目　录

第一章　MS-DOS 环境下高水平程序设计必备知识

1.1 MS-DOS 的装入过程

系统加电或重置，程序在地址 0FFFF0H 处开始执行。这是 8086 系列微处理器的特点，与 MS-DOS 无关。用这类处理器进行系统设计时应将地址 0FFFF0H 置于 ROM 区域中，并且其内容是一条跳转机器指令，使控制转给系统测试程序及 ROM 自举（bootstrap）程序。

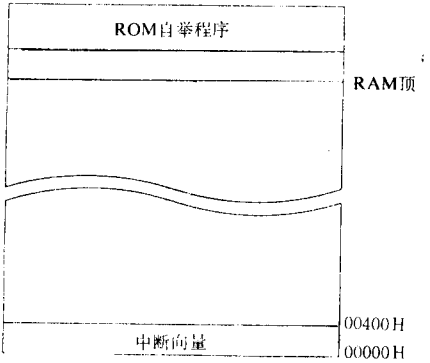

图 1-1　典型的用8086／8088的微计算机系统加电后，在 0FFFF0H 处开始执行，该处为一条 jump 指令将程序控制转给 ROM 自举程序。

ROM 自举程序从磁盘的第一个扇区（boot 扇区）中读入磁盘自举程序至内存中某一任意地址，然后将控制移交给它（此 boot 扇区中还包含一个与磁盘格式有关的表）。

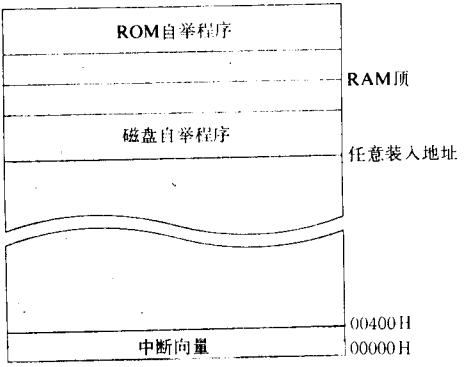

图 1-2 ROM自举程序将磁盘自举程序从系统盘第一个扇区中装入内存，然后将控制移交给它。

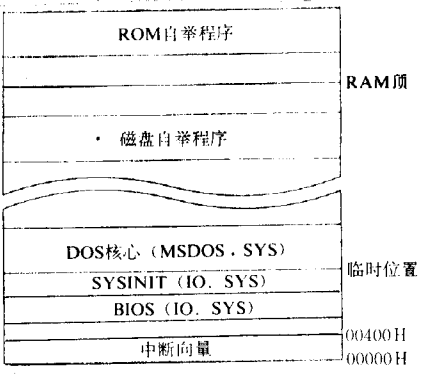

图 1-3 磁盘自举程序将文件IO.SYS读入内存 此文件包含MS-DOS，BIOS（驻留设备驱动程序）及 SYSINIT 模块．磁盘自举程序或 BIOS 再将 DOS 核心从 MSDOS.SYS 中读入内存．

磁盘自举程序查看该磁盘是否包含 MS-DOS 的拷贝，此时将由读入根目录的第一个扇区来确定前两个文件是否为 IO.SYS 及 MSDOS.SYS（顺序不能错）．如果这两个文件

不存在，它便提示操作人员更换磁盘并可打入任意键重新开始．如果查到了这两个系统文件，磁盘自举程序即将它们读入内存，控制将转到 IO.SYS 的初始进入点（有些 MS-DOS 实现磁盘自举程序时只将 IO.SYS 读入内存，再由 IO.SYS 负责去装入 MSDOS.SYS 文件）．

从磁盘装入的 IO.SYS 文件，实际上由两个单独的模块组成．第一个模块是 BIOS，其内容是互相链接的一组用于控制台、辅助端口、打印机及数据分块设备的驻留设备驱动程序，再加上一些在系统启动时间运行的与硬件有关的初始化程序．第二个模块称为 SYSINIT，由 Microsoft 公司提供，并由计算机制造商随 BIOS 链接到 IO.SYS 文件中．

SYSINIT 由制造商的 BIOS 初始化代码调用．在确定了系统中存在的相连（contiguous）内存的量之后，它再将自己重新安排到高内存区，然后将 DOS 核心 MSDOS.SYS 从原始装入位置移至内存最后位置，覆盖掉从原始的 SYSINIT 程序及包含在 IO.SYS 文件中的某些其他可扩展的初始化代码．

以下过程就是 SYSINIT 调用 MSDOS.SYS 中的初始化代码．DOS 核心初始化其内部的各种表和工作区，建立中断向量 20H 至 2FH，查找出链接后的驻留设备驱动程序清单，并为每个设备驱动程序调用初始化功能．这些驱动程序功能负责确定设备的状态并实现必需的硬件的初始化，同时为驱动程序所服务的外部硬件中断设置向量．

作为初始化序列的一部分，DOS 核心检查，由驻留块设备驱动程序返回的磁盘参数块确定系统所用的最大扇区尺寸，建立某些驱动器参数块，分配一个磁盘扇区缓冲区，然后显示出 MS-DOS 版权信息，将控制返回给 SYSINIT．

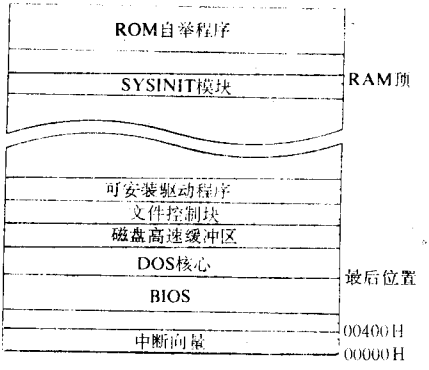

图 1-4　SYSINIT 将自身移至高内存区并重定位DOS核心,MSDOS.SYS 至最后地址,分配 MS-DOS 磁盘高速缓冲区及文件控制块区域.

至此，DOS 核心已被初始化，所有驻留的设备驱动程序已可使用．SYSINIT 可调用普通的 MS-DOS 文件服务功能去打开 CONFIG.SYS．此文件包含各种命令，使用户可配置 MS-DOS 环境．例如，用户可指定附加的硬件设备驱动程序、磁盘缓冲区数量、可同时打开文件的最大数量、命令处理程序的文件名等．

如果找到了 CONFIG.SYS，便将其装入内存进行处理:所有小写字符都转换为大写字符,一次一行地解释文件并作为命令来处理，为磁盘高速缓冲区及内部文件控制块或文件和记录系统功能用的句柄（Handle）分配内存，把 CONFIG.SYS 文件中给出的设备驱动程序顺序地装入内存，由调用其 init 模块进行初始化，并连接到设备驱动程序表中．每个驱动程序的 init 功能通知 SYSINIT 应为该驱动程序保留多少内存．

装入所有可安装的设备驱动程序后,SYSINIT关闭所有文件句柄并重新打开控制台（CON），打印机（PRN）、辅

助设备（AUX）作为标准输入、标准输出、标准出错、标准列表及标准辅助设备。它使用户安装的字符设备可覆盖掉BIOS中用于标准设备的驻留驱动程序。

最后，SYSINIT调用MS-DOS的EXEC功能以装入命令解释程序（系统中默认的命令解释程序是COMMAND.COM，但亦可通过CONFIG.SYS文件换成另一个解释程序）。一旦装入命令解释程序，便显示出提示信息并等待用户打入命令，此时SYSINIT模块被撤销。典型系统的MS-DOS启动过程的最后结果见图1-5。

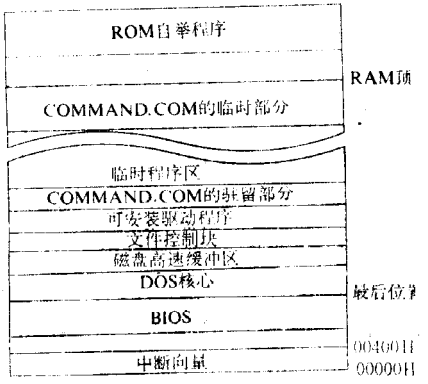

图1-5 COMMAND.COM驻留部分处于低内存地址区。临时部分包含批文件处理程序和内部命令，处于高内存区，该区可被运行在临时程序区中的外部命令和应用程序所覆盖。

1.2 两类基本的MS-DOS程序

在MS-DOS下运行的程序可分成两种基本类型。一类是COM程序，最大不超过64K，另一类是EXE程序，可大至可用内存的容量。按照Intel8086的术语，COM程序符合小模式（Small model），在此情况下所有段寄存器含

有同样的值，即代码与数据混合在一起。EXE 程序符合中或大模式（Medium，Large Model），段寄存器含有不同的值，即代码、数据、堆栈驻留在不同的段中，甚至可有多个代码和数据段。EXE 程序使用长调用及操作数据寄存器(DS) 来寻址。

驻留在磁盘上的 COM 类程序是绝对的内存映像，文件扩展名为.COM。该文件既无文件头也没有任何其他的内部标识信息。而.EXE 程序驻留在磁盘上，它是一种特殊的文件类型，带有一个文件头、一个重定位图、一个检查和以及 MS-DOS 使用的其他信息。

COM 及 EXE 程序进入内存执行所采用的是相同的机构：MS-DOS装入时用 EXEC 功能。EXEC 可用 COMMAND.COM 装入程序之文件名来调用，亦可由其他的命令解释程序或用户接口，或由以前用 EXEC 装入的另一个程序来调用。如果在临时程序区有足够的自由内存可用，EXEC 便分配一个内存块以容纳新的程序，在其基础上建立程序段前缀，然后将程序读入内存中 PSP 的直接上方。最后，EXEC 设置段寄存器和堆栈，并将控制传送给程序。调用 EXEC 时，可提供其他有关的信息，如命令 tail、文件控制块及环境块的地址。这些信息都将被传送给新程序。

COM 及 EXE 程序通常被称为"临时程序"。一个临时程序"拥有"已分配给它的内存块并在其执行过程中几乎完全获得对系统全部资源的控制。程序结束时，释放该内存块(因此称其为"临时"程序)，以供下一个装入的程序使用。

1.2.1 使用.COM 格式

COM 文件是内存中程序的准确映像。它是一类较小、

装入较快且较简单的程序．虽然 Microsoft 正在取消 COM 文件，但目前它仍是用汇编语言编写小型、快速、实用程序的较好格式．COM 文件亦可作为一个大型应用程序的前导程序．用户可键入一个小型COM文件的名字，然后由它来装入主程序．

但必须看到，COM 文件有两个重要限制:

(1)代码和初始化后的数据总量不可超过 64K，如果超过了此限制，将无法用 EXE2BIN 实用程序把 EXE 文件转换成 COM 文件．

(2)DOS 装入程序不能为 COM 文件执行段重定位．而对于 EXE 文件，该装入程序将使实际的运行时间段值赋予程序代码中要求的地址．此特点使程序员可将数据装入段寄存器，例如:

```
mov ds, dataseg
```

关于 COM 文件还有几点说明．其一是通常 COM 文件不可具有一个以上的段，其二是它不能包含堆栈段．下面的程序清单提供了生成一个较特殊的、多段的．COM 文件的基本框架 (shell)．对代码和数据使用不同的段是一个好办法．

此框架开始时对三个段:stackseg.codeseg 和 dataseg.加以说明．说明 stackseg 段仅用于连接程序以免在其找不到堆栈段时会显示出错信息，此段必须为空段且不被引用．

然后说明代码和数据段，迫使连接程序将代码段放在数据段前面，装入次序由首先引用的段次序来决定．在 COM 文件中，代码段必须位于文件的最前面 (堆栈段实际上在最前面，但它是空的)．

然后将代码段和数据段分配给称为 groupseg 的组段．考虑下面的数据引用指令:

```
page 50,130

;File:    COMFMT.ASM

;Format for a multisegment .COM File
;
;    To generate .COM file from TEST.ASM:
;         MASM TEST;
;         LINK TEST;
;         EXE2BIN TEST.EXE TEST.COM


                                        ;Declare segments to force
                                        ;loading order.

stackseg    segment stack               ;Placate linker.  This segment must
stackseg    ends                        ;remain empty.

codeseg     segment
codeseg     ends

dataseg     segment                     ;Place all data in this segment.
num_var'    db      ?
message     db      'hello from the data segment',10,13,'$'
buffer      label   byte
dataseg     ends

groupseg    group   codeseg,dataseg
            assume  cs:groupseg
            assume  ds:groupseg
            assume  es:groupseg

codeseg     segment

            org     5ch                 ;Template for Program Segment Prefix.
parm1       db      12 dup (?)          ;Parameters from command line.
            org     6ch
parm2       db      12 dup (?)
            org     80h
parmlen     db      ?                   ;Length of command line.
parmline    db      127 dup (?)         ;Full command line.

            org     100h
main        proc    near
                                            ;Main code.
            mov     ah, 09h                 ;Access data segment.
            mov     dx, offset groupseg:message
            int     21h                     ;Print message with DOS.

            mov     ah, 4ch                 ;Exit w/ errorlevel 0.
            mov     al, 0
            int     21h

main        endp

subr1       proc    near
                                            ;Subroutine code.
            ret
subr1       endp

codeseg     ends

            end     main
```

mov ah, num－var

正常情况下，象 num－var 这样的数据项被此类指令引用时，汇编程序从包含它的段的开始产生变量的偏移量，此情况下应是 0．但由于这是一个 COM 文件，数据段寄存器指向代码段的开始（所有段寄存器都一样），而不是指向数据段的开始，因此，要求有从代码段开始的 num－var 的偏移量．将数据段分配成一个组，则此类指令的偏移量自动从组中第一个段的开始产生，此时正好就是代码段．

虽然数据引用指令可自动地正确处理，但偏移量操作数必须使用组段前缀，如

mov dx, offset groupseg:message

无此前缀，所产生的偏移量将从数据段开始而不是从组开始．

文件中的下一项是实际的数据段 dataseg．注意，因为数据段出现在源文件代码段之前，故所有数据项均在文件头部列在一起．

要记住，在最后的可执行文件中，数据段在代码段之后，这种安排比将数据放在代码段前部，然后跳过它的那种常用技术好．因为如需用一个大的未初始化的缓冲区，此缓冲区可作为一个标号（如清单中的 buffer）置于数据段的尾部，而无保留空间（注意，在 COM 文件尾部前的所有用户内存在装入时均自动分配给了该程序）．如果此缓冲区在代码段开始处进行了说明，将保留下空间，但增加了磁盘上 COM 文件的长度（增加部分为缓冲区容量）．记住，堆栈指针初始化时指向 64K 程序段的顶部，如果会影响缓冲区，便应重定位该堆栈．

清单的最后一部分是代码段．在 org 100h 前面出现的

标号用于访问程序段前缀。当一个 EXE 文件转换成 COM 文件时，删除开始的 100h 个字节（在由 end 语句指定程序进入点之前的所有代码或数据全被删除），故可执行文件中出现的第一个字节将是主程序的第一条指令。

下面的批处理文件 A2C.BAT 自动生成一个 COM 文件

```
if exist %1 goto end
masm %1.asm
if not error level 1 link %1;
if not errorlevel 1 exe2bin %1.exe %1.com
del %1.obj
del %1.exe
:end
```

欲处理 TEST.ASM,可打入:

```
A2C TEST
```

1.2.2 EXE 程序结构

EXE文件的格式比COM文件的更灵活,适合于超过64K的程序,更易与将来的操作系统环境兼容.而且只有EXE文件方可用 Microsoft Codeview 检错程序进行符号 debug。下面的清单提供了一个样板，可用于从汇编语言生成 EXE 文件，并且适合于作符号 debug。

该清单说明EXE格式与 COM 格式有相当的三个段,但也有一些重要差别:

• 与 COM 文件不同，这里堆栈段是有用的，并且为堆栈保留空间。装入程序自动初始化堆栈段寄存器以指向此段的基底，而堆栈指针指向顶部。

• 在 EXE 文件中，数据段和附加段寄存器初始化后指

```
page 50,130

;File:    EXEFMT.ASM

;Format for generating an .EXE File
;
;    To generate .EXE file from EXEFMT.ASM:
;         MASM EXEFMT;
;         LINK EXEFMT;

public     message                    ;Declare all symbols 'public'
public     main                       ;for CodeView debugger.

codeseg    segment 'CODE'             ;Declare code segment to force it
codeseg    ends                       ;to load first.

dataseg    segment 'DATA'             ;Place all data in this segment.
message    db      'hello from an .EXE file',10,13,'$'
dataseg    ends

stackseg   segment stack 'STACK'      ;Set up 1K stack.
           db      128 dup ('stack   ')
stackseg   ends

           assume  cs:codeseg
           assume  ds:dataseg
           assume  es:dataseg
           assume  ss:stackseg

codeseg    segment 'CODE'

main       proc    far

           mov     ax, dataseg              ;Set up segment registers.
           mov     ds, ax
           mov     es, ax

           mov     ah, 09h                  ;Access data segment.
           mov     dx, offset message
           int     21h                      ;Print message with DOS.

           mov     ah, 4ch                  ;Exit w/ errorlevel 0.
           mov     al, 0
           int     21h

main       endp

codeseg    ends

           end     main
```

向一个单独的段 dataseg。由于装入程序为 EXE 文件执行重定位，故将 dataseg 分配给这些寄存器是可能的。又因为数据段寄存器指向一个单独的段，故不必再去说明一个段组，并有可能使用整整 64K 的数据和整整 64K 的代码（如果还说明了其他段，则还可用得更多）。

• 数据指令与 COM 文件中的相同，但偏移量算子不再使用组前缀。

• 程序的大小可任意。如果代码段超过了 64K，只需

再生成一个代码段（在段间用长调用分支）即可。类似地可说明另外的数据段，但数据段寄存器必须始终包含当前被访问段的基地址。

EXE 程序总是由 MS-DOS 装入程序装入内存中，直接处于程序段前缀上方，虽然代码、数据及堆栈段的次序是可以改变的。

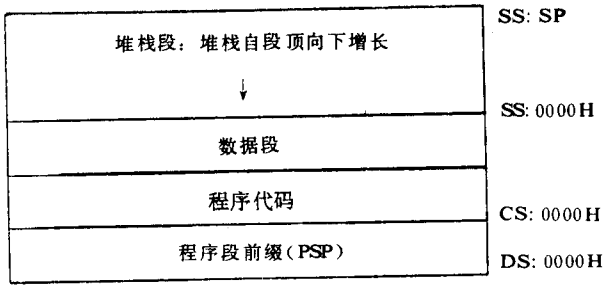

图 1-6 典型 EXE 类型程序刚装入后的内存映象。EXE 文件的内容被重定位，并被装入内存中 PSP 上方。代码、数据和堆栈驻留在不同的内存中，次序不必与图中相同。程序进入点可在代码段中任意位置，由主模块中 END 语句指定。程序接受控制后，DS 和 ES 寄存器指向PSP，程序通常保存此值，然后重置 DS，ES 寄存器以指向数据区。

EXE 文件有一个文件头（header），或称为控制信息块，它具有带特征的格式。此文件头的大小随在装入时需重定位的程序指令数而变，但总是 512 字节的倍数。

MS-DOS 将控制传送给程序之前，代码段（CS）寄存器和指令指针（IP）寄存器的初始值根据 EXE 文件头中进入点信息及程序的装入地址计算后得到。此信息是由源代码中的 END 语句推算出的。数据段（DS）及附加段（ES）