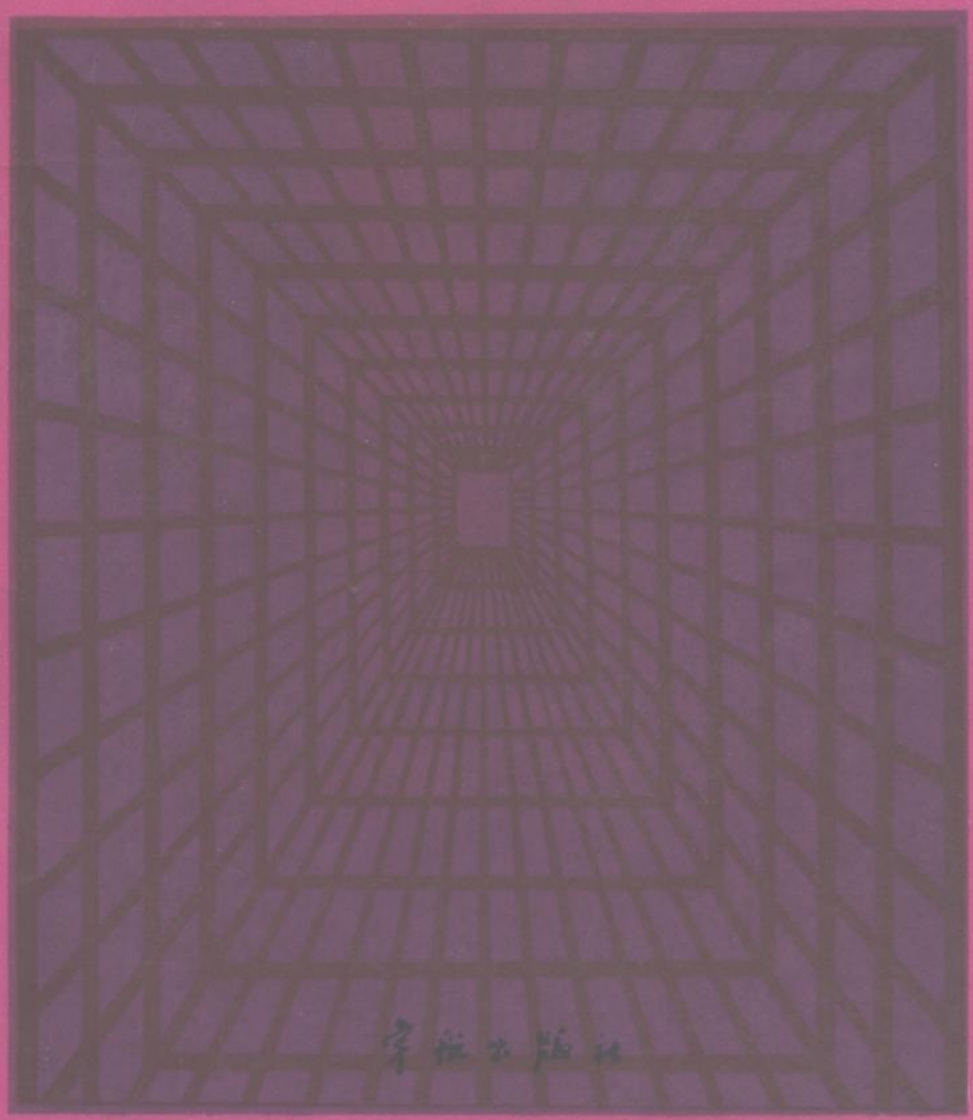
实用用户界面设计
——C 程序员参考

刘宏程　张　鹏　崔俊海　刘　军　编著



660

371385

实用用户界面设计

——C程序员参考

刘宏程　张　鹏
崔俊海　刘　军　编著

宇航出版社

(京)新登字 181 号

内 容 简 介

良好的用户界面设计已成为现今软件设计中不可缺少的一部分，据统计，用户界面部分已占程序设计总量的 40%。本书介绍了几种流行的用户界面设计方法，所有实例均以 C 语言给出(适用于 Turbo C 和 MSC)，同时也可以帮助读者进一步熟悉 C 语言的编程。本书内容主要分为两大部分：第一部分为通用的用户界面程序设计方法，通过实例，详细地介绍了在文本方式下各种类型的窗口设计方法，弹出菜单、亮条菜单和下拉菜单的编程技巧，以及如何在应用程序中增添在线帮助系统的方法等实用界面设计技巧；第二部分考虑了 EGA/VGA 用户的特殊要求，介绍了类似 NORTON 6.0，PCTOOLS 7.0 窗口和菜单的编程方法，并提供三套为用户界面服务的自定义字符集供读者参考。全书取材新颖、内容系统、构思严谨、层次分明、实例丰富，对于 IBM PC 及其兼容机上从事软件开发研制的广大计算机工作者，以及大专院校计算机、自控、无线电、管理等专业的学生的学习和实践有很好的参考价值。

实用用户界面设计

——C 程序员参考

刘宏程 张 鹏 编著

崔俊海 刘 军

责任编辑：赖巧玲

宇航出版社出版发行

北京和平里滨河路 1 号发行(100013)

发行部地址：北京阜成路 8 号(100830)

各地新华书店经销

北京科技印刷厂印刷

开本：787×1092 印张：18.75 字数：465 千字

1993 年 8 月第 1 版第 1 次印刷 印数：1-8000 册

ISBN 7-80034-600-5/TP・038 定价：13.20 元

前　言

随着计算机技术的发展和进步，软件市场逐步扩大，各种新型软件层出不穷，这些新软件具有非常优美的外观和友好的用户界面，漂亮的窗口设计、灵活的菜单操作、便利的在线帮助系统令广大用户赞叹不已。据统计，当今程序设计中，用户界面部分的设计占整个程序设计总量的40%，因此，良好的用户接口已成为软件不可缺少的一部分。而许多用户对繁杂的界面设计往往感到无从下手，本书旨在帮助这些用户澄清概念，并介绍了几种流行的用户界面设计方法。

本书介绍了如何用C语言(Turbo C和MSC)设计专业化的用户界面，并重点介绍了通用的程序设计方法；同时兼顾高级用户的一些特殊要求，又介绍了利用EGA/VGA特有资源的特殊编程技巧。第一章着重介绍各种显示适配器的结构以及访问方法；第二章和第三章，从应用的角度出发，通过实例，分别介绍窗口技术、屏幕函数的设计方法、各种实用菜单技术和编程技巧，这是本书的重点章节；第四章详细介绍用于用户接口的数据输入屏的设计方法和编程技巧；第五章和第六章，主要介绍用于菜单选择的两种实用方法——列表选择和目录函数的实现；第七章介绍如何在用户的程序中加入在线帮助窗口，以及如何编辑帮助文件等实用技术及其实现方法；对于拥有EGA/VGA显示适配器的高级用户，第八章重点介绍一些特殊的编程技巧，如装载用户自定义字符集、在文本方式下显示汉字技术等，并提供三套为用户界面服务的自定义字符集供读者参考。所有各章节均附有实例程序，供读者参阅，所有程序均在机器上调试通过。

本书适用于计算机软件研制、开发人员和大专院校的计算机、自控、无线电、管理等专业的学生。

本书的作者都是具有多年编程经验，并长期从事计算机工作的科研人员。在编写过程中，力求深入浅出、概念清楚、举例典型，既照顾大多数计算机初学者，又考虑了高级用户的要求。所编制的程序适用于Turbo C和MSC两种编译器，虽然它们的设计并非最佳，程序也未必完善，但对于读者掌握基本的设计方法，澄清概念，相信会有一定的参考作用。

由于时间仓促以及限于编者的水平，书中错误和不妥之处在所难免，敬请读者不吝批评指教。

本书所有应用与实例程序均附有软盘，如果需要，可与宇航出版社软件部联系索购。

编者　1992年12月

目　录

第一章　访问显示适配器 …… (1)
1.1　显示适配器 …… (1)
1.2　复合显示 …… (2)
1.3　视频缓冲区 …… (2)
1.4　文本属性 …… (2)
1.5　彩色文本页 …… (4)
1.6　指针和内存 …… (5)
1.7　直接访问视频缓冲区 …… (6)
1.8　BIOS 中断 …… (7)
1.9　执行一个中断 …… (8)
1.10　ANSI 控制台驱动程序 …… (9)
第二章　窗口和屏幕函数 …… (11)
2.1　简介 …… (11)
2.1.1　什么是窗口 …… (11)
2.2　应用实例 …… (12)
2.2.1　选择编译程序 …… (12)
2.2.2　函数原型和编码技术 …… (13)
2.2.3　窗口模块的使用 …… (13)
2.2.4　生成库 …… (14)
2.2.5　外部变量 …… (15)
2.2.6　窗口函数的类型 …… (16)
2.2.7　缺省函数 …… (16)
2.2.8　窗口指定函数 …… (18)
2.2.9　内部函数 …… (23)
2.2.10　函数的使用 …… (24)
2.2.11　最优窗口布局 …… (27)
2.2.12　实用技巧 …… (30)
2.3　编程技巧 …… (31)
2.3.1　窗口类型 …… (31)
2.3.2　窗口的刷新 …… (33)
2.3.3　滑动虚拟窗口 …… (33)
2.3.4　外部参数(数据结构) …… (34)
2.3.5　窗口数据结构 …… (37)
2.3.6　写屏 …… (38)
2.3.7　虚拟屏 …… (41)
2.3.8　写虚拟窗口 …… (41)
2.3.9　光标管理 …… (42)
2.3.10　消除屏幕雪花 …… (43)
2.3.11　雪花和非屏幕输出 …… (44)
2.3.12　屏幕刷新期间的最优性能 …… (45)
2.3.13　总结 …… (45)
2.4　程序清单 …… (45)
第三章　菜单设计 …… (96)
3.1　简介 …… (96)
3.1.1　用户的观点 …… (96)
3.1.2　程序员的观点 …… (98)
3.2　弹出式菜单 …… (99)
3.2.1　弹出式菜单应用实例 …… (99)
3.2.2　全菜单与部分菜单 …… (104)
3.2.3　弹出式菜单编程技巧 …… (106)
3.2.4　关键函数 get __ key() …… (107)
3.3　移动亮条菜单 …… (108)
3.3.1　移动亮条菜单简介 …… (108)
3.3.2　移动亮条菜单应用实例 …… (110)
3.3.3　移动亮条菜单编程技巧 …… (111)
3.4　下拉菜单 …… (111)
3.4.1　下拉菜单简介 …… (111)
3.4.2　下拉菜单应用实例 …… (113)
3.4.3　下拉菜单编程技巧 …… (120)
3.5　程序清单 …… (122)
3.5.1　弹出式菜单应用程序清单 …… (122)
3.5.2　移动亮条菜单应用程序清单 … (128)
3.5.3　下拉菜单应用程序清单 …… (135)
第四章　数据输入屏 …… (144)

此为试读,需要完整PDF请访问: www.ertongbook.com

4.1 简介 …… (144)
4.1.1 域编辑器 …… (144)
4.1.2 数据输入屏编辑器 …… (145)
4.2 应用实例 …… (145)
4.2.1 数据输入屏(多个域) …… (146)
4.2.2 数据校正 …… (148)
4.3 编程技巧 …… (150)
4.4 程序清单 …… (153)
第五章 列表选择 …… (167)
5.1 简介 …… (167)
5.2 应用实例 …… (168)
5.3 编程技巧 …… (168)
5.3.1 移动选择方式 …… (169)
5.3.2 快速搜索方式 …… (171)
5.4 程序清单 …… (173)
第六章 目录函数 …… (181)
6.1 简介 …… (181)
6.2 应用实例 …… (181)
6.3 编程技巧 …… (182)
6.4 程序清单 …… (185)
第七章 求助屏 …… (190)
7.1 系统简介 …… (190)
7.2 应用实例 …… (191)
7.2.1 在应用程序中增加求助屏功能 …… (191)
7.2.2 创建帮助文件 …… (192)
7.3 编程技巧 …… (194)
7.3.1 窗口布局 …… (195)
7.3.2 帮助文件的编辑 …… (195)
7.3.3 帮助文件的结构 …… (196)
7.3.4 readhelp.c 模块 …… (197)
7.4 程序清单 …… (198)
第八章 EGA/VGA 字符发生器编程 …… (216)
8.1 简介 …… (216)
8.2 字符与字符发生器 …… (216)
8.2.1 EGA/VGA 字符发生器 …… (217)
8.2.2 字模结构 …… (218)
8.2.3 EGA/VGA 字符发生器的组织 …… (219)
8.2.4 EGA/VGA BIOS 中的字符发生器处理功能 …… (220)
8.2.5 兼容性 …… (223)
8.2.6 字符间空白 …… (223)
8.2.7 编程技巧 …… (224)
8.2.8 实例——文本方式下汉字的显示 …… (226)
8.3 用户字符集 …… (231)
8.3.1 用户字符集的装载 …… (233)
8.3.2 用户字符集的应用 …… (235)
8.3.3 Ctrl-Break 与程序退出 …… (238)
8.4 窗口部件 …… (242)
8.5 修改库文件 …… (246)
8.6 程序清单 …… (247)

第一章　访问显示适配器

本章将讨论：

- 显示适配器的类型
- 视频缓冲区的内存映像
- 如何用指针来访问视频内存
- 编译器内存模式的缺省指针类型
- BIOS 中断
- ANSI 控制台驱动程序

第一节将讨论显示适配器，着重讨论视频缓冲区及访问技术。

1.1　显示适配器

现在，IBM PC 上种类繁多的显示器和文本/图形卡不外乎两大类：单色和彩色。演化树如图 1-1 所示。

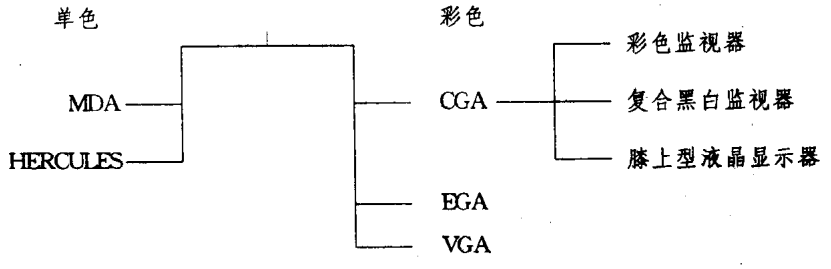


图 1-1　主要显示适配器的演变

主要显示适配器如下：

MDA＝Monochrome Display Adapter。这种卡主要用于商用，它可以显示定义得很清楚的文本字符而不能显示图形。文本按 80×25 的格式(25 行 80 列)显示。这种卡需要配单色显示器。

Hercules＝这种卡在设计上提供与 MDA 的兼容性，但也可以显示专门的高分辨率图形。这种卡也要配单色显示器。

CGA＝Color Graphics Adapter。这种卡既可以显示彩色文本也可以显示彩色图形。显示文本的最大限度是 80×25。

EGA＝Enhanced Graphics Adapter。这种卡与 CGA 兼容，但提高了文本和图形的分辨率。

支持超过 80×25 的文本显示方式。

VGA=Video Graphics Array。与 EGA 兼容,但具有更多的图形和文本显示方式。

LCD=Liquid Crystal Display。液晶显示卡,用于膝上型微机。这种卡一般等价于 CGA 卡。

后出的、更好的卡支持先出的卡的功能和显示方式。例如,VGA 卡可运行为 CGA 写的程序,Hercules 卡可仿真 MDA。如果软件与 CGA 和 MDA 兼容,该软件就能与新出的适配器保持兼容。

1.2 复合显示

应注意,CGA 可以配两种显示器:彩色和黑白。黑白复合显示将彩色转换成灰度显示,只实现了有限的功能。

彩色字符显示在复合显示器上,一般是不可读的。很多膝上机的 LCD 可以组成复合显示,因为大多数 LCD 显示器使用灰度级而不是彩色。

这种黑白显示器的用户常用 DOS 的"mode"命令来撤消彩色,只允许正常或加重显示文本,这样更清楚一些。在 DOS 的命令行敲入"mode bw80",设置成 80 列的黑白方式。

编得好的软件应通过 BIOS 调用确认用户当前选择的方式。在这种方式下,软件可为显示器选择适当的颜色(如是否要彩色)。遗憾的是,如果应用程序直接写视频缓冲区,就旁路了 BIOS,并且不受"mode"命令的影响,因此,可能出现按彩色文本属性来显示文本,即使这对用户的监视器来说不合适。

尽管软件可以检测现在使用的显示适配器的类型,但是并不能知道配到适配器上的显示器是彩色的还是复合的。软件只能确认是否已设置了黑白方式。

本书后面的程序中采用了一些技巧来确认显示适配器的类型和当前选择的显示方式。这些信息对产生清楚的显示是很基本的。

1.3 视频缓冲区

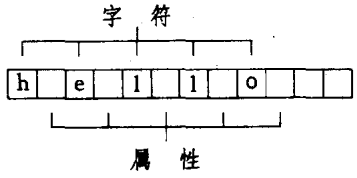


图 1-2 内存映像

IBM PC 用内存的一部分做视频缓冲区。CGA 和 CGA 的兼容卡使用内存中段址为 B800H 的一段缓冲区,单色卡(MDA,Hercules 等)使用段址为 B000H 的内存作缓冲区。第一个内存单元中存贮显示器上显示的第一个字符,下一个单元存贮这个字符的属性。这种安排称为内存映像显示。视频缓冲区的任何变化马上就可以在视频显示器上反映出来。

如果在 CGA 显示器的左上角显示"hello",内存映像如图 1-2 所示。在 80×25 显示器上的全屏显示由 2000(80×25)个字符和 2000 个属性组成。在对应屏幕上列、行按(x,y)方式编址系统中,显示器的左上角是(1,1)而右下角是(80,25)。对于视频内存映像的了解是开发直接写屏程序的基础。

1.4 文本属性

文本属性根据使用的视频卡的不同而不同。可在单显系统上使用的属性包括正常、加重、

反显和下划线。本书的程序利用头文件 mydef.h 来定义这些属性。

```
#define UNDERLINE 1   /* ATTRIBUTES FOR MONOCHROME CARDS */
#define NORMAL    7
#define HI__INTEN 15
#define REVERSE   112
```

在彩色适配器上，属性字节可用来设置前景/背景颜色，也可使前景加重和闪烁。彩色属性映像如图 1-3 所示。

mydef.h 中对彩色文本属性作了如下定义：

```
#define BLACK     0   /* THESE ARE FOR COLOR CARDS */
#define BLUE      1
#define GREEN     2
#define CYAN      3
#define RED       4
#define MAGENTA   5
#define BROWN     6
#define WHITE     7
#define YELLOW    14  /* intensity set on */
```

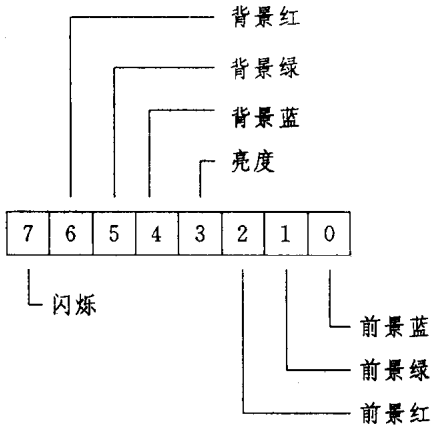


图 1-3　彩色属性的位图

下述与函数类似的宏用于设置前景/背景颜色：

```
#define set__color(foreground,background)\
(((background)≪4)|(foreground))
```

表 1-1　主要颜色位图

R	G	B	颜色	属性(十进制)
0	0	0	黑	0
0	0	1	蓝	1
0	1	0	绿	2
0	1	1	青	3
1	0	0	红	4
1	0	1	粉红	5
1	1	0	褐	6
1	1	1	白	7

注意宏是怎么将 background 左移 4 位，然后将它与前景混合(位“或”)”的。例如，利用这个宏，变量 attribute 可被设置成前景颜色为蓝色，背景为黑色。调用方式如下：

```
attribute=set__color(BLUE,BLACK);
```

如果属性的第四位被置位(设为 1)，则亮度被设成高亮。可以利用宏 set__intense()将这一位置为 1，它执行与十进制数 8(00001000B)的位或操作来将该位强制置 1。

```
#define set __intense(attribute)  ((attribute)|8)
```

1.5 彩色文本页

在80列文本模式中(CGA)可有多至4个文本页，通过视频中断，可以在计算机屏幕上显示任一页。在前面提到过，80×25显示方式的视频缓冲区占4000字节。视频缓冲区的内存地址偏移量是4096而不是4000。如果不从十六进制来理解的话，这似乎有点令人费解。正如表1-2和图1-4中所示。页起始地址增量为1000H，每页中有一小块未用，页边界以这种方式组织是为了使访问视频缓冲区时的计算更容易。

表1-2 实际占用的字节数

文本页	十进制	十六进制
页0	0至3999	0至f9f
页1	4096至8095	1000至1f9f
页2	8192至12191	2000至2f9f
页3	12288至16287	3000至3f9f

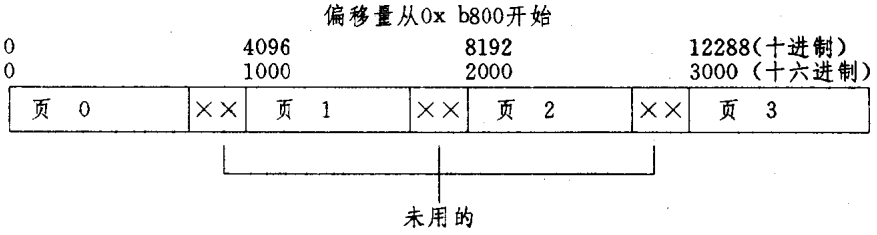


图1-4 视频缓冲区

一般情况下，计算机是对0页进行读写。一些程序员使用一种小技巧，即文本页的切换：在写一页的时候显示另一页。这使屏幕更新几乎不花时间，图1-5给出了一个例子。当然，向交替页中写信息和通常用的时间一样长，但用户看不见象正常情况下出现的一行一行的写过程。在屏幕上一下子"弹出"一整页，用户感觉相当快。

字处理程序可以用文本页来存贮当前的一页文本，也可以存贮上一页或下一页。当用户按换页键的时候，相应的文本页就显示出来了。当用户正在读新的一页时，其它页正在被重写。就在人决定下一步做什么时，软件充分利用了这段"空闲"时间。用户并不知道屏幕以外的事情。

如果一个文字处理程序可在文档间快速翻页的话，可能用的就是这种技术。你可以通过过载来检验。按住翻页键(PgDn)，看翻页是不是那么快。如果程序在两次按键之间来不及更新下一文本页，显示下一页就会有一段延迟。

尽管文本页切换很有用，但在本书中并不打算使用该技巧，原因有二：

1. 本书的代码支持单显，单显上不能用这种技术。

2. 在某些情况下，可能需要四页以上的文本页，为了达到和文本页切换同样的效果，本书

中的程序使用了虚屏技术。

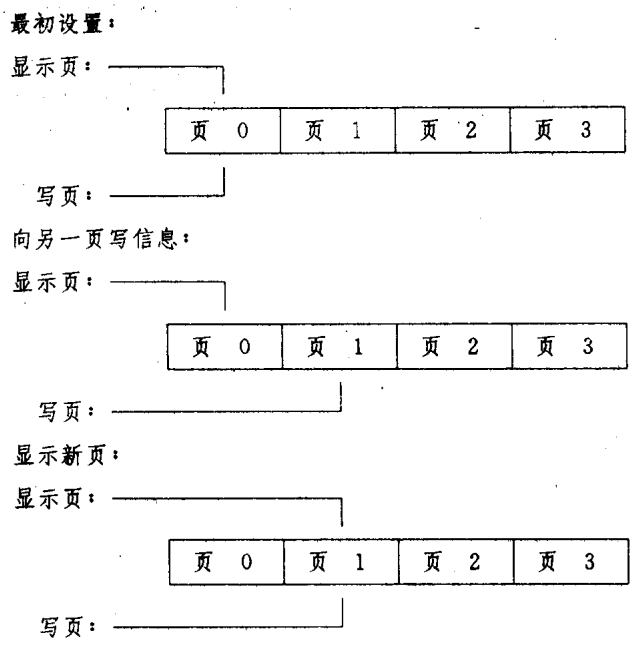


图 1-5 翻页示例

虚屏

对学计算机的人来说，虚意谓着“想象”或“虚构”。例如，RAM 盘也被称为虚盘。使用虚屏技术可以实现和页切换一样的功能。虚屏是一段连续分配的内存，一般来说和物理视频缓冲区一样大。写幕直接写到这个缓冲区而不是写到视频缓冲区。写完后，虚屏缓冲区再拷到物理视频缓冲区。

使用虚屏也有些小缺点，建立和操作缓冲区需要额外的内存和时间。但为了实现屏幕的快速更新，额外占些内存是值得的。

1.6 指针和内存

直接操作内存缓冲区使屏幕函数的性能达到了最优。为操作内存缓冲区，充分理解指针的使用及其与编译器内存模式的关系至关重要。

有几个专门的 16 位寄存器用于表示代码和数据在哪一段。段寄存器如表 1-3 所示。

表 1-3 段寄存器

寄存器	名称	用法
CS	Code segment	程序代码
DS	Data segment	数据存贮
SS	Stack segment	堆栈分配
ES	Extra segment	根据需要，一般存贮更多的数据

此为试读，需要完整PDF请访问：www.ertongbook.com

一般来说，汇编语言程序员比C语言程序员更关心段寄存器。此处要关心的只是段是如何影响对编译器内存模式的选择。Turbo C有六种模式：tiny，small，medium，compact，large和huge。所选择的模式决定了段寄存器的使用。

Tiny模式中四个段的起始地址都是一样的：代码段和数据段共占用64k的空间。这意味着可以用near指针来存取数据，程序可以使用在同一64k段中的数据。near指针只含有数据的偏移量。

large模式对数据使用far指针。far指针包含有段址和偏移量，所以比near指针要大。large模式可以存取1MB的数据。

在tiny，small和nedium模式中访问数据使用near指针，而compact，large和huge使用far指针。除非程序需要很大的数据空间，否则最好用小一些的内存模式，因为这样需要的内存较少而且速度快。

对于某些C应用程序，没有必要关心缺省的指针类型。一般来说，内存模式所对应的缺省指针对内存分配和访问变量已足够了。但对于处理视频缓冲区来说，就是另一码事了。

1.7 直接访问视频缓冲区

视频缓冲区一定要通过far指针来访问，必须保证实际使用的指针是far类型。下面程序中用缺省指针往屏幕上写一字符。

```
main()
{
char * screen;
char ch;
    screen=0xb00000000;/ * Monochrome screen address * /
    ch= 'a' ;
    * screen=ch;
}
```

这段程序能在large模式下编译并正确运行，因为large模式的缺省指针类型是far。这段程序不能在small模式下运行。因为small模式使用near指针，该指针不能指向所定义数据段外的数据。

可以这样修改screen指针的说明，使这段程序可在任意模式下运行：

```
char far * screen;
```

far修饰符并不是标准C语言提供的，但很多C编译器支持它，包括Turbo C和Quick C。另一个必须要注意的地方是C函数库的使用。有些函数使用了缺省内存模式指针，因此不能按程序员的意图去正确执行。比如函数movmen()将一块内存中的数据移到另一块内存中。它使用了三个参数：

- 源块的指针
- 目的块的指针
- 要移动的字节数

像movmen()这样的库函数可以用来一次移动一大块数据（例如，将视频缓冲区的内容拷贝到另一地方）。这样的子程序是用高效的汇编语言写的，比起C程序在循环中一个一个字节地写要快。函数movmen()也可用于将单个字符移动到视频缓冲区，例如：

```
#include "mem.h"      /* header file for movmem */
main()
{
char *screen;
char ch;
    screen=0xb0000000;/* Monochrome screen address */
    ch='a';
movmem(&ch,screen,1);
}
```

这段程序在大模式下运行得很好，但不能在小模式下运行。不象刚才的例子，将指针换成 far 也没有用。movmem()函数只接受缺省的指针。这个函数不能用来访问视频缓冲区，除非让它在大内存模式下运行。

将一大块数据移到视频缓冲区中的更好的方法是 movedata()函数。要给出源段及其偏移量，目的段及其偏移量和要移动的字节数。movedata()的调用是这样的：

```
movedata(source__seg,source__off,dest__seg,
dest__off,size);
```

不要以为可以用 movedata()函数将一个字符串直接移到视频缓冲区中。因为视频缓冲区中字符间是以属性字节分隔开的。通过 movedata()送往视频缓冲区的字符串一定要先填上属性字节。而把视频缓冲区的内容拷到另一地方则没有必要。这方面的一个实例是把虚屏中的内容拷贝到视频缓冲区中。

1.8 BIOS 中断

IBM PC 及其兼容机有很多有用的函数存贮在 ROM 中。这些函数称为基本输入输出服务或 BIOS，用于读键盘和控制屏幕输出的细节。

清屏、移动光标和在屏幕上写字符的程序在 BIOS 中都有。BIOS 的目的是使程序员不必了解硬件的细节。例如，移动光标使用的 BIOS 子程序在所有显示卡上都能运行。不管是单色、彩色显示卡，EGA 或 VGA。程序员只要用 BIOS 子程序而不必关心其内部是如何工作的。在新的计算机中修改了 BIOS 子程序以适应新的硬件结构。做出和旧版本兼容的新 BIOS 要花费很大的精力。从理论上讲，调用 BIOS 编写的程序应该与新的计算机兼容。BIOS 子程序通过 8088 中断来调用。每一个 BIOS 函数都和一个中断相对应。没有必要了解子程序的物理地址，然后调用该地址的函数，程序员只需知道中断号。新的 IBM 计算机及其兼容机中子程序的物理位置可以是任意地址，只要中断号相同就行了。

BIOS 子程序的实际地址存贮于中断向量表中。表中的每一单元包含相应子程序的地址。在软件执行中断前，必须指定 BIOS 子程序号。8088 在中断向量表中找到该子程序的地址，并调用这个子程序。

使用 BIOS 中断能保证与多任务系统以及将来的 DOS 版本兼容。例如，一个多任务、基于窗口的系统如 DESQ view 可以“窃用”(重定向)BIOS 中断并根据自己的需要处理中断。向屏幕上写字符的 BIOS 调用可以被修改，使输出重定向到 DESQ view 的虚屏上。严格按照 BIOS 写的应用程序因此可以为 DESQ view 窗口所控制，也能在其环境中操作。

1.9 执行一个中断

现在来看一下在C程序中调用BIOS中断的方法。

标准C语言中没有提供调用DOS中断的函数。幸好，大多数IBM-PC的C编译器都有执行这种任务的专用函数。在Turbo C和MSC中，这个函数是int86()，这是很多MS-DOS专用编译器提供的一个对应MS-DOS的扩充。

在调用一个BIOS程序之前，8088的通用寄存器一定要先赋值，说明要执行哪个功能和执行这个功能需要什么参数。方法是把类似于8088寄存器的数据结构传递给int86()，声明各寄存器中装入什么数据。在这里先详细讨论一下寄存器，以便更好理解其结构。

表1-4 BIOS寄存器

	8-bit(Byte)	
16-bit(word)	MSB High	LSB Low
ax	ah	al
bx	bh	bl
cx	ch	cl
dx	dh	dl

每个通用寄存器的字长是16位(一个字)，可以分成两个8位"字节"。左边的叫高字节，右边的叫低字节。高字节和低字节有时分别称为MSB(Most Significant Byte，最重要的字节)和LSB(Least Significant Byte，最不重要的字节)。比如，十六进制数22ffH的MSB是22，LSB是ff。

执行BIOS中断所要用的寄存器是ax，bx，cx，dx。如表1-4所示。它们可分为字节，寄存器ah表示ax的高字节，al表示低字节。

Turbo C和MSC都有用字和字节表示的相应的结构。这些结构定义在联合REGS中，Turbo C中的联合是这样定义的：

```
struct WORDREGS{
    unsigned int    ax,bx,cx,dx,si,di,cflag,flags;
};
struct BYTEREGS{
    unsigned char   al,ah,bl,bh,cl,ch,dl,dh;
};
union   REGS{
    struct    WORDREGS x;
    struct    BYTEREGS h;
};
```

在程序中可通过说明"union REGS regs"来定义联合变量regs。这种联合提供了对同一数据按字和字节两种方法访问的途径。在下面的例子中，会看到寄存器分别以字和字节的方式加载。示例：

1. 在程序中利用语句"regs. h. ah=5;"可以将ax寄存器的高字节装入常量5，也可以使用

语句“regs. h. ah＝(unsigned char)5;”,因为无符号字符和字节是等价的。

2. ax 的字结构可用 regs. x. ax 来指定,可以将寄存器按字来对待,通过语句“regs. x. cx＝5;”同时装入两个字节。

int86()函数需要三个参数:中断号,放在寄存器中的值和存中断返回寄存器值的地方。有些函数,比如读光标位置的函数,通过寄存器返回值,加载了各寄存器后,可调用该函数完成对应功能。

```
int86(0x10,&regs,&regs);
```

下面看看一个很简单的程序,通过 BIOS 调用移动光标。BIOS 子程序中负责移动光标的是中断为 10H 的视频中断。视频中断完成与屏幕有关的所有功能。具体哪个功能由存在 ah 寄存器中的值指定。光标移动功能是通过设置 ah＝2,然后调用视频中断来完成的。

移动光标还要知道光标要移到的行和列的位置,这是通过向 dh 装入行、dl 装入列来完成的,寄存器 bh 装入文本页号,一般为 0。

BIOS 对行和列是从 0 开始计数,而本书使用从 1 开始。左上角对 BIOS 来说是(0,0),而对于本书的系统则是(1,1)。

使用以 1 开始计数的系统,严格地说是个人的习惯。Turbo 系列软件中的窗口函数都使用以 1 开始计数的编号系统,本书的做法是对它们的模拟。函数 gotoxy()已完成了从本编号系统到 BIOS 所用系统之间的转换。

```
void gotoxy(int x,int y)
{
union REGS regs;
regs.h.ah=2;      /*ah=2,the cursor position function */
regs.h.dh=y-1;    /*row */
regs.h.dl=x-1;    /*column */
regs.h.bh=0;      /*assume page 0 */
int86(0x10,&regs,&regs);/*do interrupt */
}
```

注意视频中断 10H 是传给 int86()函数的第一个参数。

在下一章将开发一个更复杂的、基于窗口的该函数新版本。

BIOS 的功能的确很强。遗憾的是也比较慢。BIOS 不够灵活,不能在虚屏上工作,并且有些功能只能在活动文本页上起作用。大多数屏幕功能,比如清屏、滚屏、写屏,通过直接访问内存视频缓冲区来实现会更快,也更灵活。本书中大部分函数使用的是这种直接写屏方式。

与光标有关的功能是 BIOS 中不可缺少的一部分。本书中开发的移动光标和设置光标形状等子程序,都要调用 BIOS。如果不用 BIOS,程序就必须处理每个显示卡的硬件。在这种情况下,BIOS 的速度已经够了,而且我们还得益于 BIOS 调用的设备无关性。

1.10 ANSI 控制台驱动程序

美国国家标准局 ANSI 已为标准屏幕(控制台)的驱动程序设计了一整套规范。这就使软件可以在多数计算机系统上运行。

在以 MS-DOS 为操作系统的计算机中,通过在 config. sys 文件上加一行“DEVICE＝ANSI. SYS”,来完成 ANSI 驱动程序的安装。ANSI 驱动程序装好后,它就控制了所有的屏幕和键

盘操作。ANSI 指令，比如定位光标，是将一组特殊文本字符序列送给屏幕。有些 ANSI 驱动程序旁路了某些计算机的慢速 BIOS 子程序。利用 ANSI 的屏幕输出比不利用要快。

安装上 ANSI 驱动程序后要清屏，程序只需打印 Escape 字符（十六进制 1b），后跟“〔2J”。该清屏函数如下：

```
void cls(void)
{
put("\x1b〔2J");
}
```

使用这种驱动程序对程序员来说是很有吸引力的。它可以把程序员从试图熟悉很多种计算机软件的任务中解放出来。利用 ANSI 驱动程序写的程序可在支持它的任何系统上编译运行。

利用 ANSI 驱动程序也有些问题。如果程序是为 ANSI 系统写的，这就假设每个使用该软件的人在他的系统上安装了 ANSI。如果没有，就必须重新安装。一般来说，开发出的软件不应强迫用户过多地改变其计算机系统。如果一个程序使用不方便，通常被扔掉。在没有安装驱动程序的系统中使用基于 ANSI 的程序时会使软件不能正常运行，打印一堆莫名其妙的东西。

程序先检查 IBM-PC 机上是否安装了 ANSI 驱动程序。如果没装，打印出警告信息。这实际上也是很简单的。程序可以调用 BIOS 将光标移到指定位置（比如 1，1），然后程序通过 ANSI 命令将光标移到另一位置，并通过 BIOS 检查光标位置。如果光标能正确地移动，则说明 ANSI 驱动程序已经装入了。如果不能，就说明驱动程序没装。使用 ANSI 除了有装入驱动程序的负担外，ANSI 驱动程序也减少了程序可用的内存空间。

为与前面提到的目标保持一致，不依赖于任何特定的驱动程序或设备。本书中的软件不使用 ANSI 驱动程序。但如果安装了 ANSI 驱动程序，本书中的程序同样运行得很好。

总结

前面讨论了显示适配器、视频缓冲区和控制屏幕显示的技术。控制屏幕的两种方法，BIOS 中断和 ANSI 驱动程序，在处理各种不同的适配器方面是通用的。毫无疑问，控制屏幕的最快方法是直接访问视频缓冲区，这正是本书代码中使用的技术。

下一章要讨论直接写屏的方法。建立一个直接写屏函数而又不损失 BIOS 和 ANSI 方式的通用性是完全可能的。

第二章　窗口和屏幕函数

本章内容包括：

- 学习如何直接写视频缓冲区
- 讨论多种类型的窗口系统
- 生成一个用于创建、访问、移动和删除窗口的窗口/屏幕函数库

2.1　简　介

由于标准C语言不提供现成的屏幕及窗口函数，所以程序员需要自己编写这类函数，或者使用其他人编好的函数。最简单的办法莫过于使用某一C编译程序提供的同本书思想类似的特殊函数了。许多编译程序带有自己的窗口函数。从某种程序上讲选择一个特定的编译程序的确能使程序员的编程工作简单得多。

但应注意到，用某一编译程序的特殊函数编程存在一些缺点。如果该编译程序的生产商突然改变了屏幕函数库，程序员就得重写自己的代码。另外，程序员不能修改库函数，除非能得到库函数的源代码。有些源代码可以买到，但是不易理解。还存在的一个问题是，通常一个编译程序所提供的库函数并不能完全满足程序设计者的要求。

基于这种想法，就应考虑建立程序员自己的窗口环境，这样维护起来更方便，也更灵活。

2.1.1　什么是窗口

每个程序员对窗口的理解几乎都不一样。与其给出定义，不如讨论一下窗口的实际特点。

对于用户来说，一个窗口就是屏幕上的一个区域，屏幕的输出被限制在这个区域内。简单地说，一个窗口就是屏幕中的屏幕。窗口通常是个矩形区域，并用方框围起来以助于将它同屏幕其它部分分开。窗口可在其它正文上弹出，覆盖掉部分或全部正文，也可以与其它窗口并列显示。

也许最好的理解窗口的方法是想象一张办公桌。将计算机屏幕看作桌面，而窗口是一张张纸。纸可以放在桌面上并可移到任何地方。其它的纸张也可以放在桌面上任何地方，包括现有纸张的上面。纸可以挪到这堆纸的上方，或者插入到任何位置。可在任何一张纸上写上一些内容，而不管这张纸在何处。另外，随时可从桌上拿走任一张纸。

根据以上描述，窗口系统至少需要有完成以下功能的函数：

- 在物理屏幕的任何地方创建一个窗口（带边框或不带边框）；
- 向任意方向移动窗口；
- 写、清或卷动窗口；
- 在任何窗口间插入窗口；

此为试读，需要完整PDF请访问：www.ertongbook.com