际标准化组织和 Unicode 集团双方的努力下，于 1992 年最终实现了 HCS 和 UniHan 在字码一级上的统一。这样也就达到了把 Unicode 作为 ISO10646 的基本多文种平面的目的。

五、小结

ISO10646 和 Unicode 的对立和竞争促进了这两种方案的发展，最终它们又趋向于统一。目前，ISO10646 方案已经在国际标准化组织获得通过，这一方案正式向全世界公布的时间不会很长了。这一新的国际标准公布后，可望被各国和计算机工业界接受。这种新标准一旦被使用和贯彻，将会给用户和系统设计人员带来许多方便。到那时，多文种处理已不再是一个难题，汉字信息处理系统的性能可获得较大的提高，而且再也不用为中西文处理的兼容性而耗尽脑汁。当然，在 ISO10646 即将正式公布之际，我们也决不能否认 Unicode 在新的国际标准形成过程中所起的关键作用。

第三章　基于DOS的系统设计

第一节　汉字系统总体设计

一、总述

1. 引言

目前，IBM－PC 微机系列是我国应用最广泛的计算机，DOS 操作系统是我国使用最广泛的操作系统。在今后一段时间内，它们两者仍将分别为我国的主流机种和主流操作系统。因此，我们有必要以 IBM－PC 系列机的硬件环境和 DOS 操作系统的软件环境为基础，来讨论汉字信息处理系统的设计技术。本章内容均基于 DOS 进行介绍、论述和讨论，故具有普遍性。

2. 设计目标

我们对汉字信息处理系统的设计提出以下目标：

(1)中西文兼容　我们要求系统既能处理汉字信息，又能处理西文信息。我国现代社会的信息中有不少是汉字与西文混合的，故这一个目标应该是作为基本目标提出的。另外，我国从国外引进了不少西文软件，为了充分利用这些已有的软件资源，不浪费作出的软件投资，也要求设计出的系统能正常支持这些西文软件。还有，在系统的使用方法上也应作到与西文系统相兼容，便于用户迅速掌握。

(2)尽量少改动原系统　汉字系统需要在 IBM－PC 系列机

的硬件支持下运行，我们要求尽可能少变动原系统的硬件，最好是原系统不作任何硬件变动就能支持汉字系统正常运行。这里所说的变动，包括修改原系统的硬件和增加硬件。如果达到了这一目标，就能大大方便用户，并且节省用户的投资。

(3)完善的用户界面　我们要求系统必须具有完善的用户界面，这就是说，系统的用户界面既要具有友好性，又要符合用户的使用习惯。因此，系统的用户界面应该丰富多彩，要与西文系统的用户界面兼容，而且还应向用户提供更方便、更能符合用户要求的手段。

(4)通用性强　系统应具有较好的通用性，它既能适用于各种类型的用户，又能与多种硬件配置相适配。IBM－PC 系列机的硬件配置比较丰富，特别是它能与多种显示适配卡和多种打印机相适配，然而，显示适配卡和打印机的类型与系统的实现密切相关，因此系统必须具有支持各种常用显示适配卡和打印机的能力。

(5)系统开销小　系统开销小是指系统自身所占用的资源应尽可能少，以使用户能使用尽可能多的资源。其中最主要的是内存资源，如果系统软件占用的内存量较小，那末就可能运行需要较大内存量的用户程序。但是，系统占用的内存量多少，往往会影响到系统的响应速度。因此，系统应该采用严格的模块结构，以广泛使用覆盖技术，并提供一定的调整手段，以在各种不同的要求下，调整系统的内存开销。

(6)组合性强　系统具有各种类型和功能的模块，用户可以用这些模块自由组合成各种级别的系统总体。这种组合拼装可以一次性完成，亦可在系统运行过程中对某些模块实现动态替换或释放，以形成新的系统。

二、设计思想

我们可以从两个途径来考虑汉字系统的设计。一个途径是从零开始考虑汉字系统的设计，这样设计出来的系统是一个全新的

系统，它可以具有较完善的汉字信息处理功能。但是，这种作法要花费较多的设计时间和劳动量，并且形成的新系统不会与原来的西文系统具有较好的兼容性，从而不能支持大量的已有软件运行。另一个途径是基于某个西文系统来设计汉字系统，尽量利用西文系统的已有部分，然后适当地对其有关部分进行修改和扩充，使其具有汉字信息处理能力。显然，用这种方法来设计汉字系统，可以花费较少的时间和劳动量，而且较易使设计出来的汉字系统与原西文系统兼容，从而能有效地利用西文系统的已有软件。但是，因为这种方法是以西文系统为基础的，而且要尽可能利用它，因此对汉字信息处理功能的实现会有所限制，从而有可能会使系统的汉字信息处理性能存在一些不完善之处。

通过对两种设计方法的对比，再根据前面提出的系统设计目标，权衡各方面的利弊，我们认为应该采用第二种方法来设计汉字系统。DOS 是 IBM－PC 系列机的主操作系统，它不但拥有极其广泛的用户，而且拥有非常丰富的应用软件。我们把 DOS 作为汉字系统的基础，采取一系列的措施对它进行扩充和适配，形成所需的汉字信息处理系统。我们把这种基于 DOS 的汉字系统设计称作 DOS 的汉化。

操作系统由处理器管理、存贮管理、文件管理和设备管理四部分组成。由于 DOS 是单任务系统，因此它没有处理器管理部分。所以，在对 DOS 扩充汉字信息处理功能时，仅需考虑存贮管理、文件管理和设备管理三个部分。

从信息处理角度来看，对汉字信息的处理与对西文信息的处理没有本质的区别，它们都是非数值处理。从西文操作系统的功能来看，存贮管理只是管理内存空间，而与内存空间中存放的对象无关。不管是汉字信息还是西文信息，或者是各种类型的数据，它们在内存中均一样存贮，仅是所占空间大小不同而已。文件管理提供对文件的访问、管理和保护功能，而与文件中数据所代表的意义无关。所以，对于汉字系统来说，它的存贮管理和文件管理没有特殊

要求，因此，汉字系统中的这两个部分内容可以完全利用 DOS 的原有部分，不必作任何修改和扩充。当然，若要进一步提高它们的性能，则亦要作相应的优化和改造。西文系统的设备管理要实现对设备的驱动，其目标为实现西文信息的输入和输出。然而，汉字系统对设备的驱动，则要实现汉字信息的输入和输出。显然，如果仍用西文系统内的设备驱动程序来驱动原来的设备，是不可能实现汉字信息的输入和输出的。所以，必须要改造和扩充这些驱动程序。综上所述，我们只要改造 DOS 的设备驱动程序，就能使之成为汉字系统。

DOS 的设备驱动程序存在于 ROM－BIOS 中。我们知道，ROM－BIOS 是系统中的最低层软件，DOS 内各层次对设备的驱动最终均要归结到对 ROM－BIOS 中相应程序的调用。因此，我们应该把注意力集中到 ROM－BIOS 上，只要对其中的有关设备驱动程序进行扩充，使其能驱动相应设备输入或输出汉字信息，那末所有调用 ROM－BIOS 来驱动设备进行输入和输出的系统程序或应用程序，也都具备了输入和输出汉字信息的功能。

IBM－PC 系统中与汉字信息的输入输出有关的常用设备为键盘、显示器和打印机。与键盘有关的 ROM－BIOS 程序为 16H 号中断处理程序，与显示器有关的 ROM－BIOS 程序为 10H 号中断处理程序，与打印机有关的 ROM－BIOS 程序为 17H 号和 5H 号中断处理程序。我们要对这四个中断处理程序进行扩充，使它们能实现汉字信息的输入或输出。

根据前面提出的设计目标，我们应该尽可能不改动硬件，由于 ROM－BIOS 程序已经固化在 ROM 中，因此我们不准备改动 ROM－BIOS 的内容，而是用新的中断处理程序去替换 ROM－BIOS 中的相应程序。根据 IBM－PC 系统的中断响应机制可知，这种中断处理程序的替换是十分方便的，只要修改相应的中断向量即可。经过替换后，新的中断处理程序就成为 BIOS 的一部分，因此它们随时都可能被高层软件调用。所以，新的中断处理程序应

该要常驻内存，而不能象可执行文件的执行那样暂驻内存。

三、总体设计

1. 系统结构

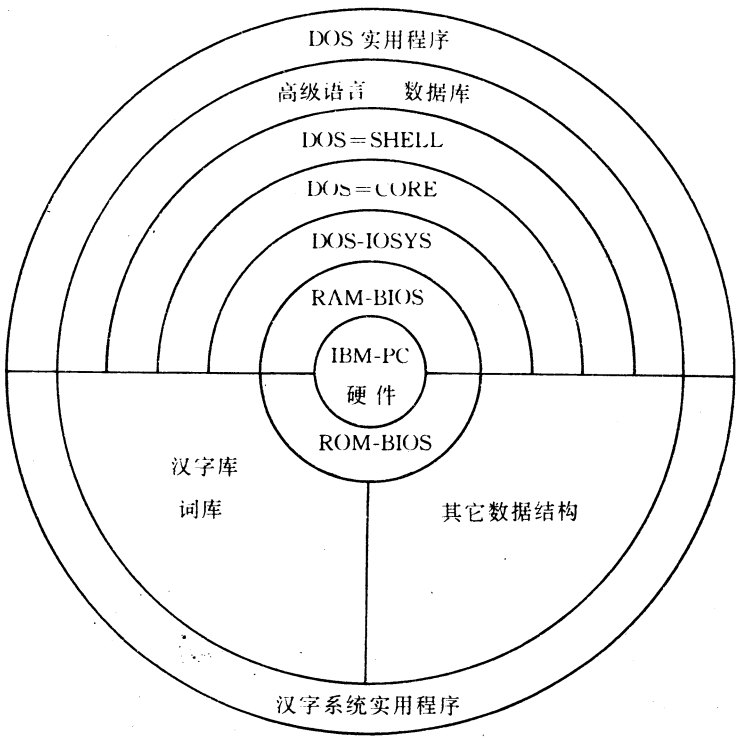
根据前面提出的汉字系统设计思想可知，我们设计的汉字系统由 DOS 的核心、ROM－BIOS 和新的具有汉字信息输入输出功能的四个中断处理程序组成，这四个中断处理程序分别对应于键盘、显示器和打印机。为了不对硬件进行改动，所以不把这四个中断处理程序写入 ROM 之中，而驻留在 RAM 内，又因为它们仍然处于系统的 BIOS 这一层次，故把它们称为 RAM－BIOS。这样，汉字系统的 BIOS 就由 ROM－BIOS 和 RAM－BIOS 组成。另外，作为一个汉字系统，还必须拥有汉字系统所特有的实用程序，比如汉字系统的维护程序、辅助程序和专用程序等。所以，汉字系统的实用程序比 DOS 丰富，它除了 DOS 的实用程序外，还包括它自己的汉字系统实用程序。除此之外，汉字系统还具有汉字库和一些汉字系统用的数据结构。图 3-1 给出了汉字系统的系统结构。

从图 3-1 中可以清楚地看到，汉字系统是基于 DOS 的，这就充分保证了汉字系统与 DOS 的兼容性。

2. RAM－BIOS 的结构

汉字系统中实现汉字信息输入和输出功能的关键部分是 RAM－BIOS，它由扩展的 16H 号、10H 号、17H 号和 5H 号中断处理程序组成。它们分别是键盘、显示器和打印机的驱动程序，所以可以把它们分成三个模块，分别称为键盘输入模、显示输出模块和打印输出模块。

这三个模块在结构上是彼此独立的，因此可以分别单独设计之。它们在逻辑上并不完全独立，彼此之间还存在着一些关系，但是这种关系并不十分密切。我们在单独设计这些模块时，只要适当注意到它们之间的这一些关系就可以了。



3—1　汉字系统的系统结构

这三个模块的结构雷同，都是由两部分组成。一部分是相应的中断处理程序，完成对相应设备的驱动和控制，并完成此设备的汉字信息输入或输出功能；另一部分是模块的自举程序，实现相应中断处理程序在RAM中的驻留和形成。这三个模块均属于RAM—BIOS，是系统的最低层软件。根据系统层次结构的规定，较高的层次能调用较低的层次，较低的层次不能调用较高的层次，否则会引

起系统死锁。DOS 就是根据这一原则设计的，而且 DOS 的内核是不可再入的，即不可以对它作递归调用或间接递归调用。鉴于这一点，这三个模块的中断处理程序部分至多只能调用 BIOS 这一层的有关功能模块，决不能调用 DOS 内核中的任何功能模块。我们必须在设计中牢记这一点，否则会引起 DOS 的再入问题，而发生死锁。但是，对于这三个模块中的自举程序部分，则不必受此限制，因为就其本身来说，只是一个一般的程序，它们在完成规定的自举任务后便退出系统，它们并不是 RAM－BIOS 的一部分。

四、显示输出模块

1. 汉字的显示输出过程

汉字的显示输出过程，实际上是把系统内欲显示输出之汉字的内码转换成屏幕上的汉字字形的过程，这个过程必须经过几个步骤才能实现。图 3-2 为汉字显示输出的过程。

在图 3-2 中，汉字显示处理程序把输出汉字的内码转换成该汉字字模在汉字库内的地址，再根据库地址从汉字库（或字符字模库）中取得汉字的字形信息，然后把字形信息送入缓冲区加工成显示规定的格式，最后把加工后的字形信息送入显示存贮区，这就实现了汉字的显示。

2. 汉字显示处理程序

汉字显示处理程序的任务是实现汉字的显示输出，它是汉字信息处理软件的必要组成部分。总的说来，汉字的显示输出在技术上要比西文字符的显示输出难得多。因此，汉字显示处理程序的设计就较为复杂。图 3-3 是汉字显示处理程序的流程。

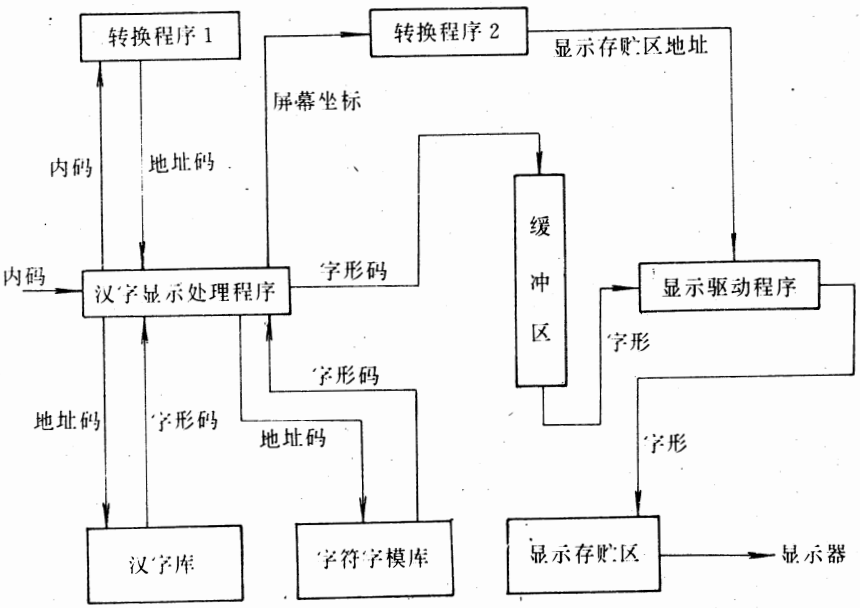


图 3-2　汉字显示输出的过程

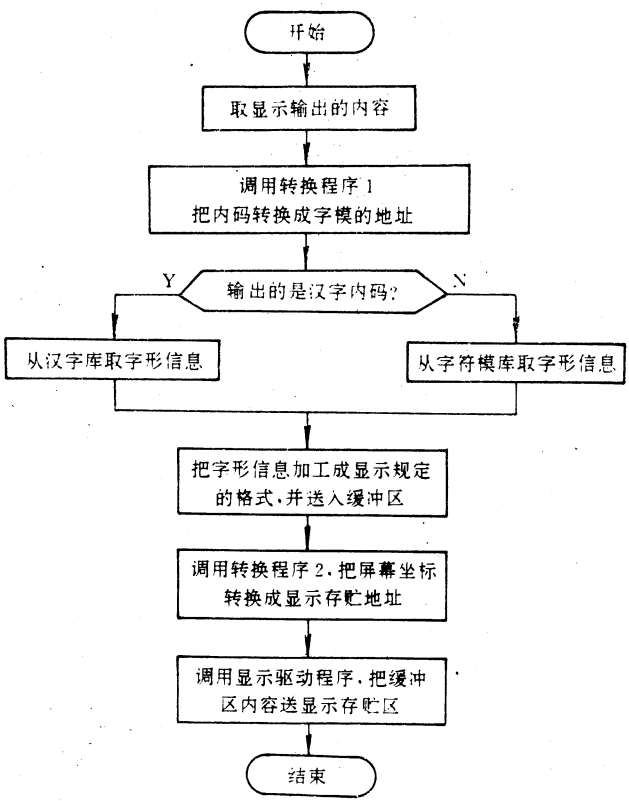


图 3－3　汉字显示处理程序的流程

汉字显示处理程序是显示输出模块的主体，当高层软件调用显示输出模块时，它才运行。它从约定的入口参数区中取出需显示输出之内容，此内容可以是汉字内码或西文字符内码；再调用转换程序 1，把取到的内码转换成它所对应汉字或字符的字模地址；接着根据取到的内码之类型，决定从汉字库或字符字模库中取出字

形信息，并把它加工成显示驱动程序规定的格式，存入缓冲区中；然后从入口参数区取出字符的显示位置(以屏幕坐标表示)，调用转换程序 2，把屏幕坐标转换成其对应的显示存贮区内之地址；最后调用显示驱动程序，把缓冲区中的内容送到指定的显示存贮区位置，就实现了显示输出处理。

3. 方案设计

目前，显示汉字的常用方案有两种：图形方式软方案和字符方式硬方案。显示输出模块可针对这两种方案设计有两个版本。用户在配置系统时，可以根据要求及系统的硬件环境选择合适的版本连入系统。

字符方式硬方案版本需要专门汉卡支持。为了使系统能支持常用的字符式汉卡，故显示输出模块的硬方案版本又为 H1 和 H2 两种版本，前者支持 CEGA 和 CVGA 汉卡，后者支持联想汉卡 。在硬方案中，汉字显示以字符方式进行，由汉卡完成相应的工作。

图形方式软方案版本不须增加新的硬件就能实现汉字的显示，这时显示器在图形方式下工作。显示输出模块的设计与系统所配置的显示适配卡有关。为了使系统能支持所有的常用显示适配卡，故显示输出模块具有五种软方案版本，它们是 S1、S2、S3、S4 和 S5，分别适用于 HGC、CGA、CGE、EGA 和 VGA 适配卡。因此，我们的系统在软方案下具有通用性。

由上述可知，显示输出模块具有多个版本，但是在运行时与系统相连的只有一个版本，这种连接可以由用户在配置系统时静态完成，亦可在系统运行时根据用户的要求动态完成。完成连接的执行者是系统配置程序，它根据接收到的连接符，静态或动态地把相应的显示输出版本与系统连接。图 3-4 给出了显示输出模块各版本与系统的连接关系。

我们知道，实现汉字显示的过程，在软方案下就是把汉字内码转换成汉字字形信息并送显示存贮区相应位置的过程。显示输出模块除完成汉字的显示外，还要完成一系列控制视频的功能。所

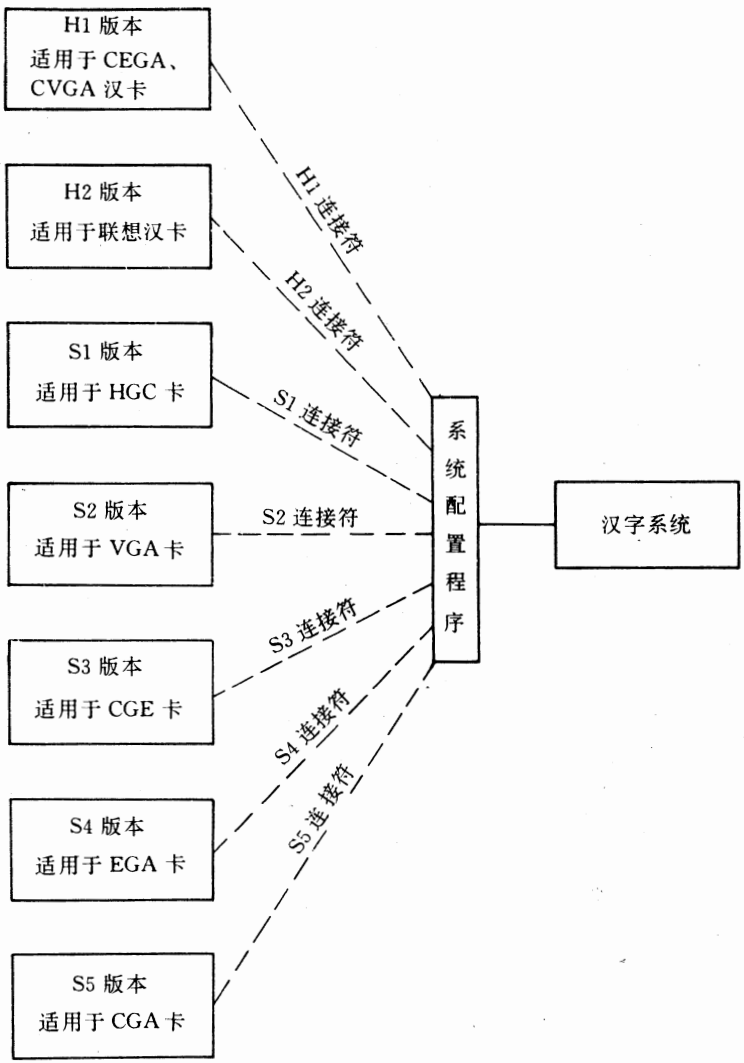


图 3-4　显示输出模块与系统的连接

以，显示输出模块还要实现在图形方式下对原来 ROM－BIOS 的显示输出模块中各项功能进行仿真。只有这样，才能使显示输出模块具有良好的中西文兼容性。视频是用户与系统交互的窗口，这个窗口中汉字信息显示的速度将直接关系到整个系统的性能。所以，在软方案中，汉字的显示速度是显示输出模块设计中的关键问题。

五、键盘输入模块

1. 汉字键盘输入过程

汉字的输入过程，实际上是把用户从键盘上敲入的汉字输入码转换成汉字内码的过程。也就是说，这个过程要实现把汉字的机外表示（汉字输入码）转换成汉字的机内表示（汉字内码）。图 3-5 给出了汉字键盘输入的过程。

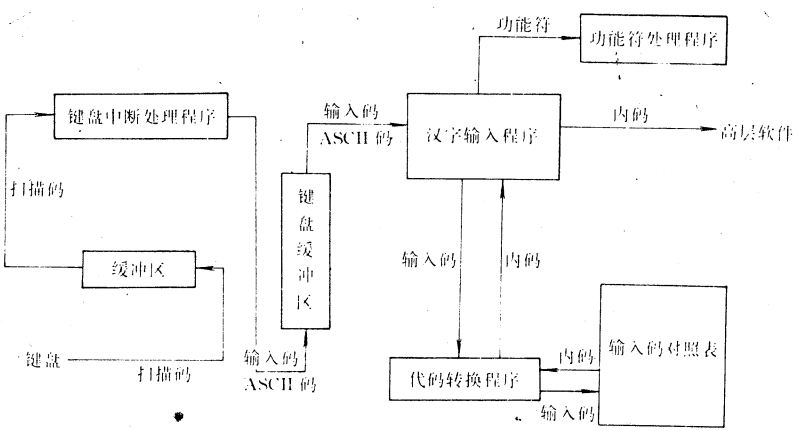


图 3-5 汉字键盘轮船入的过程

用户输入汉字时，在键盘上打入汉字的输入码（由若干个输入码符组成），键盘就产生这几个键（输入码符）的扫描码，存入键盘

内的缓冲区中；键盘发键盘完成中断，引出键盘中断处理程序执行；键盘中断处理程序取出扫描码，把它们转换成扩展 ASCII 码，并与扫描码一起存入键盘缓冲区(在内存中)；当主机内运行的程序需要输入信息时，就调用键盘输入模块；汉字输入程序从键盘缓冲区内取输入码，通过代码转换程序把它们转换成内码，转换过程中往往需要依靠输入码对照表。

2. 汉字输入程序

汉字输入程序是键盘输入模块的主体，它是解决汉字信息进入机内的关键。汉字输入程序的流程如图 3-6 所示。

汉字输入程序归属键盘输入模块，仅当高层软件调用键盘输入模块时，它才运行。它从键盘缓冲区中取出输入信息，对其进行正确性检查，若有错则进行出错处理；它再检查输入信息是否为输入码符，把非输入码符存于输出区，把输入码符存入输入码区；接着它判别输入码内的内容是否已是一个完整的汉字输入码，若是的话，就调用代码转换程序把其转换成对应的汉字内码；最后把内码存入输出区，输出区的内容为高层软件得到的输入内容。另外，在运行中它还识别系统定义的功能符，当从键盘缓冲区中取到功能符时，就调用功能符处理程序，完成相应的功能。

3. 代码转换程序

代码转换程序亦称代码处理程序，它是汉字输入程序的子程序，为汉字输入程序所调用。它实现把汉字输入码转换成汉字内码。它采用的方法须根据其对应的输入码的类型而定，对于可计算型输入码，则可通过计算公式得到汉字内码；对于非计算型输入码，则要借助于输入码对照表来完成转换，以得到汉字内码。显然，不同的输入码具有不同的代码转换程序。系统若配有多种汉字输入码，则其汉字输入程序内含有多个代码转换程序。当用户使用某一种输入码输入汉字时，系统就用相应的代码转换程序把汉字输入码转换成汉字内码。

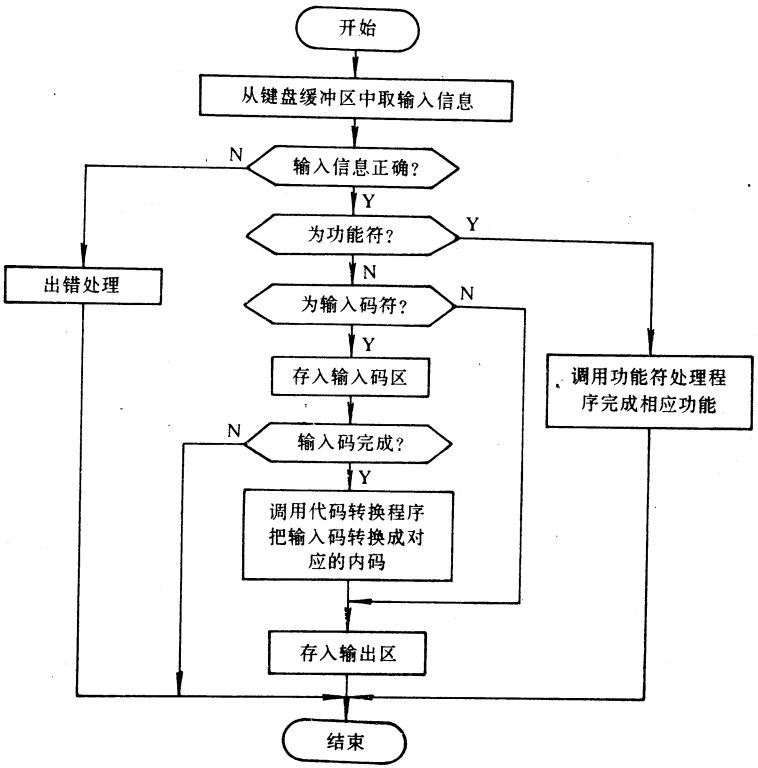


图 3-6 汉字输入程序的流程

4. 方案设计

我们要求设计出的汉字系统在输入方面要有较强的通用性，它能满足不同层次的用户的不同要求，既要面向非专业录入人员，又要面向专业录入人员。所以，系统必须配备多种输入码，以供用户选择使用。因此，键盘输入模块中拥有多个代码转换程序，以及功能符解释程序。一般来说，代码转换程序不会太小，若多个代码转换程序均进入内存，则必然会大大增加系统的内存开销。为了减小系统的内存开销，又考虑到一个用户所使用的输入码种类不会很多，所以我们规定同时只有四种输入码与系统连接。这种连接在

系统初启时 由用户选择，再由系统配置程序实现。另外又考虑到，一个用户在某一时刻只能使用一种输入码，故只要使当前被使用之输入码的代码转换程序进入内存即可。因此，我们可以设计一种切换机制，使得在系统运行过程中，用户可以随时动态切换当前输入方式，它们可以是当前与系统连接的四种输入码中的任意一种。图 3—7 给出了键盘输入模块内各处理程序间的关系。

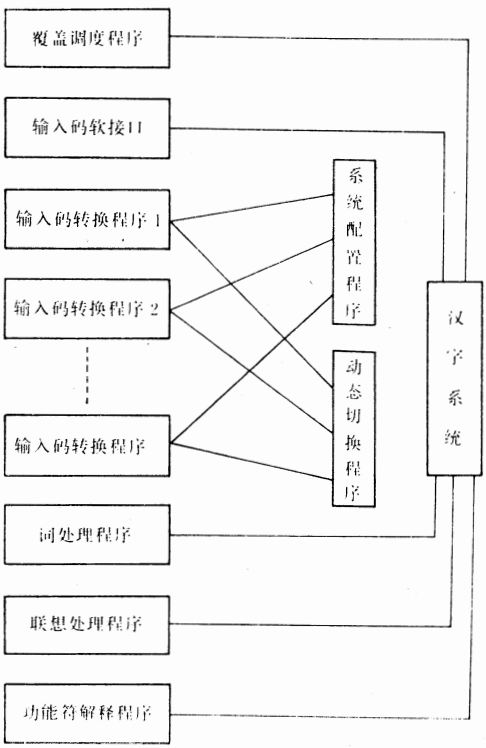


图 3-7 键盘输入模块内各处理程序的关系

以上的机制表明在各代码转换程序间实现了覆盖使用，因而在内存中必须设立一个代码转换程序覆盖区，用于存放当前使用之输入码的代码转换程序。由于代码转换程序紧密依赖于输入码对照表，为了实现有效的覆盖，故必须按统一的标准来设计代码转换程序及输入码对照表。

考虑到有些特殊用户需要把系统所配备的输入码之外的输入码用于本系统，所以，本模块中设计了两个输入码处理程序的软接口，用户可以把自行设计的输入码处理程序方便地通过软接口与系统相连接。

六、打印输出模块

1. 汉字打印输出过程

汉字的打印输出过程，实际上是把系统内欲打印输出汉字的内码转换成打印纸上的汉字字形的过程。这个过程如图 3-8 所示。

汉字打印输出处理程序把打印输出汉字的内码转换成该汉字字模的库地址，再根据库地址从汉字库中取得字模信息。若获得的字模信息为横向点阵，则还要把它转换成纵向点阵。然后把字模信息分解成打印数据送入打印缓冲区，最后把打印缓冲区的内容送打印机，就实现了汉字的打印输出。另外，如要打印当前屏幕上所显示之内容，则由屏幕拷贝程序把显示存贮区内的图形信息转换成打印数据，然后把打印数据送打印机，就实现了屏幕内容的打印输出。

2. 汉字打印处理程序

汉字打印处理程序的任务是实现汉字的打印输出。汉字的打印输出要比西文字符的打印输出困难得多，再加上用户对汉字打印输出的要求往往都比较高，因此汉字打印处理程序的实现也是比较复杂的。图 3-9 是汉字打印处理程序的流程。

汉字打印处理程序归属打印输出模块，当高层软件调用打印输出模块时，它才运行。它从约定的入口参数区中取出需打印输出的汉字内码，再调用地址转换程序把该汉字内码转换成其对应汉字之字模在汉字库内的地址；如果是打印 16×16 点阵汉字，就从 16×16 点阵汉字库中取字模，取到的是横向点阵字模，需调用字模转换程序把它转换成适合于打印用的纵向点阵字模。如果是打印 24×24 点阵汉字，就从 24×24 点阵汉字库中取字模，取到的就是适合于打印用的纵向点阵字模，接着判别是否要对打印输出的汉字作字型变换，若需要的话，就对字模点阵作字型变换处理（调用字型变换程序实现）；接着把点阵信息分解成打印数据送入打印缓冲区，然后把打印缓冲区内容送打印机，驱动打印机打印。

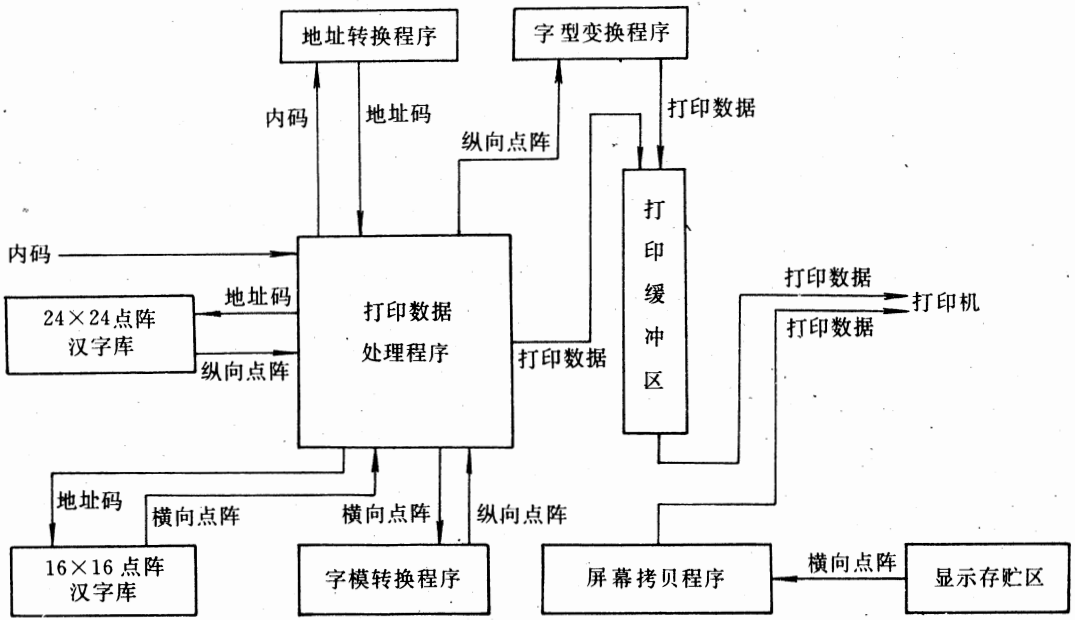


图 3—8　汉字打印输出过程

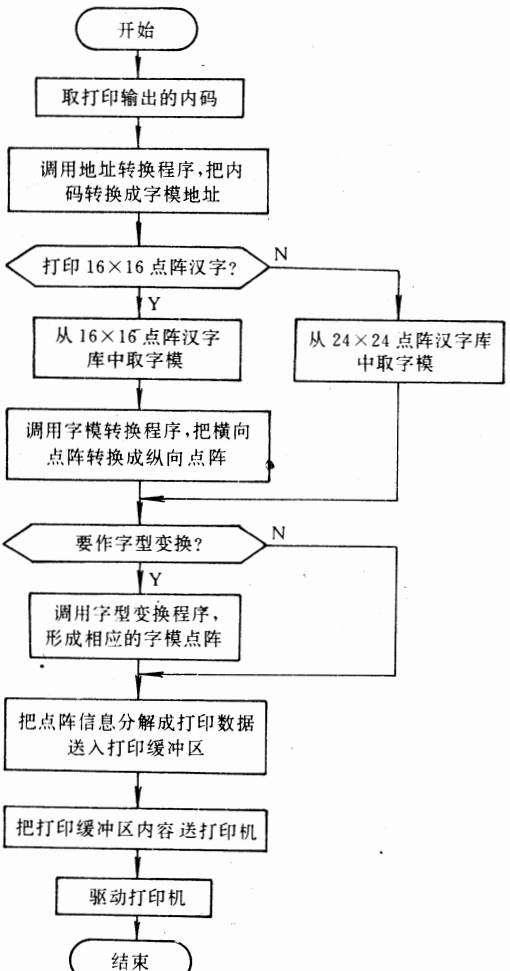


图 3-9 汉字打印处理程序的流程

这就完成了打印汉字的全过程。

屏幕拷贝亦属打印输出处理。当同时按下 Shift 键和 PrtSc 键时，就开始进行屏幕拷贝工作。屏幕拷贝程序从显示存贮区中依次取出屏幕内容对应的图形信息，然后把它们转换成打印数据，并逐一送打印机输出。这样就实现了把屏幕内容“复制”到打印纸上。

3. **方案设计**

打印机打印汉字时需要使用打印机的控制命令，由于打印机的控制命令尚未有统一标准，所以不同的打印机，其控制命令亦不同。为了使本模块能够适合于各种打印机，因此本模块设计成不确定型，即其并非针对某一特定打印机设计的，而是尽可能考虑各种打印机的共性。另外还要设计一个打印模块的装配程序，它可以根据用户提供的打印机型号或其它有关信息(如控制命令符)，把不确定的打印输出模块装配成针对某特定打印机的确定的打印输出模块。从而使本模块具有最大范围的适应性。

模块向用户提供两种工作方式，即汉字方式和西文方式，可以通过切换命令符动态地在这两种方式间进行切换。其主要目的在于开放打印机自身的所有控制命令，用户可以在西文方式下向打印机发出所要发的控制命令，然后立即再切换到汉字方式继续工作。这是开放打印机自身控制命令最简单和有效的方法。

第二节　全汉字集处理系统的设计

一、引言

目前的计算机汉字信息处理系统大多只支持汉字基本集(GB2312)中的6763个汉字，这已经能满足一般的使用要求。由于汉字的数量十分庞大，据统计共有6万多个汉字，因此随着汉字系统在各部门的进一步深入应用，用户对系统所支持的汉字集大小会提出新的要求，特别是一些专业部门，比如图书馆、印刷厂，以及考古、历史和文学研究单位，它们所使用的汉字范围远超出汉字基本集。即使在一些通常的应用中，也时常有所用汉字超出基本集范围的现象发生，如人名、地名等。所以，为了拓宽计算机汉字系统的应用面，研究和设计能支持全汉字集的汉字处理系统已成为一个迫切的问题被提上了议事日程。

我国是世界上文明发达最早的国家之一，具有丰富的文化宝藏，我国的古汉字就是这些 宝藏中的重要部分，它们是我国古代灿烂文化的象征。我国的古汉字包括甲骨文、金文、战国文和小篆等，对这些古汉字的研究，有助于对我国古代文化的了解，并且对我国历史的研究和人类发展的研究都有重要作用。因此，当前不但国内学者重视对我国古汉字的研究，而且国外的不少学者也投身到这项研究工作中去。所以，对古汉字的研究已经成为一门国际性的学科。当前，现代汉字信息的计算机处理技术获得了较快的发展和广泛的应用，但是我国古汉字研究领域内的计算机应用仍是一个空白。如何使古汉字研究走向现代化，已成为一个迫切需要解决的问题，将我国古汉字纳入到计算机信息处理的轨道上来，已成为今后进一步开展古汉字研究的必然趋势。把现代化的计算机技术与我国古老的文化遗产结合起来，其本身就是一件十分有意义的工作。

根据以上的论述和说明，我们来给全汉字集下个定义。我们认为，全汉字集处理系统的全汉字集没有必要包含所有的 6 万多个现代汉字，而应剔除其中大量的“死字”、““甚罕用字”和“特别异体字”。但是，全汉字集应包含简化汉字、繁体汉字和古汉字。关于简化汉字，国家标准总局已颁布了汉字基本集(GB2312)、汉字第二辅助集(GB7589)和汉字第四辅助集(GB7590)等国家标准，这三个集共收汉字 2 万多个。关于繁体汉字，国家标准总局拟颁布汉字第一辅助集、汉字第三辅助集和汉字第五辅助集，这三个集共收汉字 2 万多个。关于古汉字，有甲骨文、金文、战国文和小篆等文字，共计约 2 万多字。因此，全汉字集应含简化汉字、繁体汉字和古汉字总数为 7 万多个。

二、总体设计

1. 设计目标

全汉字集处理系统以 IBM－PC 系列机的硬件环境和 DOS 的

软件环境为基础，在不改变原来系统的硬件装置前提下，配上全汉字集处理系统，就能实现对全汉字集内的汉字的处理。因此，我们要设计的全汉字集系统仍然是基于DOS的。我们提出的全汉字集系统的设计目标如下：

(1)系统必须与DOS高度兼容　它能保持原西文系统的所有功能，要求DOS软件的汉化范围尽可能小，甚至不需汉化就能在全汉字集系统支持下顺利运行。对DOS本身的改动亦要尽可能少，有利于今后系统版本的升格。

(2)现代汉字和古汉字能使用到系统各级　现代汉字和古汉字不仅可以作为文件内容、字符串内容和文件名，而且可以作为高级语言及数据库中的标识符、变量名、属性名和关系名等。

(3)系统具有通用性　把对硬件的依赖程度减到最小。系统能支持各种型号的打印机，可打印出丰富的字型。系统能支持各种显示适配卡，并根据它们具有的分辨率形成相应的屏幕显示格式。

(4)系统具有完善的信息录入体系　系统配有便于各种层次操作人员使用的优选的现代汉字输入方式，还配有便于各种古汉字输入和符合古汉字特点的古汉字输入方式。

(5)系统尽可能少占内存空间　在能满足响应速度的前提下，把系统所需的内存开销降到最低限度。

(6)全汉字集内的所有汉字在系统内能同时并存，能实现对它们的混合处理。这就是说，系统能同时对简化汉字、繁体汉字和古汉字进行处理，这就满足了用户的最高要求。

为了实现上述目标，必须解决好以下问题：内码设计、输入码设计、数据结构设计和有关算法设计。显然，其中以内码设计最为关键，后面将重点介绍之。

2. 系统总体结构

根据前面提出的系统设计目标可知，全汉字集系统必须基于DOS，故要保持它的层次结构，只在极有限的范围对其进行一些修改和扩充。主要体现在BIOS层要扩充一些具有汉字输入输出功

能的模块、各种汉字库和有关的数据结构，另外在实用程序层也要增 加一些系统服务程序，用于完成建库、造字、造词和系统维护等。全汉字集系统的总体结构如图 3-10 所示。

图 3-10 系统总体结构

3. 内码的设计目标

国际标准化组织(ISO)已经公布了新的字符编码国际标准—ISO 10646，它是一种 16 位编码，这意味着，如要正式采用它，则必须修改计算机的字符处理和存贮机制，由现在的单字节处理改为双字节处理。但是，我们当前使用的计算机仍是基于单字节字符处理与存贮机制的，尚不能彻底支持 ISO 10646。而且，国内已有的

汉字系统大多以变形国标码为汉字内码，它与 ISO 10646 不兼容。所以，鉴于目前的物质基础和对已有汉字系统的兼容性，我们的全汉字集系统尚不能采用 ISO 10646 作为其汉字内码，还需自行设计全汉字集的内码。下面我们就基于目前的物质基础来讨论全汉字集内码的设计问题。

内码的设计是一个系统工程问题，它将会涉及到计算机系统的方方面面，最终会影响系统的性能和效率。我们认为，设计的全汉字集内码必须实现下列基本目标：

(1)编码空间应足够大　根据前面所述，编码总数应达到 8 万左右。

(2)中西文兼容性要好　要求汉字编码与西文字符编码不发生混淆，两者能同时存在，并且能实现中西文兼容的显示和屏幕编辑。

(3)具有较好的定义完备性　这是指汉字定义的唯一性和确定性，唯一性意味着汉字与编码是一一对应的，确定性意味着汉字的确定不受多字节内码字节偏移的影响。

(4)具有系统实现的可能性和简单性　这意味着内码的设计不能脱离系统实现，缺乏实现可能性的内码是毫无意义的。易于实现的编码会增加被采用的可能性，因为它能减少实现的工作量。

我们认为，基于目前的物质基础，设计出的全汉字集内码必须达到以上四个目标，否则就没有实用价值，也就称不上是真正的汉字内码。当然，除了上述基本目标之外，设计时还可以进一步考虑其它一些目标，比如，码长应尽可能短，易于建立汉字序值，与有关标准有简单的映射关系，以及与已有汉字系统的兼容性等目标。

三、CNCC 码的设计

1. 定义

CNCC(Character-Number-Character-Character)码是等长码，

它用四个字节表示一个汉字。这种编码由字母和数字组成，为了避免与西文字符串的表示发生混淆，故须采用字母和数字的定序混杂。采用什么样的字母数字组合，则取决于这种结构的编码总数，以及这种结构的编码与西文字符串混淆的几率。经测试，〈字母〉〈数字〉〈字母〉〈字母〉结构与西文字符串的混淆几率极小（其中的字母是指大写字母），其编码总数可达 26×10×26×26＝175760。

由于这种编码空间比我们所要求的大得多，故在实际使用中可以取其子集作为内码集，以进一步降低汉字内码与西文字符串的混淆几率。我们把表示十六进制数字的字母“A”～“F”从编码符中剔除，则可以使混淆几率接近于零，而编码总数仍可达到 20×10×20×20＝80000，显然能满足全汉字集的要求。下面给出 CNCC 码的定义。

令 CNCC 码的总集为 T，A＝〔“G”，“Z”〕，B＝〔“0”，“9”〕，则：

$$T=\{x_1x_2x_3x_4 \mid x_1,x_3,x_4\in A,x_2\in B\}$$

再令 $T_i(i=0,1,\cdots,9)$ 为 T 的子集，它们的定义如下：

$$T_0=\{x_1x_2x_3x_4 \mid x_1\in\{G,H\},x_2\in B,x_3,x_4\in A\}$$

$$T_1=\{x_1x_2x_3x_4 \mid x_1\in\{I,J\},x_2\in B,x_3,x_4\in A\}$$

$$T_2=\{x_1x_2x_3x_4 \mid x_1\in\{K,L\},x_2\in B,x_3,x_4\in A\}$$

$$T_3=\{x_1x_2x_3x_4 \mid x_1\in\{M,N\},x_2\in B,x_3,x_4\in A\}$$

$$T_4=\{x_1x_2x_3x_4 \mid x_1\in\{O,P\},x_2\in B,x_3,x_4\in A\}$$

$$T_5=\{x_1x_2x_3x_4 \mid x_1\in\{Q,R\},x_2\in B,x_3,x_4\in A\}$$

$$T_6=\{x_1x_2x_3x_4 \mid x_1\in\{S,T\},x_2\in B,x_3,x_4\in A\}$$

$$T_7=\{x_1x_2x_3x_4 \mid x_1\in\{U,V\},x_2\in B,x_3,x_4\in A\}$$

$$T_8=\{x_1x_2x_3x_4 \mid x_1\in\{W,X\},x_2\in B,x_3,x_4\in A\}$$

$$T_9=\{x_1x_2x_3x_4 \mid x_1\in\{Y,Z\},x_2\in B,x_3,x_4\in A\}$$

显然，$T_i(i=0,1,\cdots,9)$ 的编码数均为 2×10×20×20＝8000，而且有：

$$T=T_0\cup T_1\cup T_2\cup\cdots\cup T_9$$

我们把 T_0 定义为汉字基本集的内码空间，把 T_1～T_5 分别定

义为汉字第一辅助集至第五辅助集的内码空间，把 T_6～T_9 定义为古汉字的内码空间，其中 T_6 为甲骨文内码集，T_7 为金文内码集，T_8 为战国文内码集，T_9 为小篆内码集。

2. **性能**

用 CNCC 码作为汉字内码能获得以下性能：

(1)这种码易被系统内核接受，它实际上是专门序列的西文字符串，所以，凡是西文字符能正常存在的地方，汉字也能正常存在。从而使对系统内核的汉化工作量减到最小，并且有利于作高兼容性的中西文处理。

(2)这种码的长度为 4 字节，为汉字自然长度(2 字节)的整数倍，所以较易实现中西文兼容的屏幕编辑，我们将在本节中对此作进一步讨论。

(3)这种码的组成字符具有连续性，从编码的定义可知，这种码是自然有序的，故有利于汉字的排序操作 。

(4)这种码的结构是非对称的，故有利于增强编码的确定性(防移性)，并且便于系统内对汉字整字边界的识别。

(5)这种码是等长码，因此能符合通常的处理习惯，简化了系统对汉字的处理与存贮。

CNCC 码作为汉字内码还具有下列不足之处：

(1)这种码与有关标准(如 GB2312)间没有简单的映射关系，使得彼此间的转换算法较复杂，从而缺乏与已有汉字系统的兼容性。

(2)这种码的码长较长，增大了信息冗余度，使得汉字信息的存贮开销较大。

(3)这种码尽管能实现中西文兼容的屏幕编辑，但是须付出系统开销。

综观 CNCC 码的性能，可以认为，这种码作为全汉字集内码是可行的。

四、TTB 码的设计

1. 定义

TTB(Two-Three Bytes)码是一种不等长码，它用两个或三个字节表示一个汉字。为了在系统内使汉字的表示与西文字符有绝对的区别，TTB 码的组成符与变形国标码一样，它们的 MSB(即最高位)均置为 1。下面给出 TTB 码的定义。

令 TTB 码总集为 S，D_1=〔A1H，FEH〕，D_2=〔A1H，A9H〕，D_3=〔F8H，FEH〕，D_4=〔AAH，AFH〕，D_5=〔80H，A0H〕，D_6=〔B0H，F7H〕。

再令 S_i(i=0，1，…，9)为 S 之子集，并且有：

$$S=S_0 \cup S_1 \cup \cdots \cup S_9$$

则定义：

$$S_0=\{\mathbf{x}_1\mathbf{x}_2 \mid \mathbf{x}_1,\mathbf{x}_2 \in D_1\}$$

$$S_i=\{\mathbf{x}_0\mathbf{x}_1\mathbf{x}_2 \mid \mathbf{x}_0 \in D_5,\mathbf{x}_1,\mathbf{x}_2 \in D_1\} \qquad (i=1,2\cdots,9)$$

其中，S_0 的编码总数与 S_i(i=1，2，…9)的编码总数均为 94×94=8836，故 TTB 码的编码总数可以超过 8 万个。

我们把 S_0 定义为汉字基本集的编码空间，把 S_1 至 S_5 分别定义为汉字第一辅助集至第五辅助集的编码空间，把 S_6 至 S_9 定义为古汉字的编码空间。由此可知，TTB 码对汉字基本集的编码与变形国标码完全相同，它对基本集之外汉字的编码仅是在变形国标码的基础上增加一个标识字节(首字节)，该标识字节为基本集编码中不用之值，故很易把它识别出来。

2. 性能

TTB 码作为汉字内码的最主要优点是与国家标准(GB2312 等)间有非常简单的映射关系，并与已有的汉字系统兼容。TTB 码的 S_0 空间之编码规则与 GB2312 相一致，S_1 至 S_5 空间之编码(后两个字节)的规则亦与相应的国家标准相一致，从而形成了 TTB 码与国家标准间的直接映射关系。同时，由于 TTB 码对汉字基本

集的编码实际上就是目前广为采用的变形国标码，所以，在汉字基本集范围内，以 TTB 码为汉字内码的汉字系统与已有的汉字系统高度兼容。这一点无疑是十分可贵的。

TTB 码的组合符均非普通的 ASCII 符，所以，系统内汉字的表示与西文字符串的混淆几率绝对为零。TTB 码的三字节编码空间中(S_1 至 S_9)，由于三个码符中有一个为极易识别的标识字节，故这部分编码具有非对称性，从而增强了编码的防移性。另外，因其首字节为标识字节，故有可能在采取措施后实现中西文兼容的屏幕编辑，我们将在后面对此作讨论。相对于 CNCC 码来说，TTB 码的码长较短，减小了信息的冗余度，也有利于节省存贮开销。

TTB 码作为汉字内码具有下列不足之处：

(1)这种码是不等长码，会给系统的处理增添一些麻烦。

(2)这种码的组成符与西文字符不同，故有些系统的内核部分可能不允许其正常存在，因此必须对西文系统进行汉化。

(3)尽管这种码的绝大部分编码结构是非对称的，但是其基本集的编码结构是对称的，故这部分编码的防移性差。又因为日常使用中基本集的使用频率极高，因此总的说来，这种编码的防移性不如 CNCC 码。

(4)这种码的 S_0 之外的编码占用了 D_5 字符空间的一部分，D_5 空间是规定保留的控制符编码空间，如果原来系统中使用了该空间的控制符编码，则有可能会发生问题。好在 TTB 码只用了 D_5 空间内的 9 个字符编码，只要调整得好，就能解决这一问题。

通过对 TTB 码的全面考察，可以认为，TTB 码作为全汉字集内码是可行的。

五、输入码的设计

1. 现代汉字的输入码

全汉字集系统内应配置多种国内流行的优选汉字输入码，并设计有软接口，允许作进一步扩充。这些输入码包括音、形、音形、

形音和流水五种类型的编码，以满足不同层次用户的使用。

每一种音码和流水码既能适用于简化汉字，又能适用于繁体汉字。每一种形码、形音码和音形码均设计了两种版本，分别适用于简化汉字和繁体汉字。

为了加速现代汉字信息的录入速度，系统内还配置了多种词输入方式。另外，系统还提供了具有一定智能程度的联想输入方式，可以用系统内各种输入码来实现汉字的联想输入功能。

2. 古汉字的输入码

要实现计算机对古汉字信息的处理，就必须解决古汉字的输入问题，从而古汉字输入码的设计就成为必须要解决的主要问题之一。古汉字输入码的设计必须根据古汉字的特点进行，而不能照搬现代汉字的输入码。下面以甲骨文为例来介绍古汉字输入码的设计方法。

到目前为止，甲骨文中可以确切辨识的单字只有1000多个，而且在已辨识的字中仍有部分尚不知读音。这也就是说，在本系统收入的近4000个甲骨文单字中，绝大多数单字是不可辨识的和不知读音的。此外，甲骨文的大多数笔顺并不严格，笔数亦不精确，位置结构也不固定，故很难单纯从音和形的角度来考虑甲骨文的输入码。根据甲骨文的特点，本系统采用甲骨文义形码作为甲骨文输入码。这种编码结合义和形两方面因素来对甲骨文单字进行编码，所以，这是一种以义聚同类，以形别先后的编码。

为了适应小键盘输入，这种编码将甲骨文通常的分类进行筛选合并，形成26个大类，分别用26个字母表示。为了便于记忆，尽量使这些字母与大类名称的汉语拼音首字母一致。例如，“天干”类用“t”表示，“地支”类用“d”表示，“卜祀”类用“b”表示等。

义形码为四位等长码，每一位的含义为：

第一位代表大类，即为大类之表示符；

第二位代表部首，即为部首之表示符；

第三位代表分组，即为同一部首之单字组的符号；

第四位代表序号，即组内单字的序号符。

在输入过程中，操作者可以根据欲输入之甲骨文字的形体结构，确定所属的部首和大类。然后，首先键入大类的表示符，再依次键入部首表示符、分组符和序号符。输入部分设计成逐字符提示的形式，进一步提供了输入甲骨文信息的方便性。

六、信息输入处理

1. 输入模块的结构

全汉字集系统的输入模块实现现代汉字和古汉字信息的键盘输入，即把现代汉字和古汉字的输入码转换成其对应的内码。图 3—11 给出了输入模块的结构与信息流。

图 3-11 中组成输入模块的各个部分的功能如下：

(1)键盘中断处理程序　该程序把输入字符的扫描码转换成 ASCII 码，并存入键盘缓冲区。这是一个硬中断处理程序。

(2)代码识别程序　该程序把输入内容中的输入码符和功能符区分开来，并分别递交给输入码转换程序和功能符解释程序。

(3)输入码转换程序　该程序把输入码转换成其对应的内码。系统内的所有输入码转换程序用覆盖法共享内存中的同一个输入码转换程序区。

(4)输入码对照表　该表是输入码转换程序所用的一种重要数据结构，它反映了输入码与内码之间的对应关系。系统内的所有输入码对照表用覆盖法来共享内存中的同一个输入码对照表区。

(5)覆盖管理程序　该程序实现对输入码转换程序区和输入码对照表区中内容的覆盖调度，使当前输入方式所有的输入码转换程序和输入码对照表进入内存中相应的共享区域。

2. 输入码—内码转换算法

输入码—内码转换算法是输入模块中的关键算法。本系统中现代汉字的输入码至内码的转换，采用对输入码对照表的检索来实现。然而，古汉字的这种转换采用了一种独特的算法，这种算法

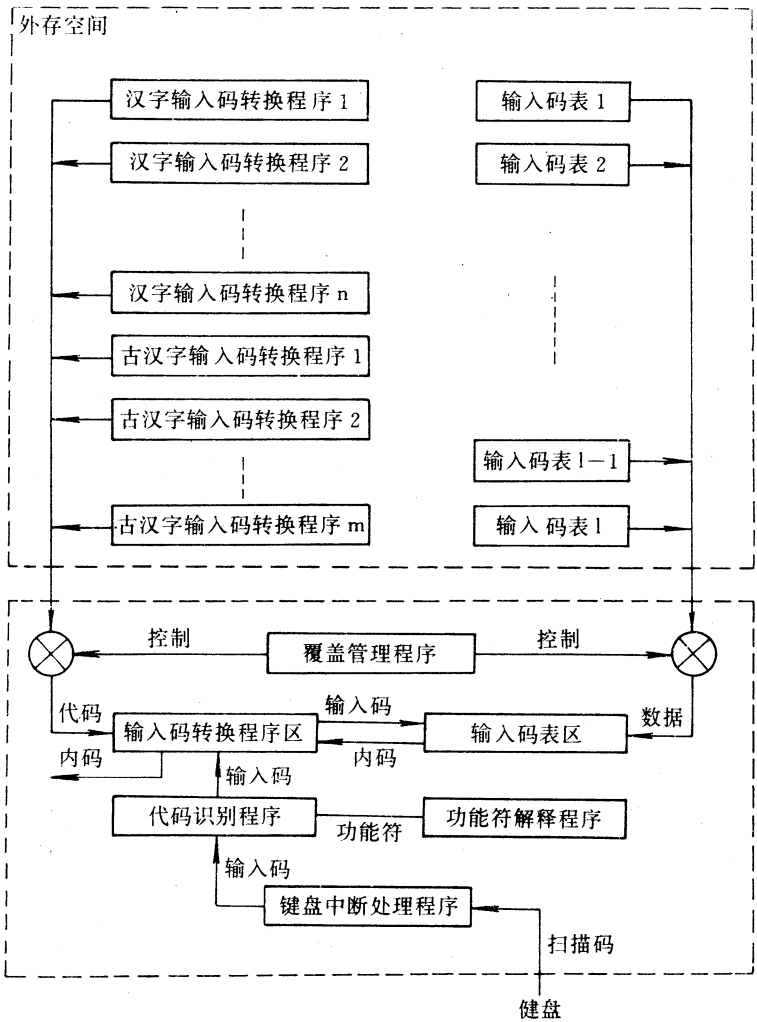


图 3—11　输入模块的结构与信息流

须依靠一张重码统计表，使用纯粹的计算法来完成输入码至内码的转换，具有较快的转换速度。重码统计表的结构如图 3-12 所示。

重码统计表由表项组成，每个表项对应于一种输入码的前三

个字符(第四个字符实际可视作选择符),表项内容为对应于这个输入码的重码字个数。表项按照其对应输入码的字母序排列。

我们在讨论算法之前,有必要先介绍一下古汉字与其内码的对应关系。这了使输入码至内码的转换算法能达到最简程度,本系统规定,古汉字的内部码序与其输入码序相一致,即古汉字的输入码在输入码集中若排在第 q 位,则其内码在内码集中也排在第 q 位。

在确定了内码序列和重码统计表后,下面以 CNCC 码作为内码为例,设计出古汉字的输入码—内码转换算法。

令某古汉字的输入码为 $e_1e_2e_3e_4$,并且有 $e_i \in \{a,b,\cdots z\}$,其中 $i=1,2,3,4$。令 a_i 为 e_i 的 ASCII 码。再令该古汉字的内码为 $c_1c_2c_3c_4$,b_i 为 c_i 的 ASCII 码,其中 $i=1,2,3,4$。

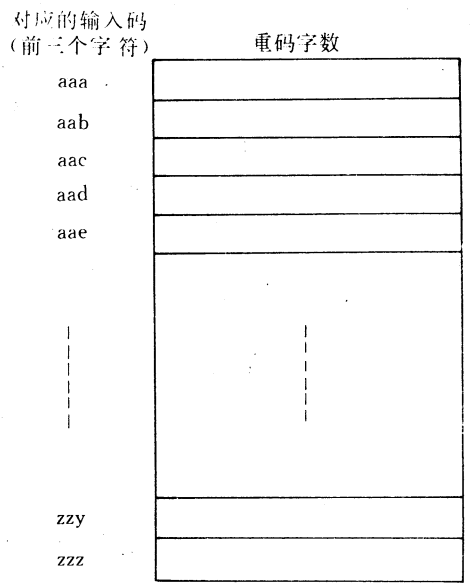


图 3-12 重码统计表的结构

(1)计算内码的序号 N

①根据输入码的前三个字符算出对应它的重码统计表表项的项序号 N_0，计算公式为：

$$N_0 = \sum_{i=1}^{3} (a_i\text{-}61H) \times 26^{3-i}$$

②根据 N_0 和输入码的第四个字符算出内码的序号 N，计算公式为：

$$N_0 = \sum_{i=0}^{N_0-1} T[i] + (a_4 - 61H)$$ 式中的 T[i]为重码统计表第 i 项的内容。

(2)根据 N 得到内码 $c_1c_2c_3c_4$

①计算内码首字符的 ASCII 码 b_1

令古汉字集中首字符的最小 ASCII 码为 r，又令：

$$t_1 = (N - N \bmod 4000)/4000$$

则：$b_1 = r + t_1$

②计算内码的第一个字符的 ASCII 码 b_2

令：$t_2 = N - t_1 \times 4000$

则：$b_2 = (t_2 - t_2 \bmod 400)/400 + 30H$

③计算内码的第三个字符的 ASCII 码 b_3

令：$t_3 = N - t_1 \times 4000 - t_2 \times 400$

则：$b_3 = (t_3 - t_3 \bmod 20)/20 + 47H$

④计算内码的第四个字符的 ASCII 码 b_4

令：$t_4 = N - t_1 \times 4000 - t_2 \times 400 - t_3 \times 20$

则：$b_4 = t_4 + 47H$

我们获得了 b_1、b_2、b_3、b_4，即获得了内码 $c_1c_2c_3c_4$。

3. 词处理和联想处理

输入模块中还有词处理和联想处理程序，借助这两部分程序可以实现词输入功能和联想输入功能。关于这两个部分的设计，请参阅本书第五章的有关内容。

七、信息输出处理

1. 输出模块的结构

全汉字集系统的信息输出模块主要实现把现代汉字和古汉字的内码转换成其对应文字的字形信息(字模),并送显示器或打印机形成相应的字形。另外,输出模块还要完成一些显示和打印的辅助操作。图 3—13 给出了输出模块的结构与信息流。

(1)显示输出模块　该模块主要完成内码到文字字形信息的转换,另外还完成显示器的一系列辅助操作。本系统有多个显示输出模块,它们分别对应于相应的显示适配卡,系统根据所配置的显示适配卡,选用相应的模块。

图 3-13 中组成输出模块的各部分之功能如下:

(2)显示输出模块配置程序　该程序在系统自举时根据系统所配置的显示适配卡,把相应的显示输出模块调入内存中的显示输出模块区,与系统连接。

(3)现代汉字库　现代汉字库包括基本集一级字库、二级字库,以及第一辅助集至第五辅助集字库。

(4)古汉字库　古汉字库包括甲骨文字库、金文字库、战国文字库和小篆字库,其中存放相应古汉字的字形信息。

(5)字库管理程序　该程序实现把内码转换成字形信息的地址,实现对字库的访问,获得相应文字的字形信息。

(6)打印输出模块　该模块主要完成把内码转换成对应的字形信息,另外还完成打印机的一系列辅助操作。本系统能使打印输出模块适用于各种型号的打印机。

(7)打印机控制命令信息库　该库中存放各种打印机的控制命令集,供打印输出模块配置程序使用。

(8)打印输出模块配置程序　该程序在系统自举时根据系统所配置的打印机,生成能适用于这种打印机的打印输出模块。

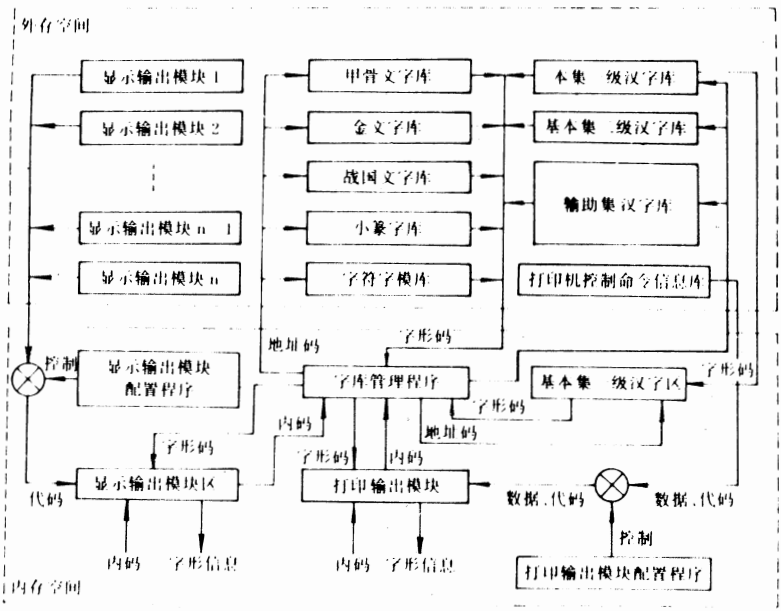


图 3-13 输出模块的结构与信息流

2. 字库结构

现代汉字库的结构十分简单，其中的字模按其对应汉字的国标码顺序排列。考虑到基本集一级汉字的使用覆盖率可达 99%以上，故把一级汉字库常驻内存，其它字库均驻在硬盘上。古汉字库均驻在硬盘上，它们中的字模按其对应古汉字之内码序排列。

为了便于访问，系统把上述字库按以下顺序编号：基本集字库、第一辅助集字库、第二辅助集字库、第三辅助集字库、第四辅助

集字库、第五辅助集字库、甲骨文字库、金文字库、战国文字库和小篆字库。它们的字库号分别定为 0～9。

3. 访问字库的算法

访问字库的算法是输出模块中最主要的算法，其目的是根据内码获得字库号和库内地址，以取得对应的字模。这就是说，我们要构造一个函数 f，使得：

$$y=f(ln,a)=f(x)$$

上式中 ln 为字库号，a 为库内地址，x 为内码。

具体的算法如下：

令内码为 $c_1c_2c_3c_4$，b_i 为 c_i 的 ASCII 码，i=1,2,3,4。

(1)根据内码首字符算得字库号 ln，计算公式为：

$$ln=((b_1-47H)-(b_1-47H)\bmod 2)/2$$

(2)计算字模的字库内序号 n，计算公式为：

$$n=((b_1-47H)\bmod 2)\times 1000+(b_2-30H)\times 100+(b_3-47H)\times 20+b_4$$

(3)计算字模的库内地址 a，计算公式为：

$$a=m\times n$$

上式中 m 为一个字模的字节数。

由于本系统的大多数字库驻留在硬盘上，所以有必要关心访问硬盘字库的速度。为了获得 较好的性能，要寻求更好的字库结构，关于这部分内容，可参阅本书第六章的有关内容。

八、系统实现技术

1. 系统内核对代码的容许度

为了实现系统对汉字信息的处理，必须使汉字内码能在系统内正常存在。但是，西文系统的内核并不容许所有的代码均能正常存在。例如，有的西文系统内常把字节的 MSB 作为特殊标识位使用。有的西文系统内始终把字符看作 ASCII 符，故常把字节的 MSB 屏蔽掉。显然，使用 MSB 作汉字标识的内码就不能在上述系

统内正常存在。这时必须对西文系统进行汉化，其具体工作反映为把用作特殊标识位的 MSB 释放出来，并去除对 MSB 的屏蔽操作，以保持 MSB 的正常值。

由此可知，全汉字集系统若采用 TTB 码作为其内码的话，则必须对 DOS 的内核进行汉化，若采用 CNCC 码作为其内码的话，则可以免去这项工作。

2. 中西文兼容的屏幕显示

显示存贮区中的字节与屏幕上的字符是一一对应的，它们之间不但内容一一对应，而且位置也是一一对应的。一个汉字的显示宽度为西文字符的两倍是最合适的，也是最自然的，那末一个汉字码所占用的显示存贮区的字节数也应为西文字符的两倍。只有作到这一点，才能实现中西文兼容的屏幕显示。CNCC 码和 TTB 码均不能直接作到这一点，因为它们的码长均不是西文字符码的两倍。为了实现上述之对应关系，必须对显示存贮区的结构进行改造。

一种有效的方法是，把显示存贮区分成两个分区，即字符分区和标志分区，前者存放显示字符之代码，后者存放显示字符之标志字节。当采用 CNCC 码作内码时，对于西文字符，字符分区中存放字符代码，标志分区中存放 00H 字节；对于汉字，字符分区中存放汉字码的两个字节，标志分区中存放汉字码的另外两个字节。这就保证了字符分区内汉字码与西文字符码 的长度之比为 2∶1。当采用 TTB 码作内码时，西文字符的处理与 CNCC 码相同；对于基本集汉字码，字符分区中存放汉字码，标志分区中存放 FFH 字节；对于其它汉字码，字符分区中存放汉字码的后两个字节，标志分区中存放汉字码的首字节。这也保证了汉字码与西文字符码在字符分区内的长度比为 2∶1。

除此之外，还要修改有关的屏幕操作。一个是字符的写入(显示)操作，要使它能把写入信息按照上述规定存入显示存贮区，并组合之。另一个是退格操作，要使它能在删除操作时，同时作废字

符分区和标志分区中的相应内容。

如果完成了以上工作，全汉字集系统就可以实现中西文兼容的显示。

3. 中西文兼容的屏幕编辑

中西文兼容的屏幕编辑是汉字系统必须具有的用户界面，我们可以在实现中西文兼容的屏幕显示的基础上来实现这一功能。屏幕编辑由两部分工作组成，即显示和编辑。我们要求显示和编辑操作必须同步，也就是说，两者操作的位置和处理量必须一致。既然汉字的显示宽度为西文字符的两倍，那末汉字的编辑处理量也应为西文字符的两倍。对于不同的内码，可以采用不同的方法来实现这一目标。

当采用 CNCC 码作内码时，我们可以专门定义一种编辑用的处理码。这种处理码对于西文字符为两字节长，其首字节为 00H，次字节为西文字符码；对于汉字，该处理码即为汉字码本身，故其为四字节长。在编辑工作区内存放的是处理码，而不是内码，显然，这时汉字的编辑处理量为西文字符的两倍。此外，我们还必须修改编辑操作的基本单位，由原来的一个字节改为两个字节。这样，就实现了编辑与显示在操作上的一致。

当采用 TTB 码作内码时，我们则采用另外一种方法。我们专门开辟一个编辑辅助区，它与编辑工作区相对应。对于西文字符，辅助区内存放 00H 字节；对于基本集汉字，辅助区内存放 FFH 字节；对于其它汉字，辅助区内存放汉字码的首字节。编辑工作区内则分别存放西文字符码、基本集汉字码，以及其它汉字码的后两个字节。显然，编辑工作区内的汉字编辑处理量为西文字符的两倍。此外，还要修改有关的编辑操作，使它们能同时对编辑工作区和编辑辅助区作相应的操作。

作到这一步，已经能实现中西文兼容的屏幕编辑。但是，在此基础上还能作进一步的优化。例如，可以对某些编辑操作进行改造，使它们只能从汉字码的边界处开始操作（如插入操作等）。还可

以使某些操作必须对一个目标整体为单位进行操作(如删除操作等)。这样可以使中西文兼容的屏幕编辑更加完美。

本节介绍的全汉字集处理系统的设计技术,可以用于多文种信息处理系统和少数民族文字信息处理系统的设计。

第四章　内存管理技术

第一节　DOS 的内存管理机制

一、总述

1. 引言

内存是计算机系统中的关键性资源，它为操作系统和用户程序所共享，能否合理和有效地对内存进行管理，会直接影响到整个计算机系统的性能。

操作系统对内存进行管理的方式有多种，有分区管理、分页管理、分段管理、分段分页管理等。DOS 是运行于微型机上的单用户操作系统，所以它的内存管理模式比不上大型操作系统那样复杂。它的内存管理模块所完成的主要功能是对内存中的用户区空间的分配和回收，它并不具备诸如存贮保护和存贮扩充之类的高级功能。

DOS 的内存管理模块采用分区管理中的可变式分区分配法，其分配策略采用三种算法，即最先适应分配算法、最佳适应分配算法和最后适应分配算法。下面对可变式分区分配法及有关分配策略作一介绍。

2. 可变式分区分配法

可变式分区分配法就是把内存区域按程序的实际需要内存量划分成若干个区，供若干个程序使用。显然，这些分区的大小是不一致的。这种分配法简单有效，比较适合于小型系统采用。

为了实现这种分配法，系统内需要建立一些数据结构来反映内存分区的使用情况，内存管理模块据此作出管理。我们可以建立两张表格，一张记录已分配区的情况，称为已分配区说明表；另一张记录未分配区的情况，称为未分配区说明表。已分配区说明表由若干项组成，对于使用的项，则每项对应于一个已分配的内存区，记录它的大小和起始地址。未分配区说明表也由若干项组成，对于使用的项，则每项对应于一个未分配的内存区，记录它的大小和起始地址。

可变式分区分配法实施分配内存区的步骤如下：

①从未分配区说明表中找一个足以满足程序要求的未分配区；若找不到，则回送有关信息；

②如果这个未分配区比程序所要求的大，则将它分为两部分，一部分成为已分配区（分给程序使用），剩下部分仍为未分配区；

③分别修改两张说明表的内容，并回送出所分配区的大小和起始地址给申请内存的程序。

可变式分区分配法实施回收内存区的步骤如下：

①检查回收的内存区是否与未分配区相邻接，如邻接的话，则把它们合并为一个连续的未分配区；

②修改两张说明表的内容。

请注意，及时合并邻接的未分配区很重要。这样作可以减少内存中的未分配区“碎片”，并有利于及时满足用户的申请要求。

3. 分配策略

实施内存分配的工作中，最主要的是从未分配区中找一个满足要求的区。但是，满足要求的区可能有多个，那么到底取哪一个？这就是分配策略。DOS 采用三种内存分配策略。对分配策略的讨论与未分配区的排列顺序有关，DOS 内的未分配区按照其起始地址递增序排列。下面介绍 DOS 采用的三种分配策略。

(1)最先适应分配法　最先适应分配法规定，选择第一个满足要求的未分配区。根据 DOS 内未分配区的排列顺序可知，这种分

配法实际上是尽可能把低地址端的未分配区分给程序使用，而尽可能保存高地址端的较大未分配区，以便有足够大的未分配区来满足较大程序的申请要求。

(2)最后适应分配法　最后适应分配法规定，选择最后一个满足要求的未分配区。根据 DOS 内未分配区的排列顺序可知，这种分配法实际上是尽可能把高地址端的未分配区分给程序使用，而尽可能保存低地址端的较大未分配区，以便有足够大的未分配区来满足较大程序的申请要求。

(3)最佳适应分配法　最佳适应分配法规定，选择满足要求的未分配区中的最小区分给程序使用，即把最接近于申请要求的未分配区分出去。这种方法可以使被分配出去的未分配区的剩余部分最小，即浪费最少。但是，这种方法也会带来一些副作用，它可能会在内存中形成许多很小的未分配区，这些区无法分配使用，从而形成浪费。

二、系统的内存布局

DOS 在实方式下工作，在这种方式下 CPU 的寻址范围为 1M 字节。但是，这 1M 字节的空间中有不少被定为系统保留空间，DOS 管理的内存空间仅为 640k 字节。下面给出 DOS 系统的内存布局(如图 4-1 所示)。

从图 4-1 可知，内存空间高端的 384k 字节为硬件系统的保留区，不属于 DOS 的管理范围。DOS 的存贮管理范围就是从 00000H～9FFFFH 的 640k 字节空间。其实这 640k 字节空间中，系统本身还要占据一部分，从低地址端开始依次为中断向量表、ROM 工作区和 DOS 常驻区。DOS 常驻区内存放 DOS 的 DOS－IOSYS、DOS－CORE、可安装设备驱动程序，以及 DOS－SHELL 的常驻内存部分。另外在 DOS 的管理空间高端还有一个 DOS 暂驻区，其中存放 DOS－SHELL 的暂驻内存部分(含 DOS 的内部命令、批处理文件的处理程序和外部命令装入程序等)。当用户程序

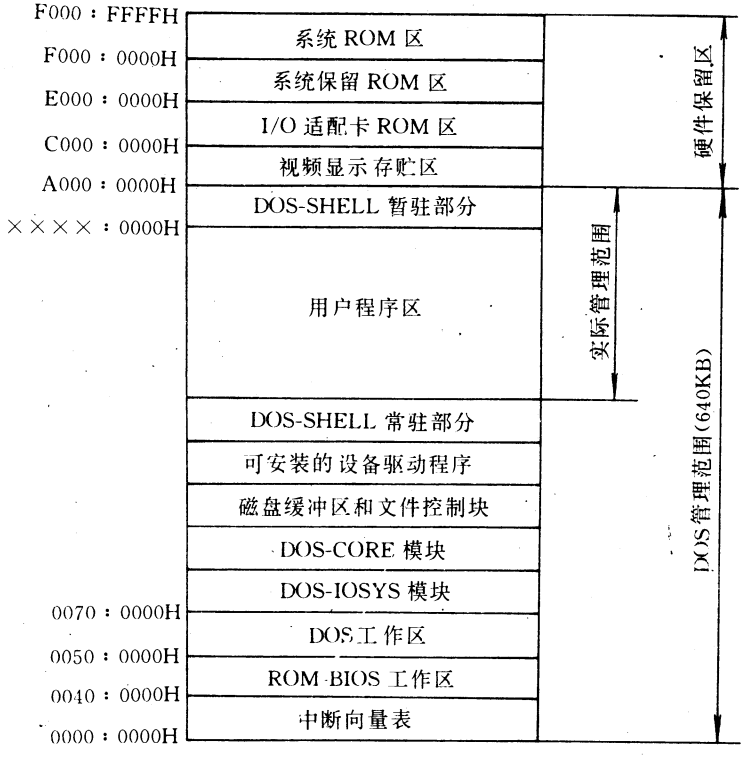


图 4-1 系统的内存布局

需要时，可以覆盖掉 DOS 暂驻区，用户程序结束后，DOS 会自动恢复该区中的内容。所以，DOS 所管理的内存空间实际上还不到 640k 字节。

三、数据结构

DOS 的内存管理模块中，并没有使用前面所介绍的那种未分

配区说明表和已分配区说明表。为了节省开销和简化算法，DOS使用了内存控制块(Memory Control Block)来记录内存区的使用情况。内存控制块位于每个内存区之首，故又称为区域头(Area Header)。图 4-2 给出了 DOS 内存管理中使用的内存区和内存控制块的结构。

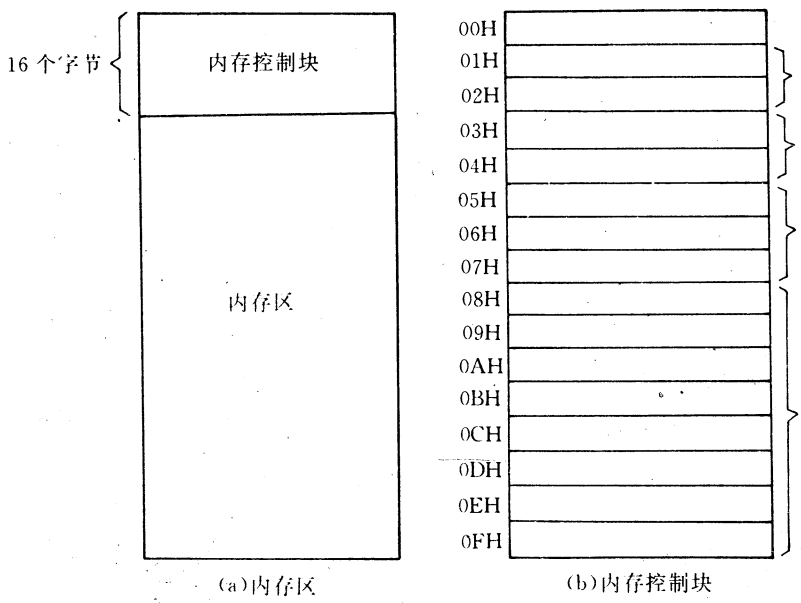


图 4-2　内存区和内存控制块

1. 内存区

在 DOS 中，内存区均由两部分组成，前 16 个字节为内存控制块(见图 4－2(a))，其中记录了该内存区的控制信息。内存控制块后为该内存区的可使用部分，可由占用该内存区的程序存放信息。内存控制块占用之空间不能被用户使用。内存区的长度是 16 字节的整数倍。所以，内存的基本分配单位不是字节，而是节(para-

graph)，1 节合 16 个字节。

2. 内存控制块

内存控制块共 16 个字节，用于记录和描述对应的内存区的情况，为 DOS 的内存空间管理提供依据。内存控制块的结构如图 4-2(b)所示。它分为标记域(tag)、内存区段址域(psp)、内存区长度域(size)、保留域和程序名域(name)。下面分别介绍这五个域。

(1)标记域　标记用于指出该内存控制块是否是内存控制块链中的最后一块。该域占用 1 个字节。其值为 4DH 时，表示该内存控制块不是链中的最后一块；其值为 5AH 时，表示该内存控制块是链中的最后一块。

(2)内存区段址域　内存区段址用于说明该内存区是否已被分配，以及内存区的起始段址。该域占用 2 个字节。其值为 0000H 时，表示内存区未被分配，是空闲区；其值不为 0000H 时，表示内存区已被分配，该域之值即为此内存区的起始段地址。

(3)内存区长度域　内存区长度用于指出该内存区的长度(以节为单位)，此长度不包括内存控制块的长度在内。该域占用 2 个字节。

(4)保留域　保留域共占用 3 个字节，该域目前未被使用，留作今后 DOS 扩展时使用。

(5)程序名域　程序名域是从 DOS4.0 版开始新增的一个域，它共占用 8 个字节。它用于记录占用该内存区的程序之名称(以 00H 字节结尾)，主要用于驻留内存程序(即 TSR 程序)。因为 TSR 程序一旦释放了其环境块，就无法确定其名称。在 DOS4.0 以前的版本中，该域归入保留域内。

3. 内存控制块链

在 DOS 内部，所有的内存控制块均连成一个链，此链称为内存控制块链。这是一个单向链，即从链首开始，可以通过间接指针 p_i 依次访问链内各个内存控制块。图 4-3 给出了内存控制块链的结构。

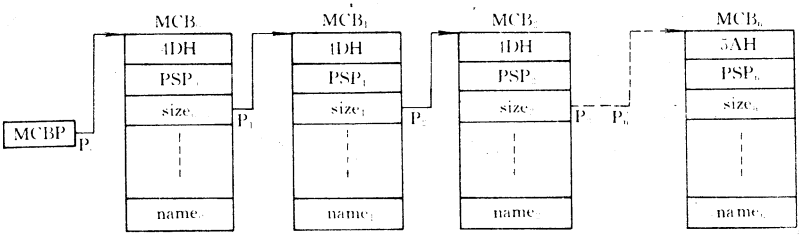


图 4-3 内存控制块链的结构

内存控制块链内仅其最后一块的标记为 5AH，其它各块的标记均为 4DH。链内各块之间的指针 p_i 并无实际记录值，而要通过计算得出，故称为隐式指针。p_i 的计算公式如下：

$$p_i = p_{i-1} + size_{i-1} + 1$$

式中 i=1,2,3…,n。p_0 为 MCBP 字单元之值，这个字单元在 DOS 内核中，它是内存控制块链的首指针。上述公式指出了，当前内存控制块的段值加上当前内存区的长度，再 加上 1，即为下一内存控制块的段值。由于内存分配的基本单位是节，所以内存控制块的地址偏移量均为 0。

我们还可以根据内存控制块的段值，用以下公式计算出其内存区的段址：

$$psp_i = p_i + 1$$

式中 i=0,1,2,…n。这就是说，内存区的段址为其内存控制块的段值加上 1。

四、内存区的分配

内存区的分配是指系统选择一个符合程序申请要求的内存区交给程序使用的过程。“分配”这一概念是针对系统而言的，这一过程如果针对用户来说，则称作内存区的申请。下面从系统和用户两个角度来分别介绍“分配”和“申请”。

1. **内存区的申请**

程序需要使用内存空间时，就向系统提出申请。程序必须给出所需内存区 的长度。系统响应后必须回答是否能满足要求，如能满足的话，还要给出所分内存区的首地址。

DOS 的 48H 号功能调用能完成以上工作，程序可以调用它来向系统申请内存区。48H 号功能调用的入口和出口参数如下：

入口参数：(BX)＝申请内存区的节数

出口参数：

成功　CF＝0

(AX)＝所分内存区的段址

失败　CF＝1

(BX)＝最大可用区的节数

(AX)＝错误代码

7：内存控制块格式错

8：内存不够

程序在调用 48H 号功能调用时，必须注意到以下两个方面：

①当 COM 文件被装入运行时，系统将当前最大的可用内存空间全部分配给它。这时如果在程序中使用 48H 号功能调用的话，则肯定会失败，因为已无内存空间可提供分配。所以，必须使用 4AH 号功能调用释放多余的内存空间后，方能调用 48H 号功能调用申请内存空间。

②当 EXE 文件被装入运行时，系统根据其文件首部 000CH～000DH 处的最大内存节数与程序本身长度之和来申请内存空间，如果不能满足(通常都不能满足)，系统就根据当前最大的可用内存空间进行申请。所以，EXE 文件也占用了当时的全部可用内存空间。因此，在 EXE 文件中也必须释放掉空余的内存空间后，才能使用 48H 号功能调用申请内存空间，否则必定会导致失败。

2. **内存区的分配**

系统提供的 48H 号功能调用的处理程序，是 DOS 的内存管

理模块的一部分。这个处理程序的执行过程就是内存区分配的过程，图 4-4 为该处理程序的流程。

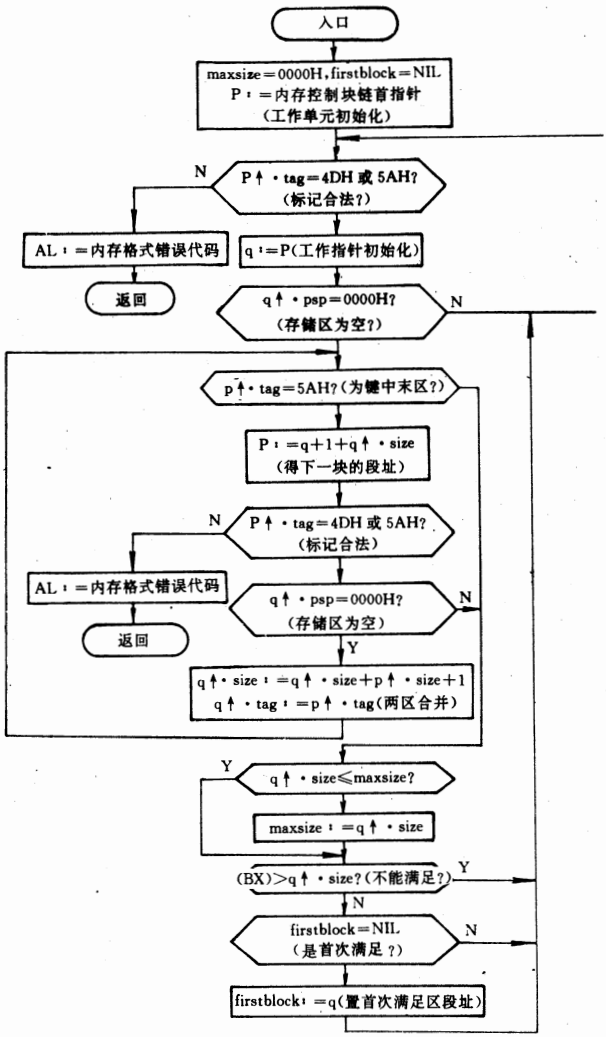
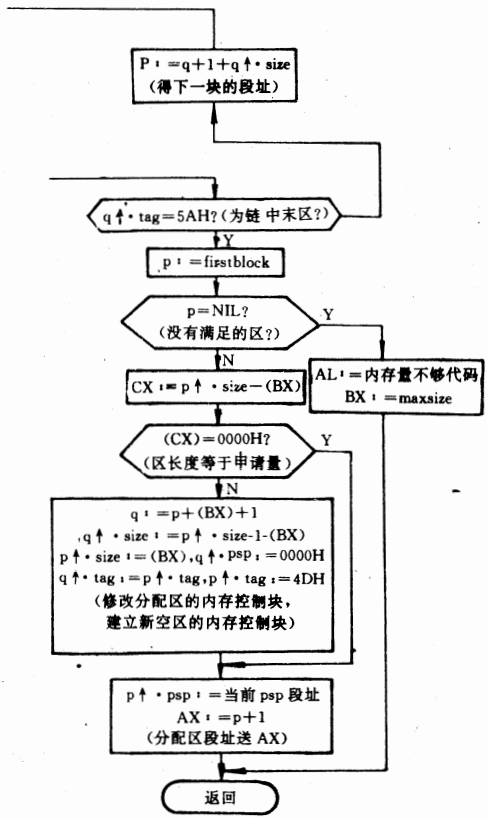


图 4-4 内存区的分配过程



续图 4-4 内存区的分配过程

图 4-4 中的 maxsize 变量用于记录最大的可用内存区长度(以节为单位),firstblock 变量用于记录第一个能满足申请容量的内存区的段址,p 和 q 均为指向内存控制块的指针变量,tag、psp 和 size 为内存控制块结构的域。图 4－3 中的分配策略以最先适应分配法为例。大家根据流程中每框内的说明信息(括号中的内容),不难读通此过程,下面仅对此过程作一个简要的总结说明。

系统首先从内存控制块链的链首扫描到链末,把邻接的未分配区合并起来,并找出最大的未分配区,把其长度记入 maxsize 变量。又找出第一个符合申请要求的空闲区,把其段址记入 firstblock 变量;如果在扫描过程中找不到能满足要求的未分配区,就把最大空闲区之长度送入 BX 寄存器,然后结束;如果在扫描过程

中找到满足要求的未分配区，当该区的长度恰好等于申请要求时，就将该区的内存控制块的 psp 域值改为 p_i+1 之值。如果找到的未分配区长度大于申请要求时，就把这个区分成两个部分，前一部分（长度为申请量）改为已分配区，后一部分（长度为原来长度减去申请量，再减去 1）为未分配区，并为其建立内存控制块。最后把找到的内存区的段址送入 AX 寄存器。

五、内存区的回收

内存区的回收是指系统接受程序用毕交还的内存区的过程。“回收”这一概念是针对系统而言的，这一过程如果针对程序来说，则称作内存区的释放。下面从系统和程序两个角度来分别介绍“回收”和“释放”。

1. 内存区的释放

程序将不用的内存空间归还给系统，就提出释放的请求。程序必须给出被释放的内存区的段址。系统响应后必须回答是否释放成功。

DOS 的 49H 号功能调用可以完成以上工作，程序可以调用它来释放内存空间。49H 号功能调用的入口和出口参数如下：

入口参数：(ES)＝释放内存区的段址

出口参数：

成功　CF＝0

失败　CF＝1

(AX)＝错误代码

7：内存控制块格式错

9：ES 为无效内存区的段址

程序在调用 49H 号功能调用时必须注意，释放的内存区应是前面曾用 48H 号功能调用申请到的内存区，否则可能会导致意料之外的系统错误。再有，利用 49H 号功能调用释放内存区，应该是全区释放，即先前用 48H 号功能调用申请到多少，现在就释放多

少。

2. 内存区的回收

系统提供的 49H 号功能调用的处理程序，是 DOS 的内存管理模块的一部分。这个处理程序的执行过程如图 4-5 所示。

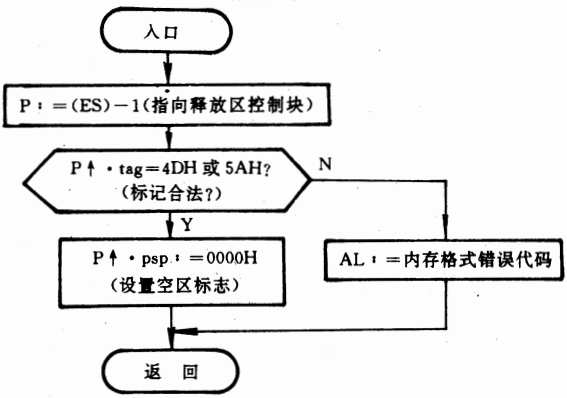


图 4-5 内存的回收过程

图 4-5 中的流程也就是内存区的回收过程，其中 p 为指向内存控制块的指针变量，该流程十分简单，它首先根据释放区的段址获得其内存控制块的段址，再把此内存控制块的 psp 域清为全 0，表示该内存区为未分配区。

六、内存区的修改

系统提供的 49H 号功能调用可以实现内存区的释放，但必须是全区释放。在某些情况下需要释放内存区的部分空间，而保留另一部分空间，故系统必须向程序提供另一种功能，这就是内存区的修改功能。内存区的修改对于系统来说，就是内存区的部分回收，而对于程序而言，则为内存区的部分释放。下面分别介绍这两个方面。

1. 内存区的部分释放

程序将申请到的一个内存区中的一部分归还系统，就可以提

出修改内存区的请求。程序必须给出修改区的段址和区中保留部分的长度。系统响应后必须回答是否修改成功。

DOS 的 4AH 号功能调用可以完成以上工作，程序可以调用它来释放一个内存区的部分空间。4AH 号功能调用的入口和出口参数如下：

入口参数：(ES)＝修改区的段址

(BX)＝修改区的保留节数

出口参数：

成功　CF＝0

失败　CF＝1

(AX)＝错误代码

7：内存控制块格式错

8：内存不够

9：ES 为无效内存区段址

程序在调用 4AH 号功能调用时必须注意，修改的内存区应该是先前曾经用 48H 号功能调用申请到的内存区，否则可能会发生不可预测的结果。

2. 内存区的部分回收

系统提供的 4AH 号功能调用的处理程序，是 DOS 的内存管理模块的一部分。这个处理程序的执行过程，就是内存区部分回收(即内存区修改)的过程，这个过程如图 4-6 所示。

图 4-6 中的 p 和 q 为指向内存控制块的指针变量。大家根据每框中的说明信息(括号中的内容)，不难读通此过程。下面仅对此过程作一个简要的总结说明。

系统首先检查被修改区的下一个内存区是否是未分配区，如果是的话，两区就合并，并继续进行下去，直到无未分配区可合并为止；如果保留的长度大于被修改区(若进行过合并，则指合并后的区)的长度，则将被修改区的节数送入 BX 寄存器后就结束。如果保留的长度不大于被修改区的长度，就修改内存区。这时，如果

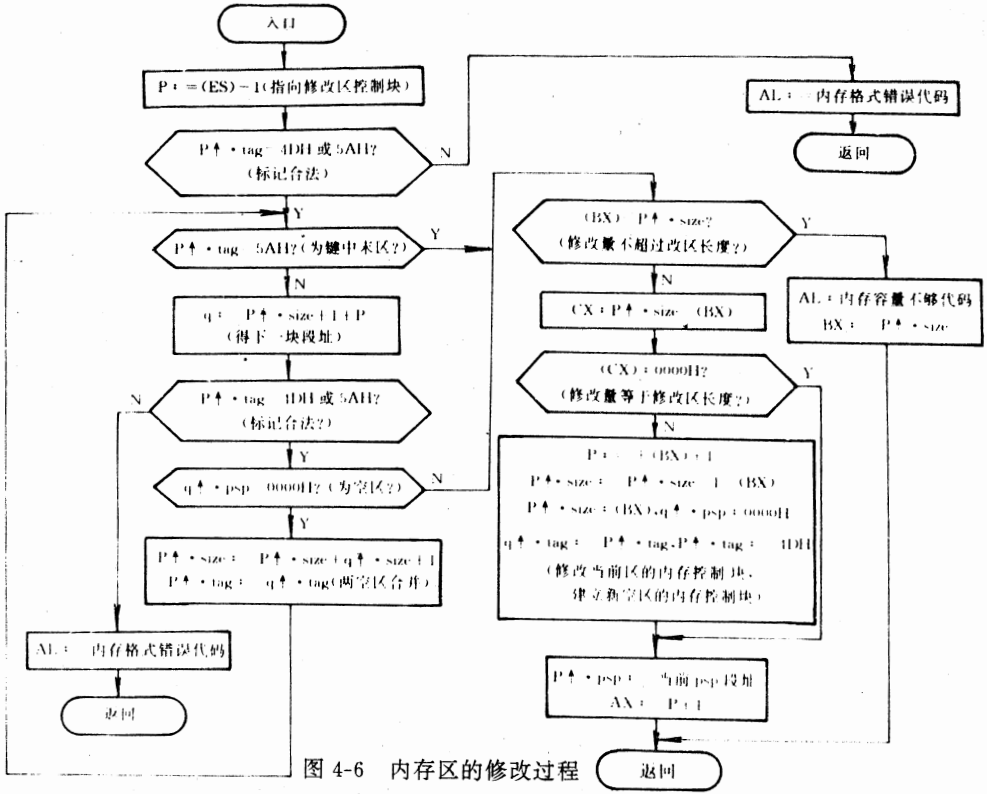


图 4-6　内存区的修改过程

保留的长度恰好等于修改区的长度，就把当前 DOS 中 psp 的段值送被修改区内存控制块的 psp 域；如果保留的长度小于被修改区的长度，就把被修改区分为两个区，前面一部分（长度为保留长度）定为已分配区，后面一部分（长度为原长度减去保留长度）定为未分配区，并为其建立内存控制块，还要把前面一部分的段址送入 AX 寄存器。

第二节　DOS 内存管理机制的优化

一、问题的提出

DOS 从其 2.0 版开始，向用户提供了申请内存区和释放内存区的功能，另外还提供了父程序加载子程序的功能。这些功能的提供，导致了 DOS 内存管理机制中不足之处的发现，主要是内存碎片问题。所谓内存碎片，是指内存空间中独立的存贮小区。

1. 碎片问题

在 DOS 的一些较复杂的应用程序中，父程序要加载多个子程序，程序会频繁地申请内存区和释放内存区，因而使内存管理模块随机地分配和回收内存区。这样，就有可能在内存空间中形成许多不连续的空闲内存区（未分配区）。当某个程序申请内存时，有可能内存中未分配区的总量能够满足申请要求，然而找不出能满足申请要求的未分配区，从而最终无法满足申请要求。这就是内存的碎片问题。显然，碎片问题影响了内存的使用效率。

一般常用移动部分内存区内容的方法来解决内存碎片问题。我们只要把内存中的已分配区向内存空间的低地址端移动，就可以使未分配区都“浮”上内存空间的高地址端，并形成一个大 的未分配区。我们把这一过程称作碎片集成（fragments compaction）。但是，碎片集成操作又会产生另一个问题——指针问题 。

2. **指针问题**

当程序申请内存时，内存管理模块执行分配职能，向申请者分配内存区。它把所分配内存区之地址返回给申请程序，这个内存区地址是一个指向该内存区的指针。申请程序(或其子程序)就以此区指针来使用该内存区。如果进行了碎片集成操作，使这个内存区的位置发生了变动，而其区指针未随之改变，显然该指针已经不再指向该区。这时，系统中就会发生紊乱，这就是指针问题。如果不解决指针问题，那末一旦进行碎片集成操作，系统就会崩溃。

二、优化方案的思想

由于 DOS 自身的规模及其支撑环境的限制，我们的优化方案不宜采用太复杂的机制，而仍然建立在分区分配法的基础上，旨在解决内存碎片问题，以提高内存空间的使用效率。所以，我们就紧紧围绕这一点来提出优化方案的思想。

1. **柄的概念与使用**

从前面的讨论可知，欲解决内存碎片问题，首先要解决指针问题。为此，我们引入了一个新的概念——柄(handle)。我们要利用柄来解决指针问题。

指针意味着一重间接，其内容为所要操作对象的地址，柄意味着二重间接，其内容为所 要操作对象的地址的地址。这就是说，柄是一个指针的指针，它指向一个指针。我们把柄所指向的指针称作主指针(master pointer)。

指针问题的实质是，指针所指向的内存区在碎片集成操作时变动了位置，而申请该内存区时，内存管理模块返回给申请者的区指针值未随之变动。由于返回的区指针在用户程序内 使用，所以在碎片集成操作后，系统欲修正其值是极其困难的，而且还须花费较大的开销。

利用柄可以圆满地解决指针问题。我们对内存管理机制作这样的优化，使得内存管理模块为申请者分配内存区时，不再向申请

者返回区指针，而是返回一个区柄，它指向一个主指针，而该主指针则指向所分配的内存区。这样，程序就以该区柄来使用这个内存区。当因进行碎片集成操作而引起此内存区移动时，内存管理模块及时修正其主指针之值，使它保持指向该内存区。为了确保在这个过程中不需要修正区柄之值，为了使内存管理模块能够方便地修改主指针之值，就必须保证主指针的地址不会因碎片集成操作而发生变化。我们可以把主指针归入系统工作区中，因为碎片集成操作不会移动系统区的位置(DOS 的内存管理模块已在内存空间的低端)，所以系统工作区的地址不会发生变化，这也就保证了主指针的地址不会发生变化。

2. 锁定的概念与使用

利用柄解决指针问题的关键在于：作碎片集成操作时及时修正被移动内存区的主指针值。这种修正当然要花费一定的时间开销。当一个程序在运行过程中，因申请内存区或加载子程序而多次引起碎片集成操作时，必须多次修正该程序所用内存区的主指针值。

为了减少为修正主指针值而付出的额外时间开销，我们给内存区定义了锁定属性。如果 某内存区被锁定，则其在碎片集成操作时不会被移动，因而就不必修正其主指针之值。例如，我们可以锁定正在执行的程序的内存区，那末即使在该程序执行过程中多次引起碎片集成操作，也不必修正主指针值。当然，我们亦可以使被锁定的内存区解除锁定(简称为解锁)。

我们引入了锁定这一概念后，可以减少系统因碎片集成操作而付出的额外时间开销。但是，我们也必须看到，锁定会给碎片集成操作的实现带来一些困难。

三、总体设计

1. 总体结构

优化后的 DOS 内存管理模块由五个例行程序组成，它们分别

是：

(1)malloc：这是分配例行程序，它实现把指定容量的内存区分配给申请者。

(2)mfree：这是回收例行程序，它实现回收由程序释放的内存区。

(3)mlock：这是锁定例行程序，它实现对指定内存区的锁定。

(4)munlock：这是解锁例行程序，它实现解除对指定内存区的锁定。

(5)compct：这是碎片集成例行程序，它把内存碎片合并成一个未分配内存区。

以上例行程序中的前面四个能被程序调用，它们在系统内作为系统功能调用的处理程序存在。最后一个例行程序 compct 只能被例行程序 malloc 调用，不能被其它任何程序调用。

2. 数据结构

优化后的 DOS 内存管理机制有三个主要特征：使用了柄，定义了锁定属性，实现了碎片集成操作。我们必须重新考虑内存管理模块的数据结构，以适应这三个特征。

我们把内存区的描述部分(即内存控制块)在物理上与内存区分离开来，存放到内存管理模块的专用区内，使它们在进行碎片集成操作时，不会随着内存区发生移动，以适合于柄的使用。我们把内存控制块的结构改为如图 4-7 所示的形式。为了在叙述上能体现出这种新的内存控制块与原内存控制块的区别，我们将把它称作 MCB。

MCB 共有 7 个字节组成，它描述了它所对应的内存区的情况。MCB 各个域的意义如下：

(1)BASE：这是段值域，它记录了 MCB 所对应内存区的起始段值，它就是这个内存区的主指针。

(2)SIZE：这是长度域，它记录了 MCB 所对应内存区的长度(以节为单位)。

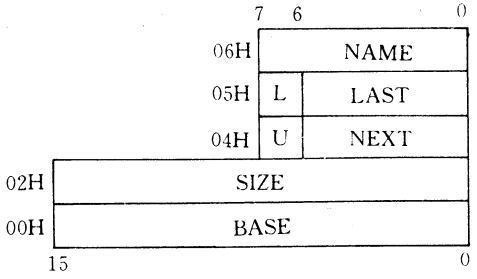


图 4-7 MCB 的结构

(3)NEXT:这是后指针域,它指向 MCB 链中后一个 MCB。

(4)LAST:这是前指针域,它指向 MCB 链中前一个 MCB。

(5)U:这是使用位,它记录了 MCB 所对应内存区的使用情况。该位为 1,表示内存区已被分配;该位为 0,表示内存区未被分配。

(6)L:这是锁定位,它记录了 MCB 所对应内存区的锁定情况。该位为 1,表示内存区被锁定;该位为 0,表示内存区未被锁定。

(7)NAME:这是程序名域,它用于记录占用该内存区的程序的代号。

这里要特别指出,LAST 和 NEXT 域中记录的内容是 MCB 的序号。由于 MCB 表中的 MCB 仅数十个,所以这两个域各占用 7 位就足够了。

所有的 MCB 均存于 MCB 表内,该表为内存管理模块的数据区,在进行碎片集成操作时不会改变位置。表中的 MCB 分属 MCB 链和空闲 MCB 链。MCB 链由已被使用的 MCB 组成,每个 MCB 对应于一个内存区,这是一个双向链,以确保分配和回收操作的迅速执行。空闲 MCB 链由未被使用的 MCB 组成,这是一个单向链。图 4-8 给出了 MCB 表的情况。

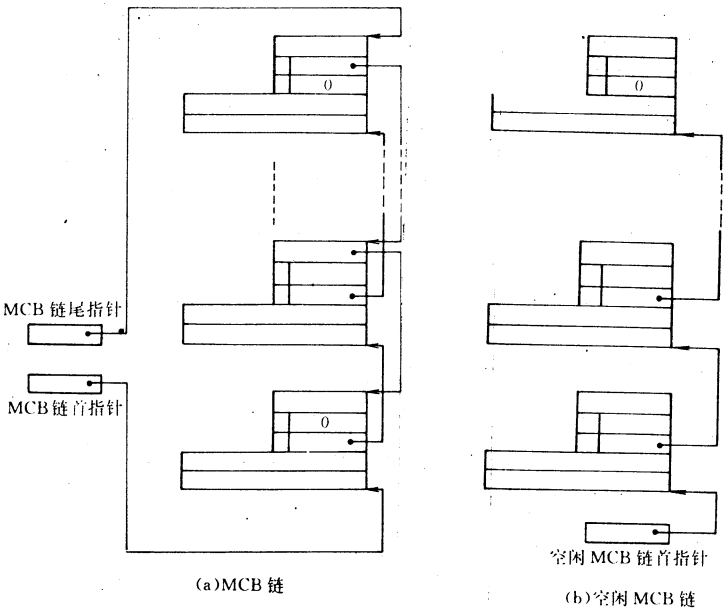


图 4-8 MCB 表的结构

图 4-8 MCB 表的结构

在 MCB 链内，MCB 按其所对应内存区的地址顺序排列。内存管理模块中设立了 MCB 链首指针和 MCB 链尾指针，以实现对 MCB 链的访问和操作。内存管理模块中还设立了空闲 MCB 链首指针，以实现对空闲 MCB 链的操作。

四、算法设计

1. 分配算法

分配例行程序 malloc 的入口参数为申请的内存区长度(以节为单位)。它的出口参数为所分配内存区的区柄，它间接指向所分配的内存区。如果当前不能满足申请要求，出口参数则给出内存空间中最大未分配区的长度。

分配例行程序 malloc 的主要任务是：根据入口参数给出的申请要求，按照一定的策略找出一个能符合申请要求的内存区，然后调整 MCB 链和有关 MCB 参数，把该内存区的区柄返回给申请者。为了简单和有效，我们在分配算法中采用“最先适应”原则来分配内存区。图 4-9 为内存分配算法的流程。

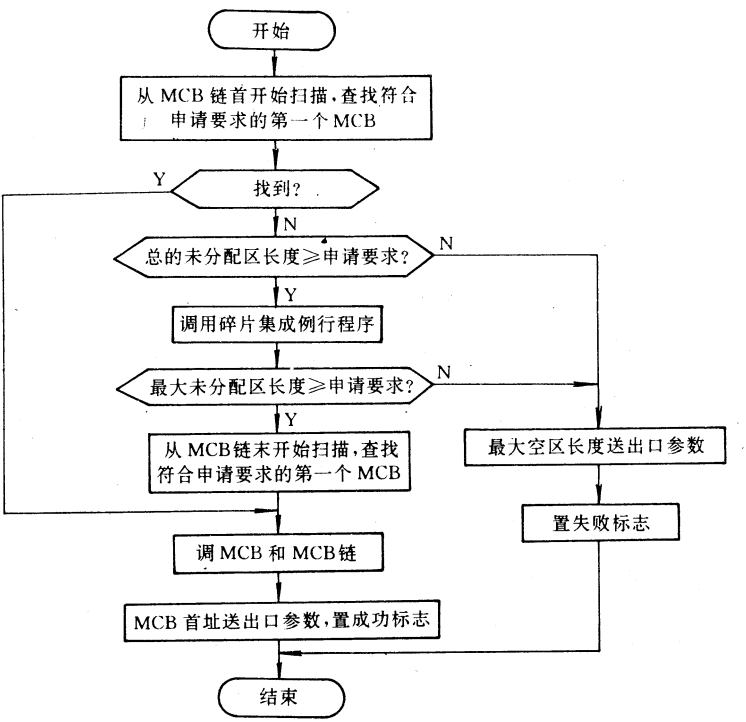


图 4-9 分配算法的流程

由于碎片集成操作要花费一定的时间开销，所以在分配算法中决不轻易作此操作，而必须在非进行不可时才进行碎片集成操作。碎片集成例行程序 compct 执行完后，会返回当时最大的未分配区的长度。碎片集成操作后，未分配区上浮到内存空间高端，所

以这时从 MCB 链末开始反向扫描，可以很快找到适合的内存区。由于 MCB 链是双向链，故有条件这样作，以加快分配的完成。

在找到符合申请要求的内存区后，则须对 MCB 链和 MCB 进行调整。如果该内存区的长度正好等于申请要求，则把该内存区的 U 位置位即可。如果该内存区的长度大于申请要求，则把该内存区的 MCB 中的 U 位置位，并把申请要求的节数写入 SIZE 域，这等于把此内存区的前面一部分分配出去了。然后从空闲 MCB 链上摘下一个 MCB，把剩余部分作为一个新的未分配区，把它的起始段址和长度填入这个 MCB 的相应域，再把这个 MCB 连入 MCB 链内。显然这时必须设置 NEXT 域和 LAST 域为相应值。

2. 回收算法

回收例行程序 mfree 的入口参数为被释放内存区的区柄，它指向被释放内存区的起始地址。回收例行程序 mfree 的主要任务是根据入口参数给出的区柄，把它所对应的内存区回收到内存空间的未分配区集内。回收算法的步骤如下：

(1)根据入口参数给出的区柄得到被释放区的 MCB；

(2)判别此 MCB 的 L 位是否为 0，若不为 0 则置出错标志，并结束；

(3)判别此 MCB 的 LAST 域和 NEXT 域指向的相邻内存区是否为未分配区，若为未分配区则合并之，否则把 MCB 的 U 位复位。

从上述算法可知，被锁定的内存区必须先解锁后才能被回收，否则不能回收。所以，内存区的锁定属性还能起一定的保护作用。

3. 锁定和解锁算法

锁定例行程序 mlock 与解锁例行程序 munloc 的入口参数均为被操作内存区的区柄。这两个例行程序的算法极其简单。先根据入口参数给出的区柄得到相应的 MCB，然后给此 MCB 的 L 位设置相应值(锁定置为 1，解锁清为 0)即可。

五、小结

以上提出的 DOS 内存管理机制的优化方案仍基于内存管理的分区分配法，由原来的分区静态分配法上升到分区动态分配法。我们认为，根据 DOS 的规模、支撑环境和使用对象，DOS 的内存管理采用分区分配法是合适的。当然，还有不少优于分区分配法的内存管理方法，如果把它们用于 DOS，则要花费较大开销，另外还要对 DOS 作较大的改动。相比之下，以上提出的优化方案是可取的。

第三节　DOS 虚存管理的实现

一、引言

1. 问题的提出

DOS 采用可变式分区静态分配法来分配内存资源，这种内存管理机制完全可以满足单任务处理环境的要求，但是对于多任务处理环境的要求就显得有些不足。上一节 中提出了采用柄和碎片集成技术，使 DOS 能以可变式分区动态分配法来分配内存空间的优化方案，从而提高了内存利用率。但是，这种方法未能解决 DOS 作为多任务处理环境的内存空间不足问题 。

DOS 工作于实方式下，其管理的内存空间不能超出 1M 字节，即使目前的微机系统大多配置有 1M 字节以上的内存，但是 DOS 的内存管理模块也无法充分利用这些内存。本节在上节提出的优化方案的基础上，进而提出一种 DOS 的虚存管理方案，从而可以解决 DOS 进行多任务处理中的内存不足问题。

2. 虚存和对换

为了便于下面的叙述和讨论，有必要先介绍虚存和对换两个概念，这两个概念在下面要多次用到。

(1) 虚存 虚存是虚拟存贮器的简称，它是内存与辅存(辅助存贮器的简称)的合成。它向用户提供了把辅存作为内存使用的手段，从而使用户程序可使用的内存空间总和能超出系统的实际内存量。虚存技术已被广泛用于操作系统。为了使 DOS 的内存管理机制能符合多任务处理环境的要求，我们认为实现 DOS 的虚存管理是一种简单有效的方法。

(2)对换 对换(swapping)是指内存和辅存之间信息的交换，它是实现虚存管理的基本操作。在实现虚存管理后，用户程序或数据的一部分会被存放到辅存，当要使用到这部分内容时，必须把它们换入(swapping-in)至内存。相反，当内存空间不够用时，则要把一部分当前不用的用户程序或数据换出(swapping-out)至辅存。这种换入和换出统称为对换。

3. EMS 和 EMS 存贮器

EMS(Expanded Memory Specification)是目前最常用的扩充存贮器规范，它由 Lotus、Intel 和 Microsoft 公司共同制定，故又称为 LIM EMS。这种规范提出了用户使用扩充存贮器的界面和系统实现扩充存贮器的方法，使 DOS 的用户有可能在实方式下使用较大的内存空间。目前，一些 DOS 的较新版本都配有 EMS 驱动程序，象 EMM386. SYS 和 XMA2EMS. SYS 等。这些驱动程序与扩充存贮器适配器相结合，就能实现对扩充存贮器的访问。值得注意的是，基于 80286 和 80386 的系统都有扩展存贮器(Extended memory)，只有在保护方式下才能访问扩展存贮器。但是，EMS 驱动程序可以把这些扩展存贮器作为 EMS 扩充存贮器来进行访问。所以，我们把通过 EMS 驱动程序可以访问到的扩展存贮器或扩充存贮器均称为 EMS 存贮器。

EMS 把 EMS 存贮器分为若干页，每个页为 16k 字节。当要访问某一页时，就把该页映射到常规存贮器(conventional memory)空间的指定区域，这一区域称为页框。通过对这一区域的操作，就实现对 EMS 存贮器内指定页的访问。EMS 向用户提供了查询

EMS、申请 EMS 存贮器空间、释放 EMS 存贮器空间和映射页到页框等功能。这此功能通常通过软中断向用户提供服务。

我们必须注意到，DOS 并没有把 EMS 存贮器纳入其内存管理的范围，而是让用户自行使用它们。这就是说，即使在配有 EMS 驱动程序和 EMS 存贮器的系统中，DOS 的内存管理范围仍是常规内存空间。

二、总体设计

1. 设计目标

DOS 的虚存管理就是要实现用辅存仿真内存，获得较大的用户可用内存空间，以满足多任务处理和运行大型程序的要求。但是，由于 DOS 只是一个微机操作系统，其规模较小，所以其虚存管理部分的开销不能太大，必须与整个系统的规模匹配。因此，我们应该尽可能在 DOS 原来的内存管理机制的基础上对其进行扩充，来实现 DOS 的虚存管理功能。

2. 设计思想

虚存的实现需要辅存的支持，在基于 80286 和 80386 的系统内，都配置有 EMS 存贮器，我们可以把 EMS 存贮器作为辅存，从而使系统的存贮总量为常规内存和 EMS 存贮器容量之和。这样，基于 80286 和 80386 的系统虚存容量最多可以分别达到 16M 字节和 4G 字节。另外，在没有配置 EMS 存贮器的系统中，我们可以利用硬盘作为辅存，其实我们只是把硬盘的一部分存贮空间(定义为一个专用文件)作为辅存。

当用户向系统申请内存空间，而当前内存空间不能满足申请要求时，则由虚存管理程序把当前暂不使用的一个或数个内存区的内容对换到辅存，腾出足够大的内存空间，以满足当前的请求。一旦需要用到被对换到辅存的内容时，则要把它们再换入内存。这样，面向用户的是一个容量大于实际内存的虚存。显然，在虚存管理中对换是不可缺少的。如果用硬盘作辅存，那么对换就是内存和

外存之间的信息交换，这不会有问题。如果用 EMS 存贮器作辅存，那么对换就是常规存贮器和 EMS 存贮器之间的信息交换，这就要利用 EMS 驱动程序来完成对换。

除此之外，我们还要设计分配算法、淘汰算法和对换算法。分配算法确定把哪个内存空区分给用户，淘汰算法确定把哪个内存区的内容对换至辅存，对换算法实现内存与辅存间的信息交换。另外，我们还要设计虚存管理部分的数据结构，用于记录虚存管理所必需的信息，为虚存管理的实现提供基础。

三、数据结构的设计

1. **内存控制块** MCB

为了遵循上面所提出的设计目标，虚存管理用的数据结构应尽可能以 DOS 原有的有关数据结构为基础，所以我们仍把内存控制块 MCB 作为虚存管理的主数据结构，用来记录各内存区的情况。由于虚存管理的要求，我们要对原来的 MCB 作一些扩充，由原来的 5 个域扩充成 8 个域。但是，MCB 的长度不变，仍为 16 个字节。图 4-10 给出了 MCB 扩充后的结构。

下面描述 MCB 各个域的意义。

(1)tag 域　这个域共 4 位，用于指出本 MCB 在 MCB 链中的位置。其值为 0H 时，表示本 MCB 为链中的首块；其值为 FH 时，表示本 MCB 为链中的末块；否则，表示本 MCB 在链中既不是首块也不是末块。

(2)mark 域　这个域共 2 位，用于指出本 MCB 所对应区之内容存放的介质。其值为 00 时，表示该区内容在内存中；其值为 01 时，表示该区内容在 EMS 存贮器中；其值为 10 时，表示该区内容在硬盘上。

(3)lock 域　这个域共 1 位，用于指出该内存区是否被锁定。其值为 1 表示内存区被锁定；其值为 0 表示内存区未被锁定。

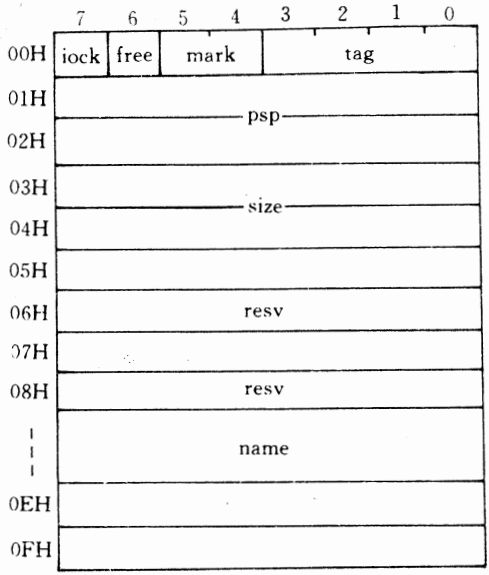


图 4-10　MCB 扩展后的结构

(4)free 域　这个域共 1 位,用于指出该内存区是否被分配。其值为 1 表示内存区未被分配;其值为 0 表示内存区已被分配。

(5)psp 域　这个域共 2 个字节,用于指出内存区的起始段地址。

(6)size 域　这个域共 2 个字节,用于指出内存区的长度(以节为单位),此长度不包括 MCB 在内。

(7)resv 域　这个域共 3 个字节,这是系统保留域,目前未被使用,留给今后扩展时用。

(8)name 域　这个域共 8 个字节,用于指出占用该内存区的程序名。要求程序名以 00H 字节结尾。

由此可见,现在的 MCB 与原来的 MCB 相比,其变化不是太大,仅增加了 mark、free 和 lock 三个域,MCB 的总长度未变,并且

MCB 中的有效信息长度亦未变。

2. MCB 链

系统内的所有 MCB 组成一个双向链，这些 MCB 按其对应内存区的起始值地址(即 psp 域之值)以递增序排列。图 4-11 给出了 MCB 链的结构。

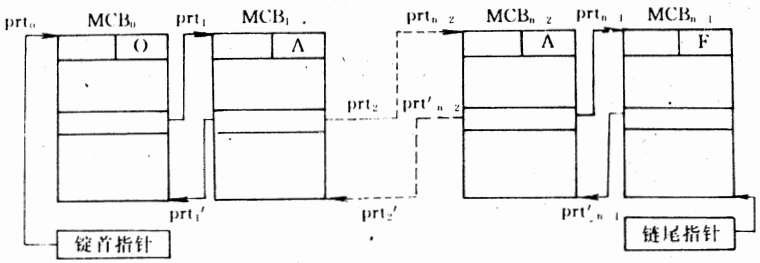


图 4-11 MCB 链的结构

图 4-11 中的 prt_i 和 prt_i' 分别为 MCB 链的正向和负向指针，它们均为隐式指针，即它们之值须经过计算得到。prt_i 和 prt_i' 的计算公式如下：

$$prt_i = prt_{i-1} + size_{i-1} + 1$$

$$prt_i' = prt_{i+1}' - size_i - 1$$

其中 i=1,2,3…,n-1。prt_0 和 prt_n' 分别为 MCB 链的首指针和尾指针之值。

我们还可以根据 prt_i 和 prt_i' 计算出内存区的起始段址，计算公式如下：

$$prp_i = prt_i + 1$$

$$psp_i = prt_{i+1}' - size_i$$

3. **换出表** SOT

当某个 MCB 所对应的内存区被对换出内存后，该内存区将被另作它用，其 MCB 之值亦将改变，故在此之前必须将此 MCB 各域之值保存起来，以便在其被对换入内存时恢复。为此，我们在系统中设立了换出表 SOT，该表用于保存被对换出内存之 MCB 的值。SOT 由若干个表项组成，每个表项用于存放 MCB 中主要域的内容。

四、系统实现

1. **用** EMS **存贮器作辅存**

既然用 EMS 存贮器作为辅存，那么就可能有多个内存区的内容被调入 EMS 存贮器，因此也存在 EMS 存贮器空间的分配问题。由于进入辅存的内容仅是作临时存放而已，并不会在辅存中把它们作为执行代码来运行，因此 EMS 存贮器空间的分配就比较简单。我们可以采用定长分区分配法来管理 EMS 存贮器空间。

EMS 把 EMS 存贮器分为 16k 字节长的若干页，若以页作为 EMS 存贮器的基本分配单位，则会造成较大的浪费。所以，我们采取把每页分为 64 个块，每块为 256 个字节，以块作为 EMS 存贮器的基本分配单位。被对换到 EMS 存贮器的内存区内容可以占用若干个块，但这些块不一定是连续的，故必须实现这些块之间的连接。

我们仿照 DOS 的文件定位表 FAT，建立一张 EMS 存贮器定位表 EMAT，以实现 EMS 存贮器中相应块之间的连接。EMAT 由表项组成，每个表项为 2 个字节，每个表项与一个块相对应。EMAT 象 FAT 那样，把一个内存区的内容在 EMS 存贮器中所占用的块连接成一个单向链。在这个链中，每个表项(除链内末项外)的内容为其下一块所在页的页号和页内块号。链内末项的内容总为 FFFFH。当表项的内容为 0000H 时，表示其对应的块为空块。EMAT 的结构如图 4-12 所示。

由于考虑到 EMS 对 EMS 存贮器的访问特点(把 EMS 存贮器内的连续几页映射入页框),所以 EMAT 不是连续存放在 EMS 存贮器中的。我们把 EMS 存贮器中每个偶数号页(即第 0、2、4、…页)内的第 0 块用于存放 EMAT。这样,第 2n 页的第 0 块内存放

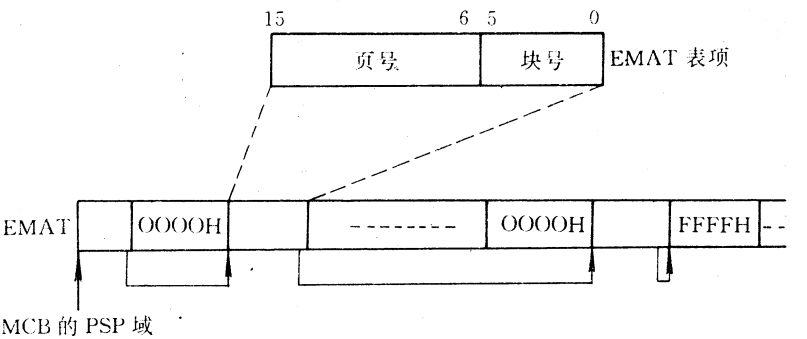


图 4-12 MEAT 的结构

的 EMAT 表项对应于第 2n 页和第 2n+1 页中的块。显然,每个偶数号页中只有 63 块可用于存放对换内容,而每个奇数号页中有 64 块可用于存放对换内容。

另外,当一个内存区的内容被对换到 EMS 存贮器后,其对应的 MCB 的 psp 域之值应改为该区内容在 EMS 存贮器中所占第一块的页号和页内块号。这样,当需要把该区内容对换入内存时,则可根据 SOT 内相应 MCB 的 psp 域和 EMAT 有关表项之值,获得该区所有的内容。

2. 用硬盘作辅存

在没有配置 EMS 存贮器的系统中,需要用硬盘作辅存。由于硬盘在系统中主要被用作文件存贮器,因此只能用其一部分空间作为辅存。我们把这部分空间定义为一个专用文件,称为对换文

件，其长度应根据具体需要来确定。我们还要考虑对换文件空间的管理问题 。

我们可以参照对 EMS 存贮器空间的管理方法来实现对换文件空间的管理。可以把对换文件分成若干个定长的记录，对换文件空间的基本分配单位是记录。鉴于硬盘的访问速度较 EMS 存贮器慢，但其空间要较 EMS 存贮器富裕，所以我们确定记录的长度为 2048 个字节。被对换到对换文件的内存区内容可以占用多个记录，这些记录不一定是连续的，故必须实现这些记录之间的连接。

我们在系统内建立了一张记录使用表 RUT，该表用于记载每个记录的状态。RUT 由字节串组成，每一位表示一个记录的状态，其值为 0 表示记录空闲，其值为 1 表示记录已被使用。另外，每个记录内的第 0 和第 1 个字节用作指针，它指向链内下一个记录(即其值为下一记录之号码)，链内最后一个记录的指针值为 FFFFH。图 4-13 给出了对换文件的结构。

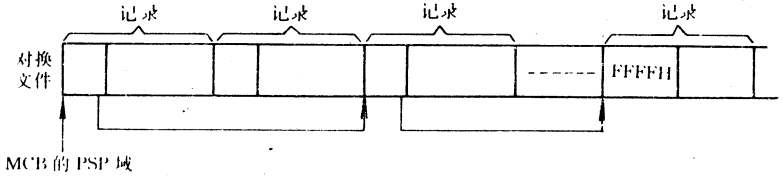


图 4-13 对换文件的结构

从图 4-13 可知，每个记录内的开头两个字节为系统所用，另外 2046 个字节用于存放内存区的内容。系统从 RUT 中可以知道对换文件中各个记录的使用情况。

当一个内存区的内容被换入对换文件后，其对应的 MCB 的 psp 域之值应改为该区内容在对换文件中所占首记录的记录号。这样，当需要把该区内容对换入内存时，则可根据 MCB 的 psp 域和有关记录内的指针之值，获得该区的所有内容。

五、算法设计

内存分区分配算法有多种，现考虑到 DOS 是一个微机操作系统，其内存管理模块不宜太复杂，所以我们仍选用最先适应分配算法，即从 MCB 链首开始向后扫描，把第一个能满足用户申请量的未分配区分配给申请者。

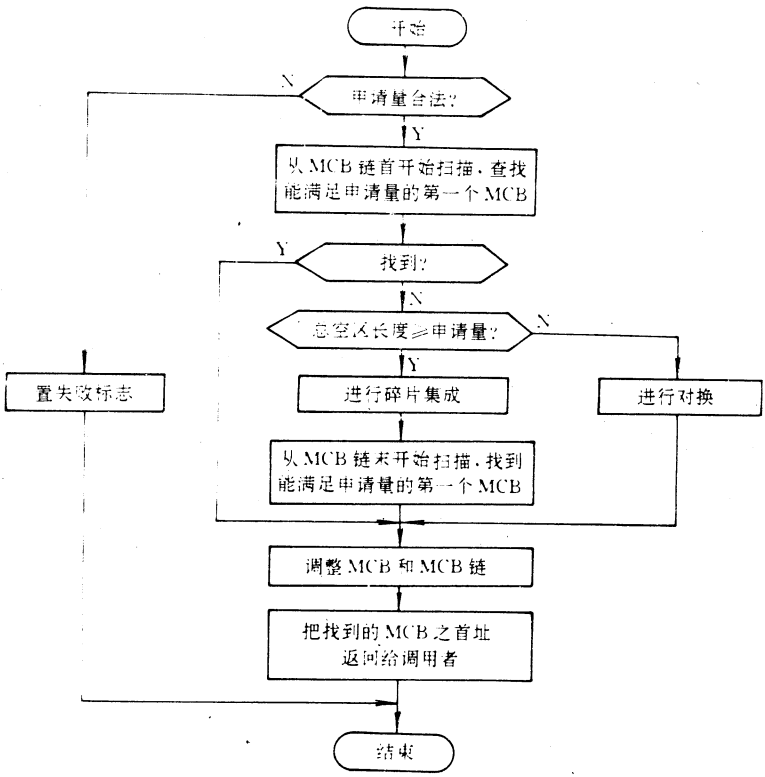


图 4-14　内存分配算法的流程

若从链首至链末找不到能满足申请量的未分配区，但内存中的未分配区总量能满足申请要求，则进行碎片集成操作，形成较大的未

分配区分配给申请者。若内存中的未分配区总量不能满足申请要求，则要进行对换操作，把内存中一个或多个区的内容对换到辅存，腾出能满足申请要求的内存未分配区。图 4-14 给出了内 存分配算法的流程。

由于碎片集成操作要移动内存中的内容，需要较大的时间开销，故多作碎片集成操作会影响系统的效率。因此，分配算法中决不能轻易作此操作，必须在非作不可时才进行碎片集成操作。碎片集成操作完成后，就在内存高地址端形成一个较大的未分配区。这时从 MCB 链末反向扫描，即可找到合适的未分配区。这一点也正是我们把 MCB 链设计为双向链的原因。

2. 淘汰算法

淘汰算法的关键是确定哪个或哪几个内存区被对换至辅存。常用的淘汰算法有 LRU 算法、LFU 算法和 FIFO 算法等，这些算法各有所长。根据前面提出的设计目标，我 们设计了一种极其简单的淘汰算法——首次适合淘汰法，这种算法能够符合 DOS 规模较小的特点。

首次适合淘汰算法的具体作法是，把 MCB 链中开头的一个或数个容量和能满足申请量的内存区对换至辅存，若其中有空区，则空区内容不必换出。这种算法显然很简单，不需要额外的数据结构支持。另外，这种算法还有一个优点，那就是它在完成对换后，向申请者提供的就是一个连续空区，不必再作碎片集成操作，这是因为它换出的内存区总是相邻的。然而，若采用其它淘汰算法的话，被换出的内存区就不一定是相邻的，故往往要经过碎片集成操作后才能成为一个连续空区。

3. 对换算法

对换算法完成内存和辅存之间的信息交换，因此它应包括换出算法和换入算法，下面分别介绍之。

(1)换出算法　换出算法实现把指定内存区的内容写入辅存，并释放该内存区。换出算法的主要步骤如下：

①根据内存区的长度，申请足够的辅存块(记录)；

②把内存区内容写入辅存块(记录)；

③把内存区的 MCB 写入换出表 SOT，并把其 psp 域改为指向相应的 EMAT 项(或对换文件记录)；

④把内存区的 MCB 之 free 域置为 1，从而释放了内存区。

(2)换入算法　换入算法实现把辅存中指定区域的内容写入指定内存区，并释放辅存区域。换入算法的主要步骤如下：

①在换出表 SOT 中取得换入内容对应的 MCB，把有关内容写入指定内存区的 MCB；

②根据 SOT 中 MCB 的 psp 域取得辅存中的换入内容，把它们写入指定内存区；

③修改 EMAT 表项(或记录使用表 RUT)，释放辅存中的相应块(或记录)。

六、进一步讨论

以上虚存管理方案的实现，较大地增加了系统的存贮空间，提高了系统对 多任务处理和大型程序的支撑能力。同时我们也要看到，由于采用了简单的首次适合淘汰算法来确定被换出的内存区，因此系统中存在发生“抖动”(thrashing)的潜在因素，即有可能发生某一部分内容频繁地被换出和换入的情况，从而降低了系统效率。为了改善这种状况，我们可以利用 MCB 中的 lock 域，把当前较频繁使用内容的内存区锁定(详情请参阅前一节)。被锁定的内存区不能被换出至辅存，从而大大减少了发生“抖动”的可能性。

另外，在一些 EMS 存贮器容量不太大的系统中，若用 EMS 存贮器作为辅存的话，有时会感到容量不足。这时，可以考虑用硬盘作为二级辅存，即被换出的内存区内容先存至 EMS 存贮器，当 EMS 存贮器存满后，则可存至硬盘(对换文件)中。这样可以使系统拥有足够大的辅存容量。在这种情况下，既要使用 EMAT，又要使用 RUT。

第五章　汉字输入处理

第一节　词输入处理技术

一、概述

随着办公自动化的普及和推广，越来越多的文件、档案和资料需要用计算机来处理。这种处理的第一步，就是要把由汉字组成的汉字信息输入到计算机内，即汉字信息的输入。汉字信息的输入效率直接关系到办公自动化系统的效率。

汉字信息的输入速度一直是汉字信息处理中的“瓶颈”问题，至今仍未彻底解决。这个“瓶颈”问题包含两个方面：一是汉字的输入速度受到很大限制。以典型的形码和音码而言，以字为单位进行输入，其速度有一个极限，而这个极限离人们期望的要求又太远。然而，汉字的字形识别输入和语音识别输入两项技术又未能真正达到实用水平，目前汉字信息的输入仍以编码输入为主。另一是输入方式和思维的不一致性。在信息处理上，需要输入计算机的内容往往是人们正在思考和创作的东西。然而汉字输入偏偏要求人们考虑怎样拆，怎样编，怎样选择，从而干扰正在进行的思考。要解决这个“瓶颈”问题，就得从以上两方面着手，其中一个解决途径是，不再把字符看成孤立的元素，满足于字符处理水平，而把字符看成是构词的元素，建立汉语词库，把词作为输入和处理的基本单元，以追求把高速度和思维的经济性结合起来的目标。

人们注意到，汉语的特点是以词和句为其语义的基本单元，而

句子又是由词和少量单字组成的。近年来，人们根据汉语的这个特点提出了词输入方案，它以词为基本的输入单位，这是一条很有前途的汉字信息输入途径。在单字输入方式下，每输入一个输入码，只意味着输入一个汉字；而在词输入方式下，每输入一个输入码，则意味着输入一个词（一般由多个汉字组成）。显然，在击键次数相同的前提下，以词为单位输入的信息量要远大于以字为单位输入的信息量。实践证明，采用词输入方式，可以大大提高汉字信息的输入速度，并能有效地减小汉字信息输入过程中的出错机率。它对解决上述的“瓶颈”问题具有相当大的作用。因此，词输入处理技术获得了较快的发展，它在汉字信息处理中的地位也越来越重要。

二、词汇量和词库

1. 词汇量问题

词输入方式是以词为单位来输入汉字信息的，所以词汇量问题应该引为一个值得注意的问题。据统计，汉语词总量可达100万左右。若把这么多的词汇全部收入计算机，显然是不切合实际的。其实，在人们日常生活中，在一般的报刊文章中，用得较多和较频繁的基本词汇是不多的。经粗略统计，一本具有2万基本词汇的词典，可以使一般文章中95%以上的词都能从中找到。另外，词是有面向性的，各行业、各种专业都有各自的基本词汇。如果我们将词分门别类，首先建立一个最基本的核心词汇集，再在这个核心词汇集中有针对性地加入一些专业词，组成面向有关专业和行业的词库，使用时根据不同的专业选用相应的词库。这样建立一个具有15000个词的词库，有可能使文章中95%以上的词都能在词库中找到。

那末文章中剩下的5%左右的词如何处理呢？这些词使用得不多，但是其数量却十分庞大，所以不能把它们收入词库。显然这些词大部分属口语中不常用的词，称之为书面词。我们可以用两种方法来处理这些书面词，一种方法是用若干个基本词合成。例如

“主旨”这个书面词，可以用基本词“主要”和“旨意”中的第一个字合成。这就要求计算机词处理部分具有“拆字”的功能。另一种方法是用输入若干个单字来合成书面词。例如“欣悉”这个书面词，可以分别输入“欣”字和“悉”字来合成。这就要求计算机词处理部分能支持单字输入。

2. 词库设计观点

为实现词的输入，必须建立词输入码表，这种表通常被称为词库。词库的设计有两种观点：一种是在广泛收集词汇的基础上，建立固定的有限词库；另一种是在汉字信息处理的过程中，利用人工分词逐步建立词库，称之为扩展式词库。这两种观点各有所长。为了便于比较，我们先引入两个概念，即词库的覆盖率和使用率。我们定义词库的覆盖率为汉字信息处理过程中被处理对象使用词库的词数与被处理对象实际包含的总词数之比。显然，词库的覆盖率越高，一般来说词库中的词数应越多。我们定义词库的利用率是词库中使用过的词数与词库中总词数之比。一种好的词库设计方法应该具有高的覆盖率和利用率。

(1)固定有限词库与扩展式词库　固定有限词库为收入主观认为是常用的词和可能用的词，由于预先收入的词无法预测使用对象的内容，故常会出现漏词现象。为了弥补固定有限词库的这一缺陷，就要设法提高词库的覆盖率，从而不得不更广泛地收词。这样作又导致词库利用率大大下降，会出现不少“死词”，从而影响对词库的访问速度。

扩展式词库的最大优点是覆盖率高，利用率也高。词库中所有的词均是使用对象使用过的。如果人工分词合理，能将使用对象所使用的词全部收集下来，处理过程中可避免漏词现象，也不存在“死词”。然而，采用该方法建立词库的初期，由于无词可寻，必须人工分选词，这样会大大降低处理速度，并且会影响词汇标准的唯一性。

(2)动态词库　由动态设置方式建立的词库称为动态词库，这

种词库的内容随处理对象的内容而变化，提高了覆盖率和利用率，同时避免因提高覆盖率而使利用率下降。

动态词库的设置可分以下三个步骤：

①建立雏型词库，其选词方法类同固定有限式词库，但是内容的广泛度要远比其小，不追求雏型词库的完善程度。一般可收词2000～3000个，使词库的覆盖率达75%左右。由此可见，雏型词库的建立可节省许多建库时间，提高词库的建造效率；

②对词库进行扩充，当被处理对象的数量增多，词库就不断扩充新的内容。扩充的方法类似于扩展式词库的建立方法。所不同的是，可以适当减少分词现象的数量，即对处理对象采取抽样方法。词库扩充是为了追求其完备度，提高覆盖率。

③词库删词，经一定周期的处理，雏型词库中可能会出现一些“死词”。另外，词库的扩充可能收入一些不必要的词和使用频率极低的词，这些词使得词库迅速膨胀。为了提高动态词库的利用率，降低其查找词汇的复杂性，随时进行删词是必要的。

三、词输入码

要实现词输入，首先要设计出词输入码，它又简称为词码。一种好的词码方案应该能符合以下条件：规则简单、码长较短、重码率较低和编码空间足够大。只有符合这些条件后，词码才能为用户乐意接受，才能在汉字信息输入过程中发挥其效率。另外，词码的设计与参加编码的总词量有关。由工程实用词库国家标准研制组主持制定的《信息处理用现代汉语五千词表》中共收词5633个，其中双字词4441个，三字以上词1192个。经实际使用证明，该词库只能作为基本词库，在实际应用中还要扩充一些日常用词，以及增加相当数量的专业或行业用词。所以，我们应把参加编码的总词数定为15000～30000。

目前已经推出的词码方案有不少，其中有不少是基于汉语拼音的。有一种比较典型的、已为大家普遍接受的词码方案，不妨称

它为常规词码，其码长为4，词码符均为字母。它的编码规则为：对于二字词其码依次由词内第一个字的声母、韵母和第二个字的声母、韵母之首字母组成（即声、韵、声、韵）；对于三字词，其码依次由词内第一个字的声母、第二个字的声母和第三个字的声母、韵母之首字母组成（即声、声、声、韵）；对于四字词及四字以上词，其码依次由词内第一个字的声母、第二个字的声母、第三个字的声母和最后一个字的声母之首字母组成（即声、声、声、声）

另外要说明一点，当要取某字的声母，而该字的拼音中又没有声母时（比如，艾：ài），则取其韵母的首字母代替之。这种常规词码能基本符合上面提出的要求，而且它是等长码，故可简化词处理程序的编制。

如果仅对《信息处理用现代汉语五千词表》中的词进行编码，则可对常规词码进行简化，从而形成一种简化词码。简化词码的码长为3，词码符均为字母。它的编码规则为：对于二字词，其码依次由词内第一个字的声母和第二个字的声母、韵母之首字母组成（即声、声、韵）；对于三字及三字以上词，其码依次由词内第一个字的声母、第二个字的声母和最后一个字的声母之首字母组成（即声、声、声）。我们要强调一点，这儿推出简化词码之目的不是为了推崇和宣扬这种词码，而是要以该码为例，开展对词输入处理技术的讨论。由于简化词码的简单性，故可以使以下的讨论比较简洁、明了。

四、词库结构设计

实现词输入的关键是把词码转换成词内码，为了实现这种转换，须借助于词库，它反映了词码与词内码之间的对应关系。显然，词库是词输入处理技术中一个重要数据结构。

最简单的词库结构如图5-1所示。显然，当词库内收词较多时，这种结构的词库需要 占用大量的存贮空间。如果让其驻留在内存，则要占用宝贵的内存资源；如果让其驻留在外存，则会大大影响对该库的检索速度，从而降低整个系统的响应速度。因此，这

种词库只能用于词汇量不大的系统中。

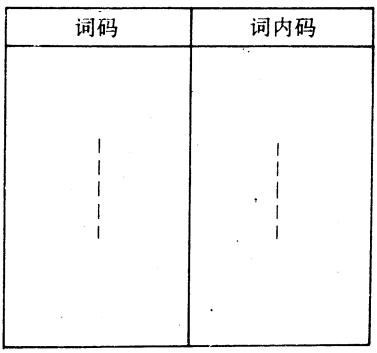


图 5-1　最简单的词库结构

从以上所述内容可知，词库的结构不但与所收词汇量有关，而且与词库所驻留的存贮介质有关，设计词库时应同时考虑到这两个因素。我们认为，词库内所收词的数量不能太少，而且应留有足够的扩充余地，以让用户增加所需之词。另外，应该让词库尽可能驻留在外存(一般指硬盘)上，以免过多地占用用户的内存空间。为此应提供附加的机构，以确保对驻留在外存上的词库的快速访问。下面就提出一种能符合上述要求的词库结构。

我们把这种词库称作外存词库，它的结构十分简单，它只是词内码的有序集合。词内码在词库内的排列规则为：具有相同词码之词(即重码词)排在一起，组成一个词块，即一个词块对应于一个词码；词块之间按照其对应的词码的字母序排列。图 5-2 给出了外存词库的结构。

图 5-2 中的分隔符用作词内码之间的间隔标志，可选用非词内码符作此分隔符。在采用变形国标码作为汉字内码的系统中，可以省略此分隔符，只要把词内码中最后一个字节的最高位屏蔽即可，可以利用这一点很容易地找到两个词内码之间的界线。这种词库可以以文件形式驻留在外存上。

为了实现对上述外存词库的快速访问，必须建立一张索引表，该表在外存上留有副本。索引表由表项组成，每个表项对应于一个词码，其内容为该词码所对应的词块含汉字的个数。图 5-3 为索引表的结构。

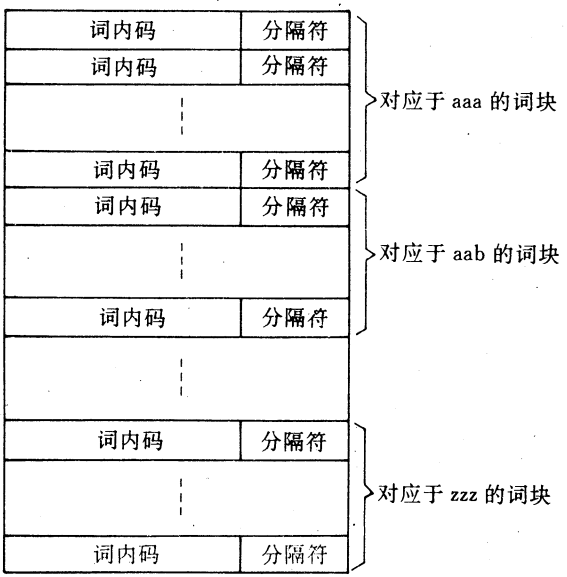


图 5-2　外存词库的结构

索引表的表项按照其对应词码的字母排列。如果某一个词码无对应的词块，则其对应表项之值为 0。根据已知的词码，经过简单的计算，再借助于索引表，即可很快地把词库中与已知词码对应的全部重码词(即一个词块)一次性读出。其详细算法将在下面介绍。

五、词处理程序的设计

词处理程序的执行过程就是实现词输入的过程，词处理程序的主要任务就是把词码转换成词内码。我们要求这个转换过程必须迅速，以加快整个系统的响应速度。此外，还要求该处理程序能

根据用户的要求，对词库进行一些操作，向用户提供对词库进行动态操作的手段。词处理程序的总体流程如图 5-4 所示。

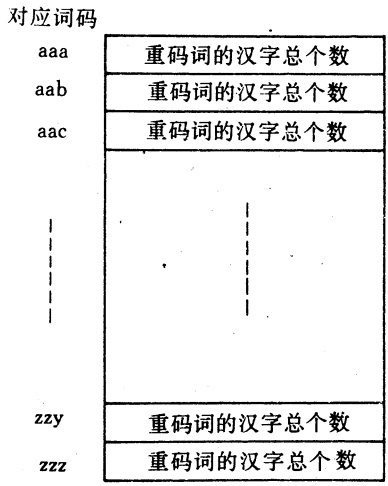


图 5-3　索引表的结构

下面对词处理程序作进一步讨论。

1. 词码转换模块

词码转换模块实现把词码转换成词内码。为了方便用户使用，采用输入码逐字符提示，即每键入一个词码符，就要提示出当前已有词码符所组成词码的对应重码词。因此，我们需要获得当前词码所对应的词块。欲获得某一词块之内容，必须知道该词块在词库内的起始地址 addr 和块长度 len。所以，词码转换模块的关键是设法构成两个函数 f_1 和 f_2，利用这两个函数分别由词码 code 算得 addr 和 len，即有：

$$addr = f_1(code)$$

$$len = f_2(code)$$

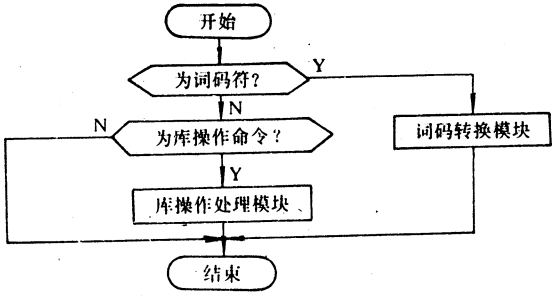


图 5-4 词处理程序的总体流程

下面以简化词码为例，讨论以上两个函数的构造方法。

令词码为 $c_1c_2c_3$，即 $c_i(i=1,2,3)$为词码符，并假定 $c_i \in \{a,b,\cdots,z\}$，再令 a_i 为 c_i 的 ASCII 码。下面分别讨论如何求得 addr 和 len。

(1)计算 len　根据图 5-3 给出的索引表的结构，我们可以把索引表表项所对应之词码看成是一个 26 进制的三位数，故可得到项序号 N 与其对应词码之间的关系为如下公式所示：

$$N = \sum_{i=1}^{3}(a_i\text{-}61H) \times 26^{3-i} \qquad (1)$$

设索引表为一维数组 T，其每一表项均为 T 的元素 T〔i〕，则可以得到对应于词码 $c_1c_2c_3$ 的重码词之总字数就是数组元素 T〔N〕，其中 N 由式(1)算出。

由于我们采用了逐字符提示，故当前词码长可能为 1、2 或 3，它们所对应的重码词总字数与当前码长有关。例如，当前词码仅为一个字母 d，则凡是首字符为 d 的词码的对应词均属当前重码词

范围。令当前码长为 n，则由下式可算出索引表中与当前词码相符的首项之项序号 N_0：

$$N_0 = \sum_{i=1}^{n}(a_i\text{-}61H) \times 26^{3-i} \qquad (n \leqslant 3) \qquad (2)$$

显然式(2)是由式(1)推广而来的。重码词总字数则为从索引表第 N_0 项开始的 26^{3-n} 项之和。又因为 len 应该以字节为单位，故其值应为总字数之 2 倍。从而有：

$$len = 2 \times \sum_{i=0}^{m} T[\sum_{j=1}^{n}(a_j\text{-}61H) \times 26^{3-j} + i]$$

其中 $m=26^{3-n}-1$，n 为当前码长，i 为对于 N_0 项的相对项序号。

(2)计算 addr　上面由式(2)求得了首项序号 N_0，根据词库与索引表之间的对应关系(见图 5-2 和图 5-3)，我们可以知道，addr 为索引表第 N_0 项之前各项之和的 2 倍，故有：

$$addr = 2 \times \sum_{i=0}^{N_0-1} T[i]$$

式中的 N_0 由式(2)算得。

这样，我们就由词码 code 获得了 addr 和 len，根据这两个值就可以从词库取得对应的词内码。由于这些词内码是连续存放的，故可用较快的速度一次性读得。

2. 库操作处理模块

库操作处理模块实现对库操作命令的解释，库操作命令应包括对词库的修改、删除、扩充和连接等。关于修改、删除和扩充命令的实现，不再进行讨论，由读者自己思考和设计。下面我们要论述一下向用户提供连接词库功能的必要性。

前面已经提到，《信息处理用现代汉语五千词表》可作为一个基本词库，各个专业和行业有必要建立适合自己使用 的专用词库。这样就要求系统具有动态切换词库的功能，以实现用户在使用过程中对词库的选择。另外，设立多个词库与设立一个大词库相比较，还具有以下优点：

①由于各词库独立编码，故有利于减少重码；

②由于只对当前连接的词库进行检索，故提高了检索速度；

③各词库可以用覆盖法来共享内存资源。

由此可见，向用户提供连接词库功能是完全必要的。实现此功能的方法并不难，可在系统内设立一个当前词库名记录单元，用于记录当前与系统连接之词库文件名。当使用词库时，系统根据该单元中记录的词库文件名，去访问相应的词库文件。词库连接命令解释程序所要完成的工作为：取得用户欲连接之词库名，并把它送入当前词库名记录单元，把该词库所对应的索引表读入索引表区。

六、词库生成法

所谓词库生成法，就是由用户产生系统所要求的词库的方法。一般来说，要求用户直接形成系统所要求的词库结构是一件十分困难的事，故有必要向用户提供一种较方便的词库生成法。

从用户的角度来看，建立词库的方法必须是方便的、直观的和容易掌握的，最好这种方法还要具有通用性。文本文件的编辑方法是为用户所熟悉的，它的建立过程不但简单和直观，而且还有建立的通用工具，比如行编辑程序 EDLIN 和字处理程序 WordStar 等均能用于建立文本文件。从系统的角度来看，词库必须紧凑和符合系统所要求的结构格式，这与用户的要求往往是矛盾的。怎样来解决这个矛盾呢？我们认为，用编译生成法来形成系统词库是一种比较好的方法。用户利用行编辑程序或字处理程序按文本文件的格式建立词库，我们把这样建立的词库称为源词库。然后由词库编译程序把源词库翻译（转换）成系统所要求格式的词库，我们把它称为目标词库。用户若要对词库进行修改或扩充，只要对源词库进行相应的修改，利用前面所说的编辑工具来做这项工作是很容易的。然后再对修改或扩充后的源词库进行编译，即获得所需的目标词库。

为了便于对源词库进行编译，故应对源词库提出一定的语法

要求，这种语法要求应该是面向用户的。我们把源词库分成若干行，每行对应于一个词码。每一行分成两个域，即输入码域和词域。输入码域的内容为该行所对应的词码，词域的内容为与输入码域中的词码相对 应的词，该域内允许有一个或多个词。当出现多个词时，说明这些词均对应于同一个词码，它们是重码词。这时，这些重码词之间应 以间隔符(逗号)分开。两个域间以域界符(冒号)分开。每一行均以行界符(回车符)结束。

为了严格起见，我们对源词库的语法作出如下的 BNF 描述：

〈词码符〉::=a・・z

〈间隔符〉::=，

〈域界符〉::=：

〈行界符〉::=〈回车符〉

〈词码〉::=〈词码符〉〈词码符〉〈词码符〉

〈词〉::=〈汉字〉〈汉字〉|〈词〉〈汉字〉

〈词表〉::=〈词〉|〈词表〉〈间隔符〉〈词〉

〈行〉::=〈词〉〈域界符〉〈词表〉〈行界符〉

〈源词库〉::=〈行〉|〈源词库〉〈行〉

从以上源词库的语法可知，这种语法符合用户编辑一般文本文件的习惯，所以可以称得上是面向用户的。

不同的系统有不同的词库结构，因此目标词库的语法描述不可能统一。但是，我们必须根据系统对词库结构的规定，对目标词库作出严格的语法描述，然后才能据此设计出词库编译程序。

根据源词库和目标词库的语法描述，我们可以编制出词库编译程序。一种最简单的方法是把它设计成三遍扫描型编译程序，第一遍扫描完成对源词库的语法检查；第二遍扫描实现把源词库的行按其词码的字母序排列，产生一个中间文件(临时文件)；第三遍扫描就根据中间文件产生目标词库。为了提高效率，亦可以把编译程序设计成一遍扫描型的，但是会增加设计和调试的难度。

第二节　词输入处理系统的设计

一、引言

我们在前一节中已对词输入处理技术作了比较广泛和深入的阐述，本节将对词输入处理系统的设计作进一步的介绍，将提出一种新的词库结构设计，使词库构造更加灵活，并能降低索引表的内存开销。另外，我们在本节中将注重系统的设计和整体功能的加强与完善。当然，我们亦可把本节看作是对词输入处理技术的又一次讨论。本节对词码设计和词库选词原则等不作讨论。

二、设计目标

我们希望词输入处理系统是汉字系统内的一个独立组成部分，它能与多种汉字系统相结合。为此，我们确定系统的设计目标如下：

(1)我们要求系统具有较好的兼容性，它能与汉字系统很好地适配，能保持汉字系统的全部功能，支持系统原有的所有软件及用户在汉字系统下开发的所有应用软件。

(2)我们要求系统具有高效性，它能合理地使用存贮资源，并要尽可能少占用宝贵的内存资源。系统对外界的响应速度要足够快，对词库的检索时间要限制在允许范围内，使汉字信息的输入速度有较大的提高。要求系统设计能处理好空间与时间的关系，尽可能使两者均获得比较满意的结果。

(3)我们要求系统具有独立性，系统应该自成一体，采用模块结构，不管在结构上还是在逻辑上，均保持相对独立。系统与汉字系统之间的连接应尽可能简单，要建立一个通用型的接口。

(4)我们要求系统的功能比较全面，它不但具有输入汉字信息的功能，而且具有对词库进行操作的功能(如装入、删除、修改和连

接等)。系统以词为基本输入单位,也能以单字为基本输入单位,即在词输入方式下,不必经过切换就可输入单字。

三、词库结构设计

词库的设计水平会直接影响到系统的内存开销和响应速度。我们权衡各方面因素,把词库设计成这样的结构:词库含有若干条链,链由若干个词块组成,词块由词项组成。下面分别进行讨论。

1. 词库驻留空间

我们不主张把词库驻留在内存,因为这样作的结果是,既限制了用户使用的内存空间,又限制了词库的容量。所以,我们主张把词库驻留在外存。由于访问外存的 速度较慢,故在设计词库结构时,应充分考虑到系统的响应速度。为了提高系统的响应速度,我们在内存空间建立一个常用词库,用于存放那些使用频率很高的最常用词(亦称为甚高频词)。这些词的词码设计成一字符型或二字符型,称之为词简码。当使用这些词时,就不再需要访问外存,从而可以提高系统的响应速度。

2. 词项

词项是组成词库的最基本单元,一个词项与一个词相对应,项中存放对应词的有关信息(词码和词内码等)。图 5-5 给出了词项的结构。

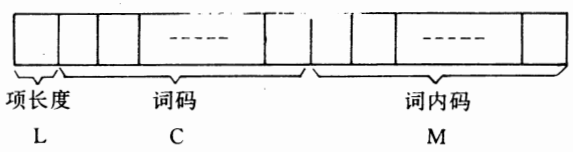


图 5-5　词项的结构

为了避免词的重码率,我们对词码的长度限制定得较宽,规定不超过 10 个字符,而且规定词码长度不必为定长,即 1～10 个字

符均可组成词码。为了实现对词项的访问，故必须用长度 L 来指出本项的长度(以字节为单位)。词码 C 和词内码 M 的长度不需要另外指出，因为这两者的组成字符分别属于两个不相交的集合，在程序中很容易把它们区分开来。

3. **词块**

为了便于对词库的内容进行增删，词库结构必须设计得具有动态性。我们把词库分割为一些词块，每个词块为 512 个字节。每个词块中含有若干个词项。有关的词块被连接成链，每个词块的末尾有一个指针，称其为链指针，它指向本链中下一个词块。链末词块的链指针值为 FFH，意味着链的结束。另外，词块内最后一个词项与链指针之间以 ESC 符(1BH)分隔。词块的结构如图 5-6 所示。

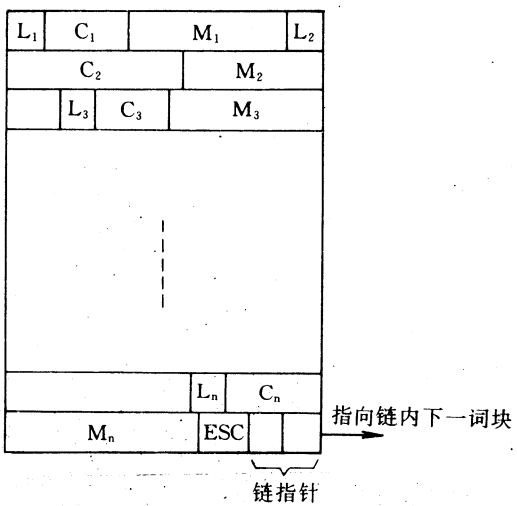


图 5-6　词块的结构

4. **索引表**

为了减少系统对词库的检索次数，以加快系统的响应速度，我们根据词码的前两个字符，把词库分成 26×26 条链，每条链由相应的词块组成。为了实现对这样结构的词库的快速访问(检索)，故

设置了索引表。图 5-7 给出了索引表的详细结构。

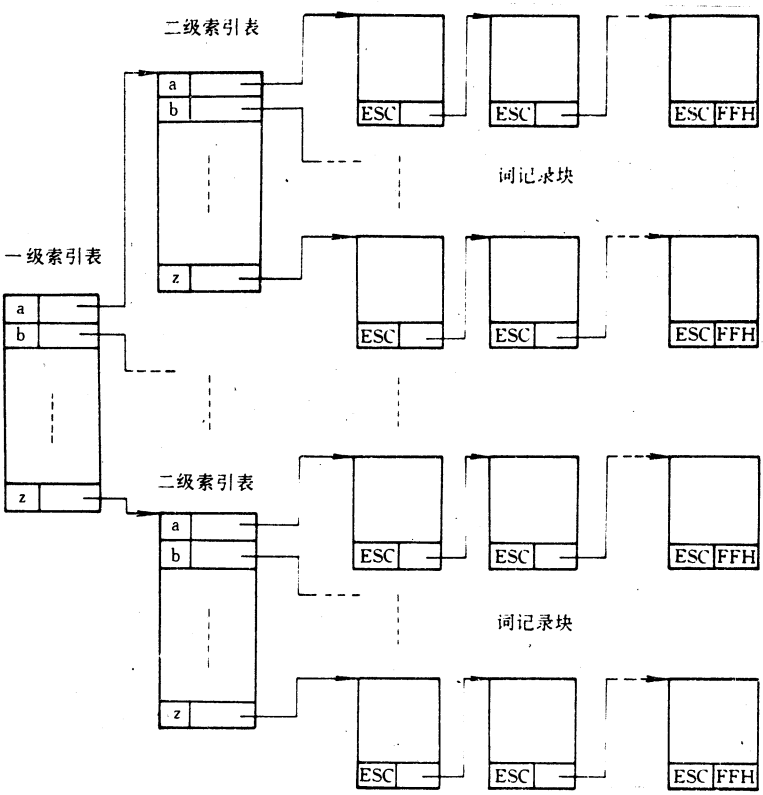


图 5-7 索引表的结构

从图 5-7 可知，我们设置了两级索引表，一级索引表针对词码的第一个字符实施索引，其表项指出词码第二字符的二级索引表的地址。二级索引表针对词码的第二个字符实施索引，其表项指向相应的词块链的链首。从图 5-7 中可以看到，两级索引可以有 26×26 条链。其实，实际链的数目决定于词码前两个字符的选择与

排列，一般不会有 26×26 条链。

5. **词库**

常用词库驻留在内存空间，其所收词的数量与外存词库相比要少得多。为了简单起见，常用词库只是其词项的连续存放，不再设链和块。它的词项结构与图 5-5 相同。

外存词库的覆盖率应该较高，由于汉语的词汇量十分庞大，故若不加选择地建库，则会形成“海量”词库。这样的词库是不可取的，因为词库越大，其占据的存贮空间就越大，对词库的检索速度就越慢。根据本章开头介绍的情况可知，按照各种专业和行业分别建立专用词库是可行的，这样建成的专用词库不会很大，而且其覆盖率较高。一般来说，建立一个约 15000 个词的词库，其覆盖率可达 95%左右。所以，本系统就采用这种方法，在外存上分专业建立若干个词库。在需要时，可调用系统的词库操作功能，把有关词库连上系统，以实现系统对该词库的使用。

四、系统实现

1. **总体流程**

词输入处理系统的主要功能是实现以词为单位输入汉字信息，即把用户输入的词码转换成其对应词的内码。另外，系统还要完成对词库进行操作的功能，即在词输入方式下能对词库进行装入词、删除词、修改词和连接词库等操作。根据上述情况，可以设计出如图 5-8 所示的系统总体流程。

系统总体流程主要由词输入模块、装入词模块、删除词模块、修改词模块和连接词库模块组成。这此模块可以分成两大部分，即词输入部分和词库操作部分，下面分别介绍这两部分内容。

2. **词输入的实现**

当用户键入词码后，系统就根据键入的词码分别对常用词库或外存词库（已与系统连接的）进行检索，把与词码对应的词项全找出来，并把词项中的词内码存入重码区。在重码区中，词内码之

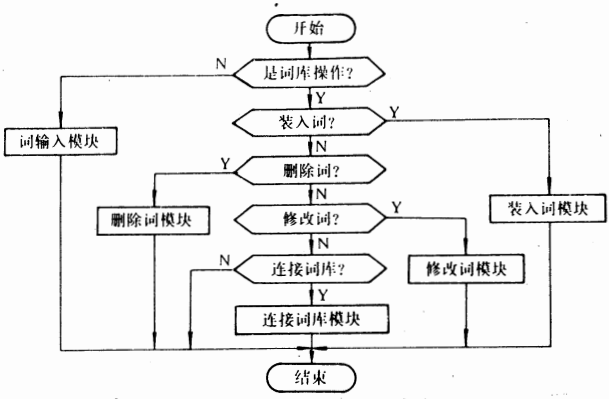


图 5-8 系统总体流程

间用规定的分隔符(如 ESC 符)分隔开。再在提示行内显示重码区中的内容(不包括分隔符)。然后根据用户键入的选择符("1"～"7"或 F_1～F_7),把重码区内相应的词内码或相应词中相应字的内码送入内码区,再把内码区内有效信息的长度送入内码区计数器。这些就是由词输入模块完成的工作。最后由系统的键盘输入模块(16H 号中断处理程序)把内码中的词内码或字内码返回给调用程序,这样就完成了词的输入。图 5-9 给出了词输入模块的流程。

从图 5-9 可知,词输入模块对输入符进行判别,若是简码,则检索常用词库,把符合的词找出并送入重码区;若为词码,则通过两级索引表检索外存词库,把符合的词找出并送入重码区;若为选择符,则调用词选择模块,实现用户对重码区中的词或单字的选择。图 5-10 给出了词选择模块的流程。

系统定义了两种选择符,用"1"～"7"作为词选择符;用 F_1～F_7 及后继选择符(可用"1"～"9")作为单字选择符,其中 F_i 指出选中重码区中的第 i 个词,后继选择符指出选中该词中的哪一个字。下面用例子来说明这两种选择符的使用。

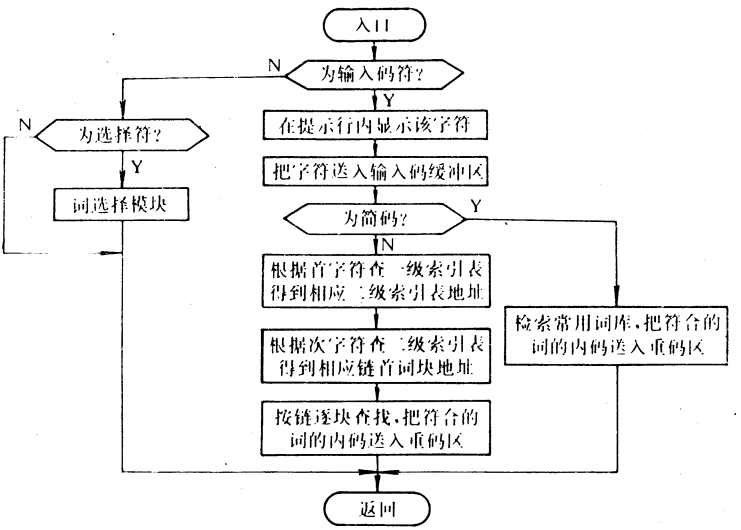


图 5-9 词输入模块的流程

如果欲输入“我们”这个词，只要先键入该词的词码，使提示行内出现“我们”这个词。假定该词在提示行内的序号为“3”，则按一下“3”键，即实现了该词的输入。

如果欲输入汉字“藏”，这时不必退出词输入方式，只要键入相应的词码，使提示行内出现任意一个含有“藏”字的词（比如“隐藏”）。假定“隐藏”这个词在提示行中的序号为“4”，则先按一下 F_4 键，这时提示行中的“隐藏”呈负向显示，再按一下“2”键（因为“藏”为该词中的第二个字），即实现了该字的输入。

这两种选择符实现选择的过程，在图 5-10 的流程中均得到了反映。

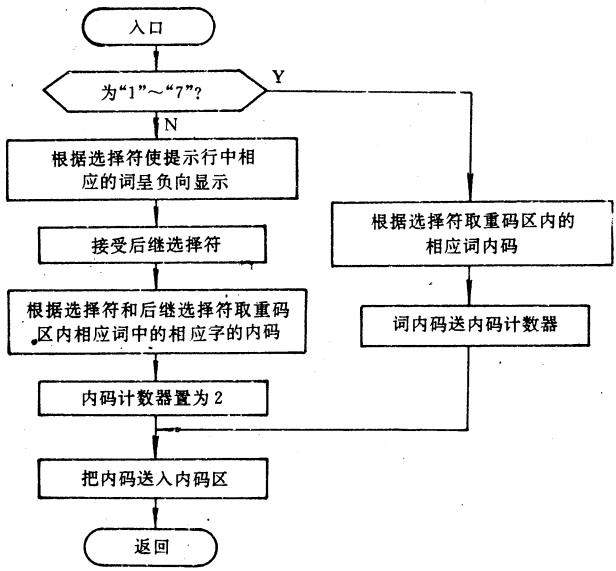


图 5-10 词选择模块的流程

3. 词库操作的实现

系统提供了四种词库操作功能，这些功能分别由四个模块来完成。下面分别介绍这四个模块的实现方法。

(1)装入词模块　装入词模块实现把用户新定义的词码和词装入词库。图 5-11 给出了这个模块的流程。

装入词模块首先接收用户给出的词和词码，按照词项格式，形成新的词项。然后根据词码查索引表，得到相应链的首词块，再找到该链之末词块。若末词块内有空，则把词项写入末词块；若末词块不空，则要申请一个空词块，把它挂到链末，并把词项写入之。

(2)删除词模块　删除词模块实现在词库中删除指定的词。它的执行过程如下：

①接收用户键入的词码；

②检索词库，把与该词码对应的词项均找出来，把词项地址逐

个填入工作区，把词内码逐个送入重码区，显示重码区的内容；

③接收用户键入的选择符；

④根据选择符取工作区内的相应项地址；

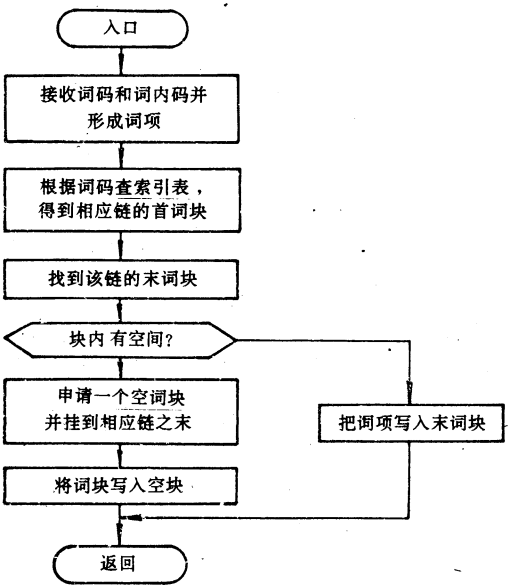


图 5-11　装入词模块的流程

⑤根据项地址把词库中相应词项内容全清为 0。

这样，就从词库内把指定的词删除掉了。

(3)修改词模块　修改词模块实现修改词库中指定词项的内容(词码或词内码)。图 5-12 给出了这个模块的流程。

修改词模块首先接收用户键入的词码，然后在词库中找到与此词码相符合的所有词项，并在提示行中显示这些重码词组；接收用户键入的选择符，确定用户当前要修改哪个词项；接收用户的修改信息，形成新的词项；把新的词项挂到相应链末，把原词项全清为 0。

(4)连接词库模块　连接词库模块实现系统与指定的词库相

连接，同时卸下原来与系统连接的词库。该模块的工作过程如下：

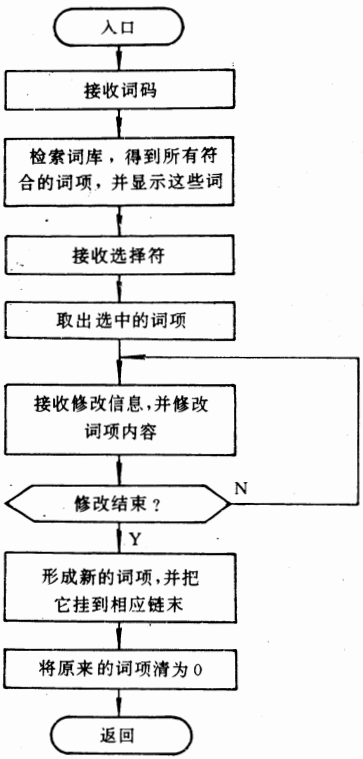


图 5-12 修改词模块的流程

①接收用户输入的词库文件名；

②根据词库名保存区中的内容，取得原连接词库文件名，关闭原连接词库文件；

③把新的词库文件名填入词库名保存区；

④打开新的词库文件；

⑤把新词库的索引表调入内存中的索引表区。

这样就实现了该词库与系统的连接。其中的词库名保存区用于存放当前与系统连接的词库文件名。系统自举时，根据该区中的内容，把相应的词库文件打开和把其索引表调入内存。所以，该区在初始化时应用一个隐含的词库文件名。

第三节　多级词库的结构与性能

一、引言

我们从前面两节的内容可知，词输入方式可以大大提高汉字信息的输入速度，它对解决汉字输入这一“瓶颈”问题具有相当大

的作用。词库是进行词输入处理必须依靠的重要数据结构，本节重点对词库的多级结构进行阐述，并对其性能作出了比较深入的分析。

词汇是词输入方式的基础。汉语的词汇量相当庞大，《现代汉语词典》中收词 5.6 万条，《辞海》中收词达 10 万条，其中一部分是“罕用词”和“冷僻词”。目前，汉字信息处理系统的词库中一般收词为 3～6 万条，这是一个不小的数目，词库要占用数百 K，乃至数 M 字节的存贮空间。因此，许多汉字系统采用全外存型词库，即把词库全部驻留在外存上。这样，在词输入过程中，必然会引起对外存的频繁访问，从而影响到系统的输入速度。有的汉字系统为了改善系统的输入速度，把一部分词库驻留内存，一部分词库驻留在外存，这就是最简单的多级词库——二级词库。

二、词库性能描述

1. 词库的作用

词库是描述词码与词内码之间映射关系的数据结构，系统必须依靠词库才能完成词输入处理。我们令 C 为词内码集，W 为词集，E 为词码集，可以用下式表示词输入处理过程：

$$c_i = f(w_i) = \Phi \cdot \psi(w_i) = \Phi(e_i) \qquad (1)$$

其中，$c_i \in C$，$w_i \in W$，$e_i \in E$，另外，f、Φ 和 Ψ 为映射，并且 f 为 Φ 与 Ψ 的复合。映射 Ψ 实现由词得出词码，这一过程由用户自己完成。映射 Φ 实现由词码得出词内码，这一过程由系统完成，词库是实现映射 Φ 的基础。由此可见，词库在词输入处理中具有十分重要的地位。

2. 性能描述

词库设计所追求的最主要目标是词库性能，所以有必要讨论一下词库的性能描述问题。词库的性能主要包括响应速度 v 和内存开销 m，显然 v 和 m 分别与词库性能成正比和反比关系。我们可以参照汉字库性能的描述方法（请见第六章），得出用 k 参数描

述词库性能的表达式：

$$k = \lambda \cdot \frac{v - v_0}{m} \tag{2}$$

其中，v 为从词库取得词的平均速度，v_0 为从词库取得词的最起码平均速度，我们可以把全外存型词库的响应速度作为 v_0 之值。m 为词库的内存开销，但 m 不能为 0。λ 为一修正量，当 λ 小于 1 时，表示侧重于对词库响应速度的考虑，当 λ 大于 1 时，表示侧重于对词库内存开销的考虑。

我们可以把式(2)变换成另一种形式。令 t 为从词库取得一个词的平均时间，t_0 为从词库取得一个词的最大平均时间，则式(2)可写成：

$$k = \lambda \cdot \frac{t_0 - t}{m \cdot t_0 \cdot t} \tag{3}$$

从式中可知，k 参数与 m 和 t 均成反比关系。

为了寻求提高词库的 k 参数的方法，有必要进一步分析 t 和 m 之间的关系。显然，t 和 m 间存在着一般的“时间”和“空间”的关系，但是它们之间还存在着更进一步的关系。t 并不单纯地决定于 m，在 m 一定的前提下，t 还决定于词库中内容分级分布的情况。如果词库内容分级分布得恰当，则会使 t 减小。我们可用下式表示 t 和 m 之间的关系：

$$t = \frac{r}{m \cdot g(W)} \tag{4}$$

其中，W 为词集，g 为 W 在词库中的分级分布法则，r 为其它因素。

三、多级词库

1. 词汇分布规律

尽管汉语的词汇量相当庞大，但是人们发现，在日常使用中常常集中在某个词子集上。这也就是说，有的词的使用频率相当高，有的词的使用频率很低。据《现代汉语频率词典》编者的统计，在含有 1314404 个词的语料中，其中使用频率最高的前 100 个词就覆

盖语料总量的40%左右，前2562个词就覆盖语料总量的85%左右。又有人对含有40302个不同词的语料进行统计，其中使用频率最高的前215个词就覆盖语料总量的50%左右，前1800个词就覆盖语料总量的80%左右。由此可见，日常对词的使用相当集中，一般不会在词集中"全面铺开"，被集中使用的词称为常用词或高频词。

另外，我们还可以发现，各部门、各专业所用的常用词并不一样，而且差别很大。还有，各人因写作内容和遣词习惯等因素，也会有个人独特的常用词。因此，不存在大量的被各专业公认的常用词，但是，在汉语词汇中确实有一些词可以说是各专业公认的常用词，不过其数量不多，这些词(如"我们"、"就是"、"因为"、"如果"等)在各种专业的文章中和口语中均有极高的使用频率。

2. 词库的分级

根据汉语词汇分布规律和词库性能描述方法，我们应该把词库分级驻留于内存和外存中，以提高词库的响应速度。但是，词库占用的内存量m不能太大，以免使k参数降低。只要合理选择词库内驻留内存部分的内容，是可以达到目标的。

我们可以把词库分为三级，第一级是常用词库，其中收入数十条公认的常用词，它常驻内存，其内容保持不变。第二级是动态词库，其中收入1500条左右当前使用的常用词，其内容能随使用环境而变化，它常驻内存。第三级是外存词库，其中存放系统所支持的全部词，它常驻外存。

动态词库的内容是至关重要的，它直接影响到词库的性能。由式(2)可知，在内存开销m一定的情况下，词库响应速度v决定了k参数的大小。要使v得到提高，则必须尽可能减少访问外存词库的次数，也就是要求动态词库的内容对输入内容具有尽可能高的覆盖率。这样，我们就要求动态词库中存放的是用户当前使用的常用词，故其内容必须根据当前使用的情况自动形成，这样的词库是"面向个人的"。通常采用的方法是，系统自动把当前使用的词存入

动态词库，一旦存满后，当要存入新的词之前，系统先淘汰掉动态库内的当前不用之词，再把新的词存入库。这样，动态库内存放的始终是当前使用的常用词。因此，这样的词库不仅是动态的，而且是面向个人的和自适应的。

根据以上介绍的情况可知，三级词库的内存开销为数十 K 字节，但是内存词库（包括常用词库和动态词库）中内容对输入内容的覆盖率很大，据测试可达 80%以上，从而大大减少了对外存词库的访问次数，使词库响应速度有明显的提高。显然，这样的多级词库有较好的性能。

四、性能分析

1. 词库响应时间

根据前面所述，多级词库通常的内存开销为数十 K 字节，这对当前的汉字系统来说，是绝对不成问题的。为了使讨论简单明了，我们不妨假定词库的内存开销 m 不变。这样，根据式(3)可知，这时的 k 参数决定于词库响应时间 t。因此，我们有必要讨论一下词库响应时间。词库响应时间 t 的计算公式如下：

$$t = p \times t_1 + (1\text{-}p) \times t_2 + (1\text{-}p) \times t_3 \qquad (5)$$

其中，p 为内存词库（常用词库和动态词库）内容对输入内容的覆盖率，t_1 和 t_2 分别为内存词库和外存词库的响应时间，t_3 为把词从外存词库送入内存词库的时间（包括淘汰操作的时间在内）。

因为内存的响应速度要远大于外存的响应速度，故有 $t_1 << t_2$ 和 $t_1 << t_3$。根据式(5)可得，欲使 t 减小，则必须使 p 增大，即内存词库内容对输入内容的覆盖率必须提高。这也就是说词库的响应时间与词库内容的分级分布情况有关，即词库性能（k 参数）与词库内容的分级分布情况有关，这与式(4)是一致的，由此可见确定内存词库（特别是动态词库）中内容的重要性。我们可以得出这样的结论：词库内存开销 m 不变时，若词库内容分级分布得恰当，则可使词库响应时间 t 减小，从而使 k 参数增大，即提高了词库的

性能。

2. **系统响应时间**

系统响应时间是指系统完成一个词输入的平均时间，我们要分析词库对系统响应时间的影响。计算机系统响应时间的公式如下：

$$t_s = t + t_k \tag{6}$$

其中，t_s 为系统响应时间，t 为词库响应时间，t_k 为用户键入词码的时间。

显然，t 和 t_k 均会影响 t_s。我们在前面已经分析了词库与词库响应时间 t 之间的关系，这儿要重点分析词库与键入词码时间 t_k 之间的关系。设击一键的平均时间为 t_d，每个字输入的平均击键次数为 n，平均每个词的组成字数为 l，则有：

$$t_k = t_d \times n \times l \tag{7}$$

其中，t_d 与用户的击键技术有关，与系统无关，那末当 l 一定时，t_k 则决定于 n。我们可以用下式计算出每个字的平均击键次数 n：

$$n = \sum_{i=1}^{h} \frac{1}{i} \sum_{W_i \in D_i} P(W_i)(d(W_i) + 1) + (1 - \sum_{W_i \in D} P(W_i)) \cdot n_r \tag{8}$$

其中，D 为词库中的词集，D_i 为词库中字数为 i 的词集，h 为词库中最长词的字数，$P(W_i)$ 为词 W_i 对输入内容的覆盖率，$d(W_i)$ 为词 W_i 的码长（字符个数），n_r 为不在词库中的词的平均每字击键次数。

从式(8)可知，每个字的平均击键次数 n 的大小决定于词对输入内容的覆盖率 $P(W_i)$。再由式(6)和式(7)可知，系统响应时间 t_s 的大小与 $P(W_i)$ 直接有关。如果词库内的词集收集得合理，则 $P(W_i)$ 就大，从而 t_k 就小，所以系统响应时间 t_s 就小。

综上所述，词库设计的优劣会直接影响到系统的重要指标——系统响应时间。词库内容的分级分布情况会影响词库响应时间，词库中词集的合理性会影响键入词码的时间，两者合起来影响系统响应时间。因此，我们在设计词库时，对词库词集和词库分级

分布的确定，应该十分谨慎和慎重。

五、词库的维护

1. 词的淘汰算法

为了保证动态词库中存放的是目前使用的常用词，系统自动地把当时使用过的词存入动态词库。但是，动态词库的容量是有限的，当词库装满词后，欲再装入当前使用之词，则必须寻求一种算法，以淘汰词库内的词，然后才能装入新的词。我们把这种算法称作淘汰算法，要求这种算法能尽可能淘汰与目前使用最无关紧要的词，从而使动态词库的内容对当前输入内容具有最大的覆盖率。

常用的淘汰算法有 FIFO(First In First Out)算法、LRU(Least Resently Used)算法和 LFU(Least Frequently Used)算法，其中 FIFO 算法总是淘汰最先进入者，其随机性太大，故效果欠佳。我们不考虑这种算法。

LRU 算法称作最近最久未用淘汰算法。它的基本思想是，用最近的过去来估计最近的将来。它认为，一个已在词库内的词在最近一段时间内未被使用的时间最长，则在最近的将来 也有可能不被使用，故可淘汰掉。

LFU 算法称作最近最少使用淘汰算法，即把最近一段时间内使用得最少的词淘汰掉。它的基本思想也是用最近的过去来估计最近的将来。它认为，在最近一段时间内使用得最少的词，在最近的将来可能不被使用，故可以淘汰掉。

LRU 算法和 LFU 算法均有不完全尽人意之处。为此，我们在这两种算法的基础上，提出一种 LFRU 算法(Least Frequently and Recently Used)，它是 LFU 算法和 LRU 算法的结合。LFRU 算法规定，首先按照 LFU 算法的原则，确定最近一段时间内使用最少的词，若这样的词只有一个，则把它淘汰掉；若这样的词有多个，就按照 LRU 算法的原则，把它们中最久未使用的那个词淘汰掉。若这样的词仍有多个，则按照 FIFO 算法的原则，把其中最先进入者

淘汰掉。实践证明，LFRU 算法有机地结合了 LFU 算法和 LRU 算法各自的长处，它的淘汰原则比较公平合理。

我们必须注意到，由于词的长度不是定长，故向动态词库内装入一个词之前，有时须使用 LFRU 算法多次来淘汰多个词，以获得足够的空间来容纳新装入的词。

2. **词的自定义**

前面已经阐述过，汉语的词汇量相当庞大，各个专业和行业的用词差别很大，汉字系统的词库不可能包罗万象，所以，系统一定要向用户提供自定义词的功能，以把用户所需之新词造入词库。这对于面向个人的动态词库尤为重要，因为用户自定义的词往往是该用户常用之词，故有必要把它收入动态词库。

一般来说，系统应该向用户提供一个自定义词的工具软件，它能把用户提供的词码和词内码转换成规定的格式装入词库，这种软件作为系统的实用程序向用户提供。它实现的是静 态自定义词功能，故适用于用户大批量地自定义词。

系统还应该向用户提供动态自定义词的手段，使用户能在输入汉字的过程中随时自定义词，即作到“一边输入一边定义”，大大提高了效率。我们可以在词输入处理模块中增加这一项功能。我们定义了两个工作区，分别定为 i__wbuf()和 e__wbuf()，i__wbuf()用于保存最近输入的单个汉字的内码，e__wbuf()用于保存这些单字的输入码。当需要时，则可把最近输入的 n 个(n 之值由用户指定)单字的内码和输入码存入词库 。这样，当用户用单字输入法逐字输入了一个词库内没有的词后，若其认为要把此词存入词库，则只要按一下自定义词的功能键，即可把此词存入词库，就完成了该词的定义工作。

六、小结

从上面对多级词库及其性能的讨论和分析来看，它具有较好性能。它所需的内存开销仅数十 K 字节，然而它的响应速度要比

全外存型词库快得多。多级词库的常用词库体现了汉语词汇使用的普遍规律，它的动态词库体现了汉语词汇在特定环境下使用的特殊规律，因此多级词库能获得理想的效果。

第四节　联想输入处理技术

一、联想输入的提出

采用词输入方式输入汉字信息，能提高输入速度。但是我们也要看到，这种输入方式仅能输入词库中已收入的内容，如欲输入词库中没有的内容，这种方式就无能为力。所以，在采用词输入方式的同时，仍要使用单字输入方式完成词库所没有的内容的输入。怎样协调好这两种输入方式之间的关系，这是一个必须要解决的重要问题。

近年来，一种称为联想式的汉字信息输入方式被提出来了，并获得了较快的发展。联想输入方式能较好地协调词输入方式和单字输入方式之间的关系，使这两种输入方式自然地、有机地结合起来。联想输入方式以单字为基础，以词汇为主导，它根据用户输入的一个单字，按照汉语的组词规律，联想出用户欲输入的下一个字的可能范围，然后给出相应的提示，供用户选择，再根据用户选中的字继续联想。我们把这个过程归结为“输入单字——联想——选择——联想——选择——…”。当然，这是一种理想过程。实际上，由于系统的联想能力是有限的，所以用户不可能完全通过选择来输入欲输入之全部内容。当发现不能选择到当前所要输入的内容时，必须采用单字输入方式输入之。这样，实际的联想输入过程为“输入单字——联想——选择——联想——选择——…——输入单字——联想——选择——…。”

从以上介绍的情况可知，用联想输入方式输入汉字信息时，只要对联想覆盖域之外的单字按编码输入，而对联想覆盖域内的字，

只要用选择符即可实现输入，这显然可以大大减少汉字信息输入的击键次数。

为便于下面对联想输入处理技术的讨论，为了使论述简明、扼要，我们先提出几点约定，以后不再另作说明。约定如下：

①设汉字内码采用变形国标码，即：

$R=\{GB_i \mid A0H<GB_i<FFH\}$　　$i=1,2$

汉字内码$=\{GB_1GB_2 \mid GB_1, GB_2 \in R\}$

②设输入码为等长码，并且码长为4。

③设汉字输入码均由小写字母组成，即：

$P=\{c_i \mid 61H \leqslant c_i \leqslant 7AH\}$　　$i=1,2,3,4$

输入码$=\{c_1c_2c_3c_4 \mid c_i \in P\}$

④设联想覆盖域足够大。

⑤设选择符由数字组成，即：

$Q=\{t \mid 30H \leqslant t \leqslant 39H\}$

选择符$=\{n \mid n \in Q\}$

二、输入码与联想处理的关系

汉字输入码是一种面向用户的编码，它根据汉字的语音、字形、部首和笔形对汉字进行编码，并力求使被编码的汉字与输入码之间有直接的规律可循，从而使用户容易记忆和使用。

根据前面的论述可知，联想输入处理的过程中分单字处理（对于联想覆盖域之外的字）和联想处理（对于联想覆盖域之内的字）。单字处理显然与输入码密切相关；联想处理仅与汉语组词规律有关，而与输入码无关。为了方便用户的使用，计算机汉字系统一般都配有多种输入码。由于联想处理与输入码无关，因此完全有可能使系统所配置的各种输入方式都具有联想功能，而且它们可以合用同一个联想处理模块。

三、联想式数据结构

1. 联想式数据结构的基本型

系统内完成联想输入处理的程序称为联想输入处理程序，该程序的正常工作，必须借助于有关的数据结构，我们把这样的数据结构称作联想式数据结构。联想输入功能的实现应包括单字处理和联想处理，下面分别介绍这两部分所涉及到的数据结构。

单字处理一般都要依靠输入码表，该表反映了输入码与内码之间的对应关系。利用输入码表可以方便地实现输入码到内码的转换。输入码表的一个表项对应于一个汉字，表项的多少取决于系统所支持汉字的个数。一般来说，系统每支持一种输入码，就配有一张相应的输入码表。

联想处理要联想出当前输入之汉字(即当前字)的后继字。这种联想决不是计算机自身的智能，而是由系统设计者事先安排好的，这种安排体现在联想表中，计算机完全依靠联想表来产生联想智能。联想表内存放各个汉字可能与之搭配成词的后继字。例如："大"的后继字可以是"家"、"小"、"会"、"使"、"约"和"概"等字。联想表中所收后继字的多少，决定了联想覆盖域的大小，如果某个汉字无后继字，那末该字就不收入联想表。

单字处理和联想处理并不是截然分开的，它们之间相互结合，互相渗透。所以，输入码表和联想表之间存在着联系，它们之间通过指针连接起来，形成联想式数据结构。图 5-13 给出了联想式数据结构的基本型。

在图 5-13 中，输入码表表项的右边部分为内码或指针。如果该表项所对应的汉字无后继字，则其内容为该汉字的内码，否则其内容为指针，此指针指向该汉字之后继字在联想表中的起始地址。联想表中的主字就是当前字，亦就是前导字，其后为该主字的后继字。为了实现整个输入过程中联想的延续，故联想表中还含有后继指针。我们规定，主字的后继指针为全 0，后继字的后继指针指向

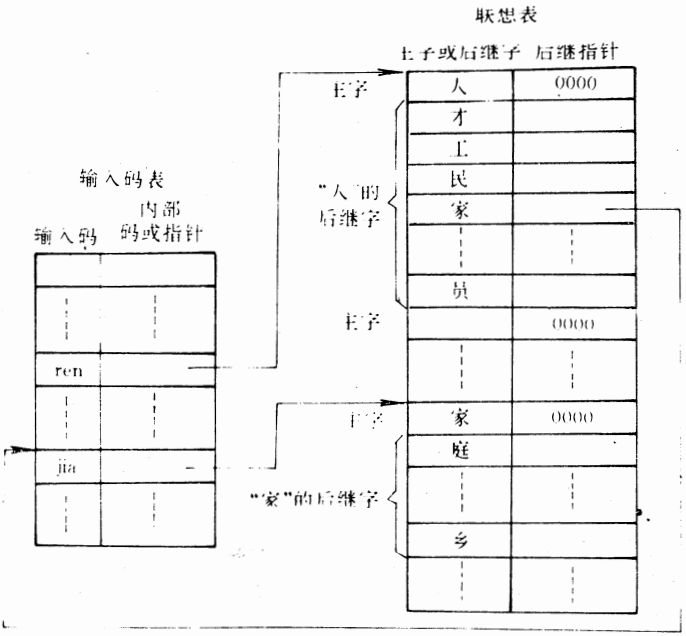


图 5-13 联想式数据结构的基本型

该字对应之输入码表表项。通过后继指针就可以逐字联想下去。另外，也可以根据后继指针来区分出两个主字的后继字边界，如果遇到为全 0 的后继指针，即知道本主字的后继字到此为止，下面为另一个主字的后继字。

2. 联想式数据结构的实用型

上面从原理方面对联想式数据结构作了讨论，下面从实用方面继续对联想式数据结构作进一步的讨论。设计一个实用的数据结构应同时顾及两个方面：一要使所占用的存贮空间小，二要使与此数据结构有关的算法简单。我们根据这两个原则，提出了一种联想式数据结构的实用型，详情如图 5-14 所示。

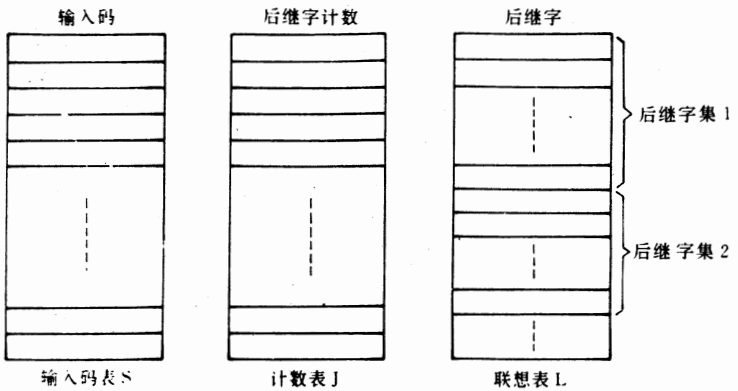


图 5-14 联想式数据结结构的实用型

图 5-14 中共有三张表，它们是输入码表 S、计数表 J 和联想表 L。我们可以把它们均看作一维数组。输入码表 S 的每一表项与一个汉字相对应，表项内容就是此汉字的输入码。该表的表项按其对应汉字的国标码顺序排列。计数表 J 的每一个表项也与一个汉字相对应，表项内容为此汉字的后继字个数。该表的表项也按其对应汉字的国标码顺序排列。由此可知，计数表 J 的表项与输入码表 S 的表项具有相同的顺序，它们之间一一对应。联想表 L 由若干个后继字集组成，每个后继字集内存放的是同一个主字的后继字。所以，一个后继字集与一个主字相对应。联想表 L 内的后继字集按照其对应主字的国标码顺序排列。

显然，图 5-14 中的数据结构要比图 5-13 中的简单得多。如何使用此数据结构实现单字处理和联想处理，将在下面作进一步阐述。

四、联想功能的实现

1. 联想输入处理程序

汉字信息的联想输入功能由联想输入处理程序实现。该程序要完成单字处理，它根据用户输入的输入码内容得到其对应汉字的内码，从而实现了单个汉字的输入；该程序还要完成联想处理，它根据当前输入的汉字（当前字）联想出它的后继字，并给出提示，再根据用户的选择完成后继字的输入，并把该字作为当前字继续联想。图 5-15 为联想输入处理程序的流程。

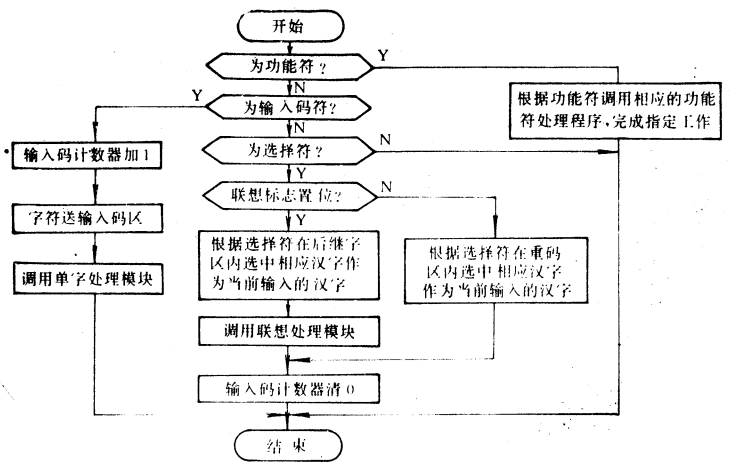


图 5-15 联想输入处理程序

图 5-15 中的功能符由系统定义，并由相应的模块完成其对应之功能。流程中涉及到的重码区和后继字区均为工作区，重码区存放重码汉字，后继字区存放一个后继字集的内容。根据前面的约定可知，选择符和输入码符分属两个不相交的集合（即 $P \cap Q = \Phi$），故以上流程中可根据对当前字符的判别，分别进入单字处理和联想处理。下面分别对单字处理模块和联想处理模块进行讨论。

2. 单字处理模块

单字处理模块通过对输入码表 S 的检索，获得与当前输入码对应的所有重码汉字的内码，并存入重码区。该模块的关键是如何

根据输入码表得出内码，下面讨论这部分算法。

设汉字内码的第一字节为 GB_1，第二字节为 GB_2，再设输入码表的第 n 项(即 S[n])与当前输入码相符。根据输入码表表项的排列规则和国标码的区位特性，从汉字内码(变形国标码)与国标码之间的关系，可以得出计算 GB_1 和 GB_2 的如下公式：

$$GB_1 = [n/94] + B0H$$

$$GB_2 = (n \text{ MOD } 94) + A1H$$

由此可见，单字处理模块是比较简单的，它与一般的以单字为基本输入单位的处理程序相比，没有特殊之处。所以，我们在这儿不再另作更深入地介绍。

3. 联想处理模块

联想处理模块依靠计数表 J 和联想表 L，实现对当前字的联想，把当前字的后继字集存入后继字区，并给出提示信息。联想处理模块的流程如图 5-16 所示。

从图 5-16 的流程可知，联想处理模块的关键部分是，如何根据当前字得到其对应的后继字集。假定当前字的内码为 GB_1GB_2，下面给出这部分的算法：

①根据当前字内码，用以下公式算出该字对应的计数表表项之序号 m：

$$m = (GB_1\text{-}B0H) \times 94 + (GB_2\text{-}A1H)$$

②用下列公式计算出当前字的后继字集在联想表 L 中的首项项序号 k：

$$k = \sum_{i=0}^{m-1} J[i]$$

③根据前两步算出的 m 和 k，把当前字的后继字集内容送入后继字区，即把联想表内的 L[k]、L[k+1]、L[k+J[m]-1]共计 J[m]个项之内容送入后继字区。

以上算法既简便又迅速，它完全是依靠图 5-14 中的数据结构来实现的。到此，我们可以对这种数据结构的设计用意有较深刻的

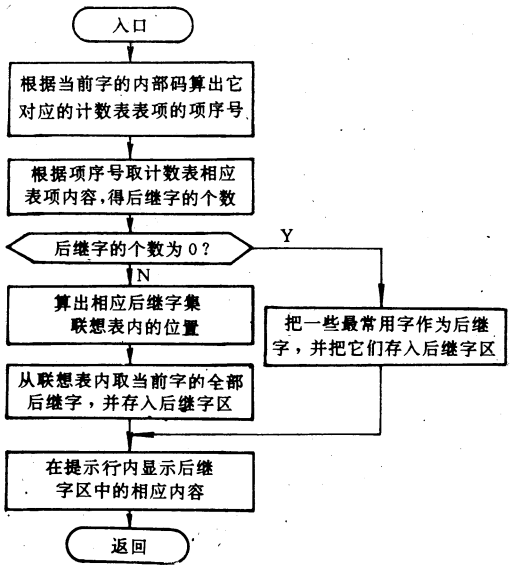


图 5-16 联想处理模块

理解了，同时也证实了这种数据结构的设计完全能符合前面提出的两个基本原则。

下面继续对图 5-16 中的流程进行说明。若当前字没有后继字，那么照例联想到此终止，下面如果再要输入内容的话，必须按输入码进行单字输入，然后重新开始联想。但是图 5-16 的流程中并没有这样作，而是把一些最常用的高频字（如“的”、“地”、“是”等字）作为当前字的后继字，并存入后继字区，其目的是为了使联想尽可能继续下去，实践表明，这样作后，在不少情况下通过不断地联想，可使许多内容在输入过程中不必再使用输入码作单字输入。另外，这样作也等于扩大了联想覆盖域。

根据联想处理模块的内容不难看出，如果要对不同的输入码实现联想输入功能，只要改变输入码表即可，而不用改变计数表和联想表的内容，这两张表可视作公用表。这也证实了联想处理与输

入码无关这一点。

五、进一步讨论

联想输入方式使汉字信息输入技术向智能化迈进了一步，使汉字信息输入方法进入智能化的“初级阶段”。目前的联想方法大多只是按词进行的，其得到的后继字集不是太小的，从而用户的选择范围也不是太小的。选择范围过大，会降低输入速度，并会增加击键次数(按翻页键)。如果把后继字集减小，则等于减小了联想覆盖域，降低了联想命中率，导致单字输入次数的增加。解决联想覆盖域与输入速度之间矛盾的最有效方法是提高智能化程度，使在联想覆盖域不变(甚至增加)的情况下，尽量减小用户的选择范围，以提高汉字信息的输入速度。这就需要我们把联想方法从按词联想发展到按词组联想和按句子联想，把产生后继字的制约条件从组词规则推广到文法和句法规则，这样肯定可以大大缩小联想产生的选择范围，甚至可使选择域内仅为一个字，这样也就不用选择了。当然，要达到上述智能化程度，还需要付出巨大的努力。

第五节　基于词组的智能型汉字输入系统

一、问题的提出

我们在前一节中对汉字信息的联想输入处理技术作了介绍，从中可以体会到，今后汉字输入技术必将向智能化发展。本节将在此基础上，进一步介绍基于词组的智能型汉字输入系统的设计和实现。

通常的汉字输入技术可以抽象化为：

$$f(e) + op = h$$

其中，f 表示编码层次，op 表示软件层次，h 表示某一特定的汉字，e 表示输入码。用户输入汉字 h 的过程是这样的：首先输入

其输入码，计算机按编码规则 f 求出与输入码对应的汉字(集)，记为 f(e)。必要时，用户必需响应编码之外的某种操作 op，以此来唯一地确认 f(e)中的汉字 h。

问题的另一方面是，用户想连续输入的汉字串是笛卡尔乘积

$$f(e_1) \times f(e_2) \times f(e_3) \times \cdots \times f(e_n) \qquad (1)$$

中的一个元素，于是，新的汉字输入模式可以表示为：

$$F(f(e_1) \times f(e_2) \times \cdots \times f(e_n)) + op = h_1h_2\cdots h_n \qquad (2)$$

其中 F 要根据汉语本身的规律和用户操作环境，把(1)中的元素个数降低到最小范围，用户在这个范围内进行 op 操作，得到一串汉字 $h_1h_2\cdots h_n$。

下面我们用一个例子来说明以上所述情况。例如，对于

$f(e_1)=\{系，矽\}$

$f(e_2)=\{统，彤\}$

那么，在 $f(e_1)\times f(e_2)=\{(系统)，(系彤)，(矽统)，(矽彤)\}$中，用户想输入“系统”的可能性要远远高于想输入其它三个元素，这是由语言现象本身所造成的。于是 $f(e_1)\times f(e_2)\times\cdots\times f(e_n)$中，合理的元素个数要大大低于它的笛卡尔积的元素个数，用户进行 op 操作的范围大大缩小了，很容易得到一串汉字(如“系统”两字)。

基于词组的智能化汉字输入是以词输入为主，以单字输入为基础。它的研究层次是使(2)中的汉字串 $h_1h\ldots h_n$ 是一个词组，其中的 F 已脱离了编码层次 f，只对 f 的结果进行操作，而不管 f 的具体含义，所以允许用户的输入码可以是不完整的，F 是对 f(e)的笛卡尔积处理。这样，它必须根据词的组成规律和词组的合成关系，能够自动推理产生词和词组，具有自动学习和更新知识的能力，体现出一定的智能特点。

基于词组的智能化汉字输入可分为单字输入技术和词组输入技术两部分。单字输入技术主要有：增强翻页功能和模糊输入功能。词组输入技术主要表现为联想输入技术、前后链输入技术和编码词组输入技术等。其中联想输入技术又可分为单字联想输入技

术和词组联想输入技术。下面分别详细介绍。

二、单字输入技术

单字输入技术是由一个被称为编码转换模块实现的，它的功能是将输入码转换为汉字内码，且返回给调用者，其实质是：不断获取西文字符(即输入码符)，并在模块内部消化解释掉，直至产生内码为止。这里要特别注意一点，就是编码转换模块中不能消化的字符，要原封不动地返回给调用者。

编码转换模块的框图如图 5-17 所示。

说明如下：

①一般汉字输入系统都配有多种单字编码输入技术，所以要用一个特殊的定义键激活相应的编码输入处理。通常这些定义键是 ALT＋F1～F10。一旦用了某个定义键激活了一个编码模块，那么，就要初始化一切条件。

②这里所谓的“RET1”和“RET2”是指 1 字节返回和 2 字节返回的意思。1 字节返回就是该字符本模块未予消化处理，直接流给调用者，2 字节返回一般就是一个汉字内码返回。

在单字输入过程中，重码汉字时有发生，通常的处理是将这些重码汉字在屏幕的系统提示行分页显示给用户挑选确认，翻页的范围仅限在重码汉字集的范围内。如果输入码错，由错误的输入码确定的重码汉字集也极可能不包含用户想输入的汉字，所以翻页功能也无助于用户的挑选。如果可以跨出重码汉字的范围，这可使得对于不能准确输入输入码的用户，基本上也能用翻页键找到欲输入的汉字(条件是第一键必须正确)。另外，也有的用户可能对输入码中的某一个输入码符或者某几个输入码符不能准确确定，若用“?”代表任意一个输入码符，用“＊”代表任意个连续的输入码符，则无疑能方便用户找到欲输入的汉字。

由于单字输入技术相对来说比较简单，在此仅以可以跨出重码汉字集范围的翻页作一个简单的描述。

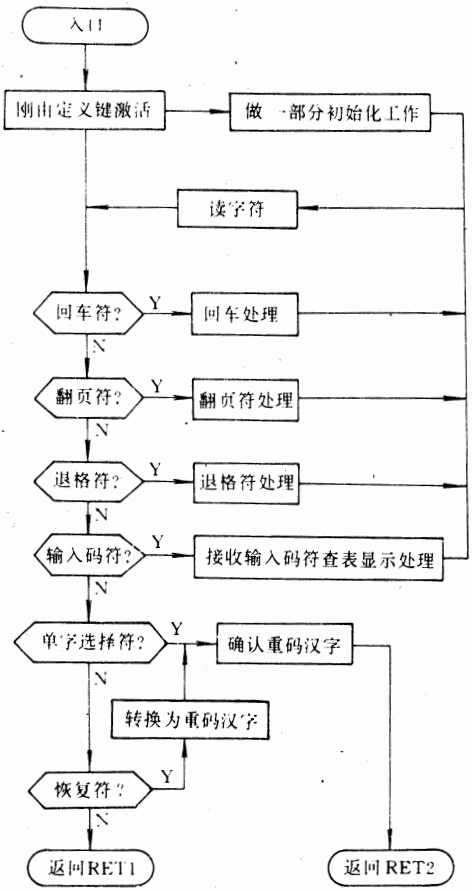


图 5-17　编码转换模块流程图

由于单字输入技术相对来说比较简单，在此仅以可以跨出重码汉字集范围的翻页作一个简单的描述。

计算机内记录每个汉字的数据结构可用列表枚举的形式表示，每条记录的格式可以定义为：

<输入码><汉字>

为了节省存贮空间，将所有输入码相同的记录合并在一起，成为：

＜输入码＞＜汉字 1＞＜汉字 2＞…＜汉字 n＞的形式。实际上，此输入码是重码，后面的汉字是重码汉字。汉字的排列次序很重要，它决定了重码汉字在提示行出现的先后，一般是按高频先见的技术排列。

这样，跨出重码汉字集范围的翻页，可按图 5-18 的数据结构实现。

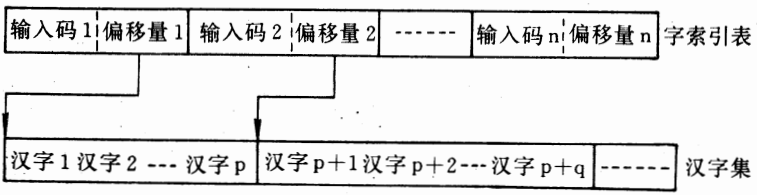


图 5-18　单字数据库结构

三、词组输入技术

所有基于词组的输入技术(联想、前后链、词组编码)都要由词组数据库支持。词组数据设计的好坏，将直接决定整个系统的性能。

1. 词组数据库设计的一般原则

对于词组数据库的设计，至少要考虑以下几个方面：

(1)简洁性　由于汉字是象形文字，每个汉字的表达至少要 2 个字节，词库仅支持几千条词组显然未能让用户享受到词组输入技术的优越性，所以，一般情况下，词库收词组至少上万条。这样的词库占用空间是相当可观的。因此，不得不考虑词库的压缩问题。而功能与性能两方面往往是矛盾的。

(2)效率　对词组数据库的访问效率直接体现了系统的性能。如果对数据库的访问慢到了不可容忍的地步，那么，这种输入技术是不实用的。在考虑效率的同时，应该注意到硬件支持环境的发展

趋向，也许目前体现的性能不够理想，但可望的将来，在新的硬件支持下，性能可以改善，这一点应该给予考虑，以便设计出结构更好的数据库。不过总的说来，在设计数据库的时候，对数据库本身不考虑硬件环境的支持，尽量保证效率也是应该的。

(3)可维护性　目前看来，在程序处理上要体现出较大的智能化是极困难的，这是由于计算机体系结构的限制。所以，如果能保证数据库有良好的动态性能，即在系统运行中，能够较快地更新变化数据库，那么，系统整体上就能表现智能现象。这就要求设计数据库时，要尽量保证能对其进行快速简便的维护，最简单基本的维护是增、删操作，其次是调整次序的操作。

2. 词组数据库的结构

首先，数以万计的词组肯定不能以列表枚举的形式存在，必然要作某些压缩工作。词组是由词构成的，所以词库中的词组是通过一组关系来表示的，这种关系描述了词与词之间的构成。这样作的目的同时也是出于维护性的考虑，使词组的增、删操作能以较快的速度完成。

其次，单字联想输入技术要求根据已输入的单字，迅速得出相应的联想域。所以，一个词分为词头和词尾，具有相同词头的词尾聚合成块，称为词尾块。为了迅速定位，词尾块的长度设置为定长。

最后，词联想输入技术又要求根据已输入的词，迅速找出相应的后继词，所以，模仿词头和词尾的做法，每个词(实际上是词尾)要对应于一个联想块，以指示其后继词。

于是，基于词组智能化汉字输入的词库结构如图 5-19 所示。

简单说明如下：

①词头库：它由若干项组成，每个项称为词组索引头。每个顶对应于一个汉字。项序号按汉字的内码排列，共 6768 项(其中由于 55 区只有 89 个汉字，因此实际汉字系数为 6763 个)。表项内容是该汉字对应的词尾块的块号。若内容为 0，表示此汉字无词尾，若为其它正整数，则表示词尾块号。

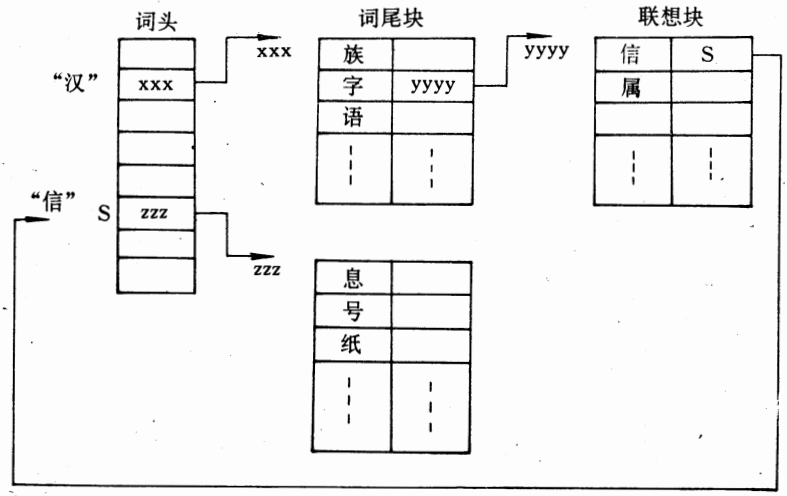


图 5-19　基于词组智能化汉字输入的词库示意图

如"汉"的项序号为 966，词尾块号为"xxx"。

②词尾库：它由若干词尾块组成，每个块由若干项构成，每项有两个字段：词尾字段和词联想索引字段。词尾字段可以是不定长的，表示本词尾块的词 头可以与自己构成一词，词联想索引字段是定长的，指示与本词对应的联想块号。每个词尾块都附带一个链指针。因为词尾块是定长的，所以当一个块容纳不下具有相同词头的词尾时，就用下一个词尾块继续表示，链指针就指示这个后继块号。若词联想索引字段值为 0，表示无联想块，若为其它正整数，则表示该词对应的联想块号。一般词尾块设置为 512 个字节。如上图，"汉字"的联想块号为"yyyy"。

③联想库：它由若干联想块组成，每个联想块由若干项构成，每个项有两个字段——词头字段和词联想序号。词联想序号表示以相应词头库定位的词尾块内的某个项所表示的词，就是所要求的联想词。如"汉字"的联想词是"信息"，则联想库中，词头字段为"信"字，词联想序号为"S"，即为"3045"。

④库号：上面描述的词库都有一个库号，用 1 个字节表示。0 号库约定为系统词库，就是所说的静态库，1～255 号库可以约定是用户私有库，即动态库。

3. 词库的动态调整

词库的动态调整是在系统内部实现的，主要通过自动学习，为词库增加知识。较完备的词库维护功能由一系列与系统并列的实用工具完成。当然，这样作也会带来一个问题：随着系统运行时间的延长，词库也许会越来越庞大，甚至可能超出允许的范围。这就要求用户经过一段时间的使用后，用维护工具对词库做一下调整。这个要求应该说是合理的、值得的，因为它节省了系统一大笔开销。词库知识的增加有两个方面：词的增加和词组的增加。

(1)词的增加　对于基于词组的汉字输入技术，用户在一次输入过程中，通常完成不止一个汉字的输入。所以，如果用户在一次输入过程中仅输入一个单字(系统是可以知道的)。那么，一种情况是输入的单字本身就是单字词；另一种情况是用户想输入的词组在系统的词库中没有收入。对于第一种情况可以认为是正常的，第二种情况的特征是有连续多个单字输入，这一点系统是很容易跟踪到的。因此，系统就可以记住这些字串，直到用户在以后的输入过程中完成了词组的输入。然后，分析记住的这些单字串表达的词(组)的知识在词库中是否存在，若否，则将这些词收录在词库中。

(2)词组的增加　对于用户连续输入的词，如果当前词库内没有它们之间的构成关系，词库就应该自动添加这对联想关系。例如，词库中仅存在“汉字”、“信息”、“处理”这些词，一旦输入“汉字信息”后，词库便自动建立“汉字”→“信息”的联系，紧接着再输入“处理”后，又把“信息”→“处理”的联想关系建立起来了。于是，词库中就增加了“汉字信息”，“信息处理”，“汉字信息处理”这些词组。

(3)动态调整复位　词库的动态调整，实际上是系统跟踪用户已经确定的输入，判定它们所体现的有关词组的知识是否存在于

词库中，若否，就要建立起来。理论上讲，这种学习应该是有条件地、分析地进行，不能“囫囵吞枣”，照单全收。

四、联想输入技术

1. 单字联想输入技术

单字联想输入技术是对单字输入技术的扩充。它根据用户刚输入系统的单字，由相应项指示的词尾块就是联想域，把它们按顺序排列在提示窗口，如果用户想输入的个体被联想域命中了，那么，用户就可以用提示在每个个体前的提示符选择那个个体。这里有两个问题必须考虑：

①选择联想域个体的选择符不能是单字编码的输入码符，如果可以是的话，系统在接受这样一个字符时，就不知道它是下一个单字的首输入码符，还是在联想域中选择个体的选择符。

②联想域有时在提示行的一次操作中不能完全显示完毕，需要提供翻页功能，查看联想域的全体。但是，翻页的次数不能太多，否则就享受不到联想技术的优越性了。与其翻页五、六次去选择联想个体，还不如直接输入下一个单字的输入码。

单字联想处理要实现的是，根据已确定的汉字，把它在词头库中的表项内容取出，找出它所确定的词尾块，再提示出来，并供用户选择用，其步骤如下：

①从汉字内码 h_i 到项序号的对应关系是：

$$i = (GB_l - 0B0H) * 94 + (GB_h - 0A1H)$$

②读词尾块模块的入口参数是汉字内码及序号，该模块的流程图如图 5-20 所示。

③如果读到了相应的词尾块，就在提示窗口将这些词尾显示出来，并马上预读一个字符。如果该字符是设置的特殊引导键，那么就设置一个刚刚按过的标志，再读取一个字符，进入编码转换模块。如果预读到的字符不是引导键，那么就说明这些联想工作对用户没有帮助，就直接进入编码转换模块。

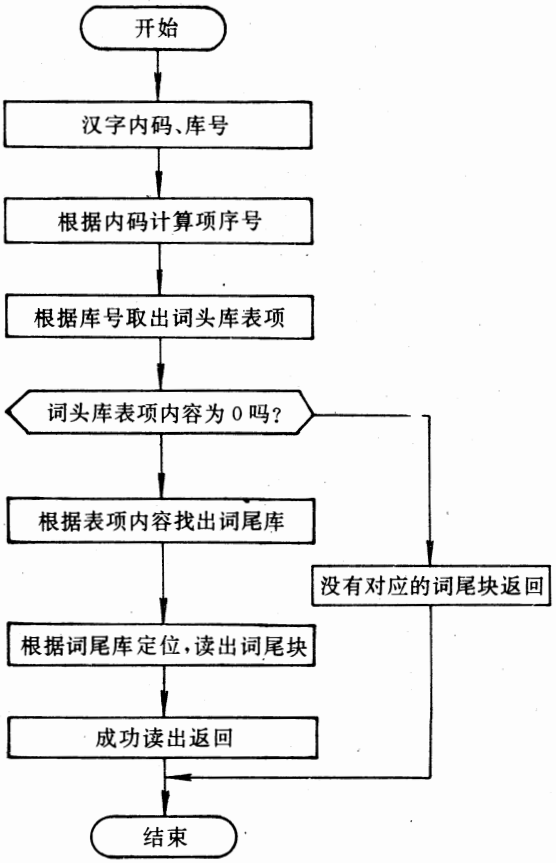


图 5－20　读词尾块流程

④如果引导键标志被设置了，那么，在图 5-20 所示的编码转换模块的“输入码处理”部分要有分支了，因为这时的输入码符已经被转义为联想词尾的选择符了。“翻页符”处理部分也有类似的分支。

2. 词联想输入技术

对于已经输入的词，找出其联想块号，接下去的处理就类似单字联想处理过程那样，不过要注意的是以下两点：

①词联想处理过程的前继操作是词组编码输入技术输入了词，也可以是由单字联想输入技术输入了词尾，这个词尾要与它的词头合并起来，作为词联想输入处理的入口。

②词联想处理过程是一个循环的过程，用选择符确认了联想域的个体后，还应该对这个个体继续进行词联想处理，直到用户用标点或新的输入码符中断本次词联想处理过程。

词联想处理实现的是：如果刚刚输入过程中完成的是词输入，那么，就取这个输入词的词组联想索引字段，称当前词联想索引字段，再根据当前词联想索引索引表项内容读出相应的联想块，最后由联想块装配出对应的联想词。

这里关键是怎样获得当前词联想索引表项的内容，下面给出此过程的实现步骤。

①在单字联想选择符处理成功时，因为这个词尾和引导这个词尾的词头构成一词，所以此时要设置相应的当前词联想索引表项内容。

②在词组编码输入中，选择处理成功时，自然地应该设置好相应的当前词联想索引表项内容。

③由当前词联想索引表项内容读出相应的词联想块的过程，类似于读词尾块的过程，参见图 5-20。

④由联想块装配取出对应的词，要将联想块中所有的词头非空的项，根据联想库的库号和词头读出相应的词尾块，并取出其中词尾块内序号的词与联想库词头组成一词。

⑤以后的处理类似字联想处理中的③～④，应该注意到由词联想处理选择符成功时，也要设置相应的当前联想索引字段的内容。

⑥词联想处理的选择引导键应当不同于字联想处理的选择引导键。

3. 联想域的讨论

联想域的命中率决定了联想处理技术实用化程度。命中率低了，就等于没有提供联想处理技术。当然联想域越大，命中率越高。但是，联想域过大，用户在挑选排在联想域后面的个体时，额外的操作特别多。既要保证一定的命中率，联想域又不能太大，这是一

个没有确定准则来平衡的矛盾。

五、词组编码输入技术

1. 词组编码输入技术的设计

这里词组的编码是不确定的，只取每个汉字输入码的一部分就可以了。下面以拼音输入码方案为例，对词组编码作了如下简化：

为了分断系统得到的输入码符的所属，在每个输入码之间以空格符作间隔，这样就可以明确每个输入码。否则，不加分断符的话，得到输入码符序列“ha”，就不能判定它是“h”开头的输入码和“a”开头的输入码所组成的词组呢，还是刚开始输“ha”开头的输入码，没来得及输入第二个输入码。这样做的目的是省略一个处理输入码序列的分析器。

用户使用词组编码输入技术的过程是这样的：

①用户敲入输入码符，当系统接收到空格符时，系统得到输入码 e_1，找出相应的同码汉字集 $f(e_1)$，提示给用户（可能要用翻页键才能把同码汉字提示完），如图 5-21 所示。

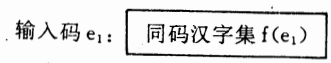


图 5－21 接受输入码 e_1 时的处理

②用户继续输入第二个汉字的输入码，当用户用空格符告诉系统输入码 e_2 已输入完成时，系统找出输入码 e_2 的同码汉字集 $f(e_2)$，在同码汉字集 $f(e_1)\times f(e_2)$ 中检索能构成词的元组，把这些词优先提供给用户，同码汉字集 $f(e_1)$ 和同码汉字集 $f(e_2)$ 的余下部分提示在词的后面。如图 5-22 所示。

③这时，用户可以用选择符来选择想要的词，处理类似于联想输入技术中所介绍的。一旦这时选择成功，那么就进入词联想处理部分。

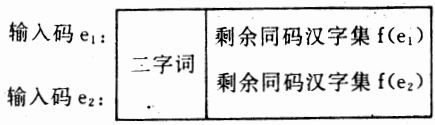


图 5－22 接受输入码 e_2 时的处理

④用户不进行③的操作，可以继续输入第三个输入码 e_3，则进行类似②的处理，如图 5-23 所示。

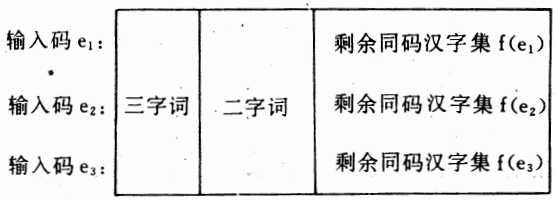


图 5－23 接受输入码 e_3 时的处理

⑤用户可以进入类似③的选择符处理，如果选择了 3 字词，就进入词联想处理部分，如果选择了 2 字词，则回到①的处理。

⑥用户不进行⑤的操作可以继续输入第四个输入码 e_4。以后类似③④⑤的处理。如图 5-24 所示。

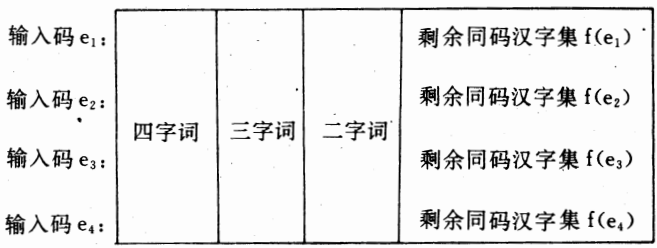


图 5-24 接受输入码 e_4 时的处理

⑦由于系统接受最长词的字数有所规定，如果还没有接收到处理掉最长词的字输入码的操作命令，就可以肯定用户在进行词组编码输入，于是，就要一直等到输入了词组的分断符，才进行有效互的处理。在数据库中按同码汉字集 $f(e_1)$，同码汉字集 $f(e_2)$，……，同码汉字集 $f(e_n)$ 去匹配找词组。为了避免系统瘫痪，接受输入码符当然不能是无休止的。

⑧这里的描述仅是就词组处理技术单方面的，前后链技术对①②都有影响。

2. 词组编码输入技术的实现

由于词组编码输入实现比较复杂，下面分几个部分介绍。

(1)词的生成算法　问题就是在状态空间 $f(e_1)\times f(e_2)\times\cdots\times f(e_n)$ 找出能与词库中词匹配的解来。这样的一个问题理论上可以是一个递归解法。但是，如果在实现中照搬的话，非但效率上极其低，其运行开销也大得不能忍受，因此，递归算法对系统来说是根本不实用的。一般规定只存在最长 4 字词，所以，分门别类地为二字词、三字词、四字词的生成设计实现还是可以接受的。原则是在硬件环境允许的范围内能够尽量快速地生成词。

(2)二字词的生成　二字词生成问题本身，简单得没有讨论的余地。但是，为什么递归算法的效率极差呢？这是因为在递归解法中，求解(*，*，*)规模问题时，还要重新求解(*，*)规模的问题。而事实上，因为输入码符是一个一个敲入的，当敲入输入码 e_3 时，关于 $F(f(e_1)\times f(e_2))$ 的问题已经求解过了，就可以利用这一问题求解中获得的信息，解决 $F(f(e_1)\times f(e_2)\times f(e_3))$ 的问题。这一点正是区别于递归解法的地方。

首先引入两个定义。

定义 1　全匹配

汉字序列 $h_1h_2\cdots h_n$ 能够从词库中生成，即词库所能描述的词和词组中有 $h_1h_2\cdots h_n$，则称 $h_1h_2\cdots h_n$ 关于该词 库是全匹配的，简称全匹配 $h_1h_2\cdots h_n$。

定义 2　子匹配

汉字序列 $h_1h_2\cdots h_n$ 能够包含在词库中的某个更长的词或者词组之中，则称 $h_1h_2\cdots h_n$ 关于该词库是子匹配的，简称子匹配 $h_1h_2\cdots h_n$。

关于 $F(f(e_1)\times f(e_2)\times\cdots\times f(e_n))$ 的解 $h_1h_2\cdots h_n$ 有两种情况：

①$h_1h_2\cdots h_n$ 是全匹配，则 $h_1h_2\cdots h_n$ 是真正的词(组)。

②$h_1h_2\cdots h_n$ 是子匹配，则要准备好 $h_1h_2\cdots h_nh_{n+1}\cdots h_m$ 是全匹配的状态空间。

关于①，也要有一种数据结构描述，在下面介绍。要存放②的状态空间，就要用下面描述的数据结构表示。由于通常规定最长是四字词，所以问题得到相当的简化。

如果 h_1h_2 是全匹配，那么 h_1h_2 就是输出给 op 的解；如果 h_1h_2 是子匹配，那么就把还未匹配的单字和双字分别形成一个队列，每个子匹配都对应这样两个队列，所有的子匹配形成一条链，称之为子匹配链，可以子匹配(例如 h_1h_2)表示链中的节点，如图 5-25 所示

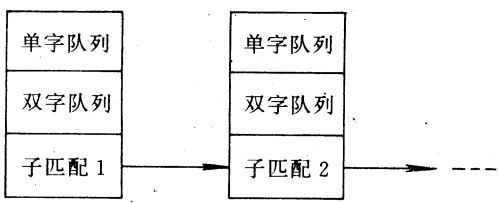


图 5－25　二字词子匹配链

显然，所有单字候选队列就是 $h_1h_2f(e_3)$ 为全匹配的状态空间，双字候选队列就是 $h_1h_2f(e_3)f(e_4)$ 为全匹配的状态空间的一部分。

经过上面准备后，可以用类 C 语言描述 $F(f(e_1)\times f(e_2))$ 的解，如下所示。

```
/* 获得输入码 e1 和输入码 e2 时的处理 */
produce2(e2)
{
  for(每一汉字 hi∈输入码 e1 的同码汉字集 f(e1))
      if(相应词头库表项内容非空){
          读出表项所指的词尾块号;
          for(每个在词尾块中的表项)
            if(汉字序列 h1h2…hn 的头 h1 在输入码 e2 的同
            码字集 f(e2)中)
            switch(汉字序列长度 n){
            case1:将 hi 和词尾段内容构到二字词全匹配链
                  中;
                  将词组联想索引字段内容存入节点中;
                  break;
             case2:将 hi 和汉字序列的头 h1 构到子匹配链
                  中;
                  将汉字序列的尾 h2h3…hn 排入相应单字
                  队列;
                  break;
            case3:将 hi 和词尾段内容构到子匹配链中;
                  将汉字序列的尾 h2h3…hn 排入相应双字
                  队列;
                  break;
            }
          }
}
```

关于全匹配链和它的内容，同时也是为词组编码输入实现而准备的，在下面叙述。注意，全匹配链就是生成的二字词的解。

(3)多字词生成　这里多字词生成是指三字词、四字词生

成，当然是在 produce2()完成的基础上生成。三字词生成实现用 produce3(e_3)表示，类似地，四字词生成实现用 produce4(e_4)表示。

三字词生成就是在二字词解的子匹配链中，遍历每个节点(h_1h_2)，相应单字队列和 f(e_3)的交集就是 produce3()的全匹配链，详细描述见后面内容。

四字词的生成，要在 produce3()完成的基础上，检索子匹配链上每个节点的单字队列在 f(e_4)上的投影，构成新的全匹配链。

注意，(e_1，e_2，e_3，e_4)序列也可能导致词组输出，同时 produce3()和 produce4()都要兼顾生成问题的求解，具体描述见下面。

(4)词组生成　现在要介绍全匹配的意义。全匹配链的节点可以用全匹配(例如($h_1h_2h_3$))表示。全匹配链是从词过渡到词组的桥梁，图 5-26 是二字词全匹配链的示意图。

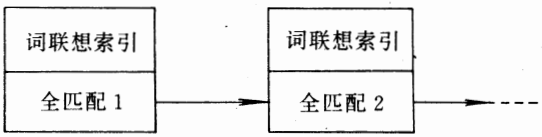


图 5－26　二字词全匹配链

三字词的生成实现中，要处理掉二字词的全匹配链，为以后词组生成作准备。produce3()的大致实现如下：

```
/* 获得输入码 e3 时的处理 */
produce3(e3)
{
for(二字词子匹配链的每个节点){
    将单字队列与同码汉字集 f(e3)的交集与本节点构成三字
      词全匹配链；
    将双字队列的首字与 f(e3)的交集与本节点构成三字词子
      匹配链；
    }
```

```
for(二字词全匹配链的每个联想块号非 0 的节点){
    读出该联想块;
    for(每个联想块中的表项)
      if(汉字 hi 在输入码 e3 的同码汉字集 f(e3)中){
      将该二字词全匹配键+hi 补充到三字词子匹配链中;
      取出该联想词,并将词尾补充到相应的字队列中;
        }
}
}
```

这样形成的三字词子匹配链中也包含了四字词和更多词组的信息。

produce4()类似于 produce3(),在此不再给出了。

需要指出的是,produce3()和 produce4()产生的子匹配链的节点所含的字队列,最多是三字队列,发生在下列情况同时成立时:

①汉字序列 $h_1h_2\cdots h_n$ 是子匹配;

②$h_1h_2\cdots h_{n-1}$是词组;

③$h_nh_{n+1}h_{n+2}h_{n+3}$是词;

于是,子匹配汉字序列 $h_1h_2\cdots h_n$ 就含有三字队列,其中一个元素是 $h_{n+1}h_{n+2}h_{n+3}$。

事实上,produce3()的实现就已经描述了词组生成的实现。对于 n 字词组的生成,是在 produce3 的第一个循环中,扫描处理的是 n-1 字子匹配链,第二个循环中,处理的是 n-1 字全匹配链。

(5)系统分词　如果在汉字序列 $h_1h_2\cdots h_{n-1}$的子匹配链和全匹配链中,对于输入码 e_n 的同码汉字集 $f(e_n)$不能产生任何子匹配链和全匹配链,那么就把 F_{n-1}也没有全匹配,在处理完 op_{n-1}后,将输入码 e_n 作为下一个输入单元。如果 F_{n-1}也没有全匹配,那再向前移动,直到全匹配为止。所以,基于词组智能化汉字输入系统的内部分词方法是最大匹配方法。但是这不同于自动分词方向的最大匹

配法。因此，此系统的词库可以自动增加，对同一个输入码序列，最大匹配方法给出的分词结果有可能不同。

(6)词组编码的注意点　从词组生成的实现来看，输入码 e 越含糊，同码汉字集 f(e)就越大，词组生成就越慢；输入码 e 越精确，词组生成越快，这一点也是自然的。如果能使 F 的状态空间较小，词组生成也就会快点，特别是输入码 e_1，它将决定后继的词组生成状态空间。越往后面，输入码对词组生成效率影响越小。所以，建议用户使用词组编码时，输入码 e_1 和输入码 e_2 尽量输得详细一点，至少输入码 e_1 应该详细一点。

在状态空间的生成中，可用动态申请内存的办法实现。当内存不够时，就中断词组生成操作，给出最近一次 F 结果，把断点处的输入码 e_i 作为输入码 e_1 继续处理下去。

六、前后链输入技术

对于 F 所研究的 $f(e_1)\times f(e_2)\times\cdots f(e_n)$，如果其中 $h_i\in f(e_i)(1\leqslant i\leqslant n)$已为用户所确认，从而使 F 能够更准确地判别 $h_1h_2\cdots h_n$ 汉字串，这种技术称为一阶前后链技术。相应地，如果用户已经确认了 $h_i\in f(e_i)$，$h_j\in f(e_j)(1\leqslant i,j\leqslant n,i\neq j)$，F 就能比一阶前后链技术更准确地判别 $h_1h_2\cdots h_n$ 汉字串，这种技术称为二阶前后链技术。依次类推，有 n-1 阶前后链技术。

前后链技术形式上是对词组编码输入和联想输入的扩充。

1. 对联想输入作用的实现

前后链输入技术对联想输入的作用，仅体现在输入码 e_1 的处理上。

当用户敲入第一个输入码的第一个输入码符后，如果之前系统刚好处在有联想域提示的情况下，那么输入码 e_1 的处理就要先进行对联想域的处理。把当前联想域内的个体的首字与同码汉字集 $f(e_1)$相交的那个联想个体提示在窗口的前部。有可能出现这样的情况，即用户实际上还未将输入码 e_1输入完毕，但提示窗口就提

示了用户想输入的词尾或词组了，这是由于用户进一步输入的信息使得联想域变小了，从而有可能使原先用户不愿意对联想域进行操作而失去的联想个体(在联想域后部)，又提示给用户了，这时用户就可以用引导链进行选择了。

需要指出的是，在这里输入的输入码 e_1 的地位等同于在联想处理中用控制操作查看联想域，但是，它们在本质上是完全不同的。敲入输入码 e_1 的操作是自然的，无损失的。一旦联想域中不存在与之相应的个体，因为用户下一个操作就是要输入 e_1，所以，从思路到实现进程中，这个过程是极其自然合理的。

2. 对词组编码输入作用的实现

在词组编码输入中，每个单字只输入了输入码 e_i 的一部分，相应地决定了同码汉字集 $f(e_i)$。本系统允许用户在输入一系列输入码时，回过头来确认 $f(e_i)$ 中的汉字 h_i，这样可以从词组编码实现中的匹配链中砍掉一大批节点，从而使 F 的全匹配解和今后要用的子匹配解大大地减少，F 判定的准确性也大为提高。假定词组编码输入正出于输入码 e_n 状态，前后链技术的实现要点如下：

①要用一个特殊引导键来中断词组编码输入操作，进入前后链处理。

②由引导键激活前后链处理后。询问用户希望在哪一个 $f(e_i)$ $(i\leqslant n)$ 中确认单字，即需要一个序号 i，使系统从保存的输入码序列中获得输入码 e_i。

③就输入码 e_i 进行单字输入处理。需要强调的是，既然用引导键进入了前后链处理，那么就要等前后链处理的一次过程完成才退出。所以这里前后链内部的单字处理与单字输入的实现的后果是不一样的，实现起来更加简单，只不过是单字输入实现的几个子功能块而已。以下说明中也有类似，不再另行说明。

④重新整理现有的匹配链，把与③确定的输入相矛盾的节点除掉。如果此时 $h_1h_2\cdots h_n$ 的全匹配链中节点只有一个，则自动将节点作为前后链处理的输出，本次处理完毕，如果此时汉字序列

$h_1h_2\cdots h_n$ 的子匹配链中有 $h_1h_2\cdots h_i$ 是全匹配的节点，则将 $h_1h_2\cdots h_i$ 作为本次前后链处理的输出，同时，作某些准备工作，将输入码 $e_{i+1}e_{i+2}\cdots e_n$ 的词组编码处理位移到输入码 $e_i\cdots$ 开始，退出本次前后链处理过程后，回到词组编码处理那儿去。

⑤如果④中的情况不发生，那么再循环进行②③的处理，直到④中情况的发生。

总而言之，上述基于词组的汉字输入的意义在于，它脱离了编码层次 f，是建立在编码层次之上的，可以直接为目前各种流行的键盘编码输入技术所享用。而且具备了以下特征：

①能根据当前的操作和信息，对将来可能发生的操作作一个猜测，这种猜测能随系统的运行而可靠到一定的概率程度；

②在当前的输入过程中，以前已输入的汉字信息能对当前的操作有所限制和帮助，有助于当前输入过程的迅速而简便地完成；

③对于同一种输入过程的反应，在不同的使用环境中有不同的响应，从而使系统体现出一定的自适应性。这就要求系统的自学习环节能够自动更新词库。

因此，我们说上述所介绍基于词组的汉字输入系统是具有智能型的。

第六节　键盘输入模块的标准化

一、问题的提出

由于汉字信息输入技术的多样性和用户需求的复杂性，我们要求键盘输入模块能够方便地进行扩充和修改，使其具有通用的或某种特定的输入技术，以满足各种用户的不同需求。这也就是要求键盘输入模块结构清晰友好、标准统一，便于对其进行再开发。

目前对基于 DOS 的汉字系统的键盘输入模块进行扩充的方法有三类。早期的做法是直接在键盘输入模块的目标代码上进行

修改，包括仅仅替换输入码对照表这样一种简单方法，这是第一类方法。第二类方法是将被扩充模块废弃，设计一个新的键盘输入模块，新模块中运用相应技巧，实现同被扩充模块的兼容。第三类方法是在分析了键盘输入模块的结构特点的基础上，采用“透明”的思想扩充原模块，这种方法对被扩充模块采取了若即若离的方式，效果较好。以上三类方法的缺陷在于：设计和实现这些方法本身，需要对目标系统有相当的了解，具体实现亦需要一定的技巧，并且调试难度较大。因此，较好地扩充键盘输入模块的手段不是一般程序员所能掌握的。

我们认为，基于 DOS 的汉字系统的键盘输入模块结构比较稳定，我们有可能且有必要实现该模块的标准化，以适应飞速发展的汉字信息输入技术对软件系统的要求。

二、键盘输入模块的模型

这儿所指的键盘输入模块是指具有汉字和西文信息输入能力的键盘输入模块，仅支持西文输入的键盘输入模块是由 ROM-BIOS 提供的，我们在下文中把它称为 ROMKEY 模块。

ROMKEY 模块的主要功能是把按键信息返回给调用者。我们可以方便地利用 ROMKEY 模块来扩展键盘输入功能，现在用一个例子来说明。假定我们需要把“A”键和“B”键的位置互换一下，也就是说，当按了“A”键后，应该得到“B”的信息，当按了“B”键后，应该得到“A”的信息。我们利用 ROMKEY 模块形成新的键盘输入模块 RAMKEY，其结构如图 5-27 所示，RAMKEY 模块可完成以上功能。

有了上述基础后，我们可以对 ROMKEY 模块提出进一步的扩充要求。比如我们要求，连续按两次“A”键后，才把“B”返回给调用者。于是，我们可以在 RAMKEY 模块中多次调用 ROMKEY 模块，截留其返回结果，并进行判断，当发现有两个连续的“A”返回时，才把“B”返回给调用者。我们把类似于这样的过程称为

RAMKEY 模块的消化分析过程。为了给 RAMKEY 模块确定不同的消化分析方式，往往要用各种功能键(或它们的组合)来定义

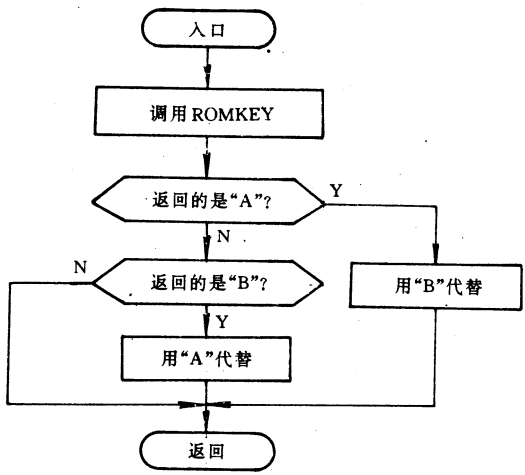


图 5-27.RAMKEY 模块

相应的消化分析方式。例如，我们定义，按 F1 键后，RAMKEY 模块将连续的两个"A"转义为"B"返回；当按 F2 键后，RAMKEY 模块将连续的三个"A"转义为"T"返回。我们还可以对 RAMKEY 模块的消化分析方式作这样的定义：当按 F3 键后，RAMKEY 模块将"A"转义为"5342"。显然，这样的转义在 RAMKEY 模块中是不难实现的，只要把"5342 保存好，当前先返回"5"，然后在以后的三次调用中依次返回"3"、"4"和"2"。

通过上述分析，我们可以把键盘输入模块抽象成如下模型：由某个功能键激活一个处理模块，该处理模块的工作是反复读取键盘上的输入信息，分析这些输入信息是否能转义为汉字内码，若能的话，则将汉字内码的多个字节代码返回给调用者。这个被激活的处理模块从形式上看是比较独立的。另外，由于汉字输入技术的多样性，故这个处理模块从内容上看是相对独立的。我们称这个处理模块为代码转换模块。

三、键盘输入模块的标准

由于代码转换模块是独立的，所以键盘输入模块就有可能分为主控模块和代码转换模块两大部分。这两个模块之间的关系是：主控模块激活代码转换模块，代码转换模块将转换得到的内码返回给主控模块。针对 ASCII 输入方式的代码转换模块是一个特殊的代码转换模块，它对代码未作任何转换。这是因为在这种输入方式下，不需要对 ROMKEY 模块的结果作任何转义。

由于汉字信息处理的特殊性，汉字系统中往往设置了一些一次性激活处理的模块，这些模块常常是为了进行人机交互。汉字系统往往要同用户就一些系统设置问题进行会话，用户藉此来控制汉字系统的行为。但是，这类交互是暂时的和一次性的，在这类交互中所占用的资源必须在交互结束后释放掉。并且，这类交互工作不能被打断，交互结束后，系统又退回到交互前的状态。例如，我们用 Ctrl＋F10 键来激活有关打印机参数设置的处理模块，对打印机参数的设置处理即以上所述的人机交互过程。我们也可以把这一类处理模块看作是一种特殊的代码转换，只是其转换结果并无实际内容返回给主控模块，主控模块所获得的激活定义键就被该模块消化(解释)，并体现为与用户发生的一系列交互活动和自身的设置工作。

在系统内，功能键与代码转换模块有一确定的对应关系。我们把这些代码转换模块的入口地址按其对应功能键的序列构成一张地址表，把它存放在一个固 定的位置。这张地址表不但能让主控模块和代码转换模块访问 到，而且还要能让用户程序访问到。例如，我们可以把该表放在键盘输入模块所在段的 0 偏移处。

键盘输入模块的主控模块的设计可以按照图 5-28 进行。

在图 5-28 的主控模块中，“根据当前工作模式计算地址表偏移”一框内的“当前工作模式”，是指最近一次定义键所激活之处理模块对应的模式，这里必须是真正的能将汉字输入码转换为汉字

内码的代码转换模块所对应的工作模式。主控模块在调用有关的代码转换模块后，将会获得被调用的代码转换模块的返回值。若返回值为 0，则表示代码转换模块已经消化处理掉了定义键，实际并未作实质性的输入工作；若返回值为 1，则表示代码转换模块有 1 个字节返回给主控模块；若返回值为非-1，则实为多字节返回，其处理类似于返回值为 1 的情况；若返回值为-1，则为未安装的代码转换模块的返回，其处理仅为一条空返回语句。注意，这样的处理是存在的。

键盘输入模块的代码转换模块的设计可以按照图 5-29 进行。该图中给出的代码转换模块的一般流程。

在图 5-29 的代码转换模块中，“多字节返回”即返回值为非-1，它当然不限于只返回 1 个汉字。特别要说明的是，在图 5-29 中没有出现返回值为 0 的情况，其实在“做一些初始化工作”一框内把这一种情况包括了。对于一些仅作交互工作的模块，这一框就处理所有的交互工作，作完后使返回值为 0，然后返回主控模块。因此，图 5-29 给出的是能产生汉字内码的代码转换模块的流程。

最后再介绍一些辅助性质的设计安排。最重要的有两点，一个是 ROMKEY 模块也应该在键盘输入模块中向外开放，比如可以把它定义为键盘输入模块内的一个功能块。另一个是能将外部提供的代码转换模块的入口地址，根据用户要求自动填入地址表内，从而实现外部输入码与系统的连接。这项工作可以由外部代码转换模块在自举时借助于键盘输入模块的有关功能块完成，所以，键盘输入模块中必须具备能完成这一功能的功能块。

四、提示行窗口标准化

键盘输入模块与用户进行交互的机会较多，交互的环境通常就是提示行。由于提示行还是早期汉字系统使用的概念，我们在这里要把这个概念扩展一个，称为提示行窗口。这个窗口就未必只有一行，也未必仅用于汉字输入。

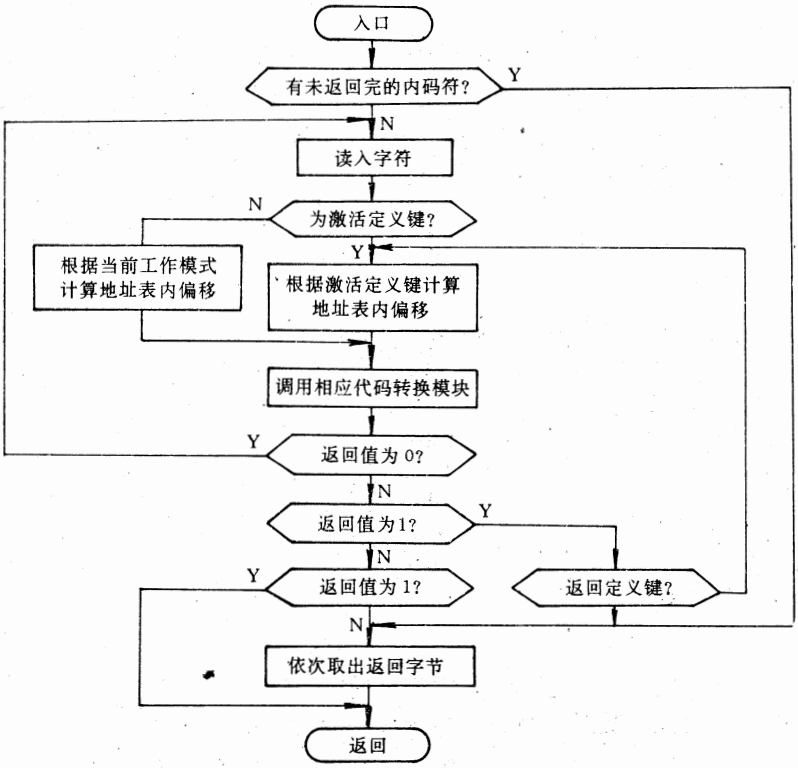


图 5-28 主控模块的流程

提示行窗口在本质上是窗口，可以用坐标(sx,sy,ex,ey)来定义，窗口即为左上角坐标(sx,sy)和右下角坐标(ex,ey)所确定的矩形区域，一般用于与用户进行一些有关的汉字输入/输出的对话。提示行窗口的功能通常由支持汉字显示的汉化视频 BIOS 支持。就目前常用的汉字系统而言，CC-DOS 4.0 把提示行功能定义为其视频 BIOS 的 20 号功能块，2.13 系统把提示行功能定义为其视频 BIOS 的 16 号功能块，UC-DOS 则索性从功能到接口定义自成一格。这种提示行窗口管理各行其事的现状，显然不利于汉字输入

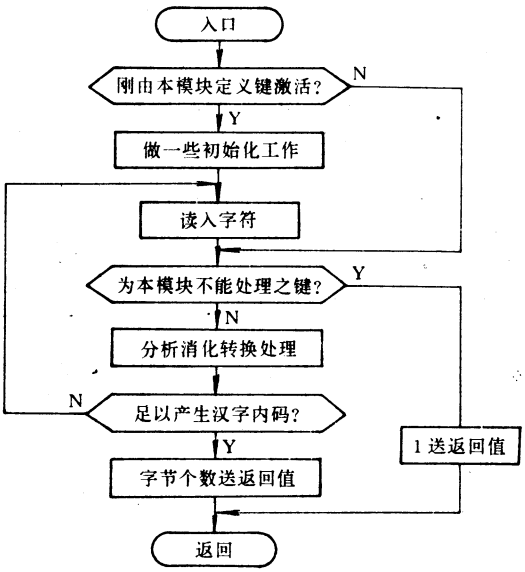


图 5-29 代码转换模块的流程

技术的交流。提示行窗口的标准化问题急待解决。

首先要解决的是提示行窗口用户界面的标准化问题。把它们定义在视频 BIOS 中所用的功能块号是一个潜在的问题。由于中西文兼容是汉字系统要达到的重要目标之一，汉化视频 BIOS 也要支持原西文视频 BIOS 所提供的功能。计算机硬件技术的飞速发展使视频 BIOS 支持的功能越来越多，因此提示行功能块块号易与西文视频 BIOS 的功能块块号发生冲突。这时，只好改变提示行功能块的块号。如果把提示行功能块定义在视频 BIOS 的高功能块号，那也未必能令人放心，谁也不能保证将来推出的西文视频 BIOS 不占用这些功能块块号。因此，似乎把提示行模块单独定义成为一个独立的中断处理程序为好(UC-DOS 就采用这种作法)。

其实，这种作法也不保险。虽然我们可以用扫描中断向量表的方法，保证使提示行模块所用中断号不与操作系统发生冲突，但是无法保证丰富的西文应用软件与提示行模块所用中断号不发生冲突。所以，我们认为最安全的做法是把提示行模块固定在汉化视频BIOS内的指定位置(如该段的0偏移处)。这样，系统程序和应用程序均可通过段间调用指令实现对它的调用。

我们还必考虑提示行窗口占用的物理资源。根据提示行的使用特点可知，提示行没有必要永久性地占用物理屏幕的固定区域，它只不过是一个弹出式(POP-UP)窗口。用户在进行汉字输入的操作时，或者出于其它目的正与系统在提示行窗口进行对话时，一般是不关心已启动的应用程序的状态，这时只要尽量保证提示行窗口不开设在应用程序的光标附近就可以了。在提示行窗口不被使用时，可以隐去它。当然，这里并不是说在物理条件许可的情况下硬要用提示行窗口去侵占用户屏幕，如果能够在满足面向用户的屏幕之外建立窗口环境，那么就不必再去干扰用户屏幕了。

最后让我们来讨论一下提示行窗口功能的标准化问题。提示行窗口的使用面相对比较狭窄，这就使提示行所支持的功能得到相应的简化。提示行窗口的功能一般定义为以下五个功能就可以了，它们分别是：开窗口，窗口光标定位，在窗口当前光标处写字符，窗口TTY方式写字符，隐去窗口。再考虑到目前已流行的汉字操作系统及相当数量的相关系统软件，提示行窗口的功能及其入口参数定义如下：

(1)0号功能：开窗口。给出左上角和右下角的坐标，在屏幕上划出一块区域，填入一定属性的空格。

入口参数：CH＝左上角横坐标；

CL＝左上角纵坐标；

DH＝右下角横坐标；

DL＝右下角纵坐标。

(2)1号功能：写字符。在窗口当前光标位置显示指定个数的

带属性的字符。

入口参数:DL=显示字符的内码;

CX=显示字符的个数;

BX=显示字符的属性。

(3)2 号功能:光标定位。把窗口光标定位在指定坐标处,坐标以窗口左上角为相对基准,从 0 开始计算。

入口参数:DH=窗口内行号;

DL=窗口内列号;

(4)3 号功能:TTY 写字符。以 TTY 方式在窗口当前光标位置写一个指定属性的字符。

入口参数:DL=显示字符的内码;

BX=显示字符的属性。

(5)4 号功能:隐去窗口。恢复原有的被窗口覆盖的内容。

五、小结

我们在上面提出的键盘输入模块标准具有内外一致和高低一致的特点。内外一致是指系统内部输入模块和外部扩充的输入模块从本质上统一起来了,从而也就没有内外之分,仅仅是这些输入模块入口地址的填换而已。高低一致是指高级语言和低级语言能够统一运用于汉字输入模块的开发,使这类开发工作变得比较简便。另外,这种键盘输入模块的结构又使系统层和应用层统一了起来,开发汉字输入模块的工作就成为应用层次的工作,等到模块开发成功以后将其入口地址填入地址表就可以了。这样,汉字输入模块的开发工作在方法上就变得十分简便易行。

第六章　汉字输出处理

第一节 VGA 显示输出模块的设计

一、VGA 概述

PC 系列机可以配用多种显示适配卡，这些显示卡中以 VGA 卡最为典型，目前购买的 PC 系列机都配置 VGA 卡。另外，VGA 卡的标准分辨率为 640×480，完全有可能在屏幕实现 25 行汉字的显示，这就使汉字屏幕格式与西文屏幕格式完全一致了。

VGA(Video Graphics Array)是 IBM 公司随 PS/2 系列机推出的一种新的视频标准。它最初作为 PS/2 系列机的视频子系统，由系统板上的专门硬件实现。现在，PC 系列机也已广泛采用 VGA 视频标准，并制成实现 VGA 视频标准的显示卡，称为 VGA 卡。从功能和特性来看，VGA 基本上是 EGA(Enhanced Graphics Adapter)的改进型版本。但是，在视频显示模式的分辨率方面，VGA 的灵活性要远大于 EGA，特别当使用可变频率的显示器时，则更是如此。

VGA 视频子系统是一个专门的 VLSI 门阵列电路，它包括下列控制部件：CRT 控制器(CRTC)、定序器、属性控制器和图形控制器，以及视频数模转换器(DAC)。我们可以直接对这些控制部件编程，以改变视频子系统内的基本定时信号和定址模式。VGA 中的 ROM－BIOS 的 10H 号中断处理程序之 0 号功能块含有一组例行程序，可以利用它们对 VGA 控制部件进行编程，把 VGA

配置成 24 种不同的视频模式。图 6-1 给出了 VGA 视频子系统的结构，从该图中可以知道各个控制部件间的关系。

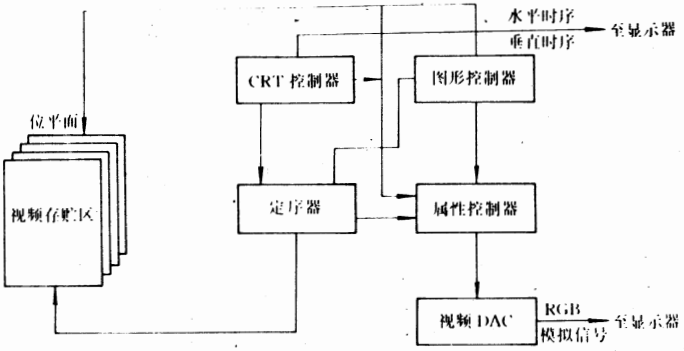


图 6-1 VGA 视频子系统的结构

VGA 的每个控制部件都含有若干个寄存器，用于对控制部件的控制。这些寄存器被映射到 I/O 口空间。VGA 对这些寄存器的访问方法是：先把欲访问之寄存器的序号写入一个 I/O 口（索引口），再在另一个 I/O 口（数据口）对指定寄存器进行读或写。CRT 控制器有 25 个寄存器，它们的索引口和数据口地址分别为 3D4H 和 3D5H；定序器有 5 个寄存器，它们的索引口和数据口地址分别是 3C4H 和 3C5H；图形控制器有 9 个寄存器，它们的索引口和数据口地址分别是 3CEH 和 3CFH；属性控制器有 5 个寄存器，视频数模转换器有 16 个寄存器，它们的索引口和数据口地址分别是 3C0H 和 3C1H。

VGA 的 ROM－BIOS 中有一张参数表，表中含有各种视频模式下各个寄存器所需之值，只要调用 ROM－BIOS 就能选择所需要的视频模式，而不必去直接修改寄存器之值。但是，如果要建立

VGA 的 ROM－BIOS 所不支持的视频模式，则必须知道对应于这种视频模式的各寄存器之值，然后才能配置成这种视频模式。在某些情况下，有必要建立一些 VGA 的 ROM－BIOS 所不支持的特殊视频模式。例如，若要求获得比 ROM－BIOS 所支持的视频模式更高的分辨率时，就要配置新的视频模式。

二、设计思想

VGA 提供了较高的显示分辨率，它的 12H 号视频模式是 640×480 分辨率的图形模式，并且可以支持 16 种颜色，这为我们设计出功能更加完美的汉字显示输出模块提供了基础。我们用 17 根扫描线来显示一行汉字，其中 16 根线用于显示汉字字形，还有 1 根线用作行间隔。这样，当处于 12H 号视频模式时，每帧共 480 根扫描线，故可以显示 28 行汉字。另外，因为每根扫描线有 640 个像元，一个汉字横向要占用 16 个像元，故每行可以显示 40 个汉字。因此，每帧可以显示的汉字总数为 40×28＝1120 字。为了使设计出来的显示输出模块与西文系统兼容，我们规定，屏幕的 28 行中有 25 行用作屏幕正文区，另外 3 行用作提示行。这种屏幕格式不但能与西文系统兼容，而且与国产长城系列机的 CEGA 卡和 CVGA 卡的屏幕格式相同。

在设计 CGA 卡的显示输出模块中，为了使屏幕的 10 行正文区能反映 25 行的内容，故设立了虚屏存贮区 vram。在 VGA 中，屏幕的正文区已达到 25 行，是否还需要设立虚屏存贮区呢？我们认为，仍然要设立虚屏存贮区 vram，因为要在图形工作方式下仿真字符方式工作，必须要记录屏幕显示内容的字符代码和属性字节，只有这样才能与西文系统兼容。所以，我们把虚屏的概念又引入 VGA 的显示输出模块。

VGA 的 12H 号视频模式具有 16 种颜色，在设计中必须充分注意到这一点，特别要注意对字符属性字节的处理。这样，显示输出模块在显示汉字时，亦能支持 16 种颜色。我们在设计虚屏存贮

区时，也要注意到这一点。我们把虚屏存贮区设计成三个分区，它们是字符分区（存放字符内码或汉字内码）、属性分区（存放属性字节）、标志分区（存放配对标志）。在 CGA 的显示输出模块中没有设置标志分区，配对标志是挪用属性字节中的位来实现的。在 VGA 中不宜这样作，应保持属性字节的完整，故另外设立了标志分区。这三个分区均为 80×28＝2240 字节，它们中的字节是一一对应的，对这三个分区的访问也应该同步进行。

因为 VGA 的屏幕具有 25 行汉字正文，所以实屏与虚屏之间的关系就十分简单（为一一对应关系），因而简化了显示输出模块的设计，特别是光标定位和屏幕滚动部分。我们在了解了 CGA 的显示输出模块后，再来讨论 VGA 的显示输出模块，会明显感到后者比前者简单，而且两者有不少地方是相同或类似的。下面我们将重点介绍 VGA 显示输出模块设计中的独特部分。

三、光标功能的实现

在字符工作模式下，光标由 CRT 控制器形成，在图形工作模式下，光标由显示输出模块产生。我们把光标图形确定为一根下划线，它处于每行的第 16 根扫描线上（即作为行间隔的那根扫描线）。我们设计一个子程序来产生或删除光标，其入口参数为 flag，当其为 FFH 时表示建立光标，当其为 00H 时表示删除光标。图 6-2 为光标产生/删除子程序的流程。

我们从图 6-2 可知，仅当自动光标标志置位时，该子程序才实现光标的产生或删除。子程序利用下列公式算出光标图形在视频存贮区 rram 中的地址 raddr：

$$raddr = 行号 \times 80 \times 17 + 列号 + 16 \times 80$$

VGA 的视频存贮区不分奇数线区和偶数线区，故其每行扫描线数实取 17。另外，因为光标图形在行内第 16 根扫描线上，故最后要再加上 16×80，这两点在以上公式中已经体现出来了。因为屏幕可以显示 16 种颜色，即屏幕上每个字符的前景色和背景色均可以

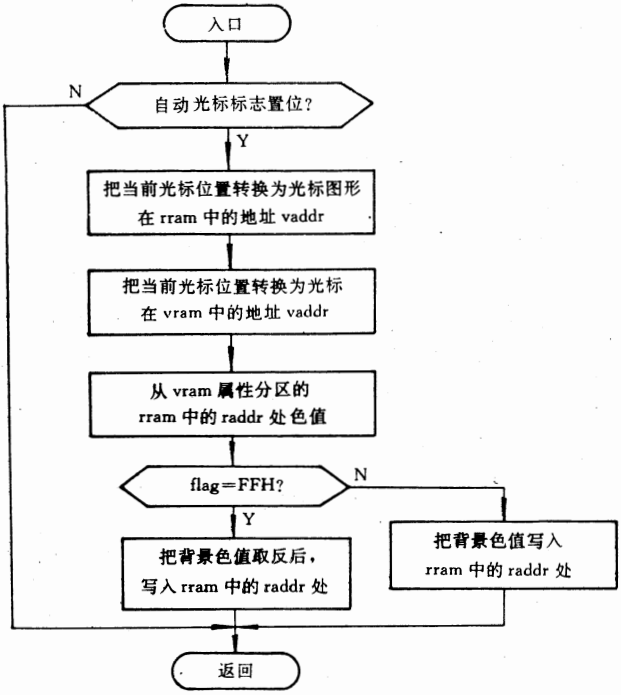


图 6-2 光标产生/删除子程序的流程

不相同,光标图形的显示与背景色有关,所以要知道光标位置处的背景色。vram 的属性分区中的属性字节记录了背景色值,故要算出光标图形在 vram 内的地址。我们用下列公式计算光标图形在 vram 内的地址 vaddr:

$$\text{vaddr} = \text{行号} \times 80 + \text{列号}$$

子程序根据 vaddr 从 vram 的属性分区取得背景色值。若要删除光标,则直接把背景色值填入 rram 中 raddr 处的 8 个像元;若要产生光标,则要把背景色值取反后填入这 8 个像元。

在实现光标的产生和删除的基础上,我们讨论光标的定位问题,即 2 号功能块的设计。图 6-3 给出了 2 号功能块的流程。

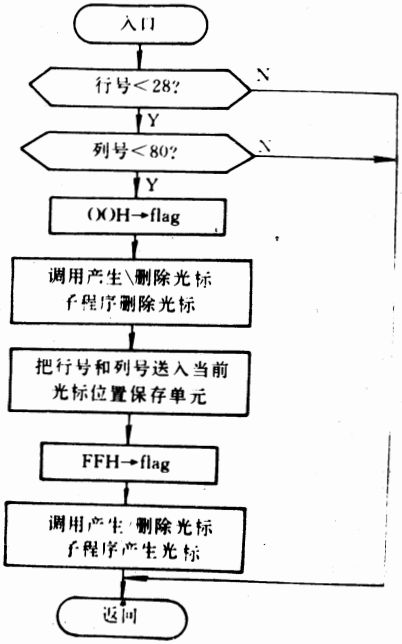


图 6-3 2 号功能块

在图 6-3 给出的 2 号功能块流程中，先要判别光标行号和列号的合法性，对于非法的行号和列号则不予处理。然后，首先把原光标删除，再修改当前光标位置单元之值，最后形成新的光标。显然，这要比 CGA 的显示输出模块中的光标定位简单得多，它不再需要作映像区的上浮或下沉处理。

四、汉字和字符的显示

1. 在光标处显示汉字和字符

显示输出模块的 9 号和 10 号功能块实现在当前光标位置上显示汉字和字符，由用户给出字符代码(ASCII 码或汉字内码)和属性字节(指出字符颜色属性)。VGA 显示输出模块的这两个功能块与 CGA 的相应部分类似，只是具体实现细节上稍有不同。所以，我们只讨论这部分实现细节，对其它部分不作讨论。

我们知道，不管是显示汉字，还是显示字符，都要把字模信息送入字模缓冲区内排好，汉字字模占用整个字模缓冲区，字符字模

占用前半个字模缓冲区。当显示汉字或字符时，则把字模缓冲区中的内容送入 rram 相应处，即实现了汉字或字符的显示。我们设计一个子程序完成这项工作，该子程序称作字模写入 rram 子程序。它的入口参数为 bufpoint，它指向被写入字模信息的首地址，该子程序就把 bufpoint 指向处的 16 个字节写入光标所在的 rram 地址。显示字符只要调用该子程序一次，显示汉字则要调用该子程序两次。

字模写入 rram 子程序的实现与 VGA 的特性有关，所以我们有必要讨论这个子程序的设计。图 6-4 给出了字模写入 rram 子程序的流程。

这个子程序首先根据光标当前位置算出光标在 rram 内的地址 rramddr，计算公式如下：

rramddr＝行号×80×17＋列号

接着再算出光标在 vram 内的地址 vaddr，并根据此地址从属性分区中取出背景色值和前景色值。在彩色图形模式下，向 rram 写入字模时，既要写入前景色，又要写入背景色，这一点与在黑白图形模式下不同。接下来，子程序为了完成把字模的一个字节写入 rram，它先根据 bufpoint 取字模的一个字节送入 dotbyte，并把它作为分离字，把前景色写入 rramddr 处的 8 个像元；然后，子程序把 dotbyte 取反后作为分离字，再把背景色写入到 rramddr 处的 8 个像元。这样就使 rramddr 处的 8 个像元之值为：对应于 dotbyte 中值为 1 的位的像元为前景色值，对应于 dotbyte 中的值为 0 的位的像元为背景色值。这就达到了我们的目的。接着，子程序调整 bufpoint 和 rramddr 后，再重复以上过程，直到把 16 个字节全部写入 rram 为止。

另外，我们还要指出一点，VGA 的视频 BIOS 中向用户提供了 8×16 的字符字模，可以通过对 43H 号功能块的调用来取得。因此，我们在显示输出模块中可以不再使用 8×8 的字符字模。这样作后，显然有利于提高西文字符的显示输出质量。

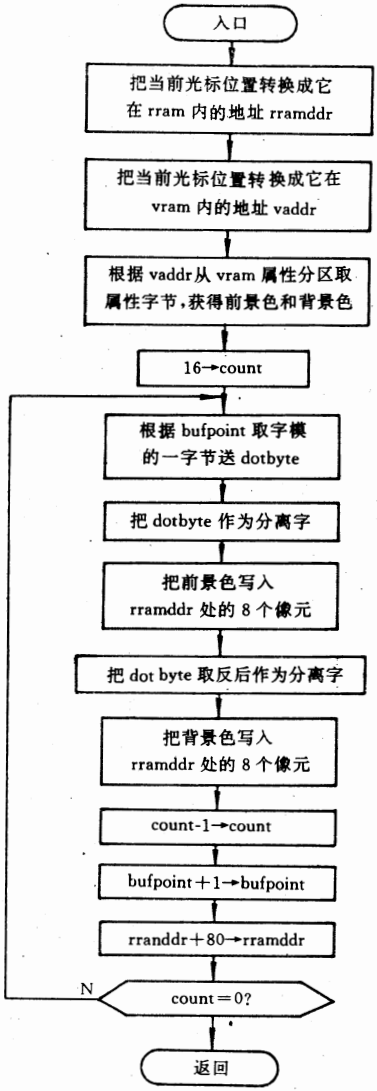


图 6-4 字模写入 rram 子程序

这儿不再对它们作介绍了。

2. TTY 方式显示

显示输出模块的 14 号功能块实现汉字和字符的 TTY 方式显示，即在当前光标位置上显示汉字或字符，然后把光标移至下一个显示位置。VGA 显示输出模块的这个功能块与 CGA 的相应部分基本类似，只是在换行处理上有所不同，而且是大大简化了，主要原因是它具有 25 行的实屏。

如果当前字符显示在行末，或者接收到了换行符，则均要进入换行处理。我们是这样设计换行处理程序的：如果当前行号小于 24，则把行号加上 1；如果当前行号为 24(正文区最后一行)，则把屏幕内容(指正文区内容)上滚一行，行号不变；如果当前行号大于 24(在提示行区)，则不作处理。

这个功能块的其它部分与 CGA 的显示输出模块的 14 号功能块相同，故

五、屏幕滚动和提示行

1. 屏幕滚动的实现

显示输出模块的 6 号功能块和 7 号功能块分别实现屏幕的上滚和下滚。VGA 显示输出模块的这两个功能块与 CGA 的相应功能块基本相同，它也是根据入口参数确定滚动窗口和滚动区在 vram 和 rram 中的位置，先滚动 vram 各个分区的内容，再滚动 rram 中的内容，分别完成虚屏和实屏的同步滚动。由于 VGA 屏幕有 25 行正文区，所以其滚动窗口和滚动区的大小及位置，在虚屏和实屏中是一致的。因而可以不必根据滚动窗口和映像区的关系，重新确定实屏中的滚动窗口和滚动区位置，这样也就使程序得到了简化。这两个功能块的其它部分均与 CGA 显示输出模块相同。值得指出的是，只能滚动屏幕正文区内的内容，提示行中的内容不能参与滚动。

2. 提示行的实现

显示输出模块的 255 号功能块实现对提示行进行管理，VGA 显示输出模块的这个功能块与 CGA 显示输出模块基本相同，其子块功能的划分也完全一样。由于 VGA 屏幕有三行提示行，而且它具有彩色功能，因此它在 255 号功能块中有一些独特的地方，下面对此作一简单的介绍。

在清除提示行内容时，应该向提示行的三行内填入具有一定颜色属性的空格符，以使提示行的背景色与正文区的背景色有所区别，从而使提示行更加醒目。可以调用 9 号功能块完成这一工作。当然，也可以通过直接向提示行所在的 rram 直接写入同一值的像元来实现。另外，在实现提示行光标定位时，应先将入口参数给出的光标位置线序号转换成行号和列号，然后再调用 2 号功能块完成光标定位。

六、窗口管理

1. 总述

在CGA的显示输出模块中，没有提供窗口管理功能，这是因为它的屏幕分辨率太低，总共才能显示11行汉字，不适合再开设窗口。VGA的屏幕具有较高的分辨率，共可显示28行汉字，这为窗口的开设提供了物理基础。大家知道，利用窗口功能可以使软件具有友好的用户界面，而且能使设计出来的软件效果更佳，因此，VGA的显示输出模块中有必要引入窗口管理功能。为此，我们增加了一个254号功能块，它提供了一系列的窗口管理操作。下面，以不嵌套的弹出式窗口为例，介绍窗口管理模块(254号功能块)的设计。读者可以在此基础上进一步设计出能对更复杂的窗口进行管理的模块。

2. 数据结构

为了保存窗口中显示的内容，我们设立了一个窗口存贮区，该区的大小与可开设窗口的大小有关。考虑到整个屏幕的正文区为80×25＝2000个字符，现规定可开设的最大窗口面积为其一半，即1000个字符。窗口存贮区的结构与虚屏存贮区vram相同，它也分为三个分区，即字符分区、属性分区和标志分区，分别存放窗口内显示字符的代码、属性字节和标志字节。这三个分区中的内容是一一对应的，对这三个分区的访问必须同步进行。

另外，我们还设立一个窗口参数保存区，其中保存当前开设窗口的左上角和右下角的坐标(行号，列号)。根据这两个参数，我们可确定窗口的位置和窗口的大小(行数和列号)。另外还设立了一个窗口标志wflag，该标志置位，表示已开设窗口，该标志复位，表示未开设窗口。

3. 模块结构

窗口管理模块(254号功能块)由五个子块组成，每个子块完成一项窗口功能。这些子块的情况如下：

①0 号子块:实现窗口的开设;

②1 号子块:在窗口光标处显示字符;

③2 号子块:实现窗口光标的定位;

④3 号子块:在窗口内以 TTY 方式显示字符;

⑤4 号子块:实现窗口的撤销。

4. 窗口的开设

窗口的开设由 0 号子块实现,我们把这个子块的入口参数定为:CX=窗口左上角的坐标(行号,列号),DX=窗口右下角的坐标(行号,列号),BH=窗口区充填符的属性字节(背景色值)。开设窗口就是把对应于窗口区的视频存贮区中填满 BH 寄存器内给出的背景色值,即在窗口区充满具有 BH 寄存器指定属性的空格符。图 6-5 给出了 0 号子块的流程。

从图 6-5 中可以看到,0 号子块首先判别入口参数中给出的左上角和右下角坐标的合法性,如果它们没有超出正文区的范围(行号为 0～24,列号为 0～79),则为合法的。对于非法的坐标值,则不予开设窗口。然后,0 号子块判别窗口面积是否超出最大许可值(1000 个字节),若超出此值,则不予开设窗口。接着,0 号子块把窗口标志 wflag 置位后,就把窗口左上角坐标送入窗口光标位置单元,以便使窗口光标定位在窗口的左上角。该子块接下去就对窗口存贮区的三个分区进行初始化,然后反复调用 1 号子块在窗口区中显示满具有指定属性的空格符,最后调用 2 号功能块在窗口左上角建立窗口光标。

5. 光标定位

窗口光标的定位由 2 号子块实现,我们确定这个子块的入口参数为:DX=窗口光标的坐标(行号,列号),这里的行号和列号是窗口内的行号和列号。窗口光标定位的实质是修改窗口光标位置保存单元之值,这个单元中记录了当前窗口光标的位置,它是用屏幕内行号和列号来记录的。2 号子块的流程如下:

①判别入口参数给出的光标是否超出窗口范围,若超出的话,

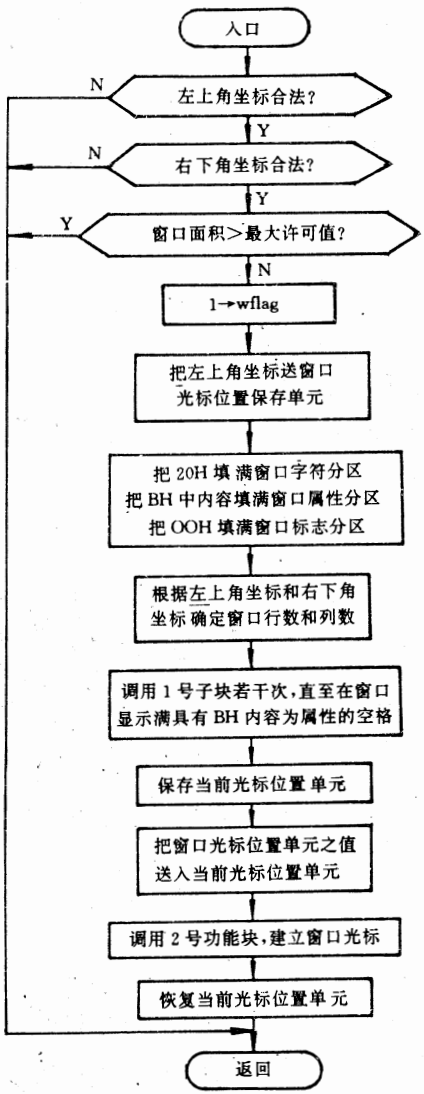


图 6-5 0 号字块的流程

则不予定位;

②保存当前光标位置保存单元之值;

③把窗口光标位置保存单元之值送当前光标位置保存单元;

④把入口参数中的窗口内行号和列号,分别加上窗口左上角的行号和列号,形成屏幕内行号和列号;

⑤调用 2 号功能块,完成光标定位;

⑥把当前光标位置保存单元之值送窗口光标位置保存单元;

⑦恢复当前光标位置保存单元之值。

6. 字符的显示

窗口管理模块的 1 号子块和 3 号子块分别实现在窗口光标位置显示字符和 TTY 方式显示字符。1 号子块的入口参数为:DL=字符代码,BL=属性字节,CX=重复显示次数。3 号子块的入口参数为:BL=属性字节,DL=字符代码。这两个子块的设计分别与 9 号功能块和 14 号功能块的设计类似,

因此我们可以参照这两个功能块进行设计，并能调用这两个功能块中的不少内容来完成有关功能。所以，这儿不再对这两个子块的设计进行讨论。

7. 窗口的撤销

窗口的撤销由 4 号子块实现，这个子块没有入口参数，该子块所要完成的主要工作是恢复被窗口所覆盖的屏幕内容，这些内容可以从虚屏中相应位置的内容中转换得到。我们设计的 4 号子块的流程如图 6-6 所示。

前面已经说过，4 号子块的主要工作是恢复被窗口所覆盖的屏幕内容，在此恢复过程中必须要处理好窗口边界的汉字问题。开设窗口是随意进行的，有可能把窗口边界正好设在一个汉字的当中，即把半个汉字的位置放在窗口外，把另外半个汉字的位置放在窗口内。这样，在撤销窗口时，就要恢复窗口内的那半个汉字。我们知道，单独的一个汉字内码符是没有意义的，必须要两个汉字内码符合起来才有意义，才能形成一个汉字内码和确定一个汉字。所以，在恢复窗口内半个汉字时，必须把 vram 中窗口内的那个汉字内码符，与窗口外的那个汉字内码符合成一个汉字内码，然后在相应处显示此汉字，就等于恢复了那半个汉字。从以上所述可知，这就要解决汉字内码符的配对问题，必须表示出哪两个汉字内码符是配对的。这可以借助于 vram 中的标志分区，在 VGA 显示输出模块中增设了这一分区，vram 中的每个字符代码均可有一个标志字节作说明。我们对标志字节作如下定义：对于 ASCII 字符，其标志字节为 00H；对于未配对的汉字内码符，其标志字节为 80H；对于已配对的汉字内码符，则汉字内码的首内码符的标志字节为 01H，次内码符的标志字节为 02H。这样，我们就可以根据标志字节之值，识别出对哪些汉字内码符要进行边界处理。显然，对于位于窗口左边界的其标志字节为 02H 的汉字内码符，以及位于窗口右边界的其标志字节为 01H 的汉字内码符，均要作边界处理。

图 6-6 中的流程所采用的方法是，取 vram 中对应于窗口之区

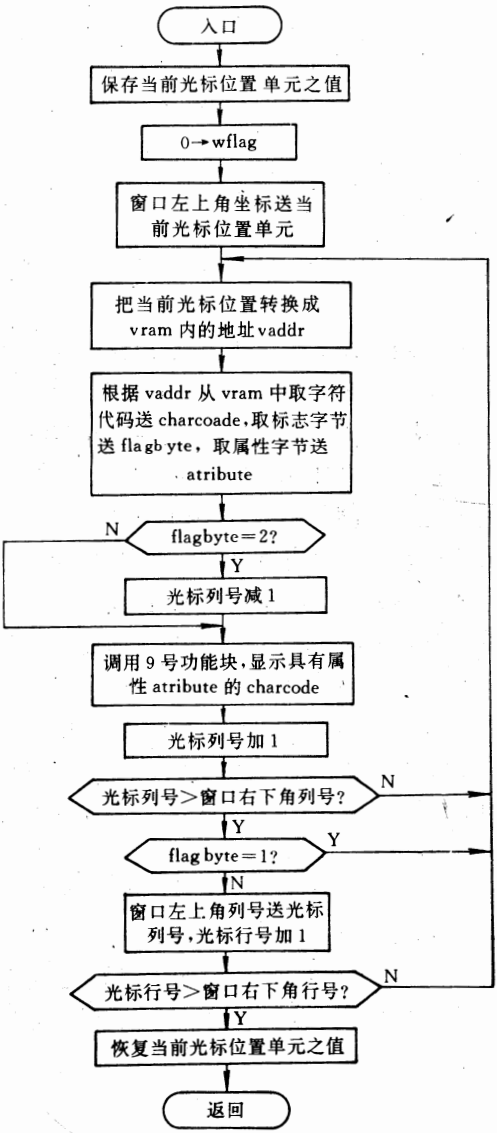


图 6-6 4 号字块的流程

域中的字符代码，通过对 9 号功能块的调用，把其对应字符或汉字在屏幕窗口区相应位置显示出来。我们只要对 vram 的这个区域中的所有字符代码均作以上操作，就实现了恢复屏幕窗口区内容，即撤销了窗口。在上述过程中，在处理到窗口边界处的字符时，则要根据其标志字节确定是否要作边界处理。若要进行边界处理，

则要把它与窗口外的相应字符代码合为汉字内码，然后显示此汉字。根据以上所述内容，不难理解我们所设计的 4 号子块的流程。

七、窗口管理对模块的影响

显示输出模块中增加了窗口管理功能后，会影响到模块中有关功能块的工作，因此，必须对这些功能块的流程作相应的扩充和调整，以使它们能与管理部分协调地工作。下面，我们就来讨论对有关功能块的调整问题。这些功能块包括 0 号、2 号、6 号、7 号、9 号和 10 号功能块。

1. 对 0 号和 2 号功能块的调整

我们希望窗口开设后，在其撤销之前，应该保持在屏幕上，即使是在屏幕重新设置工作方式后，窗口仍应存在。所以，应该在 0 号功能块的末尾加入以下内容：判别窗口标志 wflag 是否置位，若置位的话，则根据窗口参数保存区中的窗口左上角和右下角坐标，调用 254 号功能块的 0 号子块开设窗口，再调用 3 号子块把窗口存贮区中的内容送窗口区显示。这样就又重新建立了跟原来一样的窗口。

引入窗口管理后，对实现光标功能的 2 号功能块也要作些调整。我们希望当屏幕光标在窗口区时，仍能保持光标图形，而且仍然可以被删除。因此，就要对 2 号功能块中的产生/删除光标子程序作以下调整：当产生或删除光标时，先判别光标是否在窗口区，若在窗口区，就从窗口存贮区的属性分区相应处取得背景色值；如果光标不在窗口区，就从 vram 的属性分区相应处取得背景色值。然后再按原来的方法对取得的背景色值进行处理。这样处理后，就可以保证光标在窗口区内的产生和删除。

2. 对 6 号和 7 号功能块的调整

在开设窗口后，我们希望在进行屏幕滚动时，不影响窗口的位置和内容，同时要求在滚动范围内的被窗口覆盖的内容能正常滚动。若不对 6 号和 7 号功能块作调整，则不可能达到以上要求。

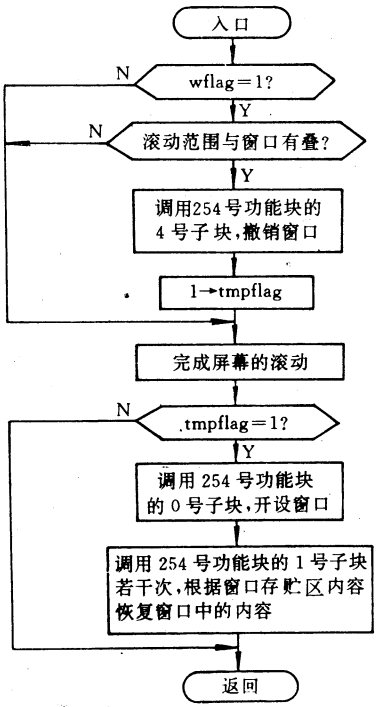


图 6-7 调整后的屏幕滚动程序

图 6-7 给出了调整后的屏幕滚动程序的流程。

从图 6-7 可知，我们采用的方法既简单又巧妙，在屏幕滚动之前先撤销窗口，恢复屏幕所显示的内容，接着完成屏幕的滚动，然后再重新开设窗口，在窗口内恢复原来的内容。

3. 对 9 号和 10 号功能块的调整

在增加了窗口管理功能后，必须对 9 号和 10 号功能块作相应调整。我们要求，如果在被窗口覆盖的屏幕区域内显示字符或汉字时(注意，这是指在屏幕显示的内容，而决不是指在窗口中显示的窗口内容)，被显示的字符或汉字字形则不出现，但它们的有关内容仍被存入 vram 中。所以，我们应该对 9 号和 10 号功能块作如下调整：在把当前字符代码等存入 vram 的三个分区后，应先判断当前光标是否处于窗口区内，若在窗口区内的话，则不再取对应字模送 rram 了，即不显示字符或汉字字形。如果光标不在窗口区，就按原来流程执行，继续显示字符或汉字的字形。

另外，为了配合撤销窗口时完成汉字的边界处理，对把标志字

节存入 vram 分区部分也要作一调整。对于字符，则把 00H 存入标志分区；对于未配对的汉字内码符，则把 80H 存入标志分区；对于配对的汉字内码符，若为汉字内码的第一字节，则把 01H 存入标志分区，若为汉字内码的第二字节，则把 02H 存入标志分区。

第二节　汉字库结构设计及性能分析

一、引言

1. 汉字库总述

汉字库是汉字信息处理系统中的一种重要数据结构，它的结构设计得是否合理，会影响整个汉字系统的性能，特别会影响输出模块的效率。输出模块的最主要工作是驱动外部设备输出汉字字形，因此输出模块必须能实现把汉字内码转换成相应的汉字字形信息。事实上，输出模块正是依靠汉字库才得以完成这种转换处理。由此可知，汉字库在汉字输出处理中占有相当重要的地位。

汉字库中存有汉字系统所支持的所有汉字及符号的字形信息，它需要占用较大的存贮空间。存贮汉字库的介质有多种，如 RAM、ROM、磁盘和磁带等。有一种称为基于 RAM 的汉字库深受人们的关注，这种汉字库需要用 RAM 存放汉字库的全部或部分内容。

当前的微计算机系统大多配有 1M 字节以上的内存，除去操作系统占用的空间外，尚有较充裕的内存空间，为使用基于 RAM 的汉字库创造了良好的条件。因此，现在有相当多的汉字系统采用基于 RAM 的汉字库。这种汉字库具有灵活性大、对原系统改动少和投资低等优点，它的最主要缺点是需要占用内存资源。不过，在内存资源相当富裕的条件下，这个缺点就会显得不太明显。本节将对基于 RAM 的汉字库的结构及性能展开讨论，以寻求一种发扬这种汉字库优点和弥补这种汉字库缺点的结构。

2. **约定**

为便于叙述和讨论，我们特作如下几点约定：

①设系统采用变形国标码作汉字内码，即：

$$R\{GB_i \mid A1H \leqslant GB_i \leqslant FEH\} \quad (i=1,2)$$

$$汉字内码=\{GB_1GB_2 \mid GB_1, GB_2 \in R\}$$

②汉字库内含有 GB2312 中的全部汉字和符号的字形信息。

显然，这些约定不会影响所讨论问题的普遍性，我们可以对在此约定条件下推出的结论再作进一步的推广。

二、汉字库性能的描述

汉字库的性能主要包括访库速度 v 和内存开销 m，v 和 m 分别与汉字库的性能成正比和反比关系。因此，我们可以用 k 参数来描述汉字库的性能，k 参数之值越大，表示汉字库的性能越佳。我们用下式计算出 k 参数：

$$k=\frac{v-v_0}{m}$$

式中 v 为输出模块用此汉字库后所能达到的输出汉字平均速度，m 为汉字库的内存开销，v_0 为输出模块应具备的输出汉字的起码平均速度，它的取值决定于 k 参数描述的有效范围。对于要求较高的系统，v_0 可以取得大一些；反之，对于要求较低的系统，v_0 可以取得小一些，但是其最低值不能低于输出模块使用全外存型字库时所达到的输出汉字平均速度。这就意味着，可以用 k 参数作性能描述的汉字库，必须是真正的基于 RAM 的汉字库。

为了寻求提高汉字库的 k 参数的方法，我们必须进一步分析 v 和 m 之间的关系。显然，v 和 m 间存在着一般的“时间”和“空间”之间的关系，但是它们之间还存在着更进一步的关系。v 并不单纯地决定于 m，在 m 一定的前提下，v 还决定于汉字库中内容的分布。如果汉字库常驻内存部分的内容选用得较合理，则会使其中的汉字在系统输出内容中有较高的出现频率，从而可获得较高的

v。我们可以用下式表示 v 与 m 之间的关系：

$$v=m \cdot f(s) \cdot w$$

式中 s 为 GB2312 的全集，f 为确定其子集作为汉字库常驻内存部分的法则，w 为其它因素。从以上的分析可以知道，欲进一步提高 k 参数，必须寻求 f(s)的最佳情况。

三、静态汉字库结构

静态汉字库的内容是事先确定好的，在系统运行过程中，汉字库内容不会发生变化。一般的汉字系统多采用静态汉字库。

1. 全内存型字库

全内存型字库的所有内容均常驻内存，它由系统自举程序在进行系统初始化时调入内存。一些简单的汉字系统常采用这种字库。这种字库是基于 RAM 的汉字库的基本型。

这种字库的结构与访问字库的算法密切有关。为了使访问字库的算法最简单，这种字库内的字形信息严格按照其对应汉字或符号的国标码序存放，这样就可以根据汉字内码 GB_1GB_2，通过极其简单的运算，获得其对应汉字字形信息的库内地址 addr，计算公式如下：

$$addr=((GB_1-A1H)\times 94+(GB_2-A1H))\times len$$

式中的 len 为每个汉字的字形信息的长度(以字节为单位)。

显然，对这种汉字库的访问是快速的，但是，这种汉字库要占用大量的内存资源，因此它的 k 参数不可能很大。在应用程序日趋庞大和复杂的今天，这种汉字库的缺点会越来越突出。

2. 全外存型字库

全外存型字库的所有内容均常驻外存，一般为驻留在磁盘上，这样就使汉字库完全不与应用程序争用内存空间。这种字库是基于 RAM 的汉字库的极端型。

全外存型字库的结构与全内存型字库的结构完全一样，只是存贮介质不同。库内的字形信息也按照其对应汉字或符号的国标

码序排列。访问这种字库的算法如下：

①根据汉字内码 GB_1GB_2 算出其对应字形信息的库内字节序号 n，计算公式为：

$$n=((GB_1-A1H)\times 94+(GB_2-A1H))\times len$$

②根据 n 算出字形信息所在的库文件内相对区号 q 和区内字节序号 r，计算公式如下：

$$r=n \text{ MOD } l$$

$$q=(n-r)/l$$

式中 l 为扇区所含字节数。

③根据磁盘文件的信息结构，依靠系统中的文件定位表 FAT，获得字形信息所在的面号 f、道号 t、区号 s。

④根据面号 f、道号 t、区号 s 和扇区长度 l 取字形信息。

显然，上述算法的实现还是比较快速的。但是，由于外存本身的响应速度远低于内存的响应速度，因此这种字库的最主要缺点为响应速度较慢。实践证明，如果这种字库驻留在硬盘上，那么其响应速度可以满足一般用户的使用要求。

3. 内外存结合型字库

内外存结合型字库由内存字库和外存字库组成，前者的内容常驻内存，后者的内容常驻外存。这种字库是全内存型和全外存型字库之间的一种折衷方案，旨在克服这两种字库的缺点和发扬它们的优点。

据统计，在日常使用汉字中，GB2312 内一级汉字的出现频率可达 99%以上，而二级汉字的出现频率极低。据此，我们可以把一级汉字的字形信息驻留在内存，作为内存字库；把二级汉字的字形信息驻留在外存，作为外存字库。这种方案可以达到既减少内存开销（与全内存型字库相比），又保证访问速度的目的。

访问内外存结合型字库的算法是前两种算法的结合。当 $GB_1 \leqslant$D7H 时，则访问内存字库，其算法与访问全内存型字库的算法相同；当 $GB_1>$D7H 时，则访问外存字库，这时先按以下公式算出

汉字字形在外存字库内的字节序号 n：

$$n=((GB_1-D8H)\times 94+(GB_2-A1H))\times len \qquad (1)$$

然后再按照全外存型字库的访问算法进行下去。

实践证实了内外存结合型字库具有比较高的效率，目前有不少汉字系统采用内外型结合型汉字库。

4. 性能分析

内外存结合型字库具有较好的性能，但是，其 k 参数之值的范围较广。下面我们对这种字库的性能作出比较深入的分析，寻求获得字库的较高 k 参数的方法。

从字库性能描述公式来看，内外存结合型字库的 k 参数，主要决定于内存字库的容量及其内容的命中率(即出现频率)。为了解决这两个问题，我们可以通过统计来获得启发。为了使统计的结果能比较客观地反映实际，我们随机地抽取有关数学、物理、化学、地理、历史、新闻和文学等方面的资料，共 552624 个字，以此作为统计材料，然后按内存字库中所含汉字的个数，分别进行统计。

我们根据汉字频度统计表，把频度最高的汉字排入内存字库中(排满为止)。然后按照图 6-8 中给出的算法来计算命中率。

把内存字库设置成不同的字数后，分别运用图 6-8 中的统计算法 1，可以得到一系列的命中率值，其结果如表 6-1 所示。

表 6-1　静态内存字库命中率统计表

内存字数	100	200	300	400	500	600	700	800	900	1000
命中率	0.3651	0.5158	0.6248	0.7046	0.7534	0.7959	0.8327	0.8568	0.8781	0.8942

从表 6-1 中可以看到，使用频率最高的前 300 个汉字的命中率为 62.48%，前 600 个汉字的命中率为 79.59%，这可以说是相当高了。但是我们也要看到，由于用户不同，其所使用的常用汉字会有所不同，有时甚至会有很大的差别。比如“碱”、“酐”等字，对于化学专业的用户是很常用的，而对计算机专业的用户则几乎是不用的。所以，常用字不是固定不变的。

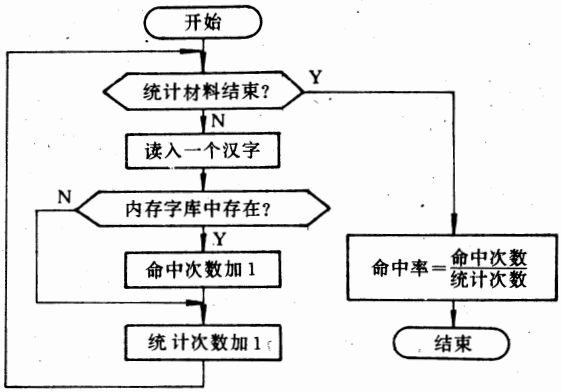


图 6-8 统计算法 1

为了寻求内存字库所含字数的最佳值，以使其命中率达到极值，我们可以作以下的数学推导。

设 H 为汉字集，即：

$$H=\{h_i \mid i=1,2,\cdots,n\},\ |H|=n \tag{2}$$

若 n 为汉字的总数，则 H 表示全体汉字组成之集合。在此我们设 H′ 为统计材料中全体汉字所组成之集合，令：

$$H'=\{h_{ij} \mid j=1,2,\cdots,s\},\quad s\leqslant n \tag{3}$$

则有：$|H'|=s$，$H'\subset H$

当 s=n 时，说明统计材料已包含所有不同的汉字。

若某汉字 h_{ij} 在统计材料中出现的次数为 η_{ij}，则统计材料中汉字的统计个数为：

$$T=\sum_{j=1}^{s}\eta_{ij}$$

又设内存字库中的汉字集为 H_1，且 $|H_1|=p$，则有 $H_1\subset H$，而且 H_1 不一定包含于 H′ 中。

又令 H_2 是 H′ 中出现次数最高的前 p 个元素组成的集合，显

然$|H_2|=p$，H_1 与 H_2 不一定相同。

再令 $H_3=H_1\cup H_2=\{h'_i|i=1,2,\cdots,q\}$ (5)

则有：　$|H_3|=q$，$q>>p$

并且：　$H_1\subset H_3, H_2\subset H_3$

假定：　$\vec{x}=(\varepsilon_1,\varepsilon_2,\cdots,\varepsilon_q)$ (6)

$\vec{y}=(\zeta_2,\zeta_2,\cdots,\zeta_q)$ (7)

若某个汉字 $h'_i(1\leqslant i\leqslant q)$ 在 H_1 中不存在，定义 $\varepsilon_i=0$，否则，ε_i 表示内存字库中的汉字在 H' 中出现的次数。

若某个汉字 $h'_i(1\leqslant i\leqslant q)$ 在 H_2 中不存在，定义 $\zeta_i=0$，否则，ζ_i 表示集合 H_2 中的汉字在 H' 中出现的次数。

很明显，若 $h'_i\in H_1\cap H_2$，则 $\varepsilon_i=\zeta_i$，而且，内存字库 H_1 中的汉字在统计材料中出现的总次数 N，可以用下式算出：

$$N=\sum_{i=1}^{q}\varepsilon_i$$

所以，命中率 ρ 可以用下式得出：

$$p=\frac{\sum_{i=1}^{q}\varepsilon_i}{\sum_{j=1}^{s}\eta_j} \tag{8}$$

我们可以用下式来表示内存字库在统计材料中可能出现的命中次数之最大值 M，也就是极值：

$$M=\sum_{i=1}^{q}\zeta_i$$

如果用 $\|\vec{x}-\vec{y}\|$ 来表示 $\vec{x}$、$\vec{y}$ 的距离，则定义：

$$\|\vec{x}-\vec{y}\|=\sum_{i=1}^{q}|\varepsilon_i-\zeta_i| \tag{9}$$

根据上面的假设可知，$\varepsilon_i\geqslant\varepsilon_i$

故有

$$\|\vec{x}-\vec{y}\|=\sum_{i=1}^{q}(\zeta_i-\varepsilon_i)=\sum_{i=1}^{q}\zeta_i-\sum_{i=1}^{q}\varepsilon_i \tag{10}$$

以上式子表明 $\vec{x}$ 与 $\vec{y}$ 的距离就是指实际内存字库的命中次数

与最大可能的命中次数之间的差距。

因此，当 $H_1 \neq H_2$ 时，$\| \vec{x} - \vec{y} \| > 0$ (11)

当 $H_1 = H_2$ 时，$\| \vec{x} - \vec{y} \| = 0$ (12)

当 $\| \vec{x} - \vec{y} \| = 0$，则表示此时内存字库中的汉字是统计材料中最为常用的字，此时命中率 η 达到极大值。

但是，在一般情况下，$H_1 \neq H_2$，这时如果 $\| \vec{x} - \vec{y} \|$ 之值越大，则说明内存字库的命中率离开极值越远。

四、动态汉字库结构

动态汉字库的内容，在系统运行过程中能进行调整。它能根据当前的使用环境，调整到一个最佳状态（或接近于最佳状态），以用较小的内存开销，获得较高的输出速度。所以，动态汉字库具有良好的自适应性。

1. 准动态型字库

(1)结构　准动态型字库是由静态汉字库中的内外存结合型字库改进而成。它也分为常驻内存部分（内存字库）和常驻外存部分（外存字库），但是其内存字库的内容为一特定应用中要用到的汉字之全集。比如某单位的工资管理系统，它运行时所用到的汉字集是一定的，现就把这些汉字的字形信息集作为内存字库的内容。这样，运行该工资管理系统时，其速度可达到使用全内存型字库的水平。但是，其 k 参数却可远大于全内存型字库，也大于内外存结合型字库。

为了建立内存字库，可以设计一个专门的建库工具，用于统计某个特定系统或程序内所用到的汉字，然后形成内存字库。内存字库中的字形信息按照其对应汉字的国标码序排列。同时还要设立一张字库登记表，表中登记了其字形信息存于内存字库的汉字内码，这些汉字内码也按序排列。显然，登记表项与内存字库的字形信息一一对应。外存字库中存放 GB2312 汉字集的字形信息（同全外存型字库）。

(2)算法　根据准动态型字库的结构，可以设计出访问这种汉字库的如下算法：

①根据汉字内码检索字库登记表，并进行表项序号计数；

②如果检索到相符之项，则根据计数值(即字形信息的序号)取内存字库中的字形信息；

③如果检索不到相符之项，则根据汉字内码，按照全外存型字库的算法从外存字库中取字形信息。

大家不难看出，准动态型字库对环境的适应性不强，当运行另一个系统或程序时，需要通过建库工具重新建立内存字库，否则会大大降低 k 参数。由于这种调整字库内容的方法不是由系统在运行过程中自动完成的，所以严格地说，这种字库还称不上是真正的动态型字库，因而只能够称为准动态型字库。不过，这种字库非常适合于专用系统。

2. 自适应型字库

(1)设计思想　人们日常使用汉字有一个规律，即某一段时间内所用到的汉字比较集中在一个不大的汉字集内，而且一个汉字往往要多次被使用。自适应型字库正是根据这一规律提出来的。这种汉字库也分为内存字库和外存字库两个部分，内存字库中的内容由系统根据使用环境进行自动调整。这是一种真正的动态型字库。为了追求较高的 k 参数，故内存字库的容量不能太大，但也不能太小，否则会影响到 v，继而又影响到 k 参数。根据前面所作的统计可知(见表 6-1)，使用频率最高的前 500 个汉字在一般使用中的出现频率已经高达 75%左右，再考虑到数据结构设计方面的因素，我们可以把内存字库的容量定为 512 个汉字的字形信息总量。

内存字库的初始值为汉字使用频率统计表内前 512 个汉字的字形信息。系统运行后，即根据一定的原则更新库中的内容，使得该库能最佳地适应当前的使用环境。由于自适应型字库与使用环境间存在着这样一种特定的关系，故在系统运行过程中，内存字库

中的汉字在系统实际使用汉字内的出现频率会远大于75%。

(2)结构　为了实现对内存字库的访问和提供更新内存字库内容的依据，系统中必须建立一张索引表。该索引表共有512个表项，每项对应于内存字库中一个汉字，每项含“内码”和“使用次数”两个内容。图6-9给出了索引表和汉字库的结构，从中可以体现出它们之间的关系。

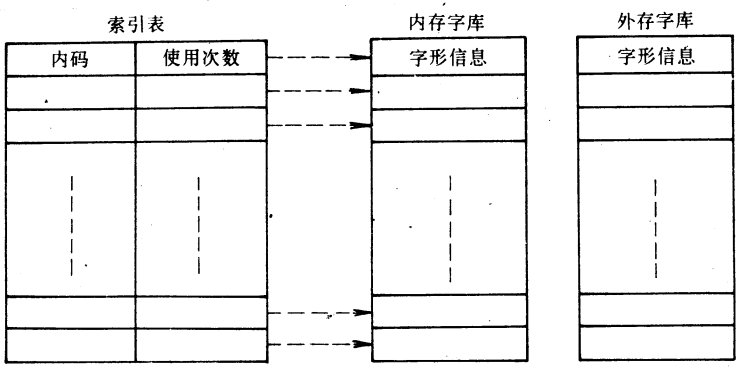


图6-9　索引表和自适应型字库

索引表表项中“内部码”部分的顺序与内存字库中汉字的顺序是一致的。所以，根据索引表表项序号可以从内存字库中取得相应的字形信息。为了使读取外存字库的算法简单和迅速，外存中仍含有一、二级汉字的字形，这样就可以使用全外存型字库的算法来访问外存字库。

(3)算法　访问字库的算法中要涉及到更新内存字库的原则。当欲输出的汉字在内存字库中不存在时，应该把该字的字形信息调入内存字库，并要淘汰内存字库中的一个汉字。应该淘汰哪一个汉字，这正是我们要确定的原则。常用的淘汰原则有三种，其一是先进先淘汰法(FIFO)，其二是最近最少用淘汰法(LFU)，其三是最近最久未用淘汰法(LRU)。经过分析和测试(在下面介绍)，我们发现最近最少用淘汰法的效果最佳。根据这一原则，我们可以设计出如图6-10所示的访问自适应型字库的算法。

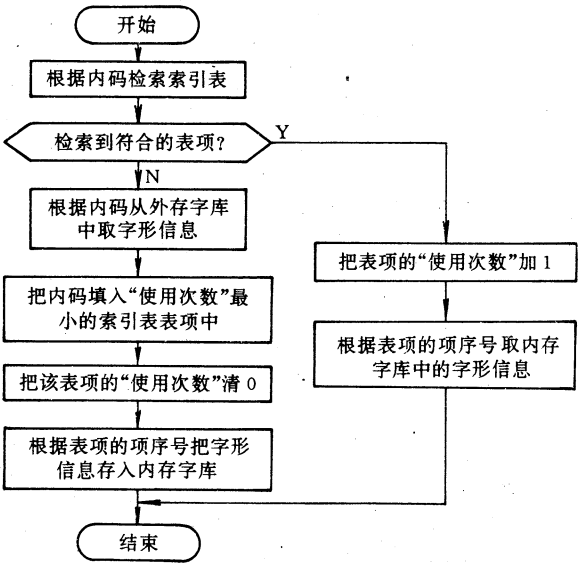


图 6-10　访问自适应型字库的算法

由于图 6-10 中的算法采用的是最近最少用淘汰法，故索引表表项中的"使用次数"即为淘汰依据。为了能体现出"最近最少用"，因此必须每隔一定时间就把各索引表表项的"使用次数"清为 0。我们可以通过扩充系统的时钟中断处理程序来完成这项工作。

(4)讨论　前面已经提到，三种淘汰算法数最近最少用淘汰法为最佳。下面我们要给出这一结论的根据，从而要围绕这三种算法作一些讨论。

我们仍然用统计的方法来比较这三种算法的效果。我们仍采用前面用过的统计材料，然后按内存字库中所含汉字的个数和三种淘汰算法，分别进行统计。

同样，我们先根据汉字频度统计表，把频度最高的汉字排入内存字库中(排满为止)。然后按照图 6-11 中给出的算法来计算命中率。

把内存字库设置成不同的字数后，再分别选用三种淘汰算法，

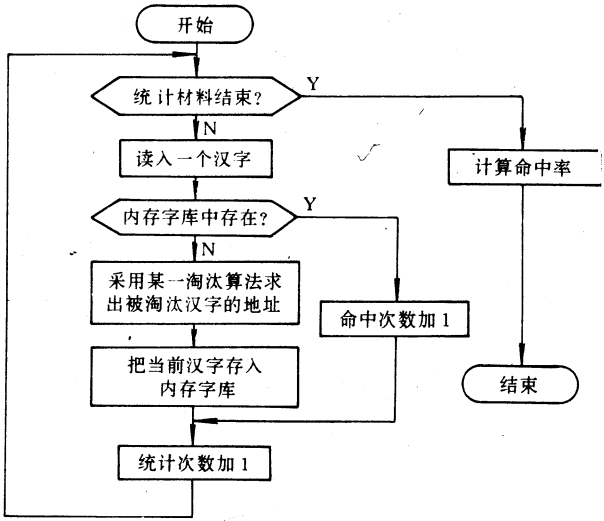


图 6-11　统计算法 2

然后运用图 6-11 中的统计算法 2，可以得到一系列的命中率值，其结果如表 6-2 所示。

表 6-2　动态内存字库命中率统计表

内存字数	100	200	300	400	500	600	700	800	900	1000
LFU	0.5854	0.7289	0.8191	0.8649	0.9046	0.9235	0.9391	0.9501	0.9584	0.9649
LRU	0.5684	0.7218	0.8054	0.8576	0.8931	0.9161	0.9330	0.9452	0.9551	0.9621
FIFO	0.5207	0.6726	0.7454	0.8154	0.8539	0.8823	0.9033	0.9205	0.9332	0.9424

从表 6-2 中的统计结果可以得出以下几点结论：

①对内存字库采用动态调整后，不管使用何种淘汰算法，其命中率均明显高于静态内存字库；

②随着内存中汉字个数的增加，其命中率的增长幅度不是呈线性的，而是越来越小；

③内存字库中字数为 500 个时，其命中率竟能和 1000 个汉字的静态内存字库相当；

④三种淘汰算法中，以最近最少用淘汰法的效果最佳。

从上面的情况来看，我们在前面把内存字库的容量定为512个字是合适的。另外，选用最近最少用淘汰原则来淘汰内存字库中的汉字是合适的。总之，动态型字库的效果要比静态型字库的效果好得多。

(5)问题　在一般情况下，自适应型字库具有较高的k参数值，但是在一些特殊情况下，这种字库的k参数值会降低。这是因为内存字库中的“最近最少用”汉字常常会有多个，那末对它们的淘汰就带有一定的随机性。然而，被淘汰出内存字库的汉字一旦又被用到时，又要被调入内存字库。显然，这两个因素有碍于自适应型字库k参数的提高，而且后者的影响较前者大。下面对此作进一步的讨论。

事实上，不管使用何种淘汰算法都不可避免地有一定的随机性，因为我们无法正确无误地预测将来。根据前面的数学推导可知，某个汉字 h_{ij} 在统计材料中的出现概率 p_j 为：

$$p_j = \frac{\eta_j}{\sum_{j=1}^{s} \eta_j}$$

一般来说，p_j 是一个很小的量，不超过0.01。根据 p_j 的大小，我们可以知道该汉字的常用程度(即频度)。由于统计材料的不同，也许有些常用字在整个材料中的概率也是比较大的，而它们的出现次序又有可能是它刚被淘汰不久之后又出现。如果有多个这样的字的情况出现，则会影响命中率 ρ，若要进一步解决这个问题，则要寻求新的、更完善的汉字库结构。

3. 多级型字库

(1)设计思想　多级型字库是对自适应型字库的改进。为了避免把常用汉字淘汰出内存字库，故设计了三级汉字库，它们分别为常用字库、动态字库和外存字库，其中常用字库和动态字库常驻内存，两者合起来称为内存字库。常用字库内存放最常用的汉字，动态字库内存放当前使用的汉字(最常用字除外)，外存字库内存放

一、二级汉字全集。常用字库和外存字库的内容是不变的，动态字库的内容在系统运行时被调整。由于常用字库是静态的，所以其中的最常用字不可能被淘汰出内存，从而克服了自适应型字库的主要不足之处。

(2)结构　首先，我们要确定内存字库中静态部分和动态部分的容量，也就是要确定常用字库和动态字库的容量。另外，我们仍然把内存字库的容量定为 512 个汉字。

下面，我们用统计的方法来得到常用字库容量的最佳值。由于内存字库的容量一定(512 个汉字)，故得出常用字库的最佳容量后，立即可以得到动态字库的最佳容量。我们仍采用前面用过的统计材料，然后按常用字库所含的汉字数和三种淘汰算法，分别进行统计。图 6-12 给出了所采用的统计算法。

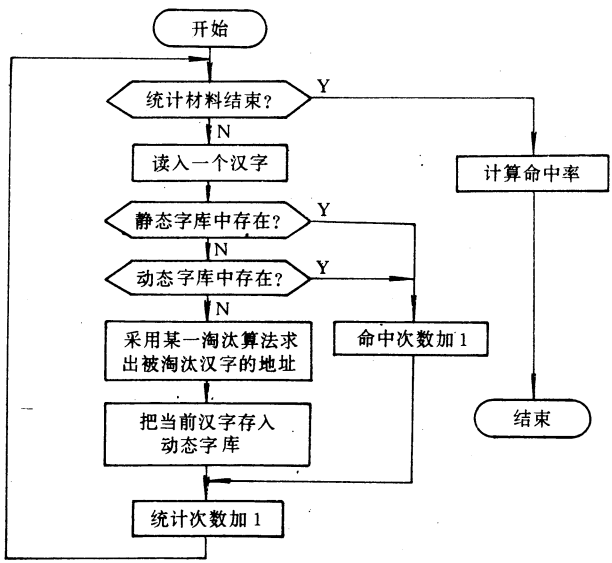


图 6-12 统计算法 3

把常用字库设置成包含不同的汉字数后，再分别选用三种淘汰算法，然后运用图 6-12 中的统计算法 3，可以得到一系列的命中率值，其结果如表 6-3 所示。

表 6-3　动静态内存字库命中率统计表

常用字库容量	128	160	192	224	256	288	320	352	384
百分比	25.00	31.25	37.50	43.75	50.00	56.25	62.50	68.75	75.00
LFU	0.9126	0.9128	0.9124	0.9115	0.9112	0.9104	0.9099	0.9080	0.9057
LRU	0.9067	0.9068	0.9070	0.9067	0.9066	0.9064	0.9059	0.9041	0.9022
FIFO	0.8845	0.8876	0.8902	0.8911	0.8940	0.8940	0.8944	0.8958	0.8930

从表 6-3 的统计结果可知，多级字库的性能要优于前面几种字库；并且，在相同条件下三种淘汰算法中仍是最近最少用淘汰法为最优；常用字库容量可以取为 256 个汉字，那末动态字库容量亦为 256 个汉字。

由于字库的级数增加了，所以对汉字库的管理层次也增加了，如果仍沿用前面那种简单的数据结构，则势必会影响访库速度。为此，可以设计一个 Hash 函数，利用它来建立常用字库和动态字库，在输出模块中再利用这个函数实现对字库的访问。除此之外，还需要设计一张定位表。图 6-13 给出了多级型字库的数据结构情况。

从图 6-13 可知，利用 Hash 函数建立的常用字库和动态字库实为一体(内存字库)。常用字库内的汉字，根据 Hash 函数和定位表基本上能作到一次定位；动态字库内的汉字则需二次或多次散列后定位。所以，采用 Hash 函数定位要比顺序检索定位快得多。

(3)算法　为了尽可能提高 v，以获得较大的 k 参数值，Hash 函数的设计是至关重要的。设计这个函数时应注意下面两个方面：

①必须减少函数值的“冲突”现象，使函数值分布比较均匀；

②函数不宜设计得太复杂，以免影响函数的运算速度。

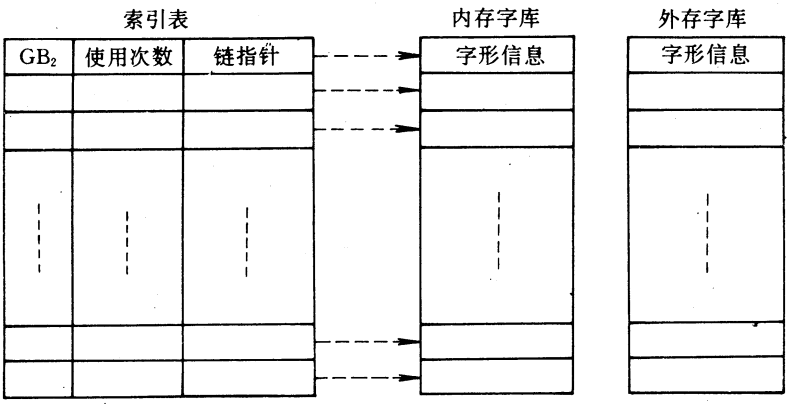


图 6-13　多级型字库的数据结构

当然，以上两个方面是矛盾的。我们认为，应该寻求一种折衷方案来兼顾这两个方面。例如，可以提出以下函数作为我们所要的 Hash 函数：

$$y=(GB_1-A0H)+((GB_2-A0H)MOD\ 8)$$

我们可以用这个函数来建立常用字库和动态字库，使常用字处于链首，使动态字库内的字处于链域。不过，上述函数并非最佳函数，它仅作为一个实例而已，读者完全可以设计出更加完美的 Hash 函数。

下面，我们开始讨论访库算法。我们利用确定好的 Hash 函数建立了常用字库和动态字库之后，则可用图 6-14 给出的算法来访问我们的三级字库。

从图 6-14 的流程中不难看出，这种访库算法是简单和快速的，非常适合于多级型字库使用。当动态字库装满后，也采用最近最少用淘汰法来更新内容，这就需要定时把定位表中的“使用次数”清为 0。我们可以象自适应型字库中采用的方法那样，借助于时钟中断处理程序来完成这项工作。

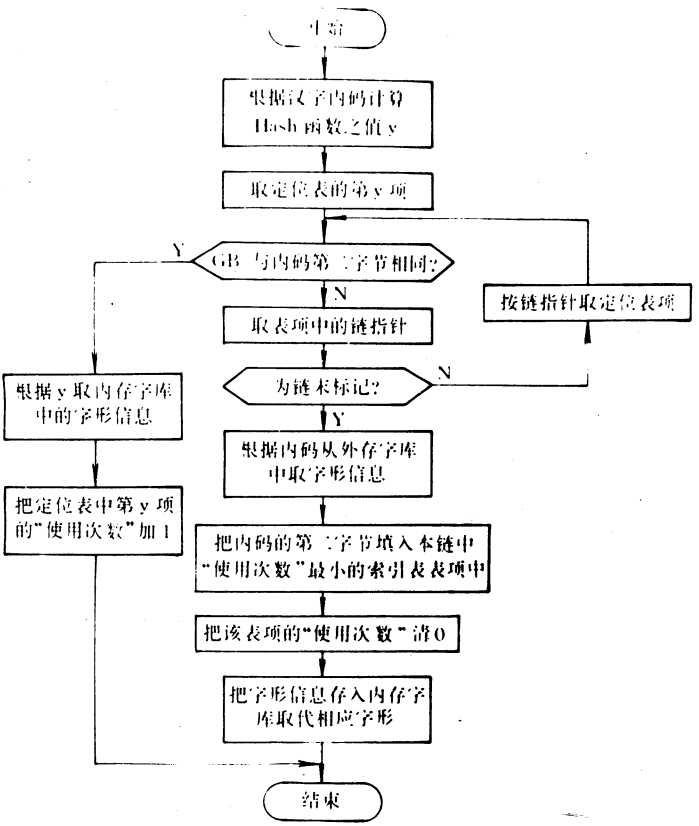


图 6-14 访局多级型字库的算法

多级型字库的结构比较完善，它具有较大的 k 参数值。它所需的内存开销不足 20k 字节，然而利用它来输出汉字的速度却与全内存型字库不相上下。它的常用字库体现了汉字使用的普遍规律，它的动态字库又体现了在特定环境下使用汉字的特殊规律，因此这是一种很有前途的汉字库。

以上对汉字库结构所作的讨论，特别适用于非压缩字模的汉字库，对压缩字模的汉字库结构的研究也有指导作用。对汉字库结构的讨论，必须与当前的硬件支撑环境和硬件价格相联系，以得出

一种较合理的汉字库结构。

第三节　多级型汉字库的设计与实现

一、概述

从上一节所讨论的内容可知，多级型汉字库具有很好的性能，可以广泛应用于汉字系统。本节将重点介绍多级型汉字库及其管理模块的设计和实现。

我们的目标是设计一个三级汉字库和它的管理模块。这个三级汉字库分为常用字库、动态字库和外存字库，其中常用字库和动态字库常驻内存，外存字库常驻硬盘。常用字库中存放最常用汉字，动态字库中存放当前用户使用的汉字（不包括常用字库中的汉字），外存字库中存放一、二级汉字全集。根据上一节中的分析，我们把常用字库和动态字库的总容量定为 512 个汉字，所占内存空间为 16k 字节。据上一节的统计，这样常用字库和动态字库对用户使用汉字的命中率可达 90%左右。我们在这儿也选用最近最少用淘汰法来更新动态字库的内容，使其尽可能地反映当前使用环境。

我们必须充分注意到，汉字库分为三级后，必然会增加对它的管理难度，所以，应该专门设计多级型汉字库的管理模块。该模块除了实现对多级字库的访问外，还要实现对动态字库内容的更新。其中值得注意的是，不能忽视对外存字库的访问，要求作到尽可能快地访问它。

二、数据结构

为了实现对内存中的常用字库和动态字库的快速访问，所以应该分别为 它们建立定位表，分别称为常用字库定位表和动态字库定位表。它们与常用字库及动态字库一起驻留内存。

系统自举时，由初始化程序把常用字序、动态字库和两张定位表装入内存，形成汉字库的内存映像，详情如图 6-15 所示。

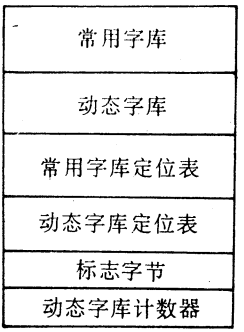


图 6－15　汉字库的内存映像

定位表由若干个表项组成，每个表项与相应字库中的一个汉字相对应，故定位表的表项数应等于其对应字库内的汉字数。由于动态字库的内容要随时更新，而且要根据一定的原则来更新，因此动态字库定位表的表项内容与常用字库定位表的表项有所不同。

图 6-16 给出了这两种定位表的结构。

常用字库定位表

关键字	链指针
⋮	⋮

动态字库定位表

关键字	使用次数	链指针
⋮	⋮	⋮

图 6－16　定位表的结构

常用字库定位表表项有两个内容，即关键字和链指针。关键字用于表征一个汉字；链指针同于形成同 Hash 函数值的汉字链，它指向链内下一个汉字。动态字库定位表表项中增加了一个内容（即使用次数），它用于记录对应汉字在最近一段时间内的使用次数，

为动态字库的内容更新提供依据。

为了提高输出汉字的速度，就必须缩短定位汉字库内汉字的时间。根据以上定位表的结构可知，我们能利用 Hash 函数定位法来定位常用字库和动态字库中的汉字，从而可使定位速度远高于顺序检索定位法。当 Hash 函数值发生冲突时，则可用线性拉链法，顺着链指针在同 Hash 函数值链内查找下去，以定位可需之汉字。

常用字库和动态字库均是按照定好的 Hash 函数建立的，虽然这样建立的字库排列不甚紧凑，装填因子不是太大，但是其获得的效率还是比较高的。

三、Hash 函数的设计

显然，Hash 函数的设计是十分关键的，它将会直接影响到访问字库的速度和字库的存贮开销。正象上一节所介绍的那样，Hash 函数的设计应充分考虑到函数值的分布性和函数自身的复杂性。

我们通过对汉字频度表的分析，挑选了其中的最常用字和常用字作为常用字库的内容。然后，根据这些汉字的内码分布规律，设计出如下 Hash 函数：

$$y=((GB_1-A0H)\times 5+(GB_2-A0H)\times 3)MOD\ 129$$

其中，GB_1 和 GB_2 分别为汉字内码的高位字节和低位字节。显然，该函数值的范围为 0～128。

确定了 Hash 函数之后，就要确定定位表中的关键字。必须保证，具有相同 Hash 函数值的汉字的关键字应不相同，也就是说，同 Hash 函数值汉字链内的汉字应具有不相同的关键字。只有这样，才能在发生 Hash 函数值冲突时，能利用关键字定位所需之汉字。

根据已经确定的 Hash 函数，我们确定以汉字内码的低位字节作为关键字。我们可以证明，在这种情况下，决不会出现 Hash

函数值相同且关键字亦相同的情况。下面给出证明过程。

设有两个不同的汉字，其内码分别为 $GB_{11}GB_{12}$和 GB_{21} GB_{22}，并且 $GB_{12}=GB_{22}$。根据 GB2312 可知，不同汉字的内码也不同，故必有 $GB_{11}\neq GB_{21}$。

下面用反证法证明。

设由这两个汉字内码计算所得之 Hash 函数值分别为 H_1 和 H_2，并且假设 $H_1=H_2$，则：

$$H_1=((GB_{11}-A0H)\times 5+(GB_{12}-A0H)\times 3)MOD\ 129$$

$$H_2=((GB_{21}-A0H)\times 5+(GB_{22}-A0H)\times 3)MOD\ 129$$

因为 $H_1=H_2$，所以下列三种情况中必有一种成立：

$$GB_{11}=GB_{21} \tag{1}$$

$$(GB_{11}-A0H)\times 5=(GB_{21}-A0H)\times 5+m_1\times 129 \tag{2}$$

$$(GB_{11}-A0H)\times 5+m_2\times 129=(GB_{21}-A0H)\times 5 \tag{3}$$

其中，m_1 和 m_2 为自然数。

若(1)式成立，则两个汉字实为同一字，这与前面的条件矛盾，故假设不成立。

再看(2)式和(3)式，显然这两式属于同一类情况，不失一般性，我们对(2)式进行讨论：

首先，将(2)式略作转换为：

$$5\times(GB_{11}-GB_{21})=m_1\times 129 \tag{4}$$

因为 $GB_{11}\neq GB_{21}$，并且 GB2312 中规定共 94 个区，每区含 94 个汉字，故有：

$$1\leqslant(GB_{11}-GB_{21})\leqslant 93 \tag{5}$$

欲使(4)式成立，则必须有 $m_1=5\times n$，其中 n 为自然数。

于是，$GB_{11}-GB_{21}=n\times 129$ (6)

(6)式与(5)式显然是矛盾的，故假设不成立。

所以，结论得证。

综上所述，我们选择汉字内码的低位字节作关键字是可行的。

四、汉字库管理模块

1. 总体流程

汉字库管理模块的主要功能是实现对三级汉字库的访问，讲得更透彻一点，就是要根据用户提供的汉字内码，获得该汉字的字模(字形信息)或其所在地址。我们把该模块的入口参数定为汉字内码，把出口参数定为该汉字字模的地址。图 6-17 给出了该模块的总体流程。

下面对图 6-17 中的流程作几点说明。

(1)由汉字内码计算出 Hash 函数值后，总是先从常用字库定位表开始检索，检索不到时才进入动态字库定位表。所以，需要设立一个标志字节，用于指出当前检索到的汉字是在常用字库内，还是在动态字库内。若在两张定位表中均检索不到，则要访问外存字库。

(2)动态字库定位表表项的“使用次数”之值，是在动态字库存满后实行淘汰的依据。所以在运行过程中，要随时为其增量。

(3)动态字库是否已装满，由动态字库计数器指出。当字库满时，就要根据“最近最少用”原则进行淘汰。为了简化算法，可以直接淘汰当前检索链中“使用次数”最少的那个汉字，从而不必搜索整个动态字库定位表，这在一定程序上提高了效率。

采用淘汰当前检索链中“使用次数”最少的汉字的办法，虽然使得算法简化了，但是会 产生一个副作用。那就是，各条 Hash 链(即相同 Hash 函数值的汉字链)始终不变，直至系统运行结束。换言之，从动态字库存满的那一时刻起，动态字库的模式就定下来了。这样很有可能造成某些汉字被频繁地调进和调出动态字库，从而影响系统的效率。为了解决这一问题，我们可以每隔一定时间对动态字库作一次清除。动态字库的清除工作非常简单，只要把常用字库定位表中与动态字库定位表表项链接的表项的链指针均置为链尾标志，再把动态字库计数器清为 0。这样作后，能使动态字库

不断更新，有利于降低某些汉字被频繁地调进和调出动态字库的几率。

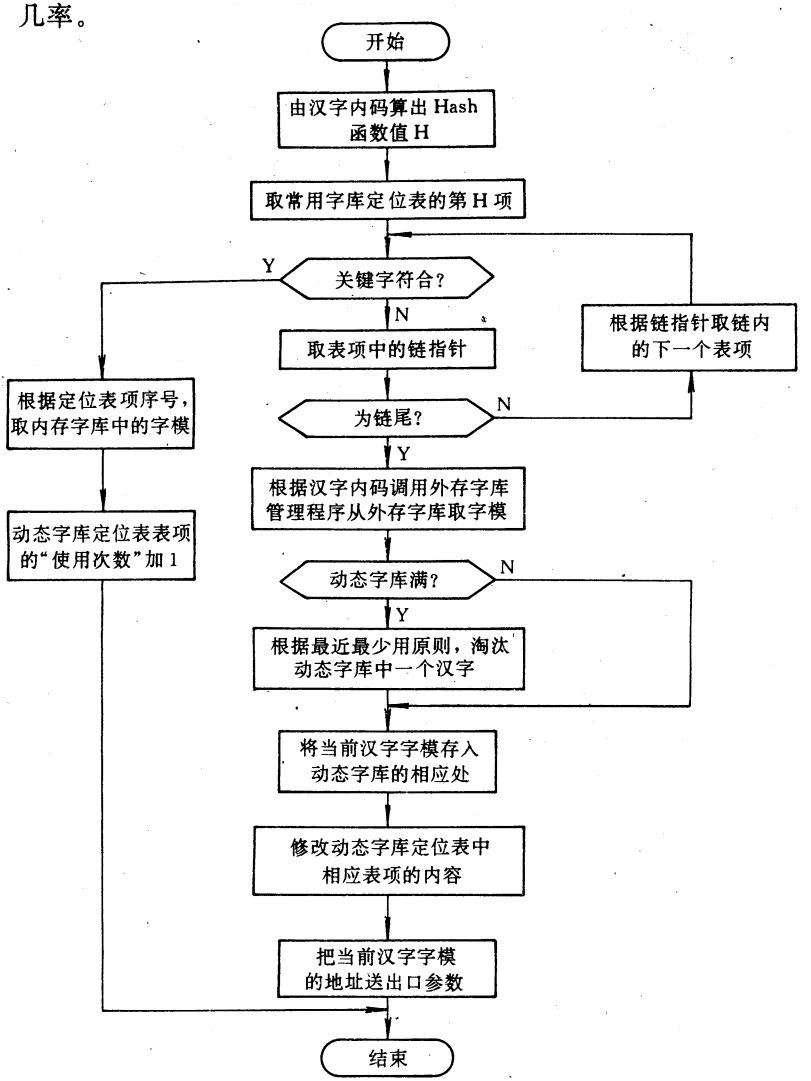


图 6-17　汉字库管理模块的总体流程

2. 时钟中断处理程序的扩充

多级型字库的响应速度，与动态字库的内容有很大关系。根据上一节的统计，对动态字库的内容采用最近最少用淘汰法，可以获得较好的效果。这种淘汰法的实行，主要要把握两个方面，即“最近”和“最少用”。动态字库定位表内的“使用次数”可以作为“最少用”的依据，然而怎样体现出“最近”呢？上一节内曾提到过，可以使用时钟中断定时把“使用次数”清 0，这样就充分体现出“最近”的意义。所以，有必要考虑时钟中断处理程序的扩充。

利用时钟中断处理程序实现对动态字库定位表中的“使用次数”定时清 0 是可以的，从原理上来看并不复杂。但是，真要付诸实现却并不容易，因为时间周期（或称时间间隔）的确定是个难题。上述淘汰法类似于内存管理中的页面调度算法，周期定得太长和太短，都有可能使一些当前常用字被淘汰出去，从而使一些汉字被频繁地调进和调出。因此，我们必须慎重地选择时间周期 。

系统每隔 55ms 就发生一次时钟中断，由 8H 号中断处理程序完成对时钟中断的处理。但是 8H 号中断处理程序中又调用了 1CH 号中断处理程序。因此，1CH 号中断处理程序也是每隔 55ms 被调用一次。这个中断处理程序中仅有一条中断返回（IRET）指令，可见它是专门被用于用户扩充的。所以，我们可以对 1CH 号中断处理程序进行扩充，使它完成对动态字库定位表中的“使用次数”清 0。为了便于从各种方面实现对 1CH 号中断处理程序的扩充，可以在初始化程序中取得原 1CH 号中断处理程序的地址，并把它保存起来，在扩充程序结束处通过一条间接转移指令返回原 1CH 号中断处理程序的入口点执行之。扩充的时钟中断处理程序如图 6-18 所示。

下面对图 6-18 作两点说明：

（1）时间计数器的初值为 0，它记录了发生时钟中断的次数，实际上也记下了时间间隔 T，若某一时刻该计数器之值为 t，则 T＝t×55ms。

(2)我们已确定内存字库的容量为512个汉字,所以内存字库是很容易充满的(严格的讲是动态字库很容易被充满)。如果与文本文件打交道,那末在有限时间之后就有必要对动态字库进行更新了,因此在内存字库未满前,就没有必要对动态字库定位表的“使用次数”作频繁的清除工作。

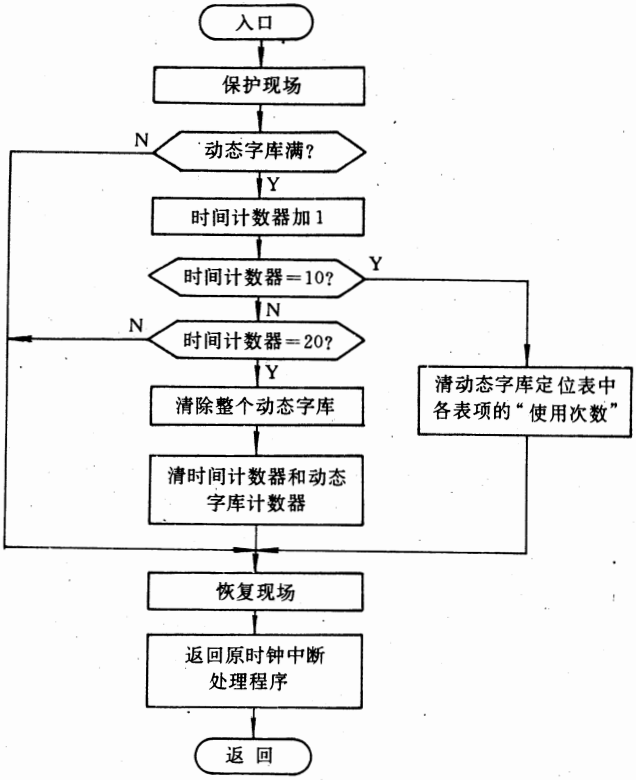


图 6-18 扩充的时钟中断处理程序

3. 外存字库的管理

外存字库的管理主要表现为对外存字库的读取操作。不管内存字库的命中率有多高,但是它总是不可能达到100%的,所以在

使用中免不了要访问外存字库。因此，有必要讨论一下外存字库的管理问题 。

为了使访问外存字库的速度能尽可能提高，也为了避免出现DOS 内核的重入，所以对外存字库的访问不能采用 DOS 中断(如21H 号中断)，而须采用 BIOS 中断中的 13H 号中断来实现。由于13H 号中断有许多入口参数，故在调用它之前必须按规定把有关参数送入规定的寄存器。因此，外存字库管理程序就围绕着对 13H 号中断的调用及其入口参数的获得来设计。这个程序主要由以下几个子程序组成：

(1)子程序 1　子程序 1 的主要功能是读取盘上的引导记录，获取 BPB(BIOS 参数块)参数。它的执行流程为：

①从磁盘的 0 道 0 面 1 扇区读得盘的主引导记录；

②检查本次读盘是否正确，若不正确，则复执三次，如果复执三次均不正确，则结束；

③查主引导记录中的分区表，根据引导指示符确定当前引导分区；

④读取 BPB 参数，送至内存缓冲区；

⑤保存一些必要的数据至指定单元。

(2)子程序 2　子程序 2 的主要功能是根据指定的文件路径名获得 DOS 分配给该文件的首束之束号。该子程序的入口参数为一个指向文件路径名(用 ASCIIZ 串表示)的指针，它的出口参数为文件的首束束号。图 6-19 给出了这个子程序的流程。

(3)子程序 3　子程序 3 的主要功能是由束号获得该束在盘上的物理地址(道号，面号，区 号)。子程序 3 的入口参数为束号，它的出口参数为盘上的三维物理地址。这个子程序的执行流程如下：

①根据入口参数提供的束号计算出该束内首扇区的逻辑扇区号，计算公式为：

逻辑扇区号＝(束号－2)×每束扇区数＋保留扇区数＋

FAT 表数×FAT 表占扇区数＋根目录占扇区数＋分区首扇区号

②把逻辑扇区号转换成物理地址，即该扇区所在的道号、面号、区号。

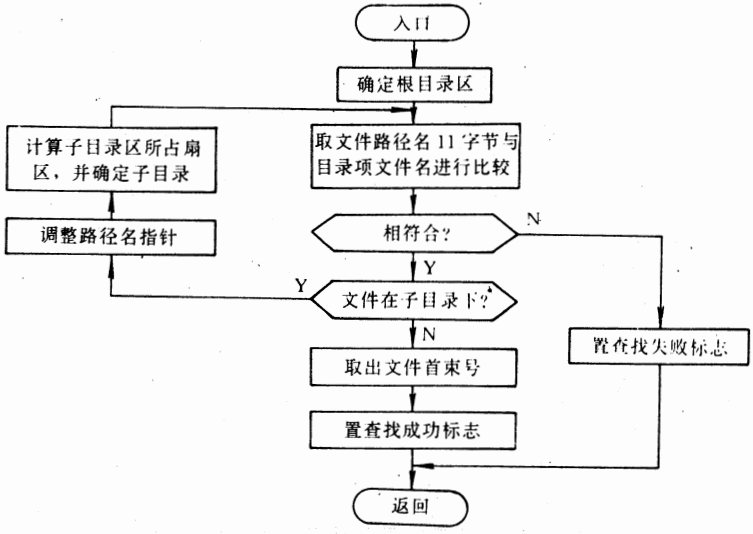


图 6-19 子程序 2 的流程

(4)子程序 4　子程序 4 的主要功能是由当前束号得出该文件中的下一个束之束号。这个子程序的入口参数是指定束号，它的出口参数是该束的下一束之束号。图 6-20 给出了子程序 4 的流程。

在子程序 4 中应引起重视的是 FAT 表项的跨扇区情况，这种情况只在使用 1.5 字节表项的 FAT 时发生，而在使用 2 字节表项的 FAT 时不会出现这种情况。因此，子程序 4 中必须对此作相应的处理。

(5)汉字字模的读取　汉字字模的读取是外存字库管理的主要目的，故这部分内容亦是其精华。我们可以利用上述几个子程序来获得所需汉字字模的存放地址，然后用 13H 号中断去读取其内

容。以下为读取外存字库中汉字字模的步骤。

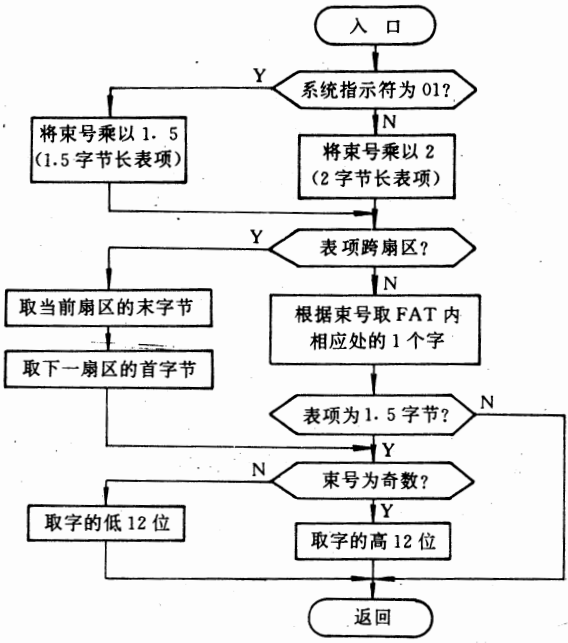


图 6-20 子程序 4 的流程

①根据汉字内码算出该汉字字模在外存字库中的序号 N,其计算公式已在上一节内给出过了;

②将 N 乘以汉字字模的字节数,得到该汉字字模在外存字库中

的起始字节号 m;

③根据 m 算出该字模在外存字库文件中束的相对束号;

④调用有关子程序获得 13H 号中断所需之入口参数;

⑤调用 13H 号中断,读取所需汉字字模至指定区域。

五、优化与讨论

从前面的分析和讨论来看，我们设计的多级型汉字库确实具有许多优点。它把内存和外存结合起来使用，不仅提高了内存的利用率，而且又充分利用了外存资源。所以，这种汉字库的 m 小，而且又保证其 v 有足够大，从而保证它的 k 参数值相当大。但是，我们也要看到，以上的设计并非十全十美，在不少地方还有优化的余地。下面我们就此进行一些讨论。

1. 从 Hash 函数的设计来看

我们在前面给出的 Hash 函数具有较好的效果，但是从其本身来看，它 的计算偏于复杂。它用到了两次乘法、一次除法和一次加法，其原因是为了尽可能使常用字库能把一些最常用字和常用字包含在内。其实，在不同的使用环境下，最常用字和常用字的定义也不同。如果完全依据汉字频度表的统计结果，又要使 Hash 函数设计得简单，那并非轻而易举的事情。需要对那些最常用字和常用字的内码分布作深入细致的分析，找出其中的规律。即使是这样，设计出的结果也往往不尽如人意。如果我们不完全拘于汉字频度表的统计结果来设计 Hash 函数，那末就比较容易设计出既简单又高效的函数。但是，这样作又可能会影响多级型汉字库中的内存字库的命中率。因此这是矛盾的两个方面，我们在设计 Hash 函数时应该兼顾到这两个方面。从这一点出发，我们有可能对前面设计的 Hash 函数进行简化。

接下来，我们对已设计的 Hash 函数作一些分析：

①由该 Hash 函数建立的常用字库中的 258 个字基本上在汉字频度表的前 1000 字以内(仅个别字例外)，常用字库内一半以上的字在汉字频度表的前 500 字内。

②根据 GB2312 可知，每个 Hash 链的平均链长 c，可由以下公式算出：

$$c=\frac{79\times 94}{129}=57.5$$

③由于乘法和除法指令执行的时钟周期比较长，约为 70～90，故该 Hash 函数值的计算约需 200 多个时钟周期，显然还不是太理想的。

④利用该 Hash 函数建立的定位表所占内存空间比较少。

经过以上分析后，我们可以从以下几个方面来考虑对 Hash 函数的优化：

①适当扩大 Hash 函数值的范围，以缩短 Hash 链长。

②在 Hash 函数中尽量少用乘法和除法，以加快计算函数值的速度。但是，为了使函数值散列均匀，除留取余法又是常用之方法，故这一点也应引起注意。

③从定位表结构来看，适当增加常用字库的容量，可使定位表所占内存空间减小。但是并不是说常用字库容量越大越好，理论上似乎是如此，但实际实现结果并不一定令人满意，有时会造成某些字频繁调进调出现象。因为内存字库容量是一定的，那末一旦常用字库扩大，则动态字库就会减小，这时出现上述现象的几率就会增大。

2. 从淘汰算法的合理性来看

淘汰算法的合理性已在前面作了一些介绍。这儿需要提醒大家注意的是，并非所有 Hash 链的链尾均在动态字库定位表中，有的链尾有可能在常用字库定位表中。如果 Hash 链的链尾在常用字库定位表内，那末这时就谈不上淘汰当前链中出现频度最小的汉字。若一旦发现这样的情况，那末应该淘汰哪一个汉字较合适呢？我们认为可以直接淘汰动态字库定位表首项所对应的汉字，这样作可以实现快速定位。但是这样作也存在不足之处，那就是这个被淘汰出去的字可能是使用频度很高的字。为了尽量作到被淘汰的字是目前出现次数最小的字，则可以通过检索整个动态字库定位表，然后选择一个“使用次数”最小的字淘汰之，不过这样肯定会

降低定位速度。所以，在上述情况下，究竟淘汰哪一个字才能获得较好的效果，这是一个值得探讨的问题。

3. 从减少读盘次数来看

从外存管理程序的算法来看，每次被调用时，最多只进行一次读盘取汉字字模操作，这似乎已经达到了最小限度了。其实不然，还可以对外存管理程序进行优化，以进一步减少读盘次数。

因为读盘缓冲区的大小为 512 个字节(一个扇区)，所以每次读盘时读入该区的不是 1 个汉字字模，而是 16 个汉字字模(指 16×16 点阵字模)。所以，当某个汉字字模不在内存字库而已在上次读盘时读到了缓冲区时，我们可以直接从缓冲区中取字模，而没有必要再去读盘，从而可以减少一次读盘操作，而且也省去了对该汉字字模在外存字库中位置的计算过程，加快了汉字的输出速度。对于连续输出内码相邻近的汉字，其效果更为明显。根据以上所述，我们可以对图 6-17 中的汉字库管理模块总体流程作相应的优化。

第四节　字形变换技术

在计算机汉字信息输出处理中，由于汉字具有多种不同的字体和形号，又加上汉字数目的繁多。因此想要在计算机中存贮各种不同大小字形的字库，这是几乎不可能的事，通常情况下，人们除了采用汉字库的压缩技术之外，还往往采用汉字字形的变换技术来获得各种字形。

汉字字形的变换技术就是通过对一种基本汉字库中的汉字进行放大(或缩小)、平滑等技术处理，以满足人们输出汉字的不同需要。

一、汉字字形的放大和缩小原理

汉字字形的放大和缩小就是采用对基本字形点阵元素数目的一种变换，不妨设基本字形点阵为 n×m，通过变换处理后的字形

点阵为 N×M，令 $\alpha=\frac{N}{n}$，$\beta=\frac{M}{m}$，则汉字字形变换的原理为：

当 α，β 均大于 1 时，此变换为汉字的放大；

当 α，β 均小于 1 时，此变换为汉字的缩小；

当 α<1 及 β>1 时，汉字横向扩大，纵向缩小；

当 α>1 及 β<1 时，汉字纵向扩大，横向缩小。

下面先介绍 α，β 为正整数时的整倍放大法。

二、汉字字形的整倍放大法

目前，以微处理机为中心的汉字信息处理系统中，大多具有对汉字库中存贮的汉字字形采取整倍放大的性能，它是经直接映射得出放大后的字形数据，供输出设备输出。

使用整倍放大法放大汉字可分为两种：一种是先纵扩，后横扩；另一种是先横扩，再纵扩。由于纵扩与横扩本质上区别不大，因此这两种方法实际上是一样的；下面对整倍放大法作一简单描述。

1. 横扩 β 倍

设基本汉字点阵为 n×m，横向为 x 方向，纵向为 y 方向，于是在横向扩大 β 倍，即扩大后的汉字点阵为 n×βm，也就是说放大后的汉字点阵在 x 方向为基本汉字点阵的 β 倍。

设：base__x 为基本汉字点阵 x 方向的元素个数

算法 1.1：横扩 β 倍的算法

①FOR　numx=1 TO base__x

②取基本汉字点阵 x 方向第 numx 条线送 L1

③FOR　henlarge__num=1 to β

④将 L1 送放大汉字点阵

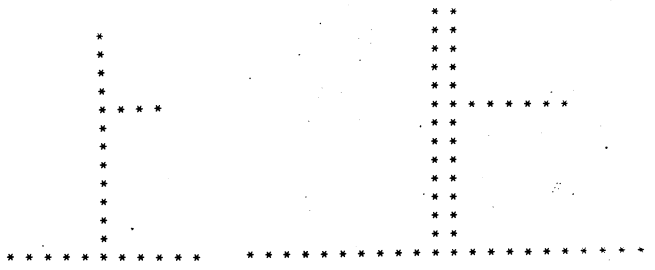
⑤NEXT henlarge__num

⑥NEXT numx

⑦END

例如，设基本点阵为 16×15，在横向扩大一倍，即为 16×30，

如图 6－21 所示。



(a) 基本字形点阵 (16×15)　　(b) 放大字形点阵 (16×30)

图 6-21　汉字横扩一倍字例

2. 纵扩 α 倍

设基本字形点阵为 n×m，在纵向扩大 α 倍，即扩大后的字形点阵为 αn×m，也就是说放大后的汉字点阵在 y 方向为基本汉字点阵的 α 倍。

设：base__y 为基本汉字点阵 y 方向的元素个数。

算法 1.2：纵扩 α 倍的算法

①FOR　numy＝1 TO base__y

②取基本汉字点阵 y 方向第 numy 条线送 L2

③FOR　venlarge__num＝1 TO α

④将 L2 送放大汉字点阵

⑤NEXT venlarge__num

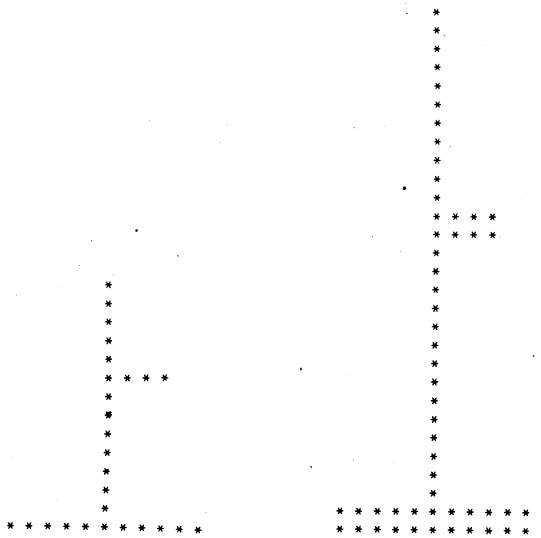
⑥NEXT numy

⑦END

例如，设基本点阵为 16×15，在纵向扩大一倍，即为 32×15，如图 6-22 所示。

3. 纵向、横向同时放大

设基本汉字点阵为 n×m，在纵向扩大 α 倍，横向扩大 β 倍，也就是说放大后的汉字点阵为 αn×βm。



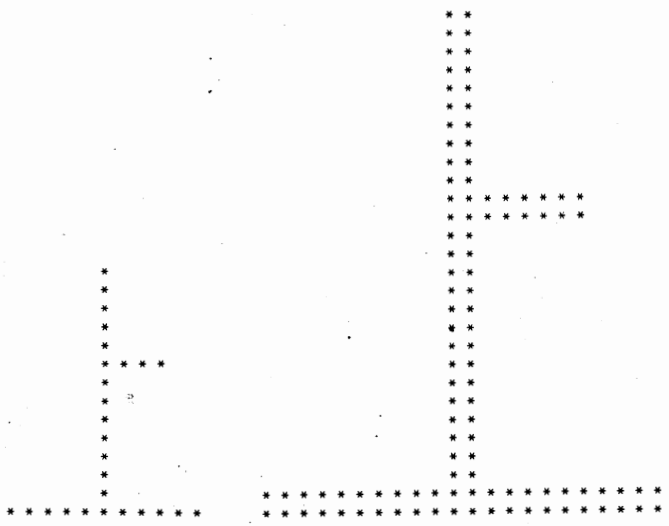
(a) 基本字形点阵（16×15）　　(b) 放大字形点阵（32×15）

图 6-22 汉字纵扩一倍字例

放大时先在内存中设立 n×m、n×βm 和 αn×βm 三个缓冲区。前一个用来存放基本字形点阵数据，中间一个用来存放横扩 β 倍的字形点阵数据，后一个用来存放放大后的字形点阵数据。先按算法 1.1 对基本字形点阵进行横扩 β 倍，存放在中间一个缓冲区中，然后以横扩 β 倍的字形点阵为基本字形点阵，按算法 1.2 进行纵扩 α 倍，存放在后一个缓冲区中，这样就 完成纵向横向同时放大的过程。

例如，基本字形点阵为 16×15，纵向横向同时扩大一倍后，字形点阵如图 6-23 所示。

综上所述，汉字字形的整倍放大法简单易行，在放大倍数不是很高的情况下，字体质量尚能保证，但当放大倍数很大时，往往会出现“锯齿”形，影响汉字字形的质量，对此我们可以利用平滑处理技术来提高放大后汉字字形的质量。



(a) 基本字形点阵(16×15)　　(b) 放大字形点阵 (32×15)

图 6-23 汉字纵向横向扩大一倍字例

三、平滑处理技术

有关汉字放大后的平滑处理的方法很多,在此仅介绍两种较为有效的平滑处理方法。

1. 简单补偿方法

对于任意 n×m 阶(0,1)矩阵 Z, 记第 i 行,第 i 列的元素为 x(i,j),简单补偿方法就是根据它邻近点的情况,进行简单补偿平滑处理。

点 x(i,j)的邻近点的情况为:

x(i-1,j-1)	x(i-1,j)	x(i-1,j+1)
x(i,j-1)	x(i,j)	x(i,j+1)
x(i+1,j-1)	x(i+1,j)	x(i+1,j+1)

平滑操作时,若:

$$(x(i,j-1)Vx(i,j+1))\&(x(i-1,j),Vx(i+1,j))=1 \qquad (1)$$

且 x(i,j)=0,则将 x(i,j)置为 1。

在 X 中将满足(1)式的所有元素 x(i,j)都处理一遍后就完成了一遍平滑处理,有时为了使字形质量更为美观,可以经过多遍平滑处理。如图 6-24 所示,其中约定 &,V,～(分别为 AND,OR,NOT 三种逻辑算符)。

(a) 原图　(b) 放大三倍　(c) 一遍平滑处理　(d) 二遍平滑处理

图 6-24 简单补偿方法

虽然,简单补偿方法简单有效,使用广泛,但是存在冗余效应,产生在不该补偿的地方补偿了多余的点,如图 6-25 所示。

其中 # 为冗余补偿点

图 6－25　简单补偿冗余效应

为了避免冗余点的产生,可以利用下面的结构补偿方法作平滑处理。

2. 结构补偿方法

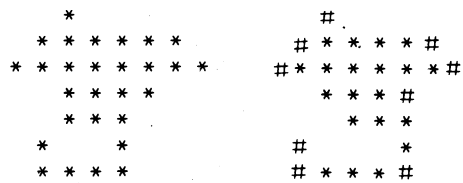
补偿点应该在图像中的什么位置呢?我们通过对补偿平滑后的图像仔细观察,不难发现,正常情况下补偿点出现在图像边缘突

出点的邻近，称边缘上的突出点为凸点，如图 6-26 所示。基于这个特性，我们不再象简单补偿方法那样满足(1)式的所有点都作补偿处理，而是采用先分离凸点，然后根据它和邻近点之间的关系，计算出需要进行补偿点位置的关联补偿点，建立与原图像同阶的补偿点阵。

在汉字字形放大时，关联补偿点补偿放大，在输出图像的指定位置并入预定形状的补偿块，我们这种将根据汉字字形的结构特征，进行平滑处理的方法称之为结构补偿方法。

(1)补偿点阵的确定　先令 S 为凸像点阵，其中 s(i,j)可由下式决定：

$$s(i,j)=x(i,j)\&\sim((x(i-1,j)\&x(i+1,j))\vee(x((i,j-1)\&x(i,j+1)) \quad (2)$$



(a) 一个点阵图像　　(b) 标记凸点图像

其中#表示凸点

图 6－26　点阵图像及其中的凸像点

容易验证(2)式能提取任意图像的凸点，所有的平滑操作补偿点都是在这些凸点的周围，但并不是说所有凸点的周围都要进行平滑处理。为此先对每个凸点的 2×2 邻域可能出现的组合情况进行分析。

令@为 s(i,j)＝1

为 s(i,j)=0,x(i,j)=1

$ 为 s(i,j)=(0,1),x(i,j)=(0,1)

0 为 x(i,j)=0

则与凸点有关的 2×2 邻域可分为如下 3 类情况：

①@$　　$@　　0$　　$0

　$0　　0$　　$@　　@$

②@$　　#$　　@$

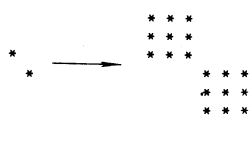
　$#　　$@　　$@　　约定记为* *

③$@　　$#　　$@

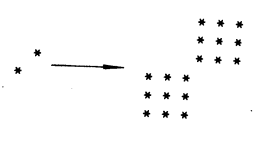
　#$　　@$　　@$　　约定记为* *

显然，对于第①类情况，$ 无需处理

对于第②和③类情况，放大后就会产生前面所说的“锯齿”形现象，因此放大后对这些凸像点邻近伴随补偿块，参见图 6-27 和图 6-28。

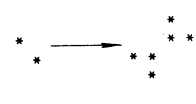


(a)



(b)

图 6-27 直接放大字形点阵



(a)

(b)

图 6-28 放大状态的伴随补偿块

如果我们将上述伴随补偿块并入直接放大图像的对应位置，放大图像边缘就能够平滑了。不过，由于* *和* *对应的补偿块

是互补的，所以可以用同一形式来简化表示，记[x]为伴随补偿点，令其定义在相关邻域的左上角，如图 6-29 所示，在放大情况下，该补偿点对应于一个菱形补偿块，如图 6-30 所示。

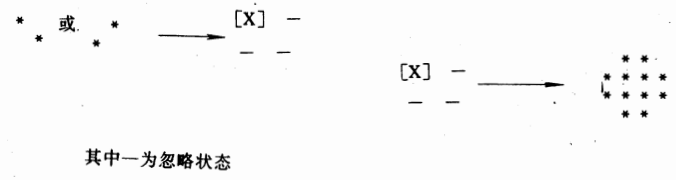


图 6-29 结构对应补偿点　　图 6-30 伴随补偿块

令 XCP 为补偿点阵，xcp(i,j)为 X 中伴随的第 i 行，第 j 列关联补偿点。

$$xcp(i,j)=x(i,j)\&x(i+1,j+1)\&(s(i,j)Vs(i+1,j+1))V$$
$$x(i,j+1)\&x(i+1,j)\&(s(i,j+1)Vs(i+1,j)) \quad (3)$$

有了 X 的补偿点阵 XCP，结构补偿放大平滑操作就很容易实现了。

(2)结构补偿方法的实现　为了实现结构补偿方法，先作下面的假设。

令 n×m 阶点阵 X 的第 i 行向量记为 X[i]

$$X[i]=(x(i,0),\cdots,x(i,j),\cdots,x(i,m-1))j\in[0,m)$$

X 可记为：

$$X=X_{n\times m}=(X_{(i)})_{i\in(0,n)}=(x(i,j))_{i\in[0,n]j\in[0,m]}$$

令 α、β 仍为上面所规定的正整数

则当矩阵 X 纵向扩大 α 倍，横向扩大 β 倍后，记放大后的矩阵为 $X_{(\alpha*n)\times(\beta*m)}$，用分块的形式表示为：

$$X_{(\alpha*n)\times(\beta*m)}=(X_{i,j})_{\substack{i\in(0,n)\\ j\in(0,m)}}=\begin{matrix}X_{0,0}\cdots Z_{0,j)}\cdots X_{0,m-1}\\ \vdots\quad\vdots\quad\vdots\\ X_{i,0}\cdots X_{i,j}\cdots X_{i,m-1}\\ \vdots\quad\vdots\quad\vdots\\ X_{n-1,0}\cdots X_{n-1,j}\cdots X_{n-1,m-1}\end{matrix}$$

每个 $X_{i,j}$ 为 n×m 阶(0,1)矩阵。

也就是说当 x(i,j)=0 时

$$X_{i,j}=\begin{bmatrix}0 & 0\cdots0\\ 0 & 0\cdots0\\ \cdots\cdots\cdots\\ 0 & 0\cdots0\end{bmatrix}_{\alpha\times\beta}$$

当 x(i,j)=1 时

$$X_{i,j}=\begin{bmatrix}1 & 1\cdots1\\ 1 & 1\cdots1\\ \cdots\cdots\cdots\\ 1 & 1\cdots1\end{bmatrix}_{\alpha\times\beta}$$

令⓪表示 & 或 V 算符，二元运算定义为：

$$\begin{aligned}C_{n\times m}&=A_{n\times m}⓪B_{n\times m}=(A[i]⓪B[i])_{i\in(0,n)}\\&=(a(i,j)⓪b(i,j))_{\substack{i\in(0,n)\\ j\in(0,m)}}\\&=(c(i,j))_{\substack{i\in(0,n)\\ j\in(0,m)}}=(C[i])_{i\in(0,n)}\end{aligned}$$

令～为非算符，一元运算定义为：

$$\begin{aligned}C_{n\times m}&=\sim A_{n\times m}=(\sim A[i])_{i\in(0,n)}\\&=(\sim a(i,j))_{\substack{i\in(0,n)\\ (o,m)}}\\&=(C(i,j))_{\substack{i\in(0,n)\\ j\in(0,m)}}=(C[i])_{i\in(0,n)}\end{aligned}$$

令 $\vec{X}[i]$ 表示 x[i]右移 1 位，称→为右移算符：

$$\vec{X}[i]=(0,x(i,0),x(i,1),\cdots x(i,m-2))_{i\in[0,n]}$$

同理令 $\overleftarrow{X}[i]$ 表示 $\vec{X}[i]$ 左移 1 位，称←为左移算符：

$$\overleftarrow{X}[i]=(x(i,1),x(i,2),\cdots,x(i,m-1),0)_{i\in[0,n]}$$

对于任意的 $X_{n\times m}$和 $F_{\alpha\times\beta}$直积算符(◇)定义为：

$$XF_{\alpha*n\times\beta*m}=X_{n\times m}\diamond F_{\alpha\times\beta}=(XF_{i,j})_{\substack{i\in(0,n)\\ j\in(0,m)}}$$

其中　$XF_{i,j}=(x(i,j)\&f(i,j))_{\alpha\times\beta}$

例如，　$X_{2\times2}=\begin{bmatrix}1 & 0\\ 1 & 1\end{bmatrix}$　　$F_{2\times2}=\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}$

$$XF_{4\times4}=X_{2\times2}\diamond F_{2\times2}=\begin{bmatrix}0 & 1 & 0 & 0\\ 1 & 1 & 0 & 0\\ 0 & 1 & 0 & 1\\ 1 & 1 & 1 & 1\end{bmatrix}$$

对于补偿点阵 XCP 和任意伴随补偿矩阵 $CP_{2*\alpha\times(2*\beta)}$：

$$CP_{2*\alpha\times(2*\beta)}=\begin{bmatrix}CP_{0,0} & CP_{0,1}\\ CP_{1,0} & CP_{1,1}\end{bmatrix},CP_{i,j}\text{为 }\alpha\times\beta\text{ 阶}(0,1)\text{阵},\quad i,j\in[0,1]$$

交积运算⊙定义为：

$$XCPS_{(\alpha*n\times\beta*m)}=XCP_{n\times m}\odot CP_{(2*\alpha)\times(2*\beta)}=(XCPS_{i,j})_{\substack{j\in[0,n]\\ j\in[0,m]}}$$

其中 $XCPS_{i,j}$为 α×β 阶(0,1)阵：

$$\odot CPS_{i,j}=xcp(i,j)\diamond CP_{0,0}\,Vxcp(i,j\text{-}1)\diamond CP_{0,1}\,Vxcp(i\text{-}1,j)\diamond CP_{1,0}\,Vxcp(i\text{-}1,j\text{-}1)\diamond CP_{1,1}$$

不难验证 XCPS 为点阵 $X_{n\times m}$放大后的结构补偿点阵。

再根据上述有关的定义，不难得到：

$$S[i]=X[i]\&(X[i\text{-}1]\&X[i+1]V\vec{X}[i]\&\vec{X}[i])\quad i\in[0,n]$$

$$XCP[i]=X[i]\&[i+1]\&(s[\dot{}]V\tilde{S}[i+1]V[i]\ \&X[i+1]\&(\tilde{S}[i]Vs[i+1]))\qquad i\in[0,n)$$

则平滑处理放大后的汉字字形点阵 MX：

$$MX_{(\alpha*n)\times(\beta*m)}=X\diamond F_{\alpha\times\beta}VXCPS=X_{n\times m}\diamond F_{\alpha\times\beta}Vxcp\odot CP$$

四、点阵汉字无级变倍方法

当 α、β 为任意实数时，称之为无级变倍方法，无级变倍的方法

很多，在此介绍两种比较常用的方法。

1. **抽线加线无级放大法**

(1)无级压缩法　对基本汉字点阵进行压缩($\alpha<1,\beta<1$)，最直观的方法就是抽去一定数目的点线，但必须满足下列两个条件：

①保证汉字的基本结构不变；

②在压缩了的汉字中，保留被删去的线的黑白特征。

设 Compress-x 为压缩后的汉字点阵 x 方向的线数

算法 1.3：抽线判别算法

①del__x=base__x−Compress__x

②flag=0

③del-flag=[base__x/del__x]

④for numx=1 to base__x

⑤flag=flag+1

⑥if flag=del__flag then

⑦抽去第 numx 条竖线(见算法 1.4)

⑧flag=0

⑨endif

⑩next numx=1

⑪end

为了符合上面提出的两个条件，在抽去线时将与该相邻的线进行逻辑运算操作，以保证字形的质量。

设 al 为要抽去线，ah 为右边相邻的线，dl 为左边相邻的线，dh 为左边第二条线

算法 1.4：抽去竖线的算法

①l_1=dl AND al

②l_2=(dl XOR al) AND dl

③l_3=(ah XOR al) AND al

④dl=l_1 OR l_2 OR l_3

⑤删去 al

⑥end

以上介绍的是横向压缩的方法，同理可以进行纵向压缩，从而实现整个字体的压缩。

(2)无级放大法　汉字无级放大与压缩相反，它是采用均匀加线的方法。

设 enlarge__x 为放大汉字点阵 x 方向的线数

则 β=enlarge__x/base__x，当 β 为整数时，前面已作介绍，在此设 β 不为整数。

算法 1.5：加竖线的算法

①Add__x=enlarge__x__base__x * [β]

②flag=0

③add__flag=[base__x/Add__x]

④for numx=1 to base__x

⑤取基本汉字点阵 x 方向第 numx 条线送 L

⑥for enl__num=1 to [β]

⑦将 L 送放大汉字点阵

⑧next enl__num

⑨flag=flag+1

⑩if flag=add__flag then

⑪将 L 送放大汉字点阵

⑫flag=0

⑬endif

⑭next numx

⑮end

用同样的方法，进行纵扩 α 倍，然后将放大后的点阵汉字进行平滑操作，得到字体质量较高的汉字字形。

2. 公式变换法

采用下述公式可以对字形点阵行列元素进行任意倍数的无级变换，假定基本汉字字形点陈 X 为 n×m，无级放大后的点阵 Y 为 N×M。原来的字形点阵用{x(i,j)}表示，其中 i=1,2,3,…,n,j=1,2,…,m，无级放大后的字形点阵用{y(k,l)}表示，其中 k=1,2,…,N,l=1,2,…,M。

(1)无级放大(n<N,m<M)

令$[\frac{n}{N}\times k]=i,[\frac{m}{M}\times l]=j$

约定用[a]表示 a 的整数部分，$\langle a\rangle_b$ 表示$\frac{a}{b}$的余数部分。

显然，k=1,2,…,N,l=1,2,…,M 时实现字形点阵 Y 到字形点阵 X 中元素的下标变换。

再令 $x(i,j)=X_1$　　$x(i+1,j)=X_2$

$x(i,j+1)=X_3$　　$x(i+1,j+1)=X_4$

$f_k=\min(n,N-\langle nk\rangle_N)$

$f_l=\min(m,M-\langle ml\rangle_M)$

使得：$y(k,l)=f_k\cdot f_l\cdot X_1+(n-f_k)\cdot f_l\cdot X_2$

$+f_k(m-f_l)X_3+(n-f_k)(m-f_l)X_4$　　(4)

那末，y(k,l)的值就表示了无级放大后的小方格对“1”的隶属程度。由于 $X_i(i=1,2,3,4)=0$ 或 1，显然有不等式 $0\leqslant y(k,l)\leqslant n\cdot m$成立，即有 $0\leqslant\frac{y(k,l)}{n\cdot m}\leqslant 1$ 成立。于是 N×M 阶矩阵$\{\frac{y(k,l)}{n\cdot m}\}$的所有元素值都在 0 与 1 之间，选取一个适当的阈值 $\gamma(0\leqslant\gamma\leqslant 1)$，当$\frac{y(k,l)}{nm}\geqslant\gamma$ 时，规定 y(k,l)=1。当$\frac{y(k,l)}{nm}<\gamma$ 时，规定 y(k,l)=0，其中 k=1,2,…,N,l=1,2,…,M。这样就得到了放大后的字形点阵{y(k,l)}

(2)无级压缩(n>N,m>M)算法　根据上面的讨论，放大时{y(k,l)}中每个元素和{x(i,j)}中的四个相邻元素有关，而缩小变换时，{y(k,l)}中的每个元素的值可能和{x(i,j)}中的 9 个元素有关，容易使原来分离的笔画粘连，所以不宜直接应用放大算法。为

了避免这种情况，先引入下面的约定。

我们约定：在基本字形点阵中与元素{y(k,l)}相对应的小方格S中划一个小方格S1，使S1的中心与S的中心重合。S1的面积等于S面积的$(\frac{N\cdot M}{n\cdot m})$倍，那么，S1叫做S的特征方格；由于$\frac{N\cdot M}{n\cdot m}<1$，再令：

$$[\frac{n}{N}\times k+\frac{n-N}{2N}]=i,[\frac{m}{M}\times l+\frac{m-M}{2M}]=j$$

那末$X_i(i=1,2,3,4)$的定义如前。

当$k=1,2,\cdots,N$　$l=1,2,\cdots,M$时可实现由字形点阵Y到字形点阵X的下标变换。

再令：$f_k=N-\langle nk+\frac{n-N}{2}\rangle N$

$$f_l=M-\langle ml+\frac{m-M}{2}\rangle M$$

使得：

$$y(k,l)=f_k\cdot f_l\cdot X_1+(n-f_k)\cdot f_l\cdot X_2+f_k\cdot(m-f_l)\cdot X_3+(n-f_k)\cdot(m-f_l)\cdot X_4 \qquad (5)$$

这里$X_i(i=1,2,3,4)$的值仍为0或1，用和放大算法类似的方法，只要选取一个适当的阈值$\gamma(0\leqslant\gamma\leqslant1)$，就可得到压缩后的字形点阵{y(k,l)}。

(3)整形判别条件及修正量△的计算　在对字形质量要求不太高的情况下，上述变换方法均有实效，但在字形质量要求较高的情况下，由于上述变换方法所得到的字形点阵，有可能在直线段的端部和交叉产生变形(如图6-31所示)，所以仍需整形处理。

现在我们来进行整形处理，处理的方法是当满足一定的逻辑条件时，使y(k,l)的值加上一个修正项后，将能取消变形。

我们先讨论放大时的情况。

当$\begin{cases}\langle nk\rangle_N>N-n\\ \langle ml\rangle_M>M-m\end{cases}$

$$\text{如果}\begin{cases}X_1\cdot\bar{X}_2\cdot\bar{X}_3\cdot\bar{X}_4\cdot\bar{x}(i\text{-}1,j+1)\cdot\bar{x}(i+1,j\text{-}1)=1\\ \text{或 } X_1\cdot\bar{X}_2\cdot\bar{X}_3\cdot\bar{X}_4\cdot\bar{x}(i,j+2)\cdot\bar{x}(i+2,j)=1\end{cases}\tag{6}$$

$$\text{则}\quad \triangle_1=\min[(n\text{-}N+\langle nk\rangle_N)\cdot(M\text{-}\langle ml\rangle_M),\ (N-\langle nk\rangle_N)\cdot(m\text{-}M+\langle ml\rangle_M)]\tag{7}$$

$$\text{如果}\begin{cases}\bar{X}_1\cdot\bar{X}_2\cdot\bar{X}_3\cdot\bar{X}_4\cdot\bar{x}(i,j-1)\cdot\bar{x}(i+2,j+1)=1\\ \text{或 } \bar{X}_1\cdot\bar{X}_2\cdot\bar{X}_3\cdot\bar{X}_4\cdot\bar{x}(i-1,j)\cdot\bar{x}(i+1,j+2)=1\end{cases}\tag{8}$$

$$\text{则}\quad \triangle_2=\min[(n\text{-}N+\langle nk\rangle_N)\cdot(m\text{-}M+\langle ml\rangle_M),\ (N-\langle nk\rangle_N)\cdot(M\text{-}\langle ml\rangle_M)]\tag{9}$$

(a) 变换前　　(b) 变换后

图 6-31　由公式变换产生的直线段端部和交叉处的外形

这 min 表示取极小值的意思，修正项$\triangle_1$，$\triangle_2$将使与直线段端部相应的那些 y(k,l)的值增加，从而防止本来该为“1”的元素变为“0”。

$$\text{如果}\begin{cases}\bar{X}_1\cdot X_2\cdot X_3\cdot X_4[x(i\text{-}1,j+1)\cdot x(i+1,j\text{-}1)+x(i\text{-}1,j+1)\\ \bar{x}(i+2,j)+x(i+1,j\text{-}1)\cdot\bar{x}(i,j+2)]=1\\ \text{或 } X_1\cdot X_2\cdot X_3\cdot\bar{X}_4[x(i,j+2)\cdot x(i+2,j)+x(i,j+2)\\ \bar{x}(i+1,j-1)+x(i+2,j)\cdot\bar{x}(i-1,j+1)]=1\end{cases}\tag{10}$$

则$\triangle_3=-\triangle_1$

$$\text{如果}\begin{cases}X_1\cdot\bar{X}_2\cdot X_3\cdot X_4[x(i,j-1)\cdot x(i+2,j+1)+x(i,j-1)\cdot\\ \bar{x}(i+1,j+2)+x(i+2,j+1)\cdot\bar{x}(i-1,j)]=1\\ \text{或 } X_1\cdot X_2\cdot\bar{X}_3\cdot\bar{X}_4[x(i-1,j)\cdot x(i+1,j+2)+x(i-1,j)\\ \bar{x}(i+2,j+1)+x(i+1,j+2)\cdot\bar{x}(i,j-1)]=1\end{cases}\tag{11}$$

则　$\triangle_4=-\triangle_2$

修正项$\triangle_3$、$\triangle_4$将使与直线段交叉处相应的那些 y(k,l)的值减少，从而防止本来应该为“0”的元素为“1”。

我们再讨论缩小时的情况。

此时只要将放大时情况的计算公式加以适当改变就可运用。

当$\begin{cases}\langle nk\rangle_N > n-N \\ \langle ml\rangle_m > m-M\end{cases}$

如果(6)式成立，则：

$$\triangle_5=\min[\langle nk+\frac{n-N}{2}\rangle_N \cdot (M-\langle ml+\frac{m-M}{2}\rangle_M], \quad (12)$$

$$(N-\langle nk+\frac{n-N}{2}\rangle_N)\cdot\langle ml+\frac{m-M}{2}\rangle_M]$$

如果(8)式成立，则：

$$\triangle_6=\min[(N-\langle nk+\frac{n-N}{2}\rangle_N \cdot (M-\langle ml+\frac{m-M}{2}\rangle_M], \quad (13)$$

$$\langle nk+\frac{n-N}{2}\rangle_N)\cdot\langle ml+\frac{m-M}{2}\rangle_M]$$

如果(10)式成立，则 $\triangle_7=-\triangle_5$。

如果(11)式成立，则 $\triangle_8=-\triangle_6$。

由此可见，判别整形条件和进行修正量△的计算比较费事，在实际计算时，N 比 n 和 M 比 m 相差较大，并且同时满足条件 $\langle nk\rangle_N > |N-n|$ 和 $\langle ml\rangle_M > |M-m|$ 的 y(k,l) 只占全部 y(k,l) 的小部分，再注意到只有在 $\sum_{i=1}^{4} x_i = 1, y(k,l)/n \cdot m$ 的值大于阈值 γ 时才需要进行判别，所以进行整形处理时需要的机器时间并不多。总之，经过整形处理后的字形质量就得到了很大的提高。

第五节　假脱机打印子系统

一、概述

1. 问题的提出

DOS 是一个典型的单任务系统，它只能支持一道程序运行。在这道程序中使用处理器进行数据计算处理和使用外部设备进行

I/O 操作是串行进行的。这势必影响程序运行的速度，也限制了系统效率的提高。

汉字系统必须具有汉字 I/O 功能。众所周知，汉字的 I/O 速度要慢于西文字符的 I/O 速度，如果采用西文外部设备来实现汉字 I/O，则尤其是如此。在单任务汉字系统中，由于处理器与外部设备的串行执行，将大大影响汉字系统的效率和性能。

在中、大型操作系统中，广泛采用假脱机(SPOOLing)技术来实现处理器与外部设备之间的高度并行工作。图 6-32 给出了假脱机技术的示意图。

从图 6-32 中可以看到，假脱机输入模块利用通道把输入设备

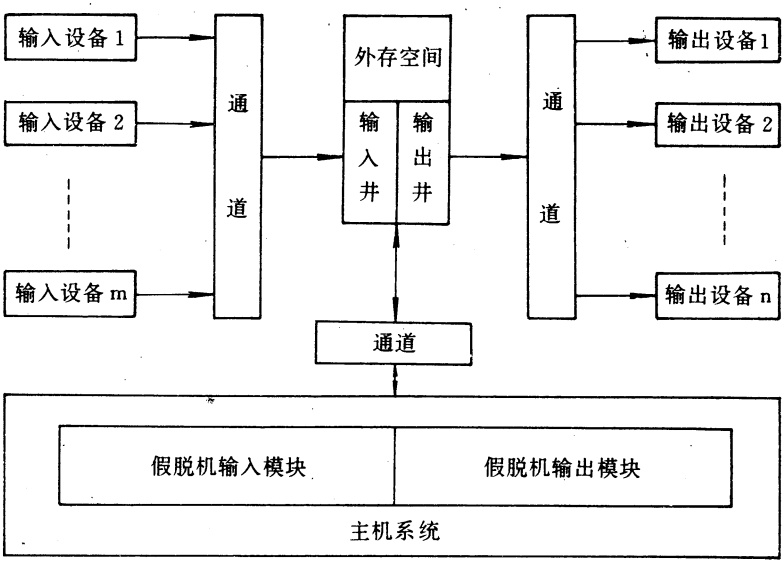


图 6-32　假脱机技术示图

上的信息输入至输入井，假脱机输出模块利用通道把输出井内的信息输出至输出设备。然而，用户程序就从输入井取输入信息，并

把输出信息存入输出井。从而实现了处理器与外部设备的并行工作。

我们可以按照这种方式为 DOS 设计一个假脱机子系统，使处理器与外部设备并行工作。这样，在基于 DOS 的汉字系统中，即使因汉字 I/O 降低了系统 I/O 的速度，但不会降低用户程序的执行速度，也就不会过份降低系统的效率。

2. 实现的可能性

假脱机子系统工作的硬件基础是中断装置和通道。在中、大型系统中，这两者均具备，而在 PC 系列机中，只有中断装置而没有通道。因此，我们必须考虑采用处理器的分时技术来建立虚拟通道，也就是让处理器用一定的时间片来仿真通道工作。目前 PC 系列机的处理器具有较高的主频，为我们采用分时技术提供了条件。有了虚拟通道，就使假脱机子系统所需的硬件要求得到满足。

在 DOS 系统中，每隔 55ms 就发生一次时钟中断，并执行一次时钟中断处理程序(8H 号中断处理程序)。我们可以利用这个中断处理程序来实现分时。

另外，DOS 支持的慢速设备主要是打印机，而且打印机是每个用户程序基本上都可使用的外部设备，因而它的工作较频繁。所以，我们考虑假脱机打印子系统的设计。

二、设计思想

在 DOS 中，当程序需要打印输出数据时，就直接把数据送往打印机。由于打印机的响应速度较慢，故程序必须等待较长时间待打印机完成输出后才能继续执行下去。我们在建立假脱机打印子系统后，当程序需要打印输出数据时，就把输出数据送入输出井，由于输出井在磁盘上，故其响应速度要比打印机快得多。所以程序很快就能继续执行下去。然后，假脱机打印子系统利用分时把输出井中的数据送往打印机。显然这时的打印过程与程序的执行是并发的。上述过程如图 6-33 所示。

为了实现在程序打印数据时把数据不送往打印机，而送往输出井，我们必须要改造 DOS 的打印机驱动程序(17H 号中断处理

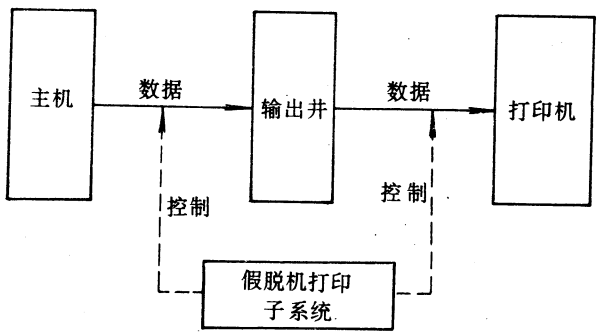


图 6-33　假脱机打印子系统的地位

程序)。为了实现分时，必须改造 8H 号中断处理程序。经过分析可知，8H 号中断处理程序中调用了 1CH 号中断处理程序，而这个中断处理程序实际上仅有一条返回指令，显然是留给用户扩充用的。因此，我们可以利用 1CH 号中断处理程序来实现把输出井中的数据送打印机打印。

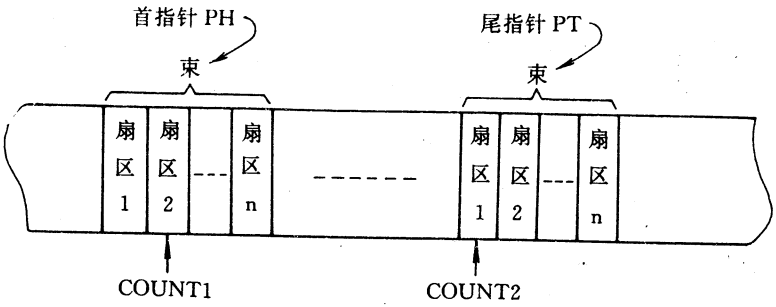
三、数据结构

1. 输出井

输出井以文件的形式建立在硬盘上，假设取名为 WELL，其长度为 200K 字节，设计为环形队列结构。为了减少续盘次数，以加快速度和减少程序的驻留长度，我们在执行程序中将系统分配给 WELL 的束号读入内存，在将来访问 WELL 时可以直接使用之。

为了实现对 WELL 的访问，我们为它设置了首指针 PH 和尾指针 PT，它们分别指向 WELL 中有效信息的首和尾的束号。另外我们还定义了两个扇区指针 COUNT1 和 COUNT2，它们分别指

向 WELL 中有效信息的首和尾的扇区号(束内扇区号)。也就是说,PH 和 COUNT1 确定了 WELL 中有效信息之首,PH 和 COUNT2 确定了 WELL 中有效信息之尾。图 6-34 给出了输出井 WELL 与其指针的关系。



6-34 输出井与其指针

为了实现 WELL 的环形队列结构的需要,我们还定义了一个输出井计数器 CT,用于记录输出井内已使用的扇区数。

2. 缓输出表

DOS 系统可以支持四台打印机,所以输出井内的内容不一定在同一台打印机上输出,因此应该记录输出井中内容的归属情况(即往哪台打印机输出)。为此,我们在假脱机打印子系统中设置了一张缓输出表。该表由若干个表项组成,每个表项对应于输出井内的一批数据,它记录了这批数据的长度和将来要送往的打印机的号码。图 6-35 给出了缓输出表的结构。

每个缓输出表表项长 3 字节,用 1 个字节记录打印机号码,用 2 个字节记录缓输出数据的字节数。该表共 50 项,占 150 个字节。

每个表项可以记录 64K 字节。输出井 WELL 为 200K 字节,若系统仅使用一台打印机的话,只需 4 个表项就能满足要求了。但是,考虑到有使用多台打印机的可能,以及多台打印机输出内容定序的要求,因此定义缓输出表为 50 个表项,当然这已足够用了。

为了使缓输出表的管理简单化,我们亦把它定义为环形队列

结构，并设置了首指针 TP1、尾指针 TP2 和表项计数器 TC。首指针 TP1 和尾 指针 TP2 分别指向缓输出表中有效表项之首和尾，表项计数器 TC 记录了当前缓输出表内有效表项的个数。

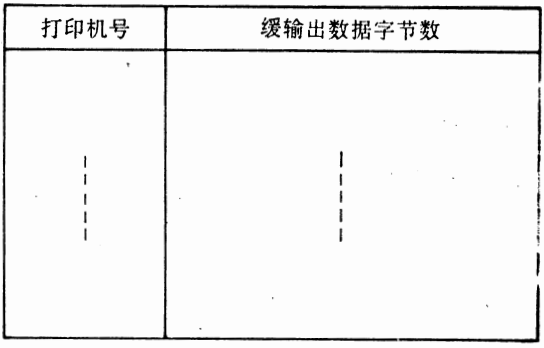


图 6-35　缓输出表的结构

3. 缓冲区

对磁盘的读写操作是以扇区为单位进行的，因此我们在内存中开辟了两个缓冲区，它们为输入缓冲区 BUFF1 和输出缓冲区 BUFF2。这儿的输入和输出是针对假脱机打印子系统而言的。这两个缓冲区均为 512 字节长。

根据我们的设计，用户程序欲送往打印机的数据先被送入输入缓冲区 BUFF1，待其送满后，则把它写入输出井，假脱机打印子系统再把输出井中的数据读入输出缓冲区 BUFF2，然后再逐一送打印机输出。注意，在某些情况下，BUFF1 中的数据可以直接进入 BUFF2，并送打印机输出。

我们还为 BUFF1 和 BUFF2 分别设置了指针和计数器，以实现对它们的管理和使用。

四、实现方法

1. 打印驱动程序的改造

打印驱动程序就是17H号中断处理程序的0号功能块，它实现把字符送打印机输出。因此，要将打印输出的数据送入输出井，就必须改造这个功能块，因为用户程序都是通过直接或间接调用此功能块来打印输出数据的。图6-36给出了改造后的0号功能块(打印驱动程序)的流程。

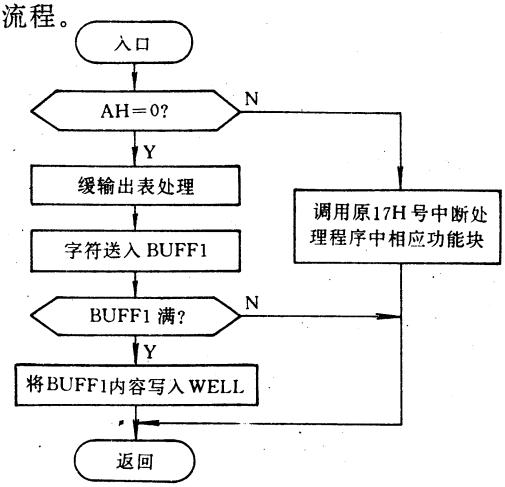


图6-36 改造后的打印驱动程序

为了保证使用假脱机打印子系统后，仍保持与DOS兼容，因此我们除了保证17H号中断处理程序的1、2、3号功能块作用不变外，还要保证0号功能块的入口和出口参数都保持不变。下面详细介绍改造后的0号功能块的情况。

(1)入口和出口参数　入口参数：AL＝打印字符，DX＝打印机号

出口参数：AL＝打印字符，AH＝状态字节

以上入口和出口参数与原来的0号功能块完全一致。所要说明的是，我们在改造后的0号功能块中，把出口参数AH总是置为0，表示打印机不出错。显然这是可以理解的，因为在这儿实际上并未用到打印机，而只是用了输出井，故打印机不会出错。

(2)现场的保护　按照惯例，在程序的开头要保护所用到的全

部寄存器，但由图 6-36 可知，仅当 BUFF1 满时才执行写盘操作，我们知道，只有在这时才会使用多个寄存器，因此我们可以在这时才进行对所用寄存器的保护。这样，在对 0 号功能块的一系列调用过程中，可以大大减少因保护现场所花的时间，也就加快了程序的执行速度。

(3)缓输出表的处理　缓输出表的处理就是在缓输出表的相应表项中记录当前的输出情况。我们可以按以下过程来完成这项工作：先根据缓输出表首指针取出表项，判断该表项是否为空表项(若缓输出数据字节数为 0，则该表项为空表项)，如为空表项，则填入打印机号，然后把缓输出数据字节数加 1；如该表项不是空表项，则判断其打印机号与当前使用的打印机号是否一致，若不一致，则将指针移至下一个表项。如果已使用满了 50 个表项，则要等待数据输出后腾出空表项后再处理；若表项的打印机号与当前使用的打印机号一致，则把该表项的缓输出数据字节数加 1，当该字节数满 64K 时，则指针移至下一个空表项。另外，由于缓输出表为环形队列结构，所以当指针指到表尾时，要自动调整至表首。

(4)输出井的写入　在完成缓输出表处理后，就要把打印的数据写入输出井。这时只要把 AL 寄存器中的内容送入 BUFF1，并调整 BUFF1 的指针和计数器。如果这时 BUFF1 中的数据未满 512 个字节，则到此结束。这无疑要比把数据送打印机输出快得多。

当 BUFF1 中的数据满 512 个字节时，就要进行写盘(输出井)操作。首先取输出井的指针 PT 和 COUNT2，获得当前要写入内容的束号和束内扇区号，再把 BUFF1 中的内容写入此扇区，然后把 BUFF1 的指针和计数器复位。最后要调整 PT 和 COUNT2，如果 COUNT2 指向的不是束内最后一个扇区，则只要把 COUNT2 加 1 即可；否则要把 PT 指向 WELL 的下一束，并且把 COUNT2 清为 0。由于对磁盘的操作速度要远快于打印机，所以即使发生写盘操作，其速度亦要比把数据送打印机快得多。

2. 分时调度程序

分时调度程序的作用是实现用户程序和缓输出在宏观上的并行执行。DOS 中的时钟中断处理程序每隔 55ms 运行一次，这样在用户程序和时钟中断处理程序之间实际上已经实现了分时，因此我们可以通过扩充时钟中断处理程序来完成缓输出。我们在时钟中断处理程序运行的时间片内调用缓输出程序，从而实现用户程序和缓输出之间的调度。图 6-37 给出了分时调度的微观情况。

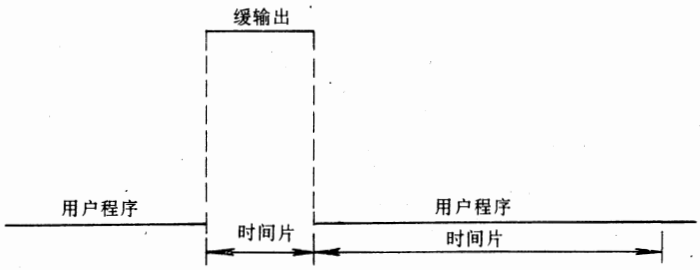


图 6-37 分时调度的微观情况

从图 6-37 中可以看出，这里采用了简单的轮转调度法来实现用户程序和缓输出程序间的调度，使它们在宏观上并发执行。我们把调度的时间片定为 55ms 的 6 倍(即 330ms)，系统在有数据输出的情况下，每隔 330ms 进行一次缓输出操作。也就是说，在发生 6 次时钟中断后，才进行 1 次缓输出处理。

我们已经在前面介绍过 1CH 号中断处理程序的特殊地位，所以我们可以把分时调度程序编写成 1CH 号中断处理程序。我们考虑到，DOS 的其它程序亦有可能会用到 1CH 号中断处理程序，为了避免与这种情况发生冲突，我们在修改 1CH 号中断向量之前，应该将其原来之值保存好，在我们的 1CH 号中断处理程序结束时，转该入口执行。这样，原来的 1CH 号中断处理程序的功能仍然保留。图 6-38 为分时调度程序的主体流程。

下面进一步介绍分时调度程序的设计和实现。

(1)打印处理　打印处理部分的主要工作是把输出井 WELL 中的数据送打印机输出。打印处理的流程如图 6-39 所示。

从图 6-39 的流程中可以看出，并非每发一次时钟中断就作一次打印处理。只有在有打印输出数据，并且时间片已到的情况下，才有可能 进行打印处理。除此之外，流程中不作任何处理而立即退出。

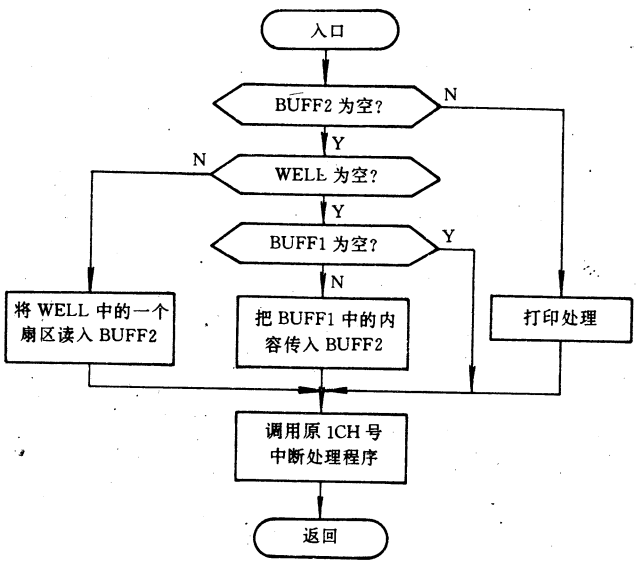


图 6-38 分时调度程序的主体流程

我们把时间片定为 330ms 是有根据的，如果把时间片定得过大，则要过较长时间才进行一次打印处理，使整个打印输出速度降低太多；如果把时间片定得过小，则会较频繁地进行打印处理，尽管打印输出

速度快了，但是过多地分用了用户程序的执行时间，从而使得用户的程序的运行速度明显降低，并会出现停顿现象。我们通过试验，得出把时间片定为 330ms，每次打印处理 15 个字符，可以获得较

满意的效果。

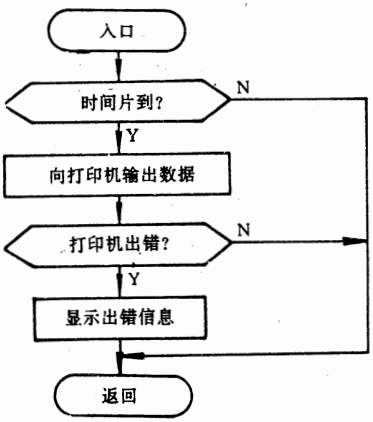


图 6-39 打印处理的流程

我们在具体实现中还发现这样一种情况，在采用假脱机打印子系统后，若一味追求打印速度，在其达到一定程度后，则会出现一种不能令人满意的现象。这时用户程序送出的打印数据会很快被打印出来，而未被或很少被送入输出井 WELL 中。显然，这样不但未能改善系统效率，而且使系统效率降低了。

我们可以这样来解决这个问题，在有大量数据到来时，应降低打印速度，在其它情况下，应加快打印速度。因此，我们在 17H 号中断处理程序的 0 号功能块中加入一段程序，使时间得以延长。如果频繁调用 0 号功能块(意味着输出数据多)，打印速度就降低，若调用不频繁，速度就比较正常。事实证明，这种方法是行之有效的。

从图 6-38 中可以知道，BUFF2 中存放有待打印输出之数据，所以我们首先要从缓输出表中取出打印机号，然后从 BUFF2 中取出一个数据，再调用原来的 17H 号中断处理程序的 0 号功能块，把它送打印机输出。这样就完成了字符的打印输出。如果字符打印输出后未出错(根据 0 号功能块的出口参数 AH 可知)，则进行 BUFF2 和缓输出表指针及计数器的调整，再打印下一个字符，直到发满 15 个字符或 BUFF2 空为止。若字符未能正确输出，则要进行出错处理。由于出错处理后不进行指针和计数器的调整，这就保证了当时数据不会丢失。

(2)BUFF2 的填入　填写 BUFF2 的数据来源有两个，一个是建立在硬盘上的输出井 WELL，另一个是输入缓冲区 BUFF1。当 BUFF2 空而且 WELL 不空时，说明有数据要打印，因此我们可以根据 WELL 的首指针 PH 和 COUNT1，将 WELL 中一个扇区的内容送入 BUFF2，完成此操作后对 WELL 指针的调整完全与打印驱动程序中相应部分相同。当 BUFF2 空，WELL 也空时，若 BUFF1 不空，则仍表示有数据要打印，这时就要把 BUFF1 中的内容送入至 BUFF2，然后把 BUFF1 的指针及计数器复位。当然，BUFF2 的指针和计数器均要作相应的调整。

由于以上向 BUFF2 填入数据已经花费了一段时间，故在填好数据后就不再进行打印处理，而应该立即退出。BUFF2 中的内容在下次调度到时，再向打印机输出。另外，装填 BUFF2 不是每隔 330ms 进行一次，而是每隔 55ms 进行一次，这样作有助于数据的尽快输出。

3. 子系统的自举

假脱机打印子系统包括 17H 号中断处理程序和分时调度程序，为了将它们装入内存并形成假脱机打印子系统的内存映像，必须设计一个自举程序。该自举程序除了完成把 17H 号中断处理程序和分时调度程序驻留内存外，还要为所定义的指针、计数器和工作区赋初值。

由于系统所用的硬盘与操作系统的版本有多种，因此会带来不少参数(如 FAT 表项长度和每束含扇区数等)的不同，因此自举程序首先就要读入硬盘的 BPB 参数表，以获得当前的重要参数。

然后，通过检索目录表，确定 WELL 文件的首束号，再把 WELL 所用各束的束号一一读入内存中形成一个束号表。由于 WELL 的长度为 200 字节，而每束含的扇区数不是常数，因此应该按可能的最大值来确定束号表的长度(200 字节)。

按下来自举程序就为 WELL 和缓输出表等数据结构的指针

和计数器设置初值，再把原来的 17H 号和 1CH 号中断向量保存好，然后设置新的 17H 号和 1CH 号中断向量。最后，自举程序把新的打印驱动程序和分时调度程序驻留在内存中。

五、状态的切换

由于一般打印机上汉字的输出要比西文的输出慢得多，所以在汉字系统中使用假脱机打印子系统就显得格外重要。我们知道，在一般打印机上输出汉字，必须装入具有汉字功能的打印驱动程序，这时要修改 17H 号中断向量。如果先装入假脱机打印子系统，再装入汉字打印驱动程序，则会使假脱机打印子系统中的打印驱动程序丢失。另外，在装入假脱机打印子系统后，不能排除用户程序中修改 17H 号和 1CH 号中断向量的可能。如果发生了这样的情况，显然假脱机打印子系统就会崩溃。

为了解决上述问题，我们可以设计一个外部命令 SPOOL，用以切换系统的状态，使用户可以自由地选择是否进行假脱机打印方式的操作。当执行 SPOOL 命令时，首先显示出当前是否在假脱机打印状态，并询问用户是否要改变此状态，若答复肯定，则改变之，否则退出。SPOOL 命令的流程如图 6-40 所示。

要判断当前是否是假脱机打印状态，只要判断当前的 17H 号和 1CH 号中断向量是否是假脱机打印子系统内打印驱动程序和分时调度程序的入口地址。只有当这两个中断向量均符合上述条件时，才能确认当前为假脱机打印状态。

如果当前是假脱机打印状态，欲取消此状态是很容易的，我们只要把原来的 17H 号和 1CH 号中断向量恢复即可。这两个中断向量在假脱机打印子系统自举时被保存在指定单元中。在非假脱机打印状态下，用户可以任意修改 17H 号和 1CH 号中断向量。若在此状态下欲改变为假脱机打印状态，则同样十分简单，只要把假脱机打印子系统中的打印驱动程序和分时调度程序的入口地址，分别写入 17H 号和 1CH 号中断向量即可。

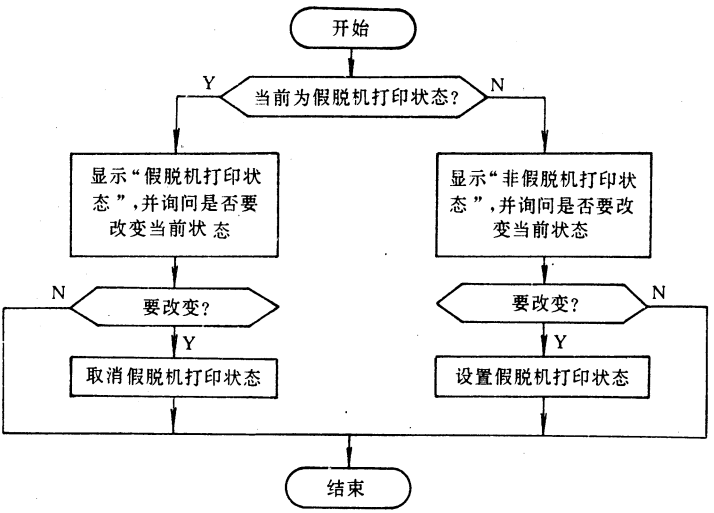


图 6-40 SPOOL 命令的流程

六、小结

实践证明，假脱机打印子系统的性能是优越的。在没有数据打印输出时，系统同原来完全一样地工作。当有数据打印时，系统效率可以大大提高。例如，发一个“COPY ABC PRN”命令，即表示要把文件 ABC 的内容在打印机上打印出来。若没有假脱机打印子系统，在发了这个命令后，要等到把文件内容全部打完后才会出现操作系统的提示符，而在这段时间中无法利用系统作任何别的工作。在使用了假脱机打印子系统后，在发了这个命令后，会马上出现操作系统的提示符，从而用户立即可以利用系统干其它事(尽管文件并未打印出来)，在用户干其它事的同时，打印机把文件的内容打印出来，形成两件工作并行处理，大大减少了用户等待打印机打印的时间。

第七章　汉字处理数学模型的建立

第一节　基　本　定　义

一、汉字集与编码集

定义 1.1：汉字集

若干个汉字组成之集合称为汉字集。

$$H=\{h_i \mid i=1,2,\cdots,n\}$$

其中，H 为汉字集，h_i 为汉字，n 为正整数，汉字集大小 $|H|=n$。我们经常以汉字基本集(GB2312)为研究对象，若用 H_0 表示汉字基本集，则 $|H_0|=6763$。

定义 1.2：码符集

编码组成符之集合称为码符集。

$$C=\{c_i \mid i=1,2,\cdots,l\}$$

其中，C 为码符集，c_i 为码符，l 为正整数，码符集大小为 $|C|=l$。

定义 1.3：编码

按一定规则组成的码符序列称为编码。

$$e=c_1c_2\cdots c_m$$

$$c_j\in C \qquad (j=1,2,\cdots,m)$$

其中，e 为编码，c_j 为码符，若 m 为常数，则码长 $|e|$ 为常数，这种编码称为码长为 m 的等长码。若 m 不为常数，则码长 $|e|$ 不为常数，这种编码称为不等长码。

定义 1.4：编码集

若干个编码组成之集合称为编码集。

$$E=\{e_i \mid i=1,2,\cdots,r\}$$

$$e_i=c_{i1}c_{i2}\cdots c_{im},\quad c_{ij}\in C\qquad (j=1,2,\cdots,m)$$

其中，E 为编码集，e_i 为编码，r 为正整数。编码集 E 的大小 $|E|=r$，r 取决于 m 个码符的排列规则，故有：

$$r<P_m$$

二、映射

定义 1.5：一对一映射

设 f 是集合 A 到 B 的一个映射，若由 $f(a_i)=f(b_i)$，可推出 $a_i=b_j(a_i\in A,b_j\in B)$，则称 f 为 A 到 B 的一对一映射。

定义 1.6：满射

设 f 是集合 A 到 B 的一个映射，若 $f(A)=B$，则称 f 为 A 到 B 的满射。

定义 1.7：——对应映射

设 f 是集合 A 到 B 的一个映射，若 f 既是 A 到 B 的一对一映射，又是 A 到 B 的满射，则称 f 为 A 到 B 的一一对应映射。

第二节　汉字输入处理的数学模型

一、汉字属性序列

定义 2.1：汉字属性集

若干个汉字属性元素的集合称为汉字属性集。

$$A=\{a_i\} \mid i=1,2,\cdots,k\}$$

其中，A 为汉字属性集，a_i 为属性元素，k 为正整数。

定义 2.2：汉字属性序列

按一定规则组成的属性元素序列称为汉字属性序列。

$$s=a_1a_2\cdots a_t,\quad t\leqslant k$$

$$a_j\in A\qquad (j=1,2,\cdots,t)$$

其中，s 为汉字属性序列，a_j 为属性元素。

定义 2.3：汉字属性序列集

若干个汉字属性序列组成之集合称为汉字属性序列集。

$$S=\{s_i\mid i=1,2,\cdots,q\}$$

$$s_i=a_{i1}a_{i2}\cdots a_{it},\quad a_{ij}\in A\qquad (j=1,2,\cdots,t)$$

其中，S 为汉字属性序列集，s_i 为汉字属性序列，q 为正整数。

汉字的属性有多种，例如音素、形素、笔画、部首、声母和韵母等，在不同的场合可以选用不同的属性。一个汉字属性序列可以描述出一个汉字，而且它们之间是一一对应的，故有

$$s_i=f_0(h_i)$$

其中，f_0 是汉字集 H 到汉字属性序列集 S 的一一对应映射。

二、汉字输入过程

目前，汉字输入方法有三种：语音识别输入、字形识别输入、汉字编码输入（亦称键盘输入）。这三种汉字输入方法都有一个共同之处，它们都致力于抽取汉字的属性信息，并由此完成相应汉字的输入。其相异之处在于，它们采用的汉字属性集不同。

我们可以把汉字输入过程归结为：

$$d_i=f(s_i),\quad s_i\in S$$

其中，d_i 为汉字内码，它属于汉字内码集 D，我们将在下一节中给出汉字内码和汉字内码集的定义。这儿要指出，di 与 hi 是一一对应的，汉字的输入意味着其对应的汉字内码的输入。映射 f 即由汉字的属性序列转换成该汉字内码的过程，这种转换的结果应该是唯一的，所以 f 应该是 S 到 D 的双射。

在汉字的语音识别输入和字形识别输入中，f 由计算机和识

别装置共同完成。在汉字键盘输入中，f 则由用户(人)和计算机合作完成。下面将对汉字键盘输入作进一步讨论。

三、汉字的键盘输入处理

根据定义 1.2、定义 1.3 和定义 1.4，可得到输入码符集 U、输入码 v、输入码集 V 的定义：

$$U=\{u_i \mid i=1,2,\cdots,l\}$$

$$v=u_1u_2\cdots u_m,\quad u_j\in U\qquad (j=1,2,\cdots,l)$$

$$V=\{v_i \mid i=1,2,\cdots,r\},\quad v_i=u_{i1}u_{i2}\cdots u_{im}$$

为了描述汉字的键盘输入处理，可以把映射 f 看作是 f_3 和 f_4 的复合，故有：

$$d_i=f(s_i)=f_4\cdot f_3(s_i)$$

其中，$d_i\in D$，$s_i\in S$

我们还可以把上式进一步分解为：

$$v_i=f_3(s_i)$$

$$d_i=f_4(v_i)$$

其中，$v_i\in V$，$s_i\in S$，$d_i\in D$

定义 2.4：编码方案

设 H 为汉字集，V 为输入码集，对于每一个 $h_i\in H$，存在某一个 $v_i\in V$，有映射 f，满足：

$$v_i=f(h_i)$$

则称 f 为汉字集 H 的编码方案。

根据前面所述内容可知，汉字集 H、汉字属性序列集 S、输入码集 V 之间有以下关系：

$$s_i=f_0(h_i)$$

$$v_i=f_3(s_i)$$

故有：$v_i=f_3\cdot f_0(h_i)$

因此，映射 $f_3\cdot f_0$ 为汉字集的编码方案。映射 $f_3\cdot f_0$ 的实现即为由汉字得出其输入码的过程，这一过程由用户完成。它可以分成两个

步骤，首先根据汉字获得其属性序列，这就是映射 f_0 的实现；然后，用户再根据编码规定，由属性序列得出其对应的汉字输入码，即：

$$a_{i1}a_{i2}\cdots a_{it}=f_0(h_i)$$

$$v_i=f_3(a_{i1}a_{i2}\cdots a_{it})$$

其中，　$h_i\in H$，　$v_i\in V, a_{ij}\in A$　　$(j=1,2,\cdots,t)$

由此可知，汉字输入码的编码方案必须面向用户，使用户能较方便和迅速地完成映射 f_0 和 f_3。为了使编码规则较简单，通常有：

$$|V|<|D|$$

也就是说，汉字输入码与汉字不是一一对应关系，而是一对多的关系，即一个汉字输入码往往会对应于多个汉字，我们称此为重码现象。注意，有少数汉字输入码与汉字具有一一对应关系，我们称它为无重码类输入码。

定义 2.5：重码汉字集 D_r

$$D_r=\{f(s_i)\mid s_i\in S\},\ |D_r|>1$$

下面我们来分析映射 f_4。映射 f_4 由计算机实现，它根据汉字输入码 v_i 确定相应的汉字内码 d_i。根据上面所述情况可知，v_i 与 d_i 间不是一一对应的，故这一过程也要分成两个步骤。我们可以把映射 f_4 看作是映射 f_1 和 f_2 的复合，即：

$$f_4=f_1\cdot f_2$$

对映射 f_2 的描述如下：

$$D_r=f_2(v_i)$$

$$D_r=\{d_r,d_{r+1},\cdots,d_{r+u}\}$$

其中，　$D_r\in D, v_i\subset V$

以上过程充分体现了一个汉字输入码会对应于多个汉字内码这一现象。

对映射 f_1 的描述如下：

$$d_i=f_1(D_r)$$

$$D_r=\{d_r,d_{r+1},\cdots,d_{r+u}\}$$

其中，　$d_i \in D, D_r \in D$

这体现了在映射 f_2 得出的重码汉字集 D_r 中确定所需汉字的过程。目前常利用组词规则、句法规定和语法限制等，甚至采用人工选择来实现之。

总结以上内容，我们可以用下式表示汉字的键盘输入处理

$$d_i = f_1(D_r) = f_1 \cdot f_2(v_i) = f_1 \cdot f_2 \cdot f_3(s_i)$$

其中　$d_i \in D, D_r \subset D, v_i \in V, s_i \in S$

我们还可进一步推出下式

$$d_i = f_1 \cdot f_2 \cdot f_3 \cdot f_0(h_i)$$

另外，对于少数没有重码的汉字输入码（如区位码、电报码等），其 $|D_r| = 1$，则可以不再进行映射 f_1 所表示的过程。此时，我们可以把 f_1 视为一个恒等映射 I，故此时有

$$d_i = f_1 \cdot f_2 \cdot f_3 \cdot f_0(h_i) = I \cdot f_2 \cdot f_3 \cdot f_0(h_i) = f_2 \cdot f_3 \cdot f_0(h_i)$$

四、词输入处理

以词为单位的汉字输入过程，实际上是把词输入码转换成词内码的过程。我们首先来定义词输入码集和词内码集。

定义 2.6：词输入码集

设汉字输入码集为 V，则词输入码集为 V 的 m 重笛卡尔积，即 $V \times V \times \cdots V$，故有：

$$V^m = \{x_{im} = v_{i1}v_{i2}\cdots v_{im} \mid v_{ij} \in V, j = 1, 2, \cdots, m\}$$

定义 2.7：词内码集

设汉字内码集为 D，则词内码集为 D 的 n 重笛卡尔积，即 $D \times D \times \cdots \times D$，故有：

$$D^n = \{y_{in} = d_{i1}d_{i2}\cdots d_{im} \mid d_{ij} \in D, j = 1, 2, \cdots, n\}$$

因此，以词为单位的汉字输入，可以反映为以下数学关系

$$y_{in} = F(x_{im}), \quad y_{in} \in D^n, x_{im} \in V^m$$

在一般情况下，m 不一定等于 n，这两者之间的关系跟词输入码的编码法则有关。映射 F 是一个复合映射，这是因为对于任何

一个词输入码 x_{im}(即 $v_{i1}v_{i2}\cdots v_{im}$),按照编码方案,完全有可能出现同码词,即同码词集内的元素个数大于 1,故必须再从同码词集内选择所需输入之词 y_{in},(即 $d_{i1}d_{i2}\cdots d_{in}$),所以有:

$$F=F_1\cdot F_2$$

$$y_{in}=F(x_{im})=F_1\cdot F_2(x_{im})$$

$$x_{im}\in V^m, y_{in}\in D^n$$

我们把上式作为词输入处理的数学模型。其中 $F_2(x_{im})$表示由词输入码获得同码词集,映射 F_1 表示同码词集中的选择处理。

如果同码词集的元素个数等于 1,则映射 F_1 为恒等映射 I,这时数学模型为:

$$y_{in}=F_1\cdot F_2(x_{im})=I\cdot F_2(x_{im})=F_2(x_{im})$$

第三节　汉字输出处理的数学模型

一、国标码和汉字内码

根据定义 1.2、定义 1.3 和定义 1.4,可得到国标码符集 B、国标码 g 和国标码集 G 的定义:

$$B=\{b_i \mid 21H\leqslant b_i\leqslant 7EH, i=1,2,\cdots,94\}$$

$$g=b_1,b_2 \qquad b_1,b_2\in B$$

$$G=\{g_i \mid i=1,2,\cdots,n\}, \quad g_i=b_{i1}b_{i2}$$

汉字国标码是 GB2312 中给汉字确定的编码,它是一种国家标准。显然,汉字集 H 到国标码集 G 的映射 φ_0 是双射,故有:

$$g_i=\varphi_0(h_i)$$

并且:　$G=\varphi_0(H)$

所以,一定存在 G 到 H 的双射 φ_0^{-1},且有:

$$h_i=\varphi_0^{-1}(g_i)$$

$$H=\varphi_0^{-1}(G)$$

根据定义 1.2、定义 1.3 和定义 1.4,可得到内码符集 A、内码

d 和内码集 D 的定义：

$$A=\{a_i|i=1,2,\cdots,t\},t<n$$

$$d=a_1a_2\cdots a_m,a_j\in A \quad (j=1,2,\cdots,m),m<t$$

$$D=\{d_i|i=1,2,\cdots,n\},d_i=a_{i1}a_{i2}\cdots a_{im}$$

内码与汉字是一一对应的，故汉字集 H 到内码集 D 的映射 ψ_0 是双射。故有：

$$d_i=\psi_0(h_i)$$

又因为，$h_i=\varphi_0^{-1}(g_i)$

所以，$d_i=\psi_0\cdot\varphi_0^{-1}(g_i)$

因此，内码与国标码之间存在着一种对应关系。目前国内的汉字内码设计中，多寻求与国标码有一种较简单的对应关系，即要充分注意到 G 到 D 的映射 $\psi_0\cdot\varphi_0^{-1}$。

二、汉字字素码和汉字字像

根据定义 1.2、定义 1.3 和定义 1.4，可得出字素码符集 K、字素码 y 和字素码集 Y 的定义：

$$K=\{k_i|i=1,2,\cdots,t\}$$

$$y=k_1k_2\cdots k_m,\quad k_j\in K \quad (j=1,2,\cdots,m),m<t$$

$$Y=\{y_i|i=1,2,\cdots,n\},\ y_i=k_{i1}k_{i2}\cdots k_{im}$$

汉字字素码是表示汉字字形或字音的编码。

定义 3.1：汉字字像、汉字字像集

汉字在外部输出设备上的映像称为汉字字像。若干个汉字字像之集合称为汉字字像集。

$$Z=\{z_i|i=1,2,\cdots,c\}$$

其中，z_i 为汉字字像，Z 为汉字字像集。一般来说，一个汉字 h_i 可以对应于多个字像 $z_{ij}(j=1,2,\cdots,p)$。输出设备不同，其汉字字像集亦不同。

三、汉字输出过程

汉字的输出过程就是由汉字内码转换成相应汉字字像之过

程，故可归结为：

$$z_{ij}=\psi(d_i)$$

$$z_{ij}\in Z, d_i\in D$$

为了进一步研究汉字输出过程，我们可以把映射 ψ 看作是映射 ψ_4 和 ψ_3 的复合，故：

$$z_{ij}=\psi(d_i)=\psi_4\cdot\psi_3(d_i)$$

$$z_{ij}\in Z, d_i\in D$$

上式还可以进一步分解为：

$$y_i=\psi_3(d_i)$$

$$z_{ij}=\psi_4(y_i)$$

$$d_i\in D, z_{ij}\in Z, y_i\in Y$$

映射 ψ_3 是 D 到 Y 的双射，由计算机实现之，即系统内的访问汉字库算法。映射 ψ_4 由计算机与输出设备共同实现。现把映射 ψ_4 看作是映射 ψ_1 和 ψ_2 的复合，故有：

$$z_{ij}=\psi_4(y_i)=\psi_1\cdot\psi_2(y_i)$$

映射 ψ_2 由计算机实现，即系统内的字形变换算法和字音变换算法。如果系统所支持的字形或字音越多，则 $|Z|$ 就越大，映射 ψ_2 的实现就越复杂。另外，映射 ψ_2 的复杂度与所采用的字形或字音变换方法有关。

映射 ψ_1 由输出设备实现，即汉字的成像过程（形成汉字字像），这一过程完全决定于输出设备的工作机制。

总结以上讨论之内容，我们可以把汉字输出过程用下式表示：

$$z_{ij}=\psi_1\cdot\psi_2(y_i)=\psi_1\cdot\psi_2\cdot\psi_3(d_i)$$

$$z_{ij}\in Z, y_i\in Y, d_i\in D$$

上式中的映射 ψ_1、ψ_2 和 ψ_3 反映了汉字输出的三个过程。

汉字输入与输出是汉字信息处理的最基本操作，对汉字输入和输出处理数学模型的研究和建立，有益于讨论汉字输入处理模块和汉字输出处理模块的结构与性能，并且有助于进一步研究这两个模块的规范化和标准化问题，显然这是十分必要的。

第四节　联想输入处理的数学模型

一、总述

汉字的联想输入处理是以汉字或词为基本单位，按照汉语中的组词规律或语法规则，联想出用户欲输入的下一个字或词的可能范围，然后给出提示，供用户选择，再根据用户选中之字或词继续进行联想。我们把这个过程归结为："输入汉字或词——联想——选择汉字或词——联想——选择汉字或词——……"。

汉字的联想输入处理可以分为两种基本类型，即字联想输入处理和词联想输入处理。

二、字联想输入处理

设 V 为汉字输入码集，D 为汉字内码集，根据汉字输入处理数学模型可知，汉字输入码 v_i 与汉字内码 d_i 之间存在以下关系：

$$d_i = f_1 \cdot f_2(v_i)$$

$$d_i \in D, v_i \in V$$

其中 d_i 即表示所要输入的汉字。

设 f_3 为 d_i 的联想映射，则 $f_3(d_i)$ 为汉字 d_i 的下一个最大可能的字集，也就是说，汉字 d_i 之后欲输入之汉字包含在字集 $f_3(d_i)$ 中。

设映射 f_4 为对字集 $f_3(d_i)$ 的选择处理，则 $f_4 \cdot f_3(d_i)$ 为联想选择后得到的汉字。我们令该汉字内码为 $d_i^{(1)}$，则有：

$$d_i^{(1)} = f_4 \cdot f_3(d_i)$$

根据字联想输入处理可知，在获得 $d_i^{(1)}$ 后，还可继续联想下去，故又设 f_5 为 $d_i^{(1)}$ 的联想映射，则 $f_5(d_i^{(1)})$ 为汉字 $d_i^{(1)}$ 的下一个最大可能的字集。若令映射 f_6 为字集 $f_5(d_i^{(1)})$ 的选择处理，设选中汉字的内码为 $d_i^{(2)}$，则有：

$$d_i^{(2)}=f_6\cdot f_5(d_i^{(1)})$$

依次类推，可得出一般表示式：

$$d_i^{(k)}=f_{2k+2}f_{2k+1}(d_i^{(k-1)})$$

其中，映射 f_{2k+1} 为 $d_i^{(k-1)}$ 的联想映射，映射 f_{2k+2} 为对字集 $f_{2k+1}(d_i^{(k-1)})$ 的选择处理，$d_i^{(k)}$ 为当前联想处理中被选中输入之汉字内码。

这样，我们得到的 $d_id_i^{(1)}d_i^{(2)}\cdots d_i^{(k)}$ 就是通过输入单字 d_i 后，经 k 次联想及选择处理后获得的一串所要输入的汉字。为了更清楚地表达这一过程，下面用一个实例说明之。

设当前已经输入了一个“大”字，则在系统内得到该字的内码 d_i，即有 d_i=“大”。根据上面给出的表达式可知：

$$f_{(3)}(d_i)=\{学,家,会,\cdots,使\}$$

设通过映射 f_4 的选择处理，选中了“学”字，则有：

$$d_i^{(1)}=f_4\cdot f_3(d_i)=“学”$$

再由“学”字进一步进行联想，于是有：

$$f_5(d_i^{(1)})=\{习,校,生,\cdots,院\}$$

设通过映射 f_6 的选择处理，选中了“生”字，则有：

$$d_i^{(2)}=f_6\cdot f_5(d_i^{(1)})=“生”$$

这样，当输入“大”字之后，经过两次联想及选择处理后，得到的一串所要输入的汉字为：

$$d_id_i^{(1)}d_i^{(2)}=“大学生”$$

因此，我们把下式作为字联想输入处理的数学模型。

$$d_i^{(k)}=f_{2k+2}\cdot f_{2k+1}(d_i^{(k-1)})$$

三、词联想输入处理

设 V^m 为词输入码集，D^n 为词内码集，根据词输入处理数学模型可知，词输入码集 V^m 与词内码集之间存在以下关系：

$$y_{in}=F_1\cdot F_2(x_{im})$$

$$y_{in}\in D^n,x_{im}\in V^m$$

设 y_{in}为当前输入的词，映射 F_3 为 y_{in}的联想映射，则 $F_3(y_{in})$为词 y_{in}的下一个最大可能的词集，也就是说，词 y_{in}后欲输入之词包含在词集 $F_3(y_{in})$中。

设映射 F_4 为对词集 $F_3(y_{in})$的选择处理，则 $F_4 \cdot F_3(y_{in})$为联想选择后得到的词，我们令该词的内码为 $y_{ip}^{(1)}$，则有：

$$y_{ip}^{(1)}=F_4 \cdot F_3(y_{in})$$

其中，p 不一定等于 n，例如，二字词的后继词可能是三字词，亦可能是二字词。我们为了叙述方便，将 p 也写作 n，以后不再区分说明。所以，上式可表示为：

$$y_{in}^{(1)}=F_4 \cdot F_3(y_{in})$$

根据词联想输入处理可知，在获得 $y_{in}^{(1)}$后，仍可继续进行联想，故又设 F_5 为 $y_{in}^{(1)}$的联想映射，则 $F_5(y_{in}^{(1)})$为 $y_{in}^{(1)}$的下一个最大可能的词集。若令 F_6 为词集 $F_5(y_{in}^{(1)})$的选择处理，则 $F_6 \cdot F_5(y_{in}^{(1)})$为本次联想所选中之词，我们设此词的内码为 $y_{in}^{(2)}$，故有：

$$y_{in}^{(2)}=F_6 \cdot F_5(y_{in}^{(1)})$$

依次类推，则可得出一般表示式：

$$y_{in}^{(k)}=F_{2k+2} \cdot F_{2k+1}(y_{in}^{(k-1)})$$

其中，F_{2k+1}为 $y_{in}^{(k-1)}$的联想映射，映射 F_{2k+2}为对词集 $F_{2k+1}(y_{in}^{(k-1)})$的选择处理。

我们将上式作为词联想输入处理的数学模型。$y_{in}y_{in}^{(1)}y_{in}^{(2)}\cdots y_{in}^{(k)}$为通过 k 次联想和选择处理后得到的一串所要输入的汉字。为了更清楚地说明这一过程，下面用一个实例说明之。

设当前已经输入了一个词，此词为“汉字”，则在系统内得到该词的内码 y_{in}，即有 y_{in}=“汉字”。根据上面给出的表达式可知：

$$F_3(y_{in})=\{\text{系统，内码，信息，…，字体}\}$$

设通过映射 F_4 的选择处理，选中了“信息”这一个词，则有：

$$y_{in}^{(1)}=F_4 \cdot F_3(y_{in})=\text{“信息”}$$

再由“信息”这个词作进一步联想，故有：

$$F_5(y_{in}^{(1)})=\{\text{工程，系统，处理，…，世界}\}$$

设通过映射 F_6 的选择处理，选中了“处理”这个词，则有：

$$y_{in}^{(2)}=F_6 \cdot F_5(y_{in}^{(1)})=\text{“处理”}$$

再由“处理”这个词作进一步联想，故有：

$$F_7(y_{in}^{(2)})=\{\text{方法，技术，方式，…，意见}\}$$

设通过映射 F_8 的选择处理，选中了“技术”这个词，则有：

$$y_{in}^{(3)}=F_8 \cdot F_7(y_{in}^{(2)})$$

这样，当输入“汉字”这个词之后，经过三次联想及选择处理后，得到的一串所要输入的词为：

$$y_{in}y_{in}^{(1)}y_{in}^{(2)}y_{in}^{(3)}=\text{“汉字信息处理技术”}$$

第八章　汉字系统辅助技术

第一节　汉字造字程序设计

一、引言

目前国内不断推出新的汉字操作系统，其功能也越来越强。一个汉字系统往往就需要配备好几种字库，以适应不同场合的需要。例如，16×16 点阵汉字采用横向点阵字模适合于屏幕显示，其它高点阵字库，如 24×24、24×48、32×32、48×48 等多采用纵向点阵字模以适合于打印机输出。汉字库中含有国家标准一、二级汉字，共计 6763 个汉字。用户如需用到国际以外的汉字，就需要系统提供造字软件，能够快速生成新汉字字模并加入到系统中，但是现有的一般造字软件多是用描点来生成字模，其效率低，生成字模质量不高，而且一种造字程序只能造出一种点阵的字模，在汉字库多样化的情况下，这种造字程序的功能就显得不足，故有必要开发新的造字程序。

二、总述

汉字字形点阵动态生成程序所追求的最主要的目标就是快速生成汉字字形点阵，尽可能减少用户进行枯燥繁琐的描点操作，并尽可能地利用已有字形点阵来生成新的点阵，而且还要能够适用于各种点阵字模相互转化，还设有比例尺功能，用户可以通过改变比例尺来观察生成点阵的效果。

为了使用户操作简便清晰，屏幕画面采用窗口形式设计(如图8-1所示)，提示行为汉字提示，放大字模区为实际操作对象，在其上进行描点等操作，小字实形区跟踪大字变化，即时显示点阵效果，部件区显示已存入部件库的部件，多级帮助或数据录入提示都是通过开窗口显示或接收，开出的窗口在低一级窗口的斜上方以弹出式出现，使用后恢复原来屏幕，这样整个屏幕比较美观和活泼，帮助提示详细及时，用户无须记忆功能键。

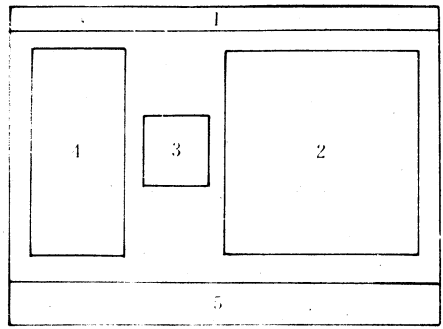


图8-1 屏幕格式

1. 程序标题 2. 字模点阵框 3. 小字实形
4. 部件区 5. 提示行

各种显示器的扫描线数和色彩都不尽相同，为充分发挥各种显示器的潜力，程序对显示器的类型进行自动识别，各窗口的位置和色彩值都是通过计算获得，程序能够适用于CGA、EGA、VGA、HGC等显示适配器。程序采用效率较高的TURBO C编程实现。

三、设计思想

要求程序可同时对三个字库进行操作，它们为：

①16×16横向点阵字库，隐含名为CCLIB；

②24×24纵向点阵字库，隐含名为CLIB24；

③用户扩展字库，它的点阵形式，存贮形式和组织形式都由用户定义，点阵 x×y(x,y∈[1,96])的任意组合，存贮形式分为横向或纵向，组织形式分为标准或非标准。

标准形式即字模按照标准字库中汉字的顺序组织，非标准形式即对字库另建索引库来组织，这种方式作为临时存放字模用。标准字库的操作区域为区位码 0101 至 9494，允许对字库进行扩充。下面从几方面来介绍程序的构思。

1. 功能选取

操作功能的选取是围绕着工具的使用效率进行的，主要分为描点操作类，特殊操作类和部件操作类，下面分别介绍它们：

(1)描点操作类　八个方向光标移动键，每按一下，光标移动一点，并且根据光标是处于插点，删点或移动状态，产生不同效果。如是插入态就写一点；如是删除态就清除一点；如是移动态则不改变。插入、删除、移动三态是由 INS 和 DEL 键控制的，如按 INS 立即进入插入态，如按 DEL 立即进入删除态，如当前为 INS 态则再按 INS 就变为移动态，DEL 也同样。另有快速移动键：CTRL－HOME、CTRL－END、CTRL－PGUP、CTRL－PGDN 分别使光标直接移动到点阵四角。

此外定义功能键：

F2：将当前点阵字模取反。

F3：写当前光标所在行。

F4：写当前光标所在列。

F5：将当前光标所在行取反。

F6：将当前光标所在列取反。

F8：清除点阵。

F1：读取已有点阵信息。

此操作从当前读方向字库读取已有的汉字点阵，若用户第一次使用该功能则系统向用户提示从三个字库中的某个库读取，而且在以后操作中默认从上次指定的库中读取(读方向可由 F9 更

改)。若从标准形式字库读取,则提示用户输入区位码,若从非标准形式字库读取,则提示输入输入码,通过索引库进行查找。

F10:保存现已生成的字模到库文件中(操作同 F1)。

F9:指定或修改读写方向。

(2)特殊操作类　这类操作中有以下选择:S 比例尺,E 缩放,V 保存部件,ESC 退出。

①S 比例尺:↑变大,↓变小,N 输入新比例尺。

'↑'或'↓'则将点阵逐级变大或变小供用户观察,N 提示输入新比例尺值,如超出显示能力范围则报警并放弃操作。

②E 缩放:提示输入新点阵,x,y 方向的点数,点阵框变成新点阵形式,如当前点阵框内有字模存在则该字模也随着转化为新点阵字模,利用这一功能可以方便地用已有字库生成新字库。

③V 保存部件将当前字模存入 16×16 或 24×24 点阵的部件库中。

(3)部件操作类　如当前是 16×16 或 24×24 点阵则会激活 PART 窗,出现光标,可通过移动光标,翻页查找部件,按回车即选中,选中部件被映射到字模框内,部件可在字模框内叠加,组合成完整汉字。

2. 汉字提示行

程序中的汉字提示选取在西文方式下实现,这是由于目前汉字操作系统的版本较多,各种版本对西文软件的支持情况不同,我们曾专门编了一段程序进行试验,得出的结论是不同版本的系统存在较大的差异,因而要使程序可在不同的汉字系统下都能正常工作是有很大困难的,而且它的通用性势必要受到一定限制。对一个工具而言,通用性是衡量其质量的一个相当重要的指标,所以最终是以程序内部的方法实现汉字提示的显示。

3. 视频缓冲区的利用

程序中多处用到了视频缓冲区来简化程序的设计,例如:把横向点阵转化为纵向点阵,如果按以往的做法,则需对点阵信息进行

变换处理，这需要较多的计算，而现在只需要将横向点阵字模在显示区显示，然后再按纵向把字横读出即完成了转换工作，尤其在本程序中汉字点阵数目可变化范围很大的情况下更显得优越。

4. **函数总述**

```
main( )/ * 程序的主函数 * /
initfile( )/ * 初始化程序将要处理的文件，允许用户改变系统隐含的文件名 * /
makewin(int * xs,int * ys,int * xe,int * ye)
/ * 用于初始化的开窗　函数 * /
disword(int　x,int y,int mp[])
/ * 用程序内部方法显示汉字的函数 * /
dismsg(int ms[],int x,int y,int n)
/ * 显示一条汉字信息的函数 * /
intialize()　/ * 图形系统初始化，获取图形状态参数 * /
cacularpara(int first)
/ * 动态计算参数 * /
cacularpartpara(int first)
/ * 动态计算部件窗的参数 * /
mainwindow()
/ * 显示主屏幕窗口 * /
mpart()
/ * 生成部件窗口 * /
dispartword(char mp[],int x,int y,int pn)
/ * 显示一个部件字 * /
drawborder
/ * 画框函数 * /
clear(int l,int t,int r,int b)
/ * 清除一片区域 * /
outmsg(char　* strl,char　* str2)
```

```
/*输出信息*/
warning(char *string)
/*发出警告信息*/
pullwin()
/*在上级窗口上开窗*/
bkwin()
/*关闭当前窗,恢复上级窗*/
gerkey()
/*读取键盘扫描码*/
verify(char *str)
/*操作确认*/
getwordqw(FILE *fp,int q,int w,int dir)
/*按区位方式获取汉字字模*/
getwordext(FILE *fp,long pos)
/*按扩展方式获取汉字字模*/
hdisplayword()
/*按横向点阵方式显示汉字*/
vdisplayword()      /*按纵向点阵方式显示汉字*/
hdispoint(int x,int y,int r,int b)
/*横向显示时的点定位*/
vdispoint(int x,int y,int r,int b))
/*纵向显示时的点定位*/
changepoint(int x,int y,int a)
/*改变某点的显示状态*/
getint(int xs,int xe,int *x)
/*图形方式下带回显录入整型数*/
getstr(int c,char * str)
/*图形方式下带回显录×字符串 */
disinput(char *str)
```

```
/*显示字符串*/
hsawetom()
/*按横向从视频缓冲区读字模到内存*/
vsavetom()
/*按纵向从视频缓冲区读字模到内存*/
f1work()
/*从字库中读取字模并显示*/
f10work()
/*把当前字模保存到汉字库中*/
partwork()
/*部件操作*/
partcursor()
/*显示部件光标*/
specialwork()
/*特殊操作*/
changescale()
/*改变字模显示的比例尺*/
enlarge(int newx,int newy)
/*字模点阵变换*/
savepart()
/*保存部件/*
putcursor(int attr)
/*从显示字模*/
script()
/*程序主要工作控制函数*/
crdir()
/*改变读取方向*/
cwdir()
/*改变写方向*/
```

四、实现方法

本程序采用 Turbo C 语言编程实现。编程规模为 C 语言源程1500 行左右。下面分几个要点讨论程序的具体实现。

1. 图形方式下开窗口和关窗口

程序中的汉字提示信息是以窗口形式出现的，由两个函数完成。

pullwin()完成开窗口工作，为产生动态效果，开窗的位置选在当前窗口的右斜上方，覆盖住当前窗的大部分，只留出部分边缘，使画面有层次感，用户对目前的操作比较清晰。bkwin()完成关闭当前窗口恢复上一级窗口。

在 pullwin()中用到的主要数据结构有：

```
int layer()
/* 用于记录当前窗口的层号 */
struct{
    int x0,y0,x1,y1;   /* 窗口的对角坐标 */
    void far  *p   /* 保存上级窗口屏幕的储存指针 */
}windat[6]
```

pullwin()完成工作有：

```
layer t =1;
windat[layer]. x0=windat[layer-1]. x0+30;
windat[layer]. y0=windat[layer-1]. y0-3;
windat[layer]. x1=windat[layer-1]. x1-5;
windat[layer]. y1=windat[layer-1]. y1-3;
getimage(windat[layer]. x0,windat[layer-1]. y0,
windat [layer]. x1,windat [layer-1]. y1,windat [layer]. p);
{在当前层窗上显示信息}
```

bkwin()完成工作有：

```
putimage(windat[layer]. x0,windat[layer-1]. y0,
```

```
        windat[layer].p,COPY-PUT);
    layer-=layer;
```

2. 部件功能的处理

部件功能使用户能够预先定义一些常用的汉字偏旁部首，存放在部件库中，造字时就可以从中取出组合或改造为新汉字。部件操作在屏幕的固定区域内，根据显示器的显示能力和当前点阵情况，动态地计算出能够显示的部件数目和精确位置。部件库只有16和24点阵两种，其他点阵可由用户通过点阵转化获得。

partwork()中用到的数据结构为：

```
struct partfile{
    char filename[13]
    /*记录部件库名的串*/
    int status,
    /*初始化结果 0=失败,1=成功*/
    num,
        /*该部件库的部件数*/
        cx,cy,
        /*激活部件窗时,部件光标的移动位置*/
        mx,my,
        /*光标最大活动范围*/
        pos,
        /*已显示部件在部件库中的位置*/
        point;
        /*部件的点阵形式*/
    void *pc;
    /*部件窗光标指针*/
    }part[1]
    /*结构构变量*/
```

用户在主菜单下按'P'键激活part窗，提示行提示：

“部件操作:光标键移动光标,页键翻页,回车选中”

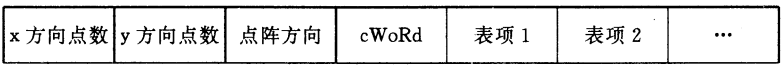
当用户选中某一部件时,就把该部件的点阵字模直接映射到字模内,完成部件操作。

3. 扩展字库管理

当用户在定义字库时要求定义扩展字库时,要请用户输入扩展字库名和组织形式(字模按区位码顺序排列或按扩展形式排列),对于按标准形式储存的字库处理很简单,只要检查该文件是否存在,如不存在则创建一新文件,而后输入点阵的 X 方向和 Y 方向的点数及纵向点阵还是横向点阵。

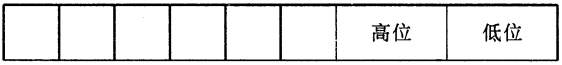
对于扩展形式储存的字库,则在检查该文件的同时还要检查索引库是否存在,检索字库中的字模是通过索引文件进行的,索引文件的格式是:

索引表头信息　　索引表项



2 字节　2 字节　1 字节　5 字节标志

索引项格式



6 字节输入码　　4 字节字模地址偏移

在创建索引文件时置入表头信息,打开索引时进行校验,防止非法文件。

4. 图形方式下的数据录入

由于程序是在图形方式下工作,不能利用标准输入输出函数进行数据输出和显示,因为标准输入输出函数会扰乱当前的图形屏幕,在行号超出当前屏幕最大行时出现滚屏,录入数据应将用户输入的字符回显,所以这里只能通过调用中断读取键盘,然后利用图形函数回显在屏幕上。int getint(int xs,int xe,int x)函数直接从键盘读入数字,然后转化为字符串在屏幕的适当位置显示,函数中的参数 xs 和 xe 为 x 的取值范围,输入数字时可使用退格键。

int getstr(int c,char *str)函数直接从键盘读入 ASC Ⅱ 字符，组成字符串由 str 返回，函数中的参数 c 为所求字符串的最大长度，输入过程按 Esc 键中断，并且返回给调用者失败信息，当输入超出最大长度时或输入非法字符时报警，并且拒绝接收，按回车键表示输入有效，返回成功信息。

5. 比例尺功能

Changescale()函数改变字模的比例尺，提供两种改变比例尺的方法：一是使用光标键逐级变化；二是用户直接输入新的比例尺值。

工作步骤如下：

①通过以上两种形式获得新比例尺的值；

②对新比例尺值进行可行性检查，使其取值最小为 1，最大不超过当前屏幕的显示能力；

③如果可以进行操作就执行以下工作，否则返回；

④如果当前字模框内有字模存在就把该字模保存到内存中；

⑤清除字模框，按照比例尺重新画出字模框；

⑥如果内存中保存有字模，就显示该字模；

⑦退出。

6. 点阵转化

设已有字库 oldx×oldy 点阵，要生成 newx×newy 点阵字模。

定义变量如下：

```
int minx=MIN(oldx,newx)
int miny=MIN(old y,newy)
float newdx=(float)newx/minx
float newdy=(float)newy/miny
float olddx=(float)oldx/minx
float olddy=(float)oldy/ miny
```

根据 minx 和 miny 将原点阵和所求点阵都划分为 minx×miny 个小区域，且新点阵中每一区域都对应于原点阵中一区域，

然后按下列算法进行变换：

```
{
int i,j
for (i=0,i<min y;++i)
for (j=0,j<min x;++j)
   {
   int light=0,m,n;
   for (m=0;m<olddy;++m)
   for (n=0;n<olddx;++n)
{
        如原点阵第(j,i)个小区域内相对应坐标(n,m)点为
        亮,则++light
     }
     if (light>0&&light>=(int)(old dx * old dy/2)
     {
     /* 点亮新点阵第(j,i)个区域内所有点 */
     for (m=0;m<new y,++m)
     for (n=0;n<new x,++n)
          {
          点亮(j,i)区内相对坐标(n,m)的点
              }
       }
}
```

下面是使用该算法进行变换的一个例子：

图 8-2 是变换前的字模点阵，图 8-3 是经变换后的字模点阵。变换中各变量的值如下：

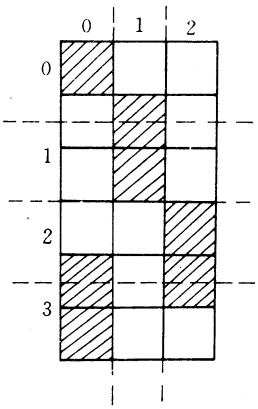


图 8－2　原字模点阵

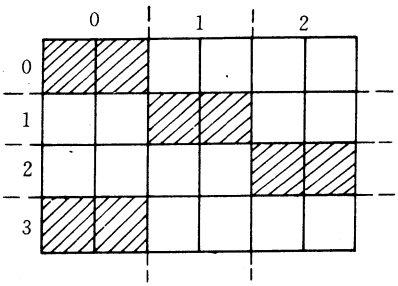


图 8－3　变换后字模点阵

oldx＝3，　oldy＝6，　newx＝6　　newy＝4，
minx＝3，　miny＝6，　olddx＝1　　olddy＝1.5，
newdx＝2，　newdy＝1。

经变换后的点阵字模可直接存盘或经简单加工修改后再存盘，在变换中如果从高点阵变换到低点阵，有可能会使一些点的信息丢失，产生失真，这是由两种点阵的分辨率不同而造成的；如果是从低点阵变换到高点阵则不会产生失真现象，原点阵中每点信息都能准确反映到新点阵中，只是在点阵相差很大时，新点阵的边缘呈“锯齿”形，不很美观，需要修改。

由于在 Turbo C 系统下汉字与图形共存问题还没完全解决，我们采用一种变通的方法，即在程序中建立一个内部字库，提供提示信息所需汉字，通过写点来显示汉字，实现西文状态下汉字显示，因为信息窗较小，故汉字的显示、窗口的建立和清除都很迅速，可与西文媲美。

7. 从字库中读取字模

f1work()函数完成从字库中读取字模，并在字模框内显示出

来。读取字模允许用户选择从哪个字库读取，此处允许三个读方向：L(16 点阵标准字库)，H(24 点阵标准字库)，E(用户扩展字库)。当第一次操作时，由于读方向变量 rdir 初值为 'N'，未指向任何字库，所以要请用户指定读方向，而在以后的操作中都隐含对上次指定的读方向操作(可通过 F9 改变读写方向)。

读取字模时按标准字库或非标准字库分别进行。

由于标准字库是按照区位码的顺序组织字模的，所以读取也是根据区位码进行的，用户输入所要求的汉字区位码，然后对区位码进行合法性检索(区号的区值为 01～09，16～94，位号为 01～94)，对于合法者才允许读取。读取通过 getwordqw(FILE *fp,int q,int w,int dir)函数进行，fp 为字模存储缓冲区指针，然后调用 displayword()函数显示字模。

对于非标准字库，要求用户输入输入码，根据扩展字库索引查找某汉字字模在字库中的偏移位置，然后调用函数 getwordest(FILE *fp,long pos)读取字模，再显示。

8. 保存字模到字库中

f10work()函数完成把造好的汉字字模存放到字库中。存字模也同 f1work()一样允许有三个存储方向，当前方向是由变量 wdir 记录。

向字库存储字模是读取的逆操作。向标准字库存储，是先通过函数 savetom()把字模读入内存，然后写入文件。向扩展字库存储是要求用户为该字定义一个长度最大为 6 的输入码，把输入码和该字在字库文件中的偏移都写到索引文件的一个索引项中，然后再把字模写入字模库。

在标准字库保存字模时，用户输入的区位码有可能超出文件长度，而这个区位码又是合法的，这时要对文件进行扩充，计算出该汉字应在文件中的偏移，减去文件已有的长度，得出一个差值 DL，然后向文件尾写 DL 个 0 字节，否则字模只是接在原文件后，并不会写到要求的偏移处。

9. 汉字提示行的实现

由于提示行信息较少，汉字的重复率也不大，为了方便，故内部字库并不是按单字组织的，而是以一条信息为单位组织的。

字库以嵌入文件的形式连接入程序，在读取汉字字模时是使用本软件的英文版本把字模存入扩展文件，然后用自己编写的一段小程序进行格式转换。

内部字库的定义形式是：

```
/ * 第一条信息 * /
msl1[][16]={{汉字个数,0,0,0,0,…}
      {第 1 个汉字的字模 16 个字}
        …
      {第 N 个汉字的字模 16 个字},};
/ * 第二条信息 * /
msl2[][16]={{汉字个数,0,0,0,…},
  …
```

dismsg(int ms[],int x,int y,int n)在(x,y)处显示一条提示信息。

disword(int x,int y,int mp[])在(x,y)处显示一个汉字。

第二节　字模压缩和还原技术

一、引言

对于通用型汉字系统，大多采用点阵式汉字库，由于计算机内的信息存贮是以字节为单位，所以每个汉字的 m×n 点阵字模所需的存贮容量为(m×n)/8 个字节，这样，汉字库的容量就等于字库中的汉字个数和每个汉字点阵所占用的字节数之乘积。然而，由于汉字字数多、字体多、字号多，因此，汉字字形库的存贮容量是十分惊人的。例如，含有 7000 个 128×128 点阵字模的汉字库，其存

贮容量为 14.34M 字节。所以，如何有效地压缩汉字库的存贮容量，是当前的一个热门课题。

字模压缩是将汉字字形的冗余数据信息进行压缩，实际上就是将每个汉字 m×n/8 个字节的信息容量通过压缩变换，使得压缩后的每个汉字所需的信息容量小于 m×n/8 个字节。还原技术就是指将压缩后的汉字字模信息，经压缩的逆变换，把它还原成原来的汉字字模。一般来说，字模压缩越甚，还原时间越长，字模压缩和存贮空间的节省是以系统成本的提高和还原速度的降低为代价的，为此评价一个汉字字模压缩方法的性能指标，主要有下列三个方面：

①字形质量：还原后的汉字字形对原来的汉字字形失真情况。

②压缩比：压缩前字库占用的存贮空间同压缩后字库所占用的存贮空间之比。

③还原速度：还原单位汉字字形信息所需的时间。

除了上面介绍的三个主要方面之外，还有几个方面不可忽视，如压缩码制作的人工干预程度，还原时软硬件的复杂程度，以及系统性能价格比等等。总之，只有当一种压缩方法能使得各项性能指标均达到优良，才能满足用户的需要。

字模压缩的方法较多，本节主要介绍线性增量压缩法、哈夫曼树压缩法、笔画压缩法、字根压缩法等。

二、线性增量压缩法

线性增量压缩法又称黑白段压缩法，它是在英国蒙纳公司激光汉字照排系统上改进设计而来的。主要是利用数字化仪逐行扫描字稿，依次记录每行所通过的空白点数（白段）和非空白点数（黑段）的要求，分别进行水平方向和垂直方向的压缩处理。

1. 水平压缩

根据汉字平均纵向笔画少于平均横向笔画的特性可知，每行中白段与黑段的数目有限，于是，只需要记录每行中的白段与黑段

的长度，就可以减少字模点阵每行所占用的空间，达到信息压缩的目的，所以水平压缩每行的信息可用如下的压缩格式来记录：

〔行标志〕〔白段〕〔黑段〕…〔白段〕〔黑段〕

*　　W_1　B_1　　W_n　　B_n

实验表明，对于 1380×1380 的汉字字形而言，水平压缩的压缩比略低于 2 倍，为了提高压缩效率，我们可以从汉字信息的纵向考虑。

2. 垂直压缩

在汉字字形点阵中，由于相邻两行或多行的信息往往是雷同的，因此就可以把信息相同的相邻行合并起来，减少纵向信息行的数目，对此只需要在水平压缩的行信息压缩格式中增加一栏〔重复行数〕就可以达到压缩的目的，关于垂直压缩的行信息压缩格式可以如下表示：

〔行标志〕〔重复行数〕〔白段〕〔黑段〕…〔白段〕〔黑段〕

*　　N　　W_1　　B_1　　W_n　　B_n

按照这种方法，以重复行数 N 来压缩冗余的信息，对 1380×1380 的汉字字形点阵而言，压缩比可达到 13 倍左右，我们所说的蒙纳方法就是将水平压缩和垂直压缩揉合在一起的压缩方法，对上述点阵的压缩比为 23.94 倍。

3. 线性增量压缩法

在汉字字形中，绝大部分汉字中都存在斜笔画的撇、捺等，这是用蒙纳方法较难压缩的，为了提高压缩效率，可以在垂直压缩的行信息压缩格式中，每行的白段或黑段后面都增加一栏〔白段或黑段增量〕，以处理汉字字形的斜笔画，其行信息压缩格式为：

〔行标志〕〔重复行数〕〔白段〕〔白段增量〕…〔黑段〕〔黑段增量〕

*　　N　　W_1　　ΔW_1　　B_n　　ΔB_1

…〔白段〕〔白段增量〕〔黑段〕〔黑段增量〕

W_n　　ΔW_n　　B_n　　ΔB_n

用这种格式可连续处理 N 条信息行，压缩效率比垂直压缩法

提高了 16 倍，目前这种方法已应用在各种字体的照相排版系统中。

4. **特点**

通过以上的分析讨论，我们把线性增量压缩法归纳为如下几个特点。

(1)还原速度快　还原汉字字形时，可依据行信息的“行标志”，“重复行数”，“白段或黑段”的长度，“白段或黑段的增量”，逐行恢复输出，直到一个汉字的所有行还原为止。

(2)对于高分辨率的汉字压缩效率高　例如，对 96p 的汉字一般采用 1380×1380 点阵来描述，用线性增量压缩法可将此点阵信息从 238,050 字节，平均压缩到 1,223.6 字节，压缩效率已高达 200.7 倍，这比蒙纳方法提高效率 8.44 倍。

但是，这种压缩方法有可能改变原来的字样，这是因为当重复行数 N 取得过大时，黑白段的增量和实际的增量间存在一定的差异，导致了字形“走”样。此外，对于低分辨率的汉字，用这种方法压缩效率比较低，故不宜使用。线性增量压缩法主要用于精密型汉字处理，它特别适合于高分辨率汉字的情况。

三、哈夫曼压缩法

1. **基本原理**

(1)最短的带权路径长度　在数据结构中，给出了二叉树路径长度的概念，如果将路径长度概念推广到一般的情况，考虑带权的结点。

给定数列$\{w_1, w_2, \cdots, w_n\}$，以此 n 个数构成森林 F，假定初始状态时，每棵树 T_i 的根结点为 w_i，其左、右子树均为空。如果将 $F=\{T_n, T_{n-1}, \cdots, T_2, T_1\}$从大到小排列，取出根节点最小的二棵树 T_1 及 T_2 组成一棵树 T，并使其根结点为 w_1+w_2，根据 T 的大小插入 F 中，反复此过程，直到 F＝T，只有一棵树为止。举例如下：

设有一组数列{2,10,5,6}，其哈夫曼树的构造过程如图 8-4

所示。

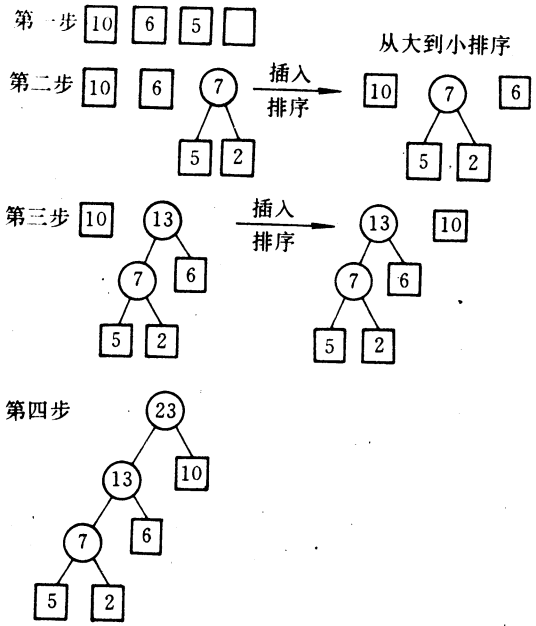


图 8-4　哈夫曼树的构造示意图

上面这棵树称作哈夫曼树，其带权路径长度（WPL——Weight Path Long）为最短（可用数学归纳法证明）。

$$WPL=10+6\times2+2\times3+5\times3=43$$

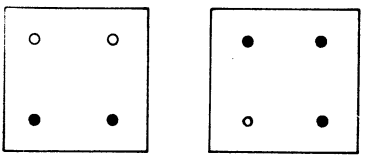
(2)汉字点阵图形的冗余度　一个汉字点阵字模可以看作由多个构造一致的 m×n 子点阵组成，这些子点阵为 2×2、1×4、1×3、2×1 等等。

不同构造的子点阵，它所表达的状态数 n 与该子点阵中的数 d 满足下面关系式：

$$n=2^d$$

由于汉字大多使用横或竖的笔画，而且横画多于竖画，因而相邻的子点阵左右之间的关系最密切，各相邻的子点阵之间具有一

定的统计规律。例如:汉字的一个子点阵的状态如图 8-5(a)所示,则其右邻子点阵的状态如图 8-5(a)的条件概率,远远大于如图 8-5(b)的条件概率。于是将汉字采用全信息点阵表示,其冗余度是很大的。



(a)　　(b)

图 8-5　子点阵状态图

因此,我们可以利用汉字点阵的冗余度和哈夫曼树具有最小带权路径的长度的这些特性,将汉字点阵字模信息进行压缩。

(3)哈夫曼树的构成以及压缩码的获得　对于某一给定的子点阵,相邻子点阵的状态的条件概率 n 个数作为叶子,依上述哈夫曼算法构成哈夫曼树,并取每一级的左路径代码为 0,右路径代码为 1,这样就可以从根出发取它到叶子路径上的 0 或 1 为该叶子的编码,因此各叶的编码不一定等长。

以 2×2 子点阵为例,由(1)知,它共有 16 种不同的状态,每种状态以 $P_0, P_1, \cdots, P_{15}$ 表示,如图 8-6 所示。

在汉字点阵字模中,各相邻子点阵之间具有一定的统计规律,

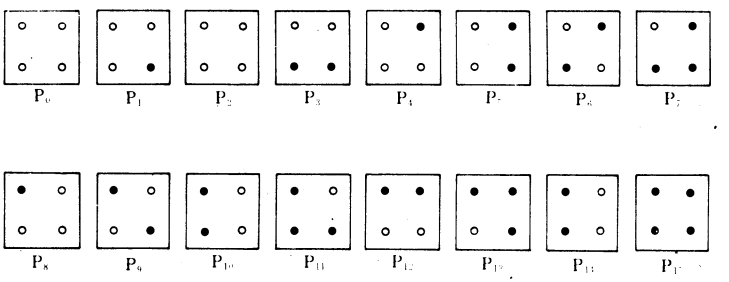


图 8-6 2×2 子点阵的 16 种状态

在给定左邻子点阵状态的前提下，比如给定的子点阵状诚为 P_0，其右邻子点阵的 16 种不同的状态出现的概率是不同的，通过对大量汉字点阵图形中的各种相邻子点阵关系进行统计，这 16 种不同的状态产生的条件概率分别为 $w_0, w_1, w_2, \cdots, w_{15}$，其值如表 8-1 所示。

表 8-1　16 种状态出现的条件概率值

状态	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
条件概率	W_0	W_1	W_2	W_3	W_4	W_5	W_6	W_7
条件概率值	0.438	0.050	0.011	0.041	0.032	0.132	0.032	0.014
状态	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
条件概率	w_8	w_9	w_{10}	w_{11}	w_{12}	w_{13}	w_{14}	w_{15}
条件概率值	0.011	0.024	0.148	0.023	0.025	0.002	0.015	0.001

以 $w_1, w_2, \cdots, w_n$ 作为叶子构成哈夫曼树，并取其每一级的左路径代码为 0，右路径代码为 1，于是，就得到一棵哈夫曼树，如图 8-7 所示。

从树根到每一树叶，可得到对应该树叶状态的不等长的编码，如表 8-2 所示。

表 8-2　代码分配表

状态	p_0	p_1	p_2	p_3	p_4	p_5	p_6	p_7
条件概率值	0.438	0.050	0.011	0.041	0.032	0.132	0.032	0.014
代码	1	00000	01111000	00010	01100	010	00011	011011
状态	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}	p_{14}	p_{15}
条件概率值	0.011	0.024	0.148	0.023	0.025	0.002	0.015	0.001
代码	011101	000010	001	000011	01111	01110010	011010	01110011

从表 8-2 可见，发生概率大的状态，其代码长度短，发生概率小的状态其代码长度长，因而平均代码长度就缩短了。

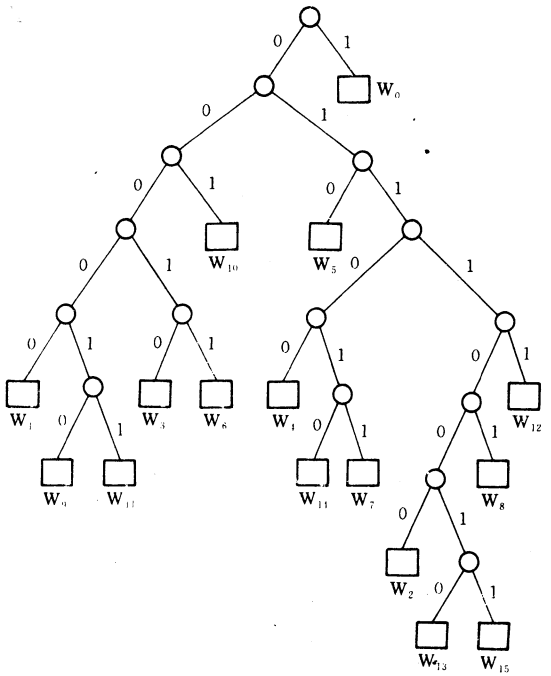


图 8-7 哈夫曼树

平均代码长度＝带权路径长度

$$=0.438\times1+0.050\times5+0.011\times7+0.41\times5$$
$$+0.032\times5+0.132\times3+0.032\times5+0.014\times6$$
$$+0.011\times6+0.024\times6+0.148\times3+0.023\times6$$
$$+0.025\times5+0.002\times8+0.015\times6+0.001\times8$$
$$=2.8$$

变换之前子点阵代码长为 4。

根据①式，给定的子点阵状态有 16 种，因而构成的哈夫曼树共有 16 棵，图 8-7 仅是其中之一。

2. 主要算法

(1)哈夫曼树的构造　设已建立 16×16 点阵的基本字库，用

2×2 子点阵将每个汉字的点阵分割为 64 个子点阵，如图 8-8 所示。

对基本字库中的每个汉字，从左到右，以每个子点阵的左邻状态为给定的条件；统计它的条件状态的条件概率，通常选用 P_0 为初始条件；对每个汉字从左上方 S_1 开始扫描；每扫描一个子点阵都以其左邻子点阵的状态为条件，统计该子点阵各个不同状态下

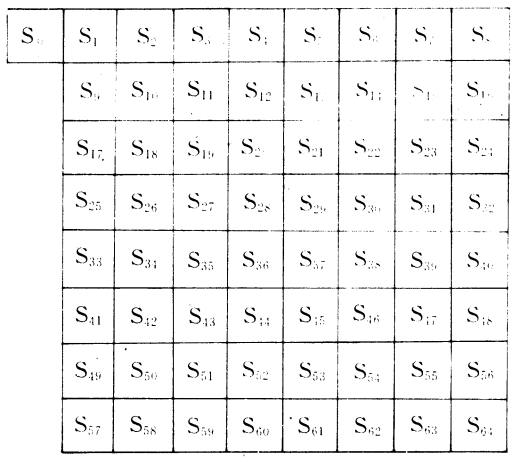


图 8-8 每个汉字的分割图

出现的条件概率，除了 S_1 外，每栏最左侧的子点阵的左邻子点阵是指上一栏的最右侧的子点阵，此扫描过程直到 S_{64} 子点阵时为止，这样就完成了一个汉字子点阵状态出现的条件概率统计。

对于 16 种给定的状态，每种状态下产生的条件概率状态也是 16 种，对应着 16 棵哈夫曼树和每棵树上的 16 片叶子，这样 16 棵树共有 256 片叶子，对应 256 个频次统计单元，占 512 个字节。我们可以在内存中开辟 512 个字节作为频次统计单元的存放区，如果将原始字库中的所有汉字都扫描一遍，就可得到一张 256 频次统计单元的频次统计表(如图 8-9 所示)，它们是构成 16 棵哈夫曼树的叶子的值。

根据频次统计表中的值，按照前面介绍的方法，不难得到 16

棵树的内存分配图(如图 8-10 所示)。

(2)汉字哈夫曼压缩码的获取　对基本字库中的每个汉字按图 8-8 的分割,从左上角开始扫描每个子点阵,每扫描一个子点阵均以其左邻的子点阵状态为初始条件,这样可以找到相应哈夫曼树的根,遍历哈夫曼树,找到该扫描到的子点阵状态所对应的叶子,从根起按左路径代码为 0,右路径代码为 1,直到该叶子为止,就可得到相应子点阵状态的代码。如果从 S_1 扫描到 S_{64} 为止,就可得到一个二进位串,也就是说得到一个汉字的哈夫曼压缩码。如果我们扫描整个基本字库就得到压缩后的压缩字库。

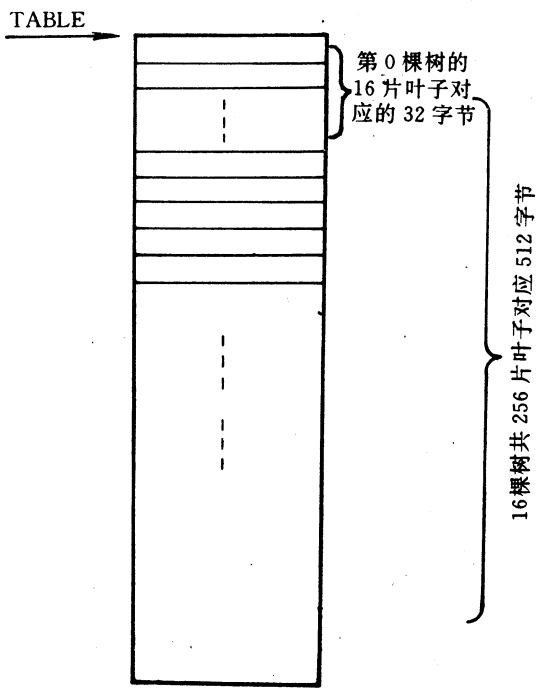


图 8—9　频次统计表

(3)汉字哈夫曼压缩码的还原　首先,我们仍以 P_0 为初始状

态，将压缩码逐位读出，每读出一位，按哈夫曼树的指针来判断是否是叶子，当它是叶子时，则可根据指针中所隐含的值可找到此二进位串相应的子点阵状态 P_i，然后，根据状态 P_i，重复上面的过程，直到 64 个子点阵全部还原，装配成一个汉字为止。

(4)特点　显然，用哈夫曼方法压缩汉字，对字体的质量几乎没有影响，而且还原速度能满足输出的要求，但是其压缩效率不很高，一般在 50%左右。

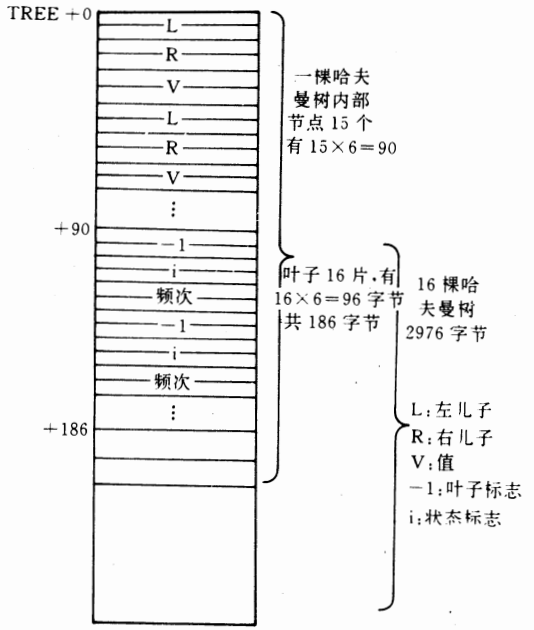


图 8－10　哈夫曼树内存分配图

图 8－10

四、笔画压缩法

1. 基本原理

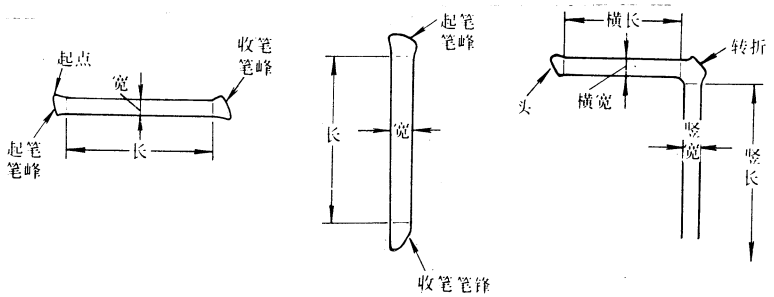
对于高精度的印刷用字，每个字都是由结构和形状不同的笔画组成，同一笔画在不同的部位其形状也不相同，同时在起笔、收笔处有明显的笔锋，因此，为了使输出的汉字字体质量高，必须详尽地描绘出字体的各个特征。

每个汉字是由二种笔画组成，即规则笔画与非规则笔画组成。规则笔画分为横画、竖画、折画等，它们是由直线段、起笔、收笔和转折等笔锋组成，对这些笔画可用轮廓折线来表示外形。非规则笔画分为点画、撇画、捺画等，对它们可用一连串折线画近轮廓曲线。于是，我们可以利用笔画的信息以及笔画的结构，画出笔画的形状和大小，最终形成输出的汉字字形信息，这就是笔画压缩的基本思想。

2. 压缩方法

(1)规则笔画的压缩表示　由于不同字体的汉字的规则笔画的形状是不一样的，为此，我们以宋体为例，宋体的横、竖、折的形状如图 8-11 所示。

图 8-11(a)表示宋体字中横画的形状。据统计，横画的收笔笔锋共有七种，因此可用三位二进制信息表示，起笔笔锋仅有一种，



(a)　(b)　(c)

图 8—11　宋体横、竖、折、形状

故只需用一位表示，另外横画的起始坐标(x,y)、横画的长度、宽度可用 2～3 个字节表示。于是，整个宋体横画只需 3～4 个字节便可详细描述。

图 8-11(b)表示宋体竖画的形状。一般只需用 3～4 个字节来表示竖画的起始坐标(x,y)、竖画的长度、宽度、起笔笔锋与收笔笔锋以及左右两侧的倾斜度。

图 8-11(c)表示宋体折画的形状，只需用 5 个字节来表示整个笔画的形状，它们的分配是：

标记:2 位	起点 x:7 位	起点 y:7 位
竖长:7 位	竖宽:2 位	头　:1 位
横长:7 位	横宽:1 位	尾　:2 位
竖的倾斜度:1 位		

经统计，规则笔画占汉字笔画总数的一半以上，规则笔画的信息表示法对提高压缩倍数起了重要作用，也对字体质量有较大影响。

(2)非规则笔画的信息压缩表示　非规则笔画的轮廓可看作是任意曲线，一般可用一连串的折线来逼近其轮廓曲线，它对字体质量影响不大。

如图 8-12 所示，不规则笔画“点”用一串向量 AB、BC、CD、DE、EF、FG、GH、HI、IJ、JA 来表示，笔画的起点坐标为(x,y)，每个向量用带符号的 Δx、Δy 表示，并规定：

第一象限：$\Delta x \geqslant 0, \Delta y < 0$；

第二象限：$\Delta x < 0, \Delta y < 0$；

第三象限：$\Delta x < 0, \Delta y \geqslant 0$；

第四象限：$\Delta x \geqslant 0, \Delta y \geqslant 0$。

因此，上述一串向量的数字表示为：

+3,+3,+2,+5;+1,+5;+3,+2;+1,−4;+3,−3;+0,−3;−3,−3;−7,−4

其中最末一个向量 JA 为封口，不必在压缩信息中列出，可自

动产生。若把向量长度分成三种情况并在控制字节中标注出来：小于 8、小于 16 及大于等于 16，则上述向量的 Δx 和 Δy 的绝对值小于 8，不规则笔画“点”只需 12 个字节表示：9 个字节表示 9 个向量的增量，2 个字节表示起点坐标，1 个字节作为控制字节。

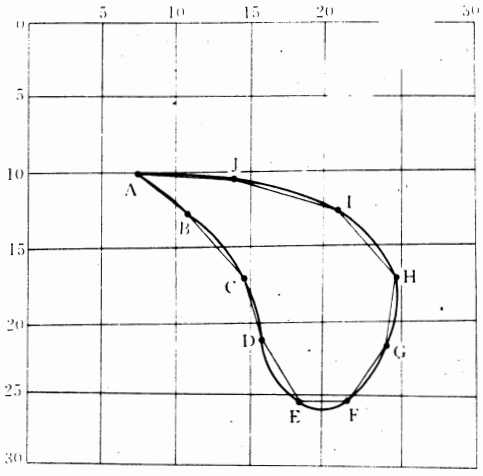


图 8-12 非规则笔画“点”的形状

3. 还原方法

根据压缩信息转换成最终输出的字形点阵，其步骤如下：

①对规则笔画和非规则笔画信息进行解释，按照规定的方式，转换成标准形式。

②把标准形式的向量转换成与之逼近的阶梯点，在经过每个阶梯点上写上两位标记，分为四种情况：00 为空点、01 为黑段的始点、10 为黑段的终点，11 为孤立黑点。

③把标记点阵转换成最终输出的点阵字形。

4. 特点

用这种方法压缩汉字对字体的质量影响很小，对高点阵的汉字压缩效率比较高，但是，为了提高信息的还原速度，需要付出较

多的硬件代价。

五、字根压缩法

1. 基本原理

字根是汉字中具有独立表意或表音能力的最小构字单位。经过对大量的查频数据统计结分析，我们知道，尽管汉字数目很多，但字根的数目却是有限的，因此我们可以把全部的汉字字根组成一个字根库，在压缩时，只需要记录汉字的结构和字根的代码，在还原时，从字根库中取出字根，按汉字的结构，叠加成一个汉字。

2. 压缩方法

下面以 GB2312 为例，对汉字的结构及字根的选择作一些简单的讨论。

GB2312 中有 6763 个汉字，仅使用 602 个字根，汉字数与字根数的关系如表 8-3 所示。

表 8-3 的统计结果表明，每增加 500 个汉字所使用字根的增长幅度越来越小，而且汉字数和字根数之间不是线性关系，汉字中字根的分布是不均匀的。通过对《中华大字典》近四万字的调查发现，“口”字根用到的次数最高，共 9831 次，其次分别是“一，十，八，木，田”等字根，其中有些字根很少使用，前 100 个字根的累计使用

表 8-3　汉字数－字根数对照表

汉字数	500	1000	1500	2000	2500	3000	3500
字根数	296	385	443	492	518	547	569
汉字数	4000	4500	5000	5500	6000	6500	6763
字根数	580	586	588	594	598	600	602

频度超过了 70%，根据字根的使用频度不同，我们可以构造哈夫曼树，对每个字根赋予一个哈夫曼代码。

另外，考虑到汉字是由字根按一定的结构组成的，根据对 GB2312 中 6763 个汉字结构的研究表明，共有 376 种不同的拓扑

结构，各种拓扑结构的使用频度相差很大，同样可以用构造哈夫曼树的方法，对每种拓扑结构进行 哈夫曼编码。这样，越常用的结构代码越短，越不常用的结构代码越长。因此，字根压缩汉字的压缩格式可记录为：

〔汉字结构〕〔字根〕…〔字根〕

3. 还原技术

对于某一输出的汉字，先读出字体结构的信息代码，然后读出各字根的信息代码，根据汉字的结构，对各字根进行适当的放大、缩小、平移处理，按各部位叠加，组成一个字体美观的汉字。

4. 特点

常用的汉字一般由 2～3 个字根组成，使用三字根以内的汉字占汉字总数的 95%以上，因此在还原时，还原的速度还是比较快的。无疑用这种压缩方法的压缩效率比较高，但字体质量方面可能有些欠缺。不过，如果在字根中对同一字根存放几种不同形状的信息，那么组成汉字时可依照汉字结构自动选取所需要使用的字根，这样组成的汉字字体质量就必然要好得多。

第三节　汉字编码辅助技术

一、引言

我国使用的文字——汉字是象形文字，不象拼音文字那样可以直接通过通用的西文小键盘输入计算机，因而计算机的汉字输入是我国普及计算机应用的一大障碍。为了克服这个难关，国内外许多计算机工作者、文字工作者在进行汉字编码的研究。目前编码大多是手工进行的，工作异常艰巨且效率低，错误率高，尤其是对编码进行统计分析的工作量更是巨大，所以手工法已经不能适应编码工作的需要了，我们开发的《汉字编码辅助技术》正是为了使编码工作变得轻松、愉快一些，希望它能对编码工作者有所裨益。

《汉字编码辅助技术》是采用 FoxBASE 数据库管理系统编程实现的，它能够完成编码工作中的一些基本任务，如录入、查询、修改、统计、分类和简单的自动编码以及生成最终码本等。

这些功能能够帮助编码工作者实现编码计算机化，提高编码工作效率，并可以方便地对已形成的编码方案进行统计分析，得出技术参数，为进一步优化提供依据。本软件的设计目标是：用户界面友好，操作简便，无需专门训练即可掌握使用，各种命令以菜单方式出现，及时给出相应的帮助信息。

二、设计思想

编码辅助技术的目的是为了帮助编码人员完成数据管理，减轻他们的工作负担。实质是一个数据管理系统，但它和通常意义上的数据管理系统是有很大差异的。用数据库管理系统来完成这一工作比较方便、安全和可靠，因而采用速度较快的 FoxBASE 来编程实现，对数据的操作也可直接利用或变相利用 FoxBASE 中的命令完成。

编码辅助技术软件的响应速度是衡量工具质量的一个重要指标，为了尽可能地加快响应速度，在构造数据库结构的时候，加入了一些冗余数据项来换取时间上的效率，这样做的依据是因为本辅助软件对磁盘空间的要求不很大，况且现在硬件技术发展很快，磁盘空间的要求比较容易得到满足，加入的冗余数据约为原数据量的 10%～30%，但在进行统计分析时却可缩短一倍以上的时间，权衡利弊，我们认为这样作是值得的。

在辅助软件的功能选取上，我们也作了适当的取舍，在保留了编码工具所必需的录入、查询、修改、统计分析、生成最终码本等功能的基础上，又增加了定义编码方式，查询编码方式，删除编码方式及初步的自动编码功能，以帮助用户更方便有效地管理文件。在录入词条和输入编码时，可根据已有的定义进行检查，如发现不符合定义的编码，则拒绝录入，从而增强了编码的可靠性。提供的统

计分析功能可做多项工作，提供给用户的信息包括：编码的总体情况及每一位输入码的码元分布情况，自动编码可以根据预定义的单字输入码对照表生成多字词的输入码对照表。

用户界面采用菜单式，并有相应的提示信息，以帮助用户操作，所以一般用户都能够比较容易掌握。功能模块如图 8-13 所示。

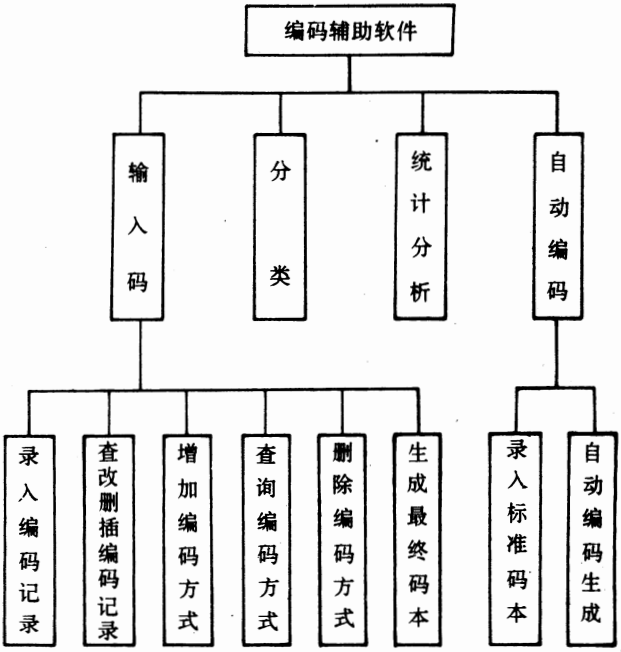


图 8—13 编码辅助技术系统模块示意图

三、实现方法

1. 数据库的构造

本辅助软件所涉及的数据结构不很复杂，但如何实现程序的高效率运行却是一个比较难处理的问题。此处所要处理的数据并

不象一般数据管理系统所面对的数据那样错综复杂，对用户来说有用的数据仅两种：词条和输入码，为了完成以上功能我们定义了四种文件：

standall. dbf，stands. dbf，standd. dbf，standmb. dbf

它们之间的相互关系如图 8-14 所示。

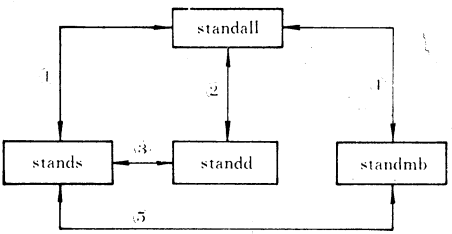


图 8-14 数据库之间的关系图

其中：

①②一般操作均需查询

③录入时检查合法性

④⑤自动编码时访问

下面具体介绍上述文件的结构。

standall. dbf

字段名	类型	长度	含义
CNAME	C	12	编码名称
CMAXLEN	N	3	输入码最大长度
CMINLEN	N	3	输入码最小长度
CEUQLEN	C	1	输入码是否为等长码
CDEFNAME	C	13	定义域库名
CSRMNAME	C	13	输入码库名
CCTLEN	N	3	词条最大长度

stands. dbf

字段名	类型	长度	含义

TJNUM	N	6	为统计而设的计数器
HANZI	C	X	词条
SURUMA	C	Y	输入码
S1	C	1	输入码第一位
…		…	
SY	C	1	输入码第 Y 位

(其中 X,Y 可变)

standd. dbf

字段名	类型	长度	含义
ALLOWC	C	1	是否允许为字母
ALLOWN	C	1	是否允许为数字
ALLOWO	C	1	是否允许为其他它 SCII 字符
CHRSET	C	26	所允许的字母集合
NUMSET	C	10	所允许的数字集合
OTHERSET	C	20	所允许的其它 ASCII 字符集合

standmb. dbf

字段名	类型	长度	含义
KBZMB	C	13	标准码本名

standall 是总体管理编码名称的库,其中 CNAME 记录用户定义的某种编码的名称,其它字段是对该编码特征的一个简要描述,CMAXLEN 为输入码的最大长度,CMINLEN 为输入码的最小长度,CEUQLEN 说明输入码是否等长,CCTLEN 为词条的最大长度,CDEFNAME 记录定义域库的名称,CSRMNAME 记录输入码对照表库的名称。

stands 是输入码对照表的标准库,其基本内容 HANZI 和 SURUMA 字段,SURUMA 的字段长度根据定义变化而调整,在该字库中含有两类冗余信息,TJNUM 是为统计分析而设立的计数

器，另一类是S1～SN(N为输入码长度)，它的每个字段为1字节，保留相应位上的输入码，这样在对编码进行逐位统计分析时，可使效率提高一倍。

Standd是定义域标准库，它是记录某编码输入码的定义情况，ALLOWC、ALLOWN和ALLOWO依次表明该位输入码的取值是否允许为字母、数字和其它ASCII字符的情况，ALLOWC和ALLOWN的取值为('A'、'Y'、'N')，ALLOWO取值为('Y'、'N')，'A'表示允许所有值，'Y'表示允许部分值，'N'表示不允许，当允许部分值时，由CHRSET、NUMSET、OTHERSET依次记录所允许的字母集合、数字集合、其它ASCII字符集合。

Standmb记录已有的标准码本的名称。

2. **定义编码方式功能**

为了使用户尽可能少地直接操作文件，以增加系统的安全性，所以用户产生新编码都必须通过此功能进行，用户为新编码命名，在以后的操作中用户只需知道该编码的名称即可，其它库文件均由系统产生和命名，对用户是透明的。

当用户输入编码名称时，程序首先在standall库中查找有无同名的编码存在，如有则报警，否则在standall中注册此编码并录入一些简要描述信息，包括输入码的最大和最小长度、是否为等长码、词条的最大长度，这些信息也保存在standall库中，然后根据上述数据创建两个库文件：定义域库和输入码库。定义域库的结构完全同standd库。输入码库的产生是：先以extened方式将stands的库结构拷贝至一临时库，然后对临时库加以修改，其中词条字段和输入码字段长度作相应调整，增加S2～SN字段，最后由"CREAT FROM <临时库>"方式建立库文件。

创建库文件结束后，录入编码定义信息，逐位对编码进行定义，定义内容如上所述。

3. 查询编码方式功能

此功能供用户了解已定义编码的信息，屏幕上显示出已定义编码总数及一屏编码名称，并且出现一光标，移动光标如超出当前页则显示上一页或下一页内容(假如存在的话)，当光标指向某种编码时打回车键，就可以得到进一步的详细信息，程序逐位显示该编码的每一位的定义情况。

4. 删除编码方式功能

该功能清除用户指定的编码，在 standall 库中将此编码注销掉，并且清除一切与此编码有关的文件。

用户定义、录入、以至生成最终码本后，定义域库和输入码库也就完成了它们的使命，没有存在的必要了，因而需要及时清理，由于库文件名对用户是透明的，所以有必要提供此功能，以保证安全。

5. 录入编码记录

用户输入编码名称后，程序就根据编码名称到 standall 库中查找看是否已经注册，如已注册过，则取出相应信息，再到定义域库取出有关的定义信息放到数组中(为提高速度，将定义信息放在内存中)，准备工作做好后，就打开输入码库录入数据，在把数据存入库以前，进行合法性检查，检查输入码为等长码、长度范围及各位取值情况。

6. 查询、修改、删除和插入功能

查询、修改、删除和插入功能在同一模块中实现，这样一体化处理的理由是，因为对记录进行修改、删除、插入也都必须先进行查询，通过查询才能进行记录定位。为了更灵活地实现以上四种操作，设立了两类选择，一类是功能选择，另一类是检索条件选择。功能选择包括：查询记录、修改记录、插入记录、退出。这样两类选择就可以对任意两个组合使用，进行 16 种不同操作，这样做不仅便于屏幕菜单的驱动，而且增加了灵活性。

按记录号进行操作时，屏幕上只显示一条记录，按其它检索进

行操作时，每屏上只显示十条符合条件的记录，选择退出便中止，若选择显示下一页便继续查找。

这里按输入码检索是这样进行的，用户输入一个输入码的匹配模式，该匹配模式可使用‘*’、‘?’和其它 ASCII 字符等构成通配符，然后对匹配模式进行分析，形成检索条件。

具体过程如下：

设 SMODE 为匹配模式

```
SMODE=TRIM(SMODE)
IF AT('*',SMODE)
  CONDI='LEN(TRIM(SURUMA))=LEN(SMODE)'
ELSE
  CONDI='.T.'
ENDIF
I=1
DO WHILE I≤MAXLEN. AND. I≤LEN(SMODE)
    IF SUBSTR(SMODE,I,1)='*'
        I=MAXLEN
    ELSE
        IF SUBSTR(SMODE,I,1)≠'?'
        CONDI=CONDI+'. AND. S'+IIF(I>9,STR(I,2),
        STR(I,1))+'='+CHR(39) + SUBSTR (SMODE,I,1) +
        CHR (39)
        ENDIF
        I=I+1
    ENDIF
ENDDO
LOCATE FOR & CONDI
DO WHILE ! EOF()
    CONTINUE
ENDDO
```

7. 最终码本的生成

用户完成输入和校对工作后就需要产生最终码本，码本的产生很简单，用户给出编码名称和最终码本的名称，通过：

“COPY　TO ＜文件＞ FIELDS　SURUMA，HANZI SDF”

即产生最终码本。

最终码本的格式是：每行一词条，行末是 0DH，0AH，文件尾是文件结束符。

每行内容是：

输入码　［最大长度，不足补空格］

空格　　［2 字节］

词条　　［最大长度，不足补空格］

8. 分类功能

分类功能是完成从一已有库中按照一定条件选取一部分构造一新库，该新库另外命名，它的名字也被加入 standall 库中，选取的条件是按输入码进行，其语法同查询中的输入码格式一样。

9. 统计分析功能

统计分析功能包括：显示编码定义的内容，现有的词条总数，已占用的编码单元数，编码单元总数，编码空间利用率，进一步操作选择有：重码统计和编码构成分析。

重码统计的内容有：显示重码编码单元数、最大重码数、重码率（重码单元数/已用编码单元数）。还可进一步分类统计并显示出重码，用户给出输入码的匹配模式，就可检索出所有重码的编码单元及重码次数。

编码构成分析主要是对编码按位进行统计，统计出每位所使用的码元情况，显示出码元和使用次数。前面谈到的数据冗余项主要用于统计分析，下面是具体的使用过程：

建立临时库文件 TJSRM1，TJSRM2

TJSRM1＝SYS(3)＋‘.DBF’

TJSRM2＝SYS(3)＋‘.DBF’

USE & SURUMANAME && 打开输入码库

此时的记录数即是词条总数

SORT TO & TJSRM1 ON SURUMA FIELDS TJNUM,HANZI,SURUMA

USE & TJSRM1

TOTAL TO & TJSM2 ON SURUMA

此时记录数即占用的编码单元数

且 TJNUM>1 的记录中的输入码都是有重码的

重码次数=TJNUM

在对重码进行统计分析时，只需对 &TJSRM2 库进行统计 TJNUM>1 的记录个数即得到重码编码单元数，选取 TJNUM 的最大值即最大重码数，在分类统计并显示重码时也需加上 TJNUM>1 的选择条件。

编码构成分析：

建立临时库

TJSRM3=SYS(3)+'.DBF'

TJSRM4=SYS(3)+'.DBF'

USE & SURUMANAME && 分析第 BITMAP 位输入码

SFIELD='S'+IIF(BITMAP>9,STR(BITMAP,2),STR(BITMAP,1))

SORT TO &TJSRM3 ON &SFIELD FIELDS TJNUM,&SFIELD

USE &TJSRM3

TOTAL TO &TJSRM4 ON &SFIELD

&TJSRM4 库中 SN 字段和 TJNUM 字段即分别为码元和对应的使用次数。

10. **自动编码**

自动编码的实现分为两个模块：一是录入标准码本，二是自动

生成。自动生成的原理是这样的，根据定义输入码的每一位取值，到标准码本库中查得相应的每一位的编码值，然后组合成词条的输入码。

(1)录入标准码本　此处标准码本为一汉字对应几项数据项。

例如：定义拼音码标准码本是这样的：

汉字，声母，韵母

这样在以后自动编码时就可定义：某二字词的输入码为词中第一字的声母加上第二字的韵母便构成该词条的输入码。

用户输入标准码本名称，然后在 standmb 库中查找是否已注册，如未注册过则问"新码本，建立吗？"如肯定则创建新码本，否则放弃操作，创建新码本需用户输入每个汉字所对应的数据项个数，然后依次输入数据项的名称，系统就会自动创建库文件，建成的库文件结构为：

字段名	类型	长度	含义
汉字	C	2	汉字
数据项1	C	1	编码信息
…	…	…	…
数据项 N	C	1	编码信息

其中汉字这一项以后并没有在程序中使用，汉字的查找都是通过计算进行的，在此设立它只是为显示或用户通过数据库系统编辑修改标准库提供方便而已。

库文件建立后就向库中添加 6763 个空白记录，每个记录对应国标中的一个汉字，顺序相当于按区位码不间断依次排列，库中的"汉字"字段由程序预先赋值，不需要用户输入，预赋值的办法是：

```
I=1
DO WHILE I<6763
    GO I
    HZQW=RECN0()
    HZW=MOD(HZQW,94)   && 计算汉字位号
```

```
        HZQ=INT(HZQW/94+16)  && 计算汉字区号
        IF HZW=0
            HZW=94
            HZQ=HZQ-1
        ENDIF
        HZ=CHR(HZQ+160)+CHR(HZW+160)  && 转化为汉字
        REPLACE &BZMB->汉字 WITH HZ
        I=I+1
    ENDDO
```

(2)自动生成　进行自动生成前,用户必须在编码功能中先定义编码,提供这种编码的简单描述信息。自动生成时,用户输入此编码的名称,程序查证编码名称的合法性。当该编码的输入码库中已有记录存在时,将发出警告:"原编码库已有记录存在",并且请求指示进一步操作:"请选择:A(追加新记录),O(覆盖原内容),Q(放弃操作)",然后用户输入词条文本的文件名。词条文本的格式是每行一词条(符合数据库的.TXT 文件格式),自动编码中的词条是从该词条文本中获得的。

以上工作做好后,就要进行生成规则定义。

提问内容为:"第[X]位输入码由词条中[]个汉字根据[](库名)标准码本的[](域名)值决定。"

在录入以上各项数据时,都要进行合法性检查。

自动生成是这样进行的,先把词条文本按 SDF 形式追加到输入码库中,然后根据以上定义进行编码,以对其中一位输入码编码为例说明:

①根据定义打开某标准码本库;

②根据定义读取词条中的某一汉字;

③由汉字得出此位输入码值。

由汉字得出输入码值是由计算代替查找进行的,标准码本的

结构在前面已经介绍过了，它的记录项是有规律排列的，因而此处定位查询汉字就可通过计算进行。

假设取词条 HANZI 中的第 X 个汉字，则：

HZQ＝ASC(SUBSTR(HANZI,2＊X－1,1))－160

HZW＝ASC(SUBSTR(HANZI,2＊X,1))－160

RECNUM＝(HZQ－16)＊94＋HZW

RECNUM 即是对应汉字的记录号，可直接用“GO 记录号”语句定位，这样比依次查找要迅速得多，在六千多条记录的库中进行查找则尤为如此。

按以上方法即可逐位编码，从而实现自动编码。

自动编码产生的库与通过输入功能产生的库文件一样可以进行其它任何操作。

第四节　保护方式编程技术

一、总述

80286 和 80386 分别为 Intel 公司的 16 位和 32 位微处理器，它们广泛用作 PC 系列机的 CPU。众所周知，DOS 只能在 80286 和 80386 的实方式下运行，由于 DOS 的关系，因此人们对实方式编程技术相当熟悉。然而，80286 和 80386 的保护方式具有比实方式更强的功能，因此我们有必要掌握保护方式编程技术。

目前，在 80286 和 80386 保护方式支持下工作的操作系统已相继推出，如 XENIX 和 OS/2 等。这些操作系统依靠保护方式向用户提供了强大的多任务处理和虚拟存贮功能，这均是实方式所望尘莫及的。所以，掌握保护方式的编程技术有利于充分发挥 80286 和 80386 的潜力，可以获得在实方式下无法得到的功能。掌握保护方式编程技术后，可以把已有的实方式程序改写为保护方式程序，或重新编写新的保护方式程序。一般来说，保护方式要比

实方式具有更高的可靠性。另外，在保护方式编程中会增加一些规定，本节将讨论这些规定。

80286 和 80386 上电后即处于实方式，在实方式下它们仿真 8086 工作。实方式不支持存贮保护，而且仅能支持 1M 字节存贮空间。我们重新设置 80286 和 80386 的机器状态字(MSW)的最低位，再作相应的辅助工作，就可以从实方式切换到保护方式。图 8-15 给出了 80286 和 80386 的机器状态字的格式。在保护方式下，处理器要检查对存贮器的每次访问，看其是否合法，以致系统内的每个程序(任务)均不能侵犯其它程序(任务)的存贮区。保护方式提供了多任务处理和虚拟存贮功能，只要用一条指令就可以在 22μs 内完成任务的切换。在保护方式下，80286 和 80386 能支持的最大存贮空间分别为 16M 字节和 4G 字节。另外，80386 还具有虚拟 8086 方式(VM86)。这种方式能使 80386 的保护方式的操作系统，把一个未经修改的 8086 程序作为系统内的一个任务来执行。

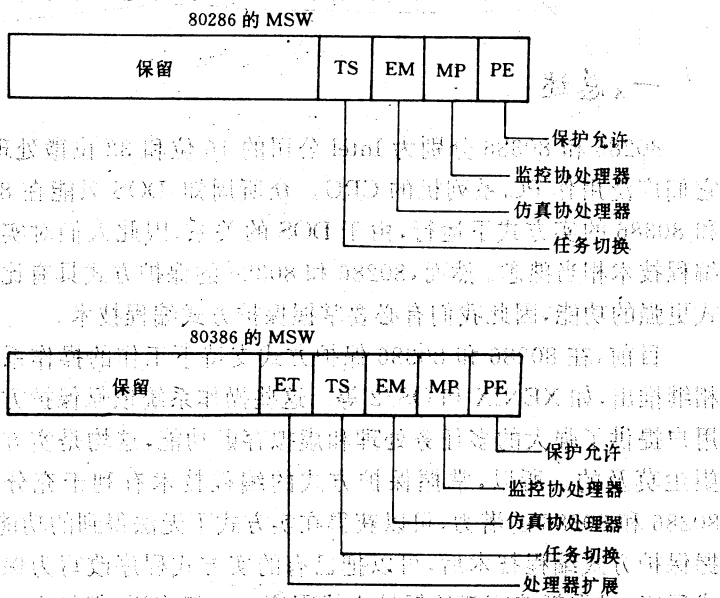


图 8-15 机器状态字的格式

二、数据结构

1. 段描述符

我们首先提出一个观点，不论在哪种方式下编程，均最好用符号来给段寄存器定值，而不要用直接地址值给段寄存器定值。例如，我们用段名符号 DSEG 来初始化 DS 寄存器，在实方式下，DSEG 代表段地址，在加载时系统装入程序用一确切的数值取代 DSEG。在保护方式下，系统装入程序则用一个选择符(selector)来取代 DSEG，这个选择符依靠一个称为描述符(descriptor)的 8 字节数据结构，间接指向这个段。

每个使用的段均有一个段描述符，段描述符内记录了其对应段的描述信息，它由系统装入程序建立。图 8-16 给出了段描述符的格式。

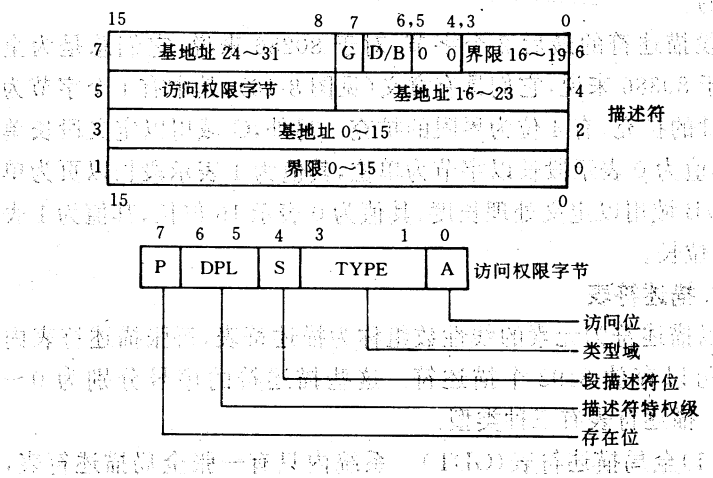


图 8-16 段描述符的格式

段描述符由 8 个字节组成。其中的“基地址”为段描述符所对应段的起始地址，系统装入程序把该段存放到这个区域，处理器也

使用这个地址来定位该段。它与实方式中的段地址类似。“界限”为段内允许的最大偏移量。若程序企图在某指令中使用大于该段界限的偏移量，处理器的保护机构会产生一个异常(exception)，避免执行这条指令。这样就能确保一个程序不会破坏其它程序的内容。

下面介绍“访问权限”字节。该字节中的 DPL 域用于指定该描述符的特权级，共有 0～3 级，0 级表示拥有最大的特权，3 级则反之。一个段不能访问特权级比它小的其它段。用特权级后，有助于保护功能的完备。例如，可以有效地防止应用程序破坏操作系统的代码和数据。访问权限字节的其它域还指出了允许对该段作什么样的访问。对于数据段，规定总是可读的，但是否可写入，可作出选择；对于代码段，规定总是可以执行的，但是否可读，可作出选择。这样，数据段的访问权限有“只读”和“可读可写”，代码段的访问权限有“只执行”和“可执行可读”。值得注意的是，代码段总是不允许写入的。

段描述符的最后 2 个字节，对于 80286 来说，它们总是为全 0，对于 80386 来说，它们具有意义(见图 8-16)。其中有 1 个字节为基地址的扩充，有 4 位为界限的扩充。另外，G 域用以定义段长单位，其值为 0 表示段长以字节为单位，其值为 1 表示段长以页为单位。D/B 域用以定义处理长度，其值为 0 表示 16 位长，其值为 1 表示 32 位长。

2. 描述符表

以描述符为元素的线性数组称为描述符表，每张描述符表内最多可以存放 8192 个描述符。这些描述符的序号分别为 0～8191。描述符表有三种类型：

(1)全局描述符表(GDT)　系统内只有一张全局描述符表，表内的描述符可以为各个程序使用，该表为各个程序所公用。

(2)局部描述符表(LDT)　每个程序均有一张局部描述符表，表内的描述符只能为拥有该表的程序使用。系统内可以有多张局

部描述符表，每表为一个程序私有。

(3)中断描述符表(IDT)　系统内只有一张中断描述符表，该表由系统内部使用。

每一种类型的描述符表都有一个与之相联系的描述符表寄存器。全局描述符表寄存器(GDTR)含有全局描述符表所在段的基地址和界限。局部描述符表寄存器(LDTR)和中断描述符表寄存器(IDTR)含有类似 GDTR 的内容。

3. 选择符

在保护方式下，为了对某个段进行访问，就应该把一个选择符装入段寄存器，选择符的格式如图 8-17 所示。其中的表指示域(TI)指出该段的描述符存放在何种类型的描述符表中(指 GDT 或 LDT)；索引域(Index)指出该段的描述符在描述符表中的序号。在把选择符装入段寄存器的过程中，处理器就可以根据以上两个域来定位该段的描述符，然后把该描述符中的基地址、界限和访问权限复制到扩充段寄存器中，扩充段寄存器为 CACHE 寄存器，与段寄存器一一对应。

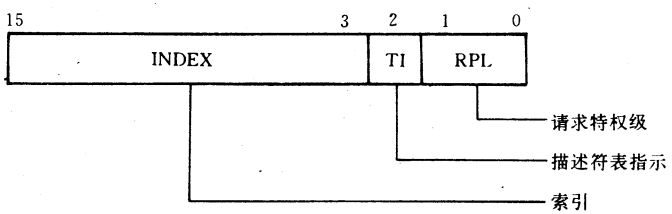


图 8-17 选择符的格式

当保护方式程序作装入段寄存器操作时，这个段寄存器和对应的扩充段寄存器中含有这个段的地址、界限和访问权限，处理器能据此实施存贮保护。当保护方式程序违反了保护规则时(如越界访问和越权访问等)，处理器就产生一个异常。

我们必须注意到，由于扩充段寄存器的内容只有在段寄存器的内容被改变时才变化，所以，修改描述符表的程序在改变了描述符之值后，必须对相应的段寄存器进行重新装入。

三、编程要点

保护方式与实方式有较大的差异，故保护方式的编程不能套用实方式的那套传统方法。有不少传统的编程方式在保护方式下会发生问题，下面对此作一讨论。

1. 段寄存器的定值

前面已经提到，提倡用符号给段寄存器定值，不难想到，在保护方式下和实方式下，同一个符号具有不同的内在含意，由相应的系统装入程序给于解释。因此，用符号给段寄存器定值的程序可以适合于两种方式。

那些使用直接地址值进行寻址的实方式程序，就不可能在保护方式下正常运行，其原因是，选择符的值并不等于段地址值。请看下列程序。

```
MOV   AX,0B800H                ;彩卡的段地址
MOV   ES,AX                    ;ES 指向显示存贮区
MOV   BYTE PTR ES:[0],'A';显示一个字符 A
```

这个程序实现向 PC 系列机的彩色显示卡存贮区写一个字符，以显示该字符。在实方式下，B800H 是彩色显示卡显示存贮区的段地址。在保护方式下，B800H 送入段寄存器后作为选择符来解释。其意义为，在局部描述符表中以 1700H 为索引来定位一个描述符。显然，这与段地址 B800H 完全是两码事。所以，该程序在保护方式下肯定不能完成预定的工作，甚至可能使这条指令的执行产生一个异常，因为若把 B800H 作为一个选择符的话，完全有可能是一个无效的选择符。

2. 对硬件的直接访问

在保护方式下，程序要严格避免对系统硬件作直接访问，而应

该利用操作系统提供的有关I/O服务来实现这种访问。由于保护方式下的环境往往是多任务环境，若在这种环境下不经过任何协调(如互斥和同步等)就对硬件直接访问，很有可能会产生预料不到的结果。其实，在实方式编程中也应提倡这一原则。在上述程序中，如果采用系统功能调用来显示字符，而不用直接写彩色显示卡显示存贮区的话，程序还可更简洁明了。有时出于性能的原因(如提高速度)而要直接访问硬件时，则应把这种操作限制在一个过程内。这样，今后若要在保护方式下对程序进行改进的话，则只要顾及这个过程就行了，而不必涉及到程序的主体部分。

3. 段值的运算

在保护方式下，如果在段值上进行运算的话，则会发生问题。例如，设一个程序中有两个相邻接的段，每个段均为64K字节。在传统方法中，常会把第一个段的段地址值加上1000H，以指向第二个段，实现对第二个段的访问。同样，也可能通过从第二个段的段地址中减去第一个段的段地址，来计算第一个段的装入量。由于保护方式下的选择符并不等于段地址，所以这两种作法在保护方式下均会发生问题。因此，保护方式程序绝对不能依赖段值进行运算。

4. 对代码段的操作

在保护方式下，代码段的访问权限只能是"只执行"或"可执行可读"，而绝对不可能为"可写"。所以，保护方式程序中决不可把写代码段作为某条指令的目标。也就是说，保护方式下不能进行写代码段单元的操作。因此，保护方式程序的代码段中不能定义工作单元和工作区，而应该把它们定义在数据段内。

5. 对中断向量的操作

中断向量表是保护方式程序设计中必须谨慎对待的一个对象，稍不注意就可能产生意外的结果。在实方式下，操作系统把中断处理程序的入口地址存放在中断向量表中，中断向量表位于内存的最低地址处400H个字节。在保护方式下，中断处理程序的入

口地址由中断描述符来描述，中断描述符存放在中断描述符表中。中断描述符表的格式与中断向量表完全不同，而且中断描述符表的地址也不是中断向量表的地址。所以，直接对中断向量表进行操作的实方式程序，就不能在保护方式下正常运行。为了避免发生这一问题，故应该在程序中采用系统功能调用来对中断向量进行操作。

四、特殊指令

在 80286 和 80386 的指令系统中，有一些指令在保护方式下被定为特殊指令。对这些特殊指令的使用必须十分细心，以防产生差错。这些特殊指令可分为敏感指令和特权指令两类，下面分别介绍它们的使用要点。

1. 敏感指令

为了使系统尽可能的可靠，操作系统应该控制保护方式下敏感指令的执行。以下指令为敏感指令。

IN	从 I/O 口读
OUT	向 I/O 口写
INS	向 I/O 口读一个串
OUTS	向 I/O 口写一个串
CLI	不允许中断(关中断)
STI	允许中断(开中断)

如果在程序中不能正确地使用上述指令，则有可能使系统崩溃。例如，若错用了一条 OUT 指令，则有可能会关闭 DMA 控制器，从而导致存贮器校验错。

操作系统可以根据标志寄存器 FLAG 中的 I/O 特权级(IOPL)域，来控制敏感指令的执行许可权。只有在当前代码段的特权级 CPL 小于 IOPL 时，才能执行当前代码段内的敏感指令，否则处理器将产生一个异常(详见图 8-18)。

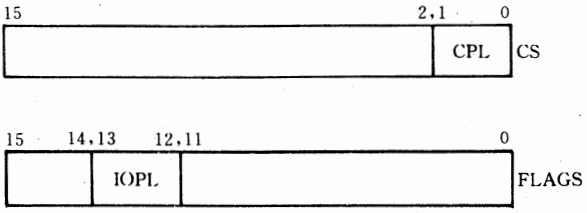


图 8-18　代码段的 CPL 和 FLAGS 寄存器的 IOPL

已有的实方式程序中若使用了敏感指令，则其在保护方式下可能会失败，这与操作系统对 CPL 和 IOPL 的设置有关。为了避免产生这种失败，可以先对这些特权级进行比较，以确定某个程序中能否执行敏感指令。下列过程可以完成这种比较。

```
checks    PROC
          PUBLIC   checks
          PUSH     AX
          PUSH     BX
          PUSHF              ;标志(IOPL)入栈
          POP      AX        ;复制标志到 AX
          AND      AX,3000H  ;屏蔽除 IOPL 外的所有位
          SHR      AX,12     ;向右调整 IOPL
          MOV      BX,CS     ;CPL 在 CS 中
          CMP      BX,AX     ;比较 CPL 和 IOPL
          JA       nosens    ;若 CPL>IOPL 则转
          CLC                ;可执行敏感指令
          JMP      exit
nosens    STC                ;产生异常
exit      POP      BX
```

```
        POP     AX
        RET
checks  ENDP
```

2. 特权指令

特权指令是另一类特殊指令，只有 CPL 为 0 级的系统程序和其它高特权程序才能执行这一类指令。由于这些特权指令易给系统带来极大的风险，故在应用程序中要严格限制它们的使用。以下指令为特权指令。

HLT	暂停处理器
LGDT	装入 GDT 寄存器
LIDT	装入 IDT 寄存器
LLDT	装入 LDT 寄存器
CLTS	清任务切换标志
LMSW	装入机器状态字
LTR	装入任务寄存器

另外，80386 的特权指令还有：把控制寄存器作为源或目标的 MOV 指令、把测试寄存器作为源或目标的 MOV 指令、把调试寄存器作为源或目标的 MOV 指令。

以上这些特权指令中，只有 HLT 指令能存在于实方式下。所以对已有的实方式程序，我们只要关心这一条指令就可以了。我们可以在保护方式程序中用以下过程来确定 HLT 指令能否执行。

```
checkp  PROC
        PUBLIC  checkp
        PUSH    AX
        MOV     AX,CS       ;CPL 在 CS 中
        AND     AX,3        ;屏蔽 CPL 外的所有位
        JNZ     noprivi     ;若 CPL 不为 0 则转
        CLC                 ;可执行特权指令
        JMP     exit
```

```
noprivi   STC              ;产生异常
exit      POP     AX
          RET
checkp    ENDP
```

五、界限问题

在实方式程序中，可以自由地使用比段的实际范围大的偏移量。例如，有两个段相互邻接，我们对第一个段设置段寄存器，然后使用比第一个段的段长大的偏移量，来实现对第二个段的访问。但是，这种作法在保护方式下是行不通的。由于保护机构的作用，这将会产生一个异常。我们为了在保护方式下避免发生这样的问题，在程序中可以使用装入段界限指令 LSL 来检查偏移量是否越出段界限。

LSL 指令有两个操作数，一个是 16 位的目标寄存器，另一个是寄存器或内存单元中的选择符。该指令能核实操作数中的选择符是否有效。如果发现选择符是无效的，则 LSL 指令会把零标志(ZF)复位。实际上，由于某些原因，选择符有可能会是无效的，最常见的是特权级不符合规定。如果选择符是有效的，LSL 指令就把选择符对应的描述符中的段界限复制到目标寄存器内。以下过程能通过 LSL 指令来确定某个选择符的偏移量是否有效。

```
checkl    PROC
          PUBLIC   checkl
          PUSH     DX
          LSL      DX,BX     ;得到段的界限
          JNZ      except    ;若选择符无效，则退出
          CMP      CX,DX     ;比较偏移量与界限
          JA       except    ;偏移量超出界限则转
          CLC                ;偏移量有效
          JMP      exit      ;返回调用者
```

```
except    STC                ;偏移量无效
exit      POP     DX
          RET
checkl    ENDP
```

六、80286 工作方式的切换

1. 进入保护方式

为了使 80286 从实方式切换到保护方式，从表面上看只要将机器状态字 MSW 中的最低位 PE 置为 1 即可，实际上在切换前还须作许多准备工作，否则无法达到目的。为简单起见，我们假定进入保护方式后，所有执行的程序的特权级均为 0。下面逐步介绍进入保护方式的工作。

(1)建立 GDT 表　我们不考虑局部描述符表 LDT，而只使用全局描述符表 GDT，这样可以使问题进一步简化。由于 DOS 本身并不是在保护方式下，所以对于其扩展部分和应用程序作这样处理也未尝不可。其实，系统提供的 15H 号中断处理程序的 87H 号功能块，也是以这样的方式在保护方式下实现扩展内存与常规内存间的数据传送的。

在进入保护方式前，应该根据进入保护方式后所要作的工作，建立相应的 GDT 表，以便进入保护方式后在程序把选择符装入段寄存器时，处理器能够正确地引用。GDT 表至少应该包括进入保护方式后执行代码之代码段描述符和存放 GDT 表的数据段的段描述符。如果还要涉及到栈操作，则应有用于栈的数据段描述符。如果要涉及存贮器的数据操作，则还应包括有关数据段描述符。

我们必须按照 GDT 表的规定结构来建立 GDT 表，具体地说，就是要按描述符的数据结构设置好 GDT 表内各个描述符域的初始值。在保护方式下，在程序的执行过程中，我们可以修改 GDT 表中描述符各个域之值，使其描述别的段。

(2)设置 GDTR　前面已经介绍过,80286 有一个全局描述符表寄存器 GDTR,用于存放 GDT 表所在段的基地址和界限。在保护方式下,当把选择符装入段寄存器时,80286 将根据 GDTR 指出的 GDT 表自动引用由选择符索引域确定的描述符,把其中的段基地址、界限和访问权限装入相应的段描述符 CACHE 寄存器(段扩充寄存器),以保证后面执行指令时,可以从 CACHE 寄存器内快速获取有关段的参数,从而形成物理地址。

为了使处理器能够在保护方式下使用我们建立的 GDT 表,所以应该根据已建立的 GDT 表来设置 GDTR 的初始值。

(3)选通地址线 A20　地址线 A20 有它的特殊性,我们从 PC 系列机的电路图上可以发现这一点。在实方式下,因为寻址范围仅 1M 字节,故不必使用 A20。但是在保护方式下,寻址范围可达到 16M 字节,要使用全部 24 根地址线,故在进入保护方式前必须选通地址线 A20。

选通地址线 A20 的方法是,把选通字节(DFH)送入键盘控制器 8042 的 I/O 口,这个口的地址为 60H。

(4)建立 IDT 表和设置 IDTR　严格地说,为了进入保护方式,还要建立中断描述符表 IDT 和设置中断描述符表寄存器 IDTR,以保证在保护方式下发生中断和异常时获得中断向量和进行相应的处理。为了简单起见,我们可以在进入保护方式前关中断,并且假设在保护方式下运行时不发生异常。当然,如果不能保证以上假设成立的话,我们可以引用 ROM－BIOS 中的 IDT 表,以及象设置 GDTR 那样来设置 IDTR。

(5)为退出保护方式作准备　程序在进入保护方式后,往往还要回到实方式,所以在进入保护方式前,要为将来退出这一方式作些准备工作,其中最重要的工作是保存中断控制器 8259A 的当前状态。在退出保护方式而回到实方式时,必须对 8259A 进行初始化。为了使程序在回到实方式后能正常工作,故必须使 8259A 初始化后,仍能保持其进入保护方式前的状态。因此,在进入保护方

式前必须保存 8259A 的当前状态，以便将来回到实方式时恢复。具体的作法是，保存 I/O 口 21H 和 A1H 的当前值。

(6)状态切换　在作好了上述准备工作后，就可以进行状态切换，进入保护方式了。原则上讲，只要执行一条装入机器状态字指令 LMSW，把 MSW 中的最低位置为 1，就进入保护方式了。但是，仅仅作这一步还不能正确地由实方式切换到保护方式，因为此时的代码段寄存器 CS 内装的并不是对应于代码段描述符的选择符，相应的 CACHE 寄存器内也未装入当前执行代码段的有关参数。所以，在执行 LMSW 指令后，必须马上执行一条远程转移指令，从而实现把指向当前代码段描述符的选择符装入 CS 寄存器，以及把代码段的有关参数装入相应的 CACHE 寄存器。

事实上，如果在 LMSW 指令后安排一条远程转移到进入保护方式后执行的代码段相应处的指令，那末在执行 LMSW 指令时，这条远程转移指令已经被预取到指令队列中。执行完 LMSW 指令，处理器已经切换到保护方式，由于指令已被预取，故其能在保护方式下，不依赖 CS 段 CACHE 寄存器而执行远程转移。在执行这条远程转移指令时，将清指令预取队列，用相应的选择符装入 CS 寄存器，用相应的段参数装入 CACHE 寄存器。由此可见，在执行完这条远程转移指令后，重新取指令时，CS 寄存器和 CACHE 寄存器的内容已经都符合要求了。

2. 退出保护方式

当程序需要退出保护方式时，同样不是仅将机器状态字 MSW 的最低位清为 0 就能实现的，也要为此作一系列的工作。下面介绍退出保护方式的方法。

我们可以利用键盘控制器 8042 来重新设置系统，从而回到实方式。具体作法是，向 8042 的状态口写入 FEH 值，该状态口的 I/O 口地址为 64H。这种方法既简单又方便。

系统重置后，处理器从 FFFF:0000H 处开始执行指令，此处存放的是一条转移指令，转移至 F000:E05BH 处。随后是进行处

理器自身的最基本检测，然后判别是否从停止中恢复，若是的话，则按照指定的停止请求进行相应的恢复工作。停止请求信息保存在CMOS中的停止状态字节(0FH)中。如果我们使用5号停止请求，则在进行8259A的初始化工作后，恢复工作只是执行一条远程间接转移指令。这条间接转移指令所使用的转移地址单元为0040:0067H～006AH。

针对上述机制，我们使用5号停止请求进行恢复工作的话，应该在进入保护方式前的准备工作中，事先将CMOS中的停止状态字节置为5(CMOS的索引口和数据口的地址分别为70H和71H)，在内存单元0040:0067H～006AH中设置程序退出保护方式后，在实方式下执行代码的偏移量和段值。这样，便可以从保护方式切换到实方式了，并从程序的相应处开始在实方式执行。

七、80386工作方式的切换

1. 进入保护方式

(1)建立GDT表　我们把在保护方式下要用到的代码段和数据段描述符组成GDT表。为了简单起见，可以不考虑LDT表，甚至可以假定不发生中断和异常而不考虑IDT表。GDT表的第一个描述符是空的，第二个描述符起是有效的，其中的描述符可以是代码段和数据段描述符，也可以是系统描述符。一般来说，GDT表中至少有一个代码段描述符、一个数据段描述符和一个堆栈段描述符。

建立GDT表就是要根据进入保护方式后对段使用的要求，按照描述符的数据结构，对各个描述符的域置上有意义的值。数据段描述符的访问权限字节可置为93H，表示该段为已在内存中存在、可读可写、向上增长的数据段；代码段描述符的访问权限字节可置为9BH，表示该段为在内存中存在、向上增长的代码段；堆栈段与数据段类似。为了兼容和简单起见，我们可以把段界限定为FFFFH，把G域和D/B域均定为0。另外必须要作的是设置32位

的段基地址，若系统中未配置满 16M 字节的内存，则基地址的高 8 位均为 0，故所要作的工作是把实方式下用两个 16 位表示的地址值转换成 24 位表示。

(2)设置 GDTR　前面已经介绍过 GDTR 的作用，在建立了 GDT 表后，应该为 GDTR 设置相应的值，使其含有当前所建 GDT 表的段基地址和界限。80386 的 GDTR 用 16 位保存界限，用 32 位保存基地址。我们在程序中可以用一个 6 字节的变量来存放 GDT 表的段基地址和界限，然后用装入 GDTR 指令 LGDT，把该变量中的内容装入 GDTR。

(3)其它准备　如果具体情况并不象我们前面所作出的假设那样简单，则要建立 LDT 表和 IDT 表，并要设置 LDTR 和 IDTR。

为了在保护方式下能使用 1M 字节以上的内存，故在进入保护方式前须选通地址线 A20。这部分工作已在前面介绍过了，所以在这儿就不再重述。

为了确保工作方式切换的顺利进行，我们可以在工作方式切换前关中断。如果在保护方式下不考虑中断和异常的发生，那末在进入保护方式后也不开中断，一直到重新回到实方式后才开中断。这样作后，能使许多事情得到简化。

(4)状态切换　80386 有三个 32 位的控制寄存器(CR0，CR1，CR3)，用于存放全局特性的机器状态。控制寄存器 CR0 的高 16 位中仅用了最高位(用于表示分页允许)，其它 15 位均为保留。为了与 80286 兼容，CR0 的低 16 位与 80286 的机器状态字相同，只要把其最低位置为 1，即可进入保护方式。

现在我们要从实方式切换到保护方式，同时不允许分页，因此只要把 00000001H 装入 CR0 就能实现这种切换。装入操作可以用“MOV　CR0，EAX”指令实现。但是，为了在工作方式切换后程序能正确地执行下去，故必须在上述指令后安排一条远程转移指令，使其转移至保护方式下执行代码的入口处。这样作的具体原因已在前面介绍过了，这儿不再重复。

2. 退出保护方式

将控制寄存器CR0的最低位清为0，便能从保护方式退出，并进入实方式。但是在这样作之前，还必须作一些准备工作，主要是把适合于实方式的段界限和其它参数装入各段相应的CACHE寄存器中。如果在保护方式下是允许分页的，则还应该清控制寄存器CR3。

段CACHE寄存器适合于实方式工作时各个域之值如下：

①BASE域(基地址)＝相应段地址

②LIMIT域(界限)＝64KB

③G域(单位)＝0，以字节为单位

④ED域(扩展方向)＝0，向上扩展

⑤W域(写许可)＝1，可读可写

⑥P域(存在)＝1，已在内存中

为了使各个段的CACHE寄存器具有上述值，我们可以在GDT表中安排一个具有以上值的描述符，然后把指向该描述符的选择符装入到SS、DS、ES、FS、GS等段寄存器，由80386自动把描述符之值装入相应段的CACHE寄存器。

我们通过把控制转向具有上述合适值的一个代码段，也可使代码段的CACHE寄存器内含有适合于实方式的值。为了简单起见，我们可以在由实方式切换到保护方式时，就切换到具有上述合适值的代码段。

在关中断的前提下，把CR0中的最低位清为0，就实现由保护方式到实方式的切换。以上操作也是通过“MOV　CR0,EAX”指令，把00000000H装入CR0来完成的。同样，在该指令后应安排一条远程转移指令，以保证进入实方式后程序能正确地执行下去。

参 考 资 料

[1]钱培德，计算机中文信息处理技术，电子科技大学出版社，1992年。

[2]钱培德，CC－DOS V4.0 操作系统高级技术分析，吉林科学技术出版社，1991 年。

[3]钱培德等，CC－DOS 操作系统技术大全，清华大学出版社，1992 年

[4]钱培德等，微机汉字操作系统实用开发技术，北京师范学院出版社，1991 年

[5]周利化、李凤华，DOS 操作系统内核剖析，西安电子 科技大学出版社，1992 年。

[6]张载鸿，局部网操作系统 DOS 高级技术分析，国防工业出版社，1990 年

[7]郭平欣、张淞芝，汉字信息处理技术，国防工业出版社，1985 年。

[8]赵珀璋、徐力，计算机中文信息处理，宇航出版社。1987 年。

[9]王绪龙，汉字信息处理，国防工业出版社，1990 年。

[10]曾庆辉，汉字信息处理系统，东南大学出版社，1989 年。

[11]李亦何，软件的汉化技术，同济大学出版社，1991 年。

[12]李盘林等，第六代计算机信息处理基础——联想记忆原理，大连理工大学出版社，1991 年。

[13]贝贵琴、张学涛，汉字频度统计，电子工业出版社，1988 年。

[14]钱培德，试论基于 RAM 的汉字库结构，中文信息学报，1989 年第 3 期。

[15]钱培德，一个 MS－DOS 的 SPOOLing 系统，中文信息学报，1989 年第 2 期。

[16]钱培德，计算机汉字I/O处理数学模型的研究，中文信息学报，1992年第2期。
[17]钱培德，MS－DOS的内存管理机制及其优化，计算机研究与发展，1990年第9期。
[18]钱培德，多级中文词库的研究，计算机研究与发展，1993年第6期。
[19]钱培德，全汉字集内部码的设计与研究，计算机研究与发展，1992年第3期。
[20]钱培德，MS－DOS虚存管理系统VMMS的设计，计算机研究与发展，1993年第6期。
[21]吕强、钱培德，基于词组的智能化汉字输入系统CIIIS/2的设计，中文信息学报，1992年第1期。
[22]吕强，试论键盘管理模块标准，中文信息学报，1991年第2期。
[23]张旭波，面向个人的词码输入方式及其软件支持系统PLED，计算机学报，1991年第12期。
[24]吴克西，汉字二级存贮系统及其LRU算法分析，软件学报，1992年第1期。
[25]蒋贤春，中文词输入的自适应系统，中文信息学报，1990年第3期。
[26]张侃、陈一凡，汉字键盘输入的认知模型，中文信息学报，1991年第4期。

津新登字(90)003号

责任编辑:徐　彤

DOS 汉字系统高级技术

钱培德　陆建明　编著

*

天津科学技术出版社出版

天津市张自忠路 189 号　邮编 300020

天津市武清县振兴印刷厂印刷

新华书店天津发行所发行

*

开本 850×1168 毫米 1/32 印张 11 字数 271 000

1994 年 4 月第 1 版

1994 年 4 月第 1 次印刷

印数:1—3 000

ISBN 7-5308-1533-4

TP·48　定价:8.75元