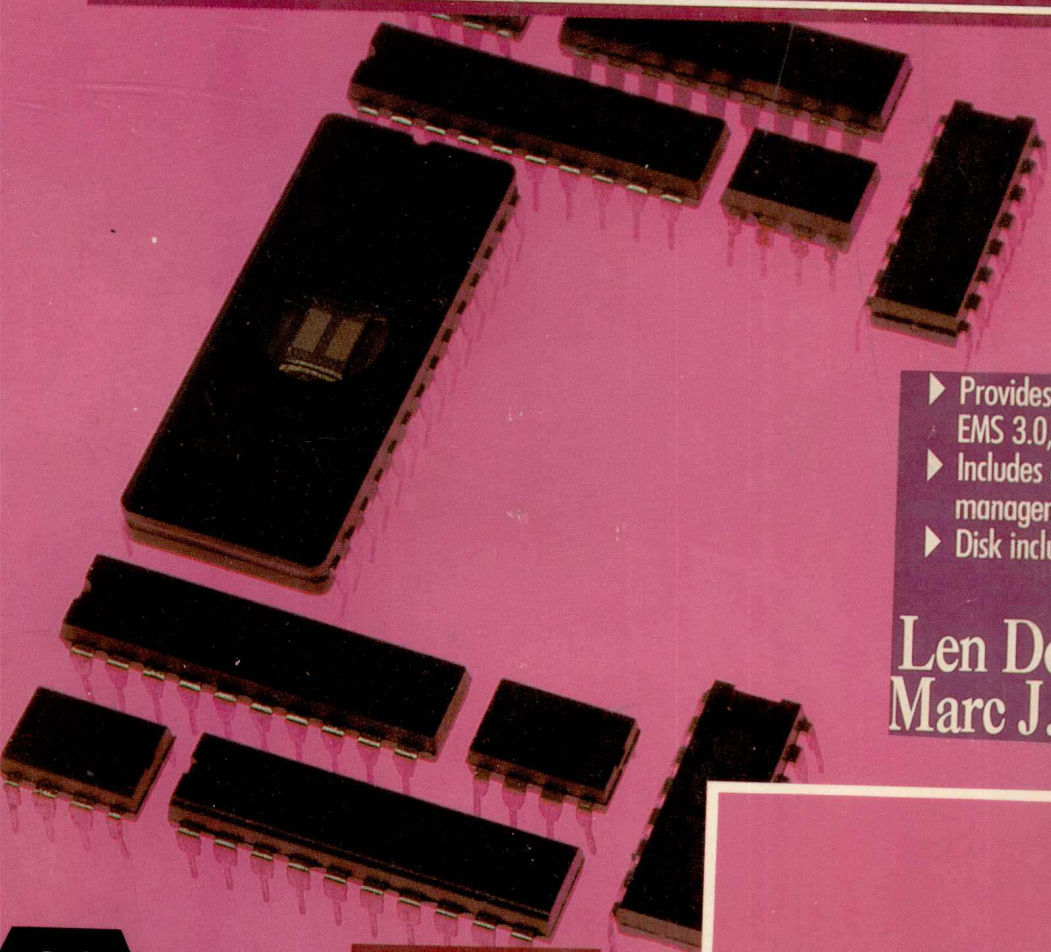
微机应用软件系列丛书

C 内存管理技术

C MEMORY MANAGEMENT TECHNIQUES



- Provides C interfaces to XMS and EMS 3.0, 3.2, & 4.0
- Includes a virtual memory management system
- Disk included

Len Dorfman and
Marc J. Neuberger

希望

THE Len Dorfman PRACTICAL

学苑出版社

C 内存管理技术

Len Dorfman
Marc J. Neuberger 著
孟文辉 译
熊可宜 校

学苑出版社
1994

(京)新登字 151 号

内 容 简 介

本书为 C 程序员提供了一些实用性很强的工具，使得 C 程序员可以有效地利用系统高区内存。书中提供了 EMS、XMS 和虚拟内存管理 VMM 的详细原理和工作过程，并带有丰富的源程序，许多功能函数的源代码和函数原型可由程序员直接调用和借鉴。

欲购本书的用户，请直接与北京 8721 信箱联系，邮编：100080，电话：2562329。

微机应用软件系列丛书

C 内存管理技术

编　　著：Len Dorfman　Marc J. Neuberger
翻　　译：孟文辉
审　　校：熊可宜
责任编辑：甄国宪
出版发行：学苑出版社　　**邮政编码**：100036
社　　址：北京市海淀区万寿路西街 11 号
印　　刷：北京市地质矿产局印刷厂印刷
开　　本：787×1092　1/16
印　　张：24.25　**字数**：566 千字
印　　数：1～5000 册
版　　次：1994 年 8 月北京第 1 版第 1 次
ISBN7－5077－0974－4/TP・33
本册定价：49.00 元

学苑版图书印、装错误可随时退换

用户请注意

欲购本书配套软盘的朋友，请按下列方法汇款：

单价：50.00 **元(含邮费)**

注：从银行电汇款的朋友请按下列帐号和收款单位汇款：

收款单位：北京希望电脑公司
开户银行及帐号：北京海淀工商行中关村城市信用社　　帐号：05079—08
注：要增值税发票的朋友，请仔细填下表。

购货单位	名称	纳税人登记号
	地址电话	开户银行及帐号

注：本次共订盘　　张　　应收款为￥(小写)：

注：用户填好此单后请连同此单、信汇单一并传真 01—2561057 或 01—2579874，收到传真后即发货。或邮寄 100080 北京海淀 8721 信箱资料部朱红收

联系电话：01—2562329，2541992

前　言

这是一本实用性很强的书，编写本书的目的是将一个实用的工具提供给C程序员，此工具可用来有效的在应用程序中使用高区内存(扩展内存、扩充内存以及硬盘)。本书提供了需要在80x86/8088实模式下进行虚拟内存管理的工具。

EMS和XMS内存管理函数为支持第六章中提供的虚拟内存管理的系统提供了基本的模块，VMM系统的全部源代码是与EMS和XMS接R函数一并提供给用户的，并且在磁盘上还提供给用户一个复杂的内存管理库。

读完本书后，用户就可以理解EMS，XMS和虚拟内存管理系统的工作原理，在理解的情况下，用户会有效的在应用程序中使用EMS、XMS和硬盘内存，用户的手头就会有一套使用起来很简单的虚拟动态内存分配函数集，它可以允许用户在程序中寻址多兆字节的存储器空间。

本书一开始就直接了当的讨论PC机的内存管理机制。在此之后，第二章介绍了内存区域(Microsoft这样称呼它)或者内存控制块(MCB—Memory Control Block)，并且提供了显示内存控制块内容的演示程序。

第三章讲解EMS3.0和3.2功能函数，本章中的演示程序目的是用来告诉用户如何在程序中使用众多的EMS3.0和3.2功能函数，每个EMS功能函数的源代码和函数原型，以及演示程序的源代码都提供给了用户。

在对如何使用EMS3.0和3.2功能函数有了一定的了解之后，第四章通过分析EMS4.0扩充内存管理标准继续EMS话题，像在第三章那样，本章提供给用户大量的EMS4.0功能函数的原型和源代码以及EMS4.0的演示程序。

第五章介绍了扩展内存规范(XMS—Extended Memory Specification)2.0，沿袭讲述EMS的章节中的做法，提供给用户每一功能函数的全部源代码以及XMS演示程序。

第六章提供了虚拟内存管理的源代码。此虚拟内存管理器可使用户在自己的应用程序中使用一个2M字节的内存，这个动态分配的内存可以被打开、写入、读出以及释放，像处理标准的动态分配的内存一样，VMM接口函数的原型和两个VMM演示程序的源代码一并提供给了用户，我们给本章中的源代码做了详细的注解，这样做全是为了方便用户理解VMM复杂的内部工作机制。

如何使用本书

为了更有效的使用本书，用户至少应具有C语言的初级知识，在书中的程序均是在Borland公司的Turbo C++和Microsoft的C6.0&7.0集成编辑环境下编写和测试的。

虽然本书中的汇编语言代码都是使用TASM和MASM5.1进行汇编的，但是为了使用本书中提供的内存管理功能函数，用户并不需了解汇编语言程序设计。

目　录

第一章 内存管理概述

虽然许多程序员认为 80x86 系列 CPU 的段—偏移结构是拜占庭式的，CPU 工作得却非常好，用一句话来说“它们是好样的！”

80x86 系列的处理机有两种操作方式：实模式在此模式下，CPU 工作像 8086/8088 那样工作，采用段—偏移量地址和保护模式（在此模式下，程序可以使用一个大的、线性的虚地址空间）。许多 MS—DOS 和 PC—DOS 程序均是工作在实模式下，相应地，本书也是用来解决实模式下的内存管理问题。

当用户在实模式下，而不是在 80286/80386/80486 保护模式下编程的时候，CPU 不能寻址位于 1M 字节以上的内存，1M 以下的内存分为两部分：从 0K 到 640K 的内存被称作低端内存，从 640K 到 1024K（1M）的内存被称作是高端内存区，而在 1M 字节以上的内存则被称作扩展内存区。

此表列出上面的范围及其名称

内存范围	名称
0～640K	低端内存区
640～1M	高端内存区
1M 以上	扩展内存区

1M 界限以下的内存空间可被 PC 的 BIOS（基本输入输出系统）、DOS（磁盘操作系统）、BIOS 和 DOS 表、中断向量表、设备驱动程序、显示缓冲区，以及 TSR 程序占用，剩下的提供给程序使用的是暂驻程序区（TPA）。

低端内存被分成许多内存块，每一个内存块均由一个内存控制块结构（MCB）来描述，这些内存块中包含有标识内存块大小和内存块是自由的还是被一个程序占用着这样的信息。

DOS 使用低端内存以满足各种需要，因它们中有 DOS、DOS 中断表、中断向量表、设备驱动程序和 TSR 程序，DOS5.0 以前版本的用户会发现供他们程序使用的暂驱程序区只有 450K 或更小。

虽然 DOS5.0 和一些商业性的内存管理实用程序通过允许用户将 TSR 程序，设备驱动程序以及 DOS 装入到高端内存的这种方法极大的改善了内存管理的处境。但是，TPA 还是受限制于微不足道的 640K 的内存空间。

为什么将 640K 的空间说成“微不足道”？好，当你需要分配 1M 字节的内存空间供一个程序使用时会发生什么？简单而言，640K 版本不可能完成工作，随着软件设计者们编写的程序变得更大、更复杂，迫切需要能够存取 640K 以上内存的方法。

一个解决办法就是，用磁盘存储做为 RAM 的一个代替物，虽然在这种情况下你能工作，但是使用硬盘做为 RAM 的一个替代物被证明是非常的麻烦的。磁盘存取的方式是顺序存取，这比 RAM 的存取速度慢许多，使用硬盘来进行超出 640K 限制的工作非常慢，于是，用户会体会到“时间就是金钱”。

但是硬件设计者们，努力向 640K 的界限挑战，并最终出现了扩充内存（EMS），EMS 提供

了一个灵活的页切换机制，在这种机制下，可以将成串的扩展内存映像到高端内存的一个位置，被称作扩充内存管理器(EMM)程序就是设计用来便利此基于硬件的页切换内存管理机制。

这个安排被证明是相当的有效，因为 Lotus ，Intel 和 Microsoft(LIM)一起创建了 EMS 标准，有一个 EMS 标准被证明是必要的，这种必要性在于编写运行良好的不会破坏其它程序中基于 EMS 的数据。虽然页切换机制不如基于 RAM 的线性地址机制那样优美，但是它确实比使用硬盘强。

80286 芯片对市场的冲击通过允许 CPU 寻址 1M 以上的内存地址而宣告了在保护模式下编制程序的开始，1M 以上的内存被公认为扩展内存，但是 DOS，并不是一个保护模式操作系统，而且它没有寻址 1M 字节以上内存的手段，一段时间内，在程序中使用扩展内存被认为是危险的，因为没有一个规范标准来协调扩展内存的使用。

在 80 年代末期，扩展内存(XMS)规范 2.0 版本出现了，这个 XMS 标准的出现以及 Microsoft 的 HIMEM.SYS XMS 驱动程序，允许程序员们以一种有序的方式寻址扩展内存。

DOS 内存管理的实用程序允许将低廉的扩展内存(XMS)当作扩充内存(EMS)来使用，这是一个非常经济的解决办法，许多新一代的 80386 母板上可以安装许多兆字节的 XMS 内存，内存管理器实用程序允许此扩展内存被分割成 EMS 和 XMS 的各种形式的结合。

在写本书的时候，工作在实模式的程序员有三种不同的打破 640K 内存界限进行动态内存分配的方法，它们是：

- 使用 EMS 页切换机制
- 使用扩展内存
- 使用硬盘驱动作为 RAM 的替代物

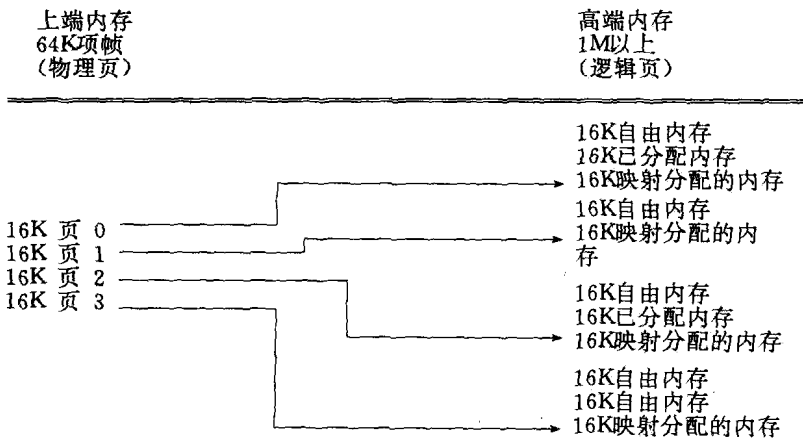
1.1 使用扩充内存进行动态内存分配

一个 EMM 程序允许在 RAM 中 640K 和 1024K 之间分配一个 64K 的称作页面区的段，这样 1M 字节以上的四个 16K RAM 块就可以通过调用 EMM 程序来映像到此页面区。

64K 的页面区是由四个 16K 的物理页构成的，这四个物理页是从被称作逻辑页的集合中取(实际上是映像)来的，让我们来形象化的说明 16K 逻辑页，16K 物理页，以及 64K 页面之间的关系，图 1－1 描述了了解 EMS 物理的和逻辑的页之间的关系。

EMS 页映像过程可被描述成以下几个步骤：

1. 使用 EMM 将四个不同的 16K 逻辑页映像到四个不同的 16K 物理页。
2. 从物理页中读或写
3. 将新的 16K 逻辑页映像到物理页
4. 返回第二步



1－1　EMS物理和逻辑页之间的关系

表1－1是与EMS3.0,3.2,4.0有关的功能函数的清单。

表1－1　EMS3.0,3.2和4.0的说明清单

版本	功能	子功能	描述
3.0	40h		取得EMS状态
3.0	41h		取得EMS反帧地址
3.0	42h		取得EMS 16K页的数目
3.0	43h		分配EMS句柄和16K页
3.0	44h		将EMS逻辑页映射到物理页
3.0	45h		自由的EMS句柄和逻辑页
3.0	46h		取得已安装的EMS版本
3.0	47h		存储EMS页映射
3.0	48h		恢复EMS页映射
3.0	49h		（保留以便将来使用）
3.0	4Ah		（保留以便将来使用）
3.0	4Bh		取得EMS句柄数
3.0	4Ch		取得EMS句柄页
3.0	4Dh		取得所有句柄的所有EMS页
3.2	4Eh	00h	存储EMS页映射
3.2	4Eh	01h	恢复EMS页映射
3.2	4Eh	02h	存储和恢复EMS页映射
3.2	4Eh	03h	取得EMS页映射信息大小
4.0	4Fh	00h	存储部分EMS页映射
4.0	4Fh	01h	恢复部分EMS页映射
4.0	4Fh	02h	取得部分EMS页映射信息的大小
4.0	50h	00h	按数目映射多个EMS页
4.0	50h	01h	按地址映射多个EMS页
4.0	51h		按句柄重定位EMS页
4.0	52h	00h	取得EMS句柄属性

4.0	52h	01h	设置EMS句柄属性
4.0	52h	02h	取得EMS属性容量
4.0	53h	00h	取得EMS句柄名
4.0	53h	01h	设置EMS属性容量
4.0	54h	00h	取得所有EMS句柄名
4.0	54h	01h	查找EMS句柄名
4.0	54h	02h	取得所有EMS句柄
4.0	55h	00h	按数目和JMP映射EMS页
4.0	55h	01h	按地址和JMP映射EMS页
4.0	56h	00h	按数目和CALL映射EMS页
4.0	56h	01h	按地址和CALL映射EMS页
4.0	56h	02h	取得EMS映射页和CALL的大小
4.0	57h	00h	移动内存区域
4.0	57h	01h	交换内存区域
4.0	58h	00h	取得可映射EMS页的地址
4.0	58h	01h	取得可映射EMS页的数目
4.0	59h	00h	取得硬件配置信息
4.0	59h	01h	取得原始16K页的数目
4.0	5Ah	00h	分配句柄和标准EMS页
4.0	5Ah	01h	分配句柄和原始16K页
4.0	5Bh	00h	取得其它EMS映射寄存器
4.0	5Bh	01h	设置其它EMS映射寄存器
4.0	5Ch		为热启动预备EMM
4.0	5Dh	00h	禁止EMM操作系统功能
4.0	5Dh	01h	允许EMM操作系统功能
4.0	5Dh	02h	释放EMS访问键

我们建议EMS做为程序员打破640内存界限进行动态内存分配的第一个选择。

1.2 使用扩展内存进行动态内存分配

我们将用XMS进行扩展内存使用作为程序员紧接在EMS方法后的第二个选择是有以下原因的：

1. EMS允许将页映像到可寻址内存中，而XMS要求数据向扩展内存传输或从扩展内存中读出，映像比起传输16K的数据来说快得多。

2. EMS提供了比XMS丰富的功能特性。

表1－2是XMS2.0规范的清单。

1.3 使用硬盘进行动态内存分配

在C中使用I/O文件相对来说是一个直接了当的工作。以基于DOS的文件I/O功能函数来创建扩展内存管理函数为在主机中没有可利用的EMS和XMS的情况下提供了一个有用

的使用扩展内存的备用方法。

我们认为磁盘动态内存分配管理操作是一个“备用”方法是因为它比基于 RAM 的方法慢得多。

1.4　小结

DOS 是一个实模式操作系统并且只能给程序提供最多 640K 的空间使用，用来供程序使用的可利用内存称作 TPA(暂驻程序区)

表 1－2　XMS2.0 说明表

版本	功能	描述
2.0	00h	取得 XMS 版本号
2.0	01h	申请高端内存
2.0	02h	释放高端内存
2.0	03h	全局允许 A20
2.0	04h	全局禁止 A20
2.0	05h	局部允许 A20
2.0	06h	局部禁止 A20
2.0	07h	查询 A20
2.0	08h	查询自由扩充内存
2.0	09h	分配扩充内存块
2.0	0Ah	自由扩充内存块
2.0	0Bh	移动扩充内存块
2.0	0Ch	锁定扩充内存块
2.0	0Dh	解锁扩充内存块
2.0	0Eh	取得句柄信息
2.0	0Fh	重新分配扩充内存块
2.0	10h	申请上端内存块
2.0	11h	释放上端内存块

许多程序对内存的需要远远不止 640K，于是产生出各种内存管理策略。

使用 EMS 进行动态内存分配被证明是可靠的和快速的，EMS 的可靠性支柱在于标准的出现，编程标准保证了程序的正确运行，EMS 做为我们进行动态内存分配的第一个选择是因为与 EMS3.0，3.2 和 4.0 规范相关的函数特性，以及映像的速度与传输数据速度的比较。

使用 XMS 进行动态内存分配同样被证明是快速的和可靠的，我们将 XMS 做为进行动态内存分配的仅次于使用 EMS 的第二个选择是因为 XMS2.0 规范的功能特性比 EMS 规范少。

鉴于以上原因，内存管理方法很自然地分成以下三个等级：

1. EMS(扩充内存)
2. XMS(扩展内存)
3. 硬盘

第二章 理解内存控制块

要全面地理解内存管理必须包括一个 DOS 如何在低端内存处理内存分配的描述，简而言之，DOS 将 640K 的低端内存分成一系列连续的内存块，这些内存块的大小是 16 字节内存段的倍数，这样做是很有意义的，因为段寄存器(DS,SS,CS,ES)是以段为边界而不是以单一的字节为边界。

DOS 创建了一个 16 字节大小的内存段，用来描述与其相连的内存块的特性，在本书中，这个一个段大小的(16 字节)内存块描述结构被称作 MCB—内存控制块，还有一些其它的称呼：内存区以及区头：

在用户 PC 的低端内存实际上有许多内存块用来描述内存被用户程序、设备驱动程序占用、以及自由内存，这么多个 MCB 的集合统一描述为一个 MCB 链。

在内存中，紧接着 MCB 的就是它所描述的内存块，让我们首先了解一下在 DOS4.0 以前可能使用的 MCB 的结构，注意我说“可能使用”，是因为在早期的 Microsoft 文档中，并未公开 MCB 的内容。事实上，定位第一个 MCB 的 DOS 调用仅是一个未公开的 DOS 调用。

DOS4.0 版本以前的 MCB。

```
typedef struct{
    char chain_status;
    unsigned int owner_psp;
    unsigned int size_paragraphs;
    char dummy[3];
    char reserved[8];
}MCB;
```

MCB 结构的第一项是一个用来标识内存链的状态的 8 位的值，若其值是 ASCII 字符"M"则是指在内存块链中还有许多内存块。若其值是 ASCII 字符 Z，则是指此 MCB 是 MCB 链中的最后一个。

MCB 结构中的第二项是占用本 MCB 所描述的内存块的程序的 PSP(程序段前缀)的段地址值，DOS 在装入一个程序时首先创建这个 256 字节的 PSP，在本章中，我们使用一程序的 PSP 中的以下信息。

PSP 段地址值用作程序 X 的标识号(ID)。

PSP:[0x2c]程序的环境段。

PSP:[0x80]程序的命令行长度

PSP:[0x81]命令行起始

在 MCB 结构中的第三个元素是一个 16 位的整数，它包含了 MCB 所描述的内存块所用

的大小，以段为最小单位：在以字节计　此内存块的大小的时候，只将此元素的值乘以 16 即可。当然，对写汇编语言的程序员来说，左移 4 位比乘以 16 更贴切

知道 MCB 所描述的内存块的长度之后，很容易找到链中的下一个内存块，用户所需要去做的是取得 MCB 的段地址，加上以段为单位的记录内存块大小的值，再加上 1(MCB 本身)，得出的结果就是在 MCB 链中下一个 MCB 的段地址。

程序 MAPMEM，MEM，TDMEM，以及 PROG 2－3.C(图 2－1 中提供的一个内存显示实用程序)管理得到第一个 MCB 的段地址，显示其中一些信息，再得到下一个 MCB，显示更多的信息，当且继续此过程直到到达 MCB 键的末尾，如果用户能够进到 MCB，PSP 以及程序的环境，可以报告有关内存块占用表的许多信息。

在 DOS4.0 以及以后的版本中 MCB 的第五项使用了一个 8 字节，此 8 字节中存放占用 MCB 描述的内存块的程序的文件名(不加后缀)，这样做减轻了寻找占用 MCB 的程序的文件名的任务，因为用户不需再从占用程序的环境段中得到文件名了。

下面是 DOS4.0 及以后版本中的 MCB 的结构

```
typedef struct{
    char chain_sttus;
    unsigned int owner_psp;
    unsigned int size_paragraphs;
    char dummy[3];
    char file_name[8]
}MCB;
```

在介绍了 MCB 及其结构之后，我们开始介绍程序。

2.1　内存管理预处理过程

图 2－1 是 GMCBP.ASM 的源代码清单，这个大的汇编例程样本使用一个未公开的 DOS 调用在 DX:AX 寄存器中返回一个指向内存链中的第一个 MCB 的远指针。我们并不喜欢使用未公开的 DOS 调用，但是我们还没有其它方法能得到第一个 MCB 的段地址。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * *  GMCBP.ASM                                                              * * *
;* * *                                                                        * * *
;* * *  VFP getMcbPointer(void)                                               * * *
;* * *                                                                        * * *
;* * *  Returns a far pointer to the first MCB                                * * *
;* * *                                                                        * * *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *

;-------------------------------------------------------------------------------
;  Large model, C language
;
```

```
        .Model  Large,C

;-------------------------------------------------------
; declare name public
;

        Public  getMcbPointer

;-------------------------------------------------------
; begin code segment
;

        .Code

;-------------------------------------------------------
; begin program
;

getMcbPointer PROC

;-------------------------------------------------------
; get list of lists via DOS int 21h
;

        mov     ah,52h
        int     21h

;-------------------------------------------------------
; prepare to return far pointer to
; first Mem Control Block via DX:AX
;

        mov     dx,word ptr es:[bx-2]
        sub     ax,ax

;-------------------------------------------------------
; return to caller from procedure
;

        ret

;-------------------------------------------------------
; end of getMcbPointer procedure
```

```
getMcbPointer     ENDP


  END

;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * *   End of getMcbPointer.asm                                          * * *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

图 2－1　GMCBP.ASM 的源代码清单

注意所有的汇编程序清单均已经使用 Microsoft 的 MASM5.1 和 Borland 的 TASM 汇编过，我们决定工作在大模式下，因为需要复杂的内存管理的众多程序很可能要在大模式下编写。

下面是我们用来编译链接汇编程序模块的 MASM 批处理文件。

```
masm /ml %1;
```

下面是 TASM 批处理文件

```
tasm /jMSC51 /ml %1
```

用户可以敲入下面的命令行来汇编图 2－1 的程序

```
a gmcbp
```

要将 GMCBP.OBJ 加到库中，我们仅需一个批处理文件，文件名为 ADDLIB.BAT，若用户是使用 Borland 的 c＋＋，可以用下面的 ADDLIB.BAT 版本。

```
tlib mcbl+%1
```

若是用 Microsoft，用户可使用下面的 ADDLIB.BAT 版本：

```
lib mcbl+%1;
```

图 2－2 是 GVECD.ASM 的源代码清单，此例程调用公开的中断 21h 的子功能 35h，返回一个指定的中断向量，此功能函数被内存显示程序用来确定是否有一些系统中断向量已转向指向程序中的内存块。

```
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
;* * * GVECP.ASM                                                          * * *
;* * *                                                                   * * *
;* * * VFP getVecPointer(BYTE)                                           * * *
;* * *                                                                   * * *
;* * * Returns a far pointer to interrup                                 * * *
;* * * vectors                                                           * * *
;* * *                                                                   * * *
;* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *


;-----------------------------------------------------------------------
; Large model, C language
;
```

```
        .Model  Large,C

;-----------------------------------------------------------
; declare name public
;

        Public  getVecPointer

;-----------------------------------------------------------
; begin code segment
;

        .Code

;-----------------------------------------------------------
; begin program
;

getVecPointer PROC vec_num:BYTE


;-----------------------------------------------------------
; vector number to al register
;

        mov     al,vec_num

;-----------------------------------------------------------
; get vector pointer via func 35h
;

    mov     ah,35h
    int     21h

;-----------------------------------------------------------
; prepare to return far pointer to
; to vector via DX:AX
;

        mov     dx,es
        mov     ax,bx

;-----------------------------------------------------------
```