微型计算机软件资料汇编

第 一 册

机械工业部计算中心
合肥工业大学微型机应用研究所 编译

GRAPHICS

muLISP

rDBMS

CP/M

机械工业部仪表局情报室
《仪表工业》编辑部

微型计算机软件资料汇编（第一册）

机械工业部计算中心
合肥工业大学微型机应用研究所 编译

*

机械工业部仪表局情报室《仪表工业》编辑部编辑出版
北京印刷二厂排版印刷
北京市建华书刊发行社发行

*

1984年12月北京
代号：8303

内 容 提 要

《微型机软件资料汇编》第一册，包括两篇文章。

“微型机操作系统CP/M2.2使用手册”对CP/M操作系统(2.2版本)的功能作了详细的叙述。CP/M是Digital Research公司为微型计算机设计的一种操作系统，该系统可靠性高、适应性强，便于修改和移植，且支持软件丰富，是目前使用最广泛的微型机操作系统。为方便使用CP/M1.4版本的用户，本文对CP/M1.4版本的功能仍保留其完整的描述，对功能扩充及系统修改均进行了详尽的说明，并附有程序样本及实例。对CP/M2.2在Heath/Zenith 8位计算机系统的实现（即CP/M2.2.03）也作了专门介绍。本文可作为CP/M操作系统用户的使用手册，也可作为学习微型机操作系统的参考资料。

“微型机关系数据库CONDOR SERIES/20 rDBMS使用手册”描述了一个小型关系式数据库系统的外部模式及使用规则。这个数据库管理系统是CONDOR计算机公司的一个实验性开发系统，已在Z-80机上提供给用户使用。本文着重叙述在CP/M操作系统下，使用CONDOR SERIES/20 rDBMS开发应用数据库的过程，包括建立数据库以及数据库的查询、组合、联接、重构等操作，并附有大量实例，使读者对于小型关系数据库有一个具体的了解。本文可作为微型机用户使用该数据库系统的参考手册，对于数据库专业人员、大专院校有关专业学生及企事业管理人员也有一定的参考价值。

微型计算机软件资料汇编

第 一 册

机械工业部计算中心
合肥工业大学微型机应用研究所 编译

机械工业部仪表局情报室
《仪表工业》编辑部

编 译 出 版 说 明

本资料汇编收集了近期从国外引进的微型计算机软件，包括CP/M操作系统及其支持程序、高级程序设计语言、数据库管理系统和应用软件包，可以在Zilog Z80系列、Intel 8080系列微型机上使用，并已在H/Z89微型机上验证。

收集在资料汇编中的有：

微型机操作系统 CP/M2.2；

小型关系数据库管理系统 CONDOR SERIES/20；

高级语言 COBOL-80，PASCAL/MT+，FORTRAN-80
MBASIC，PL/I-80，C，muLISP；

编辑和字处理系统；

分类/合并程序；

库存管理程序；

图形软件包；

远程终端仿真程序等

这些资料大部分以使用手册形式提供，可作为微型机用户手册，也可供计算机系统软件和应用软件人员以及大专院校有关专业师生学习参考。

本资料汇编由机械工业部仪器仪表工业局组织，机械工业部计算中心和合肥工业大学微型机应用研究所编译，刘运基、康兴钨审校，并请旅美学者赵鉴芳教授指导审定。在编译出版过程中，得到许多同志的大力协助，谨在此表示谢意。

由于编译者水平所限，难免有错漏之处，敬请读者指正。

本资料汇编共分六册，由机械工业部仪表局情报室《仪表工业》编辑部陆续出版。

一九八三年五月

目　录

微型机操作系统 CP/M 2.2使用手册

微型机关系数据库 CONDOR SERIES/20 iDBMS 使用手册

微型机操作系统CP/M2.2使用手册

第一章　CP/M2.2使用指南

第一节　功能概述

CP/M2.0是高性能的单用户操作系统，它使用表驱动技术，允许进行现场重配置以满足各种不同的磁盘容量。CP/M2.0主要特性包括可指定16个磁盘驱动器，每个容量可达8MB；对于任一个特定文件，其容量最大可达整个驱动器容量；在将来版本中，容量还可扩充到32MB。目录表大小可现场决定，可包含适量的登记项，且每个文件可有选择地标记为只读文件或系统文件。CP/M2.0用户可以按不同的用户号在物理上互相独立。并且可以将文件从一个用户区拷贝到另一个用户区。CP/M2.0提供了强有力的相对记录随机存取功能，可对任一个8MB文件的65536个记录直接存取。

CP/M2.0所有关于磁盘的信息均放在驻存于BIOS的磁盘参数块中，这部分既可以手工编码，也可以通过使用CP/M2.0的磁盘定义宏指令库自动产生。终端用户只需指定最大联机磁盘数，开始及结束扇区号，数据单元大小，最大逻辑磁盘区，目录表大小及保留磁道值。宏指令用这些信息，产生相应的表格及表格相关项，供CP/M2.0运行时使用；同时还提供解块信息，用于大小为128字节基本数据单元倍数的扇区组块和解块。“CP/M2.2修改指南”一章提供通用的子程序，这些子程序使用解块信息，使得用户在用到较大扇区尺寸时开销较小。这些子程序和表格驱动数据存取算法一起使用，使CP/M2.0成为真正的通用数据管理系统。

文件扩充是通过提供多达512个逻辑文件区实现的，其中每个逻辑区域包含16K字节的数据。但是，CP/M2.0的组织使每个物理磁盘区（相应于一个目录表登记项）可寻址多达128K字节数据，因此，既充分利用了目录表空间，又可保持和以往版本的兼容性。

CP/M2.0具有随机存取功能，可直接访问一个8MB文件的任一个记录。使用CP/M独特的数据组织，数据块只在实际需要时才被定位，且传送到记录位置只需很少的查找时间。顺序文件存取对整个8M字节范围均保持与早期版本向上兼容，但随机存取兼容性只限于512K字节文件。由于CP/M2.0具有更简单、更快的随机存取功能，使应用程序员更有充分信心修改自己的程序以充分利用CP/M2.0提供的各种功能。

相应于文件系统功能的扩充，CP/M2.0有些模块也作了一些改进。STAT和PIP说明文件属性及用户区，而CCP提供登录(Login)功能，将一个用户区改变为另一个用户区。CCP还以更方便的形式使目录表显示规格化，并为CRT和硬拷贝设备提供增强的行编辑功能。

下面各节说明CP/M1.4和CP/M2.0之间的差别，读者应该熟悉CP/M1.4或者参阅CP/M1.4手册。关于I/O系统修改请参考“CP/M2.2修改指南”一章。

第二节　用户接口

控制台行处理增加了三种新的控制字符（下表中带*号），表中“ctl”符表示同时按

control 键。

rub/del	取消和送回最后一个字符
ctl-C	在一行开始时重新启动
ctl-E	物理行结束
ctl-H	后移一个字符的位置 *
ctl-J	（换行）结束当前输入 *
ctl-M	（回车）结束输入
ctl-R	在一新行后再打当前行
ctl-U	在一新行后移去当前行
ctl-X	退到当前行的开始 *

特别要注意到 ctl-H 产生适当的退格重写功能（ctl-H 可只通过改变一个字节，从内部变成另一个字符，如删除）。而且，行编辑程序记下当前提示符列位置以便操作员可以适当地在 ctl-U、ctl-R 或 ctl-X 命令之后调整数据输入。

第三节　控制台命令处理程序（CCP）接口

在 CCP 级上，CP/M1.4与 CP/M2.0 有四种功能上的差别。CCP 现在可以跨屏幕显示目录信息（每行四个元素），可以使用USER 命令维护同一目录表中不同的文件，且改变了“ERA *.* ”和“SAVE”命令的动作。变化了的 DIR 格式是自解释性的，而USER 命令取形如：

USER n

这里 n 是 0 ～15内的整数。在冷启动时，用户被自动地登记到 0 号用户区，它与标准的 CP/M1.4目录兼容。操作员可以在任何时间发出USER 命令将用户号改变到同一目录表中另一逻辑域。当改变到另一用户号时，寻址一个用户号时记入的驱动器被自动地激活。因为用户号只是联机磁盘上存取特定目录表项的前缀。

激活的用户号一直维持到被下一个USER 命令改变或直到冷启动操作再次假定用户为 0 号时为止。

用户号现在标识各个目录表项，因此ERA * · * 命令具有不同效果。在 1.4 版本中，这一命令可用来删除一个具有无用信息的目录，这个无用目录可能是因为使用了在另一操作系统下的磁盘而产生（这应是严格禁止的）。然而在 2.0版本中，ERA * · * 命令只影响到当前用户号。因此有必要写一简单的实用程序去删除无用盘（该程序在整个磁盘上写上十六进制形式E5）。

在1.4版本中的 SAVE 命令只允许一次存储器存储操作，因为在磁盘区边界变化之后的目录操作，有可能破坏内存映象。而 2.0 版本在磁盘写过程之后不在用户数据区进行目录操作，因此，SAVE 操作可使用多次而不改变内存映象。

第四节　STAT 扩充

STAT 程序作了一些功能上的扩充，允许显示磁盘参数、用户号以及设置文件指示符。

命令

STAT VAL:

概括列出有效的状态命令，产生如下输出：

```
Temp R/O DISK:d:=R/O
Set Indicator:d:filename.typ $R/O $R/W $SYS $DIR
Disk Status:DSK:d:DSK:
User Status:USR:
Iobyte Assign:
```

(列出可能的赋值)

命令

STAT d:filename.typ $S

(这里d:是任选的驱动器名，filename.typ是单义的或有歧义的文件名) 产生的输出形式是

Size	Recs	Bytes	Ext	Acc	
48	48	6K	1	R/O	A:ED.COM
55	55	12K	1	R/O	(A:PIP.COM)
65536	128	2K	1	R/W	A:X.DAT

这里$S参数引起Size字段显示 (否则就跳过Size段，但显示出其余的段)。Size字段列出了虚文件记录的大小，而Recs字段综合了每一个磁盘区内虚文件记录的数量总和。对于顺序组织的文件，Size和Recs段相同。Bytes字段列出了实际相应的文件所分配的字节数。最小存储单元在配置时决定，因此对顺序文件，字节的数目相应于记录数加上最后存储块中剩余的没有用到的空间。随机存取文件只在写入时给出数据区，因此Bytes字段仅包括精确的盘位图。在随机存取情况下，Size字段给出逻辑文件结束记录位置并且Recs字段记录每个磁盘区逻辑记录的数目 (但每个磁盘区可能包含未分配的“空白区”，已加到记录数中)。Ext字段计算分配给文件的16K逻辑区域的数目。与1.4版本不同，单一目录表项可以直接寻址128K字节 (8个逻辑磁盘区)，这取决于存储单元的大小。在特殊情况下，实际可有256K字节被物理磁盘区直接寻址，因此Ext不必按照给定文件的目录表项目的数目进行计算。

Acc字段给出只读或读写存取方式，它可以利用下列命令来改变。同样地，PIP.COM文件名的括号表示设置了系统标志符，这样在DIR命令时不会被列出。四种命令形式：

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

置或重置各种永久的文件标志。R/O标志将文件置成只读状态，直到被下一个STAT命令改变。R/O状态记录在文件的目录中以便在冷启动干预时仍保留只读；R/W标志置文件成永久的读写状态；SYS标志将系统标志符连接至该文件；而DIR命令取消系统标志符。“filename.typ”是任选的，但无论在哪种情况下，在改变文件属性时，则该文件名在控制台上列出。由d:标志的驱动器是任选的。

当文件标志是只读后，以后若企图删除或写入文件将会产生终端 BDOS 信息

Bdos Err on d：File R/O

然后 BDOS 在执行下一个热启动之前将等待控制台输入（按 Return 键即可继续）。命令形式：

STAT d：DSK：

列出以 d：为名的磁盘驱动器特性，d：在 A：，B：，…，P：范围内。驱动器特性以下面格式列出：

d：	驱动器特性
65536：	128K字节记录容量
8192：	K字节（Kilobyte）驱动器容量
128：	32字节目录表项
0：	被校验的目录表项
1024：	记录/每磁盘区域
128：	记录/每块
58：	扇区/每磁道
2：	保留磁道

这里 d：是所选的驱动器，后面是总的记录容量（65536是一个 8MB 字节驱动器），及以 K 字节为单位列出的总容量。接着列出目录大小，后面是被校验项。对可移动存储介质，被校验项数目通常与目录表大小相等，这是由于这个机构用来在 CP/M运行期间没有热启动干预时检测变化了的介质。对于固定介质，被校验项数目通常是零，这是由于若没有冷启动或热启动，介质是不被改变的。每个磁盘区的记录数决定每个目录表项的寻址能力（在上例中为 1024×128字节或128K 字节）。每块的记录数表示基本单元大小（在上例中为128记录/块×128 字节/记录或 16K 字节/块）。接着列出每个磁道物理扇区数和保留的磁道数。对于共享同一物理磁盘的逻辑驱动器，保留磁道数可以相当大，这是由于这个机构用来跳过定位到其他逻辑磁盘的较低号的磁盘区。命令形如：

STAT DSK：

对所有当前联机驱动器产生一个驱动器特性表。最后的 STAT 命令形式是：

STAT USR：

它列出在当前定址的磁盘上具有文件的用户号。显示格式为：

```
Active User：0
Active Files：0 1 3
```

这里第一行列出当前寻址到的用户号，它由最后一次 CCP USER 命令所设置。接着列出从当前目录中扫描到的用户号。在上面情况下活动的用户号是 0（在冷启动时缺省），有三个用户号在当前盘上具有活动文件。操作员可以利用记入 USER1、USER2 或 USER 3 命令，在 CCP 级连同使用 DIR 命令来检查其他用户号的目录。

第五节 PIP 扩充

PIP 提供三种新功能，它们可以说明 CP/M2.0的特性。这三种功能取文件参数的形式，

在文件名后用一方括号括起来。这些命令是：

Gn　从用户号 n 取文件（n 的范围是从 0～15）

W　写到只读文件上（无控制台询问）

R　读系统文件

G 命令允许一用户区从另一用户区接收数据。假定操作员已在 CCP 级执行了 USER 4 命令。

PIP语句：PIP　X.Y=X.Y[G_2]

从用户号 2 读文件 X.Y 至用户区号 4

命令：PIP A:=A:*.*[G_2]

将用户号 2 的 A 驱动器目录中的所有文件拷贝到当前登录的用户号的 A 驱动器目录中去。注意，为保证文件安全，不能将文件拷贝到不同的区，但USER 命令当前寻址的区域除外。

同样要注意，PIP 程序本身使用 SAVE 命令先拷贝到用户区中（以便以后的文件能被拷贝）。以下给出的一系列操作把 PIP 从一个用户区传送到下一个用户区：

```
USER   0    登录用户 0
DDT  PIP.COM   装 PIP 到内存
 (注意 PIP 尺寸 s)
G 0    返回 CCP
USER   3    登录用户3
SAVE  s   PIP.COM
```

这里 s 是 PIP 所占的内存页面的整数值（256字节段）。s 值可以在 PIP.COM 由 DDT 装入时通过参照 NEXT 显示的值确定。例如，如果下一个有效地址是1D00，则 PIP.COM 需十六进制1C 页（或 1×16+12=28页），因此在以后的SAVE 中 s 值是 28。PIP 以这种方法被拷贝之后，就可以通过通常的 PIP 传送，被拷贝到另一张属于同一用户号的盘上去。

在通常运行情况下，PIP 不在置成永久只读状态的文件上进行写操作。如果企图这样做，则提示：

DESTINATION FILE IS R/O, DELETE (Y/N) ?

如果回答 Y，则文件被重写。否则回答 * *NOT DELETED* *，跳过文件传送，PIP继续下面的运行。为避免在只读文件上重写操作时提示和响应，命令行可以包括 W 参数，如下所示：

PIP A:=B:*.COM[W]

它从驱动器 B 将所有非系统文件拷贝到驱动器 A，并且在处理中进行所有只读文件的重写操作。如果运行包括数个连接文件，只需在所列文件的最后一个中包括W参数。如下所示：

PIP A.DAT=B.DAT, F:NEW.DAT, G:OLD.DAT[W]

具有系统属性的文件，如果 PIP 命令中使用 R 参数，则也能被 PIP 传递，否则系统文件是不能被认识的。以命令行

PIP ED.COM=B:ED.COM[R]

为例，它可以从 B 驱动器读文件 ED.COM，尽管它已被置成只读和系统文件。如果系统文件属性存在的话，则被拷贝。

注意，只有当文件不超过 1 兆字节，未曾置文件属性且文件由用户 0 建立时，才能保持与当前 CP/M 版本的向下兼容性。如果需要与1.4的非标准(例如双密度)版本兼容，则当构

造内部磁盘参数块时（参考“CP/M2.2修改指南”及描述 BIOS 差别的第十 节），有必要选择1.4兼容方式。

第六节 ED 扩充

CP/M 标准文本编辑程序在2.0版本中提供了一些新的功能。经验已经表明，大 多数 操作员都使用 ED 的相关行编号特性，因此编辑程序将 “V” (Verify Line) 任选 项 置 成初值。当然操作员可通过打 “-V” 命令仃止行编号。如果用户不熟悉 ED 的行编号方式，可参考第三章附录。

ED 同样考虑文件属性。如果操作员试图编辑只读文件，则在控制台上显示

* * FILE IS READ/ONLY * *

信息。只读文件可以被装入和检查，但不能以任何方式改变。正常情况下，操作员可结束编辑过程，并使用 STAT 把文件属性改成读写。如果编辑的文件具有系统属性， 则出现信 息

“SYSTEM” FILE NOT ACCESSIBLE

编辑过程中止。如果需要，可再使用 STAT 命令改变其属性。

最后，插入方式（i）命令允许 CRT 行编辑功能，如上面第二节所述。

第七节 XSUB 功能

CP/M2.0版本增加了实用程序 XSUB，它将 SUBMIT 功能扩充到可以对程序及控制台命令处理程序进行行输入。XSUB 命令包括在用户提交文件的第一行中，且当执行时，直接在 CCP 下自重定位。所有后面的提交命令行均由 XSUB 处理，以便读缓冲的 控 制 台 输入（BDOS 功能10）的程序直接地从提交文件接收其收入。例如，文件 SAVER.SUB 可 包含以下提交行：

```
XSUB
DDT
I$1.HEX
R
G0
SAVE 1 $2.COM
```

后面带一个 SUBMIT 命令：

SUBMIT SAVER X Y

它将命令流中的 $1 及 $2 分别换成 X 和 Y。XSUB 程序装 入 然 后 进入 DDT，送命令 行 “I$1.HEX”、“R” 和 “G0” 后返回CCP。最 后 命 令 “SAVE 1 Y.COM” 由 CCP 处理。

XSUB 程序驻留在内存中，并在每次热启动时显示信息

(xsub active)

来表示其存在。以后的提交命令流不需要 XSUB，除非发生冷启动干预。注意，如果要同时运行 XSUB 及 DESPOOL，则 XSUB 必须在 DESPOOL 之后被装入。

第八节 BDOS 接口规则

CP/M2.0系统调用方式与早期版本完全相同，调用地址0005H，功能号在寄存器C，信息地址在寄存器对DE。单字节值在寄存器A中返回，双字节在HL（考虑到兼容性，在所有情况下返回时，寄存器A=L，B=H)。下面列出CP/M2.0的调用，带星号的功能或是新增的或是由1.4版本修改的。注意，超出功能号范围返回零值。

0	系统复位
1	控制台输入
2	控制台输出
3	读入机输入
4	穿孔机输出
5	打印机输出
6*	直接控制台I/O
7	取I/O字节
8	置I/O字节
9	打印字符串
10*	读控制台缓冲区
11	检测控制台状态
12*	返回版本号
13	磁盘系统复位
14	选择磁盘
15*	打开文件
16	关闭文件
17*	查找第一个文件目录项
18*	查找下一个文件目录项
19*	删除文件
20	顺序读
21	顺序写
22*	建立文件
23*	重新命名文件
24*	返回登录向量
25	返回当前磁盘
26	置DMA地址
27	取地址（ALLOC）
28*	磁盘写保护
29*	取只读向量
30*	置文件属性
31*	取地址（磁盘参数）

32* 置或取用户码

33* 随机读

34* 随机写

35* 计算文件大小

36* 置随机记录

（功能28、29及32在应用程序中应避免使用以维持同MP/M 的向上兼容性）。下面描述新的或修改的功能。关于系统功能的详细描述，请参阅第六章“CP/M2.2接口指南”。

功能6： 直接控制台I/O

直接控制台 I/O 在 CP/M2.0下支持，用在有必要避免 BDOS 控制台 I/O 的地方。当前通过 BIOS 执行直接 I/O 的程序，应转变为在 BDOS 下使用直接 I/O，以便它们可以在将来的 MP/M 及 CP/M 版本下得到完全支持。

进入功能 6，寄存器 E 或者包含十六进制 FF，指示控制台输入请求；或者包含一个 ASCII 码字符。如果输入值是 FF，则如果字符没有准备好，就以 A = 00返回，否则 A 包含下一个控制台输入字符。

如果 E 中输入值非 FF，则功能 6 假定 E 包含一个送到控制台的有效 ASCII 字符。

功能10：读控制台缓冲区

控制台缓冲区读操作保持不变，不同的是支持控制台行编辑（如第二节所述）。同样要注意某些回车到最左边位置的功能（即 ctl-X）只是回到指示符仃止的列位置（以前回车到最左边界上）。这一新的规则使操作员对数据输入及行修改更加清楚。

功能12：返回版本号

功能12已经重定义以提供与版本无关的编程信息(在1.4版本中以 HL = 0000返回,但是不执行操作）。由功能12返回的值是两个字节的，对 CP/M，H = 00(对 MP/M，H = 01)；对所有2.0以前的版本 L = 00。CP/M2.0在寄存器 L 中返回一个十六进制数20，对以后版本 2 的各版（Release）为十六进制数 21～2F。例如，使用功能12，用户可编写能同时提供顺序和随机存取功能的应用程序。在早期 CP/M 版本下运行时随机存取无效。

在下面描述的文件操作中，DE 寻址一个文件控制块（FCB)。所有目录操作发生在保留区，这个保留区不影响写缓冲区，但查找第一个文件段及查找下一文件段例外，那里需要兼容性。

文件控制块数据区对顺序存取为33个字节序列,对随机存取为36个字节序列。通常定位在 005CH 的缺省文件控制块能用来随机存取文件，这是由于字节007DH、007EH 及 007FH可用于这个目的。为便于说明，FCB格式如下面区域所示：

dr	f1	f2	//	f8	t1	t2	t3	ex	s1	s2	rc	d0	//	dn	cr	r0	r1	r2
00	01	02	…	08	09	10	11	12	13	14	15	16	…	31	32	33	34	35

这里：

dr： 驱动器代码（0～16）

0 对文件使用缺省驱动器

1 自动选择驱动器 A

2 自动选择驱动器B

…

16 自动选择驱动器 P

f1…f8 包含大写 ASCII 码文件名，高位 = 0

t1、t2、t3 包含大写 ASCII 码文件类型，高位 = 0

t1′、t2′、t3′代表这些位置的位

t1′ = 1 只读文件

t2′ = 1 系统文件，不能由 DIR 列出

ex 包含当前磁盘区号，通常由用户置成00，但文件 I/O 期间，取值范围为0～31

s1 保留为系统内部使用

s2 保留为系统内部使用，对 OPEN，MAKE，SEARCH 调用时置零

rc 磁盘区“ex”的记录数，取值范围为 0～128

d0…dn 由 CP/M 填入，保留为系统使用

cr 在顺序文件操作中读和写的当前记录，通常由用户置成0

r0、r1、r2 在 0～65535 范围内的任选的随机记录数，溢出至 r2，r1、r0形成一个16位值，低字节在 r0，高字节在 r1

功能15：打开文件

打开文件功能与以前定义类似，不同的是 s2 字节自动置0。注意，以前 CP/M 版本定义这一字节为0，但不作检查来确保其一致。因此在必要时可将这一字节清除以保证与最近版本的向上兼容性。

功能17：查找第一个文件目录项

该功能扫描目录表，查找与 DE 定址的 FCB 指定的文件相符合的文件。如果文件没有找到，返回值255（十六进制数FF），否则 A 的值为0、1、2或3，以指示文件的存在。在文件找到的情况下，当前 DMA 地址填入包含目录表项的记录，相应的开始位是 A*32（即 A 寄存器左移 5 位或 ADD A五次）。目录信息可以在这一位置从缓冲区中取出，但对应用程序通常不需要。

从 f1 到 ex 任何位置的 ASCII 码问号（十进制63，十六进制3F）与在缺省或自动选择的磁盘驱动器上的任何目录表项相应的区域相匹配。如果 dr 字段包含 ASCII 问号，则不能使用自动磁盘选择功能，这时，查找缺省磁盘，查找功能返回已分配地址域，或未分配的、属于任何用户号的相匹配的表项。这后面的功能通常不用于应用程序中，但可以灵活地扫描所有当前目录表值。如果 dr 字段不是问号，则 s2 字节被自动置0。

功能18：查找下一个文件目录项

该功能同上一功能很相似，所不同的是目录扫描从最后相符合的登记项开始连续下去。同功能17相似，当没有相匹配的目录表项时，在 A 中的返回值是255。

功能19：删除文件

本功能删除与由 DE 定址的 FCB 相匹配的文件。文件类型包含有歧义性引用（即问号在各种位置出现），但驱动器选择码不能有歧义性，像在查找第一个文件段及查找下一个文件段功能一样。

如果被访问文件没有找到，则功能19返回值为255，否则返回值范围是0～3。

功能22：建立文件

本功能与CP/M以前版本相同，不同的是字节s2在进入到BDOS时置0。

功能23：重新命名文件

文件重新命名功能的动作和以前版本相同，例外的是如重新命名没有成功（要重新命名的文件没有找到），则返回值是255，否则返回值是0～3。

功能24：返回登录向量

由CP/M2.0返回的登录向量值是在HL中的16位值，这里L的最低位对应于第一个驱动器A，H的高位对应于第十六个驱动器P。注意，由于返回时寄存器A与L包含相同的值，因此保持和早期版本的兼容性。

功能28：磁盘写保护

磁盘写保护功能为当前选定的盘提供了临时的写保护。在下一次冷启动或热启动之前任何企图写入该磁盘的操作，均会产生信息：

Bdos Err on d：R/O

功能29：取只读向量

本功能在寄存器对HL中返回一位向量，指出驱动器已临时置了只读位。同功能24相似，最低位相应于驱动器A，而最高位相应于驱动器P。只读位或由对功能28的明确调用、或由在CP/M内的检测变化盘的自动软件机构来设置。

功能30：置文件属性

置文件属性功能允许对与文件相关的永久指示符作程序的调整。特别是只读及系统属性（上面的t1'及t2'）可置位或复位。DE寄存器对定址具有适当属性置位或复位的确定文件名。功能30检查匹配性，并改变相匹配的目录项，以包含被选的指示符。指示符f1'到f4'目前不使用，但对应用程序是可用的，这是因为在文件打开和关闭操作期间，应用程序不包括在匹配过程中。指示符f5'到f8'及t3'保留为将来系统扩充用。

功能31：取地址（磁盘参数）

BIOS常驻磁盘参数块的地址作为这一功能的调用结果在HL中返回。磁盘参数值可取出用于显示和空间计数；或者当磁盘环境变化时，暂驻程序可在必要时动态地改变当前磁盘参数的值。一般应用程序不需这种功能。

功能32：置或取用户码

应用程序可以通过调用功能32来改变或询问当前用户号。如果寄存器E=FFH，那么当前用户号的值在A中返回，取值范围是0～31。如果E中不是FF，则当前用户号改变到E的值（模32）。

功能33：随机读操作

随机读操作功能同以前版本的顺序操作相似，所不同的是读操作对一特定记录号进行，用FCB后三个字节段构成的24位值来选择（字节位值r0在33、r1在34、r2在35）。注意24位序列的存储方式是：首先是低字节(r0)，其次是中间字节(r1)，最后是高字节(r2)。CP/M 2.0版本不参考字节r2，除非在计算文件大小时（功能35）。然而，字节r2必须置0，这是由于非零值表示在文件结束产生溢出。

因此，在2.0版本中，r0、r1字节对被作为双字节或“字”值来处理，它包含要读的记录。这个值的范围从0到65535，可对8兆字节文件任一特定的记录进行存取。为了使用随机存

取来处理一个文件，基本磁盘区（区域0）必须首先被打开。虽然基本磁盘区可能包括或不包括任何存储的数据，但这样可确保文件被适当地记录在目录中，并可用DIR命令显示出来。然后选择的记录号被存到随机记录区（r0、r1），并调用BDOS去读记录。调用返回时，寄存器A或者包含误码（如下面列出），或者包含00值表示操作成功。在后一种情况下，当前DMA地址包含随机存取记录。注意到与顺序读操作相反，随机读操作记录号不变化，因此，以后的随机读操作继续读同一个记录。

在每一随机读操作时，逻辑盘区及当前记录值被自动地置位。因此，文件可以从当前随机存取位置开始顺序地读或写。然而要注意的是，在这种情况下，当你从随机读转为顺序读时，最后一个随机记录将被重读一次；当你转向随机写时，最后一记录也将被重写。当然用户可简单地在每次随机读或写之后提高随机记录位置以获得顺序I/O操作的效果。

在随机读之后，在寄存器A中返回的错误码列出如下：

01 读未写数据
02 （不是以随机方式返回）
03 不能关闭当前磁盘区
04 寻找未写磁盘区
05 （不是以读方式返回）
06 寻道操作超过磁盘的物理结束

功能34：随机写操作

随机写操作同随机读调用相似，不同的是数据从当前DMA地址写到磁盘上。进一步说，如果要写的磁盘区或数据块还没有分配，则在写操作继续之前进行存储单元的分配。像在随机读操作时一样，写操作结束时，随机记录号不变。逻辑磁盘区号及文件控制块的当前记录位置设置为相应于正在写的随机记录。还有，在随机写之后可以开始顺序读或写操作。要注意的是，只要顺序操作开始，当前寻址到的记录就被重读或重写。用户同样可在每次写之后提高记录位置以得到顺序写操作的效果。特别要注意，以随机方式读或写一个磁盘区的最后一个记录，不能自动地引起磁盘区的转换。

由随机写操作返回的错误码同随机读操作一样，只是增加了误码05，它指出由于目录溢出而不能建立新磁盘区。

功能35：计算文件大小

当计算一个文件大小时，DE寄存器对以随机方式格式寻址FCB（存在字节r0、r1及r2）。FCB包含单义性文件名，这个名字使用在目录扫描中。返回时，随机记录字节包含“虚”文件尺寸，实际上它是跟在文件结束后的记录的记录地址。在调用功能35之后，如果高记录字节r2是01，那么文件在2.0版本中包含最大的记录数65536。否则，字节r0及r1构成一个16位的值（r0是最低位字节），它就是文件的大小。

通过调用功能35将随机记录位置设置在文件末尾，可将数据附加到现存文件后面，然后从预置的记录地址开始执行一系列随机写操作。

当顺序写文件时，文件的虚尺寸与物理尺寸相同。但如果文件是以随机方式产生，且在存储分配上存在有“空白区”，那么文件实际上包含的记录比指示的要少。例如，如果一个8兆字节文件只有最后一记录以随机方式写入（即：记录号65535），那么虚尺寸是65536个记录，实际上只有一个数据块。

功能36：置随机记录

本功能使BDOS自动地从已被顺序读或写到一特定点的文件产生随机记录位置，它可用于两个方面。

首先，常常有必要首先读和扫描一个顺序文件，以获得各种键字段的位置。当遇到每个键字段时，就调用功能36计算相应于这个键的数据的随机记录位置。如果数据单元的大小是128字节，则结果产生的记录位置放进一带有键字段的表中，以备将来检索。在扫描了全部文件并把键字段及它们的记录号列表之后，用户可通过使用以前存入的相应的随机记录号，执行一个随机读操作，迅速传送一个特定的关键字记录。当包括变长记录时，这一模式可容易地加以推广。因为程序只需存放缓冲区相对字节位置、关键字及记录号，所以可在以后找到有关键字数据的确切开始位置。

功能36的第二个用途用在从一个顺序读或写转换到随机读或写的时候。一个文件被顺序存取到文件中的一个特定点，调用功能36置记录号，以后的随机读和写操作从文件中选择的点开始继续。

下面的功能是CP/M2.2新增加的：

功能37：驱动器复位

该功能允许将特定的驱动器复位，所传递的参数是被复位驱动器的16位向量，最低有效位是驱动器A。为与MP/M兼容，CP/M返回零值。

功能40：零填充的随机写

该功能与功能34相似，区别是原来未分配的数据块在写入数据之前用零填充。

这一节我们以一相当广泛但又很完整的随机存取操作的例子作为结束。下面列出的程序执行根据终端来的命令进行随机读或写记录的简单功能。假定：程序已建立，汇编并放进一标号为RANDOM.COM的文件中，CCP级命令：

RANDOM X.DAT

开始测试程序。程序通过X.DAT寻找文件（在这特定情况下）。如果找到，提示控制台进行输入。如果没有找到，则在给出提示符之前建立文件。每个提示符取下列形式：

next command?

后面要操作员输入，并以回车作为结束。输入命令取下列形式：

nW nR Q

这里n为0～65535的整数，而W、R和Q则分别相应于随机写、读和退出处理的简化命令字符。如果执行W命令，则RANDOM程序提示：

type data :

然后操作员打入字符回答（可有127个字符），并以回车作为结束。RANDOM然后将字符串写到X.DAT文件第n个记录上去。如果执行R命令，RANDOM读记录号n并在控制台上显示字符串值。如果执行Q命令，X.DAT文件被关闭，程序返回到控制台命令处理程序。为简单起见（程序并非如此简单），错误信息只是：

error, try again

程序以一个初始化程序段开始，在这里输入文件被打开或建立；接着是在标号"ready"处连续循环，在这里进行命令的翻译。005CH的缺省文件控制块及0080H的缺省缓冲区使用在所有磁盘操作中。接着是实用子例程，它包括主输入行处理程序，称为readc。这个程序

实例给出了随机存取处理的程序结构，可以作为进一步开发程序的基础。

CP/M2.0随机存取程序样本

```
;  ***********************************************
;  *                                             *
;  *  sample random access program for cp/m 2.0  *
;  *                                             *
;  ***********************************************
;
0100                    org     100h        ; base of tpa
;
0000 =      reboot      equ     0000h       ; system reboot
0005 =      bdos        equ     0005h       ; bdos entry point
;
0001 =      coninp      equ     1           ; console input function
0002 =      conout      equ     2           ; console output function
0009 =      pstring     equ     9           ; print string until '$'
000a =      rstring     equ     10          ; read console buffer
000c =      version     equ     12          ; return version number
000f =      openf       equ     15          ; file open function
0010 =      closef      equ     16          ; close function
0016 =      makef       equ     22          ; make file function
0021 =      readr       equ     33          ; read random
0022 =      writer      equ     34          ; write random
;
005c =      fcb         equ     005ch       ; default file control block
007d =      ranrec      equ     fcb+33      ; random record position
007f =      ranovf      equ     fcb+35      ; high order (overflow) byte
0080 =      buff        equ     0080h       ; buffer address
;
000d =      cr          equ     0dh         ; carriage return
000a =      lf          equ     0ah         ; line feed
;
;  ***********************************************
;  *                                             *
;  *  load sp, set-up file for random access     *
;  *                                             *
;  ***********************************************
;
0100  31bc0             lxi     sp,stack
```

```
                ;       version 2.0?
0103  0e0c      mvi     c,version
0105  cd050     call    bdos
0108  fe20      cpi     20h         ; version 2.0 or better?
010a  d2160     jnc     versok
                ;       bad version, message and qo back
010d  111b0     lxi     d,badver
0110  cdda0     call    print
0113  c3000     jmp     reboot
                ;
                versok:
                ;       correct version for random access
0116  0e0f      mvi     c,openf     ; open default fcb
0118  115c0     lxi     d,fcb
011b  cd050     call    bdos
011e  3c        inr     a           ; err 255 becomes zero
011f  c2370     jnz     ready
                ;
                ;       cannot open file, so create it
0122  0e16      mvi     c,makef
0124  115c0     lxi     d,fcb
0127  cd050     call    bdos
012a  3c        inr     a           ; err 255 becomes zero
012b  c2370     jnz     ready
                ;
                ;       cannot create file, directory full
012e  113a0     lxi     d,nospace
0131  cdda0     call    print
0134  c3000     jmp     reboot      ; back to ccp
                ;
                ;  ********************************************
                ;  *                                          *
                ;  *  loop back to "ready" after each command *
                ;  *                                          *
                ;  ********************************************
                ;
                ;
                ready:
                ;       file is ready for processing
                ;
```

```
0137  cde50           call    readcom     ; read next command
013a  227d0           shld    ranrec      ; store input record#
013d  217f0           lxi     h,ranovf
0140  3600            mvi     m,0         ; clear high byte if set
0142  fe51            cpi     'Q'         ; quit?
0144  c2560           jnz     notq
                ;
                ;       quit processing, close file
0147  0e10            mvi     c,closef
0149  115c0           lxi     d,fcb
014c  cd050           call    bdos
014f  3c              inr     a           ; err 255 becomes 0
0150  cab90           jz      error       ; error message, retry
0153  c3000           jmp     reboot      ; back to ccp
                ;
                ;  ***********************************************
                ;  *                                             *
                ;  *  end of quit command, process write         *
                ;  *                                             *
                ;  ***********************************************
                notq:
                ;       not the quit command, random write?
0156  fe57            cpi     'W'
0158  c2890           jnz     notw
                ;
                ;       this is a random write, fill buffer until cr
015b  114d0           lxi     d,datmsg
015e  cdda0           call    print       ; data prompt
0161  0e7f            mvi     c,127       ; up to 127 characters
0163  21800           lxi     h,buff      ; destination
                rloop:  ; read next character to buff
0166  c5              push    b           ; save counter
0167  e5              push    h           ; next destination
0168  cdc20           call    getchr      ; character to a
016b  e1              pop     h           ; restore counter
016c  c1              pop     b           ; restore next to fill
016d  fe0d            cpi     cr          ; end of line?
016f  ca780           jz      erloop
                ;       not end, store character
```

```
0172  77          mov   m,a
0173  23          inx   h          ; next to fill
0174  0d          dcr   c          ; counter goes down
0175  c2660       jnz   rloop      ; end of buffer?
            erloop:
            ;       end of read loop, store 00
0178  3600        mvi   m,0
            ;
            ;       write the record to selected record number
017a  0e22        mvi   c,writer
017c  115c0       lxi   d,fcb
017f  cd050       call  bdos
0182  b7          ora   a          ; error code zero?
0183  c2b90       jnz   error      ; message if not
0186  c3370       jmp   ready      ; for another record
            ;
            ;  ***********************************************
            ;  *                                             *
            ;  *  end of write command, process read         *
            ;  *                                             *
            ;  ***********************************************
            ;
            notw:
            ;       not a write command, read record?
0189  fe52        cpi   'R'
018b  c2b90       jnz   error      ; skip if not
            ;
            ;       read random record
018e  0e21        mvi   c,readr
0190  115c0       lxi   d,fcb
0193  cd050       call  bdos
0196  b7          ora   a          ; return code 00?
0197  c2b90       jnz   error
            ;
            ;       read was successful, write to console
019a  cdcf0       call  crlf       ; new line
019d  0e80        mvi   c,128      ; max 128 characters
019f  21800       lxi   h,buff     ; next to get
            wloop:
```

```
01a2  7e          mov   a,m       ; next character
01a3  23          inx   h         ; next to get
01a4  e67f        ani   7fh       ; mask parity
01a6  ca370       jz    ready     ; for another command if 00
01a9  c5          push  b         ; save counter
01aa  e5          push  h         ; save next to get
01ab  fe20        cpi             ; graphic?
01ad  d4c80       cnc   putchr    ; skip output if not
01b0  el          pop   h
01b1  c1          pop   b
01b2  0d          dcr   c         ; count=count-1
01b3  c2a20       jnz   wloop
01b6  c3370       jmp   ready
                ;
                ; ***********************************************
                ; *                                             *
                ; *  end of read command, all errors end-up here*
                ; *                                             *
                ; ***********************************************
                ;
                ;
                error :
01b9  11590             lxi   d,errmsg
01bc  cdda0             call  print
01bf  c3370             jmp   ready
                ;
                ; ***********************************************
                ; *                                             *
                ; *  utility subroutines for console i/o        *
                ; *                                             *
                ; ***********************************************
                ;
                getchr :
                        ; read next console character to a
01c2  0e01              mvi   c,coninp
01c4  cd050             call  bdos
01c7  c9                ret
                ;
                putchr :
                        ; write character from a to console
01c8  0e02              mvi   c,conout
```

```
01ca  5f              mov   e,a          ; character to send
01cb  cd050           call  bdos         ; send character
01ce  c9              ret

                      ;
                      crlf :
                            ; send carriage return line feed
01cf  3e0d            mvi   a,cr         ; carriage return
01d1  cdc80           call  putchr
01d4  3e0a            mvi   a,lf         ; line feed
01d6  cdc80           call  putchr
01d9  c9              ret

                      ;
                      print :
                            ; print  the buffer addressed by de until $
01da  d5              push  d
01db  cdcf0           call  crlf
01de  dl              pop   d            ; new line
01df  0e09            mvi   c,pstring
01e1  cd050           call  bdos         ; print the string
01e4  c9              ret

                      ;
                      readcom :
                            ; read the next command line to the conbuf
01e5  116b0           lxi   d,prompt
01e8  cdda0           call  print        ; command?
01eb  0e0a            mvi   c,rstring
01ed  117a0           lxi   d,conbuf
01f0  cd050           call  bdos         ; read command line
                      ;     command line  is present, scan it
01f3  21000           lxi   h,0          ; start with 0000
01f6  117c0           lxi   d,conlin     ; command line
01f9  1a        readc :  ldax  d         ; next command character
01fa  13              inx   d            ; to next command position
01fb  b7              ora   a            ; cannot be end of command
01fc  c8              rz
                      ;     not zero, numeric?
01fd  d630            sui   '0'
01ff  fe0a            cpi   10           ; carry if numeric
0201  d2130           jnc   endrd
```

```
                ;       add-in next digit
0204  29                dad   h           ; *2
0205  4d                mov   c,l
0206  44                mov   b,h         ; bc = value  *2
0207  29                dad   h           ; *4
0208  29                dad   h           ; *8
0209  09                dad   b           ; *2 + *8 = *10
020a  85                add   l           ; + digit
020b  6f                mov   l,a
020c  d2f90             jnc   readc       ; for another char
020f  24                inr   h           ; overflow
0210  c3f90             jmp   readc       ; for another char
                endrd:
                ;       end of read, restore value in a
0213  c630              adi   '0'         ; command
0215  fe61              cpi   'a'         ; translate case?
0217  d8                rc
                ;       lower case, mask lower case bits
0218  e65f              ani   101$1111b
021a  c9                ret
                ;
                ;  ***********************************************
                ;  *                                             *
                ;  *  string data area for console messages      *
                ;  *                                             *
                ;  ***********************************************
                ;
                badver:
021b  536f79            db    'sorry, you need cp/m version 2$'
                nospace:
023a  4e6f29            db    'no directory space$'
                datmsg:
024d  547970            db    'type data:$'
                errmsg:
0259  457272            db    'error, try again.$'
                prompt:
026b  4e6570            db    'next command? $'
                ;
```

```
;  ***********************************************
;  *                                             *
;  *   fixed and variable data area              *
;  *                                             *
;  ***********************************************
;
027a  21    conbuf: db    conlen      ; length of console buffer
027b        consiz: ds    1           ; resulting size after read
027c        conlin: ds    32          ; length 32 buffer
0021 =      conlen  equ   $-consiz
;
029c              ds    32          ; 16 level stack
            stack:
02bc              end
```

第九节　存储器组织

同早期版本相似，CP/M2.0根据主计算机内存组织可作现场修改，以适合各种内存尺寸。对一般内存的典型基地址列表如下：

模　块	20K	24K	32K	48K	64K
CCP	3400H	4000H	6400H	A400H	E400H
BDOS	3C00H	4C00H	6C00H	AC00H	EC00H
BIOS	4A00H	5A00H	7A00H	BA00H	EA00H
RAM顶	4FFFH	5FFFH	7FFFH	BFFFH	FFFFH

提供的磁盘包含已配置成带有标准 IBM 8″软盘驱动器的20K Intel MDS-800 的 CP/M 2.0系统。磁盘布局如下所示：

扇区	磁道00	模块	磁道01	模块
1	（引导装入程序）		4080H	BDOS+480H
2	3400H	CCP+000H	4100H	BDOS+500H
3	3480H	CCP+080H	4180H	BDOS+580H
4	3500H	CCP+100H	4200H	BDOS+600H
5	3580H	CCP+180H	4280H	BDOS+680H
6	3600H	CCP+200H	4300H	BDOS+700H
7	3680H	CCP+280H	4380H	BDOS+780H

8	3700H	CCP + 300H	4400H	BDOS + 800H
9	3780H	CCP + 380H	4480H	BDOS + 880H
10	3800H	CCP + 400H	4500H	BDOS + 900H
11	3880H	CCP + 480H	4580H	BDOS + 980H
12	3900H	CCP + 500H	4600H	BDOS + A00H
13	3980H	CCP + 580H	4680H	BDOS + A80H
14	3A00H	CCP + 600H	4700H	BDOS + B00H
15	3A80H	CCP + 680H	4780H	BDOS + B80H
16	3B00H	CCP + 700H	4800H	BDOS + C00H
17	3B80H	CCP + 780H	4880H	BDOS + C80H
18	3C00H	BDOS + 000H	4900H	BDOS + D00H
19	3C80H	BDOS + 080H	4980H	BDOS + D80H
20	3D00H	BDOS + 100H	4A00H	BIOS + 000H
21	3D80H	BDOS + 180H	4A80H	BIOS + 080H
22	3E00H	BDOS + 200H	4B00H	BIOS + 100H
23	3E80H	BDOS + 280H	4B80H	BIOS + 180H
24	3F00H	BDOS + 300H	4C00H	BIOS + 200H
25	3F80H	BDOS + 380H	4C80H	BIOS + 280H
26	4000H	BDOS + 400H	4D00H	BIOS + 300H

特别要注意，CCP 同1.4 版 本 一样，在磁盘上的同一位置，且占有同样的空间。然而BDOS 部分占有另一些256字节页，BIOS部 分则是01磁道所剩余的部分。因此CCP 长度是800H（十进制2048）字节，BDOS 长度为 E00H(十进制3584)字 节，BIOS 达 380H（十进制898）字节。在2.0版本中，BIOS 部 分 包 含1.4版本的标准子例程以及一些已初始化的表空间，如下节所述。

第十节 BIOS 差别

CP/M2.0基本 I/O 系统 (BIOS)在概念上与从前的只有一些微小差别。它定义了两个新的转移向量入口点，包括了一个新的扇区转换子例程以及定义了一个磁盘特性表。这些变化的大概形式可以在下面给出的程序中找到。

```
1:              org     4000h
2:              maclib  diskdef
3:              jmp     boot
4: ;            ...
5:              jmp     listst ; list status
6:              jmp     sectran  ; sector translate
7:              disks   4
8: ;            large capacity drive
9: bpb          equ     16*1024  ; bytes per block
```

```
rpb     equ     bpb/128  ; records per block
maxb    equ     65535/rpb  ; max block number
        diskdef 0,1,58,3,bpb,maxb+1,128,0,2
        diskdef 1,1,58,bpb,maxb+1,128,0,2
        diskdef 2,0
        diskdef 3,1
;
boot:   ret     ; nop
;
listst: xra     a       ; nop
        ret
;
seldsk:
        ; drive number in c
        lxi     h,0     ; 0000 in hl produces select error
        mov     a,c     ; a is disk number 0 ... ndisks-1
        cpi     ndisks  ; less than ndisks?
        rnc             ; return with HL = 0000 if not
;       proper disk number, return dpb element address
        mov     l,c
        dad     h       ; *2
        dad     h       ; *4
        dad     h       ; *8
        dad     h       ; *16
        lxi     d,dpbase
        dad     d       ; HL = .dpb
        ret
;
selsec:
        ; sector number in c
        lxi     h,sector
        mov     m,c
        ret
;
sectran:
        ; translate sector BC using table at DE
        xchg            ; HL = .tran
        dad     b       ; single precision tran
;       dad b again if double precision tran
```

```
49:             mov     l,m     ; only low byte necessary here
50: ;           fill both H and L if double precision tran
51:             ret             ; HL = ? ? SS
52: ;
53: sector:     ds      1
54:             endef
55:             end
```

参考上面所示程序，3～6行代表17个元素的 BIOS 入口向量（1.4 版本仅定义了15个转移向量元素）。后面两个元素提供了对 DESPOOL 的“LISTST”（列表状态）入口点的存取。这个特定入口点的使用在 DESPOOL 文件中定义，并且与 1.4 版本没有差别。应该注意，1.4版本的 DESPOOL 程序不能在2.0版本下运行，但对不久以后的 Digital Research 的修改版本将是有效的。

在第 6 行转换向量中示出的“SECTRAN”（扇区号转换）提供了对 BIOS 中的扇区转换子例程的存取。这一机构允许用户为一特定的磁盘系统指定扇区偏差因数及转换。这将在下面描述。

宏定义库称为 DISKDEF，包含在第 2 行，并在 12～15 行中引用。用户可以不使用宏定义库，但它可以大大地简化磁盘定义过程。尽管在 CP/M2.0 所有提供盘上已包含宏定义库，但用户若想使用 DISKDEF 功能，必须会使用 MAC 宏汇编（详见“CP/M2.2 修改指南”）。

BIOS 磁盘定义包括以下宏指令语句序列：

```
MACLIB      DISKDEF
.....
DISKS       n
DISKDEF     0, .....
DISKDEF     1,
.....
DISKDEF     n-1
.....
ENDEF
```

这里 MACLIB 语句将 DISKDEF.LIB 文件（与 BIOS 在同一张盘上）装入 MAC 的内部表中。接着是 DISKS 宏调用，它确定用户系统要配置的驱动器数，n 为 1～16 的整数。然后是一系列的 DISKDEF 宏调用，它定义 0～(n-1)（相应于逻辑驱动器 A～P）的每一个逻辑磁盘特性。注意 DISKS 及 DISKDEF 宏指令产生一系列固定的数据表，因此，一定要放在 BIOS 的非执行区域，一般是直接跟在 BIOS 转移向量后面。

在 DISKDEF 宏指令之后定义 BIOS 的剩余部分，在 END 语句之前是 ENDEF 宏调用。ENDEF 宏指令产生必要的未初始化的 RAM 区域，它定位于 BIOS 之上。

DISKDEF 宏调用的形式是

DISKDEF dn, fsc, lsc, (skf), bls, dks, dir, cks, ofs, [0]；这里：

dn 逻辑磁盘号（0 到 n-1）

fsc 第一个物理扇区号（0或1）
lsc 最后一个扇区号
skf 任选的扇区偏差因数
bls 数据定位块尺寸
dir 目录表项数
cks 被检验的目录表项数
ofs 到逻辑磁道00的位移
[0]任选的1.4兼容标记

值 dn 是用 DISKDEF 宏指令定义的驱动器号。fsc 参数说明不同的扇区编号系统，其值一般是 0 或 1。lsc 是一磁道的最后一个编号扇区。当 skf 参数存在时，它定义扇区偏差因数。根据这个偏差，可以建立一个扇区转换表。如果扇区数少于256，则产生单字节表；否则每一转换表元素占两个字节。如果略去 skf 参数（或为 0），则不建立转换表。bls 参数指定分配到每一数据块上的字节数，且取值1024，2048，4096，8192 或 16384。

通常，随着数据块尺寸增大，性能可提高。因为访问目录表次数减少，逻辑相关的记录在磁盘物理位置上相互靠近，而且每一目录表项访问的数据较多，BIOS 驻留的 RAM 空间减少。dks 指定总的磁盘容量，以 bls 为单位。即：如果 bls = 2048 且 dks = 1000，那么总的磁盘容量为2,048,000字节。如果 dks 大于255，则块尺寸参数 bls 必须大于1024。dir 的值是总的目录表项数目，如果需要，可以超过255。cks 参数决定在每次目录扫描时校验的目录项数目，它用于系统运行期间，未发生人工干预冷启动或热启动时，内部检查变更的磁盘（当检测到这种情况时，CP/M 自动将该盘标识为只读，以免破坏数据）。正常情况下，当介质易于改变时，cks 的值等于 dir，如软盘子系统。如果磁盘是永久安装的，则 cks 的值一般为 0，因为不进行重启动而改变磁盘的可能性很小。ofs 的值决定当寻址到这一特定驱动器时要跳过的磁道数，它可用来保留附加的操作系统空间，或在单个大容量物理驱动器上模拟几个逻辑驱动器。最后，当1.4版本已经修改，可用于高密度磁盘时，如果需要与1.4版本文件兼容，可包括[0]参数。它保证对每一目录记录只分配16K，同以前版本一样。通常情况下，不包括这一参数。

为表空间使用的方便性和经济性，特殊形式

DISKDEF i, j

给磁盘 i 与以前定义的驱动器 j 相同的特性。一个标准的与1.4版本兼容的四驱动器单密度系统可使用下面的宏调用定义：

```
DISKS    4
DISKDEF  0, 1, 26, 6, 1024, 243, 64, 64, 2
DISKDEF  1, 0
DISKDEF  2, 0
DISKDEF  3, 0
.....
ENDEF
```

假定所有磁盘的参数值相同：每一磁道（编号 1 到26）26个扇区，每一次存取跳过 6 个扇区，每个数据块1024字节，对243K 字节磁盘总容量有243个数据块，64 个被校验的目录

表项及两个操作系统磁道。

上面所示程序（12～15行）给出的定义提供了对 CP/M2.0 最大可寻址的磁盘的存取。所有磁盘具有相同的参数，不同的是驱动器 0 和 2 在每存取一个数据时跳过 3 个扇区，而驱动器 1 和 3 当磁盘旋转时按顺序存取每一扇区（然而在这些驱动器上可以有一个透明的硬件偏差因数）。

DISKS 宏指令产生 n 个“磁盘标题块（disk header block）”，起始于由宏指令产生的标号地址 DPBASE。每一个磁盘标题块包含16个字节，而且顺序地对应于定义的每一个驱动器。例如，在四驱动器标准系统中，DISKS 宏指令产生以下形式的表格：

```
DPBASE EQU $
DPE 0 :DW XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV0, ALV0
DPE 1 :DW XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV1, ALV1
DPE 2 :DW XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV2, ALV2
DPE 3 :DW XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV3, ALV3
```

这里 DPE（disk parameter entry）标号用于参考目的，表示为 0 到 3 每一个驱动器的开始表地址。包含在磁盘参数标题内的值在“CP/M2.2 修改指南”中详细介绍，该值说明驱动器的转换向量（所有对 XLT0 的引用，在上例中是驱动器 0 的转换向量），三个16位的暂存器地址，目录缓冲地址，磁盘参数块地址，校验向量地址以及分配向量地址。校验及分配向量地址均由在 BIOS 码及表格之后的 RAM 域中的 ENDEF 宏指令产生。

DELDSK 功能在2.0版本中略有扩展。被选的磁盘号同以前一样，放在寄存器 C 中送到 BIOS 中去，且 DELDSK 子例程执行适当的软件及硬件动作去选择磁盘。然而2.0 版本还需要 SELDSK 子例程在寄存器 HL 中返回被选的磁盘参数标题地址（在上例中为 DPE0、DPE1、DPE2 或 DPE3）。如果 SELDSK 在 HL 中的返回值为 0000H，则 BDOS 假定磁盘不存在，并在终端上显示选择错信息。程序22～36行给出一个样本 CP/M2.0 SELDSK 子例程，只表示磁盘参数标题地址的计算。

在 CP/M2.0 版本中同样包括了子例程 SECTRAN，它执行实际逻辑到物理扇区的转换。在 CP/M 早期版本中，扇区转换处理是 BDOS 的一部分，并置成在每次读之间跳过六个扇区。为了区分各种磁盘的旋转速度，在2.0版本中转换功能已变成 BIOS 的一部分。因此，BDOS 送顺序的扇区号到 SECTRAN，起始于扇区号 0。SECTRAN 子例程使用顺序扇区号去产生一个经过转换的扇区号，送回到 BDOS。在执行实际读或写前，BDOS 顺序送转换的扇区号到 SELSEC。注意许多控制器具有在磁盘本身记录扇区偏差的能力，因此没有必要转换。在这种情况下，skf 参数在宏调用中省略，并且 SECTRAN 返回的值与接收到的值相同。例如，当在 DISKDEF 宏调用中指定标准偏差因数 skf = 6时构成下面所示的表：

```
XLT0: DB 1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21
      DB 2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22
```

如果需要 SECTRAN 转换一扇区，则进行下面的处理。要转换的扇区在寄存器对 BC 中接收。如果扇区值不超过255，则只有寄存器 C 实际有效（此时 B = 00）。寄存器对 DE 寻址这个驱动器的转换表，它由以前对 SELDSK 的调用确定，相应于磁盘参数标题第一个元素（在上面情况中为 XLT0）。然后 SECTRAN 子例程通过将输入扇区号加至转换表的

基地址取出经转换的扇区号，得到索引转换表地址（见上面程序的第46，47，48行）。然后该单元的值返回寄存器 L 中。注意，如果扇区号超过255，则转换表包含16位元素，它们的值必须在 HL 中返回。

在 ENDEF 宏调用后，定义一些未初始化数据区。这些数据区不一定作为 BIOS 的一部分，在冷启动时装入，但它必须在 BIOS 与内存结束地址之间可用。未初始化 RAM 区域的大小由 ENDEF 宏指令产生的 EQU 语句来决定。对于一个标准的四驱动器系统，ENDEF 宏指令可以产生

```
4C72 =    BEGDAT  EQU $
          (数据区域)
4DB0 =    ENDDAT  EQU $
013C =    DATSIZ  EQU  $-BEGDAT
```

指明未初始化 RAM 起始于 4C72H，终止于 4DB0H-1，占 013CH 字节。用户必须保证在系统装入之后这些地址是自由空间。

CP/M2.0同样可容易地适用于扇区大小是128字节倍数的磁盘子系统，BDOS 提供了在扇区写操作时，不必要进行预读操作的有关信息，因此允许在 BIOS 级进行程序组块和解块。

对特定的硬件配置剪裁 CP/M 系统的其他细节介绍，请参阅“CP/M2.2修改指南”一章。

第二章　介绍CP/M特性和功能

第一节　引　　言

CP/M 是为微型计算机系统开发的一种监控程序，可以使用与 IBM 兼容的磁盘作为后援存储器。用 Intel 8080微型机作为主机，CP/M 提供了程序设计、存储和编辑以及汇编和程序调试等功能。CP/M 的一个重要特点是它可以很容易地转换到任何以 Intel 8080（或 Zilog Z-80）作为中央处理器的微型机上运行，这些微型机至少应有16K 字节以上的内存，可多达四个与 IBM 兼容的磁盘驱动器。关于在特定硬件环境下所需的修改详细描述请参看“CP/M2.2修改指南”一章。

虽然标准的 Digital Research 版本是在单密度的 Intel MDS 800 上运行，但其他一些硬件制造商也为 CP/M 提供了他们自己的输入/输出驱动器。

CP/M 监控程序，通过一个内容丰富的文件管理软件包，为程序提供了快速存取。这个文件子系统提供了一个命名的文件结构，允许动态分配文件空间，并可对文件进行顺序和随机存取。使用这个文件系统时，大量不同的程序可以按源程序和可执行的机器指令存放。

CP/M 还有一个强有力的文本编辑程序、与 Intel 兼容的汇编程序以及调试程序子系统。任选软件包括功能很强的与 Intel 兼容的宏汇编，符号查错程序以及各种高级语言。如果把它们与控制台命令处理程序连接在一起，那么，它的功能可相当于甚至超过类似的大型计算机。

CP/M 在逻辑上划分为几个不同部分：

BIOS	基本 I/O 系统（与硬件有关）
BDOS	基本磁盘操作系统
CCP	控制台命令处理程序
TPA	暂驻程序区

BIOS 提供了基本的磁盘驱动器存取操作和标准外部设备接口（电传打字机、CRT、纸带输入机、纸带穿孔机以及用户定义的外部设备）。用户可以修改 BIOS 部分以适应特定的硬件环境。

BDOS 是一个磁盘管理系统，可以控制一个或多个包含独立的文件目录表的驱动器。BDOS 的磁盘分配原则是使存取过程中磁头移动最少，所以它完全是一个动态文件结构。文件可以有任意个记录，但不能超出一个磁盘的容量。标准的 CP/M 系统每张盘最多可容纳64个不同文件。BDOS 有许多入口点，包含下列按程序进行的基本操作：

SEARCH	按名检索某个文件
OPEN	为后面的操作打开一个文件
CLOSE	处理结束后关闭一个文件
RENAME	改变指定文件的文件名
READ	从指定文件上读一个记录

WRITE　　往磁盘上写一个记录

SELECT　　为后面的操作选择一个磁盘驱动器

CCP 提供用户控制台和 CP/M 系统其他部分之间的联系。CCP 从控制台读入命令并进行处理。这些命令包括：列文件目录、打印文件内容和控制暂驻程序（例如汇编程序、编辑程序和调试程序）等的操作。CCP 的标准命令在第四节中列出。

CP/M 的最后一部分是暂驻程序区（TPA)。TPA 保存用 CCP 命令从磁盘装入的程序。例如，在编辑一个程序时，TPA 将保存 CP/M 文本编辑程序的机器码和数据区。同样，在 CP/M 控制下建立的程序，可以装入到 TPA 中执行，对这个程序进行校核。

需要提及的是 CP/M 的子系统可以部分或全部被执行程序复盖。也就是说，用户程序一旦装入 TPA，则 CCP、BDOS 和 BIOS 区都可以作为用户程序的数据区。当 BIOS没有被复盖时，引导程序可以工作。因此，用户程序只需在执行程序结束时转移到引导程序，就可以再一次从磁盘装入整个 CP/M 监控程序。

CP/M 操作系统可以被划分为不同的模块，包括 BIOS 。前面已经说过，BIOS 用于定义 CP/M 运行的硬件环境，因此，改变外部设备有关部分去适应特定硬件系统，便可将标准系统很容易地改成任何非标准环境下运行的系统。

第二节　CP/M 功能描述

用户和 CP/M 系统之间的联系，主要是通过 CCP 进行的。CCP 读入和解释从控制台进入的命令。通常，CCP 从若干个联机磁盘驱动器中选址。标准系统有四个磁盘驱动器，标号为 A、B、C、D。如果 CCP 定址在一个磁盘上，则这个磁盘被登录为联机磁盘。为了清楚地表示哪一个磁盘是当前的联机磁盘，CCP 在磁盘文件名后跟一个提示符“>”显示，表明准备好接收命令。初始启动后，CP/M 系统从磁盘 A 读入，并且 CCP 显示如下信息：

xxK CP/M VER m.m

其中 xx 表示该 CP/M 系统管理的内存大小（以 K 字节为单位），m.m 是 CP/M 版本号。所有 CP/M 系统在初始化时均被置成在 16K 内存空间运行，但是很容易重新改变配置，使之适合于任何容量内存的系统（参看 MOVCPM 命令)。系统建立后，CP/M 自动登录磁盘 A，给出提示符“A>”（表示 CP/M 当前定址在磁盘 A）并且等待命令。命令分为内部命令和外部命令两种。

2.1　一般命令结构

内部命令是 CCP 程序本身的一部分。外部命令从磁盘装入 TPA 并执行。

内部命令有：

ERA　删除指定文件

DIR　列出目录表中的文件名

REN　文件重新命名

SAVE　以文件形式保存内存的内容

TYPE　打印联机磁盘上的文件内容

几乎所有的命令都要引用特定的一个文件或一组文件，文件引用的形式下面详细说明。

2.2 文件引用

文件引用标识 CP/M 联机磁盘上一个特定文件或一组文件。这些文件引用或者是单义的（无歧义的，ufn），或者是多义的(有歧义的,afn)。单义性文件名唯一地标识一个文件，多义性文件名可以对应若干个不同文件。

文件引用包括两个部分：基本名和扩展名。虽然扩展名是任选的，但它通常表示一种类属。例如，扩展名“ASM”表示该文件是汇编语言源文件。而基本名用于区别每一个源文件。两个名用“·”分开，如下所示：

pppppppp.SSS

其中 pppppppp 表示基本名，可以为 8 个字符或少于 8 个字符。SSS 是扩展名，不能多于 3 个字符。如上所述，名

pppppppp

也是允许的。这意味着扩展名是三个空格。用于表示单义性文件引用的字符，不能包含下列特殊字符：

<>. , ; : = ? * [ ]

而字母数字和除这些特殊字符之外的所有字符都允许使用。

多义性文件名用于目录检索和形式匹配。其格式与单义性文件名相似，只是基本名和扩展名中可以包含“？”号。在 CP/M 的各种命令中，符号“？”的位置可以是任意可用字符。因此，多义性引用

X?Z.C?M

与单义性文件名

XYZ.COM 和

X3Z.CAM

相对应。

注意，多义性引用 *.* 等同于

????????.???

而 pppppppp.* 和

*.SSS

分别是 pppppppp.???和

????????.SSS

的缩写，例如：

DIR *.*

CCP 把它解释为列出目录表中所有的磁盘文件名，而

DIR X.Y

只检索名为 X.Y 的文件。类似地，命令

DIR X?Y.C?M

表示检索满足上述多义性引用的所有单义性文件名。

下列文件名作为单义性文件引用是有效的：

X	XYZ	GAMA
X.Y	XYZ.COM	GAMA.I

为方便起见，程序员一般可以指定文件名，同时指定磁盘驱动器名。这时，驱动器名由一个字母（A～E）后跟一个冒号“：”组成。在文件操作之前，这个指定的驱动器被登录，因此，下列是有效文件名，它们是以磁盘名为前缀的文件名：

A：X.Y	B：XYZ	C：GAMMA
Z：XYZ.COM	B：X.A?M	C：*.ASM

应该注意的是，所有文件名和磁盘驱动器名的小写字母通过 CCP 处理时，均变成大写字母。

第三节　磁盘切换

当 CCP 等待控制台输入时，操作者可以从控制台打入磁盘驱动器名（A、B、C、D），其后面跟一个冒号“：”，用于选择当前的联机磁盘。因此，CP/M 系统从磁盘 A 装入后，显示下列命令和提示符：

16K CP/M	VER1.4
A>DIR	列出磁盘 A 上的所有文件
SAMPLE	ASM
SAMPLE	PRN
A>B：	选择 B 盘
B>DIR*.ASM	列出 B 盘上所有“ASM”文件
DUMP	ASM
FILES	ASM
B>A：	返回 A 盘

第四节　内部命令形式

上面所描述的文件和驱动器引用，可以用来说明内部命令的结构，用缩写表示如下：

ufn——单义性文件引用

afn——有歧义性文件引用

cr——回车

前面说过，CCP 总是把小写字母变成大写字母，因此，小写字母在命令名和文件名中都作为大写字母看待。

4.1 ERA afn cr

ERA（删除）命令是从当前的联机盘（也就是 CP/M 显示的提示符“>”前面的磁盘名）中把文件删去。被删除文件是满足那些有歧义性引用的 afn 文件。下面举例说明 ERA 的使用方法：

ERA X.Y　当前联机盘上名为 X.Y 的文件从磁盘目录中删去，并且退回所占的磁盘空间

ERA X.*　所有基本名为 X 的文件从当前盘中删去

ERA *.ASM　所有扩展名为 ASM 的文件从当前盘中删去

ERA X?Y.C?M 当前盘上满足有歧义性引用X?Y·C?M的所有文件都删去

ERA *.* 删去当前盘上所有文件。这时，CCP在控制台上显示信息："ALL FILES (Y/N)?"，回答Y，文件才实际被删除

ERA B:*PRN B盘上所有符合有歧义性引用????????·PRN的文件都从当前盘中删去

4.2 DIR afn cr

DIR（列目录）命令将使符合有歧义性文件名afn的所有文件都在控制台上显示出来。而命令DIR表示显示当前联机盘上的文件（"DIR"命令等价于"DIR *.*"），有效的DIR命令举例如下：

```
DIR  X.Y
DIR  X?Z.C?M
DIR  ? ? .Y
```

类似于其他CCP命令，afn前面可以添上驱动器名。下列DIR命令用于在查找目录表前对选定的驱动器寻址：

```
DIR  B:
DIR  B:X.Y
DIR  B:*.A?M
```

如果在选定磁盘目录中找不到该文件，则在控制台上显示信息：

"NOT FOUND"

4.3 REN ufn1=ufn2 cr

REN（重命名）命令，允许用户改变磁盘上文件的名字。如果当前盘上有满足ufn2的文件，则重新命名为ufn1，假定重命名后的文件存放在当前联机盘上。如果用户控制台提供左箭头字符，则可以用它代替"="号。REN命令举例如下：

```
REN  X.Y=Q.R      文件名Q.R改成X.Y
REN  XYZ.COM=XYZ.XXX
                  文件名XYZ.XXX改成XYZ.COM
```

操作者可以在ufn1或ufn2之前加一个选择驱动器地址。假定ufn2前有驱动器名，则认为ufn2是在与ufn1同一个驱动器上。同样地，如果ufn1有驱动器名，则认为ufn2也在那个驱动器上。如果ufn1和ufn2都有驱动器名，则这两个驱动器名必须相同。下面的REN命令说明了这种格式：

```
REN  A:X.ASM=Y.ASM
                  在驱动器A上，文件Y.ASM改成X.ASM
REN  B:ZAP.BAS=ZOT.BAS
                  在驱动器B上，文件ZOT.BAS改成ZAP.BAS
REN  B:A.ASM=B:A.BAK
                  在驱动器B上，文件A.BAK改成A.ASM
```

如果文件ufn1已经在盘上，那么REN命令将回答错误信息"FILE EXIST"，并且不执行更改命令。如果ufn2不在指定磁盘上，则在控制台上显示信息：

"NOT FOUND"

4.4 SAVE n ufn cr

SAVE 命令是从 TPA 置入 n 页（每页256字节）到磁盘上，并把这个文件取名为ufn。在提供的 CP/M 系统中，TPA 从100H（16进制）开始，它是内存的第二页。因此，如果用户程序是从100H 到 2FFH，那么 SAVE 命令必须指定两页内存，然后，机器码文件可以装入并执行。

例如：

```
SAVE  3  X.COM   把100H 到3FFH 复制到 X . COM上
SAVE  40 Q       把100到28FFH 复制到Q[注意：28是28FFH 的页数，
                 28H = 2*16 + 8 = 40(十进制)]
SAVE  4  X.Y     把100H 到 4FFH 复 制 到 X.Y上
```

SAVE 命令也可以用afn 命令形式表示磁盘驱动器。

请看下例：

```
SAVE  10  B:ZOT.COM  把10页 (100H 到 AFFH) 内存复制到磁盘B的
                     文件 ZOT.COM 上
```

4.5 TYPE ufn cr

TYPE 命令在控制台上显示当前联机磁盘的 ASCII 码源程序文 件 ufn 内容，有效命令格式为：

```
TYPE  X.Y
TYPE  X.PLM
TYPE  XXX
```

TYPE 命令用 ctl-I 作扩展标记，每 8 列置一个标记，ufn 也能引用驱动 器名，如下所示：

```
TYPE  B:X.PRN  显示磁盘 B 上的文件 X.PRN
```

第五节　行编辑和输出控制

在打入命令行时，CCP 提供某些行编辑功能：

rubout　删去或回送从控制台打入的最后一个字符

ctl-U　删去从控制台打入的一行

ctl-X　（与 ctl-U 相同）

ctl-R　重新打入当前命令行：使用 rubout 删去字符之后，接着 打 入 “clean line”

ctl-E　命令行实际结束，回车，打入回车键后，命令行才被送入

ctl-C　CP/M 系统重新启动（热启动）

ctl-Z　结束控制台输入（用在 PIP 和 ED）

控制功能 ctl-P 和 ctl-S 影 响 控 制输出，说明如下：

ctl-P　把后继控制台输出送到当前指定的列表设备（参看 STAT 命令），输 出送到列表设备和控制台设备，直至打入下一个 ctl-P 为止

ctl-S　暂停控制台输出。从控制台打入下一个字符时（如打 ctl-S），程序继续执行和输出。这个功能常常用于暂停高速控制台的输出。例如对 CRT，为了看一看 连续

输出中某一段，可以使用这个功能

注意，上面提到的功能是同时按下 ctl 键和字符键，CCP 命令一般长度可达到 255 个字符，只有按回车键，命令才起作用。

第六节　外部命令

外部命令是从当前联机盘上装入 TPA 中并执行的。在 CCP 控制下外部命令的执行含义如下所示。用户很容易定义附加的功能（参看 LOAD 命令定义）。

STAT　列出当前联机磁盘上余下的存储器字节数，提供关于一个特定文件的统计信息，显示或改变驱动器指定

ASM　把CP/M 的汇编程序装入内存并且从磁盘上汇编指定的程序

LOAD　以 Intel 十六进制机器代码格式装入文件，并产生一个可执行的机器码形式，这个可执行的形式指的是可以装入 TPA （这个被装入程序变成 CCP 控制下的一个新命令）

DDT　把 CP/M 调试程序装入 TPA 并启动执行

PIP　装入外部设备交换程序，用于后继的磁盘文件操作和外部设备的传输操作

ED　装入和执行 CP/M 文本编辑程序

SYSGEN　建立一个新的 CP/M 系统磁盘

SUBMIT　提供一个命令文件作批处理用

DUMP　以十六进制形式倾出文件内容

MOVCPM　对于指定的内存量重新生成 CP/M 系统。如同对内部命令一样，描述外部命令。用户很容易定义附加的命令。为方便起见，外部命令前面可以加驱动器名。这就使得外部命令可以从指定的驱动器装入 TPA 并执行。因此，命令

B：STAT

将引起 CP/M 临时登录联机盘 B 作为 STAT 外部命令的源盘，然后返回到原来联机盘作后继处理。

基本的外部命令详细叙述如下：

6.1 STAT cr

STAT 命令提供了关于文件存储和设备分配的统计信息。可打入下列命令之一来启动：

STAT cr

STAT “命令行” cr

特殊形式的命令行允许检查当前的设备分配，并且可作改变。不同的命令行在下面作解释，右边是说明：

STAT cr　如果用户打入空命令行，则 STAT 外部命令计算所有活动驱动器上剩余的存储量并显示信息：

X：R/W，SPACE：nnnK

或　X：R/O，SPACE：nnnK

对于每一个活动的驱动器X，其中R/W表示可读写，R/O表示只读（一个驱动器可以按要求变成只读，也可能由于不小心没有通过热启动而变成只读）。X驱动器上的剩余空间以千字节为单位，用nnn表示

STAT X：cr 如果驱动器名给定，则该驱动器在计算存储容量之前已经选定。因此，当登录驱动器A时，发出命令“STAT B：”，将显示信息：

BYTES REMAINING ON B： nnnK

STAT afn cr 该命令行是通过STAT扫描一组文件，符合afn的文件按字母顺序一一列出，在每个文件标题下列出所需存储器数量。

RECS BYTS EX D：FILENAME·TYP

rrrr bbbk ee d：pppppppp·sss

其中rrrr是分配给这个文件的128字节记录的数量，bbb是分配给这个文件的K字节数(bbb = rrrr * 128/1024)，ee是16K区域数(ee = bbb/16)，d是存放该文件的驱动器名（A……Z），pppppppp是基本文件名，最多为8个字符，sss是扩展名，最多为3个字符。列出每个文件之后，还给出存储器使用的总量。

STAT X：afn cr 为方便起见，驱动器名可以放在afn之前。在这种情况下，指定的驱动器第一个被选中，并执行“STAT afn”。

STAT X：=R/O cr 这种形式使指定的驱动器X变为只读。这种情形一直保留到下一个热启动或冷启动开始。当一个磁盘已是只读时，企图在其上写信息，则显示：

BDOS ERR ON X：READ ONLY

在执行自动热启动之前（热启动可使之变成R/W），一直处于等待。

STAT命令可以控制物理设备到逻辑设备分配的转换(参看“CP/M接口指南”和“CP/M系统修改指南”上的IOBYTE功能）。通常有四个逻辑外部设备，任何时候每一个逻辑设备将分配给一个实际的外部设备。这四个逻辑设备的名称是：

CON： 系统控制台设备（用于CCP和操作员之间的通讯）

ROR： 纸带输入设备

PUN： 纸带穿孔设备

LST： 输出打印设备

连接到计算机系统的实际设备都是由CP/M的BIOS子程序启动的。例如逻辑设备RDR实际上可以是一台高速输入机、电传打字机或盒式磁带机。为了使设备命名和分配有一定的灵活性，下面定义了若干个外部设备：

TTY： 电传设备（慢速控制台）

CRT： 阴极射线管（高速控制台）

BAT： 成批处理（控制台就是当前的RDR：，输出到当前运行的LST：设备）

UC1： 用户定义的控制台

PTR： 纸带输入机（高速输入机）

UR1： 用户定义的 *1 输入机

UR2： 用户定义的 *2 输入机

PTP： 纸带穿孔机（高速穿孔机）

UP1： 用户定义的 #1 穿孔机

UP2： 用户定义的 #2 穿孔机

LPT： 行打印机

UL1： 用户定义的 #1 列表设备

需要着重指出，物理设备名可以但不一定要实际对应该名所指的设备。例如，如果用户愿意的话，PTP：设备可以执行盒式带机的写操作。准确的对应关系和驱动子程序在CP/M的BIOS上定义。在CP/M标准版本中，这些设备所对应的名都在MDS 800开发系统上。

如果打入

STAT VAL: cr

所有可能的逻辑设备和物理设备对应的分配都能显示出来，STAT将显示出对每一个逻辑设备的那些可能的物理设备：

CON:=TTY: CRT: BAT: UC1

RDR:=TTY: PTR: UR1: UR2

PUN:=TTY: PTP: UP1: UP2

LST:=TTY: CRT: LPT: UL1

每一行左边的逻辑设备可以对应右边的四个物理设备中任一个。打入命令：

STAT DEV: cr

可以显示当前逻辑设备和物理设备的对照表，也就是列出左边每个逻辑设备和所对应的右边的物理设备，例如列表如下：

CON:=CRT:

RDR:=UR1:

PUN:=PTP:

LST:=TTY:

当前的逻辑设备和物理设备的分配，可以由如下形式的STAT命令加以改变：

STAT ld1=pd1, ld2=pd2, ……ldn=pdn cr

其中ld1到ldn是逻辑设备名。pd1到pdn是对应的物理设备名（亦即ldi和pdi出现在上面命令“VAL:”的同一行上）。下面的STAT命令可以改变当前的逻辑设备和物理设备的分配：

STAT CON:=CRT: cr

STAT PUN:=TTY:, LST:=LPT:, RDR:=TTY: cr

6.2 ASM ufn cr

ASM命令装入并执行CP/M8080汇编程序。ufn表示源程序文件是用汇编语言写的，其中扩展名假定为ASM，因此不需要说明。下面的ASM命令是有效的：

ASM X

ASM GAMA

两遍扫描后，汇编程序自动执行。如果第二遍扫描有错误，就在控制台显示错误信息。

汇编程序产生一个文件

X.PRN

其中 X 是 ASM 命令中指定的基本名。PRN 文件包含源程序列表（在源程序中带有标记字符）以及每个语句产生的机器码。如果有错误，则还包含诊断错误信息。用 TYPE 命令，PRN 文件可以在控制台上显示出来，或者用 PIP（参看 PIP 命令的结构）送到一个外部设备上去。还要注意，PRN 文件包含有源程序，而这个程序最左边16列可能包含各种各样的汇编信息（如程序地址和十六进制机器码）。因此，PRN 文件可以作为初始源程序文件的后备文件。当源程序文件被破坏，则可以通过对文件 PRN 进行编辑，将每一行左边16个字符去掉（可以通过使用单个编辑宏命令）。最后的结果文件与初始源文件相同，并且可以用 REN 将该文件重新命名。为 ASM 用于后继的编辑和汇编，汇编程序还产生一个文件

X.HEX

包含 Intel "十六进制" 形式的8080机器语言，用于后继的装入和执行（参看LOAD 命令）。如果要更详细了解 CP/M 汇编语言程序，可参阅 "CP/M 汇编程序（ASM）" 一章。

与其他外部命令类似，准备汇编的源程序文件可以从某个磁盘取出，可在汇编语言文件名前面加上驱动器名来决定是哪一个磁盘。因此，命令

ASM B:ALPHA cr

是从当前联机磁盘中装入汇编程序，并在 B 驱动器对源程序 ALPHA.ASM 进行汇编。HEX 和 PRN 文件在这种情况下，也是放在驱动器 B 上的。

6.3 LOAD ufn cr

LOAD 命令是读入文件 ufn，假设这个文件是 "十六进制" 机器代码，读入后产生一个内存映象文件，可在以后执行。文件名 ufn 的形式假定为：

X.HEX

因此，仅需在这个命令中指定名 X。LOAD 命令建立一个名为

X.COM

的文件，这说明该文件包含可执行的机器代码。当用户在 CCP 给出提示符 ">" 之后，立即打入文件名 X，则该文件装入内存并实际执行。

一般情况下，紧接着提示符之后，CCP 读入名 X，并查找内部功能名。如果未找到，CCP 检索系统磁盘目录，按名 X.COM 查找文件。如果找到，将机器代码装入 TPA，并且执行这个程序。因此，用户仅需装入十六进制文件一次，以后只需简单地打入基本名，便可执行任意次。这样，用户可以创造新命令，经初始化的磁盘将外部命令作为 COM 文件保存，它可以按用户的选择删除。如果文件名前面有驱动器名，就可以在替换磁盘上操作。因此，命令

LOAD B:BETA

把 LOAD 程序从当前联机盘装入 TPA，开始执行后转向 B 盘上操作。

需要注意的是 BETA.HEX 文件必须包含有效的 Intel 形式的十六进制机器代码记录（好象由 ASM 产生的记录一样），这个记录从100H 开始，它是 TPA 的开始地址。并且，十六进制记录中的地址一定是递增的。十六进制记录被读入时，那些没有使用的内存区被 LOAD 命令填上零。因此，仅当建立在 TPA 中运行的 CP/M 标准 "COM" 文件时，才使用 LOAD 命令。占据除 TPA 外的内存的程序，可用 DDT 命令装入内存。

6.4 PIP cr

PIP 是CP/M 的外部设备交换程序。该程序执行装入、打印、穿孔、复制以及组合磁盘文件时需要转换存储介质的操作。

PIP 程序通过打入下列命令形式之一进行启动：

（1）PIP cr

（2）PIP “命令行” cr

这两种命令都可以把 PIP 装入 TPA，并且执行该程序。第一种是直接从控制台读入命令行，它的提示符是“*”，一直到遇有空命令行（即操作员打入回车）为止。每打入一个命令行，按下面的规则产生介质转换。第二种格式和第一种一样，不同的是 PIP中的单个命令行是自动执行的。而 PIP 结束时，控制台不再提示请求输入命令行。每个命令行的形式是：

destination = source#1, source#2, …, source#n cr

其中 “destination”是文件名或接收数据的外部设备。而 “source#1, …, source#n” 表示一系列的文件或设备。其内容被从左到右复制到destination上。

如果多个文件是在命令行中给出（即n＞1），则认为每个文件都包括ASCII字符，而且每个文件末尾都带有CP/M的文件结束符（ctl-Z），要取代这个假设请参看O参数。如果控制台提供了左箭头字符，则可以用它代替“=”号，以改进可读性。小写字符在内部要变成大写字符，使得与CP/M文件和设备命名规则相一致。整个命令行的长度不得超过 255 个字符（如果命令行超过控制台宽度，可用ctl-E强制回车）。

目的和源文件可以作为CP/M的单义性源文件，前面可以带有驱动器名，也可以不带驱动器名。也就是任何文件前，可以带驱动器名（A：、B：、C：、D：），用于定义存取该文件的驱动器。如果驱动器名没有标出，则认为当前联机盘就是该驱动器名。进一步说，目的文件也可作为一个或多个源文件出现。在这种情况下，在整个联结没有完成前，源程序文件是不能变动的。如果目的文件已经存在，则在命令行正确形成之后，该目的文件取消（如果有错误则不能取消）。下列命令行对于输入PIP是有效的：

X = Y cr　把Y文件复制到X文件上去，其中X和Y是单义性文件名，Y保留不变

X = Y,Z cr　把Y和Z两个文件连起来，再把它复制到文件X中去。Y和Z不改变

X.ASM = Y.ASM, Z.ASM, FIN.ASM cr　把Y、Z、FIN文件连起来（这些文件都属于ASM型），建立一个X.ASM文件

NEW.ZOT = B：OLD.ZAP cr　在驱动器B上，把OLD.ZAP文件传送到当前盘上，取名为NEW.ZOT文件

B：A.U = B：B.V, A：C.W, D.X cr　连结驱动器B上的文件B.V和驱动器A上的文件C.W以及当前盘上的文件D.X，在B上建立一个新文件A.U

为便于使用，PIP允许在驱动器之间传输使用缩写命令，缩写形式为：

PIP X：=afn cr

PIP X：=Y：afn cr

PIP ufn=Y： cr

PIP X：ufn=Y： cr

第一种形式是从当前盘上，把满足afn的所有文件复制到驱动器 X 上（X = A, …Z），取相同的文件名。第二种形式和第一种一样，其中要复制的文件在驱动器Y上(Y = A, …Z)。第三种形式等价于命令 “PIP ufn = Y：ufn cr”，把驱动器Y的ufn文件复制到驱动器X上。

第四种形式等价于第三种，其中源盘是Y。

注意，在以上情况中，源文件盘和目的文件盘应该是不同的。如果指定了afn，则PIP在复制时列出每一个满足afn的ufn文件；如果存在一个同名的目标文件，则在复制完后消去，由新复制文件代替。

下面给出有效的从磁盘到磁盘复制操作的PIP命令实例：

B：=*.COM cr　将当前盘上所有扩展名为“COM”的文件复制到驱动器B上

A：=B：ZAP.* cr　从驱动器B上将所有基本名为“ZAP”的文件复制到驱动器A上

ZAP.ASM=B：cr　等同于ZAP.ASM=B：ZAP.ASM

B：ZOT.COM=A：　cr　等同于B：ZOT.COM=A：ZOT.COM

B：=GAMMA.BAS cr　等同于B：GAMMA.BAS=GAMMA.BAS

B：=A：GAMMA.BAS cr　等同于B：GAMMA.BAS=A：GAMMA.BAS

PIP允许引用物理和逻辑设备，只要这些设备属于CP/M系统。设备名与使用STAT命令时给出的相同，并有一些特殊命名的设备。

STAT命令给出的逻辑设备有：

CON：（console）控制台

RDR：（reader）读入机

PUN：（Punch）穿孔机

LIST：（list）打印机

物理设备有：

TTY：（控制台，输入机，穿孔机或打印机）

CRT：（控制台或打印机）　UC1：（控制台）

PTR：（读入机），UR1：（读入机），UP2：（读入机）

PTP：（穿孔机），UR1：（穿孔机），UP2：（穿孔机）

LPT：（打印机），UL1：（打印机）

注意：“BAT：”物理设备不包括在内，因为这个设备指定仅仅表示RDR：和LST：用作控制台输入/输出。

RDR，LST，PUN和CON设备都在CP/M的BIOS部分定义。因此，对于特定的I/O系统很容易修改（当前物理设备的对照是由IOBYTE给出的，参看“CP/M接口指南”一章中关于这个功能的描述）。目的设备一定要能接收数据（也就是说，数据不可能送到穿孔机），源设备一定要能产生数据（也就是说LST设备不可能读数据）。

可用于PIP命令的其他设备名是：

NUL：　把40个空字符（ASCII的0）送至设备（这可在穿孔输出结束时进行）

EOF：　把CP/M的文件结束符(ASCII ctl-Z)送至目的设备(通过PIP命令进行ASCII数据传输结束时自动送出)

INP：　特殊的PIP输入源程序，它可以插入PIP程序本身中，PIP通过调用103H单元逐个字符取输入数据，并将数据返回109H单元（奇偶位一定是0）

OUT：　特殊的PIP输出目的数据，可以插入到PIP程序中。PIP调用106H单元(其他数据存放在C寄存器中）传送每一个字符。注意，PIP内存映像的109H到IFFH单元是没有用的，在用DDT时可被专用驱动器取代（参看DDT操作说明）

PRN：和LST：一样，不同的是在每第八个字符位置扩展标记，进行行编号，每60行组成一页（与[t8np]相同）

在PIP命令中，可以插入文件和设备名，这时，从指定的某一个设备一直读入到文件的结束符为止（ASCII文件是ctl-Z，非ASCII码磁盘文件是文件的实际结束）。磁盘上或文件上的数据都是从左到右连接在一起，直到读完最后一个数据。可以把源程序和数据写入到目的设备和数据中，并在ASCII文件后附加上文件结束符（ctl-Z）。注意，如果目的数据文件是磁盘文件，那么就建立一个临时文件(扩展名为$$$)。如果复制文件完成，则临时文件名改为实际文件名（带有“COM”扩展名的文件被看成是非ASCII码文件）。

当复制在进行时，按下任何键（也可能是误动作）都会导致复制失败。PIP用信息“ABORTED”显示操作没有完成。注意，如果操作失败，或在处理过程中出现错误，PIP将取消在用SUBMIT时建立的任何未完成的命令。

如果目的文件是一个十六进制（Intel十六进制形式的机器代码）磁盘文件，而源文件是在一个外部设备（如纸带读入机）上时，就应该注意，在这种情况下，PIP程序核对源程序以保证源程序文件是一个标准的十六进制格式文件，并具有合法的十六进制数值以及校验和记录。当发现一个无效输入记录时，PIP将在控制台上回答一个出错信息，并等待修改动作。通常可以打开输入机，重读一段纸带（先把纸带退回20英寸）。当纸带准备好重新输入时，在控制台上打一个回车，则PIP在错误信息显示之后，按一下回车键，便可继续读带。在构成磁盘文件之后，可用ED输入记录。为方便起见，如果源程序文件是RDR：设备，PIP允许从控制台上输入文件结束符。这时，PIP程序读该设备并且监控键盘。如果在键盘上打入ctl-Z，则读操作正常结束。

有效的PIP命令如下：

PIP LST：=X.PRN cr

把X.PRN复制到LST设备上，结束PIP程序

PIP cr　　启动PIP处理命令序列（提示符*）

*CON：=X.ASM,Y.ASM,Z.ASM cr

连接三个ASM文件，并将它复制到CON设备上去

*X.HEX=CON：,Y.HEX,PTR：　cr

通过读CON（直到打入ctl-Z），随后读Y.HEX的数据，最后读PTR的数据，建立一个HEX文件。整个过程以ctl-Z为结束符

*cr　　回车，停止PIP

PIP PUN：=NUL：,X.ASM,EOF：,NUL：　cr

把40个空字符送到穿孔设备，然后把X.ASM文件复制到穿孔设备上，跟以文件结束符ctl-Z，再送40个空字符

用户可以指定一个或多个PIP参数，这些参数放在左右方括号中，用零和空格分开。每一个参数均影响复制操作。括号内的参数表必须紧跟在受影响的文件或设备后面。一般情况下，每一个参数可以跟随一个任选的十进制整数值（除S和Q参数外），有效的PIP参数叙述如下：

B　成组传输，通过PIP把数据送入缓冲区，直至从源设备收到ASCII x-off字符(ctl-S)时结束送入。这可以从一个连续读入设备（如盒式带机）把数据传送到磁

盘文件。接受了x-off之后，PIP就清缓冲区并返回以便再输入数据。可以被输入缓冲区的数据量取决于主系统的内存大小。如果缓冲区溢出，PIP将发出错信息

Dn　删去从源文件传送数据到目的文件时超出第n列的数据字符。这个参数常用于截断送到窄行打印机或控制台设备的长字符行

E　将所有传输操作回送到控制台

F　从文件中筛选打印格式，取消所有嵌入的打印格式馈给，P参数同时可用于插入新的打印格式

H　十六进制数据的传送：所有数据都要按Intel十六进制数的文件格式检验。复制时，将所有十六进制记录中的非基本字符取消。如果发生错误，控制台将提示请求校正动作

I　在传输Intel十六进制格式文件中忽略“:00”记录（I参数自动设置H参数）

L　把大写字母转换为小写字母

N　给每一行填写行号，从第一行开始，每传送一行，递增1，前零可以忽略，行号后跟以冒号。如果指定N2，则包括了前零，且在行号后插入一个标记。如果置成T，则标记扩展

O　目的文件（非ASCII码）传输：忽略通常的CP/M文件结束符

Pn　每n行跳一页（带有初始跳页）。如果n = 1，则一页是60行，如果使用参数F，在新的跳页参数插入之前，不产生打印纸进给

Qs↑Z　当碰到字符串s（以ctl-Z结尾）时，就退出从源设备和文件中复制

Ss↑Z　当碰到字符串s（以ctl-Z结尾）时，就开始从源设备复制。S和Q参数用于“提取”一个文件的指定的一段（如子程序）在复制操作中总是包含启动或退出字符串

注意：如果使用形式（2）的PIP命令，那么跟在S和Q参数后面的字符串由CCP变成大写字符，但PIP形式（1）不能转换成大写字符

（1）PIP cr

（2）PIP“命令行”cr

Tn　在从源文件到目的文件传输时，对每n列加扩展标记（ctl-I字符）

U　在复制中，把小写字母转换成大写

V　在写操作之后，通过重读的方法核实已复制的数据的正确性（目的文件必须为磁盘文件）

Z　将每一个ASCII字符输入的奇偶位置零

下列是有效的PIP命令，并指定了文件传输参数：

PIP X.ASM = B:[V] cr

把磁盘B上的X.ASM文件复制到当前盘上，并核实复制数据的正确性

PIP LPT: = X.ASM[nt8u] cr

把X.ASM复制到LPT:设备上去，每行标行号，每8列加扩展标记，并把小写字母转换成大写字母

PIP PUN: = X.HEX[i],Y.ZOT[h] cr

首先把X.HEX文件复制到PUN：设备上去，并去掉文件 X.HEX 中的尾记录“:00”，然后继续读Y.ZOT数据，它是十六进制记录，包括“：00”记录

PIP X.LIB = Y.ASM[sSUBR 1 : ↑z qJMP L3↑z] cr

把文件Y.ASM复制到文件X.LIB中，当找到字符串“SUBR1：”时开始复制，遇到字符串“JMP L3”就停止复制

PIP PRN：=X.ASM[p50]

把X.ASM传送到LST：设备，标上行号，每 8 列加扩展标记，每50行组成一页。注意，nt8p60是作为PRN文件的参数表，用p50取代缺省值

6.5 ED ufn cr

ED程序是CP/M系统文本编辑程序。在CP/M环境下可以建立文件和修改ASCII文件。详细操作可参看“CP/M磁盘系统文本编辑程序”。总的来说，ED 允许操作者建立和运行源文件，这个源文件由ASCII 字符序列组成，中间由行结束符（回车-换行序列）分隔开。行的长度实际上没有限制（每一行不能超过工作存储器大小）。长度是括在两个回车之间的字符数。

ED有许多命令，用于字符串检索、替换和插入，这些命令对于在CP/M系统下建立与修改程序和文本是很有用的。虽然 CP/M 的内存单元工作区有限（在 16K 的 CP/M 中约5000字符），但要建立的文件大小是不受限制的，因为数据在工作区中是分页的。

启动ED，如果源文件不存在，ED就建立一个新文件，并打开该文件以便存取。如果源文件已经存在，程序员可以在源文件中添加数据（参看A命令），送入工作区进行编辑。送入的数据可以显示，修改并从工作区写回到磁盘上去（参看W命令）。可在程序中特定点自动分页和定位（参看N命令），这个命令允许方便地从一个大文件存取某一段。

如果操作者打入

ED X.ASM cr

则ED程序建立一个名为

X.$$$

的中间工作文件，用于在ED运行期间，保存已编辑的数据。ED完成后，X.ASM文件(原来的文件）改名为X.BAK，已编辑好的工作文件重新命名为X.ASM。因此，X.BAK 文件包含（没有编辑的）源文件，而X.ASM 文件包含已编辑好的新文件。操作者可以通过消除最新版本返回到原先的文件版本，并且给先前的版本重命名。例如，若文件X.ASM编辑不当，则可用下面的CCP命令序列回收备用文件。

DIR X.* 检验BAK文件是否可用

ERA X.ASM 删去最新版本

REN X.ASM=X.BAK BAK文件重新命名为X.ASM

注意，操作者可在任何地方中止编辑（由于重引导、电源故障、ctl-C或Q命令）而不破坏原来文件。在这种情况下，BAK文件没有建立，原来文件仍是完整无损的。

ED程序还允许用户在两个磁盘之间来回传递文件和建立后备文件。这种情况的命令形式是

ED ufn d：

其中ufn是在当前联机盘上编辑的文件名，d是替换驱动器名。ED 程序读入并处理源文

件，用文件名ufn往驱动器d上写新文件。处理完后，源文件变成备用文件。因此，如果操作者寻址磁盘A，那么下列命令是有效的：

ED X.ASM B:

该命令是在磁盘A上编辑文件X.ASM，在磁盘B上建立新文件X.$$$，编辑完成后，A:X.ASM改名为A:X.BAK，B:X.$$$重新命名为B:X.ASM。为方便用户，在编辑结束后，当前联机盘变为B盘。注意，在编辑开始前，如果名为B:X.ASM的文件存在，则在控制台上显示信息

FILE EXISTS

以防止不小心破坏该源文件。在这种情况下，操作者首先要删去现存文件，然后重新启动编辑程序。

与其他外部命令相似，可在源文件名前面加上驱动器名，编辑一个不在当前联机磁盘上的文件。

下面是有效的编辑命令：

ED A=X.ASM　在磁盘A上编辑文件X.ASM，产生的新文件和备用文件仍在A上

ED B:X.ASM A:　在磁盘B上编辑文件X.ASM，放在A盘的临时文件X.$$$中，编辑结束时，把B盘上的X.ASM变为X.BAK，A盘上的X.$$$变为X.ASM

6.6 SYSGEN cr

SYSGEN命令允许生成一个包含CP/M操作系统的初始化磁盘，SYSGEN程序以交互方式提示控制台请求输入，如下所示：

SYSGEN cr　启动SYSGEN程序

SYSGEN VERSION m.m　SYSGEN注册信息

SOURCE DRIVE NAME (OR RETURN TO SKIP)

回答包含CP/M系统的磁盘驱动器名（A、B、C或D），如果由于使用MOVCPM命令，CP/M已复制到内存，则仅需打入回车。当打入一个驱动器名X时，响应为：

SOURCE ON X THEN TYPE RETURN

把包含CP/M操作系统的磁盘插入到驱动器X（X是A、B、C或D），就绪后，打入回车

FUNCTION COMPLETE

系统复制到内存，SYSGEN作如下提示：

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

如果要对磁盘进行初始化，可把一个新磁盘放入驱动器中，且用驱动器名回答，否则打一个回车，系统将从驱动器A中重新引导。打入驱动器名X后，SYSGEN给提示信息：

DESTINATION ON X THEN TYPE RETURN

把新磁盘放入驱动器X，就绪后，打入回车

FUNCTION COMPLETE

新磁盘已在驱动器X上初始化完毕

“DESTINATION”提示符一直重复出现到打入回车键，这样可以对多个磁盘进行初始化。

系统生成成功后，新磁盘上包含操作系统，但仅有内部命令可以使用。一个新的与IBM兼容的磁盘对CP/M来说是带有一个空目录的磁盘。因此，操作者一定要从现存CP/M磁盘上复制一定数量的COM文件到新磁盘上。这可以使用PIP命令来做。

打入命令

```
PIP B:A:*.*[V] cr
```

用户可以从磁盘A上复制所有文件到磁盘B，并核实每个已复制好的文件的正确性。当复制操作进行时，每一个文件名都在控制台上显示出来。

应该注意的是，SYSGEN不破坏已经存在磁盘上的文件，它仅仅是构成一个新的操作系统。进一步说，如果磁盘只是从驱动器B到D上使用，不在驱动器A上放引导操作的源文件，则不需要SYSGEN。事实上，对CP/M来说，使用一个新磁盘是不需要初始化的。

6.7 SUBMIT ufn parm#1 … parm#n cr

SUBMIT命令允许将CP/M命令成批放在一起作自动处理。ufn是文件名，它必须在当前联机盘上，并假定其文件型为“SUB”。SUB型文件包含CP/M的原型命令，并可作参数替换。将实际参数parm#1，…，parm#n代入原型命令中，如果不产生错误，则被替代的命令文件由CP/M进行顺序处理。

原型命令文件是通过使用ED程序建立的，带有形式参数 $\$_1\$_2\$_3\cdots\$_n$，它们与文件提交执行时的实际参数的数目相同。当SUBMIT命令执行时，实际参数parm#1，…parm#n是与原型命令中的形式参数 $\$_1\$_2\cdots\$_n$ 相对应的。如果形式参数和实际参数的数量不一样，则SUBMIT命令的功能中止，控制台显示出错信息。SUBMIT的功能是在联机盘上建立一个替换命令文件，命名为$$$.SUB。当系统重新引导时（SUBMIT结束），这个命令文件被CCP作为一个输入源文件读入，而不是从控制台上读入。如果SUBMIT功能是在A盘以外的其他盘上执行，则在这个磁盘插入驱动器A和系统重新引导之前，该命令是不执行的。进一步说，当该命令读入和回送时，用户可在任何时候从控制台打入rubout，中止该命令的处理。这时，$$$.SUB文件取消，后继命令从控制台上打入。如果CCP发现命令中有错误，则命令也中止。当删去任何存在的$$$.SUB文件而产生错误时，CP/M控制下执行的程序可中止命令文件的处理。

为在SUBMIT文件中引入美元符$，用户可在命令文件中打入“$$”，在命令文件内部简化为一个“$”字符。一个向上的箭头符号“↑”放在字母X前面，则在文件中产生一个ctl-X字符。

SUB文件的最后一个命令可以启动另一个SUB文件，将成批的命令链接起来。

假设磁盘上存在文件ASMBL.SUB，且含有原型命令：

```
ASM   $1
 DIR  $1.*
ERA   *.BAK
 PIP  $2:=$1.PRN
ERA   $1.PRN
```

而命令

```
SUBMIT ASMBL X PRN cr
```

是由操作者发出的。SUBMIT程序读ASMBL.SUB文件时，用“X”替换所有的$1，“PRN”

替换所有的$2，结果产生$$$.SUB文件，它包括如下命令：

```
ASM  X
DIR  X.*
ERA  *.BAK
PIP  PRN:=X.PRN
ERA  X.PRN
```

这些命令通过CCP按顺序执行。

SUBMIT功能可以存取备用驱动器上的SUB文件，这时，文件名前面要带有驱动器名，但被提交的文件仅仅在驱动器A上才起作用。因此，在驱动器B上建立一个文件，然后插入到磁盘驱动器A上运行，是完全可能的。

6.8 DUMP ufn cr

DUMP程序用十六进制格式将磁盘文件（ufn）内容显示在控制台上，文件内容一次列出16个字节，绝对字节地址列在每一行的左边。在控制台上按rubout键，可以使长输出打印中止。关于DUMP程序实例可参看“CP/M接口指南”一章。

6.9 MOVCPM cr

MOVCPM程序允许用户对于特定内存重新形成CP/M系统。可以使用两个任选参数表示：

（1）新系统内存的大小

（2）新系统在内存中的程序结束位置

如果没有第一个参数或仅仅给出“*”号，则MOVCPM程序将以最大内存容量重新形成系统。它按主系统内存RAM以K字节为单位计算（从0000H开始）。如果没有第二个参数，则系统运行，但不作永久性记录。如果给出“*”号，则系统留在内存中，准备执行SYSGEN操作。MOVCPM程序重新定位内存映象，并且把这个映象放入内存，准备用于系统生成操作。命令形式是：

MOVCPM cr　重新定位并执行CP/M以管理当前内存配置（从100H开始，对连续的内存进行检测），重新定位后，新系统开始运行，但不永久性记录在磁盘上。生成的系统包含了Intel MDS 800的BIOS。

MOVCPM n cr　建立一个重新定位的CP/M系统以管理一个nK字节系统（其中n必须在16到64之间）并开始执行该系统。如上所述。

MOVCPM * * cr　对当前内存配置构成一个重新定位的内存映象，并把内存映象留在内存，准备用于SYSGEN操作。

MOVCPM n * cr　对nK字节内存系统构成一个重新定位的内存映象，并把内存映象留在内存，准备用于SYSGEN操作。

举例：MOVCPM * *

构成一个新的CP/M系统版本，并把它留在内存，准备用于SYSGEN。操作完成后，在控制台显示如下信息：

```
READY FOR "SYSGEN" OR
 "SAVE 32 CPM XX.COM"
```

其中XX是当前内存大小（以K字节为单位），然后操作者可以打入：

SYSGEN cr　开始系统生成

SOURCE DRIVE NAME (OR RETURN TO SKIP)

以回车作为响应，跳过CP/M读操作。因为经过前一次MOVCPM操作，系统已经在内存中

SOURCE DRIVE NAME (OR RETURN TO REBOOT)

回答B，把新系统写到B盘上，SYSGEN提示：

DESTINATION ON B, THEN TYPE RETURN

把新磁盘放在驱动器B上，准备好之后，打入回车

注意：如果上面回答“A”而不是“B”，那么系统就写到驱动器A上而不是写到B上，SYSGEN将继续显示提示信息：

DESTINATION DRIVE NAME

(OR RETURN TO REBOOT)

直至操作者打入回车，SYSGEN程序停止，重新引导系统。用户还可以用新的或旧的磁盘，走一遍重引导过程。完成了MOVCPM功能之后，用户可以不进行SYSGEN操作，而是打入：

SAVE 32 CPMxx.COM

把CP/M内存映象以可补订形式写入当前联机盘上。当运行在非标准环境下时，这是很必要的，因为在非标准环境下必须修改BIOS以适应特定的外部设备配置。见“CP/M修改指南”。

有效的MOVCPM命令如下：

MOVCPM 48　cr　形成一个内存48K的CP/M版本并开始执行

MOVCPM 48 * cr　形成一个内存48K的CP/M版本，准备作永久性记录，响应信息：

READY FOR “SYSGEN” OR

“SAVE 32 CPM 48.COM”

MOVCPM * * cr　形成一个使用最大内存的CP/M版本并开始执行

需着重指出，新建立的系统按原来磁盘上的号码顺序排列，并遵照“数字研究公司软件许可协议”的规定。

第七节　BDOS 错误信息

在文件处理过程中，基本磁盘系统可能有三种错误的情况。在检测出其中一种错误后，BDOS印出信息：

BDOS ERR ON X : error

其中X是驱动器名，“error”是三个错误信息之一，三个错误信息是：

BAD SECTOR

SELECT

READ ONLY

“BAD SECTOR”信息表示这个磁盘控制器的电子线路在读写这个盘的操作时检测出错误。如果每月不止一次发现这个错误，就应该检查控制器电子线路和存储介质。用不同厂家生产的控制器，在读文件时，也会出现这种情况。虽然控制器与IBM兼容，但人们常常发现记录格式有小的差异。例如MDS-800控制器需要在CRC字节之后跟两个“1”字节，但IBM格式则不需要。这就导致Intel MDS磁盘几乎都可以由其他与IBM兼容的系统所读取。

而在MDS读其他厂家生产的磁盘时，就会产生“BAD SECTOR”信息。在任何情况下，发现这种错误要进行恢复，可通过打入ctl-C重引导（这是最安全的），或按回车键，忽略文件操作时的坏扇区。但要注意，如果恰好在写目录时打回车键，则会损害磁盘的完整性。为防止这种情况，必须有足够的后备文件。

“SELECT”错误是指在A到D范围以外寻址。这时，错误信息中X值指出所选磁盘驱动器，在这种情况下可从控制台打入任何字符重新引导。

“READ ONLY”错误是指企图往只读盘上写。只读是通过STAT命令指定的，或者是通过BDOS设置的。一般情况下，在换磁盘后，操作者应该通过热启动（ctl-C）重新引导CP/M，或执行冷启动。如果换上去的磁盘是只读盘，则BDOS容许变换磁盘而不进行热启动或冷启动，但在内部标志该驱动器为只读。如果作热或冷启动，则驱动器状态变为可读/写的。出现这个信息时，CP/M等待从控制台输入，输入任意字符后，自动进入热启动。

第八节　在MDS上的CP/M操作

这一节叙述在Intel MDS微型机开发系统上使用CP/M的过程。假定读者已具备MDS的硬件和软件系统的基础知识。

CP/M启动方法基本上和Intel的ISIS操作系统相同。MDS上的磁盘驱动器标号是0到3，分别对应CP/M驱动器A到D。CP/M系统盘插入0号驱动器，顺序按下BOOT和RESET开关，中断方式2指示灯应该继续亮。如果按下空格杠，指示灯应该熄灭（如果没熄灭，就要检查连接线和波特率）。然后断开BOOT开关。CP/M注册信息应该在所选择的控制台上出现，后面跟以系统提示符“A>”，然后用户可以发出各种常驻和暂驻命令。

按前面板的INT0开关，CP/M系统可以重启动（热启动）任意次数。内部的Intel ROM监控程序可以通过打开INT7开关进行启动（它将产生RST7）。如果是在DDT下操作，则是不行的。因为在这种情况下DDT程序获得控制。

在任何时候磁盘都可以从驱动器中取出，在不影响数据完整性的情况下，系统可以关闭。但要注意，除非插入的磁盘是只读盘，否则，用户不能不重新引导系统（冷启动或热启动）就取出磁盘，换上另一张盘。

如果出现硬件故障，CP/M显示如下信息：

BDOS ERR ON X：BAD SECTOR

其中X是发生永久性错误的驱动器。这个错误可能由于驱动器门打开后没有关好，或是磁盘、驱动器、控制器故障引起。用户可以在控制台上打入一个回车，忽略这个错误。这种错误可能产生一个坏的数据记录，要求重新初始化128个字节数据。操作者可以重引导CP/M系统并再操作一次。

结束CP/M对话期间不需要特殊的动作，但在关掉电源之前，要将磁盘取出，以避免随机的暂态过程，这对于驱动器电路来说是经常发生的。应该注意的是宁愿用工厂新出品的与IBM兼容的磁盘，而不要用原先用于ISIS任何版本的磁盘。特别是ISIS“FORMAT”操作在整个盘上产生非标准的扇区编号，这种非标准编号严重降低了CP/M的性能，其运行速度比提供的版本明显减慢。如果要重新格式化一个磁盘(不是标准磁盘)，可写一个在CP/M控制下的程序，使得MDS800控制器用顺序扇区编号重新格式化每一个磁道的26个扇区。

第三章　CP/M磁盘系统文本编辑程序

第一节　文本编辑指导

1.1　文本编辑简介

ED是CP/M系统的文本编辑程序，用来建立与修改CP/M的源文件。ED在CP/M系统中是由打入如下的命令格式来启动的：

$$\text{ED}\ \left\{\begin{array}{l}\langle\text{文件名}\rangle\\ \langle\text{文件名}\rangle\cdot\langle\text{文件类型}\rangle\end{array}\right\}$$

一般情况下，文本编辑程序ED把由〔文件名〕或〔文件名〕·〔文件类型〕所给定的源文件段读入内存中，该文件由操作人员从键盘上进行修改，并且修改好之后就写回到磁盘上去。如果在编辑前源文件不存在，就由编辑程序ED建立并初置为空文件。文本编辑程序ED的全部操作如图1所示。

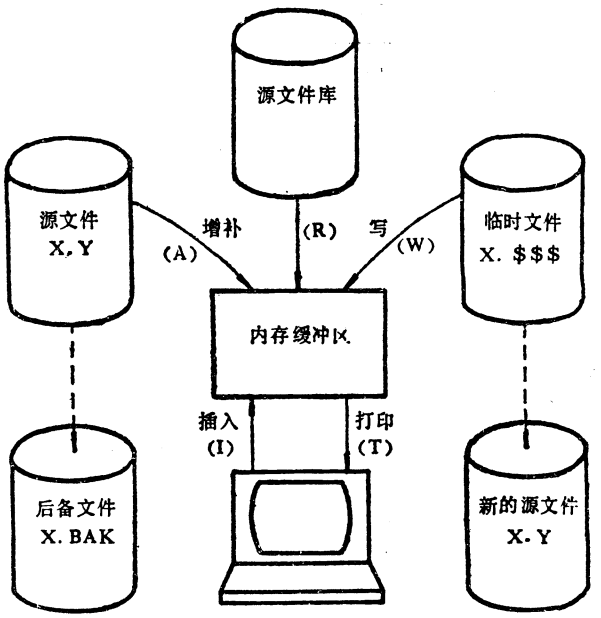


图1　ED总操作

注：ED程序允许大写与小写的ASCII字符作为控制台的输入字符。单字符命令也可以大写或小写。可以使用U命令把控制台输入字符串中的小写字符转换成大写字符，存入内存缓冲区中，但显示的控制台字符不变。-U命令使ED变为非转换方式。文本编辑程序ED在开始时假定为-U方式。

1.2 文本编辑操作

文本编辑程序对图1中的X·Y所指明的源文件进行操作，并经过一个内存缓冲区来扫视整个文本。在缓冲区，文本可以检查或修改（内存缓冲区中所含的行数随着行的长度而变化，在16K的CP/M系统中总容量约为6000个字符）。在操作员命令作用下，已编辑好的文本内容写到一个临时工作文件中，当编辑结束时，内存缓冲区及源文件中剩余部分（尚未读入内存缓冲区）的文本写到该临时文件中，且原先的文件名由X·Y变为X·BAK以便在需要时可将本次编辑之前的源文件重新取出（可参照CP/M的清除ERASE与重命名RENAME的命令）。然后临时文件由X·$$$变为X·Y，该X·Y文件成为编辑结果文件。

内存缓冲区逻辑上在源文件与临时文件之间，如图2所示。

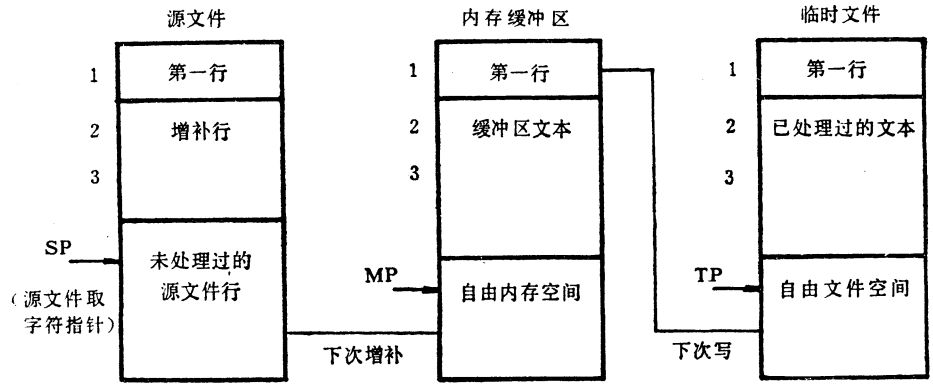


图2 内存缓冲区组织

1.3 文本传送功能

假定n是一个0～65535间的整数值，下列ED命令把源文件文本经由内存缓冲区传送到临时文件（或最终文件）中：

nA〈cr〉 从源文件的SP（即源文件取字符指针）处将下面n行尚未经处理的源文件附加到内存缓冲区的MP（内存缓冲区指针）处，SP与MP均增加n

nW〈cr〉 把内存缓冲区的首n行写到临时文件的自由空间，余留的n+1行经由MP移位到内存缓冲区顶部，指针TP增加n

E〈cr〉 文本编辑结束，把缓冲区中所有文本及未处理过的源文件拷贝到临时文件中，并重新命名这些文件，如前所述

H〈cr〉 自动执行E命令并回到新文件开始处，临时文件变成新的源文件，且内存缓冲区成为空区，并建立一个新的临时文件（相当于先执行E命令，随后重新调用ED以X·Y作为源文件来编辑）

O〈cr〉 返回到源文件，内存缓冲区成为空区，临时文件被删除，指针SP返回到源文件开始位置，先前的编辑命令无效

Q〈cr〉 退出编辑，源文件不变，并返回到CP/M

这里需考虑到一些特殊的情况。由于在ED命令中n允许为整数，因此，如果整数n被省略，就假定n=1。于是命令A与W分别表示为增补一行（将源文件一行写到缓冲区中）与写一行（将内存缓冲区中当前行写到盘上）。另外，如果以英镑符号#取代n，则表示n为最大数65535（n所允许的最大整数值为65535），因为大多数容量大小合理的源文件可以整个装入内存缓冲区，通常在编辑开始时发#A命令把整个源文件读到内存中。类似的命令#W是把整个缓冲区中的内容写到临时文件中去。为方便，向用户提供命令A和W的两种特殊形式。0A命令至少将内存缓冲区装满一半，而0W命令至少将一半内存缓冲区的内容写到盘的临时文件中。还要注意当内存缓冲区溢出时，会发出一种错误信息，此时操作员可以输入任何命令，只要该命令不增加对内存的要求就行（例如W命令）。在溢出时需读的行的余留部分可以由下次增补命令填入到内存缓冲区中。

1.4 内存缓冲区组织

内存缓冲区可以看作是由A命令从源文件中引进的一系列源文件行。内存缓冲区有一个相关的（假想）字符指针CP，可以在操作员的命令下，在整个内存缓冲区中移动。内存缓冲区在逻辑上如图3所示。其中：虚线表示长度不定的源文件行字符，用回车〈cr〉键与换行〈lf〉键来表示本行结束。CP表示假想字符指针。注意：指针CP总是位于第一行第一个字符的前面，最后一行最后一个字符的后面，或者介于两个字符之间。当前行（CL）是指含有指针CP的源文件行。

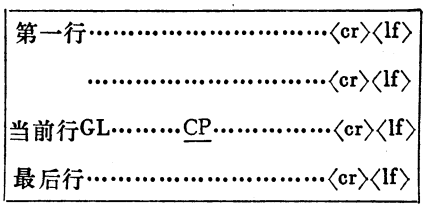


图3 · 内存缓冲区逻辑组织

1.5 内存缓冲区操作

在文本编辑ED开始之前，内存缓冲区是空区（即指针CP在第一个字符之前、最后一个字符之后）。操作员可以用A命令从源文件中增补若干行到缓冲区中，也可以用插入命令

I〈cr〉

直接从控制台输入若干行。编辑程序ED可接收任意数量的输入行，其中每一行需以回车〈cr〉键作结束（换行〈lf〉是自动供给的），直到操作员打入control-z键为止（control-z用↑z表示）。指针CP位于最后一个输入字符之后。序列

```
I〈cr〉
NOW IS THE 〈cr〉
TIME FOR 〈cr〉
ALL GOOD MEN 〈cr〉
↑z
```

使内存缓冲区如下所示：

NOW IS THE 〈cr〉〈lf〉

TIME FOR ⟨cr⟩⟨lf⟩

ALL GOOD MEN ⟨cr⟩⟨lf⟩ CP

然后，可以发出控制指针CP的各种命令，或显示在指针CP左右的源文件文本。下述命令之前的n表示一个可以指定的任选无符号数，或者可在前面有一个任选的+或－号。如前所述，英镑符号#表示65535。如果整数n是任选的但没有指定，则假定n=1。另外，如加号是任选的但没有指定，就假定其为+。

命令	功能
±B⟨cr⟩	如为“+”，指针CP移到缓冲区的顶部，否则指针CP移到缓冲区底部
±n⟨cr⟩	将指针CP移动±n个字符。如为“+”，指针CP往缓冲区前方移动n个字符；如为“－”，指针CP往后移动n个字符。源文件行中的⟨cr⟩⟨lf⟩算作两个字符
±nD⟨cr⟩	删除n个字符。如为“+”则在指针CP前删除n个字符；如为“－”则在CP后删除n个字符
±nK⟨cr⟩	以指针CP所在当前行为参考，将源文本行往前或往后删除n行，如果发出K命令时，指针CP不在当前行首，若指定为“+”，CP前的字符保留；指定为“－”，CP后的字符保留
±nL⟨cr⟩	如n=0，则指针CP移动至当前行首（如果CP未在行首）；如n≠0，则首先把指针CP移到当前行首，为“+”则指针下移n行，否则上移n行。如指定的n值太大，则CP停留在缓冲区顶部或底部
±nT⟨cr⟩	如n=0，则印出当前行首到指针CP间的内容；如n=1，则印出CP到当前行结束间的内容；如n＞1且指定为“+”时，印出当前行及其后继n－1行；若n＜1且指定为“－”，则印出前n行直到指针CP为止。Break键可用来中止长打印输出
±n⟨cr⟩	相当于±nLT，上移或下移且只打印一行

1.6 命令串

可以连续地打入任意数目的命令（只要不超过CP/M控制台缓冲区的容量），且仅在打入回车键⟨cr⟩后才执行。于是操作员可以使用CP/M控制台命令功能去处理输入命令：

命令	功能
Rubout	删除最后一个字符
Control-U	删除整行
Control-C	重新启动CP/M系统
Control-E	对长输入行不传送到缓冲区（最大128个字符）时，可用作回车键

假定内存缓冲区中已含有1.5节中所述的那些字符，且指针CP是在缓冲区最后一个字符之后，则下述命令串将会产生其右边所示的结果。

	命令串	效果	由此引起的内存缓冲区
1.	B2T⟨cr⟩	指针CP移到缓冲区开始位置并打印2行	CP NOW IS THE ⟨cr⟩⟨lf⟩
		“NOW IS THE	TIME FOR ⟨cr⟩⟨lf⟩

	TIME FOR"	ALL GOOD MEN ⟨cr⟩⟨lf⟩
2．5C0T⟨cr⟩	指针CP移5个字符，并打印当前行首到CP间的字符 "NOW I"	NOW I CP S THE ⟨cr⟩⟨lf⟩
3．2L-T⟨cr⟩	将指针下移2行，并打印前一行 "TIME FOR"	NOW IS THE ⟨cr⟩⟨lf⟩ TIME FOR ⟨cr⟩⟨lf⟩ CP ALL GOOD MEN ⟨cr⟩⟨lf⟩
4．-L*K⟨cr⟩	指针CP上移一行，并删除其后之65535行	NOW IS THE ⟨cr⟩⟨lf⟩ CP
5．I⟨cr⟩ TIME TO ⟨cr⟩ INSERT ⟨cr⟩ ↑z	插入两行	NOW IS THE ⟨cr⟩⟨lf⟩ TIME TO ⟨cr⟩⟨lf⟩ INSERT ⟨cr⟩⟨lf⟩ CP
6．-2L*T⟨cr⟩	指针CP上移2行，并打印其前面65535行 "NOW IS THE"	NOW IS THE ⟨cr⟩⟨lf⟩ CP TIME TO ⟨cr⟩⟨lf⟩ INSERT ⟨cr⟩⟨lf⟩
7．⟨cr⟩	指针CP下移一行并打印一行 "INSERT"	NOW IS THE ⟨cr⟩⟨lf⟩ TIME TO ⟨cr⟩⟨lf⟩ CP INSERT ⟨cr⟩⟨lf⟩

1.7 文本检索及修改

文本编辑ED还有这样一个命令，用来检索内存缓冲区中的字符串，命令格式如下：

$$nF c_1c_2c_3\cdots\cdots c_k \left\{\begin{matrix}\langle cr\rangle \\ \uparrow z\end{matrix}\right\}$$

其中c_1到c_k表示需要匹配的字符，其后可由回车键或control-z键表示命令行结束（如果在↑z后跟有其他命令则打入control-z）。文本编辑程序ED从字符指针CP当前位置开始，努力去匹配所有可能的K个字符，匹配可试n次。如果成功，则指针CP就立即移到该字符串中c_k字符的后面；如果匹配不成功，则指针CP停留在其初始位置不变。检索字符串可以包含control-l键，它可由符号对⟨cr⟩⟨lf⟩来取代。

下面列出F命令的用法：

命令串	效　果	结果缓冲区
1．B*T⟨cr⟩	指针CP移到缓冲区首，并打印其所有的行	CP NOW IS THE⟨cr⟩⟨lf⟩ TIME FOR⟨cr⟩⟨lf⟩ ALL GOOD MEN⟨cr⟩⟨lf⟩
2．FST⟨cr⟩	找字符串"ST"之末尾	NOW IS T CP HE⟨cr⟩⟨lf⟩
3．FI↑z0TT⟨cr⟩	找下一个字符"I"，并打印	NOW IS THE⟨cr⟩⟨lf⟩

当前行首到CP间的内容，然后打印当前行“TIME FOR”的剩余部分	TI CP ME FOR⟨cr⟩⟩1f⟩ ALL GOOD MEN⟨cr⟩⟨1f⟩

还允许使用插入命令的缩写形式，它常与F命令一起使用，完成简单的文字修改。其格式为：

I $c_1 c_2 c_3 \cdots c_n$ ↑z 或

I $c_1 c_2 c_3 \cdots c_n$ ⟨cr⟩

其中，c_1 至 c_n 是要插入的字符，如果被插入的字符串以↑z结尾，则 c_1 至 c_n 字符直接插在CP之后，然后CP移至字符 c_n 后面。如果命令以⟨cr⟩结尾，则操作相同，但在文本中插入⟨cr⟩⟨1f⟩，紧接在 c_n 之后。下面是I和F命令序列实例：

命令字符串	效果	内存缓冲区结果
BITHIS IS ↑z⟨cr⟩	将“THIS IS”插入文本开始处	THIS IS CP NOW⟨cr⟩⟨1f⟩ TIME FOR⟨cr⟩⟨1f⟩ ALL GOOD MEN⟨cr⟩⟨1f⟩
FTIME ↑z-4DIPLACE ↑z⟨cr⟩	找到“TIME”，删除之后，插入“PLACE”	THIS IS NOW THE⟨cr⟩⟨1f⟩ PLACE CP FOR⟨cr⟩⟨1f⟩ ALL GOOD MEN⟨cr⟩⟨1f⟩
3FO ↑z-3D5DICHANGES⟨cr⟩	找到第三个出现的“O”（即GOOD中第二个O），删除前面三个字符然后插入“CHANGES”	THIS IS NOW THE ⟨cr⟩⟨1f⟩ PLACE FOR⟨cr⟩⟨1f⟩ ALL CHANGES CP ⟨cr⟩⟨1f⟩
-8 CISOURCE⟨cr⟩	往回移动8个字符，并插入一行	THIS IS NOW THE⟨cr⟨⟨1f⟩ PLACE FOR⟨cr⟩⟨1f⟩ ALL SOURCE⟨cr⟩⟨1f⟩ CP CHANGES⟨cr⟩⟨1f⟩

ED还提供了一个把F和I命令组合起来的命令，可完成简单字符串的替换。其格式为：

$$nS\,c_1 c_2 \cdots c_k \uparrow z\, d_1 d_2 \cdots d_m \left\{ \begin{matrix} \langle cr \rangle \\ \uparrow z \end{matrix} \right\}$$

它与使用n次命令串

$$F\,c_1 c_2 \cdots c_k \uparrow z - k D I\, d_1 d_2 \cdots d_m \left\{ \begin{matrix} \langle cr \rangle \\ \uparrow z \end{matrix} \right\}$$

的效果完全一样。即：ED从CP的当前位置开始搜索内存缓冲区，并成功地用第二个字符串替换第一个字符串，直至缓冲区末，或直至完成n次替换。

为方便起见，ED还提供了一个类似于F的命令。它在搜索过程中能自动添加每一行并

写入。其格式为：

$$n\,N\,c_1\,c_2\cdots c_k\left\{\begin{matrix}\langle cr\rangle\\ \uparrow z\end{matrix}\right\}$$

它可以搜索整个源文件，寻找第 n 次出现的字符串 $c_1\,c_2\cdots c_k$（如果当前行缓冲器中找不到该字符串时 F 命令中止）。N 命令的操作除了当前内存缓冲区找不到给定字符串的情形外，与 F 命令完全一样。在这种情况下，整个内存的内容都被写入（即自动调用一次*W）。然后读入输入行，直至缓冲区至少装满一半或整个源文件读完。搜索按此方式一直进行下去，直至给定字符串被找到 n 次或源文件已经全部传送至临时文件。

最后一个行编辑功能叫做邻接。其格式为：

$$nJ\,c_1\,c_2\cdots c_k\uparrow z d_1\,d_2\cdots d_m\uparrow z e_1\,e_2\cdots e_q\quad\left\{\begin{matrix}\langle cr\rangle\\ \uparrow z\end{matrix}\right\}$$

它对内存缓冲区施行 n 次如下操作：从当前 CP 开始查找，若找到下一次出现的字符串 $c_1\,c_2\cdots c_k$，则插入字符串 $d_1\,d_1\cdots d_m$，并将 CP 移到 d_m 后面，删除 CP 之后的所有字符，直至（但不包括）字符串 $e_1\,e_2\cdots e_q$，仍使 CP 紧接在 d_m 之后，如果找不到 $e_1\,e_2\cdots e_q$，则不作任何删除。如果当前行为：

CP NOW IS THE TIME⟨cr⟩⟨lf⟩

则使用命令

JW ↑z WHAT ↑z ↑l⟨cr⟩

的结果为：

NOW WHAT CP ⟨cr⟩⟨lf⟩

（↑l 在寻找和替换字符串时，表示一对⟨cr⟩⟨lf⟩）。

必须注意的是，ED 要求 F、S、N 和 J 命令之后的字符数限制在100以内。

1.8 源库文件

ED 还允许使用 R 命令在编辑时插入源库文件。命令格式为：

R $f_1\,f_2\cdots f_n$ ↑z或

R $f_1\,f_2\cdots f_n$⟨cr⟩

其中 $f_1\,f_2\cdots f_n$ 是磁盘上假定文件类型为'LIB'的源文件。ED 读入规定的文件，将文件中的字符插在内存缓冲区中 CP 之后，与 I 命令类似。所以，如果操作员发命令

RMACRO⟨cr⟩

则 ED 读文件 MACRO.LIB，直至文件读完，并自动将这些字符插入内存缓冲区。

1.9 命令的重复执行

宏命令 M 允许 ED 使用者将 ED 命令组成一组，重复执行。M 命令格式为：

$$nM\,c_1\,c_2\cdots c_k\left\{\begin{matrix}\langle cr\rangle\\ \uparrow z\end{matrix}\right\}$$

其中 $c_1\,c_2\cdots c_k$ 表示一串 ED 命令，但不允许再含有一个 M 命令。如果 $n>1$，ED 将执行命令串 n 次。如果 $n=0$ 或 1，ED 重复执行命令串直至遇到出错条件（例如：使用 F 命令时到达内存缓冲区结束）。

下面这个宏命令的例子将当前缓冲区中所有出现的 GAMMA 改为 DELTA，并将修改过的每一行打印出来：

MFGAMMA ↑z－5 DIDELTA ↑z 0 TT〈cr〉或
MSGAMMA ↑zDELTA ↑z 0 TT〈cr〉

第二节　ED 出错条件

发生错误时，ED 打印出错之前读入的最后一个字符以及错误指示符：

？　不可识别命令

＞　内存缓冲区满（使用 D、K、N、S 或 W 命令移动字符时），F、N 或 S 命令串太长

＊　命令不能执行指定次数（例如使用 F 命令时）

O　使用 R 命令时打不开 LIB 文件

CP/M 还将循环冗余校验（CRC）信息与每个记录一起写入磁盘以供后面的读操作检测错误用。如果检测到 CRC 错误，CP/M 会打印出：

PERM ERR DISK d

其中 d 是当前选中的磁盘（A、B…），操作员可在控制台上打任何字符跳过这个错误（这时应检查内存缓冲区是否发生读错误），或者，如果存在后备文件，则使系统复位，修改后备文件。可先列出后备文件内容，确信它含有所需信息：

TYPE x．BAK〈cr〉

其中 x 是被编辑的文件。然后删除原始文件：

ERA x．y〈cr〉

并将后备文件重命名：

REN x．y＝x．BAK〈cr〉

这样，就可对原来的文本进行重新编辑。

第三节　控制字符与命令

下面简要列出 ED 提供的控制字符和命令：

控制字符	功　能
↑c	重新引导系统
↑e	物理〈cr〉〈lf〉（实际不在命令中送入）
↑i	逻辑 tab 标记（第 1、8、15…列）
↑l	查找和替换字符串时所用的逻辑〈cr〉〈lf〉
↑u	行删除
↑z	字符串结束符
rebout	字符删除
break	阻止命令继续执行（例如停止显示）
命　令	**功　能**
nA	添加行
±B	缓冲区顶或底
±nc	移动字符位置

±nD	删除字符
E	结束编辑，关闭文件（正常结束）
nF	找字符串
H	结束编辑，关闭并重新打开文件
I	插入字符
nJ	在相邻位置上插入字符串
±nK	删除若干行
±nL	往上/下移动若干行
nM	宏定义
nN	自动扫描，找下一个出现的字符串
O	返回至原始文件
±nP	移动并打印若干页
Q	文件不改变，退出编辑方式
R	读源库文件
nS	替换字符串
±nT	打印若干行
±U	U将小写字母替换成大写字母， -U不作转换
nW	写若干行
nz	静止
±n〈cr〉	移动并打印（±nLT）

附录A　ED 1.4功能增强

ED文本编辑增添了一些加强其编辑功能的命令。这些功能包括增加行号，询问可用空间以及改进错误显示报告。

CP/M 1.4支持下的文本编辑允许使用“V”（核对行号）命令以在行首产生绝对行号。使用V命令后行号按以下格式显示在每行开始：

nnnnn：

其中nnnnn是在1到65536范围内的绝对行号。如果内存缓冲区空，或当前行处于缓冲区末，则行号为5个空格。

用户只要在任何命令之前加一绝对行号，后面跟一冒号（与显示行号时的格式相同），就可对该行进行编辑。所以下面这条命令：

345：T

被解释为“移至绝对行345并打印该行”。要注意绝对行号只在编辑过程中产生，并不记录在文件里，特别是文件被部分删除或增加之后行号会改变。

如果用户在绝对行号前加一冒号，就可通过当前行到某一绝对行之间向前或向后的距离来编辑该行。所以下面这条命令：

：400T

解释为“从当前行一直打印到绝对行号为400的行”。将两种行编辑命令合在一起，下面这条命令：

345：：400T

解释为“移至绝对行345，然后一直打印到绝对行400”。注意，这类绝对行引用可用于任何标准 ED 命令之前。

V 命令的一个特殊用法“0 V”，可按如下格式打印出内存缓冲区的统计信息：

可用空间/总空间

可用空间指内存缓冲区中可以使用的字节数（十进制），总空间是内存缓冲区的长度。

ED 1.4通过使用“X”（传送）命令还提供一个“成组传送”功能。下面这条命令

nX

将当前行以后的 n 行传送至一名为

X$$$$$$$$.LIB

的临时文件。该文件仅在编辑过程中存在。一般来说，用户可将当前行指针定位在源文件的任何位置，然后传送若干行至临时文件。传送过去的行在该文件中一行一行接在一起。打一个读库文件命令中最简单的命令

R

又可将它们重新读出来。这时传送过去的所有行都被读入内存缓冲区。注意，X 命令并不把传送的行从内存缓冲区中删去，尽管可以在 X 命令后接着使用 K 命令。R 命令也不会清除传送过去的行文件。也就是说，使用 X 命令把行传送出去以后，可以将这些行重新读入源文件任意次。但是提供了一条

0 X

命令用于删除传送出去的行文件。

注意，一旦通过 Q 或 E 正常结束 ED 程序以后，临时 LIB 文件便被删除。如用 ctl-C 退出 ED，则已传送的 LIB 文件行仍将存在，但一般是空的（后面再调用 ED 会把临时文件删除）。

为了避免排字出错，ED 1.4要求几个容易导致严重错误的命令打成单个字符，而不能打成组合命令。以下命令

E（结束），H（开头），O（原始），Q（退出）

必须打成单字符命令。

ED 1.4还按以下格式打印错误信息：

BREAK “x” AT c

其中 x 是错误字符，c 是错误发生时的命令。

第四章　CP/M 汇编程序(ASM)

第一节　引　　言

CP/M 汇编程序从磁盘读汇编语言源文件，并以 Intel 十六进制形式产生 8080 机器语言。CP/M 汇编程序的调用是在控制台上按下面格式打入信息：

ASM 〈文件名〉或

ASM 〈文件名〉.参数

在这两种格式中都假设在磁盘中含有名为〈文件名〉.ASM 的8080汇编语言源文件。上述两种格式的差别仅在于第二种格式允许加上参数，以控制源文件存取和十六进制文件及打印文件的目的地。

在打入任何一种格式后，CP/M 汇编程序装入，并显示信息：

CP/M ASSEMBLER VER n·n

其中 n·n 是当前版本号。如果打入第一种格式，汇编程序读一个文件型为 ASM 的源文件，产生两种输出文件：

〈文件名〉·HEX 和

〈文件名〉·PRN

HEX 文件是对应原来程序的 Intel 十六进制形式的机器码，PRN 文件是一个带注释的列表文件，这个列表文件含有生成的机器码、出错标志和源程序。如果在翻译期间产生错误，将在控制台和列表文件中列出错误。

第二种格式可以给出输入文件和输出文件的磁盘名等信息。这时命令中的参数部分是一个三字母组，这个字母组规定了源文件、十六进制文件和打印文件的目的地。其形式为：

〈文件名〉·$P_1P_2P_3$

其中P_1、P_2和P_3都是单个字母。

P_1：	A，B，…，Y	存放源文件的盘名
P_2：	A，B，…，Y	接收十六进制文件的盘名
	Z	不产生十六进制文件
P_3：	A，B，…，Y	接收打印文件的盘名
	X	在控制台上列出打印文件
	Z	不产生打印文件

例如，命令“ASM X·AAA”表示源文件 X·ASM 在A 盘，产生的十六进制文件(X·HEX) 和打印文件 (X·PRN) 也放在 A 盘。如果汇编程序在 A 盘，则命令可以改成“ASM X”。命令“ASM X·ABX”表示源文件在 A 盘，十六进制文件放在 B 盘，打印文件送到控制台。命令“ASM X·BZZ”表示从 B 盘取源文件而不产生输出文件（这种命令用于检查源程序语法，使汇编程序执行时间缩短）。

源程序格式和 Intel 8080 **汇编程序及** PTSP＊1 (Processor Technology Software

Package＊1）汇编程序兼容（但没有宏功能），即CP/M汇编程序可以接受以这二种格式中任一种写的源程序。CP/M汇编程序作了一些扩充，使用更为方便。这些扩充在下面叙述。

第二节　程序格式

由汇编程序接受的汇编语言程序由若干语句组成，其语句格式是：

行号　标号　操作　操作数　;注释

格式中任一部分都可以有特殊情况，每个汇编语言语句都用回车和换行结束（但换行由ED程序自动形成），也可以用符号“！”结束。汇编程序把符号“！”作为行结束符处理，这就使得在一个物理行中可以写多个汇编语言语句，每个语句之间用“！”分隔开。

行号是一个任选的十进制整数，表示源程序的行号。使用行号是为保持和PTSP＊1汇编程序的兼容性。如果使用具有行号编辑功能的程序去构成源程序，则行号自动插入，ASM实际上不处理行号。

标号字段的形式是：标识符

或标识符：

除了某些特殊语句，标号可有可无。标识符由字母和数字组成，但第一个字符必须是字母。程序员可以自由地用标识符去标志元素（如程序号、汇编语言指令等），但标识符长度不得超过十六个字符，除美元符“$”外，所有字符均是有效的，“$”用于增进标号的易读性。在标识符中，所有小写字母都作为大写字母来处理。标识符后的“：”号是任选的，这是为了保持和Intel及PTSP＊1的兼容性。

下面是一些合法标号的实例：

X　XY　long$name

X：　Yx1：XY1：　longer$name$data：

X1Y2　X1x2　x234$5678$9012$3456：

操作字段有一个汇编语言指令或伪操作，或8080机器操作码。伪操作和机器操作码在下面叙述。

注释字段包含符号“；”后的任何字符，直至实际行结束或逻辑行结束。汇编程序读和列出这些字符，但不作其他处理。为了保持和PTSP＊1汇编程序的兼容性，CP/M汇编语言也是把用“＊”号开头的一行作为注释行来处理，只在列表中出现，而在汇编时忽略掉。而PTSP＊1汇编程序在忽略被扫描的操作数字段后面的字符时有附带作用，如想保持和Intel语言的兼容性（它允许任意的表达式）就会导致多义性。为了正确汇编，必须在注释段前加“；”号。

汇编语言程序用一系列语句组成，并用END语句结束，汇编程序将把END后面的所有语句忽略。

第三节　操作数的形式

为了完整地叙述操作码和伪操作，必须首先给出操作数字段的形式，因为它几乎在所有

语句中都要使用。操作数的表达式由简单操作数（标号、常数和保留字）和算术逻辑操作符组成的子表达式构成。表达式在汇编时由汇编程序计算求出，其值必须是一个十六位数。但最后结果有效位数不得超过想要使用的位数，即：如果表达式用在按字节的立即数传送指令中，表达式的高8位必须是零，表达式有效位数的限制由各个指令给出。

3.1 标号

上面已经说过，标号是出现在具体语句上的标识符。一般来说，标号的具体值由它后面的语句类型确定。如果标号用在产生机器码或保留内存空间的语句中（如一个MOV指令或一个DS伪操作），则这个标号的值是它所标志的程序地址。如果标号是在EQU或SET前，则其值从操作数求出。除SET语句外，一个标识符只能标识一个语句。

如果标号出现在操作数段，它的值由汇编程序替换，然后，这个值可以和其他操作数和操作符组合形成某个指令的操作数。

3.2 数值常数

数值常数是一个十六位数，其值以某个数作为基数，在一个数后跟一个基数表示数制，常用基数指示符有：

B　二进制常数（2为基）
O　八进制常数（8为基）
Q　八进制常数（8为基）
D　十进制常数（10为基）
H　十六进制常数（16为基）

其中Q作为备用的八进制基数指示符，原因是字母O容易和数字0混淆。不加基数指示符的常数被认为是十进制常数。

常数由一串数字组成，后面带一个任选基数指示符，数字必须适合于基数的范围，即二进制常数只能由0和1组成；八进制常数可以包含数字0～7；十进制常数包含十进制数；十六进制常数除包含十进制的10个数字外，还可包含A（10D）、B（11D）、C（12D）、D（13D）、E（14D）以及F（15D）。为了避免十六进制数和标识符混淆，十六进制常数的前面第一位数必须是一个十进制数字（可在前面加一个0来满足这个要求）。以这种方式构成的常数折合为二进制数后必须能容纳在16位计数器中，否则，汇编程序将截断右边多余的数。和标识符一样，常数中间可嵌入$以增加其可读性，如果基数符是小写字母，则把它们转换为大写字母。下面是一些合法的常数实例：

```
1234     1234D      1100B   1111$0000$1111$000B
1234H    0FFEH      3377O   33$77$22Q
3377O    0fe3h      1234d   0ffffh
```

3.3 保留字

在语句的操作数段已定义了几个保留字符，下面给出的是8080寄存器名，当遇到这些名时，则产生右面的值：

A　7
B　0
C　1
D　2

E	3
H	4
L	5
M	6
SP	6
PSW	6

同样，这些字母的小写形式与大写形式有着相同的值，在操作数字段也可以用机器码指令，且等价于它们的内码。对于需要操作数的指令，这时，特殊操作数成为该指令二进制位形式的一部分（如MOV A，B），而指令值（MOV）就是在任选字段为零的二进制位形式的指令（如MOV产生40H）。

如果操作数字段出现符号“$”不是嵌在标识符或常数中间，则它的值就是生成的下一个指令的地址（不包括当前逻辑行的指令）。

3.4 字符串常数

字符串常数表示一串ASCII字符，并用单引号‘ ’将字符串括起来。所有字符串必须放在当前物理行中（在字符串中可以有“！”），并且不得超过64个字符。单引号本身也可以包含在字符串内，这时好象是双引号，汇编程序读到它时和单引号一样处理。在大多数情况下，字符串的长度被限制在一个或两个字符（DB伪指令操作例外），这时字符串分别变为8位或16位二进制的值，两个字符就是一个16位常数，前面字节作为高位字节，后面字节是低位字节。

字符的值就是与之对应的ASCII码，字符串内不作大、小写转换。因此，大写字母和小写字母都可以存在。但要注意，只允许图形ASCII字符（可打印的字符）。下面是一些合法的字符串实例：

```
'A'     'AB'     'ab'     'C'
''''    'a'''    ''''''   ''',,'
'Walla  Walla  Wash.'
'She  Said  "Hello"  to me.'
'I Said  "HeIlo"  to her.'
```

3.5 算术和逻辑操作符

上面所述的操作数可以用通常代数规则组成，即由操作数、操作符及带括号的表达式组成，操作数字段认识的操作符是：

a+b	a与b的无符号算术和
a-b	a与b的无符号算术差
+b	一元加（产生b）
-b	一元减（等于0-b）
a*b	a与b的无符号值积
a/b	a与b的无符号值商
a MOD b	a除b所得余数
NOT b	b的逻辑非（所有0变成1，所有1变成0）
	其中b为16位二进制值

a AND b　　a和b按位逻辑“与”
a OR b　　a和b按位逻辑“或”
a XOR b　　a和b按位逻辑“异－或”
a SHL b　　a向左移b位，右边填零
a SHR b　　a向右移b位，左边填零

a和b表示一个简单操作数（标号、常数、保留字和一个或二个字符）或带括号的简单表达式。例如：

10+20　10h+37Q　L_1/3　(L_2+4)　SHR　3
('a' and 5fh) +'0'　('B'+B)　OR　(PSW+M)
(1+(2+C))　SHR　(A-(B+1))

这些计算都是在汇编时作为16位二进制数进行无符号操作的，因此－1作为0－1计算，其结果是0ffffh（即全为1），结果表达式必须适合所用的操作码。例如，如果在ADI（加立即数）指令中有一个表达式，则表达式值的高8位必须是零，因此，“ADI－1”操作就会产生错误信息（－1变为0ffffh不能表示为8位值），而“ADI（－1）AND 0FFH”可被汇编程序接受，因为AND操作使表达式的高位为零。

3.6　操作符的优先权

为使程序员方便，ASM假设操作符有相对的优先权，这就允许程序员在写表达式时，可以不用多层括号。在结果表达式中按如下操作符优先级给出假想括号。在无括号表达式中，操作符使用顺序列表如下，表中的优先级别从高到低，同一行的操作符优先级别相同，当在表达式中遇到时，从左到右进行运算。下面是操作符表：

* / MOD　SHL　SHR
－　+
NOT
AND
OR　XOR

汇编程序在读到下例中左边的表达式时，就把它们翻译成右边的带括号表达式：

a*b+c	(a*b) +c
a+b*c	a+ (b*c)
a MOD b*c SHL d	((a MOD b)*c) SHL d
a OR b AND NOT c+d SHL e	a OR (b AND (NOT (c+ (d SHL e))))

用括号括起来的表达式优先于任何无括号的算式，如将上面最后一个表达式重写成：

(a OR b) AND (NOT c) +d SHL e

则汇编程序将其翻译成：

(a OR b) AND ((NOT c) + (d SHL e))

第四节　汇编程序指令

汇编程序指令用于在汇编期间，给标号设置特定值，执行条件汇编，定义内存区域及指

定程序的开始地址。每个汇编指令都用一个伪操作来表示，这些伪操作出现在命令行的操作段。汇编程序能识别的伪操作是：

ORG　设置程序或数据的开始地址
END　表示程序结束，后面可带开始地址
EQV　数值“等于”
SET　数值“置为”
IF　开始条件汇编
ENDIF结束条件汇编
DB　定义数据字节
DW　定义数据字
DS　定义数据存储域

下面给出各个伪操作的详细说明。

4.1 ORG 指令

ORG语句格式是：

标号 ORG 表达式

其中标号是任选的程序标号，表达式是16位二进制表达式，由ORG语句前面定义的操作数组成。汇编程序按表达式所决定的位置开始生成机器码。对于一个具体的程序，ORG语句的数量可以是任意的，但因汇编程序不检查ORG后的表达式值，故无法保证程序员给出的值是否超出内存区。注意，大多数在CP/M下写的程序都从ORG语句开始，其格式为：

ORG　100H

这就使得产生的机器码是在CP/M的暂驻程序区的基地址开始。如果ORG语句有标号，这个标号的值由表达式赋给（这个标号可在其他语句的操作数段使用来表示这一表达式）。

4.2 END 指令

END语句在汇编语言程序中是任选的，但如果有END语句，则它必须是程序的最后一个语句（它后面的所有语句在汇编时均忽略）。

END语句可有下面两种格式：

标号　END

标号　END 表达式

其中标号也是任选的。如果用第一种格式，汇编过程停止，且汇编假定程序的开始地址是0000；如果用第二种格式，则求出表达式的值，该值即成为程序的开始地址（在汇编结束时，Intel格式的机器码‘HEX’文件的最后一个记录里包含了这个地址）。大多数CP/M汇编语言程序都用下面这个语句结束：

END　100H

这就给定了蕴含开始地址是100H（暂驻程序区的开始）。

4.3 EQU 指令

EQU语句说明该语句两边的命名具有相同的值。格式是：

标号　EQU　表达式

这里标号是必须有的，且不能再用该标号标识其他语句。汇编程序求出表达式的值，并将该值赋给标号字段给出的标识符。标识符通常是一个名，是用人们更习惯的方法描述一个值，

且该名在整个程序中用作某些函数的参变量。例如，从电传机上接收在一特定输入口的一个数据，接着通过下一个输出口送到这个电传机上。在某种硬件条件下，可用下面的一些等值语句规定输入或输出口：

```
TTYBASE  EQU  10H          ; TTY的基本口
TTYIN    EQU  TTYBASE      ; 输入口
TTYOUT   EQU  TTYBASE+1    ; 输出口
```

接着就可以用下面的程序存取电传数据：

```
IN   TTYIN                 ; 从电传机上读数据到寄存器A
.....
OUT  TTYOUT                ; 把寄存器A的内容送到电传机
```

这比用绝对I/O口写的程序更易读。

如果重新定义电传通信口是7FH而不是10H，则只需把第一个语句改写成：

```
TTYBASE  EQU  7FH          ; TTY的基本口
```

而不需要改变任何其他语句就可以重新汇编。

4.4 SET指令

SET语句和EQU语句类似，其格式是：

标号 SET 表达式

这里标号可在同一程序中的另一个SET语句中出现。而EQU的标号则不能出现在其他EQU语句中。执行时计算出该表达式的值并作为该标号的当前值。EQU语句只能赋予某标号一个单值，而SET语句可以把几个值赋给一个标号。SET语句主要用于控制条件汇编。

4.5 IF和ENDIF指令

IF和ENDIF语句规定了一个程序段，这个程序段在汇编时可包含在程序中或不包含在程序中，其格式是：

```
IF  表达式
语句*1
语句*2
.....
语句*n
ENDIF
```

汇编程序一遇到IF语句就求出其后表达式的值（该表达式所有的操作数必须在IF语句之前定义）。如果其值不等于零，就汇编语句*1到语句*n；如果其值等于零，就不汇编该程序段。条件汇编常常用于写一个通用的程序，它包括运行时各种可能的环境，而对某一具体程序，汇编时只选用其中一部分。下面的程序是和电传或CRT控制台通讯程序的一部分，在汇编语言程序前给某个值就选择某种设备：

```
TRUE     EQU  0FFFFH      ; DEFINE VALUE OF TRUE
FALSE    EQU  NOT TRUE    ; DEFINE VALUE OF FALSE
;
TTY      EQU  TRUE        ; TRUE IF TTY, FALSE IF CRT
;
```

```
TTYBASE EQU  10H          ; BASE OF TTY I/O PORTS
CRTBASE EQU  20H          ; BASE OF CRT I/O PORTS
        IF   TTY          ; ASSEMBLE RELATIVE TO TTYBASE
CONIN   EQU  TTYBASE      ; CONSOLE INPUT
CONOUT  EQU  TTYBASE+1    ; CONSOLE OUTPUT
        ENDIF
;
        IF   NOT TTY      ; ASSEMBLE RELATIVE TO CRTBASE
CONIN   EQU  CRTBASE      ; CONSOLE INPUT
CONOUT  EQU  CRTBASE+1    ; CONSOLE OUTPUT
        ENDIF
        ...
        IN   CONIN        ; READ CONSOLE DATA
        ...
        OUT  CONOUT       ; WRITE CONSOLE DATA
```

在这种情况下，对连接电传机的环境汇编该程序，基本口在10H。如果改成：

```
TTY  EQU  FALSE
```

则对连接CRT环境汇编该程序，基本口在20H。

4.6 DB指令

DB语句允许程序员用单精度（单字节）格式定义开始的存储区域，其格式是：

标号 DB e#1, e#2, …, e#n

其中e#1到e#n可以是一个表达式，但其值不得超过二进制8位数（高8位为零），也可以是一个长度不超过64个字符的ASCII字符串。在一个命令行中，对表达式的数目没有限制。汇编时，汇编程序求出表达式的值，并顺序放到机器码文件中（跟在汇编生成的最后一个地址后面），字符串同样顺序放入存储器，从第一个字符开始，到最后一个字符结束。多于两个字符的字符串不能在更复杂的表达式中作为操作数（即必须用逗点分开）。注意，ASCII字符放在内存中时，总是把奇偶位置零（即最高位为零），且在字符串中没有小写到大写的转换。利用任选的标号可以引用程序其余部分的数据区。下面是一些合法的DB语句实例：

```
data:   DB 0, 1, 2, 3, 4, 5
        DB data and offH, 5, 377Q, 1+2+3+4
signon: DB'please type your name', cr, lf, 0
        DB'AB'SHR 8, 'C', 'DE'AND 7FH
```

4.7 DW指令

DW语句和DB语句类似，差别是DW定义双精度（双字节）字存储区。其格式是：

标号 DW e#1, e#2, …, e#n

其中e#1到e#n是表达式，表达式的值不得超过二进制的16位数。对ASCII字符串允许长度为一个或二个字符，但不得超过二个字符，数据存储按8080处理器格式，即低位字节先存放，高位字节后存放。下面是一个例子：

```
doub: DW  0ffefh, doub+4, signon-S, 255+255
```

```
DW  'a', 5, 'ab', 'CD', 6 shl 8 or 11b
```

4.8 DS指令

DS语句用于保留一个未初始化的存储区，其格式是：

标号 DS 表达式

其中标号是任选的，汇编程序遇到DS语句时，在保留存储区后开始生成代码串。因此，上面给出的DS语句和下面两个语句的组合作用完全相同：

```
标号  EQU  $              ;  当前代码位置
      ORG  $ + 表达式     ;  跳过保留区
```

第五节 操作码

汇编语言操作码是汇编语言程序的基本部分，并构成指令的操作段。一般来说，ASM能识别所有的8080微计算机标准助记符（这些助记符在Intel 8080汇编语言编程手册中详细列出）。每个输入行可加标号，如果有标号，标号的值就是该指令的地址。下面给出主要操作符，在各个操作符中：

e3 代表3位二进制数0～7，它可以是预先定义的寄存器A、B、C、D、E、H、L、M、SP或PSW

e8 代表8位二进制数0～255

e16代表16位二进制数0～65535

这些数可由操作数和操作符任意组合而成。有时，操作数受指令的限制（如PUSH指令），当遇到这些指令时，再给以解释。

下面给出每个操作码的一般形式，并列举一个例子，附上简单的注解和特殊限制。

5.1 转移、调用和返回

转移、调用和返回指令根据8080微处理器CPU的条件标志，有如下几种形式：

```
JMP   e16   JMP   L1        无条件转移到标号
JNZ   e16   JNZ   L2        不等于零时转移到标号
JZ    e16   JZ    100H      等于零时转移到标号
JNC   e16   JNC   L1+4      无进位时转移到标号
JC    e16   JC    L3        有进位时转移到标号
JPO   e16   JPO   $+8       奇数时转移到标号
JPE   e16   JPE   L4        偶数时转移到标号
JP    e16   JP    GAMMA     结果为正转移到标号
JM    e16   JM    a1        结果为负转移到标号
CALL  e16   CALL  S1        无条件调用子程序
CNZ   e16   CNZ   S2        非零时调用子程序
CZ    e16   CZ    100H      零时调用子程序
CNC   e16   CNC   S1+4      无进位时调用子程序
CC    e16   CC    S3        有进位时调用子程序
CPO   e16   CPO   $+8       奇数时调用子程序
```

CPE	e16	CPE	S4	偶数时调用子程序
CP	e16	CP	GAMMA	结果为正时调用子程序
CM	e16	CM	b1$C2	结果为负时调用子程序
RST	e3	RST	0	程序重新开始，除字节数不同外，与CALL 8*e3等价
RET				从子程序返回
RNZ				不等于零则返回
RZ				等于零则返回
RNC				无进位则返回
RC				有进位则返回
RPO				奇数则返回
RPE				偶数则返回
RP				结果为正则返回
RM				结果为负则返回

5.2 立即操作数指令

有几种指令可以向单精度寄存器、双精度寄存器和单精度存储单元放常数，还有几条指令可在累加器（寄存器A）上进行立即数算术或逻辑运算，这些指令是：

MVI	e3,e8	MVI	B,255	将立即数存入寄存器A，B，C，D，E，H，L或M
ADI	e8	ADI	0FFH	加立即数到A（不带进位）
ACI	e8	ACI	0FFH	加立即数到A（带进位）
SUI	e8	SUI	L+3	从A减去立即数（不带借位）
SBI	e8	SBI	L AND 11B	从A减去立即数（带借位）
ANI	e8	ANI	$ AND 7FH	A和立即数进行逻辑“与”
XRI	e8	XRI	1111$0000B	A和立即数进行逻辑“异-或”
ORI	e8	ORI	L AND 1+1	A和立即数进行逻辑“或”
CPI	e8	CPI	'a'	A和立即数比较（与SUI的差别是不改变寄存器A的内容）
LXI	e3,e16	LXI	B,100H	将扩展立即数存到寄存器对（e3必须是B，D，H或SP）

5.3 增量和减量指令

8080提供了对单精度和双精度寄存器进行增量和减量运算的指令。这些指令是：

INR	e3	INR	E	单精度寄存器（A、B、C、D、E、H、L、M）增1
DCR	e3	DCR	A	单精度寄存器（A、B、C、D、E、H、L、M）减1
INX	e3	INX	SP	双精度寄存器（B、D、H或SP）加1
DCX	e3	DCX	B	双精度寄存器（B、D、H或SP）减1

5.4 数据传送指令

下面给出的是一些把数据从存储器传送到CPU和从CPU传送到存储器的指令：

MOV	e3,e3	MOV	A,B	把最右边单元的数据传送到最左边单元（e3是A、B、C、D、E、H、L或M之一），但左右两边不能是同一单元（例如，不允许MOV M,M）

LDAX	e3	LDAX	B	从给定的地址取数到寄存器A（e3一定是B或D）
STAX	e3	STAX	D	把寄存器A的数据送到给定地址(e3一定是B或D)
LHLD	e16	LHLD	L1	从e16单元取数到HL（双精度）
SHLD	e16	SHLD	L5+x	把HL的内容送到e16单元（双精度）
LDA	e16	LDA	Gamma	取e16地址的信息到A
STA	e16	STA	X3-5	将A的内容送到内存e16单元
POP	e3	POP	PSW	从堆栈中取信息到寄存器对，设置栈指针SP（e3必须是B、D、H或PSW）
PUSH	e3	PUSH	B	把寄存器对的内容送入堆栈，设置栈指针SP（e3必须是B、D、H或PSW）
IN	e8	IN	0	从e8口取数据装入寄存器A
OUT	e8	OUT	255	把寄存器A的数据送到e8口
XTHL				栈顶的数据和HL的数据互换
PCHL				把HL的数据填入程序计数器PC
SPHL				把HL的数据填入栈指针SP
XCHG				DE寄存器对和HL寄存器对互换

5.5 算术和逻辑单元运算

在单精度累加器上进行算术和逻辑操作的指令是：

ADD	e3	ADD	B	把e3确定的寄存器内容加到累加器（e3必须是A、B、C、D、E、H、L之一），不带进位
ADC	e3	ADC	L	将寄存器内容加到A，带进位，e3定义同上
SUB	e3	SUB	H	从A中减去e3确定的寄存器内容（不带进位），e3定义同上
SBB	e3	SBB	2	从A中减去寄存器内容，e3定义同上
ANA	e3	ANA	1+1	A和寄存器逻辑“与”，e3定义同上
XRA	e3	XRA	A	A和寄存器逻辑“异-或”，e3定义同上
ORA	e3	ORA	B	A和寄存器逻辑“或”，e3定义同上
CWP	e3	CMP	H	A和寄存器比较，e3定义同上
DAA				对最后算术逻辑运算结果进行十进位校正
CMA				对寄存器A求补
STC				置进位标志为1
CMC				进位标志求补
RLC				循环左移，最高位送入进位位
RRC				循环右移，最低位送入进位位
RAL				A寄存器带进位循环左移
RRL				A寄存器带进位循环右移
DAD	e3	DAD	B	双精度加，把e3确定的寄存器内容加到HL（e3必须是P、D、HSP中之一）

5.6 控制指令

余下的四条指令是控制指令，列出如下：

HLT 暂停8080处理器

DI 屏蔽中断

EI 开中断

NOP 空操作

第六节 错误信息

当汇编语言程序出现错误时，汇编程序以单字符在源程序行的左边给出标志，同时将错误行回送到控制台，这样，程序员无须检查源程序列表就可知道错误出在哪里。错误标志码是：

D 数据错：数据语句的内容不能放进规定的数据区

E 表达式错：表达式格式错，汇编时不能计算

L 标号错：在这个地方不能出现标号（可能是重复标号）

N 不执行，可能是未来版本才有的功能（如宏操作）。在本版本中已标志，但不能执行

O 溢出：表达式太复杂，以致无法计算，应简化之

P 标号值错：一个标号在程序的两次扫描中取值不同

R 寄存器错：规定为寄存器的值和操作码不相容

V 值错：表达式中的操作数形式不恰当

由于终端错误条件，可能打印下列错误信息：

NO SOURCE FILE PRESENT	汇编命令所规定的文件不在盘上
NO DIRECTORY SPACE	磁盘目录满，应删除不必要的文件，重试
SOURCE FILE NAME ERROR	源文件名错（例如指定“？”为文件名）
SOURCE FILE READ ERROR	汇编程序不能读出源文件，可执行TYPE去确定错误所在
OUTPUT FILE WRITE ERROR	输出文件不能写入磁盘，可能是磁盘满，清除无用文件，重试
CANNOT CLOSE FILE	不能关闭输出文件，检查是否写保护

第七节 实 例

下面的实例说明在开发汇编语言程序时，汇编程序和调试程序的交互作用。

```
ASM SORT

CP/M ASSEMBLER-VER 1.0

015C
```

```
003H USE FACTOR
END OF ASSEMBLY

DIR SORT.*

SORT    ASM
SORT    BAK
SORT    PRN
SORT    HEX
A>TYPE SORT. PRN


                ;       SORT PROGRAM IN CP/M ASSEMBLY LANGUAGE
                ;       START AT THE BEGINNING OF THE TRANSIENT
                          PROGRAM AREA
0100                    ORG  100H

0100  214601  SORT:  LXI  H,SW  , ADDRESS SWITCH TOGGLE
0103  3601           MVI  M,1   , SET TO 1 FOR FIRST ITERATION
0105  214701         LXI  H,I   , ADDRESS INDEX
0108  3600           MVI  M,0   , I=0
                ;
                ;       COMPARE I WITA ARRAY SISE
010A  7E      COMP:  MOV  A,M   , A REGISTER=I
010B  FE09           CPI  N-1   , CY SET IF I<(N-1)
010D  D21901         JNC  CONT  , CONTINUE IF I<(N-2)
                ;
                ;       END OF ONE PASS THROUGH DATA
0110  214601         LXI  H,SW  , CHECK FOR ZERO SWITCHES
0113  7EB7C20001     MOV A,M! ORA A! JNZ SORT , END OF SORT IF
                                                SW=0
                ;
0118  FF             RST 7      , GO TO THE DEBUGGER INSTEAD
                                  OF REB
                ;
                ;       CONTINUE THIS PASS
                ;       ADDRESSING I, SO LOAD AV(I) INTO REGISTERS
0119 5F16002148 CONT: MOV E,A! MVI D,0! LXI H,AV! DAD D! DAD D
0121  4E792346       MOV C,M! MOV A,C! INX H! MOV B,M
```

```
                ;       LOW ORDER BYTE IN A AND C, HIGH ORDER
                          BYTE IN B
                ;
                ;       MOV H AND L TO ADDRESS AV(I+1)
0125  23                IHX   H
                ;
                ;       COMPARE VALUE WITH REGS CONTAINING
                          AV(I)
0126  965778239E        SUB M! MOV D, A! MOV A,B! INX H! SBB M
                              ; SUBTRACT
                ;
                ;       BORROW SET IF AV(I+1)>AV(I)
012E  DA3F01            JC    INCI    ; SKIP IF IN PROPER ORDER
                ;
                ;       CHECK FOR EQUAL VALUES
012E  B2CA3F01          ORA D! JZ INCI ; SKIP IF AV(I)=AV(I+1)
0132  56702B5E          MOV D,M! MOV M,B! DCX H! MOV E,M
0136  712B722B73        MOV M,C! DCX H! MOV M,D! DCX H! MOV M,E
                ;
                ;       INCREMENT SWITCH COUNT
013B  21460134          LXI H,SW! INR M
                ;
                ;       INCREMENT I
013F  21470134C3 INCI:  LXI H,I! INR M! JMP COMP
                ;
                ;       DATA DEFINITION SECTION
0146  00        SW:     DB   0          ; RESERVE SPACE FOR SWITCH
                                          COUNT
0147            I:      DS   1          ; SPACE FOR INDEX
0148  050064001E AV:    DW  5, 100, 30, 50, 20, 7, 1000, 300, 100, -32767
000A  =         N       EQU ($-AV)/2 ;COMPUTE N INSTEAD OF PRE
015C                    END
A>TYPE SORT. HEX


:10010000214601360121470136007EFE09D2190140
:100110002146017EB7C20001FF5F16002148011983
:10012000194E79234623965778239EDA3F01B2CAA7
:100130003F0156702B5E712B722B732146013421C7
```

```
:07014000470134C30A01006E
:10014800050064001E08320014000700E8032C01BB
:040158006400018BE
:0000000000
A>DDT SORT. HEX

16K DDT VER 1.0
HEXT PC
015C 0000
-XP

P=0000 100

-UFFFF

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100
  LXI H,0146 * 0100
-T10

C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0100
  LXI M,0146
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0103
  MVI H,01
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0105
  LXI H,0147
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0147  S=0100  P=0108
  MVI M,00
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0147  S=0100  P=010A
  MOV A,M
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=010B
  CPI 09
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=010D
  JNC 0119
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=0110
  LXI H,0146
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0146  S=0100  P=0113
  MOV A,M
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0114
  ORA A
```

```
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0115
  JNZ0100
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0100
  LXI H,0146
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0103
  MVI M,01
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0146  S=0100  P=0105
  LXI H,0147
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0147  S=0100  P=0108
  MVI M,00
C0Z0M0E0I0  A=01  B=0000  D=0000  H=0147  S=0100  P=010A
  MOV A,M*B10B
-A10D

010D    JC 119
0110
-XP

P=010B 100

-T10
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=0100
  LXI H,0146
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0146  S=0100  P=0103
  MVI M,01
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0146  S=0100  P=0105
  LXI H,0147
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=0108
  MVI M,00
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=010A
  MOV A,M
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=010B
  CPI 09
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=010D
  JC 0119
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=0119
  MOV E,A
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=011A
  MVI D,00
```

```
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0147  S=0100  P=011C
  LXI H,0148
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0148  S=0100  P=011F  DAD D
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0148  S=0100  P=0120  DAD D
C0Z0M0E0I0  A=00  B=0000  D=0000  H=0148  S=0100  P=0121
  MOV C,M
C0Z0M0E0I0  A=00  B=0005  D=0000  H=0148  S=0100  P=0122
  MOV A,C
C0Z0M0E0I0  A=05  B=0005  D=0000  H=0148  S=0100  P=0123  INX H
C0Z0M0E0I0  A=05  B=0005  D=0000  H=0149  S=0100  P=0124
  MOV B,M*0125
-L100

0100    LXI   H,0146
0103    MVI   M,01
0105    LXI   H,0147
0108    MVI   M,00
010A    MOV   A,M
010B    CPI   09
010D    JC    0119
0110    LXI   H,0146
0113    MOV   A,M
0114    ORA   A
0115    JNZ   0100
-L

0118    RST   07
0119    MOV   E,A
011A    MVI   D,00
011C    LXI   H,0148

-G,118
*0127
-T4

C0Z0M0E0I0  A=38  B=0064  D=0006  H=0156  S=0100  P=0127
  MOV D,A
C0Z0M0E0I0  A=38  B=0064  D=3806  H=0156  S=0100  P=0128
  MOV A,B
```

```
C0Z0M0E0I0  A=00  B=0064  D=3806  H=0156  S=0100  P=0129  INX  H
C0Z0M0E0I0  A=00  B=0064  D=3806  H=0156  S=0100  P=012A
  SBB M*012B
-D148

0148  05  00  07  00  14  00  1E  00  ........
0150  32  00  64  00  64  00  2C  01  E8  03  01  80  00  00  00  2.D.D.,.....
0160  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  00  ......

DDT SORT. HEX

16K DDT VER 1.0
NEXT PC
015C 0000
-XP

P=0000  100

-L10D

010D JNC  0119
0110 LXI  H,0146

-A10D

010D JC   119

0110

-L100

0100 LXI  H,0146
0103 MVI  M,01
0105 LXI  H,0147
0108 MVI  M,00

-A103

0103 MVI  M,0
```

```
0105

-^C

SAVE 1 SORT. COM

A>DDT SORT. COM

16K DDT VER 1.0
NEXT PC
0200 0100
-G

*0118
-D148

0148 05 00 07 00 14 00 1E 00 ..........
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2.D.D., ..........
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..........
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ..........
ED SORT. ASM

*N,0^Z 0TT
        MVI     M,0     ; I=0
*-
        LXI     H,I     ; ADDRESS INDEX
*-
        MVI     M,1     ; SET TO 1 FOR FIRST ITERATION
*KT
        LXI     H,I     ; ADDRESS INDEX
*I
        MVI     M,0     ; ZERO SW
*T
        LXI     H,I     ; ADDRESS INDEX
*NJNC^Z0T
        JNC*T
        CONT            ; CONTINUE IF I<=(N-2)
*-2DIC^Z0LT
        JC      CONT    ; CONTINUE IF I<=(N-2)
```

```
*E

ASM SORT. AAZ

CP/M ASSEMBLER - VER 1.0

015C
003H USE FACTOR
END OF ASSEMBLY

DDT SORT. HEX

16K DDT VER 1.0
NEXT PC
015C 0000
-G100

*0118

-D148

0148 05 00 07 00 14 00 1E 00 ..........
0150 32 00 64 00 64 00 2C 01 E8 03 01 80 00 00 00 00 2.D.D.,.....
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-

-G0

ASM SCAN. AAZ

CP/M ASSEMBLER - VER 1.0
0122
002H USE FACTOR
END OF ASSEMBLY

DDT SCAN. HEX

16K DDT VER 1.0
```

```
NEXT PC
0121 0000
-L116

0116 JMP    0000
0119 STAX   B
011A NOP
011B INR    B
-

-G100,116

*0116
-D121

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 ..”!..^.W.#..X.
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00.:···) ·····
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ···············

-

-G0
```

第五章　CP/M动态调试工具（DDT）

第一节　引　　言

DDT程序允许对CP/M环境下产生的程序进行会话式的检查和调试。调试程序通过在CP/M命令级打入下述几种命令来启动：

DDT

DDT文件名.HEX

DDT文件名.COM

其中，文件名为被装入的待检查程序名，HEX表示十六进制文件，COM表示二进制可执行文件。在上述情况下，DDT程序从磁盘引入内存，放在控制台命令处理程序区(CCP)，紧跟在磁盘操作系统（BDOS）之下。BDOS开始地址（位于005H单元JMP 指令地址字段）则被改变以反映减少了的暂驻程序区（TPA）。

第二种和第三种命令除执行与第一种命令相同的功能外，还读入一个十六进制或二进制文件，其作用相当于下面三个命令的组合：

DDT

I〈文件名〉.HEX或I〈文件名〉.COM

R

其中，I和R命令建立和读入指定要测试的程序（详见I和R命令解释）。

启动之后，DDT印出下面格式的信息：

nnK　DDT-s　VBRm.m

这里nn是内存大小（必须和正在使用的CP/M系统相符合），s是假定的硬件系统，对应如下代码：

D-Digital Research标准版本

M-MDS版本

I-IMSAI标准版本

O-Omron系统

S-Digital Systems标准版本

m.m表示版本号

在上述信息之后，DDT给出提示符“-”，等待从控制台输入命令。操作员可打入任何单字符命令，用回车键结束，并开始执行该命令。每个输入行均可用标准CP/M控制键进行行编辑。

rubout　删去最后打入的一个字符

ctl-U　删去整行，准备再输入

ctl-C　系统重新引导

命令长度可达32个字符（自动回车字符作为第33个字符插入）。首字符决定了命令类

型：

A 输入汇编助记符和操作数
D 用十六进制和ASCII码显示内存
F 将常数填入内存
G 开始执行带有任意断点的程序
I 建立一个标准的输入文件控制块
L 使用汇编助记符列出内存内容
M 把内存中一段信息从源程序区传送到目的程序区
R 读入要检验的程序
S 替换内存值
T 跟踪程序执行
U 不跟踪程序监控
X 检查并可任选地改变CPU状态

在有些情况下，命令字符后面可跟以0、1、2或3个十六进制数值，这些数用逗号或一个空格字符分开。所有的DDT数字输出均为十六进制形式。在任何情况下，都要在命令末尾打入回车键后，该命令才开始执行。

在调试运行的任何时候，操作员都可以用打入ctl-C或G0（转移到000H单元）来停止DDT的执行，并用SAVE命令保存当前内存映象。SAVE命令形式如下：

SAVE n 文件名.COM

这里n是保存到磁盘上的页数。块数由取出最高装入地址的高位字节并转换为十进制数来确定。例如，如果在暂驻程序区中最高地址是1234H，则页数是12H或十进制数18。因此在调试运行期间，操作员可打入ctl-C，返回控制台处理程序级，接着打入：

SAVE 18X.COM

内存映象作为X.COM文件存入软盘，而且只打入文件名X就可直接执行。如果需要进一步测试，可以打入

DDT X.COM

将内存映象重新调回内存，原来保存的程序可以重新装入内存中从100H单元开始的18页（12FFH）。机器状态不是COM文件的一部分，因此，为了正确测试，该程序必须重新开始。

第二节 DDT 命令

本节详述DDT的各条命令。在每次送入命令之前，操作员必须等待提示符“-”，如果控制传送给一个被测试程序，且程序未到达其断点，则可以通过从前面板执行RST7指令将控制返回DDT（注意，如果程序正在执行T或U命令，则应使用rubout键）。在解释每条命令时，命令字母后面有时带有用逗号分开的数，这些数用小写字母表示。它们都是十六进制码，长度为1到4位（多于4位的数字被自动地右截断）。

许多命令操作依赖于被测试程序所对应的“CPU状态”，CPU状态包含被测试程序的寄存器，且在开始时，除程序计数器（P）和栈指针（S）外的所有寄存器和标记位均预置

为零。而P和S的缺省值是100H。如果装入文件是HEX型，则程序计数器随后被置为该文件最后一个记录的开始地址（见I和R命令）。

2.1 A（汇编）命令

DDT允许使用A命令将一行汇编语言程序插入到当前内存映象中，A命令形式为：

As

其中s是汇编程序行的十六进制开始地址。DDT 用下一条被填入指令的地址提示控制台并读控制台，查找汇编语言记忆符（见Intel8080汇编语言参考卡记忆符表），跟着是寄存器引用名和绝对十六进制形式的操作数。以后每次读控制台之前，先打印出后续的装入地址。当从控制台输入第一个空行时，A命令结束。

当完成汇编语言输入时，操作员可使用DDT反汇编（见L命令）检查该内存段。

注意，DDT的汇编/反汇编部分可被正在测试的暂驻程序覆盖，在这种情况下，当使用A和L命令时，DDT程序将回答出错信息。

2.2 D（显示）命令

D命令允许操作员以十六进制和ASCII格式检查内存的内容。命令形式为：

D

Ds

Ds,f

第一种情况下，内存从当前显示的地址（100H）开始连续显示16行，每个显示行取如下形式：

aaaa bb bb……bb bb cc……cc……cc

这里aaaa是十六进制的显示地址，bb代表从内存aaaa地址开始的数据。右边给出从aaaa开始的 ASCII 字符（由一连串 c 表示）。非图形字符显示为句号“ . ”。注意，大小写字母都显示，但对于只支持大写字母的控制台设备则只出现大写字母。每个显示行显示16字节的数据值，但第一行可能被截断，以使下一行开始地址为16的倍数。

D命令的第二种形式与第一种形式相似，只是把显示的开始地址设置为s。第三种形式是从地址s开始连续显示到地址f。在所有情况下，显示地址均被设置为该显示命令没有显示到的第一个地址，使得连续发出D命令能够实现连续显示而不用指明地址。

按下rubout键可使过长的显示中断。

2.3 F（填充）命令

F命令形式如下：

Fs,f,c

这里s是开始地址，f是结束地址，c是十六进制字节常数。该命令的作用如下：DDT将常数c存放在地址s，取增量值s，对照f进行检查。如果s超过f则操作结束，否则重复该操作。因此，填充命令可用于将内存块设置为一个指定的常数值。

2.4 G（执行）命令

用G命令开始程序的执行，可带两个任选的断点地址。G命令形式如下：

G

Gs

Gs,b

Gs,b,c

G,b

G,b,c

第一种形式是在当前机器状态下，从程序计数器的当前值开始执行该程序测试，且不设置断点（DDT重新得到控制的唯一办法是通过执行RST7命令）。可以通过打入X或XP命令检查当前程序计数器。

第二种形式与第一种形式相似，只是在开始执行之前，当前机器状态中的程序状态计数器设置为地址s。

第三种形式与第二种形式相同，只是当碰到地址b时，停止执行该程序（b必须在被测试的程序区）。当遇到断点时，单元b的指令不执行。

第四种形式与第三种形式相同，只是指定了两个断点，一个在b，另一个在c。碰到任何一个断点均造成执行停止，且两个断点依次被清除。

最后两种形式从当前机器状态取程序计数器为开始地址，分别设置一个或两个断点。

程序从开始地址实时执行直到下一个断点。即从开始到指定的断点地址之间，DDT不进行干预。因此，如果被测试程序未达到断点，若不执行RST7指令，控制不能返回到DDT。当碰到断点时，DDT停止执行，并印出

*d

这里d是停止地址。可以使用X（检查）命令检查该点的机器状态。操作员指定的断点必须与G命令开始的程序计数器地址不同，因此，如果当前程序计数器是1234H，则命令：

G，1234和

G400，400

都会产生一个立即断点而不执行任何指令。

2.5 I（输入）命令

I命令允许操作员将一个文件名插入到5CH单元的缺省文件控制块中（CP/M为暂驻程序建立的文件控制块放在该单元，见“CP/M接口指南”），如缺省的FCB已由控制台命令处理程序传送，则它可以由被测试程序使用。注意，该文件名还被DDT用于读附加的HEX和COM文件。I命令形式是：

I〈文件名〉或

I〈文件名〉·〈文件型〉

如果用第二种形式，且文件型为HEX或COM，则可用后继R命令读纯二进制或十六进制格式的机器码（见R命令）。

2.6 L（列表）命令

L命令用于列出某特定程序区的汇编语言记忆符，命令格式为：

L

Ls

Ls,f

第一种命令形式从当前列表地址开始列出12行反汇编机器码。第二种形式将列表地址设置为s，然后列出12行机器码。第三种形式从地址s开始，一直到f的反汇编机器码全部列出。在上述三种形式中，列表地址均设置在下一个未列出的单元，以准备使用下一条L命令。在

碰到执行断点时，列表地址设置为程序计数器的当前值（见G和T 命令）。同样在列表过程中，可以使用rubout键中断过长的打印输出。

2.7 M（移动）命令

M命令允许将程序或数据区成组地从内存的一个地址单元移到另一单元，命令形式为：

Ms,f,d

其中，s是移动的开始地址，f是移动的最后地址，d是目的地址。数据首先从s移动到d，然后两个地址分别加一个增量，如果s超出f，则移动操作停止，否则重复进行该操作。

2.8 R（读）命令

R命令和I命令一起使用，将COM文件和HEX文件从软盘读入暂驻程序区，准备调试运行。命令形式如下：

R

Rb

其中b是任选的地址偏移，在装入时每个程序和数据地址都要加上这个偏移值。装入操作不得重写000H到0FFH单元（即内存首页）中的任何系统参数，如果略去b，则假定b = 0000。R命令要求前面有一个I命令，指定HEX或COM文件名。每个HEX记录的装入地址都是从每一个记录得到，但对COM文件，则取假设的装入地址100H。注意，在I命令后面可跟着发出任意个R命令，重读被测试程序，假定被测试程序不破坏5CH单元的缺省域。进一步假定，任何指定为COM型的文件均包含纯二进制形式的机器码（由 LOAD 或 SAVE 命令建立），且所有其他文件均包含Intel十六进制格式的机器码（例如由 ASM 命令产生的文件）。

再看一下命令：

DDT〈文件名〉·〈文件型〉

该命令启动DDT程序，等效于命令

DDT

-〈文件名〉·〈文件型〉

-R

每当发出R命令，DDT的回答或者是出错指示符“？”（文件不能被打开，或 HEX 文件校验和出错），或者为装入信息，其形式为：

NEXT PC

nnnn pppp

其中，nnnn是跟在被装入程序后面的下一个地址，pppp是假定的程序计数器（对COM文件为100H，对HEX文件从最后一个记录中提取）。

2.9 S（设置）命令

S命令允许检查内存单元，并任选地进行修改，命令形式为：

Ss

其中s是进行内存检查和修改的十六进制开始地址。DDT回答一个数字提示符，给出内存单元及当前保存在该内存单元中的数据。如果操作员打入回车，则该数据不作修改。如果打入一个字节的值，则该值被存放在提示的地址。上述两种情况下，DDT均连续提示后继地址和数据值，直至操作员打入一个句号（·）或检测到一个无效的输入值为止。

2.10 T（跟踪）命令

T命令允许从1～65535个程序步中有选择地跟踪程序的执行。命令形式为：

T

Tn

第一种情况，显示CPU状态，且执行下一个程序步。程序立即停止执行，并且显示出结束地址：

*hhhh

这里hhhh是下一个要执行指令的地址，显示地址（在D命令中使用）置为HL的值，列表地址（在L命令中使用）置为hhhh。程序结束时的CPU状态可用X命令检查。

T命令的第二种形式与第一种相似，只是在程序断点发生之前跟踪n步（n是十六进制值）。在跟踪方式下，可打入rubout字符强迫产生断点。跟踪方式在每执行一个程序步之前先显示CPU状态。显示格式与X命令中所述相同。

注意，程序跟踪在进入与CP/M接口处时不再继续，在从CP/M返回到被测试程序之后又重新恢复，继续进行。因此，CP/M访问I/O设备（如软盘驱动器）的功能，均以实时方式运行，避免定时问题。程序在跟踪方式下运行比实时方式下运行约慢500倍，因为，在执行每条用户指令之后，DDT都要获得控制。中断处理子程序也可被跟踪，但要注意，使用断点功能的那些命令（G、T和U命令）要用RST7指令来完成断点功能，这就意味着被测试程序不能使用这个中断单元。此外，跟踪方式总是在能进行中断的方式下运行被测试程序，如果在跟踪期间接收异步中断，就可能发生问题。

还要注意，在跟踪期间，操作员应使用rubout键使控制返回DDT，而不是执行RST7，以保证在中断之前完成对当前指令的跟踪。

2.11 U（不跟踪）命令

U命令与T命令相同，但中间程序步不显示。不跟踪方式允许以监控方式执行1～65535（0FFFFH）个程序步，它主要用于执行程序达到稳定态条件时，获得对它的控制。T命令的所有条件均适用于U命令。

2.12 X（检查）命令

X命令允许对被测试程序进行有选择的显示和修改，命令形式为：

X

Xr

其中，r是8080CPU寄存器中之一个：

C 进位标志（0/1）
Z 零标志（0/1）
M 负号标志（0/1）
E 偶数奇偶标志（0/1）
I 中间数字进位（0/1）
A 累加器 （0-FF）
B BC寄存器对（0-FFFF）
D DE寄存器对（0-FFFF）
H HL寄存器对（0-FFFF）

S　栈指针　(0-FFFF)

P　程序计数器 (0-FFFF)

第一种命令形式下CPU寄存器状态显示格式为：

CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd

inst

其中，f是0或1标志值，bb是字节值，dddd是对应寄存器对的双字节值。inst字段包含该CPU状态的程序计数器定址的单元反汇编产生的指令。

第二种形式允许显示和修改寄存器值。其中r是上述寄存器（C、Z、M、E、I、A、B、D、H、S或P）中的一个。在每一种情况下，标志或寄存器值首先在控制台上显示，然后DDT程序接受从控制台的输入。如果打入回车，则标志或寄存器值不修改；如果打入一个适当的数值，则修改标志或寄存器值。注意，BC、DE和HL作为寄存器对显示，因此要修改B、C或BC寄存器对时，操作员应打入整个寄存器对的值。

第三节　执行过程

DDT的程序组织允许某些非基本部分被覆盖，以便获得较大的暂存器域，用于调试大程序。DDT程序由两部分组成：DDT核心和汇编/反汇编程序模块。DDT核心装入控制台命令处理程序的位置，而汇编/反汇编程序虽然和DDT核心一起装入，但除使用汇编和反汇编时外，它是可以被覆盖的。

需特别指出，在6H单元的BDOS地址（JMP指令地址段在5H单元）由DDT进行修改作为DDT核心的基地址单元，而DDT核心又包含至BDOS的转移指令JMP。因此，用该地址字段检查内存大小的程序应把DDT核心的基地址，而不是把BDOS的基地址看作内存的逻辑结束。

汇编/反汇编程序模块直接驻存在DDT核心下面的暂驻程序区。如果在调试程序过程中使用A、L、T或X命令，则DDT程序再次修改6H单元地址段以包括这个模块，从而进一步减小了内存逻辑结束地址。如果装入程序超过汇编/反汇编程序模块的开始地址，则A和L命令丢失（使用这两个命令时回答“?”），且跟踪和显示命令（T和X）列出inst字段，以十六进制显示而不是作为译码后的指令显示。

第四节　实　　例

下面的实例介绍一个简单程序的编辑、汇编和调试。该程序读入一组数据，并决定在该组数据中的最大值。最大值从向量中取出，并在程序结束时放入“LARGE”单元中。

ED SCAN ASM

```
*I
        ORG  100H       L_L; START OF TRANSIENT AREA
        MVI  B,LEN      ; LENGTH OF VECTOR TO SCAN
        MVI  C,0        ; LARGER_RST VALUE SO FAR
```

```
LOOP_P_0_0_L LXI     H,VECT , BASE OF VECTOR
LOOP         MOV A,M     , GET VALUE
             SUB C       , LARGER VALUE IN C?
             JNC NFOUND  , JUMP IF LARGER VALUE NOT FOUND
;            NEW LARGEST VALUE,STORE IT TO C
             MOV C,A
NFOUND:      INX H       , TO NEXT ELEMENT
             DCR B       , MORE TO SCAN?
             JNC LOOP    , FOR ANOTHER
;
;            END OF SCAN,STORE C
             MOV A,C     , GET LARGEST VALUE
             STA LARGE
             JMP 0       , REBOOT
;
;            TEST DATA
VECT:        DB  2,0,4,3,5,6,1,5
LEN          EQU $-VECT  , LENGTH
LARGE:       DS  1       , LARGEST VALUE ON EXIT
             END
*B0P
             ORG 100H    , START OF TRANSIENT AREA
             MVI B,LEN   , LENGTH OF VECTOR TO SCAN
             MVI C,0     , LARGEST VALUE SO FAR
             LXI H,VECT  , BASE OF VECTOR
LOOP:        MOV A,M     , GET VALUE
             SUB C       , LARGER VALUE IN C?
             JNC NFOUND  , JUMP IF LARGE VALUE NOT FOUND
;            NEW LARGEST VALUE, STORE IT TO C
             MOV C,A
NFOUND:      INX H       , TO NEXT ELEMENT
             DRC B       , MORE TO SCAN?
             JNC LOOP    , FOR ANOTHER
;            END OF SCAN, STORE C
             MOV A,C     , GET LARGEST VALUE
             STA LARGE
             JMP 0       , REBOOT
;
;            TEST DATA
```

```
VECT :      DB  2,0,4,3,5,6,1,5
LEN         EQU $-VECT , LENGTH
LARGE :     DS  1          , LARGEST VALUE ON EXIT
            END
*E

ASM SCAN

CP/M ASSEMBLER -VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY
TYPE SCAN.PRN

0100                    ORG  100H     , START OF TRANSIENT
                                        AREA
0100  0608              MVI  B,LEN    , LENGTH OF VECTOR TO
                                        SCAN
0102  0E00              MVI  C,0      , LARGEST VALUE SO FAR
0104  211901            LXI  H,VECT   , BASE OF VECTOR
0107  7E        LOOP :  MOV  A,M      , GET VALUE
0108  91                SUB  C        , LARGE VALUE IN C?
0109  D20D01            JNC  NFOUND   , JUMP IF LARGER VALUE
                                        NOT FOUND
                ;
                , END OF SCAN, STORE C
0112  79                MOV  A,C      , GET LARGEST VALUE
0113  322101            STA  LARGE
0116  C30000            JMP  0        , REBOOT
                ;
                ,        TEST DATA
0119  0200040305 VECT :  DB   2,0,4,3,5,6,1,5
0008 =           LEN     EQU  $-VECT , LENGTH
0121             LARGE : DS   1       , LARGEST VALUE ON EXIT
0122                     END

DDT SCAN.HEX
```

```
16K DDT VER 1.0
NEXT PC
0121 0000
-X

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0000 OUT 7F
-XP

P=0000 100

-X

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100
  MVI B,08
-L10B

0100    MVI   B,08
0102    MVI   C,00
0104    LXI   H,0119
0107    MOV   A,M
0108    SUB   C
0109    JNC   010D
010C    MOV   C,A
010D    INX   H
010E    DCR   B
010F    JNC   0107
0112    MOV   A,C
-L

0113    STA   0121
0116    JMP   0000
0119    STAX  B
011A    NOP
011B    INR   B
01C     INX   B
011D    DCR   B
011E    MVI   B,01
0120    DCR   B
0121    LXI   D,2200
```

```
0124    LXI     H,0200
-A116

0116    RST     7

0117

-L113

0113    STA     0121
0116    RST     07
0117    NOP
0118    NOP
0119    STAX    B
011A    NOP
011B    INR     B
011C    INX     B

-X

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100
  MVI B,08
-T

C0Z0M0E0I0  A=00  B=0000  D=0000  H=0000  S=0100  P=0100
  MVI B,08*0102
-T

C0Z0M0E0I0  A=00  B=0800  D=0000  H=0000  S=0100  P=0102
  MVI C,00*0104
-T

C0Z0M0E0I0  A=00  B=0800  D=0000  H=0000  S=0100  P=0104
  MVI H,0119*0107

-T3

C0Z0M0E0I0  A=00  B=0800  D=0000  H=0119  S=0100  P=0107
  MOV A,M
```

```
C0Z0M0E0I0  A=02  B=0800  D=0000  H=0119  S=0100  P=0108  SUB C
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0109
  JNC 010D*010D
-D119

0119 02 00 04 03 05 06 01 ·····
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 ···"!···^.W.#..X.
0130 02 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 .'···>···
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····------
-X

C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010D  INX H
-T5
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010D  INX H
C0Z0M0E0I1  A=02  B=0800  D=0000  H=011A  S=0100  P=010E  DCR B
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=010F
  JNZ 0107
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=0107
  MOV A,M
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=0108
  SUB C*0109
-U5
C0Z1M0E1I1  A=00  B=0700  D=0000  H=011A  S=0100  P=0109
  JNC 010D*0108
-X

C0Z0M0E1I1  A=04  B=0600  D=0000  H=011B  S=0100  P=0108  SUB C
-G

*0116
-X
```

```
C0Z1M0E1I1  A=00  B=0000  D=0000  H=0121  S=0100  P=0116  RST 07
-XP

P=0116 100

-X

C0Z1M0E1I1  A=00  B=0000  D=0000  H=0121  S=0100  P=0100
  MVI B,08
-T10

C0Z1M0E1I1  A=00  B=0000  D=0000  H=0121  S=0100  P=0100
  MVI B,08
C0Z1M0E1I1  A=00  B=0800  D=0000  H=0121  S=0100  P=0102
  MVI C,00
C0Z1M0E1I1  A=00  B=0800  D=0000  H=0121  S=0100  P=0104
  LXI H,0119
C0Z1M0E1I1  A=00  B=0800  D=0000  H=0119  S=0100  P=0107
  MOV A,M
C0Z1M0E1I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0108  SUB C
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0109
  JNC 010D
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010D  INX H
C0Z0M0E0I1  A=02  B=0800  D=0000  H=011A  S=0100  P=010E  DCR B
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=010F
  JNZ 0107
C0Z0M0E0I1  A=02  B=0700  D=0000  H=011A  S=0100  P=0107
  MOV A,M
C0Z0M0E0I1  A=00  B=0700  D=0000  H=011A  S=0100  P=0108  SUB C
C0Z1M0E1I1  A=00  B=0700  D=0000  H=011A  S=0100  P=0109
  JNC 010D
C0Z1M0E1I1  A=00  B=0700  D=0000  H=011A  S=0100  P=010D  INX H
C0Z1M0E1I1  A=00  B=0700  D=0000  H=011B  S=0100  P=010E  DCR B
C0Z0M0E1I1  A=00  B=0600  D=0000  H=011B  S=0100  P=010F
  JNZ 0107
C0Z0M0E1I1  A=00  B=0600  D=0000  H=011B  S=0100  P=0107
  MOVV A,M*0108
-A109
```

```
0109 JC 10D

010C

-G0

SAVE 1 SCAN.COM

A>DDT SCAN.COM

16K DDT VER 1.0
NEXT PC
0200 0100
-L100
0100    MVI     B,08
0102    MVI     C,00
0104    LXI     H,0119
0107    MOV     A,M
0108    SUB     C
0109    JC      010D
010C    MOV     C,A
010D    INX     H
010E    DCR     B
010F    JNZ     0107
0112    MOV     A,C
-XP

P=0100

-T10

C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100
  MVI B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102
  MVI C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104
  LXI H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107
  MOV A,M
```

```
C0Z0M0E0I0  A=02  B=0800  D=0000  H=0119  S=0100  P=0108  SUB C
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0109
  JC 010D
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010C
  MOV C,A
C0Z0M0E0I1  A=02  B=0802  D=0000  H=0119  S=0100  P=010D  INX H
C0Z0M0E0I1  A=02  B=0802  D=0000  H=011A  S=0100  P=010E  DCR B
C0Z0M0E0I1  A=02  B=0702  D=0000  H=011A  S=0100  P=010F
  JNZ 010 7
C0Z0M0E0I1  A=02  B=0702  D=0000  H=011A  S=0100  P=0107
   MOV A,M
C0Z0M0E0I1  A=00  B=0702  D=0000  H=011A  S=0100  P=0108  SUB C
C1Z0M1E0I1  A=FE  B=0702  D=0000  H=011A  S=0100  P=0109
  JC 010D
C1Z0M1E0I0  A=FE  B=0702  D=0000  H=011A  S=0100  P=010D  INX H
C1Z0M1E0I0  A=FE  B=0702  D=0000  H=011B  S=0100  P=010E  DCR B
C1Z0M0E1I1  A=FE  B=0602  D=0000  H=011B  S=0100  P=010F
  JNZ 0107*0107
-X

C1Z0M0E1I1  A=FE  B=0602  D=0000  H=011B  S=0100  P=0107
  MOV A,M
-G,108

*0108
-X

C1Z0M0E1I1  A=04  B=0602  D=0000  H=011B  S=0100  P=0108  SUB C
-T

C1Z0M0E1I1  A=04  B=0602  D=0000  H=011B  S=0100  P=0108
  SUB C*0109
-T

C0Z0M0E0I1  A=02  B=0602  D=0000  H=011B  S=0100  P=0109
  JC 010D*010C
-X
C0Z0M0E0I1  A=02  B=0602  D=0000  H=011B  S=0100  P=010C
  MOV C,A
```

```
-G

*0116
-X

C0Z1M0E1I1  A=03  B=0003  D=0000  H=0121  S=0100  P=0116  RST 07
-S121

0121  03

0122  00

0123  22

0124  21

0125  00

0126  02

0127  7E

-L100

0100      MVI     B,08
0102      MVI     C,00
0104      LXI     H,0119
0107      MOV     A,M
0108      SUB     C
0109      JC      010D
010C      MOV     C,A
010D      INX     H
010E      DCR     B
010F      JNZ     0107
0112      MOV     A,C
-L

0113      STA     0121
0116      RST     07
```

```
0117      NOP
0118      NOP
0119      STAX    B
011A      NOP
011B      INR     B
011C      INX     B
011D      DCR     B
011E      MVI     B,01
0120      DCR     B
-XP

P=0116  100

-T

C0Z1M0E1I1  A=03  B=0003  D=0000  H=0121  H=0100  P=0100
  MVI B,08*0102
-T

C0Z1M0E1I1  A=03  B=0003  D=0000  H=0121  S=0100  P=0102
  MVI C,00*0104
-T

C0Z1M0E1I1  A=03  B=0800  D=0000  H=0121  S=0100  P=0102
  LXI H,0119*0107
-T

C0Z1M0E1I1  A=03  B=0800  D=0000  H=0119  S=0100  P=0107
  MOV A,M*0108
-T

C0Z1M0E1I1  A=02  B=0800  D=0000  H=0119  S=0100  P=0100
  SUB C*0109
-T

C0Z0M0E0I1  A=02  B=0000  D=0000  H=0119  S=0100  P=0109
  JC 010D*010C
-T
```

```
C0Z0M0E0I1  A=02  B=0800  D=0000  H=0119  S=0100  P=010C
  MOV C,A*010D
-T

C0Z0M0E0I1  A=02  B=0802  D=0000  H=0119  S=0100  P=010D
  INX H*010E
-T

C0Z0M0E0I1  A=02  B=0802  D=0000  H=011A  S=0100  P=010E
  DCR B*010F
-T

C0Z0M0E0I1  A=02  B=0702  D=0000  H=011A  S=0100  P=010F
  JNZ 0107*0107
-T

C0Z0M0E0I1  A=02  B=0702  D=0000  H=011A  S=0100  P=0107
  MOV A,M*0100
-T

C0Z0M0E0I1  A=00  B=0702  D=0000  H=011A  S=0100  P=0100
  SUB C*0109
-T

C1Z0M1E0I0  A=FE  B=0702  D=0000  H=011A  S=0100  P=0109
  JC 010D*010D
-T

C1Z0M1E0I0  A=FE  B=0702  D=0000  H=011A  S=0100  P=010D
  INX H*010E
-L100

0100    MVI     B,08
0102    MVI     C,00
0104    LXI     H,0119
0107    MOV     A,M
0108    SUB     C
0109    JC      010D
010C    MOV     C,A
```

```
010D    INX     H
010E    DCR     B
010F    JNZ     0107
0112    MOV     A,C
-A108

0108    CMP     C

0109

-G0

SAVE 1 SCAN.COM

A>DDT SCAN.COM

16K DDT VER 1.0
NEXT PC
0200 0100
-XP

P=0100

-L116
0116    RST     07
0117    NOP
0118    NOP
0119    STAX    B
011A    NOP
-

-G,116

*0116
-XC

C1

-X
```

```
C1Z1M0E1I1  A=06  B=0006  D=0000  H=0121  S=0100  P=0116  RST 07
-S121

0121  06

0122  00

0123  22

-G0


ED SCAN.ASM

*NSUB
*0LT
        SUB     C       ; LARGER VALUE IN C?
*SSUB ^Z CMP ^Z 0LT
        CMP     C       ; LARGER VALUE IN C?
*
        JNC     NFOUND; JUMP IF LARGER VALUE NOT FOUND
*SNC ^Z C ^Z 0LT
        JC      NFOUND; JUMP IF LARGER VALUE NOT FOUND
*E
ASM SCAN.AAZ

CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY

DDT SCAN.HEX

16K DDT VER 1.0
NEXT PC
0121  0000
-L116
```

```
0116  JMP     0000
0119  STAX    B
011A  NOP
011B  INR     B
-

-G100,116

*0116
-D121

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 ···"!···^···W·#··X·
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 ·'···)··············
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ····················

-

-G0
```

第六章　CP/M2.2接口指南

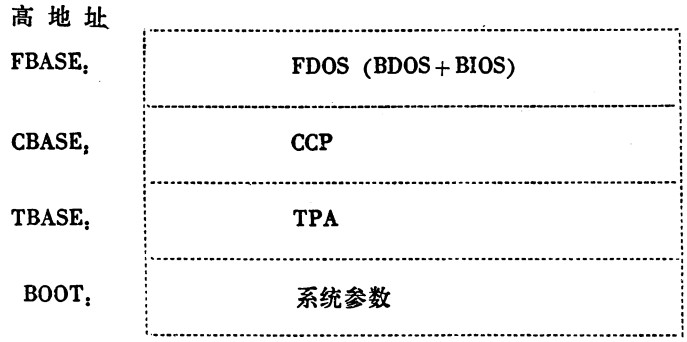
第一节　引　　言

本章描述CP/M版本2的系统组织，包括内存结构和系统入口点，可帮助用户编写在CP/M下运行、使用系统外设及磁盘输入输出功能的程序。

CP/M逻辑上分为四个部分：基本I/O系统（BIOS），基本磁盘操作系统（BDOS），控制台命令处理程序（CCP）和暂驻程序区（TPA）。BIOS是与硬件有关的模块，它定义到特定计算机系统的低级接口，这对于外设输入输出是必须的。数据研究公司提供了标准BIOS，同时为了现场重配置BIOS以适应各种硬件环境，还提供了详细说明(见“CP/M修改指南”)。

BIOS和BDOS逻辑上组合成一个模块，具有公共入口点，称为FDOS。CCP是另一个程序模块，它使用FDOS提供人与磁盘信息之间的接口。TPA是一块内存区(即不被FDOS和CCP占用的部分)，用于执行各种非常驻的操作系统命令及用户程序。内存低址部分保留用来存放系统信息（详见后面章节）。

CP/M系统内存组织如下所示：



对应于BOOT、TBASE、CBASE和FBASE的确切地址随版本的不同而异，并在“CP/M修改指南”中详细描述。但所有CP/M版本均假设BOOT = 0000H，这是随机存取存储器的基地址。在BOOT单元的机器码执行系统“热启动”，装入并预置控制返回CCP所必须的程序和变量。因此，用户程序要返回CP/M，命令控制只需转移至BOOT单元即可。并且，标准版本假设TBASE = BOOT + 0100H(通常是0100H单元)。至FDOS的主入口点是BOOT + 0005H单元（通常是0005H)。此处有一个至FBASE的转跳指令。在BOOT + 0006H（通常是0006H）地址段包含FBASE的值，假定CCP被暂驻程序复盖时该值可用于确定可用存储器的大小。

暂驻程序装入TPA并按如下方式运行：在每次提示符出现之后，操作员打入命令行与CCP进行联系，命令行取如下三种格式之一种：

{命令
命令　文件1
命令　文件1　文件2}

上述命令可为内部命令，如DIR或TYPE，也可为暂驻命令或程序的名。如果命令是CP/M内部命令，则立即执行。否则，CCP在当前选址的磁盘上查找名为“命令·COM”的文件。若找到该文件，则假设它就是能在TPA中运行的某程序的映象。该程序在TPA中执行，开始于内存的TBASE。CCP将该COM文件从磁盘装入内存，起始于TBASE并可以一直装至CBASE。

如果该命令之后跟以一个或二个文件说明，则CCP在系统参数区设置一个或两个文件控制块（FCB）。这些任选的FCB在形式上必须通过FDOS存取文件。详见下节。

暂驻程序从CCP接受控制，开始执行。可能要用到FDOS功能。暂驻程序从CCP被调用，因而在它处理完成以后将控制返回CCP，或跳转至BOOT将控制返回CP/M。在前一种情况下，暂驻程序不能使用内存中CBASE以上的区域，而在后一种情况下，内存一直至FBASE－1都是可用的。

暂驻程序可使用CP/M I/O功能与操作控制台及外设（包括磁盘子系统）通讯。I/O系统访问是通过在BOOT＋0005H单元的FDOS入口点将“功能号”和“信息地址”传递给CP/M进行的。例如，在读磁盘时，暂驻程序将对应于读磁盘的功能号和FCB地址送至CP/M FDOS。FDOS执行该操作，并返回读盘完成标记或表示读磁盘不成功的错误码。功能号和错误码在后面给出。

第二节　操作系统调用规则

本节目的是提供从用户程序直接调用操作系统功能的方法。但是，下面所列功能通过I/O宏定义库及MAC宏汇编实现则更为方便。详见《MAC宏汇编：语言手册及应用指南》。

为暂驻程序提供的CP/M功能分为两类：一般设备输入输出和磁盘文件输入输出。

一般设备操作包括：

读控制台字符
写控制台字符
读顺序带字符
写顺序带字符
写列表设备字符
取或置I/O状态
打印控制台缓冲区
读控制台缓冲区
询问控制台状态

执行磁盘输入输出的FDOS操作包括：

磁盘系统复位
驱动器选择
建立文件

打开文件
关闭文件
检索目录表
删除文件
文件改名
随机或顺序读
随机或顺序写
询问可用磁盘
询问选用磁盘
置 DMA地址
置/重置文件指示符

前面提到，对 FDOS功能的访问是通过在BOOT + 0005H单元的主入口点传递功能号及信息地址来实现的。通常，功能号放在 C寄存器中传送，信息地址放在 DE寄存器对中以双字节传送。单字节值返回 A寄存器中，双字节值返回 HL寄存器对中（当功能号超出范围时，返回 0 值）。为了兼容性，在所有情况下返回时寄存器 A = L，寄存器 B = H。注意，CP/M的寄存器传递规则与 Intel的 PL/M系统程序设计语言兼容。

CP/M功能号列出如下：

0 系统复位
1 控制台输入
2 控制台输出
3 阅读机输入
4 穿孔机输出
5 打印机输出
6 直接控制台 I/O
7 取 I/O字节
8 置 I/O字节
9 打印字符串
10 读控制台缓冲区
11 检测控制台状态
12 返回版本号
13 磁盘系统复位
14 选择磁盘
15 打开文件
16 关闭文件
17 查找第一个文件目录项
18 查找下一个文件目录项
19 删除文件
20 顺序读
21 顺序写

22 建立文件
23 文件重命名
24 返回登录向量
25 返回当前磁盘
26 置 DMA 地址
27 取地址（ALLOC）
28 磁盘写保护
29 取只读向量
30 置文件属性
31 取地址（磁盘参数）
32 置/取用户码
33 随机读
34 随机写
35 计算文件大小
36 置随机记录

在进入暂驻程序时，CCP将栈指针设置于八级栈区，并将CCP返回地址推至栈上，在发生溢出之前留下七级。虽然暂驻程序通常不使用这个堆栈（即大多数暂驻程序都通过跳转至0000H单元返回CCP），但对于建立CP/M调用，这个堆栈是足够大的，因为FDOS在系统入口转换至一局部堆栈。

例如，下面的汇编语言程序段是连续读字符，直至遇到*号时控制返回CCP（假定是标准CP/M系统BOOT = 0000H）：

```
BDOS    EQU   0005H      ; 标准的CP/M入口
CONIN   EQU   1          ; 控制台输入功能
;
        ORG   0100H      ; TPA 基地址
NEXTC:  MVI   C,CONIN    ; 读下一个字符
        CALL  BDOS       ; 返回字符在 A寄存器中
        CPI   '*'        ; 处理结束?
        JNZ   NEXTC      ; 如没结束就循环
        RET              ; 返回到CCP
        END
```

CP/M在每个磁盘上实现命名的文件结构，这种逻辑结构使任何一个文件可含有任意数目的记录，从全空到整个磁盘容量。每个驱动器逻辑上互相独立，带有磁盘目录表和文件数据区。磁盘文件名分为三部分：驱动器选择码；文件名，包含一至八个非空格字符；文件型，包含零至三个非空格字符。文件型标明文件所属种类，而文件名用来区分同一类型文件中的各个文件。

下面列出已建立的几种文件类型：

ASM 汇编程序源文件
PRN 打印机列表文件

HEX	十六进制机器码
BAS	BASIC 源文件
INT	BASIC 中间代码
COM	CCP 命令文件
PLI	PL/I 源文件
REL	可重定位模块
TEX	TEX 格式源文件
BAK	ED 源文件备份
SYM	SID 符号文件
$$$	临时文件

源文件被看成 ASCII 字符序列，每行以回车—换行结尾（0DH 后面跟以 0AH）。因此，一个128字节的 CP/M记录可包含几行源文件。一个 ASCII 文件结束以 control-Z 字符（1AH）标明，或者是实际文件结束，由 CP/M读操作返回。嵌在机器码文件（即COM文件）内的control-Z字符被忽略，用 CP/M返回的文件结束条件结束读操作。

CP/M的文件可被看成一系列128字符的记录，最多可达65536个记录，编号从0到65535，因此每个文件最大可达 8 MB。要注意，虽然这些记录逻辑上可认为是连续的，但它们在磁盘数据区物理上可以不连续。从内部看，所有文件都分为16K字节的段称为逻辑区域，因而计数器保留为 8 位值是很容易的。

从功能码15开始的文件操作中，DE 通常定址一个文件控制块 FCB。暂驻程序常常使用 CP/M 保留的缺省文件控制块域，在 BOOT + 005CH（通常是 005CH）单元作简单的文件操作。对所有文件操作，所用的文件信息基本单位是128字节。因此，CP/M 提供的对磁盘输入输出的缺省地址是 BOOT + 0080H单元（通常是0080H），这是缺省的 DMA初始地址（见功能26）。所有目录表操作都在保留区进行，因此不影响写缓冲区操作。但查找第一个和查找下一个文件目录项时例外，这时需要兼容性。

文件控制块数据区包含一系列33字节区（顺序存取）或36字节区（随机存取）。缺省的文件控制块通常分配在005CH单元，可用于随机存取文件。因为从 BOOT + 007DH 开始的三个字节可用于此目的。

FCB 格式如下所示：

dr	f1	f2	//	f8	t1	t2	t3	ex	s1	s2	rc	d0	//	dn	cr	r0	r1	r2
00	01	02	…	08	09	10	11	12	13	14	15	16	…	31	32	33	34	35

这里：

dr	驱动器码（0～16）
0	使用缺省驱动器
1	自动选择驱动器 A
2	自动选择驱动器 B
⋮	
16	自动选择驱动器 P

f1～f8　包含文件名，ASCII大写字符，高位＝0

t1、t2、t3　包含文件型，ASCII大写字符，高位＝0

t1′、t2′、t3′　标明了这些设备的bit值

t1′＝1＝＞只读文件

t2′＝1＝＞SYS文件，不能用DIR列出

ex　包含当前磁盘区域数，通常由用户设置为00，但在文件输入输出期间，取值范围为0～31

S1　保留作系统内部使用

S2　保留作系统内部使用，调用OPEN、MAKE、SEARCH时设置为0

rc　对区域ex的记录计数，取值0～128

d0…dn　由CP/M填入，保留作系统使用

cr　顺序文件操作中读写的当前记录，通常由用户设置为0

r0、r1、r2　任选随机记录数，取值范围0～65535，r0、r1组成16位值，低位字节r0，高位字节r1，溢出至r2

通过CP/M存取的每个文件都必须有一个相应的控制块，它为所有后继文件操作提供文件名和单元分配信息。当存取文件时，由程序员负责填入FCB低位16字节，并初置cr字段。通常字节1～11设置为ASCII字符值，表示文件名、文件型，所有其他字段为0。

FCB存放在磁盘的目录表区，并在文件操作进行之前进入内存（见OPEN和MAKE功能）。该存储器的FCB拷贝在文件操作时被修改，并在文件操作结束时永久性地重新记录到磁盘上（见CLOSE命令）。

CCP扫描暂驻程序名之后的命令行剩余部分（在命令格式中标明为“文件1”和“文件2”），为暂驻程序建立两个任选FCB的前16字节，未指定字段设置为ASCII空格。第一个FCB在BOOT＋005CH单元构成，可以按原样用于后继文件操作。第二个FCB占据第一个FCB的d0～dn位置。在使用之前必须移到内存的另一个域。例如，操作员打入

PROGNAME B:X.ZOT Y.ZAP

文件PROGNAME.COM被装入到TPA，缺省的FCB（在BOOT＋005CH）初置为驱动器码2，文件名“X”，文件型“ZOT”。第二个驱动器取缺省值0，放在BOOT＋006CH，文件名Y，放入BOOT＋006DH单元，文件型ZAP占8个字节，放在BOOT＋0075H单元。剩余的所有区域，直至cr均置为0。还要注意，在开始于BOOT＋005CH的文件打开之前，程序员负责将第二个文件名和文件型移到另一个区域，通常是一个单独的文件控制块。因为打开文件操作会把第二个文件名和文件型复盖掉。

如果在原命令中不含文件名，则始于BOOT＋005DH和BOOT＋006DH的字段都包含空格。在任何情况下，CCP总是将小写字母转换为大写字母，以符合CP/M文件命名规则。

为了方便，BOOT＋0080H单元的缺省缓冲区初置为操作员打入的命令行后部，跟在程序名后面。第一个位置存放字符数及字符本身。对上面给出的命令行，在开始于BOOT＋0030H单元的区域，初置值如下：

BOOT＋0080H:

＋00 ＋01 ＋02 ＋03 ＋04 ＋05 ＋06 ＋07 ＋08 ＋09 ＋10 ＋11 ＋12 ＋13 ＋14

14 “” “B” “:” “X” “.” “Z” “O” “T” “” “Y” “.” “Z” “A”“P”

其中字符转换为大写 ASCII字符，最后一个有效字符后面是未初始化存储区。此外，在文件操作执行之前，程序员负责将缓冲区信息提取出来，除非缺省的 DMA 地址已经改变。

下面详述每一个功能：

功能 0：　　系统复位

入口参数：

寄存器 C：　　00H

系统复位功能返回控制至 CP/M操作系统CCP级。CCP 重新初置磁盘子系统，选定并登录磁盘驱动器 A，该功能与跳转至 BOOT单元的作用完全相同。

功能 1：　　控制台输入

入口参数：

寄存器 C：　　01H

返回值：

寄存器 A：　　ASCII字符

控制台输入功能将下一个控制台字符读入寄存器 A，图形字符以及回车、换行、退格(ctl—H) 均返到控制台。Tab 字符 (ctl—I) 按 8 个字符位置扩展。对启动/停止滚动 (ctl—S) 和启动/停止打印机回送 (ctl—P) 进行校验。在打入一个字符以后FDOS才返回到调用程序。如果没有准备好字符输入则执行挂起。

功能 2：　　控制台输出

入口参数：

寄存器 C：　　02H

寄存器 E：　　ASCII 字符

寄存器 E的 ASCII字符被送至控制台设备，类似于功能 1 。标记按 8 个字符位置扩展。对启动/停止滚动和启动/停止打印机回送进行检查。

功能 3：　　阅读机输入

入口参数：

寄存器 C：　　03H

返回值：

寄存器 A：　　ASCII 字符

阅读机输入功能从逻辑阅读机读下一个字符至寄存器A(见“CP/M修改指南”中IOBYTE定义)，在字符读入之后控制才返回。

功能 4：　　穿孔机输出

入口参数：

寄存器 C：　　04H

寄存器 E：　　ASCII字符

穿孔机输出功能从寄存器 E送字符至逻辑穿孔机设备。

功能 5：　　打印机输出

入口参数：

寄存器 C：　　05H

寄存器 E：　　ASCII 字符

打印机输出功能将寄存器 E的 ASCII字符送至逻辑列表设备。

功能 6：　　直接控制台I/O

入口参数：

寄存器 C：　　06H

寄存器 E：　　0FFH(输入)

或char(输出)

返回值：

寄存器 A：　　char 或状态

直接控制台 I/O功能在 CP/M下支持那些需要往控制台输入输出的特殊应用。通常应避免使用这个功能，因为它不用所有的 CP/M 正常控制字符功能（如 Control-S 和 Control-P)。但是，通常 CP/M早期版本下的 BIOS执行直接 I/O的程序应改为使用 BDOS下的直接 I/O，以便能在 MP/M和 CP/M未来版本下能被充分支持。

在功能 6 的入口，寄存器 E或包含十六进制 FF，表示控制台输入请求；或包含 ASCII字

符。如果输入值是FF，则功能6返回A = 00(字符未准备)，否则A包含下一个控制台输入字符。如果E的输入值不是FF，则功能6假定E包含送往控制台的一个有效ASCII字符。

功能7：	取I/O字节
入口参数：	
寄存器C：	07H
返回值：	
寄存器A：	I/O字节值

取I/O字节功能在寄存器A中存放IOBYTE当前值（见“CP/M修改指南”关于IOBYTE定义）。

功能8：	置I/O字节
入口参数：	
寄存器C：	08H
寄存器E：	I/O字节值

置I/O字节功能将系统IOBYTE值改变为寄存器E中给出的值。

功能9：	打印字符串
入口参数：	
寄存器C：	09H
寄存器DE：	字符串地址

打印字符串功能将存放在内存单元（地址由DE给出）中的字符串送至控制台设备，直至在字符串中碰到“$”号为止。其他功能类似于功能2。

功能10：	读控制台缓冲区
入口参数：	
寄存器C：	0AH
寄存器DE：	缓冲区地址
返回值：	
缓冲区控制台字符	

读缓冲区功能将一行经编辑的控制台输入读至寄存器DE定址的缓冲区中。当输入缓冲溢出时，控制台输入结束，该缓冲区格式如下：

DE:	+0	+1	+2	+3	+4	+5	+6	+7	+8	……	+n	
	mx	nc	c1	c2	c3	c4	c5	c6	c7	……	? ?	

这里mx是缓冲区可容纳的最大字符数（1～255），“nc”是读入字符数（在返回时由FDOS设置）。c1、c2……为控制台读入的字符。如果nc＜mx，则在最后一个字符后面为未初始化位置，在上图中用“??”表示。下面是在行编辑时的一些控制字符功能：

rub/del　消去或回送最后一个字符
ctl—C　在行开始位置重新引导
ctl—E　产生物理行结束
ctl—H　退回一个字符位置
ctl—J　（换行）结束输入行
ctl—M　（返回）结束输入行
ctl—R　在新行后面重印当前行
ctl—U　在新行后移去当前行
ctl—X　退至当前行的开始位置

要注意，某些回车至最左位置的功能（如ctl—X），只是回到提示符结束列位置（在以前诸版本中，回车至最左边缘位置）。这一规则使操作员作数据输入和行修改时更易识别。

功能11：　检测控制台状态

入口参数：
　寄存器 C：　0BH
返回值：
　寄存器 A：　控制台状态

控制台状态功能检测是否从控制台打入了字符。如果字符就绪，则返回0FFH值至寄存器A。否则返回00H值。

功能12：　返回版本号

入口参数：
　寄存器 C：　0CH
返回值：
　寄存器 HL：　版本号

功能12提供与版本无关的程序设计信息。返回2字节值。H=00表示CP/M版本（H=01是MP/M），L=00表示2.0以前的各版本。CP/M2.0在寄存器L返回十六进制20，以后各版本以十六进制21、22至2F表示。用功能12，可以在写程序时提供顺序存取和随机存取功能。但在CP/M过去版本下运行时，使之不执行随机存取。

功能13：	磁盘系统复位
入口参数：	
寄存器 C：	0DH

磁盘复位功能用程序方法将文件系统复位到初始状态。所有磁盘均置成读/写状态（见功能28和29），但只选择驱动器A，并且将缺省DMA地址重新置为BOOT+0080H。例如，当应用程序需改变磁盘但不重新引导系统时可使用该功能。

功能14：	选择磁盘
入口参数：	
寄存器 C：	0EH
寄存器 E：	被选定磁盘

选择磁盘功能指明以在寄存器E中存有名字的磁盘驱动器作为后继文件操作的缺省磁盘。在16个驱动器的系统中E从0到15（E=0为驱动器A，1为驱动器B，…，15为驱动器P）。该驱动器被置于联机状态，激活它的目录表，直至下一次冷启动、热启动或磁盘系统复位为止。如果磁盘在联机时存储媒介改变，则在标准CP/M环境下，该驱动器自动进入只读状态（见功能28）。指定驱动器码为0的FCB（dr=00H）自动访问当前选定的缺省磁盘驱动器。但驱动器码数值在1和15之间时，则不管选定的缺省磁盘而直接访问驱动器A到P。

功能15：	打开文件
入口参数：	
寄存器 C：	0FH
寄存器 DE：	FCB地址
返回值：	
寄存器 A：	目录表代码

打开文件操作用于激活当前用户指定的存在于磁盘目录表中的文件。FDOS扫视被访问的磁盘目录表，查找与寄存器DE指出的FCB位置1～14相匹配的字节（s1字节自动取0值）。其中ASCII的问号（3FH）与任意位置的任一个目录表字符相匹配。通常不包含问号，且FCB的“ex”和“s2”字节为0。

如果与目录表项相符，则相应的目录表信息被拷贝到FCB的d0到dn字节。因此，可通过后继读写操作访问这些文件。注意，要对现存文件进行读写操作，必须首先打开文件。如果打开操作成功，则返回目录表代码取值0～3；如果文件找不到，则返回值为0FFH（十进制255）；如果在FCB中包含问号，则第一个相匹配的FCB被激活。注意，如果文件从首记录开始顺序存取，则当前记录（“cr”）必须被程序置0。

功能16：	关闭文件
入口参数：	
寄存器 C：	10H
寄存器 DE：	FCB地址
返回值：	
寄存器 A：	目录表代码

关闭文件功能执行与打开文件功能相反的操作。假如由寄存器DE定址的FCB已经通过打开或建立文件操作（见功能15和22）被激活，则关闭功能将新的FCB永久记录在被访问的磁盘目录表中。关闭文件操作时FCB处理过程与打开文件功能相同。返回目录表代码在关闭成功时为0、1、2或3，而当该文件名在目录表中找不到时，返回代码为0FFH（十进制255）。如果只进行读操作，则不必关闭文件，但如果进行了写操作，则必须关闭文件，以便永久记录新的目录表信息。

功能17：	查找第一个文件目录项
入口参数：	
寄存器 C：	11H
寄存器 DE：	FCB地址
返回值：	
寄存器 A：	目录表代码

该功能扫视目录表，查找与由寄存器DE定址的FCB所给定文件相符合的文件。如果文件找不到，返回值为255（十六进制FF）；如果该文件存在，返回值为0、1、2或3。当找到该文件时，在当前的DMA地址中填入包含该目录表项的记录，相应的开始位置是A*32

（即A寄存器循环左移5位或连加5次）。目录表信息可从缓冲区中提取，但通常对应用程序是不需要的。

ASCII问号（十进制63，十六进制3F）出现在“f1”至“ex”的任意位置，匹配缺省磁盘或自动选定磁盘驱动器上任意目录表项的相应字段。如果“dr”字段包含ASCII问号，则不能使用自动选定磁盘功能，这时，查找缺省磁盘。查找功能返回任意相匹配的项（属于任意用户号的已分配项或自由项）。后一种功能通常不用于应用程序，但可以灵活地扫视所有当前目录表值。如果“dr”字段不是问号，则“s2”字节自动置0值。

功能18：	查找下一个文件目录项
入口参数：	
寄存器 C：	12H
返回值：	
寄存器 A：	目录表代码

查找下一个文件目录项类似查找第一个文件目录项，但不同的是目录表扫视从最后一个相符合的登记项继续进行。其返回值与功能17相同。当没有相匹配的目录表项时，返回值为十进制255。

功能19：	删除文件
入口参数：	
寄存器 C：	13H
寄存器 DE：	FCB 地址
返回值：	
寄存器 A：	目录表代码

删除文件功能将与DE寄存器定址的FCB相符合的文件删去。该文件名和文件型可包含有歧义的引用名（即在每个位置出现问号），但是驱动器选择代码不能有歧义性。

当被访问文件找不到时，返回值为十进制255，否则返回值为0到3。

功能20：	顺序读
入口参数：	
寄存器 C：	14H
寄存器 DE：	FCB 地址

返回值：

寄存器 A：　　目录表代码

假定寄存器 DE 定址的 FCB 已通过打开或建立文件（功能15和22）操作被激活，顺序读功能将文件中下一个128字节的记录读入内存当前 DMA 地址，该记录从磁盘区的"dr"位置被读入，且"cr"字段自动增加，指向下一个记录位置。如果"cr"字段溢出，则自动打开下一个逻辑磁盘区，并将"cr"字段复位至 0，以准备下一个读操作。如果读操作成功，寄存器返回值为00H；如果下一个记录位置不存在数据（例如文件结束），则返回非 0 值。

功能21：　　顺序写

入口参数：

寄存器 C：　　15H

寄存器 DE：　　FCB 地址

返回值：

寄存器 A：　　目录表代码

假定寄存器DE定址的 FCB 已通过打开或建立文件操作被激活，顺序写功能将当前DMA地址中128字节的数据记录写至 FCB 指定的文件上，该记录放在文件的"cr"位置，且"cr"字段自动增加，指向下一个记录位置。如果 cr 字段溢出，则自动打开下一个逻辑磁盘区，并将 cr 字段复位至 0，以准备下一个写操作。写操作可对一现存文件进行，这时，新写入的记录复盖原来存在于该文件中的记录。寄存器 A 的返回值当写操作成功时为00H，当磁盘已满，写操作失败时为非 0 值。

功能22：　　建立文件

入口参数：

寄存器 C：　　16H

寄存器 DE：　　FCB 地址

返回值：

寄存器 A：　　目录表代码

建立文件操作类似于打开文件操作。不同的是在建立文件操作中，FCB 给出的文件名在当前访问的磁盘目录表中并不存在（该磁盘由非 0 的 dr 码命名，如果 dr 为 0 则为缺省磁盘）。FDOS 建立该文件，且初置目录表和主存值为一个空文件。程序员必须保证在目录表中不发生重名，如果有可能重名，必须事先进行删除操作。建立文件操作成功时，寄存器A返回值为 0 、 1 、 2 或 3 。如果目录表空间不够用，则返回值为 0 FFH（十进制 255）。建立

文件功能还有一个作用就是激活FCB，因此后继操作中不必要再进行打开文件操作。

功能23：	文件重命名
入口参数：	
寄存器 C：	17H
寄存器 DE：	FCB 地址
返回值：	
寄存器 A：	目录表代码

该功能中被改名的文件名放在由DE寄存器定址的FCB前16个字节中，新文件名放在后16个字节中，第0字节的驱动器码"dr"用于选择驱动器，而第16字节为新文件名的驱动器码，假定为0。如果重命名成功，寄存器A值置为0～3；如果在目录表查找中找不到第一个文件名，则返回值为00FH（十进制255）。

功能24：	返回登录向量
入口参数：	
寄存器 C：	18H
返回值：	
寄存器 HL：	登录向量

CP/M返回的登录向量值是在寄存器对HL中的16位值。其中L的最低位对应第一个驱动器A，H的高位对应于第16个驱动器P。某位值为0表明对应的驱动器没有联机，值为1表明对应的驱动器联机（由于选择磁盘驱动器操作，或者由于文件操作指定了非0的dr字段，选择隐含的驱动器）。注意，由于寄存A和L在返回时包含相同的值，因此保持和以前的版本兼容。

功能25：	返回当前磁盘
入口参数：	
寄存器 C：	19H
返回值：	
寄存器 A：	当前磁盘

功能25将当前选定的缺省磁盘号返回寄存器A。磁盘号0～15对应于驱动器A～P。

功能26：	置DMA地址
入口参数：	
寄存器C：	1AH
寄存器DE：	DMA地址

DMA（直接存储器地址）常与直接存取计算机内存和传送数据到磁盘子系统或从磁盘子系统传送数据至内存的磁盘控制器有关。虽然许多计算机系统使用非DMA存取（即数据通过程序I/O操作传送），但是在CP/M中使用DMA地址。在写磁盘之前和读磁盘之后，128字节的数据记录驻存在该地址中。在冷启动、热启动或磁盘复位时，DMA地址自动置为BOOT＋0080H。置DMA功能可用于改变DMA缺省值将该数据记录驻留在内存另一区域地址中。DMA地址的值由寄存器对DE指定，直至新的置DMA功能、冷启动、热启动或磁盘系统复位时才改变。

功能27：	取地址（ALLOC）
入口参数：	
寄存器C：	1BH
返回值：	
寄存器HL：	ALLOC地址

对每个联机磁盘驱动器有一个位图保存在内存中。各种系统程序使用位图提供的信息确定剩余存储器的数量（见STAT程序）。功能27将当前选定磁盘驱动器位图的基地址送回寄存器对HL中。但是如果选定磁盘为只读盘，则该盘位图无效。

功能28：	磁盘写保护
入口参数：	
寄存器C：	1CH

磁盘写保护功能使当前选定磁盘具有临时写保护。在下一次冷启动或热启动操作之前，任何试图写入该磁盘的操作均会产生下面的信息：

Bdos Err on d：R/O

功能29：	取只读向量
入口参数：	
寄存器 C：	1DH
返回值：	
寄存器 HL：	只读向量值

功能29在寄存器对HL中返回一标记位，指出临时设置只读位的磁盘驱动器。类似于功能24，最低位对应于驱动器A，最高位对应于驱动器P。只读位或者由于调用功能28，或者由于CP/M内部检测改变磁盘的自动软件功能而被设置。

功能30：	置文件属性
入口参数：	
寄存器 C：	1EH
寄存器 DE：	FCB地址
返回值：	
寄存器 A：	目录表代码

置文件属性功能允许对与文件相联的永久指示符作程序的调整。特别是只读和系统属性（t1′和t2′）可以置位或复位。DE寄存器对定址一个具有适当的属性置位或复位的单义性文件名。功能30查找目录表，修改匹配的目录表项以包含选定的指示符。指示符f1′到f4′目前还没有使用，但对应用程序可能有用，因为在文件打开和关闭操作期间不作匹配处理。指示符f5′到f8′和指示符t3′保留作将来系统扩充用。

功能31：	取地址（磁盘参数）
入口参数：	
寄存器 C：	1FH
返回值：	
寄存器 HL：	DPB地址

BIOS常驻磁盘参数块地址返回至寄存器HL，作为该功能调用的结果。磁盘参数值可以提取用于显示和空间计算。当磁盘环境改变时，暂驻程序可在必要时动态修改当前磁盘参数值。通常，应用程序不需要这个功能。

功能32：	置/取 用户码
入口参数：	
寄存器 C：	20H
寄存器 E：	0FFH（取）或 用户码(置)
返回值：	
寄存器 A：	当前码（或不取值）

通过调用功能32，应用程序可改变或询问当前活动用户号。如果寄存器 E = 0 FFH，则当前用户号的值返回至寄存器 A（取值范围为0～31）。如果寄存器 E 不是0 FFH，则将当前用户号改变为寄存器 E 的值（模32）。

功能33：	随机读
入口参数：	
寄存器 C：	21H
寄存器 DE：	FCB 地址
返回值：	
寄存器 A：	返回码

随机读功能类似于以往版本的顺序读文件操作。不同的是随机读操作是对特定的记录号进行。记录号由 FCB 后面三个字节字段构成的24位值选定（字节位置 r0在第33字节，r1 在第34字节，r2 在第35字节）。注意，24位的顺序是最低位字节 r0 在前，中间是 r1，最高位字节 r2在最后。CP/M 不引用字节 r2，在计算文件大小时除外（见功能35）。但字节 r2必须为0，因为非0 值指示在文件结束产生溢出。

因此，字节 r0、r1作为双字节或"字"值看待，包含被读的记录，其取值从 0 至 65535，可对8M 字节文件任一特定记录进行存取。要对文件作随机存取处理，必须首先打开基本磁盘区（区域 0）。虽然基本磁盘区可能包含也可能不包含任何分配的数据，但这样可以保证文件适当地记录到目录表中，并用 DIR 命令可以看到。然后选定的记录号存放到随机记录字段（r0和 r1），并调用 BDOS 读该记录。从调用返回时，寄存器 A 或者包含错误码，或者值为00，说明操作成功。后一种情况下当前 DMA 地址包含随机存取的记录。注意，和顺序读操作不同，随机读操作记录号并不增加，故后继随机读操作仍然继续读同一个记录。

对每次随机读操作，逻辑磁盘区和当前记录值是自动设置的。因此，文件可以从当前随机存取位置开始顺序读写。但要注意，在此情况下，从随机读方式转换至顺序读方式时，最后一个随机读记录将被重读；当转换为顺序写操作时，最后一个记录将被重写。当然也可在

每次随机读或写操作之后前进一个随机记录位置，以得到顺序输入输出操作的效果。

每次随机读之后返回到寄存器 A 的错误码列表如下：

01　读至未写入的数据

02　（不是以随机方式返回）

03　不能关闭当前磁盘区

04　寻找至未建立的磁盘区

05　（不是以读方式返回）

06　寻道操作超出物理磁盘结束

当随机读操作存取以前未写入的数据块或未建立的磁盘区时，发生 01 和 04 错误码。错误码03通常在适当的系统操作下是不发生的，且可以重读或重打开磁盘区 0 清除该错误码，只要该磁盘不是物理写保护。错误码 6 在当前2.0版本下，只要字节 r2 非 0 时就会发生。通常，返回码非 0 可以看成是漏掉数据，返回码为 0 指出操作完成。

功能34：　　随机写
入口参数：
寄存器 C：　22H
寄存器 DH：　FCB 地址
返回值：
寄存器 A：　返回码

随机写操作类似于随机读操作。不同的是数据从当前 DMA 地址写入磁盘。如果写操作目标磁盘区域数据块尚未分配，则在写操作继续执行之前要进行分配。如同随机读操作一样，写操作结束时随机记录号不变。逻辑磁盘区号和文件控制块当前记录位置设置为与被写入的随机记录对应。在随机写操作之后可进行顺序读、写操作。当前定址的记录作为顺序操作开始被重读或重写。

注意，在随机方式下，当读写至磁盘区最后一个记录时，不象顺序方式那样产生自动的磁盘区域转换。随机写操作返回的错误码和随机读操作相同，但增加了一个错误码05，指明由于目录表溢出不能建立新磁盘区。

功能35：　　计算文件大小
入口参数：
寄存器 C：　23H
寄存器 DE：　FCB 地址
返回值：
随机记录字段设置

当计算文件大小时，寄存器对 DE 以随机方式（存在 r0、r1、r2 字节）定址 FCB，该 FCB 包含一单义性文件名，用于目录表扫视。返回时，随机记录字节包含“虚拟”文件大小，实际上这是跟在该文件结束后面记录的地址。如果调用功能35后，高位记录字节 r2 是01，则文件包含的最大记录数为65536。否则，字节 r0和 r1组成的16位值（r0 是最低位字节）就是文件大小。

通过调用功能35，将随机记录位置设置在文件结尾，可将数据附加到现存文件后面，然后从预置记录地址开始执行一系列随机写操作。

当顺序写文件时，虚拟文件大小与物理文件大小相同。但是当文件以随机方式建立，且在文件中存在“空白区”时，则文件包含的实际记录可少于所指明的文件大小。例如，当一个8MB 的文件只有最后一个记录以随机方式写入时（即记录号为65535），则虚拟文件大小为65535个记录。但实际上只放置了一个数据块。

功能36：	置随机记录
入口参数：	
寄存器 C：	24H
寄存器 DE：	FCB 地址
返回值：	
随机记录字段设置	

设置随机记录功能使 BDOS 自动地从被顺序读、写至一特定点的文件中产生随机记录的位置。该功能可用于两个方面。首先它对于开始读和扫视一个顺序文件，提取出各种“键”字段是必要的。当碰到每一个键字段时，调用功能36，计算出该键字段对应的数据的随机记录位置。如果该数据单位大小是128字节，则计算结果产生的记录位置放入一个表中，并带有键字段供以后检索。

在扫视整个文件，建立关键字及其记录号表之后，就可以利用前面保存下来的相应的随机记录号，执行随机读，迅速传送某一个特定关键字的记录。当包含变长记录时，其模式很容易产生，因为程序只需存放缓冲器相对字节位置和关键字及记录号，就可在以后找到带有关键字数据的确切开始位置。

其次，当从顺序读写转换到随机读写时，也要使用功能36。一个文件被顺序存取至一特定点，调用功能36，设置记录号，后继的随机读写操作就可从该文件中的选定点开始继续执行。

第三节　文件至文件拷贝程序样本

下面的程序提供文件操作的简单实例，程序源文件用 CP/M ED 程序建立，为 COPY·ASM，然后用 ASM 或 MAC 汇编，产生‘HEX’文件。LOAD 程序用于产生 COPY·COM 文件，直接在 CCP 下执行。

该程序开始设置栈指针至局部域，然后从 006CH 单元缺省域将第二个名传送至 33 字节

文件控制块（称为DFCB)。该DFCB清除当前记录字段，准备文件操作。在该点，源和目的FCB准备好处理，因为在005CH单元，SFCB由CCP建立，进入COPY程序。即第一个名放入缺省FCB，有些字段置0值，包括007CH单元当前记录字段。该程序继续执行，打开源文件，删除现存目的文件，然后建立新的目的文件。如果成功，则在标号COPY处循环，直至每个记录从源文件读入，放在目的文件中。完成数据传送之后，目的文件关闭，程序返回CCP命令级（转跳至BOOT)。

文件-文件拷贝程序样本

```
                ;       sample  file-to-file copy program
                ;       at the ccp level, the command
                ;             copy a:x.y b:u.v
                ;       copies the file named x.y from drive
                ;       a to a file named u.v on drive b.
                ;
0000 =          boot    equ     0000h   ; system reboot
0005 =          bdos    equ     0005h   ; bdos entry point
005c =          fcb1    equ     005ch   ; first file name
005c =          sfcb    equ     fcb1    ; source fcb
006c =          fcb2    equ     006ch   ; second file name
0080 =          dbuff   equ     0080h   ; default buffer
0100 =          tpa     equ     0100h   ; beginning of tpa
                ;
0009 =          printf  equ     9       ; print buffer func#
000f =          openf   equ     15      ; open file func#
0010 =          closef  equ     16      ; close file func#
0013 =          deletef equ     19      ; delete file func#
0014 =          readf   equ     20      ; sequential read
0015 =          writef  equ     21      ; sequential write
0016 =          makef   equ     22      ; make file func#
                ;
0100                    org     tpa     ; beginning of tpa
0100 311b02             lxi     sp,stack ; local stack
                ;
                ;       move second file name to dfcb
0103 0e10               mvi     c,16    ; half an fcb
0105 116c00             lxi     d,fcb2  ; source of move
0108 21da01             lxi     h,dfcb  ; destination fcb
010b 1a         mfcb:   ldax    d       ; source fcb
010c 13                 inx     d       ; ready next
```

```
010d 77             mov     m,a     ; dest fcb
010e 23             inx     h       ; ready next
010f 0d             dcr     c       ; count 16···0
0110 c20b01         jnz     mfcb    ; loop 16 times
                    ;
                    ;       name has been moved,zero cr
0113 af             xra     a       ; a=00h
0114 32fa01         sta     dfcbcr  ; current rec=0
                    ;
                    ;       source and destination fcb's ready
                    ;
0117 115c00         lxi     d,sfcb  ; source file
011a cd6901         call    open    ; error if 255
011d 118701         lxi     d,nofile ; ready message
0120 3c             inr     a       ; 255 becomes 0
0121 cc6101         cz      finis   ; done if no file
                    ;
                    ;       ource file open,prep destination
0124 11da01         lxi     d,dfcb  ; destination
0127 cd7301         call    delete  ; remove if present
                    ;
012a 11da01         lxi     d,dfcb  ; destination
012b cd8201         call    make    ; create the file
0130 119601         lxi     d,nodir ; ready message
0133 3c             inr     a       ; 255 becomes 0
0134 cc6101         cz      finis   ; done if no dir space
                    ;
                    ;       source file open,dest file open
                    ;       copy until end of file on source
                    ;
0137 115c00 copy:   lxi     d,sfcb  ; source
013a cd7801         call    read    ; read next record
013d b7             ora     a       ; end of file?
013e c25101         jnz     eofile  ; skip write if so
                    ;
                    ;       not end of file,write the record
0141 11da01         lxi     d,dfcb  ; destination
0144 cd7d01         call    write   ; write record
0147 11a901         lxi     d,space ; ready message
```

```
014a b7             ora     a       ; 00 if write ok
014b c46101         cnz     finis   ; end if so
014e c33701         jmp     copy    ; loop until eof
                    ;
                    eofile:; end of file, close destination
0151 11da01         lxi     d,dfcb  ; destination
0154 cd6e01         call    close   ; 255 if error
0157 21bb01         lxi     h,wrprot ; ready message
015a 3c             inr     a       ; 255 becomes 00
015b cc6101         cz      finis   ; shouldn't happen
                    ;
                    ;       copy operation complete, end
015e 11cc01         lxi     d,normal ; ready message
                    ;
                    finis:; write message given by de, reboot
0161 0e09           mvi     c,printf
0163 cd0500         call    bdos    ; write message
0166 c30000         jmp     boot    ; reboot system
                    ;
                    ;       system interface subroutines
                    ;       (all return directly from bdos)
                    ;
0169 0e0f           open:   mvi     c,openf
016b c30500         jmp     bdos
                    ;
016e 0e10           close:  mvi     c,closef
0170 c30500         jmp     bdos
                    ;
0173 0e13           delete: mvi     c,deletef
0175 c30500         jmp     bdos
                    ;
0178 0e14           read:   mvi     c,readf
017a c30500         jmp     bdos
                    ;
017d 0e15           write:  mvi     c,writef
017f c30500         jmp     bdos
                    ;
0182 0e16           make:   mvi     c,makef
0184 c30500         jmp     bdos
```

```
                ;
                ;       console messages
0187 6e6f20fnofile:     db      "no source file$"
0196 6e6f209nodir:      db      "no directory space$"
01a9 6f7574fspace:      db      "out of data space$"
01bb 7772695wrprot:     db      "write protected?$"
01cc 636f700normal:     db      "copy complete$"
                ;
                ;       data areas
01da        dfcb:       ds      33          ; destination fcb
01fa =      dfcbcr      equ     dfcb+32     ; current record
                ;
01fb                    ds      32          ; 16 level stack
            stack:
021b                    end
```

注意，在这个程序中有几处作了简化。首先，对于包含有歧义性引用的无效文件名应进行校验。这种情况可通过扫视32字节缺省域（始于005CH 单元）ASCII 问号来检查。还应该校验005DH 和006DH 单元非空格 ASCII 字符，以确保该文件名实际上存在。最后，应该校验源文件名和目的文件名，以保证其不同名。在每次读操作中应将更多数据存入缓冲区以改善速度。例如，应从0006H 单元取 FBASE 确定内存大小，并将剩余内存全部用作数据缓冲区。在这种情况下，程序员在每次读之前重置 DMA 地址于下一个 128 字节区。在写至目的文件时，DMA 地址复位至缓冲区始点，并在每个记录传送至目的文件时，按 128 字节的增量放置，直至结束。

第四节　文件转储程序样本

文件转储程序较之拷贝程序要复杂些。转储程序读输入文件（在 CCP 命令行指定）并以十六进制格式在控制台显示每个记录的内容。注意，转储程序在入口时保存 CCP 栈，重置堆栈至局部域，并在直接返回至 CCP 之前恢复 CCP 栈。因此，在处理结束时，转储程序不执行，系统热启动。

```
                ; DUMP program reads input file and displays hex data
                ;
0100                    org     100h
0005 =          bdos    equ     0005h   ; dos entry point
0001 =          cons    equ     1       ; read console
0002 =          typef   equ     2       ; type function
0009 =          printf  equ     9       ; buffer print entry
000b =          brkf    equ     11      ; break key function (true if char
```

```
000f =          openf   equ     15          ; file open
0014 =          readf   equ     20          ; read function
                ;
005c =          fcb     equ     5ch         ; file control block address
0080 =          buff    equ     80h         ; input disk buffer address
                ;
                ;       non graphic characters
000d =          cr      equ     0dh         ; carriage return
000a =          lf      equ     0ah         ; line feed
                ;
                ;       file control block definitions
005c =          fcbdn   equ     fcb+0       ; disk name
005d =          fcbfn   equ     fcb+1       ; file name
0065 =          fcbft   equ     fcb+9       ; disk file type(3 characters)
0068 =          fcbrl   equ     fcb+12      ; file's current reel number
006b =          fcbrc   equ     fcb+15      ; file's record count(0 to 128)
007c =          fcbcr   equ     fcb+32      ; current(next)record number(0)
007d =          fcbln   equ     fcb+33      ; fcb length
                ;
                ;       set up stack
0100 210000             lxi     h,0
0103 39                 dad     sp
                ;       entry stack pointer in hl from the ccp
0104 221502             shld    oldsp
                ;       set sp to local stack area(restored at finis)
0107 315702             lxi     sp,stktop
                ;       read and print successive buffers
010a cdc101             call    setup       ; set up input file
010d feff               cpi     255         ; 255 if file not present
010f c21b01             jnz     openok      ; skip if open is ok
                ;
                ;       file not there, give error message and return
0112 11f301             lxi     d,opnmsg
0115 cd9c01             call    err
0118 c35101             jmp     finis       ; to return
                openok: ; open operation ok, set buffer index to end
011b 3e80               mvi     a,80h
011d 321302             sta     ibp         ; set buffer pointer to 80h
```

```
                ;       hl contains next address to print
0120 210000             lxi     h,0         ; start with 0000
                ;
                gloop:
0123 e5                 push    h           ; save line position
0124 cda201             call    gnb
0127 e1                 pop     h           ; recall line position
0128 da5101             jc      finis       ; carry set by gnd if end file
012b 47                 mov     b,a
                ;       print hex values
                ;       check for line fold
012c 7d                 mov     a,l
012d e60f               ani     0fh         ; check low 4 bits
012f c24401             jnz     nonum
                ;       print line number
0132 cd7201             call    crlf
                ;
                ;       check for break key
0135 cd5901             call    break
                ;       accum lsb=1 if character ready
0138 0f                 rrc                 ; into carry
0139 da5101             jc      finis       ; don't print any more
                ;
013c 7c                 mov     a,h
013d cd8f01             call    phex
0140 7d                 mov     a,l
0141 cd8f01             call    phex
                nonum:
0144 23                 inx     h           ; to next line number
0145 3c20               mvi     a,
0147 cd6501             call    pchar
014a 78                 mov     a,b
014b cd8f01             call    phex
014e c32301             jmp     gloop
                ;
                finis:
                ;       end of dump, return to ccp
                ;       (note that a jmp to 0000h reboots)
0151 cd7201             call    crlf
```

```
0154 2a1502              lhld    oldsp
0157 f9                  sphl
                    ;    stack pointer contains ccp's stack location
0158 c9                  ret                     ; to the ccp
                    ;
                    ;
                    ;    subroutines
                    ;
                    break:  ; check break key (actually any key will do)
0159 e5d5c5              push h! push d! push b; environment saved
015c 0e0b                mvi     c,brkf
015e cd0500              call    bdos
0161 c1d1e1              pop b! pop d! pop h; environment restored
0164 c9                  ret
                    ;
                    pchar:  ; print a character
0165 e5d5c5              push h! push d! push d; saved
0168 0e02                mvi     c,typef
016a 5f                  mov     e,a
016b cd0500              call    bdos
016e c1d1e1              pop b! pop d! pop h; restored
0171 c9                  ret
                    ;
                    crlf:
0172 3e0d                mvi     a,cr
0174 cd6501              call    pchar
0177 3e0a                mvi     a,lf
0179 cd6501              call    pchar
017c c9                  ret
                    ;
                    ;
                    pnib:   ; print nibble in reg a
017d e60f                ani     0fh             ; low 4 bits
017f fe0a                cpi     10
0181 d28901              jnc     p10
                    ;    less than or equal to 9
0184 c630                adi     '0'
0186 c38b01              jmp     prn
                    ;
```

```
                        ;       greater or equal to 10
0189 c637       p10:    adi     'a'-10
018b cd6501     prn:    call    pchar
018e c9                 ret
                ;
                phex:   ; print hex char in reg a
0181 f5                 push    psw
0190 0f                 rrc
0191 0f                 rrc
0192 0f                 rrc
0193 0f                 rrc
0194 cd7d01             call    pnib    ; print nibble
0197 f1                 pop     psw
0198 cd7d01             call    pnib
019b c9                 ret
                ;
                err:    ; print error message
                ;       d,e addresses message ending with"$"
019c 0e09               mvi     c,printf ; print buffer function
019e cd0500             call    bdos
01a1 c9                 ret
                ;
                ;
                gnb:    ; get next byte
01a2 3a1302             lda     ibp
01a5 fe80               cpi     80h
01a7 c2b301             jnz     g0
                ;       read another buffer
                ;
                ;
01aa cdce01             call    diskr
01ad b7                 ora     a       ; zero value if read ok
01ae cab301             jz      g0      ; for another byte
                ;       end of data, return with carry set for eof
01b1 37                 stc
01b2 c9                 ret
                ;
                g0:     ;read the byte at buff+reg a
01b3 5f                 mov     e,a     ; ls byte of buffer index
```

```
01b4 1600                   mvi     d,0         ; double precision index to de
01b6 3c                     inr     a           ; index = index + 1
01b7 321302                 sta     ibp         ; back to memory
                    ;       pointer is incremented
                    ;       save the current file address
01ba 218000                 lxi     h,buff
01bd 19                     dad     d
                    ;       absolute character address is in hl
01be 7e                     mov     a,m
                    ;       byte is in the accumulator
01bf b7                     ora     a           ; reset carry bit
01c0 c9                     ret
                    ;
                    setup:  ; set up file
                    ;       open the file for input
01c1 af                     xra     a           ; zero to accum
01c2 327c00                 sta     fcbcr       ; clear current record
                    ;
01c5 115c00                 lxi     d,fcb
01c8 0e0f                   mvi     c,openf
01ca cd0500                 call    bdos
                    ;       255 in accum if open error
01cd c9                     ret
                    ;
                    diskr:  ; read disk file record
01ce e5d5c5                 push h! push d! push b
01d1 115c00                 lxi     d,fcb
01d4 0e14                   mvi     c,readf
01d6 cd0500                 call    bdos
01d9 c1d1e1                 pop b! pop d! pop h
01dc c9                     ret
                    ;
                    ;       fixed message area
01dd 46494c0signon:         db      "file dump version 2.0$"
01f3 0d0a4e0opnmsg:         db      cr,lf,"no input file present on disk$"

                    ;       variable area
0213                ibp:    ds      2           ; input buffer pointer
0215                oldsp:  ds      2           ; entry sp value from ccp
```

```
                ;
                ;       stack area
0217                    ds      64              ; reserve 32 level stack
                stktop:
                ;
0257                    end
```

第五节　随机存取程序样本

下面是随机存取程序实例，该程序根据从终端进入的命令执行简单读写随机记录的功能。假设该程序已建立，经过汇编并放入一个标号为RANDOM.COM的文件中，由CCP级命令：

RANDOM　X·DAT

启动该试验程序，该程序按名X·DAT查找文件。如果找到，控制台出现提示符，请求输入。如果未找到，则在给出提示符之前建立该文件。提示符格式是：

next　command?

接着是操作员输入命令，打回车键。

输入命令格式为

nW　nR　Q

这里，n是从0到65535的整数，W、R和Q是对应于随机写、随机读和退出操作的简化命令字符。如果发W命令，则RANDOM程序发提示符：

type　data：

操作员打入数据（最多为127字符），然后打回车。RANDOM将字符串写入X·DAT文件记录n。如果发R命令，则RANDOM读记录号n并将该字符串值显示在控制台上。如果发Q命令，则X·DAT文件关闭，程序返回控制台命令处理程序。为简短起见，只有一个错误信息，就是

error　try　again

程序从初始化段开始，打开或建立输入文件，跟着在标号“ready”处循环，翻译每一条命令。005CH单元缺省文件控制块和0080H单元缺省缓冲区用于所有磁盘文件操作，后面是实用程序，包括主要输入行处理程序（称为“readc”），这个程序示出随机存取处理的程序结构，可作为进一步程序研制的基础。

```
                ;  ***************************************************
                ;  *                                                 *
                ;  *  sample random access program for cp/m 2.0      *
                ;  *                                                 *
                ;  ***************************************************
                ;
0100                    org     100h            ; base of tpa
                ;
0000 =          reboot  equ     0000h           ; system reboot
```

```
0005 =      bdos    equ     0005h   ; bdos entry point
            ;
0001 =      coninp  equ     1       ; console input function
0002 =      conout  equ     2       ; console output function
0009 =      pstring equ     9       ; print string until "$"
000a =      rstring equ     10      ; read console buffer
000c =      version equ     12      ; return version number
000f =      openf   equ     15      ; file open function
0010 =      closef  equ     16      ; close function
0016 =      makef   equ     22      ; make file function
0021 =      readr   equ     33      ; read random
0022 =      writer  equ     34      ; write random
            ;
005c =      fcb     equ     005ch   ; default file control block
007d =      ranrec  equ     fcb+33  ; random record position
007f =      ranovf  epu     fcb+35  ; high order (overflow) byte
0080 =      buff    equ     0080h   ; buffer address
            ;
000d =      cr      equ     0dh     ; carriage return
000a =      lf      equ     0ah     ; line feed
            ;
            ;  ***************************************************
            ;  *                                                 *
            ;  *  load SP, set-up file for random access         *
            ;  *                                                 *
            ;  ***************************************************
            ;
0100 31bc0          lxi     sp,stack
            ;
            ;       version 2.0?
0103 0e0c           mvi     c,version
0105 cd050          call    bdos
0108 fe20           cpi     20h     ; version 2.0 or better?
010a d2160          jnc     versok
            ;       bad version, message and go back
010d 111b0          lxi     d,badver
0110 cdda0          call    print
0113 c3000          jmp     reboot
            ;
            versok:
```

```
                ;       correct version for random access
0116 0e0f               mvi     c,openf         ; open default fcb
0118 115c0              lxi     d,fcb
011b cd050              call    bdos
011e 3c                 inr     a               ; err 255 becomes zero
011f c2370              jnz     ready
                ;
                ;       cannot open file, so create it
0122 0e16               mvi     c,makef
0124 115c0              lxi     d,fcb
0127 cd050              call    bdos
012a 3c                 inr     a               ; err 255 becomes zero
012b c2370              jnz     ready
                ;
                ;       cannot create file, directory full
012e 113a0              lxi     d,nospace
0131 cdda0              call    print
0134 c3000              jmp     reboot          ; back to ccp
                ;
                ;  ***************************************************
                ;  *                                                 *
                ;  *  loop back to "ready" after each command        *
                ;  *                                                 *
                ;  ***************************************************
                ;
                ready:
                ;       file is ready for processing
                ;
0137 cde50              call    readcom         ; read next command
013a 227d0              shld    ranrec          ; store input record#
013d 217f0              lxi     h,ranovf
0140 3600               mvi     m,0             ; clear high byte if set
0142 fe51               cpi     'Q'             ; quit?
0144 c2560              jnz     notq
                ;
                ;       quit processing, close file
0147 0e10               mvi     c,closef
0149 115c0              lxi     d,fcb
014c cd050              call    bdos
```

```
014f 3c             inr     a           ; err 255 becomes 0
0150 cab90          jz      error       ; error message,retry
0153 c3000          jmp     reboot      ; back to ccp
                    ;
                    ;*************************************************
                    ;*                                               *
                    ;*  end of quit command, process write           *
                    ;*                                               *
                    ;*************************************************
                    ;
                    notg:
                            not the quit command, random write?
0156 fe57           cpi     'w'
0158 c2890          jnz     notw
                    ;
                    ;       this is a random write, fill buffer until cr
015b 114d0          lxi     d,datmsg
015e cdda0          call    print       ; data prompt
0161 0e7f           mvi     c,127       ; up to 127 characters
0163 21800          lxi     h,buff      ; destination
                    rloop:  ; read next character to buff
0166 c5             push    b           ; save counter
0167 e5             push    h           ; next destination
0168 cdc20          call    getchr      ; character to a
016b e1             pop     h           ; restore counter
016c c1             pop     b           ; restore next to fill
016d fe0d           cpi     cr          ; end of line?
016f ca780          jz      erloop
                    ;       not end, store character
0172 77             mov     m,a
0173 23             inx     h           ; next to fill
0174 0d             dcr     c           ; counter goes down
0175 c2660          jnz     rloop       ; end of buffer?
                    erloop:
                    ;       end of read loop, store 00
0178 3600           mvi     m,0
                    ;
                    ;       write the record to selected record number
017a 0e22           mvi     c,writer
017c 115c0          lxi     d,fcb
```

```
017f cd050          call    bdos
0182 b7             ora     a           ; error code zero?
0183 c2b90          jnz     error       ; message if not
0186 c3370          jmp     ready       ; for another record
                    ;
                    ;*************************************************************
                    ;*                                                           *
                    ;*  end of write command, process read                       *
                    ;*                                                           *
                    ;*************************************************************
                    ;
                    notw:
                    ;       not a write command, read record?
0189 fe52           cpi     "R"
018b c2b90          jnz     error       ; skip if not
                    ;
                    ;       read random record
018e 0e21           mvi     c,readr
0190 115c0          lxi     d,fcb
0193 cd050          call    bdos
0196 b7             ora     a           ; return code 00?
0197 c2b90          jnz     error
                    ;
                    ;       read was successful, write to console
019a cdcf0          call    crlf        ; new line
019d 0e80           mvi     c,128       ; max 128 characte
019f 21800          lxi     h,buff      ; next to get
                    wloop:
01a2 7e             mov     a,m         ; next character
01a3 23             inx     h           ; next to get
01a4 e67f           ani     7fh         ; mask parity
01a6 ca370          jz      ready       ; for another command if 00
01a9 c5             push    b           ; save counter
01aa e5             push    h           ; save next to get
01ab fe20           cpi     ' '         ; graphic?
01ad d4c80          cnc     putchr      ; skip output if not
01b0 e1             pop     h
01b1 c1             pop     b
01b2 0d             dcr     c           ; count = count - 1
01b3 c2a20          jnz     wloop
```

```
01b6 c3370          jmp     ready
                ;
                ;  ***********************************************************
                ;  *                                                         *
                ;  *  end of read command, all errors end-up here            *
                ;  *                                                         *
                ;  ***********************************************************
                ;
                error:
01b9 11590          lxi     d,errmsg
01bc cdda0          call    print
01bf c3370          jmp     ready
                ;
                ;  ***********************************************************
                ;  *                                                         *
                ;  *  utility subroutines for console i/o                    *
                ;  *                                                         *
                ;  ***********************************************************
                ;
                getchr:
                ; read next console character to a
01c2 0e01           mvi     c,coninp
01c4 cd050          call    bdos
01c7 c9             ret
                ;
                putchr:
                ; write character from a to console
01c8 0e02           mvi     c,conout
01ca 5f             mov     e,a         ; character to send
01cb cd050          call    bdos        ; send character
01ce c9             ret
                ;
                crlf:
                ;       send carriage return line feed
01df 3e0d           mvi     a,cr        ; carriage return
01d1 cdc80          call    putchr
01d4 3e0a           mvi     a,lf        ; line feed
01d6 cdc80          call    putchr
01d9 c9             ret
```

```
                ;
                print:
                ;       print the buffer addressed by de until $
01da d5                 push    d
01db cdcf0              call    crlf
01de d1                 pop     d       ; new line
01df 0e09               mvi     c,pstring
01e1 cd050              call    bdos    ; print the string
01e4 c9                 ret
                ;
                readcom:
                        ; read the next command line to the conbuf
01e5 116b0              lxi     d,prompt
01e8 cdda0              call    print   ; command?
01eb 0e0a               mvi     c,rstring
01ed 117a0              lxi     d,conbuf
01f0 cd050              call    bdos    ; read command line
                ;       command line is present, scan it
01f3 21000              lxi     h,0     ; start with 0000
01f6 117c0              lxi     d,conlin ; command line
01f9 1a         readc:  ldax    d       ; next command character
01fa 13                 inx     d       ; to next command position
01fb b7                 ora     a       ; cannot be end of command
01fc c8                 rz
                ;       not zero, numeric?
01fd d630               sui     '0'
01ff fe0a               cpi     10      ; carry if numeric
0201 d2130              jnc     endrd
                ;       add-in next digit
0204 29                 dad     h       ; *2
0205 4d                 mov     c,l
0206 44                 mov     b,h     ; bc=value*2
0207 29                 dad     h       ; *4
0208 29                 dad     h       ; *8
0209 09                 dad     b       ; *2+*8=*10
020a 85                 add     l       ; +digit
020b 6f                 mov     l,a
020c d2f90              jnc     readc   ; for another char
020f 24                 inr     h       ; overflow
```

```
0210 c3f90              jmp     readc       ; for another char
                endrd:
                ;       end of read, restore value in a
0213 c630               adi     '0'         ; command
0215 fe61               cpi     'a'         ; translate case?
0217 d8                 rc
                ;       lower case, mask lower case bits
0218 e65f               ani     101$1111b
021a c9                 ret
                ;
                ;*****************************************************
                ;*                                                   *
                ;*  string data area for console messages            *
                ;*                                                   *
                ;*****************************************************
                badver:
021b 536f79             db      "sorry, you need cp/m version 2$"
                nospace:
023a 4e6f29             db      "no directory space$"
                datmsg:
024d 547970             db      "type data: $"
                errmsg:
0259 457272             db      "error, try again. $"
                prompt:
026b 4e6570             db      "next command? $"
                ;
                ;*****************************************************
                ;*                                                   *
                ;*  fixed and variable data area                     *
                ;*                                                   *
                ;*****************************************************
027a 21         conbuf: db      conlen      ; length of console buffer
027b            consiz: ds      1           ; resulting size after read
027c            conlin: ds      32          ; length 32 buffer
0021 =          conlen  equ     $-consiz
                ;
029c                    ds      32          ; 16 level stack
                stack:
02bc                    end
```

可以对上述程序进行修改以增强其功能。事实上，该程序经过一番改进以后可演变为简单的数据库管理系统。例如，我们可以假定标准记录长度为128字节，记录中包含任意字段。可以编写一个程序（称为GETKEY），首先读一个顺序文件，提取出由操作员定义的特定字段。例如，命令

GETKEY NAMES·DAT LASTNAME 10 20

使GETKEY读数据库文件NAMES·DAT，从每个记录中提取出“LASTNAME”字段(开始位置10，结束字符20)，GETKEY在内存中建表，包含每个LASTNAME字段以及该文件内16位记录号单元。然后，GETKEY程序对该表进行排序，写新文件，称为LASTNAME·KEY，该文件是LASTNAME字段的字母表及相应的记录号（该表称为转换索引）。

将上面所示的程序改名为QUERY，稍加变换，使之将经过排序的关键字文件读入内存，命令如下：

QUERY NAME·DAT LASTNAME·KEY

QUERY程序不是读记录号，而是读字母数字字符串，它是NAMES·DAT数据库中特殊关键字。由于LASTNAME·KEY表是排序的，因此，可通过“对半检索”迅速找出特殊入口，类似于查找电话簿中的人名。亦即从表的两端开始，检查中间一半处的登记项，如不符合，对半检查上半部或下半部，作下一次检索，这样可很快找到所要找的项（用Log2（n）步），找到相应的记录号，取出该记录，并在控制台上显示。

在这个基础上再作一些改动，就可允许处理不同于上面所说的128字节记录的固定分组记录。方法是记下记录号以及在该记录内的字节位移。知道分组大小，可随机存取包含该组的记录，然后位移至该记录内该组的开始处，顺序读，直至读完一个组长度。

最后，还可以改进QUERY，允许用布尔表达式计算能满足几种关系的一组记录，并将这些记录显示出来。如果表格太大，不能放入内存，可从磁盘存取关键字文件。

第六节　系统功能综述

功能	功能名	输入参数	输出结果
0	系统复位	无	无
1	控制台输入	无	A = ASCII 字符
2	控制台输出	E = ASCII 字符	无
3	阅读机输入	无	A = ASCII 字符
4	穿孔机输出	E = ASCII 字符	无
5	打印机输出	E = ASCII 字符	无
6	直接控制台 I/O	见手册	见手册
7	取I/O字节	无	A = IOBYTE
8	置I/O字节	E = IOBYTE	无
9	打印字符串	DE = 缓冲区地址	无
10	读控制台缓冲区	DE = 缓冲区地址	见手册
11	检测控制台状态	无	A = 00/FF

12	返回版本号	无	HL = 版本号 *
13	磁盘系统复位	无	见手册
14	选择磁盘	E = 磁盘号	见手册
15	打开文件	DE = FCB 地址	A = 目录表码
16	关闭文件	DE = FCB 地址	A = 目录表码
17	查找第一个文件目录项	DE = FCB 地址	A = 目录表码
18	查找下一个文件目录项	无	A = 目录表码
19	删除文件	DE = FCB 地址	A = 目录表码
20	顺序读	DE = FCB 地址	A = 错误码
21	顺序写	DE = FCB 地址	A = 错误码
22	建立文件	DE = FCB 地址	A = 目录表码
23	文件重命名	DE = FCB 地址	A = 目录表码
24	返回登录向量	无	HL = 登录向量 *
25	返回当前磁盘	无	A = 当前磁盘号
26	置 DMA 地址	DE = DMA 地址	无
27	取地址(ALLOC)	无	HL = 位图地址
28	磁盘写保护	无	见手册
29	取只读向量	无	HL = 只读向量 *
30	取文件属性	DE = FCB 地址	见手册
31	取地址(磁盘参数)	无	HL = DPB 地址
32	置/取用户码	见手册	见手册
33	随机读	DE = FCB 地址	A = 错误码
34	随机写	DE = FCB 地址	A = 错误码
35	计算文件大小	DE = FCB 地址	r0, r1, r2
36	置随机记录	DE = FCB 地址	r0, r1, r2

第七节　新 CP/M2.2功能

下面是新 CP/M2.2　BDOS 功能：

功能37：　　驱动器复位

入口参数：

寄存器 C：　　25H

寄存器 DE：　　驱动器向量

返回值：

寄存器 A：　　00H

* 注：返回时　A = L, B = H

该功能允许将特定磁盘驱动器复位，传递参数是被复位驱动器的16位向量，最低位是驱动器A。

为保持与MP/M兼容，CP/M返回值为零。

功能40：	零填充的随机写
入口参数：	
寄存器 C：	28H
寄存器 DE：	FCB 地址
返回值：	
寄存器 A：	返回码

零填充的随机写操作类似于功能34，不同的是，原来未分配的块在数据写入之前用零填充。

第七章　CP/M2.2修改指南

第一节　引　　言

标准CP/M系统用在Intel MDS-800微型机开发系统上，但用户可以修改一组专用子程序以定义特定硬件操作环境。这样就可以产生一能在任何与IBM-3741格式兼容的驱动控制器及其他外围设备下运行的磁盘。

虽然标准的CP/M 2.0是为单密度软磁盘配置的，但现场修改特性使之可适用于各种磁盘子系统，从单驱动小磁盘到大容量硬盘系统。

为了实现设备独立性，CP/M分成三个不同的模块：

BIOS——基本输入输出系统，依赖于硬件环境

BDOS——基本磁盘操作系统，不依赖于硬件配置

CCP——控制台命令处理程序，它需要使用BDOS

在这三个模块中，只有BIOS依赖于特定的硬件。这就是说，用户可以补订CP/M的原版本，以产生一个能提供CP/M其他模块与用户本身硬件系统之间特定接口的新BIOS。本文旨在提供将新BIOS插入CP/M的详细步骤。

新BIOS需要一些软件开发与调试工作。除了BIOS，用户还要写一个简单的内存装入程序，称为GETSYS，将操作系统装入内存。为了把新BIOS插入CP/M，用户还要写一个与GETSYS作用相反的程序，称为PUTSYS，将修改后的CP/M版本写回磁盘。一般把GETSYS中的读磁盘命令改为写磁盘命令就能得到PUTSYS。为使系统能自动投入运行，用户还要提供一个与CP/M所提供的相似的冷启动装入程序。

第二节　一级系统重新生成

下面将分几步给出修改CP/M的过程。每步中用到的地址都以“H”作后缀，表示16进制基数，这些地址是按20K CP/M系统给出的。对于更大的CP/M系统，在每个地址上加一偏差，用后缀“+b”表示，其中b等于实际内存容量减去20K。各种标准内存标量对应的b值为：

24K：　b = 24K − 20K = 4K = 1000H

32K：　b = 32K − 20K = 12K = 3000H

40K：　b = 40K − 20K = 20K = 5000H

48K：　b = 48K − 20K = 28K = 7000H

56K：　b = 56K − 20K = 36K = 9000H

62K：　b = 62K − 20K = 42K = A800H

64K：　b = 64K − 20K = 44K = B000H

注：CP/M的标准版本按20K内存提供。所以，必须先使用20K CP/M系统，然后再按用

户实际内存容量进行配置（见二级系统生成）。

（1）参看第四节，写一GETSYS程序。该程序功能是将磁盘上的前两个磁道读入内存。从磁盘读入的数据必须从3380H单元开始存放。编写GETSYS程序时，要象附录D中说明的那样，开始于100H单元（TPA基地址）。

（2）调试GETSYS程序。从一空盘上将数据读入内存，检查读出的数据是否正确，并检查磁盘上的数据是否被GETSYS程序改变。

（3）运行GETSYS程序，用已初始置上CP/M的磁盘，检查GETSYS是否将CP/M装载至3380H（操作系统实际起始于128个字节以后的3400H）。

（4）参看第四节，写一PUTSYS程序。该程序功能是将始于3380H的内存区写回磁盘的前两磁道。PUTSYS程序应按附录D第二部分中说明的那样放在200H单元。

（5）用一未初始化的空盘调试PUTSYS程序。将内存区写入前两磁道，清除内存区，再用GETSYS读回。PUTSYS的调试必须做完，因为要用它来修改盘上的CP/M。

（6）仔细阅读第五、六、七节以及附录B中的BIOS原版本，写一个能执行与修改后环境下相类似功能的简单文本。可以附录C中给出的程序为模型。把新的BIOS取名为CBIOS（用户化的BIOS）。实现一些单驱动器基本磁盘操作以及一些简单控制台输入输出功能。

（7）完整地调试CBIOS，保证它能正确执行控制台字符输入输出以及磁盘读写，特别要保证在读磁盘操作时不会偶然发生写磁盘操作，并检查所有读写过程中的磁道、扇区寻址是否都正确。如果不作这些检查，修改以后会把初始预置的CP/M系统破坏掉。

（8）参照第五节中的图1，注意BIOS放在内存4A00H与4FFFH之间。用GETSYS读入CP/M系统，用第（6）步开发第（7）步调试出来的新CBIOS替换原BIOS段。这项替换工作是在计算机的内存中完成的，下一步才将它放回磁盘。

（9）用PUTSYS将CP/M修改过的内存映像写到一空磁盘的前两条磁道上去，以供测试。

(10) 用GETSYS再将复制的内存映像从测试盘上读回内存3380H，检查它是否正确装回（若有可能，可以在读之前清除内存）。一旦装载成功，立即转移至4A00H单元开始的冷启动程序。冷启动程序初始预置零页面，然后转移至CCP（在3400H单元），CCP调用BDOS，BDOS又调用CBIOS。CCP要求CBIOS从第二磁道上读16个扇区，如果读成功，CP/M就显示“A>”。如果遇到故障，可用现有的任何查错功能设断点，并跟踪CBIOS。

(11) 第(10)步完成以后，CP/M便提示用户，要求输入命令。可打入以下命令检查磁盘写操作：

```
SAVE  1  X.COM
```

（所有命令后面都必须跟一回车）

CP/M应再次提示（在完成几次磁盘存取之后）：

```
A>
```

如果不是这样，找出磁盘写功能中的错误，重试。

(12) 打入

```
DIR
```

检查列目录命令。CP/M应该响应：

A： X COM

(13) 打入

ERA X.COM

检查删除命令，CP/M 还应响应 A 提示符。进行到这一步，就有了一个只需要引导装载程序就完全能够运行的操作系统。

(14) 写与 GETSYS 类似的一个引导装载程序，用 PUTSYS 将它放入 0 磁道第 1 扇区（仍使用测试盘，不能用原系统盘）。有关引导操作的信息见第五节和第八节。

(15) 用经过 (11)、(12)、(13) 步得到的引导装载程序重新检查新的测试盘。完成这些测试之后，打Control-C (Control 键与 C 键同时按)。系统应执行“热启动”，重新引导系统，然后显示 A>提示符。

(16) 这时，测试盘上已经有了一个可用的 CP/M 系统的用户化文本。用 GETSYS 从测试盘上装入CP/M，拿掉测试盘，在驱动器中插入原系统盘（或合法复制盘），并用 PUTSYS 改制后的文本代替原版本。因为这一步要将数字研究公司提供的系统破坏，所以在不能确信修改准确无误时不要作这项代替工作。

(17) 装入修改过的 CP/M 系统，用

DIR

命令测试，CP/M 应显示由初始预置的磁盘提供的文件明细表。其中有一个文件应是查错程序的内存映像，称为 DDT.COM。

注：从此以后，每次取出磁盘或插入另一磁盘都要重新引导 CP/M系统，除非对新放入的磁盘只读。这一点很重要。

(18) 打入

DDT

命令，装载并测试查错程序（操作过程见“CP/M 动态查错工具 (DDT)”一文）。花一些时间熟悉 DDT，在进行以后的步骤时，它会成为你最有用的工具。

(19) 在进一步修改 CBIOS 之前，先练习一下使用编辑程序（见 ED 用户使用说明）和汇编程序（见 ASM 用户使用说明）。然后用 ED、ASM 和 DDT 重新编写，调试 GETSYS、PUTSYS 及 CBIOS 程序。编写并调试能进行一个盘与另一个盘之间扇区到扇区复制的 COPY 程序，并用它产生一原版盘的后备复制盘。注意：拷贝程序时应在每个复制盘上写入版权标志：

Copyright (c) 1979
Digital Research

(20) 修改 CBIOS，给它加上穿孔机、读带机的操作功能以及开机显示信息，如果系统存在其他驱动器，则还要加上相应的功能。可以用已经开发的 GETSYS 和 PUTSYS 来完成这些修改，或者参阅以后的章节，其中总体介绍了 CP/M的各种功能，对重新生成过程会有帮助。

至此，已形成了改制 CP/M系统的一个能正常工作的文本。注意：尽管这个已开发出来的 CP/M 中的 CBIOS 部分属于你，但所产生的修改 CP/M文本只能为你自己复制使用（见许可协定），而不能非法地复制为他人使用。

应当注意，该系统的文件仍保持与其他 CP/M 系统的文件兼容，这使得 CP/M 用户之

间的非专利软件可以相互交流。

第三节　二级系统生成

既然CP/M系统已投入运行，下一步就要将CP/M与你自己的内存容量相匹配。一般来说，可先用“MOVCPM”程序得到一个CP/M的内存映像，并把这个映像放到一个已命名的磁盘文件上。这个磁盘文件就可通过使用查错程序以及系统生成程序进行装载、检查、修改及替换。这些程序的详细操作请参阅“CP/M特性和功能”一章。

CBIOS和BOOT可使用ED修改，用ASM汇编，产生文件称为CBIOS·HEX和BOOT·HEX，包含CBIOS和BOOT机器码（以Intel十六进制格式）。

为将CP/M的内存映像读入按所需内存容量配置的TPA，可打入命令：

```
MOVCPM  xx  *
```

其中“xx”是用十进制K字节（如，32为32K）表示的内存容量。则得到响应：

```
CONSTRUCTING  xxK  CP/M  VERS  2.0
READY  FOR  “SYSGEN”  OR
 “SAVE  34  CPMxx . COM”
```

这时，TPA中的CP/M内存映像便配置成所需要的内存容量。内存映像放在0900H至227FH（即：BOOT在0900H，CCP在980H，BDOS始于1180H，BIOS在1F80H）。注意：内存映像中含有标准MDS—800BIOS与BOOT程序。此时有必要把内存映像保存在一文件中，以便将CBIOS与CBOOT插入：

```
SAVE  34  CPMxx . COM
```

由“MOVCPM”产生的内存映像装入TPA中的自由区，与其本来的位置相差一个负偏差，因此不影响内存高地址部分的CP/M的操作。这个内存映像然后可由DDT装入，检查或修改，为产生一个新系统作准备。用DDT装载内存映像可打入：

```
DDT  CPMxx . COM
 （装载DDT，然后读入CPM映像）
```

DDT应响应：

```
NEXT  PC
2300  0100
-     （DDT提示符）
```

这时可使用显示命令（D）和反汇编命令（L）检查内存映像中900H至227H的部分。但是要注意，要找到内存映像中的某个特定地址，必须用负偏差加上CP/M地址以得到实际地址。0磁道1扇区装在900H单元（应在900H至97FH寻找冷启动装载程序），0磁道2扇区装在980H（它是CCP基地址），依此类推，装完整个CP/M。例如，对20K系统，CCP驻留在CP/M地址3400H，但SYSGEN程序将它放在980H。所以，以n表示的负偏差满足等式：

$$3400H + n = 980H \quad 或 \quad n = 980H - 3400H$$

若用二的补码算法，n = D580H，可由下式验证：

$$3400H + D580H = 10980H = 0980H \text{（舍去高位溢出）}$$

对于更大的系统，n 满足：

$$(3400H + b) + n = 980H$$

或 $$n = 980H - (3400H + b)$$

或 $$n = D580H - b$$

下面给出一般 CP/M 系统的 n 值：

内存容量	偏差 b	负偏差 n
20K	0000H	D580H − 0000H = D580H
24K	1000H	D580H − 1000H = C580H
32K	3000H	D580H − 3000H = A580H
40K	5000H	D580H − 5000H = 8580H
48K	7000H	D580H − 7000H = 6580H
56K	9000H	D580H − 9000H = 4580H
62K	A800H	D580H − A800H = 2D80H
64K	B000H	D580H − B000H = 2580H

例如，假设在 DDT 状态下，要查找一个20K系统内存映像中的地址 x，先打入

Hx,n　　十六进制和、差

DDT 将显示 x+n（和）与 x−n（差）的值。DDT 显示的第一个数字就是映像中数据或代码所在的内存实际地址。例如，输入

H3400, D580

将得出和980H，它就是 DDT 状态下 CCP 在内存映像中的位置。

用 L 命令反汇编位于（4A00H + b）− n 处的 BIOS 部分，当使用 H 命令时产生实地址为 1F80H，所以反汇编命令为：

L1F80

此时有必要将 CBOOT 和 CBIOS 程序插入。BOOT 位于映像中的0900H 单元。如果实际装载地址是'n'，则可用下面的命令计算偏差（m）：

H900,n　　目标地址减去装载地址

响应该命令而显示的第 2 个数便是所需要的偏差（m）。例如：若 BOOT 在80H 处运行，则命令：

H900,80

将得到响应：

0980　0880　　十六进制和与差

所以，偏差"m"为0880H。

给以下命令，读入 BOOT：

ICBOOT . HEX　输入文件 CBOOT . HEX，然后：

Rm　读入CBOOT，偏差m(=900H − n)

然后，可用这条命令检查CBOOT：

L900

现在准备替换CBIOS。检查CBIOS原先所在位置1F80H区域，然后打入：

ICBIOS. HEX　为装载准备十六进制文件

假定CBIOS要并入20K CP/M系统，所以其基址在4A00H。为了在DDT状态下正确定出CBIOS在内存映像中的位置，在装载十六进制文件时，必须对20K系统给出负偏差n。打入以下命令可完成这项工作：

```
RD580     读入文件，偏差 D580H
```

读操作结束，再检查CBIOS装入的区域（用L1F80命令）以确信CBIOS已被正确装入。如果对所作的修改感到满意，用 Control-C或“G 0”命令退出DDT。

现在再用SYSGEN将修改好的内存映像放回磁盘（若不能保证修改正确无误，请用测试盘），其过程如下：

```
SYSGEN          启动SYSGEN程序
SYSGEN VERSION2.0  SYSGEN的开始信息
SOURCE DRIVE NAME  (OR RETURN TO SKIP)
                打回车跳过CP/M读入操作，因为系统已在内存中
DESTINATION DRIVE NAME (OR RETURN TO SKIP)
                打入B以便将新系统写入B驱动器中的磁盘
DESTINATION ON B, THEN TYPE RETURN
                将测试盘放入B驱动器，打回车完成实际写操作
FUNCTION COMPLETE
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
                将测试磁盘放入A驱动器并冷启动，建立新的CP/M系统。
```

第四节　GETSYS 与 PUTSYS 程序样本

以下程序为第二节中提到的GETSYS和PUTSYS程序提供了一个程序结构。用户必须添上用于读写特定扇区的子程序READSEC和WRITESEC。

```
;   GETSYS PROGRAM-READ TRACKS 0 AND 1 TO MEMORY AT 3380H
;   REGISTER                USE
;       A           (SCRATCH REGISTER)
;       B           TRACK COUNT(0, 1)
;       C           SECTOR COUNT(1,2,…,26)
;       DE          (SCRATCH REGISTER PAIR)
;       HL          LOAD ADDRESS
;       SP          SET TO STACK ADDRESS
;
START:  LXI   SP,3380H  ; SET STACK POINTER TO SCRATCH AREA
        LXI   H, 3380H  ; SET BASE LOAD ADDSESS
        MVI   B, 0      ; START WITH TRACK 0
RDTRK:                  ; READ NEXT TRACK(INITIALLY 0)
        MVI   C, 1      ; READ STARTING WITH SECTOR 1
```

```
RDSEC:                  ; READ NEXT SECTOR
        CALL  READSEC   ; USER-SUPPLIED SUBROUTINE
        LXI   D,128     ; MOVE LOAD ADDRESS TO NEXT 1/2 PAGE
        DAD   D         ; HL = HL + 128
        INR   C         ; SECTOR = SECTOR + 1
        MOV   A,C       ; CHECK FOR END OF TRACK
        CPI   27
        JC    RDSEC     ; CARRY GENERATED IF SECTOR<27
;
;   ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
        INR   B
        MOV   A,B       ; TEST FOR LAST TRACK
        CPI   2
        JC    RDTRK     ; CARRY GENERATEO IF TRACK<2
;
;   ARRIVE HERE AT END OF LOAD, HALT FOR NOW
        HLT
;
;   USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
;   ENTER WITH TRACK NUMBER IN REGISTER B,
;          SECTOR NUMBER IN REGISTER C, AND
;          ADDRESS TO FILL IN HL
;
        PUSH    B       ; SAVE B AND C REGISTERS
        PUSH    H       ; SAVE HL REGISTERS
        ..........................................................
        perform disk read at this point, branch to

        label START if an error occurs

        ..........................................................
        POP     H       ; RECOVER HL
        POP     B       ; RECOVER B AND C REGISTERS
        RET             ; BACK TO MAIN PROGRAM

        END     START
```

注意，该程序经汇编在附录C列出供参考，假定起始地址在0100H。列在左边的是十六进制的操作码，如果程序需要从计算机前面板开关上送入，则可能会用到它。

PUTSYS程序，只需对上面给出的 GETSYS 程序作一点修改就可得到，如附录 D。寄存器对 HL 变为转储地址（下一个要写的地址），而程序内对该寄存器对的操作并不改变。READSEC 子程序由一完成相反功能的子程序 WRITESEC 代替：HL 地址中的数据写入寄存器B给出的磁盘和寄存器C给出的扇区。通常，在调试开发的过程中，将GETSYS与 PUTSYS合并为一个程序常常是很有用的，如附录中所示。

第五节　磁盘组织

这里给出标准CP/M 系统盘的扇区分布，以供参考。扇区 1（见图 1）中放了一个可选择的软件引导程序。磁盘驱动器通常设计成能将 0 道 1 扇区读入内存规定地址(一般在0000H 单元）。这个扇区中的程序，称为 BOOT，作用是将其余的扇区读入内存，始于3400H＋b。如果用户的控制器中没有内部扇区装载功能，则可跳过 0 道 1 扇区上的程序，开始就将 0 道 2 扇区装入3400H＋b。

例如，Intel MDS-800硬件冷启动装入程序将 0 道 1 扇区装入内存绝对地址 3000H。该扇区装入后控制转移至3000H 单元，开始引导操作，将 0 磁道其余部分及 1 磁道全部装入内存，始于3400H＋b。用户须注意，如果在非MDS 环境，引导装入程序几乎没有用，但还是有必要了解它，因为其中一些功能用户在冷启动装载程序中也需要。

图 1　磁盘分布

道号 00	扇区号 01	页面号	内存地址 （引导地址）	CP/M模块名 冷启动引导装入程序
00	02	00	3400H＋b	CCP
00	03	00	3480H＋b	CCP
00	04	01	3500H＋b	CCP
00	05	01	3580H＋b	CCP
00	06	02	3600H＋b	CCP
00	07	02	3680H＋b	CCP
00	08	03	3700H＋b	CCP
00	09	03	3780H＋b	CCP
00	10	04	3800H＋b	CCP
00	11	04	3880H＋b	CCP
00	12	05	3900H＋b	CCP
00	13	05	3980H＋b	CCP
00	14	06	3A00H＋b	CCP
00	15	06	3A80H＋b	CCP
00	16	07	3B00H＋b	CCP
00	17	07	3B80H＋b	CCP

00	18	08	3C00H + b	BDOS
00	19	08	3C80H + b	BDOS
00	20	09	3D00H + b	BDOS
00	21	09	3D80H + b	BDOS
00	22	10	3E00H + b	BDOS
00	23	10	3E80H + b	BDOS
00	24	11	3F00H + b	BDOS
00	25	11	3F80H + b	BDOS
00	26	12	4000H + b	BDOS
01	01	12	4080H + b	BDOS
01	02	13	4100H + b	BDOS
01	03	13	4180H + b	BDOS
01	04	14	4200H + b	BDOS
01	05	14	4280H + b	BDOS
01	06	15	4300H + b	BDOS
01	07	15	4380H + b	BDOS
01	08	16	4400H + b	BDOS
01	09	16	4480H + b	BDOS
01	10	17	4500H + b	BDOS
01	11	17	4580H + b	BDOS
01	12	18	4600H + b	BDOS
01	13	18	4680H + b	BDOS
01	14	19	4700H + b	BDOS
01	15	19	4780H + b	BDOS
01	16	20	4800H + b	BDOS
01	17	20	4880H + b	BDOS
01	18	21	4900H + b	BDOS
01	19	21	4980H + b	BDOS
01	20	22	4A00H + b	BIOS
01	21	22	4A80H + b	BIOS
01	23	23	4B00H + b	BIOS
01	24	23	4B80H + b	BIOS
01	25	24	4C00H + b	BIOS
01	26	24	4C80H + b	BIOS

02-76　　01-26　　目录表和数据

第六节　BIOS 入口点

下面详细介绍由冷启动装载程序和BDOS 指向的 BIOS 入口点。进入 BIOS 是通过位于4A00H＋b 的一个“转跳向量”完成的，见下表（同时参阅附录 B 和C）。转跳向量是顺序的17个转跳指令，它们分别将控制转向各个 BIOS 子程序。对应于某些功能的 BIOS 子程序可能是空的（即：可能只含一条 RET指令），但是转跳向量中的入口必须存在。

4A00H＋b 处的转跳向量呈如下格式，其中左边是各个转跳地址：

```
4A00H＋b    JMP   BOOT      ；冷启动装载转至此处
4A03H＋b    JMP   WBOOT     ；热启动转至此处
4A06H＋b    JMP   CONST     ；检查控制台字符是否准备好
4A09H＋b    JMP   CONIN     ；读入控制台字符
4A0CH＋b    JMP   CONOUT    ；写出控制台字符
4A0FH＋b    JMP   LIST      ；写出列表字符
4A12H＋b    JMP   PUNCH     ；写字符至穿孔机
4A15H＋b    JMP   READER    ；读阅读机
4A18H＋b    JMP   HOME      ；移至选中磁盘上的 0 道
4A1BH＋b    JMP   SELDSK    ；选择磁盘驱动器
4A1EH＋b    JMP   SETTRK    ；置磁道号
4A21H＋b    JMP   SETSEC    ；置扇区号
4A24H＋b    JMP   SETDMA    ；置DMA地址
4A27H＋b    JMP   READ      ；读选中的扇区
4A2AH＋b    JMP   WRITE     ；写选中的扇区
4A2DH＋b    JMP   LISTST    ；返回LIST状态
4A30H＋b    JMP   SECTRAN   ；扇区转换子程序
```

每个转换地址对应于一个特定子程序，完成规定的功能。可把转跳表分成三部分：（1）调用BOOT和WBOOT完成系统重新初始化；（2）调用CONS、CONIN、CONOUT、LIST、PUNCH、READER和LISTST，完成简单字符输入输出；（3）调用HOME、SETTRK、SETSEC、SETDMA、READ、WRITE和SECTRAN，完成磁盘输入输出。

所有简单字符输入输出假设都是对大写或小写的ASCII 字符操作，其最高位（校验位）置零。用 Control—Z（1AH）作为输入设备的文件结束条件。CP/M 将各外设当作“逻辑设备”，并由 BIOS 给它们指定物理设备。BDOS 运行起来只需要 CONST、CONIN 和 CONOUT 子程序（lIST、PUNCH、READER用于PIP 而非BDOS），LISTS 入口当前只用于DESPOOL。所以，CBIOS的初始版本对于其余的ASCII设备可以只有空子程序。每种设备的特性为：

CONSOLE	主要交互式控制台，通过CONST、CONIN和 CONOUT 与操作员联系。通常控制台是诸如 CRT 或电传机之类的设备
LIST	主要列表设备，通常是诸如打印、电传机之类的硬拷贝设备
PUNCH	主要纸带穿孔设备，通常是高速纸带穿孔机或电传机

READER　主要纸带读入设备，例如光电阅读机或电传机

注意，一个外设可以同时指定成LIST、PUNCH和READER。如果不指定任何外设给LIST、PUNCH或READER，则CBIOS应产生相应的错误信息，使得PIP或其他用户程序用到这些设备时，系统不致“挂起”。另外，PUNCH和LIST子程序也可以只是简单地返回，READER子程序返回1AH（Control-Z）放在寄存器A中，指出文件立即结束。

为增加灵活性，用户可以随意使用“IOBYTE”功能，以重新指定逻辑外设与物理外设。IOBYTE功能产生从逻辑外设到物理外设的变换，它在CP/M运行过程中可以修改（见STAT命令）。以下是对应于Intel标准的IOBYTE功能定义：保留一个称为IOBYTE的内存单元（目前为0003H），它定义了逻辑外设到物理外设的变换，并在一段特定时间内生效。实现变换是将IOBYTE分为四个不同的段，每段占2位，分别称为CONSOL、READER、PUNCH和LIST字段，如下所示：

IOBYTE位于0003H

最高位			最低位
LIST	PUNCH	READER	CONSOLE
6、7位	4、5位	2、3位	0、1位

每个字段中的数值可在0～3范围内，定义每个逻辑外设所指定的源设备或目的设备。下列出每个字段所能指定的值：

CONSOLE字段（0、1位）

0——控制台指定为控制台打印设备（TTY:）

1——控制台指定为CRT设备（CRT:）

2——批处理方式：用阅读机作为控制台输入，列表设备作为控制台输出（BAT:）

3——用户定义的控制台设备（UC1:）

READER字段（2、3位）

0——阅读机为电传设备（TTY:）

1——阅读机为高速纸带输入机（PTR:）

2——用户定义的1号阅读机（UR1:）

3——用户定义的2号阅读机（UR2:）

PUNCH字段（4、5位）

0——穿孔机为电传设备（TTY:）

1——穿孔机为高速穿孔设备（PTF:）

2——用户定义的1号穿孔机（UP1:）

3——用户定义的2号穿孔机（UP2:）

LIST字段（6、7位）

0——列表设备为电传机（TTY:）

1——列表设备为CRT设备（CRT:）

2——列表设备为行式打印机（LPT:）

3——用户定义的列表设备（ULI:）

应该注意，IOBYTE功能是任选的，它仅对CBIOS的组织有影响。除了PIP和STAT之外，

CP/M系统并不使用IOBYTE（虽然它存在于系统的0003H单元）。PIP允许存取物理设备，STAT 允许指定或显示逻辑-物理关系（详见“CP/M 特性与功能”）。无论如何，在基本CBIOS完全实现和调试成功以前，不要考虑IOBYTE功能，而应在此之后加上 IOBYTE 以增加功能。

磁盘输入输出总是通过调用一系列各种类型的磁盘存取子程序来实现的。这些子程序可以设置磁盘号、特定磁盘中的道号、扇区号以及输入输出操作所需要的直接存储器存取（DMA）地址。所有这些参数设置后，可调用READ或WRITE功能去完成实际的输入输出。注意，通常调用一次SELDSK选定某个磁盘驱动器，然后进行一系列对该磁盘的读、写操作，直到选择另一个驱动器对它进行操作为止。同样，可以调用一次置 DMA 地址，然后进行一系列对选中的DMA地址读写的操作，直至DMA地址需要改变。在执行READ或 WRITE 操作之前总是要调用磁道及扇区子程序。注意，在向BDOS报告发生错误之前，READ和WRITE子程序应重复执行几遍（标准的是10次）。如果错误信息返回BDOS，BDOS就会向用户报告。根据用户磁盘控制器的特性，HOME 子程序可能实际执行或不执行寻找 0 磁道的功能。重要的是在每次操作之前一定要选中 0 磁道，通常将它与SETTRK 作同样处理，只是参数为 0 。

下面给出每个入口点子程序的作用：

BOOT　BOOT入口点从冷启动装载程序获得控制权，它负责基本系统的初始化，包括显示注册信息（初始文本中可省去）。如果 IOBYTE 功能已经实现，则这时必须设置。对由 WBOOT 入口点设置的各种系统参数必须进行初始化，并将控制转移至3400H＋b 处的 CCP，作进一步处理。注意，寄存器C应置为 0，以选中驱动器A。

WBOOT　WBOOT入口点在热启动时得到控制。每当用户程序转移至0000H单元，或在前面板上复位CPU时，则执行热启动。CP/M系统必须从驱动器 A的开始两个磁道装入，直至BIOS，但不包括BIOS（或CBIOS，若已完成修改）。系统各参数必须按如下要求进行初始化：

- 0、1、2单元　置成JMP WBOOT用于热启动（0000H：JMP 4A03H＋b）
- 3 单元　如 CBIOS 已实现 IOBYTE 功能，则置成IOBYTE的初始值
- 5、6、7单元　置成 JMP BDOS，这是暂驻程序进入 CP/M 的主入口点（0005H:JMP 3C06H＋b）

（整个零页面的详细使用情况见第九节）

完成初始化以后，WBOOT程序必须转移至3400H＋b处的CCP以（重新）启动系统。进入CCP时，寄存器C置成系统初始化后要选择的驱动器。

CONST　检查当前指定的控制台设备的状态，如果字符已准备好等待读取，则寄存器A返回0FFH；如果控制台没有准备好字符，则寄存器A返回 0 。

CONIN　读取下一个控制台字符至寄存器A，并将最高位（校验位）置零。如果控制台没有准备好字符，则等待，直至读到打入的字符才返回。

CONOUT　将寄存器 C 中的字符送至控制台输出设备，字符为 ASCII 形式，最高位

（校验位）为0。如果控制台设备要求每行结束以后有一段时间间隙（如TI Silent 700终端），则送完换行或回车符后应暂停一下，也可将那些对控制台有异常作用的控制符滤掉（例如，Control-Z会导致Lear Seigler终端清除萤光屏）。

LIST　将寄存器C中的字符送入当前指定的列表设备。字符呈ASCII形，检验位为0。

PUNCH　将寄存器C中的字符送至当前指定的穿孔设备。字符呈ASCII形，校验位为0。

RFADER　从当前指定的阅读设备中读取下一个字符放入寄存器A，并置校验位为0（校验位必须为0），返回ASCII字符Control-Z（1AH），报告文件结束。

HOME　将当前选中的磁盘（初始为驱动器A）的磁头返回0道位置。如果控制器允许访问驱动器0道标记，则可让磁头步进，直至检测到0道标记。如果控制器没有这种性能，可将调用HOME转换成参数为0的调用SETTRK。

SELDSK　选中寄存器C中的磁盘驱动器，为以后的操作做准备，其中C等于0为A驱动器，1为B驱动器，2为C驱动器，……15为P驱动器（标准CP/M提供版本可带四个驱动器）。若系统驱动器数小于4，则应在控制台上给出错误信息，并终止运行。建议用户在实际执行输入输出功能（寻道、读或写）时，再进行实际的选驱动器操作，因为磁盘选择通常并非在最后执行输入输出操作时才进行，而许多控制器要在选中新驱动器以后，才对当前磁盘磁头去载。这会产生大量噪音，并使磁盘磨损。

SETTRK　寄存器BC中含有当前选中的磁盘驱动器中下次存取的磁道号。可以立即寻找选中的磁道，也可以等到实际执行下一个读写操作时再寻道。对标准软盘驱动器，寄存器BC可以在对应于有效磁道号0～76范围内取值，对非标准磁盘子系统，取值范围为0～65535。

SETSEC　寄存器BC中含有当前选中的驱动器中下次磁盘存取的扇区号（1～26），可以立即将信息送至控制器，也可以等到执行读写操作时再选择扇区。

SETDMA　寄存器BC含有下次磁盘读写操作的DMA（磁盘存储器存取）地址。例如，如果调用SETDMA时B=00H、C=80H，则此后的读操作都将数据读入80H至0FFH，此后的写操作都从80H至0FFH得到数据，直至下次调用SETDMA。初始的DMA地址假设为80H。注意，控制器实际上并不支持直接存储器存取。如果所有数据传送都通过输入输出口，用户建立的CBIOS可在读写操作时，把从设置的DMA地址开始的128个字节当作内存缓冲区使用。

READ　假定驱动器已选中，磁道和扇区已设置，DMA地址也已经规定，READ子程序将根据这些参数读入一个扇区，并在寄存器A中返回以下错误码：

0　未发生错误

1　发生不可恢复错误

目前，CP/M仅检测返回码为0或非0值，即：如果寄存器A中的值为

0，CP/M认为磁盘操作正确完成。但是，如果发生错误，CBIOS应至少重试10次，看看错误能否恢复。若仍然有错误，BDOS显示出“BDOS-ERR ON X:BAD SECTOR”的信息。此时，操作员可以选择打〈cr〉跳过错误，或者打Control-C退出。

WRITE　将当前选中的DMA地址中的数据写入当前选择的驱动器、磁道和扇区。数据应标成“非删除数据”以保持与其他CP/M系统兼容。寄存器A中返回与READ命令同样的错误码，且作同样的错误恢复。

LISTST　返回列表设备的就绪态。在控制台操作期间，由DESPOOL程序使用，以提高响应速度。如果列表设备处于未准备好接收的状态，则00值返回寄存器A；如果能往打印机传送字符，则返回0FFH。

SECTRAN　执行逻辑扇区到物理扇区的转换，以改善整个CP/M响应。标准CP/M系统中“偏差系数”为6，即在每一个逻辑读操作之间，跳过6个物理扇区，该偏差度允许大部分程序能有足够时间装入缓冲器，不至漏掉下一个扇区。在某些使用快速处理器、存储器和磁盘子系统的计算机系统中，可改变偏差系数以改善整个系统的响应。但要注意，应保留单密度IBM兼容格式的CP/M版本，以便利用偏差系数6作为你的计算机系统与其他系统的信息转换。通常，SECTRAN接收逻辑扇区号放在寄存器BC，将转换表地址放在DE。扇区号用作进入转换表的索引，结果物理扇区号放在HL中。对标准系统，转换表和索引编码在CBIOS中提供，无需改动。

第七节　BIOS样本

附录C中列出的程序可作为第一个BIOS的基础。这个BIOS包含了一些最简单的功能，所以，如果有必要，可将它们从前面板输入到计算机中。注意，对CONST、CONIT、CONOUT、READ、WRITE及WAITIO子程序，用户必须修改并插入代码。在这些区域中已为用户提供的代码保留了存储空间。在零页面为BIOS保留了一个暂存区（见第九节）为该程序使用。所以，如果需要，可以在ROM中实现BIOS。

一旦运行起来，便可以改进这个粗略文本的功能，显示初始的注册信息，完成更好的错误恢复功能。添入LIST、PUNTH及READER子程序，并实现IOBYTE的功能。

第八节　冷启动装载程序样本

附录D中的程序可作为冷启动装载程序的基础。用户必须提供读磁盘功能，且程序必须装至内存0000H单元。注意，已为修改保留了空间，所以冷启动装载程序所需存储器的总容量为128个字节。最后，用户可以将此程序放入磁盘的第一个扇区（0磁道1扇区），并使控制器在系统启动以后能自动将它装入内存。或者，可将装载程序放在ROM中，位于CP/M之上。这时，有必要将程序定在高地址处，并在系统启动时，从键盘输入一转跳指令，使它转移至装载程序。以后的热启动不再需要该键盘输入操作，因为入口点“WBOOT”得到控制权，将系统自动从磁盘读入。还要注意，这个粗略的冷启动装载程序错误恢复功能很弱，

有待在以后的文本中进一步增强。

第九节　零页面保留单元

内存零页面，00H至0FFH中放了一些在CP/M处理过程中要用到的程序及数据段。以下给出这些程序及数据区，以供参考。

单元（从　至）	内容
0000H～0002H	包含指向4A03H+b处热启动入口点的转跳指令。它允许简单的程序重新启动（JMP　0000H）或人工前面板重新启动
0003H～0003H	包含Intel标准IOBYTE，如第六节中所述，它在用户CBIOS中是任选的
0004H～0004H	当前缺省的驱动器号（0=A，……、15=P）
0005H～0007H	包含指向BDOS的转跳指令，它有两个作用：如“CP/M接口指南”中所述，JMP 0005H提供转向BDOS的主要入口点，且LHLD指令可将该转跳指令中的地址字段装入HL寄存器对。这个数值是CP/M占用的内存最低地址（假设CCP被覆盖）。注意，DDT程序将改变该地址字段以反映出查错状态下内存容量的减少
0008H～0027H	（中断单元1至5，未使用）
0030H～0037H	（中断单元6，现在未使用，但保留）
0038H～003AH	重启动7，包含进入DDT或SID程序的转跳指令，在查错状态下程序设断点时用，但CP/M在其他情况下不使用
003BH～003FH	（现在未使用，保留）
0040H～004FH	为CBIOS保留的16个字节的暂存区，但在CP/M提供版本中未使用
0050H～005BH	（现在未使用，保留）
005CH～007CH	缺省文件控制块，由控制台命令处理程序为暂驻程序建立
007DH～007FH	缺省随机记录位置，任选
0080H～00FFH	缺省128字节磁盘缓冲区（CCP装载暂驻程序时，还填入命令行）

注意，这些信息是为CP/M系统下的正常操作建立的，如果暂驻程序不使用BDOS功能，则可由暂驻程序重写。例如，如果某特定程序仅执行简单的输入输出操作，但必须在0单元开始执行，则可先用普通CP/M功能将程序装入TPA，装入以后，让其中的一个小搬家程序得到控制权（内存搬家程序必须从0100H单元得到控制权，这个单元定为所有暂驻程序的起始单元）。然后，搬家程序就开始将整个程序映像往下搬至0单元，并将控制权转移至存储器装入的起始地址。注意，如果BIOS被覆盖，或0单元被重写（原来放热启动入口点），则程序员须用冷启动过程将CP/M系统读回内存。

第十节　磁盘参数表

在BIOS中包含一些表，描述了使用CP/M的磁盘子系统的特性。这些表可以手编，如附录C中CBIOS样本所示，也可以使用DISKDEF 宏定义库自动生成，如附录 B 所示。下面描述这些表的元素。

通常，每个磁盘驱动器都有一个相关的磁盘参数标题（16字节），它包含有关磁盘驱动器的信息，并提供某些BDOS操作用的暂存区。其格式如下：

磁盘参数标题（DPH）

XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV
16位	16位	16位	16位	16位	16位	16位	16位

其中每个元素为一个字（16位），其含义如下：

XLT　逻辑到物理转换向量地址
　　如果不发生扇区转换，即物理扇区号与逻辑扇区号相同，则该值为0000H。
　　扇区偏差系数相同的磁盘驱动器共用同一个转换表

0000　BDOS内使用的暂存器值（其起始值无关紧要）

DIRBUF　BDOS 内目录表操作的128字节暂存器域地址。所有DPH访问相同的暂存器域

DPB　该驱动器磁盘参数块地址。磁盘特性相同的驱动器访问同一磁盘参数块

CSV　更改磁盘软件校验用暂存器域地址，该地址对每一个DPH是不同的

ALV　BDOS 用于保存磁盘位图信息的暂存器域地址，该地址对每一个 DPH 是不同的

假定有几个磁盘驱动器，DPH在表中是这样排列的：第一行16字节对应驱动器0，最后一行对应驱动器n-1。表的形式如下：

DPBASE:

00	XLT00	0000	0000	0000	DIRBUF	DBP00	CSV00	ALV00
01	XLT01	0000	0000	0000	DIRBUF	DBP01	CSV01	ALV01
			⋮					
n-1	XLTn-1	0000	0000	0000	DIRBUF	DBPn-1	CSVn-1	ALVn-1

此处标号DPBASE定义该DPH表的基地址。

SELDSK子程序的作用是对选定磁盘驱动器返回DPH基地址。

下述操作返回该表地址，如果选择磁盘不存在，则返回0000H。

```
NDISKS    EQU    4    ;NUMBER OF DISK DRIVES
.....
SELDSK:
```

```
; SELECT DISK GIVEN BY BC
LXI         H, 0000H ; ERROR  CODE
MOV         A, C     ; DRIVE  OK?
CPI         NDISKS   ; CY IF  SO
RNC                  ; RET IF  ERROR
; NO ERROR, CONTINUE
MOV         L, C     ; LOW(DISK)
MOV         H, B     ; HIGH(DISK)
DAD         H        ; * 2
DAD         H        ; * 4
DAD         H          * 8
DAD         H        ; *16
LXI         D, DPBASE ; FIRST DPH
DAD         D        ; DPH (DISK)
RET
```

转换向量（XLT00至XLTn-1）在BIOS中放在另一处，与逻辑扇区号0至(扇区数-1)简单地一一对应。对每个驱动器的磁盘参数块（DPB）较为复杂，由一个或多个 DPH 访问特定的 DPB，其一般形式为：

SPT	BSH	BLM	EXM	DSM	DRM	AL_0	AL_1	CKS	OFF
16位	8位	8位	8位	16位	16位	8位	8位	16位	16位

其中每个字段取一字节或一个字长。

SPT　每磁道扇区总数

BSH　数据分配块位移因子，由数据块大小决定

EXM　区域屏蔽，由数据块大小及磁盘块数决定

DSM　决定磁盘驱动器总存储容量

DRM　决定目录表可存放在该驱动器上的登记项总数，AL_0、AL_1决定保留目录表块

CKS　目录表校验向量尺寸

OFF　在逻辑磁盘开始处保留磁道数

BSH 和 BLM 值决定数据单元大小 BLS，BLS 不是磁盘参数块登记项，假如设计者已选定 BLS 值，则BSH 和 BLM 值如下：

BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

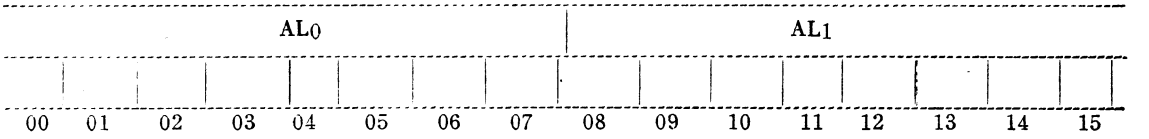
这里所有值均为十进制。EXM 的值取决于 BLS，并取决于 DSM 值是小于还是大于 255，

如下表所示：

BLS	DSM<256	DSM>255
1,024	0	不用
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

DSM 值为该特定驱动器支持的最大数据块数，以 BLS 为单位度量。BLS 乘以 (DSM+1) 的积为该驱动器包含的字节总数。当然，它必须在物理磁盘容量之内，不计保留的操作系统磁道。

DRM 是小于目录表项总数的一个登记项，可取16位值，但 AL0和 AL1的值由 DRM决定。AL0和 AL1两个值可看作一串16位值，如下所示：



这里，位置00对应于标记 AL0字节的高位，15对应于标记 AL1字节的低位。每一位位置保留一个数据块用作目录表项数，因此，共允许16个数据块指定为目录表项（从00位开始向右直至位置15），每个目录表项占32字节，结果如下表：

BLS	目录表项
1,024	32乘以位数
2,048	64乘以位数
4,096	128乘以位数
8,192	256乘以位数
16,384	512乘以位数

如果 DRM=127（128个目录表项），且 BLS=1024，则每块有32个目录表项，需要4个保留块。在这种情况下，AL0 的 4 个高位置 1，结果值AL0=0F0H，AL1 =00H。

CKS 值由下式确定，如果磁盘驱动器存储介质是可拆卸的，则 CKS= (DRM+1)/4，这里 DRM 是目录表最后一个登记项号；如果存储介质是固定的，则置 CKS=0（这时，不进行目录表记录校验）。

最后，OFF字段确定物理磁盘开始处跳过的磁道数。在调用 SETTRK 时，自动地将该值加进去，并可作为跳过保留的操作系统磁道的机构或用于将大磁盘分为较小的节段。

前面介绍过，几个 DPH 可访问同一个 DPB，只要它们的驱动器相同。并且，当一个新驱动器用改变其 DPH 中指针的方法被访问时，DPB可以动态修改。因为当调用SELDSK功能时，BDOS 将 DPB 值拷贝到一个局部区域中。

回过头来看 DPH，对特定驱动器，注意还有两个地址值 CSV 和 ALV，两个地址均引用 BIOS 后未初始化的存储域，该存储域对每个驱动器必须是唯一的，且每个域的大小由 DPB 中的值决定。

被 CSV 访问的域大小是 CKS 个字节，它应足够用于容纳该驱动器的目录表校验信

息。如果 CKS = (DRM+1)/4，则须保留 (DRM+1)/4字节作目录表校验用；如果CKS=0，则无需保留存储区。

被 AVL 访问的域大小是由该特定驱动器所允许的数据块最大数目来决定的，其算式为 (DSM/8)+1。

附录 C 所示 CBIOS 是对标准 8 吋单密度驱动器用的表格实例。它对于检查该程序以及将表格值与上面给出的定义比较是有用的。

第十一节 DISKDEF 宏定义库

附录 F 所示的宏定义库称为 DISKDEF，可大大简化表格的构成步骤。当然，要使用 DISKDEF 功能，必须得会使用 MAC 宏汇编，尽管该宏定义库已包含在所有 CP/M2.0 系统盘上。

BIOS 磁盘定义由下列宏语句组成：

```
MACLIB          DISKDEF
.....
DISKS           n
DISKDEF         0, .....
DISKDEF         1, .....
.....
DISKDEF         n-1
.....
ENDEF
```

这里，MACLIB 语句将 DISKDEF.LIB 文件（与BIOS 在同一磁盘上)装入 MAC 内部表中。接着是 DISKS 宏调用，规定系统配置的磁盘驱动器数目，n 为整数，取值范围1～16。后面是一系列 DISKDEF 宏调用，定义每个逻辑磁盘的特性。0到 n-1，对应于逻辑驱动器 A 到 P。注意，DISKS 和 DISKDEF 宏指令生成一列固定数据表，如上节所述。因此必须放在 BIOS 的非执行部分，一般是直接跟在 BIOS 转移向量之后。

BIOS 的其余部分跟在 DISKDEF 宏指令后面定义，ENDEF 宏调用紧接在 END 语句之前。ENDEF 宏指令生成必要的未初始化 RAM 域，位于存储器中 BIOS 之上。

DISKDEF 宏调用形式是：

DISKDEF dn, fsc, lsc, [skf], bls, dir, cks, ofs, [0]，这里

dn　逻辑磁盘号，0至n-1

fsc　第一个物理扇区号（0或1）

lsc　最后一个扇区号

skf　任选扇区偏差系数

bls　数据单元块大小

dir　目录表登记项数

cks　被校验的目录表登记项数

ofs　对于逻辑磁道00的磁道偏移

〔0〕　任选的1.4版本兼容标记

值“dn”是用 DISKDEF 宏调用定义的驱动器号，“fsc”参数说明不同扇区编号系统，通常是0或1。“lsc”是一个磁道上编号的最后一个扇区。“skf”参数如果出现，该参数定义扇区偏差系数，根据该偏差建立扇区转换表。如果扇区数小于256，则建立单字节表，否则每个转换表元素占2字节。如果忽略 skf 参数（或等于0），则不建立转换表。“bls”参数指定分配至每个数据块的字节数，取值1024、2048、4096、8192或16384。通常，随着数据块尺寸增大，性能提高。因为访问目录表次数减少，且逻辑相关的记录在磁盘物理位置上相互靠近。而且每个目录表登记项访问的数据较多，BIOS 驻留的 RAM 空间减少。“dks”指定磁盘总尺寸，以“bls”为单位，即如果 bls = 2048，dks = 1000，则磁盘总容量为2,048,000字节；如果 dks 大于255，则块尺寸参数bls 必须大于1024。“dir”值是目录表登记项总数，如果需要，可超过255。“cks”参数确定每次扫描目录表时校验的目录表项数。它用于系统操作期间，人工干预冷启动或热启动尚未发生时，内部检查变更的磁盘（当检测到这种情况时，CP/M 自动将该磁盘标识为只读盘以免破坏数据）。如上节所述，当存储介质易于改变时，比如软磁盘子系统，cks 值等于 dir，如果磁盘是永久性安装的，则 cks 值一般为0。因为，变更磁盘而不重启动的可能性是很小的。“ofs”值确定当这个特定驱动器被访问时跳过的磁道数，可用于保留附加的操作系统空间或在单个大容量物理驱动器上模拟几个逻辑驱动器。最后，当1.4版本已经修改可用于高密度磁盘时，如果需要与1.4版本文件兼容，则应包括〔0〕参数。该参数保证对每个目录表记录只分配16K，如上节所述，通常该参数是不需要的。

为了方便和合理使用表格空间，用特殊的形式

```
DISKDEF    i, j
```

可以给磁盘 i 与以前定义过的驱动器 j 相同的特性。

标准的四驱动器单密度系统（该系统与1.4版本兼容）是用下列宏调用定义的：

```
DISKS  4
DISKDEF 0, 1, 26, 6, 1024, 243, 64, 64, 2
DISKDEF 1, 0
DISKDEF 2, 0
DISKDFF 3, 0
……
ENDEF
```

假定所有磁盘都有相同的参数值：每个磁道26个扇区（编号1至26），每次访问之间跳过6个扇区，每个数据块1024字节，对243K总磁盘容量有243个数据块，64个被校验目录表项，2个操作系统磁道。

DISKS 宏指令生成 n 个磁盘参数标题（DPH），开始于该宏指令生成的 DPH 表地址 DPBASE。每个磁盘标题块包含16字节，并与所定义的驱动器一一对应。例如，对标准的四驱动器系统，DISKS 宏指令生成的表形式如下：

```
DPBASE EQU  $
DPE0:   DW   XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV0, ALV0
DPE1:   DW   XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV1, ALV1
```

```
DPE2:  DW    XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV2, ALV2
DPE3:  DW    XLT0, 0000H, 0000H, 0000H, DIRBUF, DPB0, CSV3, ALV3
```

这里 DPE 标号用于参考，表示每个驱动器（从0到3）开始表格地址。磁盘参数标题包含的值已在前面详细描述。校验和分配向量地址由 ENDEF 宏指令生成，在 RAM区紧接在 BIOS 编码和表的后面。

要注意，如果“skf”（偏差系数）参数省略或等于0，则转换表省略，并在该磁盘参数标题的 XLT 位置插入一个0000H。在后继调用执行逻辑至物理转换时，SECTRAN 接收转换表地址 DE=0000H，并从 BC 寄存器对将原来的逻辑扇区送回 HL 寄存器对。当出现 skf 参数时，则构成转换表，并将非0的表地址放入相应的 DPH 中。例如，当标准偏差系数 skf=6在 DISKDEF 宏调用中指定时，转换表构成如下所示：

```
XLT0: DB  1, 7, 13, 19, 25, 5, 11, 17, 23, 3, 9, 15, 21
      DB  2, 8, 14, 20, 26, 6, 12, 18, 24, 4, 10, 16, 22
```

在 ENDEF 宏调用后面，定义了一些未初始化数据区。这些数据区可以不必作为 BIOS 的组成部分在冷启动时装入，但必须在 BIOS 与内存结束地址之间可用。未初始化 RAM 区的大小由 ENDEF 宏指令生成的 EQU 语句确定。对标准的四驱动器系统，ENDEF 宏指令可能产生：

```
4C72=  BEGDAT EQU$
（数据区）
4DB0=  ENDDAT EQU$
013C =  DATSLZ  EQU$-BFGDAT
```

指出未初始化 RAM 开始于4C72H 单元，结束于4DB0H-1单元，占据013CH字节。在系统装入后，必须保证这些地址可以自由使用。

修改后可以使用 STAT 程序来校验驱动器特性，因为 STAT 使用磁盘参数块对驱动器信息进行译码。STAT 命令格式

```
STAT  d:DSK:
```

对驱动器 d 磁盘参数块（d=A，……P）译码并显示下面的值：

- r: 128字节记录容量
- k: 驱动器容量（K字节）
- d: 32字节目录表项
- c: 被校验目录表项
- e: 每个磁盘区的记录数
- b: 每块的记录数
- s: 每磁道的扇区数
- t: 保留磁道数

下面是三个 DISKDEF 宏调用及对应的 STAT 参数值的例子：

```
                DISKDEF 0, 1, 58, , 2048, 256, 128, 128, 2
r=4096, k=512, d=128, c=128, e=256, b=16, s=58, t=2
                DISKDEF 0, 1, 58, 2048, 1024, 300, 0, 2
r=16384, k=2048, d=300, c=0, e=128, b=16, s=58, t=2
```

DISKDEF 0, 1, 58, , 16384, 512, 128, 128, 2

r=65536, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2

第十二节　扇区的组块和解块

在每次调用 BIOS WRITE 入口点时，如果主磁盘子系统扇区大小为基本的128字节的倍数，则CP/M BDOS 包含有效的扇区组块和解块的信息。这里提供一种通用算法，可以包括在 BIOS 中，并可使用 BDOS 信息自动执行该操作。

每次调用 WRITE 时， BDOS 在寄存器 C 中提供下列信息：

0=写普通扇区

1=写至目录表扇区

2=写至新数据块的第一个扇区

当下一个写操作写至以前写过的区域，例如进行随机方式记录更新，当写至未分配数据块第一个扇区之外或不是写入目录表区时，出现条件0；当写入目录表区时，出现条件1；当只写至新分配的数据块第一个记录时，出现条件2。在大多数情况下，应用程序按顺序读写多个128字节扇区，因此在读写操作中记录组块解块时开销很小，因为当写记录时可避免预先读入操作。

附录G 列出组块解块算法的大略形式。通常，该算法将所有 CP/M 扇区读操作，通过一个中间缓冲器（大小相当于主磁盘扇区）映照到主磁盘上。所有的程序值和变量与CP/M 寻道操作中的扇区有关的均以“sek”为前缀，而所有与主磁盘系统有关的均以“hst”为前缀。附录 G 中29行开始 equate语句定义 CP/M 与主系统之间的映象，如果不只是包含样本主系统，则须作改变。

入口点 BOOT 和WBOOT 必须包含初始化代码，从57行开始；而入口点 SELDSK 必须从65行开始增添代码。注意，虽然入口点 SELDSK计算并返回磁盘参数标题地址，但在该点上它并不实际选定主磁盘（在后面 READHST 或 WRITEHST 选定）。并且，SETTRK, SETSEC 和 SETDMA 也只是简单地存放这些值，而不采取任何其他动作。SECTRAN 只执行返回物理扇区号的功能。

READ 和 WRITE 是主要入口点，分别从110行和125行开始。这些子程序执行前面的 READ 和 WRITE 操作。

实际物理读写操作在 WRITEHST 或 READHST 发生，那里所有值均已准备好：hstdsk 是主磁盘数，hsttrk 是主磁道数，hstsec 是主扇区数（可能需转换为物理扇区数）。在该点必须插入代码，执行将全部主扇区写入缓冲器或从缓冲器读出。缓冲器在 hstbuf，长度为 hstsiz。所有其池映象功能均由该算法执行。

这个算法曾使用80MB 硬磁盘设备检查过。该磁盘设备原来配置为128字节扇区，产生约 35MB 格式化存储区，当配置为512字节主扇区时，可用存储区增加至 57MB，而且整个响应速度改善400%。在这种情形下，扇区解块并无明显开销，且用户程序仍保持128字节扇区。当然，这主要是由于 BDOS 提供的信息避免了发生预读操作的必要性。

附录A　MDS 冷启动装入程序

```
                ; MDS-800 Cold Start Loader for CP/M 2.0
                ;
                ; Version 2.0 August, 1979
                ;
0000 =          false   equ     0
ffff =          true    equ     not false
0000 =          testing equ     false
                ;
                        if      testing
                bias    equ     03400h
                        endif
                        if      not testing
0000 =          bias    equ     0000h
                        endif
0000 =          cpmb    equ     bias            ; base of dos load
0806 =          bdos    equ     806h+bias       ; entry to dos for calls
1880 =          bdose   equ     1880h+bias      ; end of dos load
1600 =          boot    equ     1600h+bias      ; cold start entry point
1603 =          rboot   equ     boot+3          ; warm start entry point
                ;
3000                    org     3000h           ; loaded here by hardware
                ;
1880 =          bdosl   equ     bdose-cpmb
0002 =          ntrks   equ     2               ; tracks to read
0031 =          bdoss   equ     bdosl/128       ; # sectors in bdos
0019 =          bdos0   equ     25              ; # on track 0
0018 =          bdos1   equ     bdoss-bdos0     ; # on track 1
                ;
f800 =          mon80   equ     0f800h  ; intel monitor base
ff0f =          rmon80  equ     0ff0fh  ; restart location for mon80
0078 =          base    equ     078h    ; 'base' used by controller
0079 =          rtype   equ     base+1  ; result type
007b =          rbyte   equ     base+3  ; result byte
007f =          reset   equ     base+7  ; reset controller
                ;
0078 =          dstat   equ     base    ; disk status port
0079 =          ilow    equ     base+1  ; low iopb address
```

```
007a =      ihigh   equ   base+2    ; high iopb address
00ff =      bsw     equ   0ffh      ; boot switch
0003 =      recal   equ   3h        ; recalibrate selected drive
0004 =      readf   equ   4h        ; disk read function
0100 =      stack   equ   100h      ; use end of boot for stack
            ;
            rstart:
3000 310001         lxi   sp,stack  ; in case of call to mon80
            ;       clear disk status
3003 db79           in    rtype
3005 db7b           in    rbyte
            ;       check if boot switch is off
            coldstart:
3007 dbff           in    bsw
3009 e602           ani   02h       ; switch on?
300b c20730         jnz   coldstart
            ;       clear the controller
300e d37f           out   reset     ; logic cleared
            ;
            ;
3010 0602           mvi   b,ntrks   ; number of tracks to read
3012 214230         lxi   h,iopb0
            ;
            start:
            ;
            ;       read first/next track into cpmb
3015 7d             mov   a,l
3016 d379           out   ilow
3018 7c             mov   a,h
3019 d37a           out   ihigh
301b db78   wait0:  in    dstat
301d e604           ani   4
301f calb30         jz    wait0
            ;
            ;       check disk status
3022 db79           in    rtype
3024 e603           ani   11b
3026 fe02           cpi   2
            ;
```

```
                        if      testing
                        cnc     rmon80      ; go to monitor if 11 or 10
                        endif
                        if      not testing
3028 d20030             jnc     rstart      ; retry the load
                        endif
                ;
302b db7b               in      rbyte       ; i/o complete, check status
                ;       if not ready, then go to mon80
302d 17                 ral
302e dc0fff             cc      rmon80      ; not ready bit set
3031 1f                 rar                 ; restore
3032 e61e               ani     11110b      ; overrun/addr err/seek/crc
                ;
                        if      testing
                        cnz     rmon80      ; go to monitor
                        endif
                        if      not testing
3034 c20030             jnz     rstart      ; retry the load
                        endif
                ;
                ;
3037 110700             lxi     d,iopbl     ; length of iopb
303a 19                 dad     d           ; addressing next iopb
303b 05                 dcr     b           ; count down tracks
303c c21530             jnz     start
                ;
                ;
                ;       jmp boot, print message, set-up jmps
303f c30016             jmp     boot
                ;
                ;       parameter blocks
3042 80         iopb0:  db      80h         ; iocw,no update
3043 04                 db      readf       ; read function
3044 19                 db      bdos0       ; # sectors to read trk 0
3045 00                 db      0           ; track 0
3046 02                 db      2           ; start with sector 2, trk 0
3047 0000               dw      cpmb        ; start at base of bdos
0007 =          iopbl   equ     $-iopb0
```

```
                ;
3049 80     iopbl:  db      80h
304a 04             db      readf
304b 18             db      bdos1       ; sectors to read on track 1
304c 01             db      1           ; track 1
304d 01             db      1           ; sector 1
304e 800c           dw      cpmb+bdos0*128  ; base of second rd
3050                end
```

附录B MDS 基本I/O系统(BIOS)

```
                ;       mds-800 i/o drivers for cp/m 2.0
                ;       (four drive single density version)
                ;
                ;       version 2.0 august, 1979
                ;
0014 =      vers    equ     20          ; version 2.0
                ;
                ;       copyright(c)1979
                ;       digital research
                ;       box 579, pacific grove
                ;       california 93950
                ;
4a00                org     4a00h       ; base of bios in 20k system
3400 =      cpmb    equ     3400h       ; base of cpm ccp
3c06 =      bdos    equ     3c06h       ; base of bdos in 20k system
1600 =      cpml    equ     $-cpmb      ; length (in bytes) of cpm system
002c =      nsects  equ     cpml/128;   number of sectors to load
0002 =      offset  equ     2           ; number of disk tracks used by cp
0004 =      cdisk   equ     0004h       ; address of last logged disk
0080 =      buff    equ     0080h       ; default buffer address
000a =      retry   equ     10          ; max retries on disk i/o before e
                ;
                ;       perform following functions
                ;       boot    cold start
                ;       wboot   warm start (save i/o byte)
                ;       (boot and wboot are the same for mds)
                ;       const   console status
                ;               reg-a=00 if no character ready
```

```
                ;               reg-a = ff if character ready
                ;       conin   console charcter in (result in reg-a)
                ;       conout  console charcter out (char in reg-c)
                ;       list    list out (char in reg-c)
                ;       punch   punch out (char in reg-c)
                ;       reader  paper tape reader in (result to reg-a)
                ;       home    move to track 00
                ;
                ;       (the following calls set-up the io parameter bloc
                ;       mds, which is used to perform subsequent reads an
                ;       seldsk  select disk given by reg-c (0,1,2···)
                ;       settrk  set track address (0,···,76) for sub r/w
                ;       setsec  set sector address (1,···,26)
                ;       setdma  set subsequent dma address (initially 80h
                ;
                ;       read/write assume previous calls to set i/o parms
                ;       read    read track/sector to preset dma address
                ;       write   write track/sector from preset dma addres
                ;
                ;       jump vector for indiviual routines
4a00 c3b34a             jmp     boot
4a03 c3c34a wboote:     jmp     wboot
4a06 c3614b             jmp     const
4a09 c3644b             jmp     conin
4a0c c36a4b             jmp     conout
4a0f c36d4b             jmp     list
4a12 c3724b             jmp     punch
4a15 c3754b             jmp     reader
4a18 c3784b             jmp     home
4a1b c37d4b             jmp     seldsk
4a1e c3a74b             jmp     settrk
4a21 c3ac4b             jmp     setsec
4a24 c3bb4b             jmp     setdma
4a27 c3c14b             jmp     read
4a2a c3ca4b             jmp     write
4a2d c3704b             jmp     listst      ; list status
4a30 c3b14b             jmp     sectran
                ;
                        maclib diskdef      ; load the disk definition library
```

```
                        disks   4               ; four disks
4a33 + =        dpbase  equ     $               ; base of disk parameter blocks
4a33 + 824a00   dpe0:   dw      xlt0,0000h      ; translate table
4a37 + 000000           dw      0000h,0000h     ; scratch area
4a3b + 6e4c73           dw      dirbuf,dpb0     ; dir buff,parm block
4a3f + 0d4dee           dw      csv0,alv0       ; check, alloc vectors
4a43 + 824a00   dpe1:   dw      xlt1,0000h      ; translate table
4a47 + 000000           dw      0000h,0000h     ; scratch area
4a4b + 6e4c73           dw      dirbuf,dpb1     ; dir buff,parm block
4a4f + 3c4d1d           dw      csv1,alv1       ; check, alloc vectors
4a53 + 824a00   dpe2:   dw      xlt2,0000h      ; translate table
4a57 + 000000           dw      0000h,0000h     ; scratch area
4a5b + 6e4c73           dw      dirbuf,dpb2     ; dir buff,parm block
4a5f + 6b4d4c           dw      csv2,alv2       ; check, alloc vectors
4a63 + 824a00   dpe3:   dw      xlt3,0000h      ; translate table
4a67 + 000000           dw      0000h,0000h     ; scratch area
4a6b + 6e4c73           dw      dirbuf,dpb3     ; dir buff,parm block
4a6f + 9a4d7b           dw      csv3,alv3       ; check, alloc vectors
                        diskdef 0,1,26,6,1024,243,64,64,offset
4a73 + =        dpb0    equ     $               ; disk parm block
4a73 + 1a00             dw      26              ; sec per track
4a75 + 03               db      3               ; block shift
4a76 + 07               db      7               ; block mask
4a77 + 00               db      0               ; extnt mask
4a78 + f200             dw      242             ; disk size-1
4a7a + 3f00             dw      63              ; directory max
4a7c + c0               db      192             ; alloc0
4a7d + 00               db      0               ; alloc1
4a7e + 1000             dw      16              ; check size
4a80 + 0200             dw      2               ; offset
4a82 + =        xlt0    equ     $               ; translate table
4a82 + 01               db      1
4a83 + 07               db      7
4a84 + 0d               db      13
4a85 + 13               db      19
4a86 + 19               db      25
4a87 + 05               db      5
4a88 + 0b               db      11
4a89 + 11               db      17
```

```
4a8a + 17               db      23
4a8b + 03               db      3
4a8c + 09               db      9
4a8d + 0f               db      15
4a8e + 15               db      21
4a8f + 02               db      2
4a90 + 08               db      8
4a91 + 0e               db      14
4a92 + 14               db      20
4a93 + 1a               db      26
4a94 + 06               db      6
4a95 + 0c               db      12
4a96 + 12               db      18
4a97 + 18               db      24
4a98 + 04               db      4
4a99 + 0a               db      10
4a9a + 10               db      16
4a9b + 16               db      22
                        diskdef 1,0
4a73 + =        dpb1    equ     dpb0        ; equivalent parameters
001f + =        als1    equ     als0        ; same allocation vector size
0010 + =        css1    equ     css0        ; same checksum vector size
4a82 + =        xlt1    equ     xlt0        ; same translate table
                        diskdef 2,0
4a73 + =        dpb2    equ     dpb0        ; equivalent parameters
001f + =        als2    equ     als0        ; same allocation vector size
0010 + =        css2    equ     css0        ; same checksum vector size
4a82 + =        xlt2    equ     xlt0        ; same translate table
                        diskdef 3,0
4a73 + =        dpb3    equ     dpb0        ; equivalent parameters
001f + =        als3    equ     als0        ; same allocation vector size
0010 + =        css3    equ     css0        ; same checksum vector size
4a82 + =        xlt3    equ     xlt0        ; same translate table
                ;       endef occurs at end of assembly
                ;
                ;       end of controller-independent code, the remaini
                ;       are tailored to the particular operating environm
                ;       be altered for any system which differs from the
                ;
```

```
                ;       the following code assumes the mds monitor exists
                ;       and uses the i/o subroutines within the monitor
                ;
                ;       we also assume the mds system has four disk drive
00fd =          revrt   equ     0fdh    ; interrupt revert port
00fc =          intc    equ     0fch    ; interrupt mask port
00f3 =          icon    equ     0f3h    ; interrupt control port
007e =          inte    equ     0111$1110b; enable rst 0(warm boot),rst 7
                ;
                ;       mds monitor equates
f800 =          mon80   equ     0f800h  ; mds monitor
ff0f =          rmon80  equ     0ff0fh  ; restart mon80 (boot error)
f803 =          ci      equ     0f803h  ; console character to reg-a
f806 =          ri      equ     0f806h  ; reader in to reg-a
f809 =          co      equ     0f809h  ; console char from c to console o
f80c =          po      equ     0f80ch  ; punch char from c to punch devic
f80f =          lo      equ     0f80fh  ; list from c to list device
f812 =          csts    equ     0f812h  ; console status 00/ff to register
                ;
                ;       disk ports and commands
0078 =          base    equ     78h     ; base of disk command io ports
0078 =          dstat   equ     base    ; disk status (input)
0079 =          rtype   equ     base+1; result type (input)
007b =          rbyte   equ     base+3; result byte (input)
                ;
0079 =          ilow    equ     base+1; iopb low address (output)
007a =          ihigh   equ     base+2; iopb high address (output)
                ;
0004 =          readf   equ     4h      ; read function
0006 =          writf   equ     6h      ; write function
0003 =          recal   equ     3h      ; recalibrate drive
0004 =          iordy   equ     4h      ; i/o finished mask
000d =          cr      equ     0dh     ; carriage return
000a =          lf      equ     0ah     ; line feed
                ;
                signon: ; signon message: xxk cp/m vers y.y
4a9c 0d0a0a             db      cr,lf,lf
4a9f 3230               db      '20'    ; sample memory size
4aa1 6b2043f            db      'k cp/m vers'
```

```
4aad 322e30          db    vers/10+'0','.',vers mod 10+'0'
4ab0 0b0a00          db    cr,lf,0

                     ;
                     boot: ; print signon message and go to ccp
                     ;     (note: mds boot initialized iobyte at 0003h)
4ab3 310001          lxi   sp,buff+80h
4ab6 219c4a          lxi   h,signon
4ab9 bdd34b          call  prmsg       ; print message
4abc af              xra   a           ; clear accumulator
4abd 320400          sta   cdisk       ; set initially to disk a
4ac0 c30f4b          jmp   gocpm       ; go to cp/m

                     ;
                     ;
                     wboot: ; loader on track 0, sector 1, which will be skippe
                     ;      read cp/m from disk - assuming there is a 128 byte
                     ;      start.
                     ;
4ac3 318000          lxi   sp,buff     ; using dma - thus 80 thru ff ok f
                     ;
4ac6 0e0a            mvi   c,retry     ; max retries
4ac8 c5              push  b
                     wboot0: ; enter here on error retries
4ac9 010034          lxi   b,cpm       ; set dma address to start of disk
4acc cdbb4b          call  setdma
4acf 0e00            mvi   c,0         ; boot from drive 0
4ad1 cd7d4b          call  seldsk
4ad4 0e00            mvi   c,0
4ad6 cda74b          call  settrk      ; start with track 0
4ad9 0e02            mvi   c,2         ; start reading sector 2
4adb cdac4b          call  setsec

                     ;
                     ;     read sectors, count nsects to zero
4ade c1              pop   b           ; 10-error count
4adf 062c            mvi   b,nsects
                     rdsec: ; read next sector
4ae1 c5              push  b           ; save sector count
4ae2 cdc14b          call  read
4ae5 c2494b          jnz   booterr; retry if errors occur
4ae8 2a6c4c          lhld  iod         ; increment dma address
```

```
4aeb 118000             lxi     d,128 ; sector size
4aee 19                 dad     d     ; incremented dma address in hl
4aef 44                 mov     b,h
4af0 4d                 mov     c,l   ; ready for call to set dma
4af1 cdbb4b             call    setdma
4af4 3a6b4c             lda     ios   ; sector number just read
4af7 fela               cpi     26    ; read last sector?
4af9 da054b             jc      rd1
                ; must be sector 26, zero and go to next track
4afc 3a6a4c             lda     iot   ; get track to register a
4aff 3c                 inr     a
4b00 4f                 mov     c,a   ; ready for call
4b01 cda74b             call    settrk
4b04 af                 xra     a     ; clear sector number
4b05 3c         rd1:    inr     a     ; to next sector
4b06 4f                 mov     c,a   ; ready for call
4b07 cdac4b             call    setsec
4b0a c1                 pop     b     ; recall sector count
4b0b 05                 dcr     b     ; done?
4b0c c2e14a             jnz     rdsec
                ;
                ;       done with the load, reset default buffer address
                gocpm:  ; (enter here from cold start boot)
                ;       enable rst0 and rst7
4b0f f3                 di
4b10 3e12               mvi     a,12h ; initialize command
4b12 d3fd               out     revrt
4b14 af                 xra     a
4b15 d3fc               out     intc  ; cleared
4b17 3e7e               mvi     a,inte ; rst0 and rst7 bits on
4b19 d3fc               out     intc
4b1b af                 xra     a
4b1c d3f3               out     icon  ; interrupt control
                ;
                ;       set default buffer address to 80h
4b1e 018000             lxi     b,buff
4b21 cdbb4b             call    setdma
                ;
                ;       reset monitor entry points
```

```
4b24 3ec3           mvi     a,jmp
4b26 320000         sta     0
4b29 21034a         lxi     h,wboote
4b2c 220100         shld    1       ; jmp wboot at location 00
4b2f 320500         sta     5
4b32 21063c         lxi     h,bdos
4b35 220600         shld    6       ; jmp bdos at location 5
4b38 323800         sta     7*8     ; jmp to mon80 (may have been chan
4b3b 2100f8         lxi     h,mon80
4b3e 223900         shld    7*8+1
                ;       leave iobyte set
                ;       previously selected disk was b, send parameter to
4b41 3a0400         lda     cdisk   ; last logged disk number
4b44 4f             mov     c,a     ; send to ccp to log it in
4b45 fb             ei
4b46 c30034         jmp     cpmb
                ;
                ;       error condition occurred, print message and retry
                booterr:
4b49 c1             pop     b       ; recall counts
4b4a 0d             dcr     c
4b4b ca524b         jz      booter0
                ;       try again
4b4e c5             push    b
4b4f c3c94a         jmp     wboot0
                ;
                booter0:
                ;       otherwise too many retries
4b52 215b4b         lxi     h,bootmsg
4b55 cdd34b         call    prmsg
4b58 c30fff         jmp     rmon80  ; mds hardware monitor
                ;
                bootmsg:
4b5b 3f626f4        db      '?boot',0
                ;
                ;
                const:  ; console status to reg-a
                ;       (exactly the same as mds call)
4b61 c312f8         jmp     csts
```

```
                ;
                conin: ; console character to reg-a
4b64 cd03f8             call    ci
4b67 e67f               ani     7fh     ; remove parity bit
4b69 c9                 ret
                ;
                conout: ; console character from c to console out
4b6a 309f8              jmp     co
                ;
                list:   ; list device out
                ;       (exactly the same as mds call)
4b6d c30ff8             jmp     lo
                ;
                listst:
                        ; return list status
4b70 af                 xra     a
4b71 c9                 ret             ; always not ready
                ;
                punch: ; punch device out
                ;       (exactly the same as mds call)
4b72 c30cf8             jmp     po
                ;
                reader: ; reader character in to reg-a
                ;       (exactly the same as mds call)
4b75 c306f8             jmp     ri
                ;
                home:   ; move to home position
                ;       treat as track 00 seek
4b78 0e00               mvi     c,0
4b7a c3a74b             jmp     settrk
                ;
                seldsk: ; select disk given by register c
4b7d 210000             lxi     h,0000h ; return 0000 if error
4b80 79                 mov     a,c
4b81 fe04               cpi     ndisks  ; too large?
4b83 d0                 rnc             ; leave hl = 0000
                ;
4b84 e602               ani     10b     ; 00 00 for drive 0,1 and 10 10 fo
4b86 32664c             sta     dbank   ; to select drive bank
```

```
4b89 79          mov   a,c      ; 00, 01, 10, 11
4b8a e601        ani   1b       ; mds has 0,1 at 78, 2,3 at 88
4b8c b7          ora   a        ; result 00?
4b8d ca924b      jz    setdrive
4b90 3e30        mvi   a,00110000b; selects drive 1 in bank
                 setdrive:
4b92 47          mov   b,a      ; save the function
4b93 21684c      lxi   h,iof    ; io function
4b96 7e          mov   a,m
4b97 e6cf        ani   11001111b; mask out disk number
4b99 b0          ora   b        ; mask in new disk number
4b9a 77          mov   m,a      ; save it in iopb
4b9b 69          mov   l,c
4b9c 2600        mvi   h,0      ; hl=disk number
4b9e 29          dad   h        ; *2
4b9f 29          dad   h        ; *4
4ba0 29          dad   h        ; *8
4ba1 29          dad   h        ; *16
4ba2 11334a      lxi   d,dpbase
4ba5 19          dad   d        ; hl=disk heaber table address
4ba6 c9          ret
                 ;
                 ;
                 settrk:; set track address given by c
4ba7 216a4c      lxi   h,iot
4baa 71          mov   m,c
4bab c9          ret
                 ;
                 setsec:; set sector number given by c
4bac 216b4c      lxi   h,ios
4baf 71          mov   m,c
4bb0 c9          ret
                 sectran:
                               ; translate sector bc using table at de
4bb1 0600        mvi   b,0     ; double precision sector number i
4bb3 eb          xchg          ; translate table address to hl
4bb4 09          dad   b       ; translate(sector) address
4bb5 7e          mov   a,m     ; translated sector number to a
4bb6 326b4c      sta   ios
```

```
4bb9 6f             mov     l,a         ; return sector number in l
4bba c9             ret
                ;
                setdma: ; set dma address given by regs b,c
4bbb 69             mov     l,c
4bbc 60             mov     h,b
4bbd 226c4c         shld    iod
4bc0 c9             res
                ;
                read:   ; read next disk record(assuming disk/trk/sec/dma)
4bc1 0e04           mvi     c,readf     ; set to read function
4bc3 cde04b         call    setfunc
4bc6 cdf04b         call    waitio      ; perform read function
4bc9 c9             ret                 ; may have error set in reg-a
                ;
                ;
                write:  ; disk write function
4bca 0e06           mvi     c,writf
4bcc cde04b         call    setfunc     ; set to write function
4bcf cdf04b         call    waitio
4bd2 c9             ret                 ; may have error set
                ;
                ;
                ;       utility subroutines
                prmsg:  ; print message at h,l to 0
4bd3 7e             mov     a,m
4bd4 b7             ora     a           ; zero?
4bd5 c8             rz
                ;       more to print
4bd6 e5             push    h
4bd7 4f             mov     c,a
4bd8 cd6a4b         call    conout
4bdb e1             pop     h
4bdc 23             inx     h
4bdd c3d34b         jmp     prmsg
                ;
                setfunc:
                ;       set function for next i/o (command in reg-c)
4be0 21684c         lxi     h,iof       ; io function address
```

```
4be3 7e             mov     a,m       ; get it to accumulator for maskin
4be4 e6f8           ani     11111000b; remove previous command
4be6 b1             ora     c         ; set to new command
4be7 77             mov     m,a       ; replaced in iopb
                    ;       the mds-800 controller req's disk bank bit in sec
                    ;       mask the bit from the current i/o function
4be8 e620           ani     00100000b; mask the disk select bit
4bea 216b4c         lxi     h,ios     ; address the sector selec
4bed b6             ora     m         ; select proper disk bank
4bee 77             mov     m,a       ; set disk select bit on/o
4bef c9             ret

                    ;
                    waitio:
4bf0 0e0a           mvi     c,retry   ; max retries before perm error
                    rewait:
                    ;       start the i/o function and wait for completion
4bf2 cd3f4c         call    intype    ; in rtype
4bf5 cd4c4c         call    inbyte    ; clears the controller
                    ;
4bf8 3a664c         lda     dbank     ; set bank flags
4bfb b7               ora   a                 ; zero if drive 0,1 and nz
4bfc 3e67             mvi   a, iopd and 0ffh ; low address for iopb
4bfe 064c             mvi   b, iopd shr 8    ; high address for iopd
4c00 c20b4c           jnz   iodrl    ; drive bank 1?
4c03 d379             out   ilow              ; low address to controlle
4c05 78               mov   a, b
4c06 d37a             out   ihigh    ; high address
4c08 c3104c           jmp   wait0             ; to wait for complete
                    ;
                    iodrl:  ; drive bank 1
4c0b d389             out   ilow+10h          ; 88 for drive bank 10
4c0d 78               mov   a, b
4c0e d38a             out   ihigh+10h
                    ;
4c10 cd594c wait0:  call    instat            ; wait for completion
4c13 e604           ani     iordy           ; ready?
4c15 ca104c         jz      wait0
                    ;
                    ;       check io completion ok
```

```
4c18 cd3f4c          call    intype          ; must be io complete (00)
                ;    00 unlinked i/o complete,    01 linked 1/o comple
                ;    10 disk status changed       11 (not used)
4c1d fe02            cpi     10b             ; ready status change?
4c1d ca324c          jz      wready
                ;
                ;    must be 00 in the accumulator
4c20 b7              ora     a
4c21 c2384c          jnz     werror          ; some other condition, re
                ;    check i/o error bits
4c24 cd4c4c          call    inbyte
4c27 17              ral
4c28 da324c          jc      wready          ; unit not ready
4c2b 1f              rar
4c2c e6fe            ani     11111110b       ; any other errors?
4c2e c2384c          jnz     werror
                ;
                ;    read or write is ok, accumulator contains zero
4c31 c9              ret
                ;
                wready: ;not ready, treat as error for now
4c32 cd4c4c          call    inbyte          ; clear result byte
4035 c3384c          jmp     trycount
                ;
                werror: ;return hardware malfunction (crc, track, seek, e
                ;    the mds controller has returned a bit in each pos
                ;    of the accumulator, corresponding to the conditio
                ;    0    - deleted data (accepted as ok above)
                ;    1    - crc error
                ;    2    - seek error
                ;    3    - address error (hardware malfunction)
                ;    4    - data over/under flow (hardware malfunct
                ;    5    - write protect (treated as not ready)
                ;    6    - write error (hardware malfunction)
                ;    7    - not ready
                ;         (accumulator bits are numbered 7 6 5 4 3 2 1 0)
                ;
                ;    it may be useful to filter out the various condit
                ;    but we will get a permanent error message if it i
```

```
                ;       recoverable. in any case, the not ready conditio
                ;       treated as a separate condition for later improve
                trycount:
                ;       register c contains retry count, decrement 'til z
4c38 0d                 dcr     c
4c39 c2f24b             jnz     rewait          ; for another try
                ;
                ;       cannot recover from error
4c3c 3e01               mvi     a, 1            ; error code
4c3e c9                 ret
                ;
                ;       intype, inbyte, instat read drive bank 00 or 10
4c3f 3a664c     intype: lda     dbank
4c42 b7                 ora     a
4c43 c2494c             jnz.    intypl          ; skip to bank 10
4b46 db79               in      rtype
4c48 c9                 ret
4c49 db89       intypl: in      rtype+10h       ; 78 for 0,1 88 for 2,3
4c4b c9                 ret
                ;
4c4c 3a664c     inbyte: lda     dbank
4c4f b7                 ora     a
4c50 c2564c             jnz     inbytl
4c53 db7b               in      rbyte
4c55 c9                 ret
4c56 db8b       inbytl: in      rbyte+10h
4c58 c9                 ret
                ;
4c59 3a664c     instat: lda     dbank
4c5c b7                 ora     a
4c5d c2634c             jnz     instal
4c60 db78               in      dstat
4c62 c9                 ret
4c63 db88       instal: in      dstat+10h
4c65 c9                 ret
                ;
                ;
                ;
                ;       data areas (must be in ram)
```

```
4c66 00     dbank: db    0          ; disk bank 00 if drive 0,1
                                    ;           10 if drive 2,3
            iopb : ;io parameter block
4c67 80            db    80h        ; normal i/o operation
4c68 04     iof :  db    readf      ; io function, initial read
4c69 01     ion :  db    1          ; number of sectors to read
4c6a 02     iot :  db    offset     ; track number
4c6b 01     ios :  db    1          ; sector number
4c6c 8000   iod :  dw    buff       ; io address
                ;
                ;
                ;       define ram areas for bdos operation
                        endef
4c6e + =    begdat equ   $
4c6e +      dirbuf:ds    128        ; directory access buffer
4cee +      alv0 : ds    31
4d0d +      csv0 : ds    16
4d1d +      alv1 : ds    31
4d3c +      csv1 : ds    16
4d4c +      alv2 : ds    31
4d6b +      csv2 : ds    16
4d7b +      alv3 : ds    31
4d9a +      csv3 : ds    16
4daa + =    enddat equ   $
013c + =    datsiz equ   $-begdat
4daa +             end
```

附录 C　CBIOS 轮廓

```
                ;       skeletal cbios for first level of cp/m 2.0 altera
                ;
0014 =      msize   equ  20         ; cp/m version memory size in kilo
                ;
                ;       "bias" is address offset from 3400h for memory sy
                ;       than 16k (referred to as "b" throughout the text)
                ;
0000 =      bias    equ (msize-20)*1024
3400 =      ccp     equ  3400h+bias         ; base of ccp
3c06 =      bdos    equ  ccp+806h           ; base of bdos
```

```
4a00 =      bios    equ   ccp+1600h          ; base of bios
0004 =      cdisk   equ   0004h     ; current disk number 0=a,...,15=p
0003 =      iobyte  equ   0003h     ; intel i/o byte
            ;
4a00                      org   bios      ; origin of this program
002c =      nsects  equ   ($-ccp)/128       ; warm start sector count
            ;
            ;       jump vector for individual subroutines
4a00 c39c4a         jmp   boot              ; cold start
4a03 c3a64a wboote: jmp   wboot             ; warm start
4a06 c3114b         jmp   const             ; console status
4a09 c3244b         jmp   conin             ; console character in
4a0c c3374b         jmp   conout            ; console character out
4a0f c3494b         jmp   list              ; list character out
4a12 c34d4b         jmp   punch             ; punch character out
4a15 c34f4b         jmp   reader            ; reader character out
4a18 c3544b         jmp   home              ; move head to home positi
4a1b c35a4b         jmp   seldsk            ; select disk
4a1e c37d4b         jmp   settrk            ; set track number
4a21 c3924b         jmp   setsec            ; set sector number
4a24 c3ad4b         jmp   setdma            ; set dma address
4a27 c3c34b         jmp   read              ; read disk
4a2a c3d64b         jmp   write             ; write disk
4a2d c34b4b         jmp   listst            ; return list status
4a30 c3a74b         jmp   sectran           ; sector translate
            ;
            ;       fixed data tables for four-drive standard
            ;       ibm-compatible 8" disks
            ;       disk parameter header for disk 00
4a33 734a00 dpbase: dw    trans, 0000h
4a37 000000         dw    0000h, 0000h
4a3b f04c8d         dw    dirbf, dpblk
4a3f ec4d70         dw    chk00, all00
            ;       disk parameter header for disk 01
4a43 734a00         dw    trans, 0000h
4a47 000000         dw    0000h, 0000h
4a4b f04c8d         dw    dirbf, dpblk
4a4f fc4d8f         dw    chk01, all01
            ;       disk parameter header for disk 02
```

```
4a53 734a00          dw    trans, 0000h
4a57 000000          dw    0000h, 0000h
4a5b f04c8d          dw    dirbf, dpblk
4a5f 0c4eae          dw    chk02, a1102
                ;    disk parameter header for disk 03
4a63 734a00          dw    trans, 0000h
4a67 000000          dw    0000h, 0000h
4a6b f04c8d          dw    dirbf, dpblk
4a6f 1c4ecd          dw    chk03, a1103
                ;
                ;    sector translate vector
4a73 01070d trans:   db    1,7,13,19          ; sectors 1,2,3,4
4a77 19050b          db    25,5,11,17         ; sectors 5,6,7,8
4a7b 170309          db    23,3,9,15          ; sectors 9,10,11,12
4a7f 150208          db    21,2,8,14          ; sectors 13,14,15,16
4a83 141a06          db    20,26,6,12         ; sectors 17,18,19,20
4a87 121804          db    18,24,4,10         ; sectors 21,22,23,24
4a8b 1016            db    16,22              ; sectors 25,26
                ;
                dpblk: ;disk parameter block, common to all disks
4a8d 1a00            dw    26                 ; sectors per track
4a8f 03              db    3                  ; block shift factor
4a90 07              db    7                  ; block mask
4a91 00              db    0                  ; null mask
4a92 f200            dw    242                ; disk size-1
4a94 3f00            dw    63                 ; directory max
4a96 c0              db    192                ; alloc 0
4a97 00              db    0                  ; alloc 1
4a98 1000            dw    16                 ; check size
4a9a 0200            dw    2                  ; track offset
                ;
                ;    end of fixed tables
                ;
                ;    individual subroutines to perform each function
                boot:  ;simplest case is to just perform parameter initi
4a9c af              xra   a                  ; zero in the accum
4a9d 320300          sta   iobyte             ; clear the iobyte
4aa0 320400          sta   cdisk              ; select disk zero
4aa3 c3ef4a          jmp   gocpm              ; initialize and go to cp/
```

```
                ;
                wboot : ;simplest case is to read the disk until all sect
4aa6 318000             lxi   sp, 80h            ; use space below buffer f
4aa9 0e00               mvi   c,0               ; select disk 0
4aab cd5a4b             call  seldsk
4aae cd544b             call  home              ; go to track 00
                ;
4ab1 062c               mvi   b,nsects          ; b counts # of sectors to
4ab3 0e00               mvi   c,0               ; c has the current track
4ab5 1602               mvi   d,2               ; d has the next sector to
                ;       note that we begin by reading track 0, sector 2 s
                ;       contains the cold start loader, which is skipped
4ab7 210034             lxi   h,ccp             ; base of cp/m (initial lo
                load1 : ;load one more sector
4aba c5                 push  b                 ; save sector count, current track
4abb d5                 push  d                 ; save next sector to read
4abc e5                 push  h                 ; save dma address
4abd 4a                 mov   c,d               ; get sector address to register c
4abe cd924b             call  setsec            ; set sector address from register
4ac1 c1                 pop   b                 ; recall dma address to b,c
4ac2 c5                 push  b                 ; replace on stack for later recal
4ac3 cdad4b             call  setdma            ; set dma address from b,c
                ;
                ;       drive set to 0, track set, sector set dma addres
4ac6 cdc34b             call  read
4ac9 fe00               cpi   00h               ; any errors?
4acb c2a64a             jnz   wboot             ; retry the entire boot if an erro
                ;
                ;       no error, move to next sector
4ace e1                 pop   h                 ; recall dma address
4acf 118000             lxi   d,128             ; dma = dma + 128
4ad2 19                 dad   d                 ; new dma address is in h,l
4ad3 d1                 pop   d                 ; recall sector address
4ad4 c1                 pop   b                 ; recall number of sectors remaini
4ad5 05                 dcr   b                 ; sectors = sectors-1
4ad6 caef4a             jz    gocpm             ; transfer to cp/m if all have bee
                ;
                ;       more sectors remain to load, check for track chan
4ad9 14                 inr   d
```

```
4ada 7a          mov  a,d          ; sector = 27?, if so, change tracks
4adb fe1b        cpi  27
4add daba4a      jc   load1        ; carry generated if sector<27
                 ;
                 ;    end of current track, go to next track
4ae0 1601        mvi  d,1          ; begin with first sector of next
4ae2 0c          inr  c            ; track = track + 1
                 ;
                 ;    save register state, and change tracks
4ae3 c5          push b
4ae4 d5          push d
4ae5 e5          push h
4ae6 cd7d4b      call settrk       ; track address set from register
4ae9 e1          pop  h
4aea d1          pop  d
4aeb c1          pop  b
4aec c3ba4a      jmp  load1        ; for another sector
                 ;
                 ;    end of load operation, set parameters and go to c
                 gocpm:
4aef 3ec3        mvi  a,0c3h       ; c3 is a jmp instruction
4af1 320000      sta  0            ; for jmp to wboot
4af4 21034a      lxi  h,wboote     ; wboot entry point
4af7 220100      shld 1            ; set address field for jmp at 0
                 ;
4afa 320500      sta  5            ; for jmp to bdos
4afd 21063c      lxi  h,bdos       ; bdos entry point
4b00 220600      shld 6            ; address field of jump at 5 to bd
                 ;
4b03 018000      lxi  b,80h        ; default dma address is 80h
4b06 cdad4b      call setdma
                 ;
4b09 fb          ei                ; enable the interrupt system
4b0a 3a0400      lda  cdisk        ; get current disk number
4b0d 4f          mov  c,a          ; send to the ccp
4t0e c30034      jmp  ccp          ; go to cp/m for further processin
                 ;
                 ;
                 ;    simple i/o handlers (must be filled in by user)
```

```
                ;       in each case, the entry point is provided, with s
                ;       to insert your own code
                ;
                const:  ;console status, return 0ffh if character ready,
4b11                    ds      10h             ; space for status subroutine
4b21 3e00               mvi     a, 00h
4b23 c9                 ret
                ;
                conin:  ;console character into register a
4b24                    ds      10h             ; space for input routine
4b34 e67f               ani     7fh             ; strip parity bit
4b36 c9                 ret
                ;
                conout: ;console character output from register c
4b37 79                 mov     a, c            ; get to accumulator
4b38                    ds      10h             ; space for output routine
4b48 c9                 ret
                ;
                list:   ;list character from register c
4b49 79                 mov     a, c            ; character to register a
4b4a c9                 ret                     ; null subroutine
                ;
                listst: ;return list status (0 if not ready, 1 if ready)
4b4b af                 xra     a               ; 0 is always ok to return
4b4c c9                 ret
                ;
                punch:  ;punch character from register c
4b4d 79                 mov     a, c            ; character to register a
4b4e c9                 ret                     ; null subroutine
                ;
                ;
                reader: ;read character into register a from reader devic
4b4f 3e1a               mvi     a, 1ah          ; enter end of file for now (repla
4b51 e67f               ani     7fh             ; remember to strip parity bit
4b53 c9                 ret
                ;
                ;
                ;       i/o drivers for the disk follow
                ;       for now, we will simply store the parameters away
```

```
                ;       in the read and write subroutines
                ;
                home :  ;move to the track 00 position of current drive
                ;       translate this call into a settrk call with param
4b54 0e00               mvi   c, 0          ; select track 0
4b56 cd7d4b             call  settrk
4b59 c9                 ret                 ; we will move to 00 on first read
                ;
                seldsk :  ;select disk given by register c
4b5a 210000             lxi   h, 0000h      ; error return code
4b5d 79                 mov   a, c
4b5e 32ef4c             sta   diskno
4b61 fe04               cpi   4             ; must be between 0 and 3
4b63 d0                 rnc                 ; no carry if 4, 5,...
                ;       disk number is in the proper range
4b64                    ds    10            ; space for disk select
                ;       compute proper disk parameter header address
4b6e 3aef4c             lda   diskno
4b71 6f                 mov   1,a           ; 1=disk number 0,1,2,3
4b72 2600               mvi   h,0           ; high order zero
4b74 29                 dad   h             ; *2
4b75 29                 dad   h             ; *4
4b76 29                 dad   h             ; *8
4b77 29                 dad   h             ; *16 (size of each header)
4b78 11334a             lxi   d, dpbase
4b7b 19                 dad   d             ; hl=.dpbase (diskno*16)
4b7c c9                 ret
                ;
                settrk :  ;set track given by register c
4b7d 79                 mov   a, c
4b7e 32e94c             sta   track
4b81                    ds    10h           ; space for track select
4b91 c9                 ret
                ;
                setsec :  ;set sector given by register c
4b92 79                 mov   a, c
4b93 32eb4c             sta    sector
4b96                    ds     10h          ; space for sector select
4ba6 c9                 ret
```

```
                ;
                sectran:
                        ;translate the sector given by bc using the
                        ;translate table given by de
4ba7 eb                 xchg            ; hl=.trans
4ba8 09                 dad   b         ; hl=.trans(sector)
4ba9 6e                 mov   l, m      ; l=trans(sector)
4baa 2600               mvi   h, 0      ; hl=trans(sector)
4bac c9                 ret             ; with value in hl
                ;
                setdma: ;set dma address given by registers b and c
4bad 69                 mov   l, c      ; low order address
4bae 60                 mov   h, b      ; high order address
4baf 22ed4c             shld  dmaad     ; save the address
4bb2                    ds    10h       ; space for setting the dma addres
4bc2 c9                 ret
                ;
                read:   ;perform read operation (usually this is similar
                ;       so we will allow space to set up read command, th
                ;       common code in write)
4bc3                    ds    10h       ; set up read command
4bd3 c3e64b             jmp   waitio    ; to perform the actual i/o
                ;
                write:  ;perform a write operation
4bd6                    ds    10h       ; set up write command
                ;
                waitio: ;enter here from read and write to perform the ac
                ;       operation. return a 00h in register a if the ope
                ;       properly, and 01h if an error occurs during the r
                ;
                ;       in this case, we have saved the disk number in 'd
                ;                     the track number in 'track' (0-76
                ;                     the sector number in 'sector' (1-
                ;                     the dma address in 'dmaad' (0-655
4be6                    ds    256       ; space reserved for i/o drivers
4ce6 3e01               mvi   a, 1      ; error condition
4ce8 c9                 ret             ; replaced when filled-in
                ;
                ;       the remainder of the cbios is reserved uninitiali
```

```
                ;       data area, and does not need to be a part of the
                ;       system memory image (the space must be available,
                ;       however, between "begdat" and "enddat").
                ;
4ce9            track : ds    2             ; two bytes for expansion
4ceb            sector : ds   2             ; two bytes for expansion
4ced            dmaad : ds    2             ; direct memory address
4cef            diskno : ds   1             ; disk number 0-15
                ;
                ;       scratch ram area for bdos use
4cf0 =          begdat  equ   $             ; beginning of data area
4cf0            dirbf : ds    128           ; scratch directory area
4d70            a1100 : ds    31            ; allocation vector 0
4d8f            a1101 : ds    31            ; allocation vector 1
4dae            a1102 : ds    31            ; allocation vector 2
4dcd            a1103 : ds    31            ; allocation vector 3
4dec            chk00 : ds    16            ; check vector 0
4dfc            chk01 : ds    16            ; check vector 1
4e0c            chk02 : ds    16            ; check vector 2
4e1c            chk03 : ds    16            ; check vector 3
                ;
4e2c =          enddat  equ   $             ; end of data area
013c =          datsiz  equ   $-begdat      ; size of data area
4e2c                    end
```

附录D GETSYS 与 PUTSYS 程序轮廓

```
                ;       combined getsys and putsys programs from Sec 4.
                ;       Start the programs at the base of the TPA

0100                    org   0100h

0014 =          msize   equ   20            ; size of cp/m in Kbytes

                ; "bias" is the amount to add to addresses for>20k
                ;       (referred to as "b" throughout the text)

0000 =          bias    equ   (msize-20)*1024
3400 =          ccp     equ   3400h+bias
```

```
3c00 =      bdos    equ   ccp+0800h
4a00 =      bios    equ   ccp+1600h

            ;       getsys programs tracks 0 and 1 to memory at
            ;       3880h + bias

            ;       register              usage
            ;         a                 (scratch register)
            ;         b                 track count (0···76)
            ;         c                 sector count (1···26)
            ;         d, e              (scratch register pair)
            ;         h, l              load address
            ;         sp                set to stack address

            gstart:                                 ; start of getsys
0100 318033         lxi   sp, ccp-0080h             ; convenient plac
0103 218033         lxi   h, ccp-0080h              ; set initial loa
0106 0600           mvi   b, 0                      ; start with trac
            rd$trk:                                 ; read next track
0108 0e01           mvi   c, 1                      ; each track star
            rd$sec:
010a cd0003         call  read$sec                  ; get the next se
010d 118000         lxi   d, 128                    ; offset by one s
0110 19             dad   d                         ;  (hl = hl + 128)
0111 0c             inr   c                         ; next sector
0112 79             mov   a, c                      ; fetch sector nu
0113 fe1b           cpi   27                        ;   and see if la
0115 da0a01         jc    rdsec                     ; <, do one more

            ; arrive here at end of track, move to next track

0118 04             inr   b                         ; track = track + 1
0119 78             mov   a, b                      ; check for last
011a fe02           cpi   2                         ; track = 2?
011c da0801         jc    rd$trk                    ; <, do another

            ; arrive here at end of load, halt for lack of anything b

011f fb             ei
```

```
0120 76             hlt
                ;   putsys program, places memory image starting at
                ;   3880h + bias back to tracks 0 and 1
                ;   start this program at the next page boundary

0200                org   ($ + 0100h) and 0ff00h

                put$sys:
0200 318033         lxi   sp, ccp-0080h       ; convenient plac
0203 218033         lxi   h, ccp-0080h        ; start of dump
0206 0600           mvi   b, 0                ; start with trac
                wr$trk:
0208 0e01           mvi   c, 1                ; start with sect
                wr$sec:
020a cd0004         call  write$sec           ; write one secto
020d 118000         lxi   d, 128              ; length of each
0210 19             dad   d                   ; <hl> = <hl> + 128
0211 0c             inr   c                   ; <c> = <c> + 1
0212 79             mov   a, c                ; see if
0213 fe1b           cpi   27                  ; past end of t
0215 da0a02         jc    wr$sec              ; no, do another

                ; arrive here at end of track, move to next track

0218 04             inr   b                   ; track = track + 1
0219 78             mov   a, b                ; see if
021a fe02           cpi   2                   ; last track
021c da0802         jc    wr$trk              ; no, do another

                ;   done with putsys, halt for lack of anything bette

021f fb             ei
0220 76             hlt

                ; user supplied subroutines for sector read and write

                ;   move to next page boundary

0300                org   ($ + 0100h) and 0ff00h
```

```
                read$sec:
                        ; read the next sector
                        ; track in<b>,
                        ; sector in<c>
                        ; dmaaddr in<hl>

0300 c5                 push  b
0301 e5                 push  h

                ; user defined read operation goes here
0302                    ds    64

0342 e1                 pop   h
0343 c1                 pop   b
0344 c9                 ret

0400                    org   ($ + 0100h) and 0 f00h    ; another page bo

                write$sec:

                ; same parameters as read$sec

0400 c5                 push  b
0401 e5                 push  h

                ; user defined write operation goes here
0402                    ds    64

0442 e1                 pop   h
0443 c1                 pop   b
0444 c9                 ret

                ; end of getsys/putsys program

0445                    end
```

附录 E　冷启动装入程序轮廓

```
;  this is a sample cold start loader which, when modified
;  resides on track 00, sector 01 (the first sector on the
;  diskette). we assume that the controller has loaded
;  this sector into memory upon system start-up (this pro-
;  gram can be keyed-in, or can exist in read/only memory
;  beyond the address space of the cp/m version you are
;  running). the cold start loader brings the cp/m system
;  into memory at "loadp" (3400h + "bias"). in a 20k
;  memory system, the value of "bias" is 0000h, with large
;  values for increased memory sizes (see section 2). afte
;  loading the cp/m system, the clod start loader branches
;  to the "boot" entry point of the bios, which begins at
;  "bios" + "bias." the cold start loader is not used un-
;  til the system is powered up again, as long as the bios
;  is not overwritten. tne origin is assumed at 0000h, an
;  must be changed if the controller brings the cold start
;  loader into another area, or if a read/only memory area
;  is used.

0000                org   0                 ; base of ram in cp/m
0014 =      msize   equ   20                ; min mem size in kbytes

0000 =      bias    equ   (msize-20)*1024   ; offset from 20k system
3400 =      ccp     equ   3400h+bias        ; base of the ccp
4a00 =      bios    equ   ccp+1600h         ; base of the bios
0300 =      biosl   equ   0300h             ; length of the bios
4a00 =      boot    equ   bios
1900 =      size    equ   bios+biosl-ccp    ; size of cp/m system
0032 =      sects   equ   size/128          ; # of sectors to load

            ;       begin the load operation

            cold:
0000 010200         lxi   b, 2              ; b=0, c=sector 2
0003 1632           mvi   d, sects          ; d=# sectors to load
0005 210034         lxi   h, ccp            ; base transfer address
```

```
                lsect:  ; load the next sector

                ;       insert inline code at this point to
                ;       read one 128 byte sector from the
                ;       track given in register b, sector
                ;       given in register c,
                ;       into the address given by <hl>
                ;
                ; branch to location "cold" if a read error occurs
                ;       ********************************************
                ;       *                                          *
                ;       *    user supplied read operation goes here.....
                ;       *                                          *
                ;       ********************************************

0008 c36b00             jmp   past$patch        ; remove this when patche
000b                    ds    60h

                past$patch:
                ; go to next sector if load is incomplete
006b 15                 dcr   d                 ; sects = sects-1
006c ca004a             jz    boot              ; head for the bios

                ;       more sectors to load
                ;
                ; we aren't using a stack, so use <sp> as scratch registe
                ;       to hold the load address increment

006f 318000             lxi   sp, 128           ; 128 bytes per sector
0072 39                 dad   sp                ; <hl> = <hl> + 128

0073 0c                 inr   c                 ; sector = sector + 1
0074 79                 mov   a, c
0075 fe1b               cpi   27                ; last sector of track?
0077 da0800             jc    lsect             ; no, go read another

                ; end of track, increment to next track
```

```
007a 0e01       mvi   c, 1          ; sector = 1
007c 04         inr   b             ; track = track + 1
007d c30800     jmp   lsect         ; for another group
0080            end                 ; of boot loader
```

附录 F　CP/M 磁盘定义库

```
1:  ;       CP/M 2.0 disk re-definition library
2:  ;
3:  ;       Copyright (c) 1979
4:  ;       Digital Research
5:  ;       Box 579
6:  ;       pacific Grove, CA
7:  ;       93950
8:  ;
9:  ;       CP/M logical disk drives are defined using the
10: ;       macros given below, where the sequence of calls
11: ;       is:
12: ;
13: ;       disks n
14: ;       diskdef   parameter-list-0
15: ;       diskdef   parameter-list-1
16: ;       ...
17: ;       diskdef   parameter-list-n
18: ;       endef
19: ;
20: ;       where n is the number of logical disk drives attached
21: ;       to the CP/M system, and parameter-list-i defines the
22: ;       characteristics of the ith drive (i=0, 1,...,n-1)
23: ;
24: ;       each parameter-list-i takes the form
25: ;                 dn, fsc, lsc, [skf], bls, dks, dir, cks, ofs, [0]
26: ;       where
27: ;       dn        is the disk number 0,1,...,n-1
28: ;       fsc       is the first sector number (usually 0 or 1)
29: ;       lsc       is the last sector number on a track
30: ;       skf       is optional "skew factor" for sector translate
31: ;       bls       is the data block size (1024, 2048,...,16384)
32: ;       dks       is the disk size in bls increments (word)
```

```
33:  ;          dir       is the number of directory elements (word)
34:  ;          cks       is the number of dir elements to checksum
35:  ;          ofs       is the number of tracks to skip (word)
36:  ;          [0]       is an optional 0 which forces 16K/directory en
37:  ;
38:  ;          for convenience, the form
39:  ;                    dn, dm
40:             defines disk dn as having the same characteristics as
41:  ;          a previously defined disk dm.
42:  ;
43:  ;          a standard four drive CP/M system is defined by
44:  ;          disks 4
45:  ;          diskdef 0,1,26,6,1024,243,64,64,2
46:  ;          dsk       set     0
47:  ;                    rept    3
48:  ;          dsk       set     dsk+1
49:  ;                    diskdef %dsk, 0
50:  ;                    endm
51:  ;                    endef
52:  ;
53:  ;          the value of "begdat" at the end of assembly defines t
54:  ;          beginning of the uninitialize ram area above the bios,
55:  ;          while the value of "enddat" defines the next location
56:  ;          following the end of the data area. the size of this
57:  ;          area is given by the value of "datsiz" at the end of t
58:  ;          assembly. note that the allocation vector will be qui
59:  ;          large if a large disk size is defined with a small blo
60:  ;          size.
61:  ;
62:  dskhdr    macro     dn
63:  ; ;        define a single disk header list
64:  dpe&dn :   dw        xlt&dn, 0000h       ; translate table
65:             dw        0000h, 0000h        ; scratch area
66:             dw        dirbuf, dpb&dn      ; dir buff, parm block
67:             dw        csv&dn, alv&dn      ; check, alloc vectors
68:             endm
69:  ;
70:  disks     macro     nd
71:  ; ;        define nd disks
```

```
ndisks  set     nd              ; ; for later reference
dpbase  equ     $               ; base of disk parameter blocks
; ;     generate the nd elements
dsknxt  set     0
        rept    nd
        dskhdr  %dsknxt
dsknxt  set     dsknxc+1
        endm
        endm
;
dpbhdr  macro   dn
dpb&dn  equ     $                       ; disk parm block
        endm
;
ddb     macro   data, comment
; ;     define a db statement
        db      data                    comment
        endm
;
ddw     macro   data, comment
; ;     define a dw statement
        dw      data            comment
        endm
;
gcd     macro   m, n
; ;     greatest common divisor of m, n
; ;     produces value gcdn as result
; ;     (used in sector translate table generation)
gcdm    set     m               ; ; variable for m
gcdn    set     n               ; ; variable for n
gcdr    set     0               ; ; variable for r
        rept    65535
gcdx    set     gcdm/gcdn
gcdr    set     gcdm-gcdx*gcdn
        if      gcdr=0
        exitm
        endif
gcdm    set     gcdn
gcdn    set     gcdr
```

```
111:            endm
112:            endm
113:  ;
114:  diskdef   macro   dn, fsc, lsc, skf, bls, dks, dir, cks, ofs, k16
115:  ; ;       generate the set statements for later tables
116:            if      nul lsc
117:  ; ;       current disk dn same as previous fsc
118:  dpb&dn    equ     dpb&fsc       ; equivalent parameters
119:  als&dn    equ     als&fsc       ; same allocation vector size
120:  css&dn    equ     css&fsc       ; same checksum vector size
121:  xlt&dn    equ     xlt&fsc       ; same translate table
122:            else
123:  secmax    set     lsc-(fsc)           ; ; sectors 0...secmax
124:  sectors   set     secmax+1      ; ; number of sectors
125:  als&dn    set     (dks)/8       ; ; size of allocation vector
126:            if      ((dks) mod o) ne 0
127:  als&dn    set     als&dn+1
128:            endif
129:  css&dn    set     (cks)/4       ; ; number of checksum elements
130:  ; ;       generate the block shift value
131:  blkval    set     bls/128       ; ; number of sectors/block
132:  blkshf    set     0             ; ; counts right 0's in blkval
133:  blkmsk    set     0             ; ; rills with 1's from right
134:            rept    16            ; ; once for each bit position
135:            if      blkval=1
136:            exitm
137:            endif
138:  ; ;       otherwise, high order 1 not found yet
139:  blkshf    set     blkshf+1
140:  blkmsk    set     (blkmsk shl 1) or 1
141:  blkval    set     blkval/2
142:            endm
143:  ; ;       generate the extent mask byte
144:  blkval    set     bls/1024            ; ; number of kilobytes/block
145:  extmsk    set     0             ; ; fill from right with 1's
146:            rept    16
147:            if      blkval=1
148:            exitm
149:            endif
```

```
150:  ; ;       otherwise more to shift
151:  extmsk    set     (extmsk shl 1) or 1
152:  blkval    set     blkval/2
153:            endm
154:  ; ;       may be double byte allocation
155:            if      (dks)>256
156:  extmsk    set     (extmsk shr 1)
157:            endif
158:  ; ;       may be optional [0] in last position
159:            if      not nul k16
160:  extmsk    set     k16
161:            endif
162:  ; ;       now generate directory reservation bit vector
163:  dirrem    set     dir             ; ; # remaining to process
164:  dirbks    set     bls/32          ; ; number of entries per block
165:  dirblk    set     0               ; ; fill with 1's on each loop
166:            rept    16
167:            if      dirrem = 0
168:            exitm
169:            endif
170:  ; ;       not complete, iterate once again
171:  ; ;       shift right and add 1 high order bit
172:  dirblk    set     (dirblk shr 1) or 8000h
173:            if      dirrem>dirbks
174:  dirrem    set     dirrem-dirbks
175:            else
176:  dirrem    set.    0
177:            endif
178:            endm
179:            dpbhdr  dn              ; ; generate equ$
180:            ddw     %sectors, <; sec per track>
181:            ddb     %blkshf, <; block shift>
182:            ddb     %blkmsk, <; block mask>
183:            ddb     %extmsk, <; extnt mask>
184:            ddw     %(dks)-1, <; disk size-1>
185:            ddw     %(dir)-1, <; oirectory max>
186:            ddb     %dirblk shr 8, <; alloc0>
187:            ddb     %dirblk and 0ffh, <; alloc1>
188:            ddw     %(cks)/4, <; check size>
```

```
189:            ddw     %ofs, <; offset>
190:  ; ;       generate the translate table, if requested
191:            if      nul skf
192:  xlt&dn    equ     0               ; no xlate table
193:            else
194:            if      skf=0
195:  xlt&dn    equ     0               ; no xlate table
196:            else
197:  ; ;       generate the translate table
198:  nxtsec    set     0               ; ; next sector to fill
199:  nxtbas    set     0               ; ; moves by one on overflow
200:            gcd     %sectors, skf
201:  ; ;       gcdn=gcd (sectors, skew)
202:  neltst    set     sectors/gcdn
203:  ; ;       neltst is number of elements to generate
204:  ; ;       before we overlap previous elements
205:  nelts     set     neltst          ; ; counter
206:  xlt&dn    equ     $                       ; translate table
207:            rept    sectors         ; ; once for each sector
208:            if      sectors<256
209:            ddb     %nxtsec+(fsc)
210:            else
211:            ddw     %nxtsec+(fsc)
212:            endif
213:  nxtsec    set     nxtsec+(skf)
214:            if      nxtsec>=sectors
215:  nxtsec    set     nxtsec-sectors
216:            endif
217:  nelts     set     nelts-1
218:            if      nelts=0
219:  nxtbas    set     nxtbas+1
220:  nxtsec    set     nxtbas
221:  nelts     set     neltst
222:            endif
223:            endm
224:            endif                   ; ; end of nul fac test
225:            endif                   ; ; end of nul bls test
226:            endm
227:
```

```
defds   macro   lab, space
lab:    ds      space
        endm
;
lds     macro   lb, dn, val
        defds   lb&dn, %val&dn
        endm
;
endef   macro
;;      generate the necessary ram data areas
begdat  equ     $
dirbuf: ds      128             ; directory access buffer
dsknxt  set     0
        rept    ndisks          ;; once for eacn disk
        lds     alv, %dsknxt, als
        lds     csv, %dsknxt, css
dsknxt  set     dsknxt+1
        endm
enddat  equ     $
datsiz  equ     $-begdat
;;      db 0 at this point forces hex record
        endm
```

附录G　组块和解块算法

```
;*****************************************************
;*                                                   *
;*      Sector Deblocking Algorithms for CP/M 2.0    *
;*                                                   *
;*****************************************************
;
;       utility macro to compute sector mask
smask   macro   hblk
;;      compute log2(hblk), return @x as result
;;      (2** @x = hblk on return)
@y      set     hblk
@x      set     0
;;      count right shifts of @y until = 1
        rept    8
```

```
15:             if      @y = 1
16:             exitm
17:             endif
18:     ; ;     @y is not 1, shift right one position
19:     @y      @et     @y shr 1
20:     @x      @et     @x + 1
21:             endm
22:             endm
23:     ;
24:     ;  ***************************************************
25:     ;  *                                                 *
26:     ;  *          CP/M to host disk constants            *
27:     ;  *                                                 *
28:     ;  ***************************************************
29:     blksiz  equ     2048            ; CP/M allocation size
30:     hstsiz  equ     512             ; host disk sector size
31:     hstspt  equ     20              ; host disk sectors/trk
32:     hstblk  equ     hstsiz/128      ; CP/M sects/host buff
33:     cpmspt  equ     hstblk* hstspt  ; CP/M sectors/track
34:     secmsk  equ     hstblk-1        ; sector mask
35:             smask   hstblk          ; compute sector mask
36:     secshf  equ     @x              ; log2 (hstblk)
37:     ;
38:     ;  ***************************************************
39:     ;  *                                                 *
40:     ;  *       BDOS constants on entry to write          *
41:     ;  *                                                 *
42:     ;  ***************************************************
43:     wrall   equ     0               ; write to allocated
44:     wrdir   equ     1               ; write to directory
45:     wrual   equ     2               ; write to unallocated
46:     ;
47:     ;  ***************************************************
48:     ;  *                                                 *
49:     ;  *  The BDOS entry points given below show the     *
50:     ;  *  code which is relevant to deblocking only.     *
51:     ;  *                                                 *
52:     ;  ***************************************************
53:     ;
```

```
54:  ;          DISKDEF macro, or hand coded tables go here
55:  dpbase     equ     $               ; disk param block base
56:  ;
57:  boot:
58:  wboot:
59:              ; enter here on system boot to initialize
60:              xra     a               ; 0 to accumulator
61:              sta     hstact          ; host buffer inactive
62:              sta     unacnt          ; clear unalloc count
63:              ret
64:  ;
65:  seldsk:
66:              ; select disk
67:              mov     a, c            ; selected disk number
68:              sta     sekdsk          ; seek disk number
69:              mov     l, a            ; disk number to HL
70:              mvi     h, 0
71:              rept    4               ; multiply by 16
72:              dad     h
73:              endm
74:              lxi     d, dpbase       ; base of parm block
75:              dad     d               ; hl = .dpb (curdsk)
76:              ret
77:  ;
78:  settrk:
79:              ; set track given by registers BC
80:              mov     h, b
81:              mov     l, c
82:              shld    sektrk          ; track to seek
83:              ret
84:  ;
85:  setsec:
86:              ; set sector given by register c
87:              mov     a, c
88:              sta     seksec          ; sector to seek
89:              ret
90:  ;
91:  setdma:
92:              ; set dma address given by BC
```

```
93:             mov     h, b
94:             mov     l, c
95:             shld    dmaadr
96:             ret
97:     ;
98:     sectran:
99:             ; translate sector number BC
100:            mov     h, b
101:            mov     l, c
102:            ret
103:    ;
104:    ; ********************************************************
105:    ; *                                                      *
106:    ; *     The READ entry point takes the place of          *
107:    ; *     the previous BIOS defintion for READ.            *
108:    ; *                                                      *
109:    ; ********************************************************
110:    read:
111:            ; read the selected CP/M sector
112:            mvi     a, 1
113:            sta     readop          ; read operation
114:            sta     rsflag          ; must read data
115:            mvi     a, wrual
116:            sta     wrtype          ; treat as unalloc
117:            jmp     rwoper          ; to perform the read
118:    ;
119:    ; ********************************************************
120:    ; *                                                      *
121:    ; *     The WRITE entry point takes the place of         *
122:    ; *     the previous BIOS defintion for WRITE.           *
123:    ; *                                                      *
124:    ; ********************************************************
125:    write:
126:            ; write the selected CP/M sector
127:            xra     a               ; 0 to accumulator
128:            sta     readop          ; not a read operation
129:            mov     a, c            ; write type in c
130:            sta     wrtype
131:            cpi     wrual           ; write unallocated?
```

```
132:            jnz     chkuna          ; check for unalloc
133:    ;
134:    ;       write to unallocated, set parameters
135:            mvi     a, blksiz/128   ; next unalloc recs
136:            sta     unacnt
137:            lda     sekdsk          ; disk to seek
138:            sta     unadsk          ; unadsk = sekdsk
139:            lhld    sektrk
140:            shld    unatrk          ; unatrk = sectrk
141:            lda     seksec
142:            sta unasec              ; unasec = seksec
143:    ;
144:    chkuna:
145:            ; check for write to unallocated sector
146:            lda     unacnt          ; any unalloc remain?
147:            ora     a
148:            jz      alloc           ; skip if not
149:    ;
150:    ;       more unallocated records remain
151:            dcr     a               ; unacnt = unacnt-1
152:            sta     unacnt
153:            lda     sekdsk          ; same disk?
154:            lxi     h, unadsk
155:            cmp     m               ; sekdsk = unadsk?
156:            jnz     alloc           ; skip if not
157:    ;
158:    ;       disks are the same
159:            lxi     h, unatrk
160:            call    sektrkcmp       ; sektrk = unatrk?
161:            jnz     alloc           ; skip if not
162:    ;
163:    ;       tracks are the same
164:            lda     seksec          ; same sector?
165:            lxi     h, unasec
166:            cmp     m               ; seksec = unasec?
167:            jnz     alloc           ; skip if not
168:    ;
169:    ;       match, move to next sector for future ref
170:            inr     m               ; unasec = unasec + 1
```

```
171:            mov     a, m            ; end of track?
172:            cpi     cpmspt          ; count CP/M sectors
173:            jc      noovf           ; skip if no overflow
174:    ;
175:    ;       overflow to next track
176:            mvi     m, 0            ; unasec = 0
177:            lhld    unatrk
178:            inx     h
179:            shld    unatrk          ; unatrk = unatrk + 1
180:    ;
181:    noovf:
182:            ; match found, mark as unnecessary read
183:            xra     a               ; 0 to accumulator
184:            sta     rsflag          ; rsflag = 0
185:            jmp     rwoper          ; to perform the write
186:    ;
187:    alloc:
188:            ; not an unallocated record, requires pre-read
189:            xra     a               ; 0 to accum
190:            sta     unacnt          ; unacnt = 0
191:            inr     a               ; 1 to accum
192:            sta     rsflag          ; rsflag = 1
193:    ;
194:    ; ***************************************************
195:    ; *                                                 *
196:    ; *   Common code for READ and WRITE follows        *
197:    ; *                                                 *
198:    ; ***************************************************
199:    rwoper:
200:            ; enter here to perform the read/write
201:            xra     a               ; zero to accum
202:            sta     erflag          ; no errors (yet)
203:            lda     seksec          ; compute host sector
204:            rept    secshf
205:            ora     a               ; carry = 0
206:            rar                     ; shift right
207:            endm
208:            sta     sekhst          ; host sector to seek
209:    ;
```

```
;       active host sector?
        lxi     h, hstact       ; host active flag
        mov     a, m
        mvi     m, 1            ; always becomes 1
        ora     a               ; was it already?
        jz      filhst          ; fill host if not
;
;       host buffer active, same as seek buffer?
        lda     sekdsk
        lxi     h, hstdsk       ; same disk?
        cmp     m               ; sekdsk = hstdsk?
        jnz     nomatch
;
;       same disk, same track?
        lxi     h, hsttrk
        call    sektrkcmp       ; sektrk = hsttrk?
        jnz     nomatch
;
;       same disk, same track, same buffer?
        lda     sekhst
        lxi     h, hstsec       ; sekhst = hstsec?
        cmp     m
        jz      match           ; skip if match
;
nomatch:
        ; proper disk, but not correct sector
        lda     hstwrt          ; host written?
        ora     a
        cnz     writehst        ; clear host buff
;
filhst:
        ; may have to fill the host buffer
        lda     sekdsk
        sta     hstdsk
        lhld    sektrk
        shld    hsttrk
        lda     sekhst
        sta     hstsec
        lda     rsflag          ; need to read?
```

```
249:            ora     a
250:            cnz     readhst         ; yes, if 1
251:            xra     a               ; 0 to accum
252:            sta     hstwrt          ; no pending write
253:    ;
254:    match:
255:            ; copy data to or from buffer
256:            lda     seksec          ; mask buffer number
257:            ani     secmsk          ; least signif bits
258:            mov     l, a            ; ready to shift
259:            mvi     h, 0            ; double count
260:            rept    7               ; shift left 7
261:            dad     h
262:            endm
263:    ;       hl has relative host buffer address
264:            lxi     d, hstbuf
265:            dad     d               ; hl=host address
266:            xchg                    ; now in DE
267:            lhld    dmaadr          ; get/put CP/M data
268:            mvi     c, 128          ; length of move
269:            lda     readop          ; which way?
270:            ora     a
271:            jnz     rwmove          ; skip if read
272:    ;
273:    ;       write operation, mark and switch direction
274:            mvi     a, 1
275:            sta     hstwrt          ; hstwrt=1
276:            xchg                    ; source/dest swap
277:    ;
278:    rwmove:
279:            ; C initially 128, DE is source, HL is dest
280:            ldax    d               ; source character
281:            inx     d
282:            mov     m, a            ; to dest
283:            inx     h
284:            dcr     c               ; loop 128 times
285:            jnz     rwmove
286:    ;
287:    ;       data has been moved to/from host buffer
```

```
288:            lda     wrtype          ; write type
289:            cpi     wrdir           ; to directory?
290:            lda     erflag          ; in case of errors
291:            rnz                     ; no further processing
292:    ;
293:    ;       clear host buffer for directory write
294:            ora     a               ; errors?
295:            rnz                     ; skip if so
296:            xra     a               ; 0 to accum
297:            sta     hstwrt          ; buffer written
298:            call    writehst
299:            lda     erflag
300:            ret
301:    ;
302:    ;  ********************************************************
303:    ;  *                                                      *
304:    ;  *      Utility subroutine for 16-bit compare           *
305:    ;  *                                                      *
306:    ;  ********************************************************
307:    sektrkcmp:
308:            ; HL = .unatrk or .hsttrk, compare with sektrk
309:            xchg
310:            lxi     h, sektrk
311:            ldax    d               ; low byte compare
312:            cmp     m               ; same?
313:            rnz                     ; return if not
314:    ;       low bytes equal, test high ls
315:            inx     d
316:            inx     h
317:            ldax    d
318:            cmp     m               ; sets flags
319:            ret
320:    ;
```

```
;  ***********************************************
;  *                                             *
;  *  WRITEHST performs the physical write to    *
;  *  the host disk, READHST reads the physical  *
;  *  disk.                                      *
;  *                                             *
;  ***********************************************
writehst:
        ; hstdsk = host disk #, hsttrk = host track #,
        ; hstsec = host sect #. write "hstsiz" bytes
        ; from hstbuf and return error flag in erflag.
        ; return erflag non-zero if error
        ret
;
readhst:
        ; hstdsk = host disk #, hsttrk = host track #,
        ; hstsec = host sect #. read "hstsiz" bytes
        ; into hstbuf and return error flag in erflag.
        ret
;
;  ***********************************************
;  *                                             *
;  *  Unitialized RAM data areas                 *
;  *                                             *
;  ***********************************************
;
sekdsk:  ds      1          ; seek disk number
sektrk:  ds      2          ; seek track number
seksec:  ds      1          ; seek sector number
;
hstdsk:  ds      1          ; host disk number
hsttrk:  ds      2          ; host track number
hstsec:  ds      1          ; host sector number
;
sekhst:  ds      1          ; seek shr secshf
hstact:  ds      1          ; host active flag
hstwrt:  ds      1          ; host written flag
;
unacnt:  ds      1          ; unalloc rec cnt
```

```
unadsk:  ds      1              ; last unalloc disk
unatrk:  ds      2              ; last unalloc track
unasec:  ds      1              ; last unalloc sector
;
erflag:  ds      1              ; error reporting
rsflag:  ds      1              ; read sector flag
readop:  ds      1              ; 1 if read operation
wrtype:  ds      1              ; write operation type
dmaadr:  ds      2              ; last dma address
hstbuf:  ds      hstsiz         ; host buffer
;
;  ***************************************************
;  *                                                 *
;  *   The ENDEF macro invocation goes here          *
;  *                                                 *
;  ***************************************************
;
     end
```

第八章　CP/M2.2对Heath/Zenith 8位计算机系统的实现

第一节　系统启动

1.1　从主驱动器上冷启动

H/Z89允许用户从5.25″盘及8″盘上启动，也可以在Winchester硬盘进行分区操作之后进行冷启动。启动可以由系统内的主驱动器来进行，而主驱动器是由H/Z89内CPU板上的SW-501开关决定的。

典型的H/Z89主驱动器通常置成机器本身所带的驱动器。

步骤：

（1）接通电源（包括主机及驱动器）。

（2）将标有“CP/M Distribution Disk”的软盘插入主驱动器中。此盘带有CP/M操作系统。

（3）关上驱动器门。在终端屏幕预热后将显示“H:”提示符。

（4）键入B（或小写b）。

（5）按RETURN键。这时磁盘驱动器应该被激活，并以小红灯亮作为提示。这时计算机在读入CP/M操作系统。

经过短时间后，计算机显示类似于以下的信息：

32K　Heath/Zenith CP/M 2.2. 03

FOR　{type} DISKS WITH OPTION(S) {code}

这里“03”为软件的当前版本号，“32K”为CP/M当前配置的内存容量，“type”为BIOS所配置的磁盘类型，“Code”显示BIOS中当前选择的任选项。

到目前为止，用户已成功地启动了CP/M。

如果没有提示类似以上信息，可做以下两步：

（1）打SHIFT键。

（2）同时按RESET键。这时计算机重新提示“H:”，然后用户可重复前面各步骤。

1.2　从附属驱动器上冷启动

如果用户的H/Z89计算机连有多个驱动器，则可在没有指定为主驱动器的设备（即附属设备）上进行启动。

步骤：

（1）按B。

（2）再按S。

这时在屏幕左上角显示“H:Boot SD”，指示用户可在附属驱动器上进行启动。

（3）按RETURN键。

以后的启动过程同前面所述的主驱动器启动相同。

1.3 标准的系统配置

在系统启动之后，可以运行实用程序CONFIGUR来变化或修改CP/M操作系统的配置，以匹配特定的硬件环境。当它开始运行时，可显示类似于下面的信息（有关CONFIGUR程序后面还要介绍）：

```
32K HEATH/ZENITH CP/M 2.2.03
FOR Z17 DISKS WITH OPTION(S)  I

HEATH/ZENITH CONFIGURATION PROGRAM
VERSION 2.2.03
SERIAL NUMBER: nnn-nnnnn

THIS PROGRAM CONFIGURES THE CP/M OPERATING SYSTEM TO A
PARTICULAR HARDWARE ENVIROMENT.

PLEASE WAIT DURING HARDWARE VERIFICATION…

H/Z89 WITH 64K OF RANDOM ACCESS MEMORY (RAM)
03 Z17 MINIFLOPY DRIVE(S)
CRT BAUD RATE IS 9600
03 ADDITIONAL SERIAL PORTS FOUND

DRIVE A DISK IS WRITE PROTECTED.
MODIFICATIONS WILL NOT BE MADE TO THE DISK FOR THIS CONF-
  IGUR RUN.
STANDARD SYSTEM (Y OR N) ? (Y) :
```

用户可以回答Y，或直接按RETURN键（缺省为Y）。CP/M然后显示命令方式提示符A>。当用户要修改系统配置时可回答N，对系统的配置进行修改（详见CONFIGUR的介绍部分）。

1.4 备用盘的建立

用户在得到一系统盘之后，应立即拷贝一张备用盘，这是非常重要的。以下介绍如何建立备用盘。

在进行操作之前，首先要决定使用哪种操作环境（系统包括哪些外设，是如何配置的），其次要根据操作环境，拟定一系列操作过程。

（1）确定驱动器名

在拷贝备用盘之前，首要的是决定驱动器名的赋值（除非系统只配单个驱动器，这时驱动器缺省为A:）

以下是可作为启动设备的驱动器名：

H/Z17 A:, B:, C:　　Z37 A:, B:, C:

H/Z47 A:, B:　　Z67 A:, B:, C:

注：系统同时只可支持一个主设备和一个附属设备共两种设备类型。下面列出一种设备作为主设备时，其他三种可作为替换的附属设备：

1) H/Z17作为主设备

H/Z17 A:, B:, C:（启动设备）

Z37　D:, E:, F:

H/Z47 D:, E:

Z67　D:, E:, F:

2) Z37作为主设备

Z37　A:, B:, C:（启动设备）

H/Z17 D:, E:, F:

H/Z47 D:, E:

Z67　D:, E:, F:

3) H/Z47作为主设备

H/Z47 A:, B:（启动设备）

H/Z17 C:, D:, E:

Z37　C:, D:, E:

Z67　C:, D:, E:

4) Z67作为主设备

Z67　A:, B:, C:（启动设备）

H/Z17 D:, E:, F:

Z37　D:, E:, F:

H/Z47 D:, E:

（2）建立备用拷贝盘

首先按前面所述方法进行冷启动，然后做下列各步：

1) 用FORMAT程序，对空磁盘进行初始化。

2) 用SYSGEN程序，对已初始化的磁盘作系统生成。

3) 用PIP程序，将源盘上的各个程序拷贝到已作完系统生成的盘上。

这样就可以建立备用盘。也可以在空盘初始化后，直接用DUP程序同时完成SYSGEN和PIP两个程序的功能。但是，使用DUP程序，要求目的盘和源盘具有相同的介质（例如都是5英寸软分段盘或者都是5英寸硬分段盘）。

关于FORMAT、SYSGEN、PIP及DUP程序，请参阅第二节。

1.5 系统优化

系统优化是确定系统包含哪些设备的过程，以便CP/M能充分利用用户提供的资源。这可用两个主要的程序MOVCPMnn和CONFIGUR来完成。

MOVCPM使CP/M准确地知道用户有多大内存容量，CONFIGUR使CP/M知道用户连到计算机上有哪些设备。用户将这些设备的详细信息告诉CP/M，这样，当用户需要

使用这些设备时，CP/M 可以调用它们。

关于这两个程序，也在第二节中详细介绍。

第二节　实用程序

2.1 FORMAT 程序

FORMAT 程序可用来对空盘作初始化（或分区），也可以将已记录有信息的盘重新作初始化（或分区）。该实用程序还用于检查磁盘介质，以探测记录表面的缺陷。FORMAT 程序在磁盘表面建立磁映象，以便操作系统可在存有信息的盘中区分出不同的存储单元（即磁道和扇区）。

由于FORMAT 程序执行后将删除所有磁道，包括任何文件，故用户使用该程序进行磁盘格式化之前，一定要保证磁盘上无有用的信息。

新盘使用之前必须进行格式化。在 Z37 上对软分段盘进行格式化要指定磁盘使用面数和记录密度。密度决定了数据记录在磁盘上的疏密程度。所谓单密度是指在磁盘上按正常密度安排数据位；而双密度是使正常数据位的密集程度加倍，以记录近两倍于单密度的数据。这就要求磁盘精度更高。

FORMAT 程序采用应答方式，其执行步骤在下面详述。回答驱动器的选择，只能用字母 A:～F:，且要保证系统中存在所选的设备。将没有写保护（用户要进行格式化的）磁盘插入所选驱动器中，键入 X（驱动器名），按 RETURN 键，开始格式化操作。操作完成后，可选择继续对另一磁盘作格式化，或返回 CP/M 命令状态。步骤简述如下：

（1）键入FORMAT，按 RETURN 键：

A>FORMAT

（2）程序询问是否进行格式化：

Is that what you want (y/n) :

（3）回答 Y，表示要进行格式化。

（4）程序接着询问使用哪个驱动器：

Which drive do you wish to use for this operation?

（5）回答包含目的盘的驱动器名 X（这里假定为 B）。如果被格式化的是硬分段盘，则跳过下列两步。

（6）程序询问磁盘密度：

Which Density (S=Single, D=Double)

（7）询问用户盘使用单面还是双面：

Number of sides (1=Single, 2=Double)

（8）提示用户将要格式化的盘插入所选驱动器中：

Put the disk you wish to be formatted in drive X

（9）将要格式化的盘插入驱动器 X（这里为 B）中。

（10）按 RETURN 键。

（11）功能完成后，程序询问是否还要对别的盘做格式化：

Do you have more disks to format (y/n) :

（12）可根据需要回答，假定回答 n 。

（13）程序提示将可启动的盘（即带有系统的盘）放入 A 驱动器中，打入任一字符，返回 CP/M 命令状态：

Place a bootable disk in drive A and press any character:

（14）按 RETURN 键，返回 CP/M。

到此为止，用户完成了磁盘格式化工作。

2.2 DUP 程序

DUP 为 Heath/zenith 磁盘实用程序，它可以方便地进行磁盘的转储和验证两个磁盘是否相同。使用 DUP 程序必须在两个5.25″驱动器间或两个 8″驱动器间进行。

注：如果需要在两个不同介质的磁盘间传递数据，例如在 5.25″ 软分段磁盘驱动器与 5.25″ 硬分段磁盘驱动器间，要使用 PIP 程序。

在系统启动之后，用户就可使用 DUP 程序将磁盘上所有内容复制到另一张磁盘上。但两张磁盘必须有相同的尺寸及相同的密度。下面是使用 DUP 程序的例子：

A>DUP

Disk Utility Program

Version 2.nn

系统显示出一系列任选项，其中包括拷贝并校验、只进行拷贝以及只进行校验。用户可通过选择 Z 任选项退回到操作系统中。

在作 DUP 之前，目的盘如果是5.25″或 8″软分段盘，则必须先格式化，如果是 5.25″硬分段盘则不需要。由于目的盘做完 DUP 后，以前的信息均被破坏，故在 DUP 前要保证目的盘上无有用信息。

下面举例说明 DUP 操作过程，假定选择“拷贝并校验”：

（1）Source Unit: 回答 A（被拷贝盘驱动器名）

（2）Destination Unit: 回答 B（目的盘驱动器名）

（3）将源盘放入 A

（4）将目的盘放入 B

（5）按 RETURN 键开始复制操作（在拷贝和校验完成后均有提示信息）

进行 DUP 操作，源盘和目的盘必须在两个不同驱动器中，且必须具有相同的尺寸及密度。

在将源盘及目的盘插入相应的驱动器以后，用户如果不需要再进行复制操作，则可以按除RETURN 以外的任何一个键。如选择“拷贝并校验”任选，则在拷贝完成后自动进行校验。

校验功能只是检查源盘与目的盘在同一存储位置上（扇区）是否具有相同的信息。由于校验功能不能使用和认识 CP/M 文件，所以当两张磁盘具有相同文件但以不同的次序存放（即存放在不同的扇区）时，校验功能认为这两张磁盘不同。

选择的操作完成后，DUP 程序将提示进行下一次选择：

Do you want to:

A Copy and verify

B Copy only

C Verify only

Z exit to operating system

Selection:

这时可根据用户要求再次进行选择。

2.3 SETLP 程序

SETLP 程序使用户可以动态地改变行打印机接口和波特率。波特率是在 CONFIGUR 中定义的。这个程序使用户可以换用具有不同波特率的打印机，而不必用 CONFIGUR重新配置。例如，在Diablo和H14间进行切换。SETLP 程序只是临时改变内存中的值，但不影响存在盘上的波特率信息。

SETLP 需要一个变量，即新的波特率，可以取下列值中之一：

75	150	1200	9600
110	300	2400	19200
134.5	600	4800	38400

例如：

A>SETLP 1200 RETURN

A>

这个命令将行打印机接口波特率置成1200。

在通常 Heath/Zenith 硬件配置中，Diablo 打印机（WH44等）波特率置成1200，点阵式打印机（H/Z25等）置成4800，而 H36及 WH34 通常置成300。对于其他打印机，请参考随打印机的有关手册。

2.4 LIST 程序

LIST 命令可用来在与系统相连的打印机上列出一个或多个文件。通常，所要列出的文件是由汇编产生的 *.PRN 文件、BASIC 源程序或由编辑产生的简单的可读文件 LIST同样提供一些任选功能，包括分页、打印标题及标记扩展。

此命令形式类似于 PIP。LIST 命令可在系统命令状态A>时键入，可不带变量。在后一种情况下，LIST 程序提示 * 号。

使用此程序的命令格式为：

A>LIST 文件 1 {文件 2 {文件 3 …}} {参数}

或

A>LIST RETURN

*文件 1 {文件 2 {文件 3 …}} {参数}

括号{ }中的变量是任选的。但要注意，每个文件间要用空格分开。在 * 提示出现之后（假定使用无变量方式时）可按 RETURN 键或在按 CTRL 键的同时按 C 键来结束程序执行。

可用的参数列出如下：

[D xxxxxxxxxx]	表示在负号的左面直接打印10个字符（可作为日期）。如不选此参数，缺省为空格
[H text]	表示标题可打印60个字符，标题打印在每页的顶部。如不选此参数，缺省列出 LIST 文件名（即 LIST.COM）

[L nn]	置每页中用户程序的行数。缺省时为60，L = 0 时关闭分页功能
[N]	无标题行（蕴含没有页号）
[T nn]	每 nn 位置一个标记（tab），缺省值为 8
[P nn]	开始页号为 nn，缺省时为 1
[U]	强制输出大写字符
[C nn]	由命令产生的输出拷贝数，缺省时为 1
[E]	在列出文件后删除这些文件

采用提示方式时，所有选择参数(开始页号及列表后的文件删除例外)保持有效，直到退出程序为止。

注：以命令方式使用时，CCP 强迫所有标题及数据信息映象以大写形式输出。

在打印期间，LIST 输出可以在键盘上打入任何字符来使之停止。

LIST 程序打印输出的例子：

```
A>LIST MEMBERS.LST [H MEMBERSHIP LIST] [D 3/30/83]
```

或

```
A>LIST  RETURN
* MEMBERS.LST [H MEMBERSHIP LIST] [D 3/30/83)
```

将产生以下输出：

```
MEMBERSHIP LIST                          3/30/83 PAGE  1
        JONES, WILLIAM          1001  OAK STREET
                                BUSYTOWN, MI
                                02987
        MAULDIN, SHIRLEY        3344 MAPLE ROAD
                                WHOKNOWS, ID
                                26783
        SMITH, JOHN             276 ELMWOOD DRIVE
                                ANYWHERE, IL.
                                60988
```

2.5 PREL 程序

Heath/Zenith 提供的 PREL 程序取两个汇编的十六进制输出文件而产生一个可重定位文件。第一个汇编的十六进制输出文件应起始于 0000H，而第二个则定位在比其高出一页的位置（即0100H)上。第一个汇编的十六进制输出文件的扩展名应是 HX_0，而第二个则应是 HX_1。

PREL 需要两个变量：第一个为两个十六进制输出文件的基本文件名，第二个为输出页重定位文件的基本文件名。输出文件的扩展名为 PRE。以下是一个命令的例子：

```
A>PREL MYPROG TESTPROG
```

该命令从两个十六进制文件 MYPROG.HX_0及 MYPROG.HX_1 建立一个 TESTPROG.PRE文件。

PREL 程序比较两个十六进制文件并执行两个不同功能。一个是将十六进制文件转换成类似于 LOAD 命令所建立的二进制文件。另一个是在该文件末尾附加上一个重定位表。这个表对程序模块中每一字节设置一个位，如果某位是 1，则表示相应字节的值在程序装入时必须被重定位。如果给定字节的十六进制值只是 0 或 1，则产生错误信号。如果十六进制文件中的地址不按数字顺序，也会产生出错信息。

所有要重定位的地址表示成相对于模块基地址的相对地址，而没有被绝对定义。这样做的一个方便办法就是在源程序中不要“org”语句，使用 Submit 文件中的批命令来实际完成所需要的汇编及转换成重定位格式。下面是这类批命令的例子：

```
PIP TEMP0.ASM=ORG0.ASM, $1.ASM
ASM TEMP0.AAZ
REN $1.HX0=TEMP0.HEX
ERA TEMP0.ASM
PIP TEMP1.ASM=ORG1.ASM, $1.ASM
ASM TEMP1.AAZ
ERA TEMP1.ASM
REN $1.NX1=TEMP1.HEX
PREL $1 $1
ERA $1.HX0
ERA $1.HX1
```

这里文件ORG0.ASM 及 ORG1.ASM 只是分别包含“org 0000H”和“org 0100H”语句。用户可以通过打入下列命令来执行以上命令序列：

```
SUBMIT MAKEPRE MYPROG
```

这里要求上面的命令序列包含在名为 MAKEPRE.SUB 的文件中，原始的汇编文件为 MYPROG.ASM，而最后产生的页重定位文件为 MYPROG.PRE。

2.6 MAKEBIOS 程序

CP/M 基本 I/O 系统（BIOS）提供了对于各种联机外部设备（诸如磁盘驱动器、键盘、终端及打印机）输入/输出操作的子程序。由于 Heath/Zenith 计算机可配置各种不同类型的磁盘驱动器，因此，有必要对 BIOS 部分进行修改，使之在各种不同的硬件环境下有效地运行。MAKEBIOS 实用程序就是用于处理这一特殊要求的。

MAKBIOS 是批处理程序，用 SUBMIT 命令提交。它是自修改程序，要求特殊的磁盘系统组合，然后添加必要的子程序，去掉不必要的子程序。由于该程序本身的复杂性，也由于它是自修改程序，因此，用户在使用时必须格外小心。

（1）在运行 MAKEBIOS 前，必须建立完整的备份盘，并将原盘保存在安全的地方，在任何操作中，都只使用备份盘。

（2）MAKEBIOS 修改提供的软件系统盘上的源程序 BIOS.ASM，源码由主体和序言两部分组成，改变序言部分的顺序或者程序的初值都会使源程序全部报废，这也是必须建立备份盘的一个主要原因。

因为该程序是自修改的，程序认为某些值在某确定位置，如果代码的安排改变了，或者某个值改变了，而MAKEBIOS 并不知道，则还会对该程序进行修改。

（3）在将 MAKEBIOS 传送到另一张磁盘上时必须注意，MAKEBIOS 从磁盘上调用其他程序，如果该程序不在盘上，MAKEBIOS 可能不正确执行，甚至造成系统无休止运行，必须重新引导。因此，用户在拷贝 MAKEBIOS 时，应将全部有关文件都拷贝到新盘上。

MAKEBIOS 操作步骤如下：

（1）打入 SUBMIT MAKEBIOS 后跟存放新 BIOS 的驱动器名，命令格式如下：

A>SUBMIT {Y:}MAKEBIOS {X:{Filename}} {Y:}

其中“X:”是任选目的驱动器名，如果不指定，则假定为当前登记的磁盘（上例中是 A 盘）

注意：SUBMIT 程序必须放在 A 驱动器盘上。

“Filename”是任选的新 BIOS 文件名，指定新 BIOS 名可使原 BIOS 不被复盖。

“Y”是该操作所用到的 BIOS 所在的源驱动器名（MAKEBIOS.SUB 必须在该盘上）。

注：以 5 英寸软盘提供的 CP/M 系统，运行 MAKEBIOS 时，VOL.I 在 A:，VOL.Ⅲ在 B:，目的盘在 C:，用下列命令：

A>SUBMIT B:MAKEBIOS C: B:

新 BIOS 可以通过 PIP 传送（如果 BIOS 已使用 STAT 命令置成$SYS和$R/O，则带〔R〕任选项），命令格式如下：

A>STAT{X:}BIOS.SYS

屏幕上显示：

RECS	BYTES	EXT	ACC	
42	6K	1	R/O	A: (BIOS.SYS)
BYTES	REMAINING	ON	A:nnk	

“R/O”表明该文件置为只读，括号内文件名（BIOS.SYS）指明该文件置为系统文件（$SYS)

对 R/O 和 SYS 文件，使用 PIP 命令：

A>PIP B:=C:BIOS.SYS〔R〕

这里假定登记的磁盘是“A:”,PIP.COM 在 A 盘，并假定 BIOS.SYS 在“C:”,要把它拷贝到“B:”

在传送完之后，用 STAT 命令置 BIOS.SYS 为只读和系统属性：

A>STAT B:BIOS.SYS $R/O

A>STAT B:BIOS.SYS $SYS

这里，“B:”为非保护的 BIOS 所在驱动器名。

在建立新的 BIOS 之后，要使用这个新 BIOS，其步骤如下：

（1）用 MAKEBIOS 建立新 BIOS，如前所述。

（2）用 MOVCPMnn 构成新的操作系统，使之带有新建立的 BIOS 和该系统运行环境具有的内存大小。

（3）用 SYSGEN 将所构成的操作系统放进系统磁道。

（4）将新建立的 BIOS.SYS 的一份拷贝放进系统盘。

注意：在用 MAKEBIOS 建立新 BIOS 后，新 BIOS 并不在内存中，而是在目的盘上

（如果不指定目的驱动器，则在 A 盘上）。如果在用 MAKEBIOS 时指定了文件名作为新 BIOS 名，则必须将新 BIOS 改名为“BIOS.SYS”。

2.7 MOVCPMnn 程序

MOVCPM 实用程序调整操作系统以使用不同的内存容量。使用 MOVCPM，建立操作系统的内存映象，使之可以充分利用系统的内存容量。作完 MOVCPM 后，操作系统的映象被存放在系统知道的内存单元，然后可使用 SYSGEN 程序，将内存中的系统映象拷贝并写到磁盘的系统磁道上。

共有四种 MOVCPM 程序，可供用户选用。用户可以对特定的驱动器重新配置 CP/M 系统，以适应特定的内存大小。CP/M 的 Heath/Zenith 版本，可在 32K 至 64K 内存下运行。可用 MOVCPM 规定操作系统的大小，以匹配所需内存，并建立启动磁盘。nn 为启动时读磁盘的驱动控制器型号。

启动设备控制器指定：nn

MOVCPM17　对 H/Z17控制器（5.25″硬分段）

MOVCPM37　对 Z37 控制器（5.25″软分段）

MOVCPM47　对 H/Z47控制器（8″软盘）

MOVCPM67　对 Z67控制器（温式磁盘系统）

基本命令格式：

MOVCPMnn{内存尺寸}{BIOS 说明符}

这里{内存尺寸}以1024（即1K）的倍数来设置所需系统大小，{BIOS 说明符}指定用来运行的 BIOS 的存储位置。如果不指定这两个任选，则 MOVCPMnn 将自动地对硬件调查所找到的最大内存容量配置系统，并使用当前驻留在内存中的 BIOS，如果不指定{BIOS 说明符}任选，则 CP/M 使用当前装入内存的 BIOS。

注意，当用 MAKEBIOS 建立新的 BIOS 以后，使用 MOVCPMnn 必须指定上述任选参数，因为在运行 MAKEBIOS 后，新 BIOS 并没有放入内存。

上述两个参数可用的任选如下：

内存尺寸（任选）：

××　内存大小以 K 为单位的整数

*　进行硬件调查，以所找到的内存总容量来配置系统

空　同 * 一样处理

注：如后面有{BIOS 说明符}，则{内存尺寸}不能为空格，而要指定整数值或 * 号。

BIOS 说明符（任选）：

*　使用当前驻留在内存中的 BIOS

空　如不说明，则使用当前内存中的 BIOS

X:　使用驱动器“X”上的 BIOS.SYS

X:（文件名）　使用驱动器“X”上（文件名）作为 BIOS 源文件

注：如果指定{BIOS 说明符}参数，则也必须指定{内存尺寸}参数。

MOVCPMnn 选择的{内存尺寸}可少于内存实际容量，如果参数超出实际内存范围，则构成的系统不能启动。

2.8 SYSGEN 程序

系统生成程序 SYSGEN 可将操作系统写到磁盘指定的存储磁道上。这些磁道为系统磁道并且分布在磁盘记录表面的最外部分。磁盘操作系统包括一些附加的子程序，这些都是监控程序（在 ROM 中）所必需的。操作系统将这些子程序及 BIOS（也在磁盘上）放在内存相应的地方，以执行它们的所有功能。

使用 SYSGEN 程序可将操作系统中与磁盘操作有关的部分放在磁盘的系统磁道。也可以将一个盘的系统磁道上的系统拷贝到另一张盘的系统磁道上。当用其将 MOVCPMnn 建立的系统放入磁盘的系统磁道时，SYSGEN 运行过程如下：

```
A>SYSGEN
SYSGEN VER.2.2.03
SOURCE DRIVE NAME (OR RETURN TO SKIP)
```

当询问源驱动器时，可按 RETURN。因为 MOVCPMnn 已将系统放入预先确定的内存位置，在这种特定情况下，不必指定源盘。RETURN 响应使得 SYSGEN 程序将内存特定位置找到的系统作为源程序来生成新的系统。

然后 SYSGEN 提示：

```
DESTINATION DRIVE NAME (OR RETURN TO REBOOT)
```

用户可回答作为目的盘的驱动器名，插入目的盘，再按 RETURN 键。这时程序进行系统生成。在功能完成后，再次提示要求用户选择目的驱动器名。这时可根据用户需要进行选择。回答 RETURN，系统回到命令状态 A>。

如果要拷贝的系统在磁盘中，则需要将系统读到内存中来。系统生成过程同前面基本是一样的，只是对下面的提示响应不同：

```
SOURCE DRIVE NAME (OR RETURN TO SKIP)
```

这时响应应是源系统所在盘的驱动器名，并按回车。从相应的驱动器中将系统读入内存之后，SYSGEN 程序提示用户是否把 BIOS.SYS 从源盘上拷贝到目的盘中：

```
COPY BIOS.SYS (Y/N) :
```

回答 Y，则表示要进行 BIOS.SYS 的拷贝。如果源盘上没有 BIOS.SYS，则显示出错信息，然后 SYSGEN 程序继续进行。回答 N 时自动跳过对 BIOS.SYS 的拷贝。将系统写到目的盘上的过程与上面相同。

2.9 CONFIGUR 程序

CONFIGUR 程序可使用户方便地改制 CP/M BIOS，以适应特定的操作环境，用户可以使用这个程序设置终端、打印机、磁盘驱动器特性以及输入、输出到物理设备的逻辑映象的缺省值。

在 CP/M 命令状态下键入程序名 CONFIGUR，按 RETURN 键，便开始执行该程序。通常，用户在首次安装系统时，需要使用这个程序，系统配置好以后，就不再用它，除非系统中增加了新的硬设备。运行时，程序先自我标识，然后决定可用的硬件设备，自动决定 CP/M 可以使用的 RAM 容量、串行口数量及磁盘驱动器数量。当程序测试磁盘时，将对每个驱动器进行选择，确定该驱动器是否可用。对于需要 CONFIGUR 程序在配置时计入的驱动器，必须连到系统中，并接通电源。

下面叙述 CONFIGUR 的操作过程。

（1）CONFIGUR 运行开始

在调用CONFIGUR程序后，屏幕上显示出它所要找的硬件的有关信息，（见第一节1.3所示信息）。

RAM 容量（即可读写的内存容量）、磁盘驱动器类型与数量以及串行口的数量可根据特定硬件环境改变。显示在终端上的信息依赖于用户特定的硬件环境。

注：如果运行的 CONFIGUR 来自有写保护的磁盘，那么将警告用户只能修改在内存中 BIOS 的拷贝而不能改变磁盘上 BIOS 的任何值。

对于 CONFIGUR 的询问：

STANDARD SYSTEM (Y OR N) ? (Y) :

用户可以回答 Y 退回到 CP/M 命令状态，不修改任何参数，即使用标准硬件配置的BIOS。

相容版本检查功能

CONFIGUR 程序运行时将对 CONFIGUR 与 BIOS 是否有相同的版本号进行检查。否则，由于各个版本数据的存储位置不同，该程序可能不适当地修改 BIOS 的参数。

如果程序发现 BIOS 不是相同版本，则产生错误信息：

INCONSISTENT 版本号……Can NOT Configure!! 程序运行中止。这种不相容性可能是由于没有把最新版本的 BIOS.SYS 文件拷贝到驱动器 A 的磁盘中去，或者是由于使用了旧的 CONFIGUR 版本。

（2）Heath/Zenith 标准系统

所谓标准系统包括以下各项（其中每项的介绍分别在后面有关部分进行）：

驱动器步速 30ms

CRT口地址 350Q，波特率9600，回车后不送空字符

TTY口地址 320Q，波特率300，回车后不送空字符，无大写转换

MODEM口地址 330Q，波特率300，回车后不送空字符，无大写转换

行打印机口地址 340Q，波特率4800，回车后不送空字符，无大写转换

删除字符送回控制台

IOBYTE 置为 CON = CRT，RDR = ULL，PUN = UPL，LST = LPT

打印机就绪信号极性：低 (LOW)

打印机就绪信号：RTS（4 脚）

磁盘错误详细信息置为假 (FALSE)

热启动自动程序控制置为假 (FALSE)

冷启动自动程序控制置为真 (TRUE)

对 CONFIGUR 设置自动程序命令行

（3）主菜单

如果用户不选择标准系统配置，则以 N 回答最初的 CONFIGUR 提示。这时将出现一系列替换的任选项。用户可根据需要选择某个任选项，这些任选项确定包含特定的硬件。首先在屏幕上出现的菜单是主菜单。用于结束 CONFIGUR 程序或选择子菜单，它包括用户要检查或调整的任选项。

主菜单的形式为：

CP/M configuration

A Set Terminal and Printer Characteristics

B Set Disk Parameters
C Change the Default I/O Configuration
D Automatic Program Control

X Configure, making changes to memory only
Y Configure, making changes to both memory and disk
Z Quit, making no changes

Selection:

可打入任选项左面的字母选择相应的任选项。例如用户打入 B，可选择“置磁盘参数”。

选择主菜单中的 X，可使整个配置只在内存中改变。它可用来临时改变 BIOS，冷启动后，当 BIOS 的磁盘拷贝读入内存，则刚才改变的参数无效。

如果用户盘没有写保护，则可有 Y 任选项。选择 Y，使内存及磁盘中的 BIOS 均被改变。这种改变是永久性的，当然用户还可以再使用 CONFIGUR重新配置以修改 BIOS 的值。

注：如果磁盘是写保护的，则 CONFIGUR 只修改在内存中的 BIOS 拷贝，而不改变磁盘上 BIOS 的任何值。

任选项 Z 可安全地退出 CONFIGUR 程序而不改变任何 BIOS 的值。

每个子菜单同样也提供了返回到主菜单的方法，用户可通过选择 Y 或 Z 来实现。子菜单中的 Y 选择项保存改变了的 BIOS，然后返回主菜单。Z 任选项取消在子菜单中所做的任何修改而返回主菜单。

（4）终端/打印机特性子菜单

利用这个菜单，用户可以设置系统中串行设备的物理口及波特率。用户还可指定设备只能接收大写信息，这时 BIOS 将把送到该设备的所有小写信息转换成大写，进行输出。

另外，用户还可指定在回车后要送的空字符的数量。在回车后指定出现一定数量的空字符可产生一定时间的延迟，这是由于有些设备的打印头或光标返回时需要一定的延迟。

用户还可以选择 RS232C 线及“打印机就绪”信号的极性，这里所用的极性是指电平。计算机与打印机连接线的“打印机就绪”信号有两种电平。一些打印机采用低电平通知计算机，该打印机已准备好接收数据；而另一些打印机准备好接收数据时采用高电平。大多数 Heath/Zenith 打印机使用低电平信号，用户可参考打印机使用手册来决定自己的打印机使用的是什么信号线及哪种极性。

通常用于将打印机目前状态告诉计算机的 RS-232C 线是 DTR (Data Terminal Ready) 及 RTS (Ready To Send) 线。DTR 线（第20脚）当打印机接通电源时发信号给计算机。RTS 线（第 4 脚）当打印机准备好可进行操作时通知计算机。

可供用户定义功能的特性子菜单列出如下：

A CRT: Baud rate: 9600 Port: 0E8H = 350Q
B TTY: Baud rate: 300 port: 0D0H = 320Q
C LST: Baud rate: 4800 port: 0E0H = 340Q
D UL1: Baud rate: 300 port: 0D8H = 330Q

```
E  UP1:  Baud rate:  300   port:  0D8H=330Q

F  Force output to upper case on CRT:  TRUE
G  Force output to upper case on TTY:  FALSE
H  Force output to upper case on LST:  FALSE

I  Nulls outputted after CR on CRT:  0
J  Nulls outputted after CR on TTY:  0
K  Nulls outputted after CR on LST:  0
L  Echo on DELETE:  TRUE

M  Printer Ready Signal Polarity (HIGH, LOW):  LOW
N  Printer Ready Signal (DTR(pin20), RTS(pin4)):  RTS

Y  Finished, make changes and return to main menu
Z  Quit, make no changes and return to main menu

Selection:
```

选择项 A～E 提示波特率及口地址。CONFIGUR 只接收用户输入的有效波特率。以下几个均为有效波特率：38400，19200，9600，4800，2400，1800，1200，600，300，150，134．110及75。

这个任选项只改变计算机希望从指定设备接收数据所用的波特率，但不改变设备的波特率。如果要改变 CRT 的波特率，则首先要用 CONFIGUR 修改成所需波特率，然后在退出 CONFIGUR 后物理地改变 CRT 的波特率，以便与所选择的波特率匹配。如果先改变 CRT 波特率，那用户就不能与 CONFIGUR 程序进行联系，也就不能完成所希望的操作。

口地址的赋值可以十六进制形式或以八进制形式输入，但不需要用户在选择时输入 H 或 Q 基值后缀（Zenith Data System 使用字母 Q 来表示八进制（octal），这是由于字母 O 与数字 0 很容易混淆）。有效的口地址为：320Q(0D0H)，330Q(0D8H)，340Q(0E0H)及350Q(0E8H)。参看上面的例子。

任选项 F～H 不提示输入。当选择这些任选项时，真/假值交替变化。

任选项 I～K 提示在向设备发送回车之后所需要的空字符数。有效值范围是 0 到 7。

任选项 L 允许用户决定是否把用删除键删除的字符回送到终端上，或者是否用退格键在屏幕上删除字符。通常该值对硬拷贝机置为真值，对 CRT 终端置成假值。每当选择此任选项时，真/假值交替变化。每次运行 MOVCPMnn 及 SYSGEN，这个任选项自动地变成真值。

任选项 M 及 N 可使用户决定打印机就绪信号的特性及 RS-232C 线连接口的哪一脚用作为此信号。每次选择这两个任选项，程序自动地交替地切换它们相应的两个值，而不必用户输入选择项的值。

（5）磁盘参数子菜单

在这个子菜单中，用户可以选择在硬错误（不能恢复的错误）之后提示用户详细的磁盘错误信息。用户还可以选择硬分段或软分段5.25″磁盘（前提是具有相应的控制器来支持），同样还可以设置相应的5.25″盘的步速。步速是驱动器读写头从一个磁道到另一个磁道定位所需的时间。

5.25″硬分段盘驱动器（使用H17控制器）通常工作的步速设置为30ms。但它还可置成6～30ms之间任何相间2ms的值，建议步速设置为30ms。用户改变步速之前请参考下面的“步速出错检查”。

5.25″软分段磁盘驱动器（使用H37控制器）对密度为96tpi（磁道/英寸）磁盘驱动器设置在6ms下工作，而对48tpi，则在30ms下工作。这种驱动器步速可设成：30，20，12或6ms。同样在改变步速之前请参考下面的“步速出错检查”。

注：8″磁盘驱动器及“Winchester”硬盘的步速是不可调的。

步速出错检查

有时用户可通过降低驱动器的步速（即减少磁头在磁道间移动的时间）来获得某些性能的改善。要注意的是所选的步速要在合法范围内。如果步速调整得太低，系统可能根本不能工作，即系统需要多次对磁盘进行读写。如在读扇区时出错，则CP/M进行再读直至读正确（这种错误为“软错误”），或者当十次读均出错时（这种错为“硬错误”）报告出错信息为止。CP/M记住出现软错误的次数。

为检验当前磁盘出错频度，可用DDT检查BIOS表中的SECNTnn的值。只有在系统已经运行一段时间后才能这样做，这是由于SECNT在每次冷启动时置成0。

SECNT表示在磁盘读写操作中出现的软错误（可恢复的）的数目。这个值需要保持在相对低的水平上，并且在一小时内不应增得很大。如果该值变得太大，则应增加步速间隔，使读写头有时间到达所需要的磁道。

以下是显示在终端的磁盘参数子菜单的例子：

```
A   5.25 Inch Soft Sectored Unit 0  **cndefined**
B   5.25 Inch Soft Sectored Unit 1  **undefined**
C   5.25 Inch Soft Sectored Unit 2  **undefined**

D   5.25 Inch Hard Sectored Unit 0  Step Rate: 30 ms
E   5.25 Inch Hard Sectored Unit 1  Step Rate: 30 ms
F   5.25 Inch Hard Sectored Unit 2  Step Rate: 30 ms

G   Detailed Disk Error Massage: False

Y   Finished, make changes and return to main menu
Z   Quit, make no changes and return to main menu

Selection:
```

任选项A～C提示软分段磁盘驱动器在密度为48或96tpi时的步速。有效的步速是30，20，12及6 ms。如在硬件调查期间，有些磁盘没有找到，则相应显示**undefjned**。

任选项D～F提示硬分段驱动器步速，有效步速为6～30ms，增量2 ms，如未找到磁盘设备，则显示“**undefined**”。

任选项G不提示数值。每次选择时，将自动地进行真/假值交替。如选择前值为TRUE，则选择后自动地替换成FALSE。

（6）缺省I/O配置菜单

标准的BIOS支持CP/M IOBYTE来进行逻辑到物理设备的变换。见“CP/M2.2修改指南”。在标准的配置中，所支持的设备为：

CRT: 系统终端

TTY: 不具有同用户“交接处理”功能的ASCII码终端屏幕

BAT: 批处理伪设备，输入采用读入机逻辑设备（RDR:），输出采用列表逻辑设备（LST:）

UC1: 采用控制台作为逻辑输入设备，列表设备作为输出设备的伪设备

LPT: Heath WH-14或WH-24 (TI810)，具有硬件符号交换的打印机

UL1: Heath WH-44 (Diablo 1640或Diablo630)，它采用ETX/ACK规程的行打印机

PTR: 空字符源，不执行。当存取时返回文件尾（End-of-File）

UR1: 配置MODEM输入

UR2: 同CRT:

PTP: 空字符接收器，不执行

UP1: 配置MODEM输出

UP2: 同CRT:

缺省IOBYTE在冷启动时对逻辑设备进行赋值。它用来控制控制台终端的位置以及通常使用的行打印机的型号。

I/O配置菜单在终端上显示如下：

```
A  CON: = CRT: Available TTY:CRT:BAT:UC1:
B  RDR: = UR1: Available TTY:PTR:UR1:UR2:
C  PUN: = UP1: Available TTY:PTP:UP1:UP1:
D  LST: = LPT: Available TTY:CRT:LPT:UL2:
Y  Finished, make changes and return to main menu
Z  Quit, make no changes and return to main menu
Selection:
```

这里显示了当前逻辑到物理设备的映象。最左边一列字母为任选项。

任选项A～D提示用户输入一个有效的物理设备命名。在选择A、B、C或D之后，监控程序将显示相应的提示。例如，如选择D，则程序显示：

```
LST: =
```

当对于逻辑设备（LST: 为逻辑设备）提示输入三个字母的物理设备时，程序将自动地返回子菜单，并显示出所选作为替换的逻辑设备的新值。

```
LST: = TTY
```

注：冒号“:”不必输入，也不用打回车。

（7）自动程序控制子菜单

用户利用这个子菜单，可以指定在冷启动或热启动之后立即执行一个命令或程序。选择这一功能，用户必须在自动命令行中插入命令名或程序名。这可通过选择任选项C来完成。

自动程序控制子菜单的例子如下：

```
A  Run automatic command line on cold boot:TRUE
B  Run automatic command line on warm boot:FALSE
C  Automatic command line: CONFIGURE
Y  Finished,make changes and return to main menu
Z  Quit,make no changes and return to main menu
Selection:
```

任选项A、B可以真/假值交替变化，用于冷启动及热启动时程序的自动执行。任选项C要求用户输入命令行，然后将自动命令行重新显示出来。

这个任选项可在计算机启动时用于自动地装入BASIC或一些其他程序。下例说明用户怎样配置磁盘，使每次磁盘启动时自动执行BASIC程序“Adventure”。

a. 从主菜单中选择D，进入自动程序控制子菜单。

b. 在子菜单中选择C，这时显示

```
Automatic Command Line:
```

c. 打入BASIC ADVENTURE，显示

```
Automatic Command Line:  BASIC ADVENTURE
```

d. 按RETURN键。这时“BASIC ADVENTURN”将显示在菜单中，CONFIGUR程序等待下一次选择

e. 打入Y。这时CONFIGUR程序将合并上面的两次变化并重新显示出主菜单

f. 打入Y，退出主菜单。这时将变化的BIOS放入内存及磁盘中，然后返回CP/M命令状态

以后每次从这张盘上进行冷启动时，则自动程序特性将装入BASIC，然后装入程序“ADVENTURE”，并执行这个程序。

注：这时BASIC程序及ADVENTURE程序一定要在盘上，否则将显示出错信息。

每当从这张盘上进行热启动时也是如此。

2.10 ASSIGN 程序

ASSIGN是只用于Z67温式磁盘的实用程序，该程序允许用户给分区指定逻辑设备名。还可用于校验哪个分区指定了设备名，哪个分区可以指定，或取消某个分区的设备指定。使用为特定分区指定的设备，用户就可以调用CP/M和该分区打交道，就象使用独立的磁盘设备一样。

使用ASSIGN程序在温式磁盘上建立分区时，用户可以给每个分区指定一个长度达16字符的名，几个不同分区可以用相同的名，但用不同的值号来区分。例如，一个温式磁盘分为三个分区，用户可以将三个分区均指定名为CPM，第一个分区名的值为“CPM；0”，第二个分区为“CPM；1”，第三个分区为“CPM；2”。

ASSIGN程序共可执行四种操作：

（1）列出磁盘指定

用户可以打入：

A>ASSIGN?

看有哪些可能的设备指定	计算机可能回答
PARTITION NAME	MAXIMUM
	OCCURRENCE NUMBER

CPM 2

说明用户设置了三个分区：CPM;0, CPM;1, CPM;2

注：如果多个分区命名不同，则最大值号应为0。

（2）进行磁盘指定

用户可以打入：

A>ASSIGN X：=分区名{;n}

其中“X：”是设备名，“分区名”是给当前驻留的分区指定的名，“;n”是该分区名的值号。

例如：A>ASSIGN A：=CPM RETURN

A>ASSIGN B：=CPM;1 RETURN

本例假设Winchester系统中以软盘驱动器作为启动设备，且有两个称为CPM的分区存在。

（3）列出当前的指定

用户可以打入：

A>ASSIGN (然后按RETURN)

看已经完成的指定，在上例的情况下，响应为：

A:=CPM

B:=CPM;1

（4）取消逻辑设备名指定

用户可以打入：

A>ASSIGN X:

取消以前的指定，这里“X:”为设备名。

注意，对于ASSIGN使用有一些限制：

在温式磁盘系统中，对分区不能同时进行两种以上设备指定。

驱动器A:总是作为执行冷启动的设备。如果在Z67系统中，冷启动在8英寸软盘驱动器中执行，则B:和C:可指定为温式磁盘分区；如果冷启动在温式磁盘上执行，A:是冷启动分区，C:是8英寸软盘驱动器，则B:可以指定为第二个分区。

注：要使用ASSIGN实用程序，必须了解Z67及分区实用程序PART，请参阅有关资料。

2.11 BACKUP/RESTORE程序

BACKUP/RESTORE程序只用于Z67温式磁盘系统，该程序允许用户建立温式磁盘文件的拷贝备份，将这些文件放到软磁盘上，也可以将备份拷贝写回温式磁盘中。

该程序限制文件大小为8 MB，可用于顺序文件或随机文件。备份拷贝可以建立在几张盘上，并用卷号和建立日期来标识。在第一张盘上建立一个独立的目录表，用于处理分别存放在几张盘上的文件。该目录表上记有这个文件所存放的磁盘卷号。

下面叙述BACKUP/RESTORE程序的操作过程。

（1）开始操作

在系统盘上，BACKUP/RESTORE程序在目录表中名为BRS.COM。因此，在CP/M命令方式下打入

A>BRS　　按RETURN

就可以调用该程序。

然后，按照系统的提示，回答日期信息，指定所用的磁盘驱动器设备名。

（2）主菜单

在程序进入主菜单屏幕后，所有的操作都可以从主菜单进行选择，主菜单屏幕形式如下：

BACKUP AND RESTORE MASTER MENU

D DATE Change
B BACKUP Creation
R RESTORE Files
C COMPARE Files
L LIST Directory
E EXIT to CP/M

并列出只对BACKUP (B) 和RESTORE (R) 操作有效的任选：

V 写入后校核所有文件 (B/R)
W 不显示警告信息 (B/R)
A 允许只读文件恢复 (B/R)
F 保存后删除文件 (B)
P 提示所有动作 (B/R)

在屏幕底部显示选择操作提示：

Select Operation (X /Y/Y/Y) ==>

其中“X”为操作码，每个“/Y”为开关。

操作码可为D、B、R、C、L、E中任意一个。

上述五个任选项可以作为开关形式使用(/V、/W、/A、/F、/P)，并可选用多个开关参数，例如：

Select Operation (X /Y/Y/Y) ==>R/V/P

下面简要叙述一下主菜单所列的操作。

a.改变日期操作

用于改变开始时按系统提示回答的日期信息（如果回答的日期信息不正确）；选择D。

b.建立备份拷贝操作

建立备份拷贝对于防止不测事故造成磁盘文件破坏是十分重要的。这些事故包括硬件故障、电源故障、程序误操作或对磁盘的错误的写操作等。

在主菜单中选择B（并可选择所需的开关参数），则执行建立备份拷贝操作。

当用BACKUP（或RESTORE）操作传送文件时，用户必须指定该操作传送的文件名以及不被传送的文件名。BSR根据用户给出的这两种信息，建立一个文件表，用于后继操作。

首先，该操作提示指定不被传送的文件名：

Enter Rejection File Spec (s) .==>

对于上述提示回答的文件名，将不包括在被拷贝（或恢复）的文件表中，且拒绝的文件指定复盖选择的文件指定。

回答有三种形式：

对CP/M文件，送入文件名（例如 TESTFILE.ASM）

对CP/M批处理文件，文件名前面加分号（例如;BATCHFIL.SUB）

或者回答RETURN，表示没有文件拒绝该操作

BRS最多允许指定40个文件为不传送文件。

回答RETURN后进入选择文件指定，对于选择文件提示，也有三种形式的回答(同上)。

对选择文件指定，也可以有若干个开关参数（次开关参数），可供选择。选择指定任选如下：

A 所有文件型（缺省）

N 非系统文件

S 系统文件

U 当前用户号（缺省）

U* 所有用户

U。 某个指定的用户号

提示形式为：

Enter Selection File Spec (s) (X /Y/Y/Y) ==>

其中X为CP/M文件名

Y为选择开关（/A，/N，/S，/U，/U*，/U。）

c.恢复文件操作

用于将备份拷贝文件重新存入磁盘中。从主菜单选择打入R，按RETURN，则执行恢复操作。恢复操作与建立备份操作基本相同。但建立备份操作是将文件从温式磁盘传送到软盘中，而恢复操作是将文件从指定为软磁盘的设备传送到温式磁盘设备上。在主菜单中的F任选，不能用于恢复操作。另外，恢复操作要求首先插入带目录表的软盘，即在建立备份操作时的第一张盘。

d.比较文件操作

当建立备份拷贝，且需要从该备份拷贝中将文件重新存入磁盘时，必须保证备份中的文件与产生该备份拷贝的源文件完全相同，即保证文件的完整性。可用“/V”任选在文件传送过程中进行核对。如果后来文件作了修改，或传送时未选择“/V”，则可用比较操作。

对于主菜单提示，回答C，按RETURN，可执行比较操作。

e.列目录表操作

列目录表操作可使用户能存取文件目录表，除文件名外，还提供一些其他信息，如拷贝建立的日期、一组特定磁盘的卷号、每个文件所属的用户号和文件所在的卷号。

f.返回CP/M系统

主菜单提供了“E”操作，用于退出BACKUP/RESTORE操作，返回CP/M命令方式。

第三节 BIOS 系统组织

Heath/Zenith在实现CP/M2.2版本时增加了一些特性，以提高标准CP/M 系统的性能，其中包括增加了操作系统的硬件支持。因此，基本I/O系统BIOS所占的内存空间比 数 字 研究公司标准CP/M所提供的要大些。

因而，Heath/zenitn设计了一个程序将BIOS 放在系统磁道的恰当位置，该程序可以 使用CP/M文件系统打开磁盘文件，并读入整个 BIOS。这个程序称为BIOS装入程序(BLDR)。

当系统冷启动时，监控程序ROM将所选的驱动器盘从第一个扇区开始读入内存，然后 控制交给该程序（冷启动装入程序，在扇区前半部分），它将这些扇区内容读入内存，读完之后，控制交给新装入系统的冷启动引导入口点。

由于系统冷启动时要打开 BIOS.SYS文件，将它读入，并且在内存中重新定位，因此，冷启动过程要长一些。

3.1 BIOS中的基本地址

Heath/Zenith对 BIOS 进行扩展，这样，系统实用程序可以容易地存取设备地址表。首先是CONFIGUR需要，它可以改变在运行时间所使用的一些参数值。在通常CP/M版本中，许多值汇编时是不改变的。

用户程序可以通过使用0001H单元的地址值作为指针（热启动转移指令的操作数）来调用BIOS，这个值比BIOS起始地址高出三个单元。

BIOS的第一部分是标准的Digital Research入口点转移表。这个表包括到17个 BIOS入口的转移指令。

这次新的版本增加了“Listst”入口，用来检查列表设备的状态。这个入口在一些后台打印假脱机程序中使用。

紧接着标准入口的是一个单字节的BIOS版本号（二进制形式）。在BIOS中直接修改参数的程序应确切地知道所修改的程序是什么版本，这点是很重要的。例 如，CONFIGUR必须在BIOS开始增补一些表格，因此需要适当地确定这些表预计在什么地方可以找到。

在冷启动时，CP/M显示出其版本号，形如：

32K Heath/Zenith CP/M2.2.03

这里“03”为版本号。

再下面一个字节用作为BIOS的方式字节。这时有六个位被赋值，而其余作为 保 留。第七位使得CCP在每次磁盘冷启动时执行缺省磁盘上的自动命令行。第六位执行同每 个 热 启动相同的功能。第三位包含打印机就绪信号标志，为0时表示低，为1时表示 高。第 二 位支持Z67 Winchester硬盘的分区操作。第一位由 BIOS 用来打印磁盘出错信息，这样 用户可以得到扩展的错误状态信息。当配置BIOS在控制台口上使用H8-5串行板 时，将 第 0 位 置位。

后面数个字节作为串行设备控制结构。这些字节指定口地址、波特率、回车后所需的空字符数量以及对每一串行I/O设备的大写变换标志。数据安排为口地址，其后为波特率因子。

如果MSB为1，波特率因子高位半字节强迫所有输出均为大写形式，这个半字节的后三位为回车后要送的空字符个数。每五个设备有一个这样的结构（但当前只使用4个）。它们顺序为：CRT，TTY，LPT，RDR/PUN或者以相反顺序。

然后使用两个字（四个字节）作为5.25″磁盘驱动器的软错误计数器。这些字在每次软（可恢复）错误之后计数增加一次。用户可使用DDT来检查这个值，了解磁盘系统的性能。

下一个字节用来表示这个BIOS可以配置的最大磁盘数。程序使用它可以决定磁盘表的长度，并只对真正存在的表进行存取。

再下面是磁盘参数基址以及每个磁盘驱动器的磁盘参数表项。在Digital Research规定的每个磁盘16个字节的基础上，Heath/Zenith为每个表项又扩展了8个字节。

这8个字节的第一个是通用字节，它用来标识驱动器类型及相应口地址，三个最高位指定驱动器类型。第二个字节是选择码。第三个字节是每个物理扇区包含128字节记录数。第四个字节指定每个存储数据块中包含128字节的记录数。后四个字节用于除H/Z47外的所有驱动器控制器。

对5.25″磁盘驱动器，这四个字节的第一个指定此驱动器磁头的当前磁道位置。第二个字节控制其步速。当置位时，其最高位表示驱动器速度达到250 ms，而不是通常的1000ms。下一个字节包含多个标志。最后一个字节保留，在物理到逻辑驱动器映照过程中使用。

除这些表之外，有些系统实用程序还需要使用一些值来控制磁盘，这些值已经放在BIOS及内存的0页中。它们控制驱动器马达并选择超时范围，以及磁头固定和写脉冲步长的延迟。

在000DH，存储了H/Z89在口地址362Q控制锁存的当前值。这个口控制第0页RAM的存在以及时钟的激活和重置位。000EH存储对H8计算机时钟重置位所需的参数，对于H/Z89其值为0。

000FH单元保留用于存H17/H77/Z87控制口的内容。

0040-004FH为16字节保留区。

BIOS任选项

当CP/M启动时，系统产生如下信息：

```
    32K Heatn/Zenitn CP/M2.2.03
    FOR{type}DISKS WITH OPTION{code}
```

{code}表示当前在BIOS中选中的任选项。

以下列出各个任选项：

P　Z67被分区

P_2　BIOS支持两个Z67分区

I　CRT输入为中断驱动

B　CRT “BREAK” 键中断有效

E_3　BIOS支持Z37驱动器的扩展双密度

E_4　BIOS支持H/Z47的扩展双密度

T　日历时钟

E　事件计数器

3.2 磁盘格式

5.25″硬分段磁盘格式

逻辑记录每一磁道编号是1～20，并存储在0～9个物理扇区中。磁盘的磁道编号是0～39。CP/M系统保留前三个磁道供冷启动装入程序、CCP、BDOS以及BIOS装入程序使用。CP/M的目录从3磁道开始，紧接着就是可用空间。

8″和5.25″软分段磁盘格式

Heath/Zenith扩展BIOS支持单密度、双密度及扩展双密度磁盘。另外，每一种密度可以记录在单面，也可记录在双面介质。在磁盘复位功能（如热启动）之后，磁盘被登录。接着检查磁盘的密度以及磁盘可用的面数，并用来动态地修改特定驱动器的磁盘参数表。不同密度和可记录面的磁盘在互相交换后应该进行登录，这是很重要的。

对8″软盘，Heath/Zenith提供用户三种可使用的密度。单密度为每磁道26个扇区，每个扇区128个字节。双密度为每磁道26个扇区，每个扇区256个字节。而扩展双密度每磁道8个扇区，每个扇区1024个字节。在双密度及扩展双密度盘上使用解块技术，可使每个磁道分别包含52和64个扇区。

5.25″软分段盘有两种可能的密度。单密度为每磁道10个扇区，每个扇区256个字节。而双密度为每个磁道16个扇区，每个扇区256个字节。

第四节　硬件配置

这个CP/M扩展版本可支持的最小系统配置为：

1．最小随机存取读写内存为32K的H/Z89计算机，必须安装Z89-7或H88-7ROM修改装置。H/Z89内存可扩充到64K。

2．一个磁盘驱动器（H/Z89自带以H/Z17为控制器的单驱动器）。

系统可以支持的磁盘驱动器为：

1．1～3个硬分段5.25″软盘驱动器（H/Z17控制器）

2．1～3个软分段5.25″软盘驱动器（Z37控制器）

3．2个8″软盘驱动器（H/Z47控制器）

4．1个Winchester硬盘系统（10MB Winchester磁盘及单个8″软盘；Z67控制器）

系统最多可配置六个驱动器（但同时只能有两种类型的驱动器）。

系统支持的打印机为：

1．Diablo 630、1610、1620、1640或1650（兼容的）打印机，其波特率置为1200，RS-232C接线连到DCE340-347输出插口。UL1：逻辑驱动设备应连到这个口上。使用ETX/ACK信号交换规程控制打印机输出。

2．H14、WH14、WH24或H/Z25点阵式打印机，使用RTS MODEM进行信号交换，连到DCE340-347口上。如用户需要其他的信号交换规则（除CONFIGUR所可能涉及到之外的），要对BIOS物理设备驱动程序进行修改。

3．WH36或其他任何不需要“忙”信号交换的打印机。它们可以连到TTY：逻辑设备上，也能插到DCE340-347口上。在CONFIGUR程序中指定TTY：口地址为340Q，将LST：设备口地址改为320Q(H/Z89计算机没有提供320Q口，用户可以补加到串行接口板的空脚上）。

4．WH13 MODEM，连到DTE330-337口上

第五节　扩展的磁盘错误信息

Heath/Zenith BIOS 在对磁盘读写时遇到错误，BIOS 就重新再执行这个命令。每当5.25″盘出现软错误（即可恢复的错误）时，相应在BIOS表中的SECNTnn计数增加一次。如果仍不能读写扇区，这时错误标识为硬（或不可恢复）错误。硬错误使操作系统产生一个错误信息，标志遇到了读或写错误。除这个信息外，BIOS 还提供一个包含描述错误类型的错误码。

这些错误码均为两位十六进制数，它指定所遇到的错误类型。每个磁盘控制器具有其自己的错误码。每一类型驱动器的错误码表包括在相应设备的硬件手册中(H/Z17控制器例外）。

下面列出H/Z17的错误状态代码：

01　坏磁道错。大多出现在软盘机构中，它常常是由于CONFIGUR中BIOS的驱动器步速设置得太短。

02　磁头同步错。发生在软盘格式化中。

04　标题CRC错。在读扇区标识标题时出错，通常是由于存储介质损坏所引起。

08　数据CRC错。在读扇区的数据时出现，它可能是由于介质损坏、在写过程中掉电或误操作以及硬件损坏造成。

10　记录没有找到。不能对指定的磁道和扇区进行地址分配。可能是由于定位不准，存储介质不良、驱动器转速不正确以及指定了不存在的磁道或扇区而造成。

20　数据同步失配。发生在软盘格式化时。

40　写保护错，当向有写保护的磁盘进行写操作时出现这种错误。

80　设备没有准备好。在对驱动器进行读或写时，驱动器还没有准备好。这可能是由于驱动器没有磁盘或驱动器门没有关闭而造成的。

附录　CP/M2.2命令功能综述

规则说明

ufn 确定的无歧义性文件名，唯一地指定一个文件。它最多由 8 个字符组成，包含最多 3 个字符的扩展名，中间用句号“.”隔开，文件名中不能包含下列特殊字符：

<>. , ; : = ? * [ ]

扩展名描述文件类型，例如：

.ASM　汇编语言源文件

.HEX　Intel十六进制格式文件

.COM　可执行的“命令”文件

.SUB　可提交文件

.BAS　BASIC文件

.DAT　其他数据文件

afn　有歧义性文件名，与无歧义性文件名类似。不同的是它还可使用通配符去匹配一组文件名，如“*”匹配整个基本名和扩展名，“?”匹配单个字母数字字符

[ ]　方括号内的条文用于“开关”设置，方括号由用户打入

:　提示用冒号结束时只需单字符输入，无需打回车键

所有其他输入行均应以回车结束

↑　放在字符前面，意为打入命令字符时，同时按下CTRL键

{}　花括号用于指定任选输入，括号不打入

键盘操作

BACKSPACE　删除前一个字符，显示终端光标退回一个字符位置

DELETE　删除前一个字符，并送回删除符号，对打印终端是有用的

↑C　热引导（当位于一行第一个字符时）

↑E　物理行结束

↑J　换行，结束当前输入

↑M　回车，结束当前输入

↑R　重打输入行

↑U　删除整行输入

↑X　退至当前行开始位置

↑P　控制打印机开/关，当打开时，打印机复制控制台输出

MAKEBIOS　建立特定硬件参数用的BIOS：

命令行：SUBMIT {y:} MAKEBIOS x:{ufn}{y:}

x:　目的驱动器名

注意：如果指定目的地与源驱动器相同，则新 BIOS 将复盖现存 BIOS，使之不可恢复。

ufn　任选指定目的文件名，以避免当前 BIOS 被重写

y:　任选驱动器名，存放 BIOS 源文件

CCP 命令

DIR afn 列出符合文件名指定 (afn) 的所有文件目录

DIR 等价于 DIR*.*

ERA afn 清除符合文件名指定 (afn) 的所有文件

TYPE ufn 在控制台上印出指定文件的内容

USER n 把活动用户号改为n (0～15)

SAVE n ufn 从TPA的0100H开始保存n页 (每页256字节) 作为唯一命名的文件

REN ufn1=ufn2 将文件ufn2改名为ufn1

ASM 汇编程序

命令行格式: ASM FILE.MNO

其中M=源驱动器

N=十六进制输出文件所在的驱动器

或Z为非十六进制文件

O=列表文件所在驱动器

或X表示将列表文件送到控制台

或Z表示无列表文件

出错信息:

D 数据错

E 表达式——非法格式或在汇编时不能作计算

L 标号错——非法位置

N 在这个ASM版本中不能实现

O 过分复杂的表达式

P 标号值错——在程序中两次扫描标号具有不同的值

R 寄存器错——指定的寄存器在这种情况下是非法的

V 值错——非法格式的操作数

DDT 动态调试工具

(下面所有值均以十六进制表示)

命令行: DDT cr

或DDT file.ext cr

Axxxx 从xxxxH单元开始逐行汇编

Dxxxx{,yyyy}以十六进制和ASCII码倾出存储器内容

Fxxxx,yyyy,bb用十六进制字节bb填入内存xxxx到yyyy单元

G{xxxx}{,yyyy}从指针PC或xxxx单元开始执行,在yyyy单元设任选断点

Hxxxx,yyyy 十六进制和、差

Itext 用text建立标准文件控制块

L{xxxx}{,yyyy} 从xxxx单元开始反汇编

Mxxxx,yyyy,zzzz 从xxxx到yyyy单元数据块传送到zzzz单元

R{xxxx} 以任选位移读入文件

Sxxxx 从xxxx单元开始代换 (以非法十六进制字符结束)

T{n}　跟踪n步（缺省值为1）

u{n}　同T，但中间过程不显示

X{regpair}　检查CPU寄存器，如指定了寄存器对，则允许改变

ED本文编辑程序

命令行：ED y：file.ext{x：}

对文件file.ext进行编辑，如果该文件不存在，则建立一个新文件。编辑过程结束后，原文件更名为file.BAK，被编辑文件取名为file.ext。调用时，用x：指定输出文件所在的驱动器（对于编辑大于磁盘容量一半的文件是很有用的）。

大多数编辑命令均对假想字符指针（CP）所在位置的文本进行操作。CP可在整个内存缓冲区内移动。

注意：可用＊取代整数值n。＊等于“all”，最大值为缓冲区最大尺寸或65535（这是所允许的最大数字值）。数值n不带符号时为正数（没有必要打入“+”号），为负值时前面要带“-”号。

文件传送

nA　将n行文本添加到缓冲区中，从文件首开始，或者从没有被添加的文件部分开始

＊A　添加所有行，直至最大缓冲区尺寸

0A　添行填满缓冲区的一半

nW　写前n行至临时文件file.$$$

W　写满缓冲区

0W　将缓冲区的一半写出

nX　拷贝n行至临时文件X$$$$$$$$.LIB

0X　将文件置空

R　读库文件

O　返回原文件，重开始编辑

Q　退出编辑环境，不更新文件，返回CP/M命令方式

H　保存经过编辑的文件，将原文件更名为file.BAK，清除缓冲区，并留在编辑方式

E　结束编辑，保存经过编辑的文件，原文件更名为file.BAK并返回CP/M命令方式

CP移动

B　将CP移到缓冲区开始位置，-B　将CP移到缓冲区结束位置

n　将CP移到逻辑行号n

±nl　将CP向前（+）移n行，或将CP向后（-）移n行；

L为向前移一行（同1L），-L为向后移一行（同-1L）

±nC　将CP向前（+）移n个字符，或将CP向后（-）移n个字符；

C为向前移一个字符（同1C）；-C为向后移一个字符（同-1C）

nFstring（cr或↑Z）　查找并移动CP至第n次出现的F字符串，以回车（cr）或control-Z（↑Z）结束操作

nNstring（cr或↑Z）　同前一命令。但如果当前缓冲区中没有引用的字符串，则自动将附加文本添加到缓冲区中

文本显示

±nT　印出前面（－）n行，或印出后面（＋）n行，包括当前行。T为打印当前行（同1T），－T为打印当前行的前一行（同－1T）

文本修改

I（↑Z）将打入I以后一直到打入↑Z的全部输入均插入文本中，每次插入，CP指针向前移动一个字符，当打入↑Z时，定位在插入结束的地方

±nD　删除CP右边（＋）或左边（－）的n个字符，如删除右边（＋）字符时，包括CP位置的字符。被删除字符同一行右边的字符均向左移动n个空格。如果删除的是左边（－）n个字符，则CP也向左移n个空格。D删除CP所在位置的字符（同1D），－D删除前面一个字符（同－1D）

±nK　删除CP所在行前面的n行文本（－），或CP所在行后面的n行文本（＋），包括CP所在行。所有跟在被删除行后面的文本均往上移n行。如果前面行被删除（－），CP也上移n行。K删除当前行（同1K），－K删除前一行（同－1K）

其他命令

±U　如果为加号（＋）时，将字符转换为大写；如果为减号（－）时，不转换

±V　如果为加号（＋）时，打开逻辑行数设置；如果为减号（－）时，关闭逻辑行数设置

0V打印缓冲区中全部自由空间

nM　定义宏命令序列

±nP　传送及打印前n页（－），或传送及打印后面n页（＋），包括当前页

nnn：命令串：mmm　执行从逻辑行nnn：开始，逻辑行：mmm结束的给定命令串。nnn：：mmm指定命令的行号范围

LIST　实用程序

LIST ufn {…{switches}

或LIST　cr

* ufn{ufn…}{switches}

* ufn…

* cr

这里switches是

〔D　xxxxxxxxxx〕　首十个字符（或到该数据字段结束）被用作日期并印在标题中页号的左边

〔H text〕　首六十个字符（或直至该数据字段结束）被用作标题并打印在每页的首行上。如不出现，隐含为文件名

〔Lnn〕　置用户文本每页行数，缺省值60。L＝00时关闭分页设置

〔N〕　无标题行（隐含无页号）

〔Tnn〕　标记停在每个nn位置，缺省值8

〔Pnn〕　从nn开始页编号，缺省值1

〔U〕　强制所有输出为大写

〔Cnn〕　由该命令行建立的输出拷贝数，缺省值1

〔E〕　在打印后清除该命令行的文件

当在提示方式下使用时，所有设置（除开始页号和删除）均保持有效，直至程序终结。

在打印期间，可从控制台键盘打入任意字符中断输出。

SUBMIT 实用程序

SUBMIT ufn{参数}cr

SUBMIT命令使一系列CP/M命令自动执行，这些命令在文件“ufn”中顺序列出，该文件必须具有文件型“.SUB”，并驻留在驱动器A。SUBMIT命令还允许提交参数至SUB文件的命令中。这些参数在SUBMIT命令行文件名之后打入。如果参数被提交，SUB文件将包含参数“$”，其格式如下：

$\$_1$ $\$_2$ $\$_3$…$\$_n$

这些参数对应于文件提交执行时包含的实际参数。当该文件执行时，$\$_1$为命令行列出的第一个参数所取代，$\$_2$为第二个参数所取代，依此类推。

XSUB——SUBMIT扩展

当SUBMIT文件的首行为XSUB命令时，SUBMIT实用程序的功能扩充为不但接受CP/M命令，而且接受程序行输入。

PIP 外设交换程序（文件传送实用程序）

PIP命令行cr

或PIPcr

*命令行cr

*命令行cr

*cr

命令行格式如下：

destination = source1{,source2…}{switches}

其中destination为接收数据的目的文件或设备，source为一个或多个源文件，或提供数据的设备。

除标准逻辑设备之外，PIP还可以引用：

NUL: 40个空字符的源文件

EOF: CP/M文件结束标记（↑Z）

INP: 可填补到PIP中的特殊PIP输入源文件

OUT: 可填补到PIP中的特殊PIP输出目的文件

PRN: 同LST: 但标记扩展为每行第8列，对行进行编号，每60行插入跳页

switches为开关，开关设置用方括号[]

可用的开关有：

B 以数据组方式传送，数据存入缓冲区直至从源文件接受到ctl-s

Dn 删除超出n列的字符

E 执行结果送回控制台

F 从文件中取消打印纸送给

Gn 从用户n（0～15）取文件

H 十六进制数据传送，所有数据按Intel十六进制文件格式检查取消，非基本字符

I 忽略Intel十六进制文件格式：00记录，自动设置“H”

L 将大写字母转变为小写字母

N　加行号至每一行，从1开始，按1增量，取消前零，跟以冒号。N2包含前零，并跟在带标记的行号后面

O　目的文件传送，嵌入的CP/M文件结束标记（^Z）忽略

Pn　每n行包含一个跳页

Qs^Z　当碰到字符串s时停止从源文件拷贝

R　读系统文件（带有$SYS设置的文件）

Ss^Z　当碰到字符串s时开始从源文件拷贝

Tn　扩展标记至每个第n列

U　将小写字母转变为大写字母

V　在写操作后重读，以核对数据拷贝是否正确（目的文件必须在磁盘上）

W　写在R/O文件上而不进行控制台询问

Z　将每个ASCII字符奇偶校验位置0

ASSIGN　指定硬磁盘设备名

命令行：

ASSIGN　列出当前磁盘名指定

ASSIGN?　列出分区名和用于指定磁盘分区的最大的具体值

ASSIGN x：partition-name　:n　将逻辑设备名x：指定给分区名的第n个值，如果略去:n，则假定为第一个值

ASSIGN x:　取消对逻辑设备名x：的任何指定

STAT 系统状态实用程序

STAT cr　打印出所有联机驱动器自由空间数量（以K字节为单位），包括只读状态的驱动器

STATx：cr　选定磁盘，然后显示自由空间数量（以K为单位）

STAT afn　cr　显示符合afn指定的所有文件大小、存取特权和文件名(按字母顺序)，在“（）”内的文件名说明该文件标记为$SYS

STAT　x：=R/O　cr　标志驱动器x为只读（直至下一次热启动引导）

STAT VAL：cr　显示可能的STAT命令及可允许的逻辑设备到物理设备的映象

STAT DEV：cr　显示当前逻辑设备到物理设备的映象

STAT afn　指示符cr设置文件afn的永久性文件指示符，其中可允许的指示符为：

$R/O　只读

$R/W　读写

$SYS　系统文件，用DIR不能显示

$DIR　在目录表中显示的文件

STAT DSK:　列出所有当前联机磁盘驱动器的特性

STAT x：DSK:　列出指定的磁盘驱动器特性

STAT USR:　显示活动用户号及在当前缺省磁盘上有文件的全部用户号

MOVCPM

建立指定大小的操作系统以适应各种硬件要求

命令行：MOVCPM xx{memory size{BIOS descriptor}}

xx引导设备控制器（必须的）：

17 H/Z17磁盘控制器（5英寸硬分段）

37 H/Z37磁盘控制器（5英寸软分段）

47 H/Z47磁盘控制器（8英寸，IBM格式）

67 H/Z67磁盘控制器（Heath/Zenith winchester硬盘系统）

{memory size}给出所用内存大小（任选的）：

nn 整数，表示内存大小（K字节），该整数值可小于连接的可用内存最大数量，但不能大于最大内存

* 操作系统检测硬件找到连接的可用内存并全部使用之

null 当不指定{memory size}时，MOVCPM把它看作指定了“*”号

{BIOS descriptor}描述建立操作系统时所用的BIOS（任选的）：

* 使用当前内存中的BIOS，必须指定{memory size}

null 当不指定位置时使用当前内存中的BIOS,同“*”指定,但{memovy size}可为任选的

x: 使用在驱动器名x:上的BIOS，必须指定{memory size}

x:ufn 使用驱动器x:上的确定文件名ufn作BIOS，必须指定{memory size}

注：如果指定{BIOS descriptor}，则必须指定{memory size}

CONDOR SERIES/20 rDBMS
所使用的术语

Alphabetic Data:

字符类数据，它可以是字母A～Z、a～z或字符“-”、“.”、“+”

Boot-up:

对特定的微型机所进行的系统初始化过程，它将操作系统（本手册指定为CP/M）装入内存

Command Procedure:

一个存贮在磁盘文件上的命令集合。执行时，按顺序执行文件中的每条命令

Data Base:

描述屏幕格式、数据项、数据项定义及所分配的磁盘空间的磁盘文件的集合

Database1:

在本手册中，指定为目的数据库

Database2:

在本手册中，指定为源数据库

Data Definitions:

数据项的特征（例如数据类型、长度、最小值、最大值以及缺省值），数据定义被存贮在磁盘文件上，并动态地与数据库发生联系。

Data Dictionary:

一个CONDOR SERIES/20 rDBMS数据库文件，它包含的记录分别描述每一个被定义的数据库及相应的数据库文件

Data Directory:

数据库文件DAT的标题记录，其表头包含着数据项个数、记录的长度及数据库记录的个数

Data File:

数据库文件，其中包含数据库的数据

Data Item (name):

最小的信息单位，称为数据项（或数据字段），例如CITY、AGE、NAME，在存储的数据记录中分配给数据项名一个固定的空间。数据项名在CRT的屏幕格式中被指定，而在数据定义中被描述

Data (Field) Size:

分配给每一个数据项的磁盘存储器的字节数

Data (Field) Type:

指明数据项的数据类型是Alphabetil（A类）、Alphabetic-Numberic（AN类）、Num

beric（N类）、Dollar（$类)或Date（J类，这里J指定为公历日期）

Data Value:

指明存储在磁盘文件的数据项的值。对AN或A类，指出它的字符个数，对N、$，指出它的值域，对J类，指出它的日期值

Default Value:

预先定义好的数据值，当没有其他值进入的情况下，进入缺省值到相应的数据字段上。

Edit Criteria:

为数据项规定好数据的最小值、最大值和缺省值以及数据类型，本rDBMS禁止不符合要求的数据，如出错，它将通知用户

Field:

指定一个记录所占有的磁盘空间的大小

File:

用来存储分配在磁盘上的数据、数据格式、数据定义、批处理命令等磁盘空间。一个数据库文件是一个数据库记录的集合

Format:

一个能在CRT上进行“绘画”的屏幕编辑功能

Input:

将有关数据录入到数据库中的处理过程，录入数据到一个文件上，描述一个正在录入或已经录入到计算机中的数据集

Julian:

用数字代表日期值，其格式为mm/dd/yy，该值占有三个字节

Master/Transaction:

指数据文件（如主文件、事务文件）。本手册中给出了一个具体的事务处理例子，如用事务处理文件更新主文件，即以后的用journal去更新general ledger

Matching Field:

与词“Matching Data item”同义，它要求在两个文件中具有相同的名字及数据定义

Record:

逻辑上表示为数据项的集合，其内容指定为数据项值的集合

Schema:

模式：一个数据库的主要结构。用来描述数据项间的相互关系（其中包括有关的输入屏幕格式和数据项定义）

Sub-Schema:

选择的数据项、输入屏幕格式、数据定义模式的视图。例如：一个应用程序可以用一种格式输入而用另一种格式输出

微型机关系数据库CONDOR SERIES/20 rDBMS使用手册

CONDOR关系数据库管理系统的特性

1. CONDOR SERIES/20 rDBMS（简称为rDBMS）是一个关系式的数据库管理系统。它比网络和层次式的数据库更易于理解和掌握。

2. rDBMS语言自身类似于英语。它不需要其他计算机语言，例如BASIC的支持。用户仅需使用rDBMS语言就可以发展自己的应用程序。开发大部分数据库不需要专业计算机程序员。

3. rDBMS源于DBM-I版本。自1977年以来，它已在Z-80微型机上做了现场试验。

4. rDBMS支持事务处理。它有把事务文件汇总到主文件的有效方法。并且每次事务汇总时都形成帐底记录，它可作为主文件记录而加到数据库中。

5. 数据词典系统为数据库提供了同义名管理，同时它也为同一个数据库提供不同的登记项和显示格式。

6. 用户在CRT上，按照置入的编辑标准录入和更新数据。例如，录入数据库的值一定要满足定义的最大、最小值间的值域。

7. 使用定义命令（DEFINE）可以在几分钟内就建立一个数据库。

8. 计算命令（COMPUTE）允许多达32项方程计算式。

9. 分类命令（SORT）可对多达32个数据项进行快速而有效的分类。

10. 选择命令（SELECT）可选择多达32个满足逻辑条件的记录。其中，允许使用AND、OR和NOT连接而成布尔条件。

11. 联接和比较命令（JOIN和COMPARE）在指定匹配关系的数据项上形成数据库的二元联系。

12. 实现命令成批处理。

13. 广泛的错误检查、提供信息功能。

第一章 概 论

本手册是专为第一次使用微型计算机系统的用户而设计的。它通过一步步的使用过程，用英语式的语言帮助用户开发一个CONDOR SERIES/20 rDBMS数据库管理应用程序。

这一章使用CONDOR SERIES/20 rDBMS的命令实例说明这个系统的特性，在你工作之前，你应当熟悉一下CONDOR SERIES/20 rDBMS的命令语法以及在开发应用中的其他基本特性。

第一节 命令语句

所有给处理机的指示都是以命令的形式给出的。一个命令语句包含一个象动词一样的成份。如数据库名或文件名以及命令中所带的附加受限信息、任选开关等参数。这里受限信息用来描述数据库、文件数据项和进行加法、比较、赋值等逻辑运算和算术运算。一个CONDOR SERIES/20 rDBMS命令语句语法如下：

〈命令〉{驱动器名:}〈数据库名〉/〈文件名〉{受限信息}〔选择项〕

例如：

UPDATE B:EMPLOYEE WHERE NAME IS W·B·SMITH

〈命令〉：在本手册附录A中具体描述每一个CONDOR SERIES/20 rDBMS的功能和使用。

〈驱动器名〉：表示可选择的驱动器。在本系统中，允许使用的驱动器可为A～H。如B:表示B驱动器。

〈数据库名〉/〈文件名〉：不带扩展名的数据库名或带扩展名的文件名(扩展名由三个字符组成，它跟在文件名的后边，用来标识文件的类别)。

数据库名引用本rDBMS系统的标题。每个数据库由三个数据库文件组成，它们是格式文件（带扩展名.FRM）、定义文件（带扩展名.DEF）、数据文件（带扩展名.DAT)。每个数据库名不得超过8个字符。

标准的文件扩展名有：（.DAT）、(.DEF)、(.FRM)、(.DIC)、(.DIR)、(.HLP)、(.SUB)、(.CMD)及(.CVL)。这些文件将在以后各章中进行详细说明。

〈受限信息〉：用户指定的数据项名清单和随rDBMS命令变化的操作数、选择项(option)。

在表中，通配字符“*”是非常有用的，它可以用来简化数据项名。例如ADDRFSS，可以简写成ADDR*（“*”出现的地方可以用任意字符串代替）。另外，通配字符“?”指示计算机接受数据项相应位置上的任何值。例如：如果某数据项存放数据形式是以两个姓、后跟姓名组成（如W·B·SMITH），它在指令中可以写成“……WHERE NAME IS?? ? ? SMI*…”,它将指示计算机检索所有以SMI开始的姓名。

〈选择项〉：用户可以使用任选项(option)来设置任选开关，它可使rDBMS命令操作多功能化。例如:在SORT命令中，任选项〔D〕将指示SORT程序按降序对数据库进行分类。对于rDBMS任选项将在附录A中进行详细的描述。

第二节　命令行分析程序

命令行分析程序使用X≫提示用户，其中X是当前驱动器名（如A≫），这时表示录入的命令对A驱动器有效。所有rDBMS命令都使用〈C/R〉（回车键）进行录入。作为本rDBMS的命令都带有扩展名（.DBM）。用户在使用中只需录入命令名而不必录入扩展名。命令行分析程序可以识别和执行实用程序及带有扩展名（.COM）的应用程序。对于带有扩展名（.COM）的应用程序，使用时，须使用“$”后跟应用程序名（如$ED LEDGER.SUB及$PIP B:=A:CLIENTS.DAT）

注意：对于CP/M的内部命令如DIR、ERA、REN、TYPE及SAVE来说，命令分析程序不能对其识别。除了SAVE外，下列命令均与CP/M内部命令功能上相类似。即：

CP/M	rDBMS
DIR	DIR
ERA	DESTROY
REN	RENAME
TYPE	LIST

对于当前录入的命令，rDBMS命令分析程序首先检索当前的驱动器。如果找不到，就自动到主驱动器中检索同样的命令。若在命令行中指明了驱动器名和冒号（如C:LIST）或是在此之前打过SET AUTOSEL OFF，则对命令的检索仅限于当前驱动器。若不然，当AUTOSEL置为ON时用户就不必每次都指定驱动器名了。

第三节　rDBMS的内部命令

在rDBMS系统中，以DIR、DIC、LOG、RESTART、SET、SYS命令作为它的内部命令。它们在任何时刻都常驻内存中，可与外部命令一样用相同的方法录入和执行。但是，它们不能象外部命令一样直接通过HELP功能调用。这些内部命令除RESTART以外，均可放在一个批处理文件中，并且可以使用HELP功能进行调用。但是，如果你的批处理文件中包含RESTART命令，当这个批处理文件被激活时，就会自动地将批处理的工作文件名（$$.DBM）删除，所以使用时要特别慎重。

每当更换一张新磁盘时，须要打入LOGDISK或LOG DISK对新盘进行登记。LOGDISK的功能与CP/M中的磁盘复位功能键〈control-c〉是相同的（在rDBMS中不能使用〈control-c〉使磁盘重置）。若在更换新磁盘以后没有打入LOGDISK，则新换的磁盘是只读的。

DIC命令用来显示数据词典。使用时，打入DIC或DIC〔驱动器名:〕（如DIC B:）。如果在命令行中未指定驱动器名，则隐含地指定为当前的驱动器。

DIR可用来显示磁盘驱动器上所有文件的目录表。它也可以通过打入DIR〔驱动器名:〕filename来显示指定文件的目录。打入DIR〔驱动器名:〕则显示出指定驱动器上所有的磁盘文件。它还可以与通配字符“*”及“?”合用。

SET命令可以设置系统功能的状态。这些状态包括有ABORT ON/OFF、AUTOSEL

ON/OFF、MASTER〈drv:〉及ECHO ON/OFF。

打入SET AUTOSEL ON则使A驱动器作为主驱动器。那些在当前驱动器中找不到的命令都可以到主驱动器中查找。当设置SET MASTER 〈drv:〉后，AUTOSEL会自动设置为ON；一旦将AUTOSEL设置为OFF，则对命令的检索只限于当前驱动器，除非你在命令行上指明驱动器。

SYS或SYSTEM可以退出rDBMS环境，返回到CP/M环境，同时释放占用的内存空间。计算机会提示出（X>），控制权交给CP/M的CCP程序。此时你可以使用CP/M的一些命令和应用程序。

关于SET和RESTART命令的其他性质，将在以后的章节和附录A中进行讨论。

第四节　设计考虑

本rDBMS所给出的程序设计语言和其他的程序计设语言一样，也有一些规则，用户使用时，必须加以遵守。这样，才能确保你所设计的程序既满足rDBMS的要求又满足应用的要求。下面给出一些重要的命名约定、保留字及相应的数据项。

〈1〉文件名：

限制：长度不大于8个字符，并要求跟有三个字符的扩展名。

例如EMPLOYEE.FRM，这里文件名是EMPLOYEE，扩展名（.FRM）标识出这个文件是一个格式文件（formfile）。另外，本rDBMS承认的带其他扩展名的文件有

.DEF——表示数据定义文件（definition file）

.DAT——表示数据文件（data file）

.DIR——表示目录文件（directionary file）

.DIC——表示词典文件（dictionary file）

.HLP——表示辅助文件（help file）

.CMD——命令文件（command file）等

有些扩展名会自动地加在文件名的后边，例如FORMAT EMPLOYEE，执行结果会自动地创建一个叫做EMPLOYEE.FRM的格式文件。如果你完成了对一个数据库的定义后，会自动产生这个数据库的定义文件和数据文件，如EMPLOYEE.DEF和EMPLOYEE.DAT

〈2〉数据库名：

限制：长度不超过8个字符。如PURORDR作为一个购买订单数据库。

一个rDBMS的数据库名不是一个文件，因而不能使用扩展名。数据库名用来引用组成那个数据库的一系列文件，它们包括：格式文件（带扩展名 .FRM）

定义文件（带扩展名 .DEF）

数据文件（带扩展名 .DAT）

对于这些文件，将在第三章中详述。用户应选择一些便于记忆的名字作为自己的数据库名，如：PURODRDR——Purchase Order

GLEDGER——General Ledger

有些用户也常引用上述名字作为数据项名，这样选择时应格外小心。一般说来，用户

在选择数据库名及数据项名时应尽量地直观。例如：

SELECT EMPLOYEE WHERE DATEHIRE IS GT 01/01/81

（其中GT表示大于）

LIST GLEDGER BY ACCT DATE DEBIT CREDIT

〈3〉数据项名

限制：长度不大于十五个字符。如下面的情况：

①含有*、？、#、〈、〉或名中嵌有空格者。

②含有字符＇及″的字母数据项。

③含有+、－、*、/ 的数字、日期及$等类型的数据项。

在第①、③两种情况下，可能会引起识别错误，对于第②种情况，可能产生识别错误，在进行READ和WRITE操作时会发生错误。

例子：PROMDATE作为保证日期

一个数据项名中不应当含有空格，但可以写成NEW.DATE。若数据项名中包含有空格，要用引号“或’将其括起来，如“NEW DATE”或‘NEW DATE’，但决不能写成NEW DATE。

每个数据项有一个名字、数据定义和该数据项所占有的存储空间（为数据项保留的存储空间称为“数据字段”（field）），数据字段通过数据定义文件可以在输入屏幕上显示。数据项的屏幕表示和数据定义的例子说明如下：

屏幕显示：〔ADDRESS〕.....

数据定义：16个带短线的字符
AN字母/数字类型
16个字节的存储空间
最少有0个字符
最大可有16个字符
无缺省值

屏幕显示：〔CITY〕.....

数据定义：19个带短线的字符
A 字母类型
19个字节的存储空间
最小可有0个字符
最大可有19个字符
缺省值为“New York”

屏幕显示：〔COST〕.....

数据定义：9个带短线的字符（2个十进制）
$：美元类型（必须为十进制数）
4个字节的存储空间
最小值为 $0.00
最大值为 $999999.99
无缺省值

屏幕显示：〔PROMDATE〕

数据定义：8个带短线的字符

J：日期类型（格式为mm/dd/yy或mm-dd-yy或mmddyy）

3个字节的存储空间

使用DATE命令指定$TODAY缺省值

屏幕格式：〔QUANTITY〕

数据定义：6个带短线的数字

N：数字类型

3个字节的存储空间

最小值为-99999

最大值为99999

无缺省值

注意：在构成一个输入屏幕时，数据项名必须使用方括号〔 〕，而它作为录入数据进行显示时，并不显示出括号。

〈4〉数据项值：

限制：对含有*、？、#、〈、〉或嵌有空格的字母或字母数字类数据项值可能引起错误。如果数据项值需要嵌含空格，则必须使用引号'或"括起来。字母或字母数字类数据项的值中不能含有'或"，否则，会在READ和WRITE命令执行时，产生错误。例子：某一数据项的值可以为chicago

〈5〉字段的匹配：

在命令语句或对数据项的引用中，有三个可以同时调用两个数据库的命令，它们是COMPARE、JOIN和POST命令。这些命令可比较或改变数据项值。要注意：在设计中要确保比较或传送的项具有相同的名字、大小和数据类型。

例如：在购货订单文件中，有一个数据项记录收到的数量。这个数据项名叫做RECEIVED，在该数据项的数据定义中占有三个字节。并且，你可以另外再建立一个文件用来收集日常的购货订单收据。而你可以在需要的时候将它们整理并传送到购货订单文件中，因此，具有三个字节的数值数据定义的项也应当在收据文件中出现，并用来记录每个收据中的数量。

〈6〉保留字：

本rDBMS命令语句中，有些词具有专门的意义。例如ADD在POST命令中指定为加法运算。如果用ADD作为一个数据项名，则命令行分析程序可能会曲解这个名字，并相应地执行错误操作。为此，对作为保留字而命名的rDBMS的词在使用时要加以限制。这也就是说，数据项名不应该是保留字。

保留字索引：RESERVED WORDS

@ (COMPUTE)	GT (Greater Than)	STAX
ADD	IS	ST (Such That)
AND	LE (Less than or Equal)	SUBTOTAL
ARE		SUBTOT (SUBTOTAL)
AVERAGE	LT (Less Than)	SUBTRACT

AVE	MATCHING	SUB (SUBtract)
AVG	MAX	SUM
BY	MIN	TOTAL
COMPUTE	NE (Not Equal)	TOT (TOTal)
COMP	NOT	USING
EQ (EQual)	OR	WHERE
EQUAL	REPLACE	WHOSE
GE(Greater than or Equal)		REP(REPlace)

第五节 CRT 控制键

以下列出的控制键具有一些特殊的功能：

1. <control-A>：

在FORMAT 程序中作为替换方式 (replace mode) 和插入方式 (insert mode) 之间的转换开关

2. <control-C>：

中止当前的操作

3. <control-O>：

在ENTER中，作为常规方式和自动缺省方式之间的转换开关

在FORMAT中，在插入方式下，用来删除字符

4. <control-E>：

在FORMAT中，用来存储并退出屏幕格式

在UPDATE中，用来结束修改方式，它不需移动光标就可结束记录

5. <control-O>：

在ENTER 中，对进入的数据项进行最大值到最小值的范围检查。用来取代编辑标准。如果进入的数据项与字段的类型说明不一致，则不作这种取代

6. <control-P>：

启、停系统打印机的输出，并在打印机上产生CRT的硬拷贝。这里要注意：在使用ENTER和UPDATE命令时，不应当使用<control-P>

7. <control-Q>：

继续完成以前由<control-S>中断的输出

8. <control-R>：

在FORMAT中，若不慎按了清除键，则可以使用<control-R> 来恢复原来的屏幕

在ENTER中，它是常规方式和自动重复方式之间的转换开关

9. <control-S>：

暂时中断CRT或打印机上的输出直至进入一个<control-Q>

10. <control-X>：

在FORMAT中，用来删除光标所在的行

11. RETURN(<C/R>)：

回车键，用来结束一个命令句；在FORMAT的插入方式中，可以插入一个空行

第六节　光标移动控制键

1．RETURN(〈C/R〉) ↵：

回车换行键，用它结束数据的录入，同时光标指向下一个字段（或下一个项）的位置

2．〈control-H〉 ←：

向“左”或往“回”移动光标，当录入数据时，把光标移到前一个数据项的位置上

3．〈control-J〉 ↓：

向下移动光标，当录入数据时，将光标移到下一项的位置上

4．〈control-K〉 ↑：

向“上”移动光标。当录入数据时，将光标移到前一项的位置上

5．〈control-L〉 →：

向“前”或向“右”移动光标，当录入数据时，光标移到下一项的位置上

第七节　rDBMS的规定

①记录个数：最大32767个记录/文件
②记录长度：可以是2～1024字节/记录
　　且每个记录的第一个字节自动作为状态字节
③字段长度　1～127个字段/记录
④对A或AN类数据项：
　　1～127个字节/字段
⑤对$或N类数据项：
　　1～10位数字（用1～4个字节的整数存储）

第八节　数据类型

AN——字母数字类：字母可以是A～Z、a～z，数字可以是0～9
A——字母类：字母可以是A～Z、a～z，′，．，－
N——数值类：值域为±2148373647的整数
$——美元类：值域为±$21483736.47（十进制）
J——日期类：格式为mm/dd/yy，便于输出和输入
R——要求的登记项，可用于以上任何一种类型。

第九节　系统要求

Z-80——微型机：至少有48K-RAM内存（或除操作系统外有大于42K的内存空间）

CRT终端——可进行光标编址，屏幕清除及行可滚动的80×24屏幕

如：

(Adds、Beehive、Hazeltine、Infoton、Lear-Siegler、V Perhin-Elmer、Soroc、TRS80—mode—Ⅱ)

操作系统——CP/M：要求1.4版本及以后版本

MP/M：要求1.1版本及以后版本

打印机——要求具有走纸和纸宽度控制功能

驱动器——软盘驱动器（或）和硬盘驱动器。

第十节　应用程序的开发和操作

使用rDBMS命令语言语句开发一个数据库的步骤如下：

〈1〉设计数据输入的屏幕格式；

〈2〉定义屏幕格式上每个数据项的数据特性；

〈3〉在磁盘上定义数据文件；

〈4〉将数据库标题加到数据库词典中去。

以上的步骤是通过DEFINE命令实现的，其中步骤〈3〉，〈4〉是自动完成的。在开发了一个数据库以后，应用已就绪，随即可进行数据录入和报表处理。

开发和操作过程步骤如下：

启动	第二章
数据库开发	第三章
数据库输入	第四章
辅助屏幕和命令过程	第五章
关系操作	第六章
查询与报表书写功能	第七章
与非rDBMS程序的接口	第八章
FORMAT命令细则	第九章
DEFINE命令细则	第十章
数据库重组织细则	第十一章
RUN命令处理程序细则	第十二章
实用程序	第十三章

第二章　启　　动

对rDBMS的启动，首先要求把含有CP/M操作系统和所有SERIES/ 20rDBMS命令的文件放在一张磁盘上。在你获得这个磁盘后，就可以利用CP/M的批处理命令SUBMIT.COM去执行一个启动程序START.SUB。下面将逐一介绍这个过程。

本章所用到的rDBMS命令有：

DATE　登记当前日期

DBMS　装入CONDOR SERIES /20rDBMS系统

TERM　标识计算机终端类型

第一节　建立主系统磁盘

首先将计算机、CRT显示终端和打印机的电源打开

第一步：

建立一个rDBMS的主系统，你需要把原始磁盘保存好。原始盘只用来创建主系统，在新盘上标注好主系统代号，记录下系列号和有关的版权信息。

第二步：

建立一个主系统磁盘的工作拷贝盘。是为在用户开发中使用。主系统盘只用来建立它的工作拷贝。下面将描述这些过程。

在创建了一个工作拷贝之后，rDBMS 命令和文件都被传送到 CP/M 磁盘上。这时可以开发用户的应用程序了。以下我们采用操作动作和系统响应注解的形式说明以上所述的操作步骤。

1. 打开计算机、磁盘驱动器和 CRT 开关
2. 在A驱动器中插入CP/M磁盘
   a. 要确保你的计算机系统大于48K，否则要使用 MOVCPM 命令（见“CP/M 参考手册”）
3. 在 B 驱动器中插入rDBMS磁盘
   a. 该盘应当是rDBMS系统原始盘
4. 引导：打入BootuP〈C/R〉
   a. 遵循你的计算机引导过程
   b. 系统用提示符“A>”进行响应
5. 打入PIP〈C/R〉
   a. 把磁盘文件传送程序装入内存
   b. 系统用提示符“*”进行响应
6. 打入A:=B:*.*〈C/R〉
   a. rDBMS的全部命令都被传送到CP/M磁盘上，系统用提示符“*”响应
7. 打入〈C/R〉

a．完成PIP过程

b．系统用提示符“A>”进行响应

8．完善拷贝

a．带rDBMS全部命令的CP/M盘已经在驱动器A上建成

标记该盘作为主系统，记录下顺序号和相应的版权信息。从现在起，可用该盘生成一个工作拷贝盘

9．从B驱动器上取出rDBMS原始盘，并保存好

第二节　建立工作拷贝盘

1．打开计算机、磁盘驱动器和CRT电源开关

2．在A驱动器上插入主系统盘

a．要确保你的CP/M大于48K，否则须使用MOVCPM命令（见CP/M参考手册）

3．在B驱动器上插入空盘

4．引导、打入BootuP〈C/P〉

a．遵循你的计算机的引导过程

b．系统用提示符“A>”进行响应

5．打入SYSGEN〈C/R〉

a．源盘用A来响应，目的盘用B来响应

6．打入PIP〈C/R〉

a．将磁盘文件传送程序装到内存中

b．系统用提示符“*”响应

7．打入 B:=A:*.*〈C/R〉

a．将磁盘A中所有的rDBMS命令都传送到CP/M磁盘上，系统用提示符“*”响应

8．打入〈C/R〉

a．完成PIP工作过程

b．系统用提示符“A>”响应

9．传送完成

a．在B驱动器上建立了工作盘

10．将工作拷贝盘从B驱动器中取出。标记下磁盘的顺序号和版权信息，现在，你可以使用工作盘来开发自己的rDBMS应用程序

第三节　启动CONDOR SERIES/20 rDBMS

3.1　使用批处理操作启动rDBMS系统

1．将计算机、磁盘驱动器、CRT和打印机的电源开关打开

2．在A驱动器上插入rDBMS工作盘然后引导

3．打入SUBMIT START〈C/R〉

a．系统用版权号及许可证号信息响应

b. 系统向你要许可证号 (License Number)

4. 打入许可证号然后打入〈C/R〉

a. 计算机要求你进入当前日期

5. 用mm/dd/yy格式打入当前日期，然后转入〈C/R〉

6. 选择如下之一的终端标识，然后打入〈C/R〉，这些终端标识是：

ADDS　　BEEH　　HAZL　　INFO

LEAR　　PERK　　SORC　　TRS2　　MISC

a. 当你选择了一个正确的终端标识和参数时，计算机屏幕上会在四个角上出现星号“*”，在屏幕中央出现READY。若不然，说明你选择的终端标识或参数不对，究竟应该选择那种终端标识及哪些参数，要看你的计算机终端说明来定（详见TERM命令和附录B）

3.2 用非批处理操作来启动rDBMS系统

前面所述的启动过程是使用CP/M的SUBMIT来完成的。以下的过程说明了使用非批处理方式怎样启动rDBMS系统。这个过程只说明文件START中所用到的命令。

1. 在A驱动器上装入rDBMS磁盘并引导

2. 打入DBMS〈C/R〉

a. 系统显示启动信息、版权信息及卖主地址

b. 系统向你要许可证号

3. 进入许可证号，然后打〈C/R〉

4. 打入DATE〈C/R〉

a. 系统显示日期及请求你录入新的日期信息

5. 以mm/dd/yy格式录入当前日期

6. 打入TERM〈C/R〉

a. 系统要求给出终端标识ID

7. 选择一个合适的终端标识

ADDS　　BEEH　　HAZL　　INFO

LEAR　　PFRK　　SORC　　TRS2　　MISC

然后打入〈C/R〉

a. 当你选择了一个正确的终端标识以后，会在CRT屏幕的四角上出现星号“*”并在CRT屏幕中央出现一个READY。若不然，说明你选择的标识或终端参数不对。究竟应该选择哪种终端标识及哪些终端参数，要看你的计算机说明来定（详见TERM命令和附录B）

第四节　rDBMS练习

如果你有了一个工作拷贝盘，并且完成了对rDBMS系统的启动初始化，你就可以创建一个rDBMS数据库了。

下面的示范例子主要针对首次使用rDBMS数据库语言的用户，并使其尽快地熟悉创建一个rDBMS数据库文件的过程。顺便熟悉一下其他有用的rDBMS命令的功能和特性。

第一步：定义一个数据库

1. 打入DEFINE BIRTHDAY〈C/R〉

系统询问你是否要创建一个新格式。

2. 打入Y

CRT上端出现空屏幕，而在屏幕底端有一些任选信息。此时你可以建立格式文件。

3. 打入〔NAME〕_____〔BIRTHDATE〕_____打入16个短线用来存放名字，打入8个短线用来存放生日。

4. 打入〈control-E〉

该控制键结束格式文件的建立，并将文件BIRTHDAY.FRM存储在当前磁盘上。此时系统询问你是否要求定义该数据库。

5. 打入Y

系统显示"＞1. NAME："，光标指在冒号之后，系统等待对NAME数据项的定义。

6. 打入AN〈C/R〉

表示指定数据项(data-item)是字母类型的，接着系统显示"＞2. BIRTHDATE："，光标指在冒号之后，系统等待对BIRTHDATE 数据项的定义。

7. 打入J〈C/R〉

指定该数据项为Julian（儒略）日期类型。随后系统要求你检查数据定义是否成功(OK)。

8. 打入Y

系统显示出"Busy"，并自动创建定义文件和数据文件，此后系统问你是否需要一份数据定义的硬拷贝。

9. 打入Y

在打印机上打印出数据库中各个数据项及其记录长度。然后，可以进行数据录入。

第二步：录入数据

10.打入ENTER BIRTHDAY〈C/R〉

在CRT上将显示出BIRTHDAY数据库的格式，此时光标指向第一个数据项的位置。打入姓名，然后按下〈C/R〉；以mm/dd/yy的格式打入日期（生日）。日期数据不一定非要包含8个字符（如1/2/45)。如少于8个字符，则要打入一个〈C/R〉完成日期的录入。此时系统出现提示。如你要继续录入下一个人的姓名和生日，就打入C，若要结束数据录入，就打入E。

第三步：rDBMS另外一些命令的使用

11. 打入SORT BIRTHDAY BY BIRTHDATE〈C/R〉

系统按指定的数据项进行分类，分类后数据库的记录按照日期的升序排列。

12. 打入LIST BIRTHDAY BY NAME BIRTHDATE〈C/R〉

在CRT上显示出排序后的数据列表。

13. 打入PRINT BIRTHDAY BY NAME BIRTHDATE〈C/R〉

在打印机上产生一份数据库数据的硬拷贝。

到此为止，你对如何建立一个CONDOR SERIES /20rDBMS数据库有了一个初步的了解，在下面的各章里，将详细讨论rDBMS的其他特性。

第三章　开发 rDBMS 数据库

本章将总结与 rDBMS 数据库有关的文件建立及开发过程。一个数据库具有以下三种文件：

1. 屏幕格式文件（.FRM）——
   指定数据项名，并为数据输入输出提供格式。
2. 数据项定义文件（.DEF）——
   内有对数据项的定义。
3. 数据定义文件（.DAT）——
   内有用户数据。

定义以上三个文件是为了以后的数据录入，数据存贮和数据访问的应用。一个简单的数据库文件访问描述将帮助你理解开发过程。

举例：LIST EMPLOYEE BY NAME DEPTMENT SENTORITY

此命令行引用EMPLOYEE数据库名。该数据库名在数据词典文件(DATA.DIC)中用一个记录进行登记。它登记数据库文件的名字，即.FRM、.DEF和.DAT文件。LIST命令访问这些文件，然后再执行之。

第一节　定义rDBMS 数据库

定一个数据库要分两步走：

第一步：创建屏幕格式和数据项名文件（.FRM）；

第二步：创建数据项定义文件（.DEF）、数据文件（.DAT），并将数据库名登记到数据词典(DATA.DIC)中。在数据词典中，指定与每个数据库有关的.FRM、.DEF和.DAT文件。

本章用到的rDBMS命令：

DEFINE：用户用来定义屏幕格式并创建数据定义（.DEF）文件，为用户数据文件（.DAT）分配磁盘空间。更新数据词典（DATA.DIC）

FORMAT：用户用来在CRT上编辑格式文件

LIST：在CRT上显示数据库记录

PRINT：打印数据库记录

第二节　开发 rDBMS 数据库实例

1. 确定数据库名

   a. 名字要小于或等于八个字符。不带扩展名。

   b. 用一个简单分类总账系统描述rDBMS的特性。这个应用系统由一个分类总账数据库、日记总账数据库和日帐审理账底数据库组成。以下是分配给这三个数据库的名字：

〈1〉 GLEDGER——分类总账数据库

〈2〉 JOURNAL——日记总账数据库

〈3〉 JOURADT——日记账底审理数据库

本章描述这些数据库的建立，后面的章节描述数据录入这些数据库及产生记帐报表。

2. 决定在哪个驱动器上建立数据库

a. 驱动器A上有操作系统和rDBMS命令。由于磁盘空间的限制，最好在B驱动器上放一个空盘，置驱动器的格式为：〈驱动器名〉：〈C/R〉如B：〈C/R〉。

b. 在总账会计系统例子中，把数据库放在B驱动器上，在定义数据库之前，打入B：〈C/R〉。

第三节 定义总分类账数据库

3. 打入DEFINE GLEDGER 〈C/R〉

a. 系统问你是否要创建一个新格式，打入Y，转入4。

b. 系统为开发新格式给出一个空屏幕。以下是开发GLEDGER屏幕格式的例子。

```
GENERAL LEDGER ACCOUNTING SYSTEM

GENERAL LEDGER RECORD
[account]..... [assist] .....
[name] ..... [liability] .....
[debit] ..... [credit] .....
[ytdmat] ..... (year-to-date amount)
[month] ..... (last accounting month posted to generel ledger)
```

4. 设计格式

a. 光标可以在屏幕上随意移动，你可以随意设置标题和“数据项”，你必须用〔 〕把“数据项”括起来，其后跟有若干条短线。短线的个数是你所要进入字符最大值个数（对日期类数据项要求8个字符位置）。例如数据项〔NAME〕.....

b. 所有的数据项必须有精确的短线个数。这个短线要求不能中断。例如：不能出现〔AMOUNT〕.....，.....，.....这样的形式。但允许出现〔AMOUNT〕S这样的格式。

5. 打入〈control-E〉

a. 本命令自动地把屏幕格式内容记入数据库的.FRM文件。此时如果打了〈control-C〉，则系统将不再存储建立的格式信息。如果你不想保存这个信息，可以使用该控制键。

b. 系统询问你是否要定义一个新的数据库，如果你使用FORMAT来编辑（修订）格式，那么不进行此项询问。本例中，GLEDGER是一个新的数据格式。

6. 打入Y

a．系统自动地提示你定义格式信息上括弧中的数据项。数据项按其在屏幕格式上的次序（从左至右、从上至下）顺序出现。

7．打入数据类型代理字符，后跟一个〈C/R〉，数据类型简写如下（定义细则见附录C）

a．数据类型简写为：

AN 字母数字型

A 字母型

N 数值型

S 美元型

J 日期型

R 可以是以上类型之一

b．系统指定数据值大小，最小值、最大值和缺省值或改变所赋的最大值、最小值。详见第十章。

c．从下表中可以决定一个字段所占存储空间的大小：

A 或 AN 类　　1～127个字节/字段

J 类：　　3 个字节/字段

存贮大小	值域（N型/数）	值域（S型/美元）
字节	±127	±1.27
字节	±32767	±327.67
字节	±8388607	±83886.07
字节	±2147483647	±21474836.47

d．当给出所有的数据定义时，出现《END》，系统要求一个确认 OK 来肯定数据定义。

8．如果数据项定义正确无误，则打入 Y；否则打入 N。参考第 c 步

a．当打入 Y 时，系统显示出 Busy，并自动创建数据定义文件和数据文件。同时将数据库名写到数据词典中去。

b．系统询问你是否需要定义的硬拷贝。打入 Y 则获得之，然后转到 d。

c．若打入 N，系统将提示你修改数据项定义。具体修改细则见第十章。

d．如果此时你想改变数据的定义，可使用 FORMAT 或 DEFINE 命令。其中屏幕格式的修改用 FORMAT 命令改变，数据项定义用 DEFINE 命令的重定义任选改变。

9．打入 ENTER GLEDGER〈C/R〉，录入试验数据

a．设计的结构在屏幕上显示出来。你现在可以进入试验数据了。可以先录入不正确的数据，例如在要求数值的项中写入字母符号，则产生错误，系统会提示你。

b．当数据录入完毕时，按提示的要求，打入 E 结束。

10．打入 LIST GLEDGER〈C/R〉

a．系统显示格式和要测试的数据。

11．打入PRINT GLEDGER, FRM〈C/R〉。

a．在打印机上打印出格式文件。

第四节　定义日记账数据库

12. 打入 DEFINE JOURNAL〈C/R〉

a. 按照前面步骤（3～11）创建 JOURNAL 数据库，并使 JOURNAL 数据库具有如下的格式文件。

b. 为测试和形成文件，在第 9～11步中将 GLEDGER 引用变成 JOURNAL引用。

JOURNAL 数据库格式

```
        GENERAL  LEDGER  ACCOUNTING  SYSTEM
    JOURNAL  RECORD
[account] .....
[desc] .....
[debit] .....[credit] .....
[month] ..... (Accounting Month)
[date] ..... (Date of JOURNAL entry)
```

第五节　定义日账底审理数据库

13. 打入 DEFINE JOURAUDT 〈C/R〉

a. 用和 JOURNAL 一样的格式、数据定义建立 JOURAUDT 数据库。

b. 系统提示没找到名为 JOURAUDT·FRM 的文件，并询问你是否要建立一个新的格式文件，打入 N。

c. 系统请你进入一个已经存在的（.FRM）文件名。

14. 打入 JOURNAL 〈C/R〉

a. 共享 JOURNAL 格式，系统询问你是否要创建一个新的定义文件。

15. 打入 N

a. 共享某个定义文件，系统请求输入已经存在的.DEF文件名。

16. 打入 JOURNAL 〈C/R〉

a. 到此为止，JOURDUT 数据库已经创建完毕，系统出现提示符响应。

b. 为了测试和形成文件，在以上 9～11 步中将对 GLEDGER 的引用变成对 JOURADUT 的引用。

到此为止、分类账会计系统数据库已经形成。在下一章中，将详细描述：

1. 将帐目表录入 GLEDGER 数据库。
2. 将日记账登记录入 JOURNAL 数据库。
3. 将日记账登记项传送到总分类账中。
4. 为审理账底累计日记账登记项。

第四章　数据库输入指南

本章描述三种数据的输入过程。即：

（1）录入新记录；

（2）录入匹配的记录；

（3）传送记录。

因此，本章主要致力于两个目标。即：

（1）学习 rDBMS 命令的使用；

（2）学习计算机的操作过程。

信息的输入可定义成把外界信息带给计算机，计算机经过某些处理将它们存入数据库中。例如：在第三章中，把有关总分类帐会计信息从一个 CRT 终端上通过键盘送入计算机，并以数据库文件的形式存储在磁盘上。注意：输入既包括新数据的输入也包括对已存在的数据项的修改输入。输入过程有时可能受到限制。例如：发放工资系统对每个职工只能有一份工资，如果输入那个职工的另一个记录，就要消去前者。

以下是把唯一的记录或匹配（matching）的记录以及传送记录输入到数据库中去的输入过程。

Ⅰ．录入“唯一的记录”（通常称为主记录）

当把新记录录入到一个现存的数据库时，不允许出现重复的记录。例如，在总分类帐会计记录中，不允许出现两个相同帐号的记录（特殊的应用将强调这种限制，但它在 rDBMS 中不是一个一般的限制）。

Ⅱ．录入“匹配记录”（通常称为事务记录）

当把新记录录入到一个“事务”数据库时，在“主”数据库中，一定含有一个“匹配的记录”。例如：把一个客户的支付账号录入到一个接收账目的数据库中，该账号一定与总分类账数据库中某个帐号相匹配。

Ⅲ．传送（通常把事务记录传入主记录中）

把一个数据库上的新记录传送到另一数据库的记录上（通常称为更新记录）。传送的数据库（主数据库）有唯一的记录，而被传的数据库（事务数据库）可能有重复的记录。然而限制它要与主数据库的记录相匹配。一个实例是更新购货订单主数据库的购货订单收据事务。

本章所用到的 rDBMS 命令有：

1. APPEND　把一个数据库的记录添加到另一个数据库上
2. COMPARE　按指定的数据项匹配数据库Ⅰ和数据库Ⅱ的记录，并创建数据库Ⅰ的结果数据库(其中的记录是不匹配数据库Ⅱ的)
3. COMPUTE　计算一个数据库记录中指定的项，并把结果存入数据库记录中另一个指定的项
4. EMPTY　删除数据库中的所有记录
5. ENTER　把新的记录写入到数据库中

6. POST　通过指定的项找出数据库Ⅰ和数据Ⅱ匹配的记录，对指定的匹配的数据项，用数据库Ⅱ的记录去更新数据库Ⅰ的记录。并建立数据库Ⅰ记录的结果数据库。其中的记录与数据库Ⅱ的记录相匹配

7. PRINT　按照指定的项在系统打印机上打印数据库

8. SELECT　选择满足给定条件的数据库记录，并产生一个结果数据库

9. SORT　按照给定的数据项对数据库进行分类

10. STAX　对指定的项作统计计算

11. TITLE　打印报表标题

12. UPDATE　更新数据库的某些记录

第一节　结果数据库

命令COMPARE、POST、SELECT在当前的驱动器上产生一个结果数据库(RESULT)文件，如当前的驱动器名为B，则结果集文件$$.DEF、$$.FRM、$$.DAT就建立在B驱动器上。为了检索这些文件，一个称为结果的数据库被记录到B驱动器上的数据词典文件（DATA.DIC）上一次。即：

打入B：〈C/R〉

打入ENTER DATA.DIC 〈C/R〉

首先将DATA.DIC的输入格式显示在屏幕上，打入RESULT对应TITLE（词典文件中标题），打入$$对应词典文件中的定义文件

打入$$对应词典文件中的格式文件

打入$$对应词典文件中的数据文件

打入〈control-E〉结束进入数据程序，存储结果集文件。

第二节　录入唯一记录的处理

举例：分类总帐会计表的输入

在第三章中定义的分类总帐数据库GLEDGER，对每个分类帐号在“主”数据库中录入一个记录。如果同一个帐号两次录入数据库则出现错误。为了消除这种错误、必须开发一种确保数据库有唯一记录的数据录入过程。下面将描述这样一个过程。

为了确保要求有唯一记录的数据库只录入唯一的记录，则录入的记录不应直接录入到主数据库中，而是录入到一个“中间数据库”中；在完成对中间数据库记录的录入以后，将主数据库的记录与中间数据库的记录作比较，只将唯一的记录添加到主数据库中。如果每次完成数据录入后将中间数据库置空的话，则中间数据库只需定义一次。

使用rDBMS命令录入唯一记录的标准过程如下：

1. 改变驱动器，打入B：〈C/R〉
   将当前驱动器置成B。

2. DEFINE：打入DEFINE GLTEMP 〈C/R〉

系统询问你是否需要建立一个新的格式文件。打入N时，系统请求进入一个格式文件名。本例中，中间数据库是为GLEDGER而设置的，所以GLEDGER.FRM是要进入的文件名。打入GLEDGER〈C/R〉。而后，系统询问你是否须要创建一个新的定义文件。打入N时，系统请求进入一个定义文件名。本例中，GLEDGER.DEF是要用到的定义文件，故打入GLEDGER〈C/R〉。此时系统显示出Busy，并自动分配数据文件。当分配完成后，在屏幕上出现提示符“A≫”，指出已定义好中间数据库GLTEMP，可以进行数据录入工作了。

3. ENTER：打入ENTER GLTEMP〈C/R〉

① 当录入总分类帐号时，系统用GLEDGER.FRM输入格式来响应。打入新的帐号。

② 如果每个数据项所录入字符的个数等于那个数据项名后所分配短线的个数，光标自动地跳到下一项的位置上。如果录入字符的个数不能填满短线的位置，打入〈C/R〉可将光标移到下一项上。

③ 为了录入缺省值，在一个数据项的开始位置上打入〈C/R〉。此时如果原来定义了缺省值，则将它录入，光标移到下一个位置。

④ 如果你希望自动地录入缺省值，则既可以在打入ENTER GLTEMP〈C/R〉后打入〈control-D〉，又可以在命令行中使用〔D〕任选项（在这种情况下，可使用〈control-D〉作为常规方式和自动缺省值方式的转换开关）。

⑤ 自动重复方式选择后不清除以前录入的记录值，在录入期间，打入一个〈control-R〉（或在命令行中使用〔R〕任选）则置成自动重复方式。此时，再打入〈control-R〉，自动重复方式复位，回到常规方式。

⑥ 自动重复方式和自动缺省方式是互斥的。使用〈control-R〉控制键，可以作为自动缺省方式和自动重复方式的转换开关。

⑦ 自动重复方式和自动缺省方式用屏幕左下角的信息指出。

⑧ 当录入完一个记录的全部项时，系统给出提示，请你选择下一步的动作。如：

Abort(A), Continue(C), Delete(D), End(E), Print(P), Revise(R)

打入C录入另一购货订单，系统用一新的GLEDGER.FRM输入格式响应。步骤如前。

打入E结束帐目号数据录入，系统用一个提示符“X≫”响应。

4. UPDATE：如果要修改已存入的主记录文件GLEDGER中的帐号记录，就必须使用UPDATE命令。修改处理如下：

① 打入UPDATE GLEDER 〈C/R〉

系统要求进入检索条件。

② 打入ACCOUNT IS ×××××〈C/R〉

其中×××××代表帐号。

如果系统找到该记录，就把它显示在屏幕上，打入R任选，并将光标移到要修改的位置上。打入新值，然后打入〈control-E〉结束。根据选择打入C，继续更新下一个记录。

③ 为了结束UPPATE过程，在要求进入检索条件时，单纯进入一个〈C/R〉即可。

5. COMPARE：打入

COMPARE GLTEMP GLEDGER NOT MATCHING ACCOUNT〈C/R〉

① 按照给定的数据项ACCOUNT，对GLTEMP与GLEDGER数据库进行比较。找出新的ACCOUNT号，不与GLEDGER匹配的GLTEMP记录ACCOUNT被写到B驱动器上的RESULT数据库中，结果文件中的格式与数据项名和GLEDER·FRM的相同。

6. TITLE：打入

TITLE T,'GENERAL LEDGER REPORTS',S,S,'NEW ACCOUNT NUMBER ADDITIONS FOR',DATE,S,S,〈C/R〉

① 以上的标题（用引号括起来的部分）和日期在NEW ACCOUNT NUMBER ADDITIONS报表的每一页上印出。标题是用两行印出。命令行中的S表示跳行。

7. PRINT：打入

PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT〈C/R〉

"唯一"的新帐号（与主文件帐号不匹配的帐号）在系统打印机上被印出。ACCOUNT是描述帐目名称的数据项。

8. APPEND：打入APPEND GLEDGER RESULT〈C/R〉

新的"唯一"的分类总帐记录号都被添加到主文件GLEDGER中。该命令实际上是把SS.DAT文件添加到GLEDGER.DAT文件上。

9. SORT：打入SORT GLEDGER BY ACCOUNT〈C/R〉

按帐号顺序（升序或降序）对GLEDGER数据库进行分类。

10. TITLE：打入

TITLE T,'GENERAL LEDGER REPORTS',S,S,'GENERAL LEDGER CHART OF ACCOUNTS DATE',DATE,S,S〈C/R〉

在GENERAL LEDGER CHART OF ACCOUNT报表的每页上都印出以上的标题和日期。

11. PRINT：打入

PRINT GLEDGER BY ACCOUNT ACCNAME DEBIT CREDIT〈C/R〉

在系统打印机上打印GENERAL LEDGER CHART OF ACCOUNT极表。

12. EMPTY：打入EMPTY GLTEMP OK 〈C/R〉

删去GLTEMP数据库中的记录。

第三节 录入匹配记录的处理

例子：日记总帐输入

在第三章开发的日记总帐数据库JOURNAL的日帐登记项必须与分类总帐数据库GLEDGER中记录的帐号匹配，如果不匹配，则是错误的。

在向GLEDGER数据库送数据之前，要将JOURNAL的记录与GLEDGER的记录进行比较，核实帐号是否匹配。若发现有帐号不匹配的记录时，需要在传送之前进行修改。

用rDBMS命令录入匹配记录的标准过程为：

1. 改变驱动器。打入B:〈C/R〉

把当前驱动器置成B

2. ENTER：打入ENTER JOURNAL〈C/R〉

当录入日记帐事务时，系统将用JOURNAL.FRM输入格式响应。请录入新日帐事务。

3. SORT：打入SORT JOURNAL BY ACCOUNT〈C/R〉

系统按照帐号顺序对JOURNAL数据库分类，分类操作既不是必须的也不是为使用该命令而进行的。但是，如果文件经过分类排序之后，响应时间可以比原来按随机存放的改善90%。

打入SORT GLEDGER BY ACCOUNT〈C/R〉后，系统将按帐号顺序对GLEDGER数据库进行分类。

4. COMPARE：打入

COMPARE JOURNAL GLEDGER NOT MATCHING ACCOUNT〈C/R〉

通过对ACCOUNT帐号的匹配，使JOURNAL记录与GLEDGER记录作比较。不匹配的JOURNAL记录被写入到RESULT集中。RESULT数据库的格式及数据项与JOURNAL.FRM一致。

5. TITLE：打入

TITLE T,'GENERAL JOURNAL REPORTS',S, 'JOURNAL EXCEPTIONS FOR', DATE, S〈C/R〉

以上日期和标题将在JOURNAL EXECPTIONS报表的每一页上印出。

6. PRINT：打入

PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT〈C/R〉

在打印机上打印出不匹配GLEDGER数据库记录的日帐记录。确定问题所在，并根据要求或更新GLEDGER数据库或更新JOURNAL数据库。如果需作更新的话，你必须重新使用SORT命令进行分类。当打印出不匹配的记录为空时，则匹配工作完成。也就是说，JOURNAL数据库的记录与GLEDGER数据库的记录全部匹配。此时可做下一步工作，即使用POST命令将JOURNAL传送到GLEDGER数据库中去。

第四节 传 送

例子：把日帐登记传送到分类总帐中

在分类总帐会计系统例子中，应该预算一个月的日帐登记项，而到月末，把日帐登记项汇总到分类总帐中。在每次传送后，把日帐登记项累记在一个帐底审理数据库中，然后把JOURNAL数据库置空，准备在下一个月录入新的登记项。

用rDBMS命令传送记录的标准过程如下所示：

1. 改变驱动器，打入B：〈C/R〉

把当前驱动器置成B。

2. STAX：打入

STAX JOURNAL BY DEBIT, CREDIT[T]〈C/R〉

作DEBIT和CREDIT的统计计算，即对JOURNAL数据库中DEBIT和CREDIT作统计。为得到“收支平衡”，一定要使总支出等于总收入。如总计不相等，一定要找到收支错误，并打入UPDATE JOURNAL命令及查找条件，例如给出 WHERE ACCOUNT IS×××××××〈C/R〉（参考以前描述的UPDATE GLEDGER过程）。

3. ROST：打入

POST GLEDGER JOURNAL BY ACCOUNT AND ADD DEBIT, CREDIT REPLACE MONTH〈C/R〉

对于每对匹配的ACCOUNT帐号，把JOURNAL数据库中的DEBIT、CREDIT总量加到GLEDGER数据库中DEBIT、CREDIT总量上。并且GLEDGER数据库中的MONTH由JOURNAL数据库中的MONTH替换。

4. COMPUTE：打入

COMPUTE GLEDGER ST YTDAMT = DEBIT - CREDIT〈C/R〉

算出YTDAMT年终总量。

5. SELECT：打入SELECT GLEDGER ST ASSET IS Y〈C/R〉

选出是资产帐目的分类总帐记录，并将它们写到RESULT数据库中。其格式与GLEDGER.FRM一样。

6. TITLE：打入

TITLE T,'GENERAL LEDGER MONTHLY REPORTS',S
'ASSET ACCOUNTS REPORT FOR',DATE,S〈C/R〉

以上标题和日期都在ASSET ACCOUNTS REPORT报表的每一页上打印出来。

7. PRINT：打入

PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT AND COMPUTE TOTAL YTDAMT〈C/R〉

资产分类帐总记录带有说明，收支总量和年终总量一起在打印机上打出。在报表上最后统计年终总量。下一步做总计与义务帐的年终总量的比较。

8. SELECT：打入SELECT GLEDGER ST LIABILITIES IS Y〈C/R〉

在分类帐记录中，将义务帐写入RESULT数据库。其格式与GLEDGER.FRM相同。

9. TITLE：打入

TITLE'GENERAL LEDGER MONTHLY REPORTS',S,
'LIABILITIES ACCOUNTS REPORT FOR',DATE,S〈C/R〉

在LIABILITY ACCOUNTS REPORT报表上的每一页上都打印出标题及日期。

10. PRINT：打入

PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT
AND COMPUTE TOTAL YTDAMT〈C/R〉

分类总帐中的义务记录及其描述、收支总量和年终总量一起在打印机上打出。如果收支总量与年终总量不等，则说明分类总帐数据库有错，须要修改登记项；这时须象以前一样，录入修改内容到数据库JOURNAL中，再把JOURNAL传送到GLEDGER数据库上。在做登记项调整之前，要使JOURNAL数据库记录为空。如下所示。

11. APPEND：打入APPEND JOURADUT JOURNAL〈C/R〉

为审理帐底，用APPEND命令累计JOURNAL登记项。

12. EMPTY：打入EMPTY JOURNAL〈C/R〉

用EMPTY命令清除JOURNAL数据库的所有记录，并将记录计数器置成零。到目前为止，数据库JOURNAL记录为零，你可以进行下一轮的数据输入。

第五章　屏幕菜单和命令过程

本章将向你介绍HELP命令建立的菜单，并用一个实例——分类总帐会计系统显示命令过程的用法。

屏幕菜单及命令过程使得计算机的操作大为简化。它减轻了你需要记住的许多过程及许多命令这类工作。因此，首先建立一个过程，然后逐条执行过程中的命令，是实现自动操作处理的好办法。如第四章所做的那些工作，我们可以将其编成一个过程，并建立一个菜单。在这种实际应用时，可以大大减少操作过程中出现的错误。

本章用到的rDBMS命令有：

HELP　帮助操作员选择和处理命令过程

RESTART　重新启动中断了的命令过程

RUN　启动并运行一个带有指示句的命令过程

SUBMIT　启动并运行一个命令过程

第一节　HELP 命令

HELP命令可以在终端屏幕上显示描述一组程序和过程的选择菜单。菜单中的程序是在运行过程时的选择对象。

打入HELP 〈菜单名〉，计算机检索名为〈菜单名〉·HLP的文件。如找到，就在屏幕上显示出菜单。操作员选择一个操作号录入，计算机就执行与该操作号有关的命令或命令过程。这里，HELP文件可使用文本编辑程序ED.COM或rDBMS的FORMAT.DBM程序创建。

第二节　命令过程

命令过程是存放在命令过程文件中的一组命令，它们可以被 rDBMS 的命令处理程序检索及处理。

rDBMS有两类命令过程：

基本的命令处理程序叫做SUBMIT。它类似于CP/M的SUBMIT命令，可以执行一系列的CP/M命令或rDBMS命令，允许进行参数的符号替换。过程中一切命令都是相继执行的。命令过程一旦被启动，就一直执行下去，除非在执行时发生了错误。在执行过程中，符号参数是不可变的。

第二类命令过程是使用RUN处理程序运行的。它除了具有SUBMIT的基本功能外，还增加了指示句处理能力。例如在过程中使用指示句定义符号参数，在执行时，可以从键盘打入参数值。因此，RUN程序使rDBMS处理事务的能力大大地加强了。

指示句及其说明如下：

*IF　根据符号参数的值做执行组命令的条件判定，它与*ENDIF合用

*ENDIF　指定条件判定组命令结束

*MESSAGE　在终端屏幕上显示信息

*GET　在命令过程处理期间，使用键盘给符号参数赋值

*LET　作为命令过程内部符号参数的赋值语句

*ECHO　在过程执行期间封锁来自rDBMS的信息

*END　命令过程执行终界。它与常规的命令过程完成符效果相同

*ABORT　指示过程中止，不再执行任何命令

SUBMIT命令过程由打入SUBMIT 〈过程名〉〈C/R〉启动。该过程既可以在CP/M 环境下启动（用SUBMIT.COM命令），又可以在rDBMS环境下启动（用SUBMIT.DBM命令）。启动时，计算机检索名为〈过程名〉.SUB 的命令过程文件；找到后，就逐条执行过程文件中的每条命令。如果命令过程正确的话，这个过程就一直执行下去。

RUN命令过程由打入RUN 〈过程名〉〈C/R〉启动。该过程只能在rDBMS 环境下运行。启动时，计算机检索名为〈过程名〉.CMD的命令过程文件；找到后，就按照指示句处理文件中的命令，同时并将处理的命令逐条写入工作文件中，然后执行之。

第三节　HELP 和命令过程实例

这里给出的例子就是我们在第四章给出的分类总帐会计系统。我们所建立的HELP 文件名为ACCTNG.HLP，其格式如下所示。使用时，只需打入 HELP ACCTNG 〈C/R〉，就会 在屏幕上显示出菜单。从上述的菜单中选择一项，可通过打入一个号码（十进制）后跟一个〈C/R〉来实现。系统一旦接收到这个号码就执行HELP程序并调用与所给号对应的命令或命令过程。

```
GENERAL LEDGER ACCOUNTING SYSTEM

    SELECT ONE OF THE FOLLOWING OPTIONS
    1. INPUT CHART OF ACCOUNTS
    2. UPDATE GENERAL LEDGER
    3. ENTER JOURNAL ENTRIES
    4. JOURNAL TRIAL BALANCE
    5. POST GENERAL LEDGER
```

HELP命令或命令过程与建立HELP文件时定义的一个数字号相联系，这种联系的定义可使用FORMAT命令来实现。它要求命令必须放在〔 〕内，前面跟有一个数字。注意：上面的屏幕中并不显示出〔 〕号及其中的命令。它是HELP程序自动隐去的。不过，使用FORMAT命令可以显示并编辑这个HELP文件。

下面描述一个HELP文件的创建过程。该过程与定义一个数据库的格式文件大同小异。

第四节　建立HELP 屏幕菜单

上面说过，使用CP/M的文本编辑程序ED.COM和rDBMS 的 FORMAT 程序都可以对HELP文件进行编辑。要注意的是，HELP文件一定要带有扩展名“.HLP”。

打入FORMAT B：ACCTNG.HLP〈C/R〉

系统显示出一个空屏幕。你可以在屏幕上结合光标进行满屏编辑。

打入〈control-E〉则结束ACCTNG菜单格式的编辑工作。此时你已经创建了一个 ACCTNG.HLP文件。其格式如下：

```
GENERAL LEDGER ACCOUNTING SYSTEM
SELECT ONE OF THE FOLLOWING OPTIONS
 1. INPUT CHART OF ACCOUNTS  [RUN B：ACCNTS]
 2. UPDATE GENERAL LEDGER    [RUN B：UPGLEDG]
 3. ENTER JOURNAL ENTRIES    [RUN B：JOURNIN]
 4. JOURNAL TRIAL BALANCE    [RUN B：TRIALB]
 5. POST GENERAL LEDGER      [RUN B：POSTGL]
```

第五节　建立命令过程文件

以下的工作是使用CP/M的文本编辑程序ED.COM建立命令过程文件。所有的过程文件一定要求扩展名.CMD。为了说明过程文件的建立过程，以ACCTNG．HLP 文件上引用的几个过程文件为例。

引用中要建立的过程文件有：

〈1〉ACCNTS.CMD　　录入帐目表
〈2〉UPGLEDG.CMD　　更新分类总帐
〈3〉JOURNIN.CMD　　录入日帐总计
〈4〉TRIALB.CMD　　日总帐底平衡
〈5〉POSTGL.CMD　　日帐汇总到分类总帐中

从ACCNTS.CMD开始按下列步骤建立命令过程文件。

建立ACCNTS.CMD命令过程文件

编辑：打入

```
ED B：ACCNTS.CMD〈C/R〉
I 〈C/R〉
B：〈C/R〉
ENTER GLTEMP〈C/R〉
COMPARE GLTEMP GLEDGER NOT MATCHING ACCOUNT〈C/R〉
TITLE T,'GENERAL LEDGER REPORTS'S,S,'NEW ACCOUNT NUMBER
ADDITIONS DATE', DATE,S,S〈C/R〉
PRINT RESULT BY ACCOUNT NAME DEBIT CREDIT〈C/R〉
APPEND GLEDGER RESULT〈C/R〉
COMPARE GLTEMP RESULT NOT MATCHING ACCOUNT〈C/R〉
TITLE T,'GENERAL LEDGER REPORTS',S,S'DUPLICATE ACCOUNT
```

```
NUMBERS DATE',S,S<C/R>
PRINT RESULT BY ACCOUNT NAME DEBIT CREDIT<C/R>
SORT GLEDGER BY ACCOUNT<C/R>
TITLE T,'GENERAL LEDGER REPORTS',S, S, 'GENERAL LEDGER',
DATE,S,S<C/R>
PRINT RESULT BY ACCOUNT NAME DEBIT CREDIT<C/R>
EMPTY GLTFMP<C/R>
HELP ACCTNG<C/R>
<control-z>
E<C/R>
```

至此，在B驱动器上建立了一个ACCNTS.CMD命令过程文件，若要执行时，只需打入RUN ACCNTS<C/R>或打入HELP ACCTNG<C/R>然后选择菜单内的第1项即可。同样，把分类总帐会计系统中的所有操作过程都写到一个过程文件中。

创建UPGLEDG.CMD命令过程文件

编辑：打入

```
ED B:UPGLEDG.CMD<C/R>
I<C/R>
B:<C/R>
UPDATE GLEDGER<C/R>
SORT GLEDGER BY ACCOUNT<C/R>
TITLE T,'GENERAL LEDGER REPORTS',S,S,'GENERAL LEDGER CH-
ART OF ACCOUNTS DATE',DATE,S,S<C/R>
PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT<C/R>
HELP ACCTNG<C/R>
<control-z>
E<C/R>
```

文件UPGLEDG.CMD被写到B驱动器上，若欲执行之，只须打入RUN UPGLEDG<C/R>或打入HELP ACCTNG<C/R>，然后选择菜单内的第2项即可。

创建JOURNIN.CMD命令过程文件

编辑：打入

```
ED  B:JOURNIN.CMD<C/R>
I<C/R>
B:<C/R>
ENTER JOURNAL<C/R>
SORT JOURNAL BY ACCOUNT<C/R>
SORT GLEDGER BY ACCOUNT<C/R>
COMPARE JOURNAL GLEDGER BY ACCOUNT<C/R>
*MESSAGE'ENTER ACCOUNTING MONTH'<C/R>
*GET S1<C/R>
```

```
TITLE T,'GENERAL LEDGER REPORTS FOR$1',PAGE,S,S,'JOURNAL
EXCEPTIONS FOR DATE',DATE,S,S<C/R>
PRINT RESULT BY ACCNAME DEBIT CREDIT<C/R>
HELP ACCTNG<C/R>
<control-z>
E<C/R>
```

于是，文件JOURNIN.CMD被写到B驱动器的磁盘上。若执行之，只需打入RUN JOURNIN<C/R>或打入HELP ACCTNG<C/R>，然后选择第3项即可。

创建TRIALB.CMD命令过程文件

编辑：打入

```
EDB：TRIALB.CMD<C/R>
I<C/R>
B：<C/R>
*MESSAGE'ENTER ACCOUNTING MONTH'<C/R>
*GET $1<C/R>
TITLE T,'GENERAL LEDGER REPORTS FOR $1',PAGE,S,S, 'JOURNAL
TRIAL BALANCE FOR DATE',DATE,S,S<C/R>
STAX JOURNAL BY DEBIT CREDIT [T]<C/R>
TITLE T,'GENERAL LEDGER REPORTS FOR $1',PAGE,S,S, 'GENERAL
LEDGER TRIAL BALANCE FOR DATE',DATE,S,S<C/R>
STAX GLEDGER BY DEBIT CREDIT [T]<C/R>
HELP ACCTNG<C/R>
<control-z>
E<C/R>
```

于是，文件TRIALB. CMD被写到B驱动器上的磁盘上。若执行之，打入RUN TRIALB<C/R>或打入HELP ACCTNG<C/R>，然后选择第4项即可。

创建POSTGL.CMD命令过程文件

编辑：打入

```
ED POSTGL.CMD<C/R>
I<C/R>
B：<C/R>
POST GLEDGER JOURNAL BY ACCOUNT AND ADD DEBIT, CREDIT
AND REPLACE MONTH<C/R>
COMPUTE GLEDGER ST YTDAMT = DEBIT-CREDIT<C/R>
SELECT GLEDGER ST ASSET IS Y<C/R>
TITLE T,'GENERAL LEDGER REPORTS FOR $1',
PAGE,S,S'ASSETS ACCOUNT REPORT FOR DATE',
DATE,S,S<C/R>
PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT AND
```

```
COMPUTE TOTAL YTDAMT〈C/R〉
SELECT GLEDGER ST LIABILITY IS Y〈C/R〉
TITLE T,'GENERAL LEDGER REPORTS',DATE,S,S,'LIABILITIES
ACCOUNT REPORT FOR DATE',DATE,S,S〈C/R〉
PRINT RESULT BY ACCOUNT ACCNAME DEBIT CREDIT AND
COMPUTE TOTAL YTDAMT〈C/R〉
HELP ACCTNG〈C/R〉
〈control-z〉
E〈C/R〉
```

于是，文件POSTGL.CMD被写到B驱动器的磁盘上。若执行之，只要打入RUN POSTGL〈C/R〉或打入HELP ACCTNG〈C/R〉，然后选择第5项即可。

到目前为止，你已经完成了分类总帐管理系统的屏幕菜单和命令过程文件的编制。打入HELP ACCTNG并选择菜单的各项任选，就可以完成第四章所要作的工作。

第六章　关系数据库的操作

本章将讨论数据库之间有关信息建立联系的命令。所讨论的是在数据库之间建立联系。怎样组合两种数据库的数据，怎样比较两个数据库，找出它们匹配或不匹配的关键字，怎样联接两个数据库的数据字段以便形成一个新的数据库和怎样把一个事务数据库中的数据汇总到主数据库中去。

本章用到的rDBMS命令为：

APPEND　把一个数据库中的所有记录添加到另一个数据库上。

COMBINE　把两个数据库的记录组合成一个结果数据库。

COMPARE　按指定的项匹配数据库Ⅰ和数据库Ⅱ的记录，并创建一个数据库Ⅰ的结果数据库。在结果数据库中包含数据库Ⅰ中满足指定的匹配或不匹配数据库Ⅱ的记录。

DEFINE　创建或重定义一个数据库结构。

EMPTY　删除一个数据库中的全部记录。

ENTER　把新的记录录入到数据库中。

JOIN　两个数据库的数据字段合并成一个结果数据库。

POST　按指定的项，使数据库Ⅰ与数据库Ⅱ的记录匹配。用数据库Ⅱ的记录更新数据库Ⅰ中匹配的记录，并建立数据库Ⅰ的结果数据库。

PRINT　按照指定的项在系统打印机上打印数据库报表。

PROJECT　投影数据库中指定数据项，并产生一个结果数据库。

SAVE　保存结果数据库。

SORT　按照指定的数据项对数据库进行分类。

TABULATE　用汇总报表格式显示或打印数据库。

TITLE　打印报表标题。

UPDATE　更新数据库或文件中现存的记录。

第一节　结果数据库

命令COMBINE，COMPARE，JOIN，POST，PROJECT和SELECT都在当前驱动器上建立一个结果数据库文件。如果当前驱动器是B，则在B驱动器上创建一个结果集文件$$.FRM、$$.DEF和$$.DAT。为了检索这些文件，则应把数据库名RESULT加到B驱动器的DATA.DIC上一次。例如：

打入ENTER B：DATA.DIC〈C/R〉

显示出DATA.DIC输入格式。

打入RESULT作为标题名，$$作为数据文件，$$作为格式文件，$$作为定义文件。打入〈control-E〉结束录入。再打入E，则把以上文件保存起来。

第二节　建立数据库之间的关系

关系数据库的基本特性之一是无需定义专门的关键字段。但是，在许多涉及到多个数据库的命令中，都存在一个字段匹配的概念。这些命令要求匹配的字段具有相同的定义（同样的名字、大小和类型）。涉及字段匹配概念的命令有COMPARE、JOIN和POST。如果打算使用这些命令，要提前考虑设计，以确保定义一个合适的数据库。

例如，考虑如下数据库：

Employee Database

emplno	emplname	salary	deptno	taskno
1	adams	400.00	100	10
2	smith	200.00	123	30
3	jones	500.00	123	20

Task Database

taskno	taskname	respdept	complete	hours
10	wrie spec	100	1/15/81	24
20	design program	123	4/30/81	40
30	test program	123	6/10/81	60
40	write ad	200	7/15/81	12

Deptment Database

deptno	deptname	manager
100	engineering	franking
200	programming	baker
123	marketing	davis

Time Database

deptno	taskno	hours
1	10	10
2	20	12
3	20	20
4	40	5

以上两个数据库EMPLOYEE和DEPTMENT定义了同样的数据字段DEPTNO，这两个数据库可通过这个数据字段建立联系。

注意，即使数据库TASK中的属性RESPDEPT与DEPTMENT中的属性 DEPTNO 有同样的定义，也不能通过这两个属性使两个数据库相关，因为它们没有同样的名字。

如果属性TASKNO在EMPLOYEE数据库中被定义成AN类型的，而且又在 TASK 数据库中被定义成为N类型的，那么，尽管它们有同样的属性名字，也不能发生联系。

下面的例子中继续使用这些数据库。

第三节　比较两个数据库找出匹配的字段

COMPARE命令条件MATCHING可使你确定一个数据库中的记录是否在另一个数据库中有匹配的记录。反之，使用NOT MATCHING条件可以确定一个数据库中的记录有无不匹配另一个数据库的记录。

在下列三种情况下，这个命令对事务处理非常有用：

1．将记录送入到要求有唯一记录的数据库中；

2．确保事务记录按指定的数据项一定匹配主数据库中的记录；

3．保证你已录入了所有要求录入的记录。例如：或许你希望比较EMPLOYEE和TIME数据库而保证所有的雇员都录入到时间记录表中。

若指定MATCHING条件，结果数据库中只包含具有匹配关系的记录。反之，使用NOT MATCHING条件，产生的结果数据库中只包含必须检查或改正的那些记录。

第四节　传送事务记录到主记录中

POST命令允许把数据录入事务数据库，并用事务数据库的值去更新有唯一记录的主数据库。数值可以进行加、减及替换。例如，可以把报表的时间传到TASK数据库，以便按任务号累计报表的时间。

第五节　联接两个数据库

JOIN命令建立结果数据库，它含有的数据项是两个数据库项的并集。两个数据库根据匹配场的关系进行联接。

当你希望减少数据录入量时，数据库联接命令是非常有用的。它通过从主数据库提取有关信息来做。例如，你的应用是用顾客号录入订货单，而在每个订单中，你并不希望录入顾客的姓名、地址等，这可以用联接顾客主数据库和订单数据库来实现。这样，在顾客订单上就可以包含有关顾客的信息。

第六节　组合两个数据库的数据

如果两个数据库具有相同结构（数据字段个数和记录大小相同），则可以组合两个数据库的数据。组合与添加数据库的理由有四条：

1．把含有附加记录的事务数据库随时添加到主数据库上；

2．把事务数据库添加到历史数据库上；

3．你想要选择出满足几种不同条件的数据库记录，并且要把它们汇集在结果数据库中；

4．你希望组合两个数据库的数据，然后在不改变原有数据情况下，完成对记录的组合。APPEND命令把一个数据库的记录添加到另一个数据库上。而COMBINE命令

与其类似，它除了将两个数据库的组合记录写到结果数据库外，还保持原有数据库中数据不变。

第七节　数据库的投影

PROJECT命令建立一个包含由指定数据项投影而产生的结果数据库。典型的应用是在使用JOIN和POST命令之前先用PROJECT投影产生有关的信息。

第八节　数据库应用实例

下面要讨论关系数据库命令的使用。第一个例子要讨论使用命令APPEND和命令COMPARE把唯一的记录录入到主数据库。本例假设已存在TASK、TIME、DEPTMENT和EMPLOYEE这四个数据库。下面使用命令、动作及注解的形式讨论过程中的每一条命令。

改变驱动器：打入B：〈C/R〉

把当前驱动器置成B

DEFINE：打入DEFINE NEWTASK〈C/R〉

系统询问（.FRM）文件名，本例中要为TASK定义工作文件，因而需要使用TASK.FRM文件。

打入TASK〈C/R〉

系统要求进入（.DEF）文件名。在本例中，使用TASK.DEF文件。

打入TASK〈C/R〉

系统自动地为数据文件NEWTASK.DAT分配空间，此时，系统指出Busy。而后不久，显示出NEWTASK数据库定义完毕，并等待进入数据。

ENTER：打入ENTER NEWTASK〈C/R〉

系统出现NEWTASK数据库的输入格式。请打入新的TASKNO值及其他项的值。如果所录入数据的字符个数等于分配给该数据项名后所跟有短线的个数，光标自动移到下一数据项的开始位置。如果录入数据值的字符个数不等于所分配的短线数，打入一个〈C/R〉，可使光标移到下一个数据项的位置上。为了录入缺省值，可在指定数据项第一个字符位置上打入一个〈C/R〉，若以前已定义好缺省值，则缺省值被录入。同时光标移到下一项的位置上。在一个记录的所有数据项信息全部录入后，可有如下的选择。即：

Abort (A)，Continue (C)，Delete (D)，End (E)，Print (P)

打入C，则进行下一个记录的录入。此时屏幕重新出现NEWTASK的输入格式。按照上述步骤录入下一个数据。当数据录入完毕时，打入E退出录入状态。系统出现提示符“B〉〉”。

UPDATE：打入UPDATE TASK〈C/R〉

仅当主文件TASK中已有的记录要进行更新时，才用到UPDATE命令。在打入此命令后，系统要求你录入检索条件：

打入TASKNO IS ××××××〈C/R〉

其中××××××表示TASKNO的值。如果找到这样的记录，则将该记录显示在屏幕上。选择任选R，并将光标移到要修改的位置上，打入新值。不需要修改的值只需打入一个

〈C/R〉。当修改完毕时，打入〈control-E〉结束，此时系统要求进入另一个检索条件。若你希望结束修改状态，再打入〈C/R〉就可退出修改状态。

COMPARE：打入COMPARE NEWTASK TASK NOT MATCHING TASKNO〈C/R〉

比较两数据库记录的值。将NEWTASK中与TASK在TASKNO项上不匹配的NEW TASK记录写到结果数据库中。其中结果数据库的格式及数据项与TASK.FRM相同。

TITLE：打入TITLE 'PROJECT REPORTS',S, 'NEWTASK NUMBER ADDITIONS FOR',DATE,S〈C/R〉

在NEWTASK NUMBER ADDITIONS报表的每页上打印出标题和日期。其标题采用两行印出。S表示换行。

PRINT：打入PRINT RESULT BY TASKNO TASKNAME DEPTNO COMPLETE HOURS〈C/R〉

打印出新加入的唯一的TASKNO（它们是与主文件TASK不匹配的记录）。

APPEND：打入APPEND TASK RESULT〈C/R〉

把录入的唯一TASKNO号的记录添加到主数据库TASK上。即，将$$.DAT文件连接到TASK.DAT文件上。

SORT：打入SORT TASK BY TASKNO〈C/R〉

按照TASKNO的顺序对TASK数据库分类。

TITLE：打入TITLE 'PROJECT REPORTS',S,'LIST OF TASKS-DATE',DATE, S〈C/R〉

在LIST OF TASKS报表的每页上都打印出标题和日期。

PRINT：打入PRINT TASK BY TASKNO TASKNAME DEPTNO COMPLETE HOURS〈C/R〉

打印出LIST OF TASKS报表。

COMPARE：打入 COMPARE NEWTASK TASK NOT MATCHING TASKNO(C/R)

比较两数据库记录的值，将 NEWTASK 中与 TASK 在 TASKNO 项上不匹配的 NEWTASK 记录写到结果数据库中。其格式及数据项与TASK.FRM一样。

SAVE：打入SAVE NEWTASK OK <C/R>

把 NEWTASK 数据库中与 TASK 数据库不匹配的记录(存放在结果数据库中)存储在 NEWTASK 数据库中。

第二个例子是使用 rDBMS 命令 APPEND、COMPARE、JOIN 和 PROJECT 录入匹配记录的标准过程。

改变驱动器：打入 B ：〈C/R〉

置当前驱动器为 B。

ENTER ：打入 ENTER TIME<C/R>

录入 TIME 数据时，系统用 TIME.FRM 输入格式响应。请打入新的时间 事 务记录。

SORT ：打入 SORT TIME BY TASKNO〈C/R〉

系统按 TASKNO 对 TIME 数据库进行分类。分类后可以大大地节省执 行 的 时 间。

COMPARE：打入 COMPARE TIME TASK NOT MATCHING TASKNO〈C/R〉

比较两数据库记录的值，将 TIME 中与 TASK 在 TASKNO 项上不匹配的 TIME 记

录写到结果数据库中，其中格式及数据项与 TIME.FRM 相同。

TITLE：打入 TITLE 'TIME REPORTS', X'PAGE,S, 'TIME EXCEPTIONS-BAD' TASKS FOR',DATE,S ‹C/R›

在 TIME EXCEPTIONS 报表的每页上都打印出以上的标题和日期。

PRINT：打入 PRINT RESULT BY TASKNO EMPLNO HOURS ‹C/R›

打印出与 TASK 数据库中 TASKNO 不与 TIME 数据库匹配的记录。确定问题所在，根据需要，或者更新 TASK 数据库，或者更新 TIME 数据库。若做了更新，则需要从以上的SORT 命令重新开始。

SORT：打入 SORT TIME BY EMPLNO ‹C/R›

系统按照雇员号顺序对 TIME 数据库分类。

SORT：打入 SORT EMPLOYEE BY EMPLNO ‹C/R›

按照雇员号对 EMPLOYEE 数据库分类。

COMPARE：打入 COMPARE TIME EMPLOYEE NOT MATCHING EMPLNO ‹C/R›

比较两数据库记录的值，将 TIME 中与 EMPLOYEE 在 EMPLNO 项上不匹配的TIME 记录写到结果数据库中。其格式及数据项与 TIME.FRM 一样。

SAVE：打入 SAVE EMPLBAD OK ‹C/R›

将 TIME 数据库中与 EMPLOYEE 数据库匹配的 EMPLNO 记录存储到一个新的EMPLBAD 数据库中。

PROJECT：打入 PROJECT TASK BY TASKNO DEPTNO ‹C/R›

创建一个结果数据库，它包含 TASK 数据库的全部记录，但只有两个数据场（TASKNO 和 DEPTNO）。

SORT：打入 SORT RESULT BY TASKNO ‹C/R›

系统按照任务号对结果数据库分类。此处的分类工作是为以后使用 JOIN 命令作准备。

SORT：打入 SORT EMPLBAD BY TASKNO ‹C/R›

系统按 TASKNO 顺序对 EMPLBAD 数据库分类。

JOIN：打入 JOIN EMPLBAD RESULT BY TASKNO ‹C/R›

结果数据库与 EMPLBAD 数据库联接并创建一个新的结果数据库。它的数据项中含有 TASKNO,EMPLNAME,HOURS,DEPTNAME 和MANAGER。

SORT：打入 SORT RESULT BY MANAGER ‹C/R›

系统按照 MANAGER 顺序对结果数据库分类。

TITLE：打入TITLE 'TIME REPORTS',X,PAGE,S,'TIME EXCEPTIONS-BAD EMPLOYEE NUMBERS FOR',DATE,S‹C/R›

在 TIME EXCEPTIONS 报表的每页上打印标题和日期。

PRINT：打入 PRINT RESULT BY MANAGER EMPLNO TASKNO HOURS‹C/R›

打印出 EMPLOYEE 中与 TIME 不匹配的 TASKNO 记录。分类后便于找到对有关任务负责任的 MANAGER。根据需要或者更新 EMPLOYEE 数据库，或者更新 TIME 数据库。在做了更新后，则需要返回到上面的 SORT 命令重新开始。

COMPARE：打入 COMPARE EMPLOYEE TIME NOT MATCHING EMPLNO ‹C/R›

比较两数据库记录值，将 EMPLOYEE 中与 TIME 在 EMPLNO 上不匹配的 EMPLO-

YEE 记录写到结果数据库中，其格式及数据项与 EMPLOYEE.FRM 相同。

TITLE：打入 TITLE 'TIME REPORTS',X,PAGE,S,'TIME EXCEPTIONS-MISSING TIME SHEETS FOR',DATE,S‹C/R›

在 TIME EXCEPTIONS 报表的每页上都打印出该标题和日期。

PRINT：打入 PRINT RESULT BY EMPLNO EMPLNAME ‹C/R›

打印出 EMPLOYEE 数据库中与 TIME 数据库在 EMPLNO 上不匹配的记录。确定问题所在，根据需要或者更新 TIME 数据库，或者更新 EMPLOYEE 数据库。在你做了更新后，则要从以上 SORT 命令重新开始。

当打印出的报表为空记录时，则新加入的记录匹配是完整的。即所有的 TIME 记录与数据库 EMPLOYEE 和 TASK 全部匹配。下一步是把 TIME 汇总到 TASK数据库中去。

在TASK 系统实例中，TIME 登记项能以日报汇总作为预算。在月末，TIME 登记项被传送到 TASK 数据库中。汇总后，TIME 记录在账底数据库中做累计。然后 TIME 数据库置空，准备下一个月的账目录入。下面给出使用 rDBMS 命令完成汇总记录的过程。

改变驱动器：打入 B :‹C/R›

把当前驱动器置成 B。

POST：打入 POST TASK TIME BY TASKNO AND ADD HOURS‹C/R›

对每个与 TASKNO 相匹配的记录，把 TIME 数据库中的 HOURS 加到 TASK 数据库的 HOURS 上。

TITLE：打入 TITLE 'TASK MONTHLY REPORTS', PAGE, 'FOR', DATE,S,S, ‹C/R›

在 TASK MONTHLY 报表的每一页上打印标题和日期。

PRINT：打入 PRINT TASK BY TASKNO TASKNAME AND COMPUTE TOTAL HOURS‹C/R›

打印出记录的投影项，同时打印出它的描述和迄今为止的小时数。对数据项 HOURS做出总计列出。

APPEND：打入 APPEND TIMEHIST TIME‹C/R›

为了在 TIMEHIST 数据库中清理账底，使用 APPEND 命令累计 TIME 数据库的登记项。两个数据库有同样的格式文件和定义文件，但数据文件不同。

EMPTY：打入 EMPTY TIME‹C/R›

该命令删除一个数据库中的全部记录并把记录计数器置成零。于是 TIME 数据库被复位，为进行下一轮输入作准备。

联接两个数据库也是很有用的。下面，让我们来开发一个由每个经理负责管理的工资级别报表。

改变驱动器：打入 B :‹C/R›

当前驱动器置成 B。

PROJECT：打入 PROJECT EMPLOYEE BY DEPTNO EMPLNAME SALARY ‹C/R›

对 EMPLOYEE 数据库的三个数据字段进行投影并创建一个结果数据库。其中包含 EMPLOYEE 数据库的全部记录，但只有被投影的三个数据字段。

SORT：打入 SORT RESULT BY DEPTNO ‹C/R›

系统按照部门号对结果数据库进行分类。经过分类后，可以在进行 JOIN 操作时提高系统效率。

SORT：打入 SORT DEPTMENT BY DEPTNO ‹C/R›

按照顺序号对 DEPTMENT 数据库进行分类。

JOIN：打入 JOIN DEPTMENT RESULT BY DEPTNO

结果数据库与 DEPTMENT 数据库进行联接。创立一个有数据字段 DEPTNO，DEPTMENT，MANAGER，EMPLNAME 和 SALARY 的结果数据库。

TITLE：打入TITLE 'SALARY SUMMARY BY DEPARTMENT',X,PAGE,'FOR',
DATE,S,S‹C/R›

在工资汇总报表的每一页上都印出以上标题和日期。

TABULATE：打入 TABULATE RESULT BY DEPTNO DEPTNAME MANAGER
AND COMPUTE TOTAL SALARY [P]‹C/R›

打印出每个部门的部门号、名字、经理的汇总报表，并列出每个部门全部雇员的总工资。

第七章　查询和报表书写功能

本章讨论 rDBMS 的查询功能和报表处理能力。生成报表最困难的工作是得到你所希望的格式的数据。这里用几个例子说明怎样联系若干个数据库的信息而准备报表的工作。

本章所用到的 rDBMS 命令有：

CHANGE　改变一个数据库中某记录的数据项值。

COMBINE　组合两个数据库的全部记录，并创建一个结果数据库。

COMPARE　按照指定数据项比较两个数据库(数据库Ⅰ和数据库Ⅱ)，并产生一个结果数据库，其记录是数据库Ⅰ中与数据库Ⅱ中按指定项上匹配（或不匹配）的数据库Ⅰ的记录。

DISPLAY　显示一个数据库中满足检索条件的记录。

JOIN　对两个数据库的数据字段进行自然联接并创建一个结果数据库。

LIST　顺序显示数据库中的每个记录。

PRINT　顺序打印数据库中的全部记录。

PROJECT　在一个数据库中选出指定的数据项的记录，并将其投影在结果数据库中。

SAVE　保存结果数据库的记录。

SELECT　选择满足指定条件的数据库记录，并创建一个结果数据库。

SET　设置 rDBMS 操作参数。

SORT　按照指定的数据项对数据库进行分类。

STAX　显示或打印指定数据项的统计值。

TABULATE　列表或打印汇总报告。

TITLE　打印报表标题。

第一节　数据库查询

rDBMS 除了具有易于定义数据库、录入数据和存储数据的功能外，还具有象随意检索这样灵活的功能。用户通常希望用三种方法访问数据。即：

（1）非结构式查询。

（2）如果出现问题该怎么办？

（3）周期性的报表。

对于非结构式的查询有两种形式。

第一种可以立即看到满足指定条件记录的全部信息。

第二种是不断地给出检索条件，直到找到的记录个数可被用户接受。然后经过选择产生需要的信息。

对第一类查询是使用 DISPLAY 命令处理，满足检索条件的每个记录都在屏幕中按照输入格式显示出来。然后，可以浏览整个数据库并打印出任何显示出来的记录。

例如：你希望查看部门代号为20中的所有雇员的数据，而上次对雇员的检查日期是在

1980年6月以前，则你可打入：

DISPLAY EMPLOYEE WHOSE DEPT IS 20 AND REVIEW<6/1/80

在第二种查询中，用户并不希望查找数量很大的数据，而是有选择地缩小查询范围，直到找到需要的记录。这类查询需要连续地使用 SELECT 命令，也可以在操作中使用一些 SAVE 命令，直至选出合适的记录为止。

下面的例子是在雇员中物色合适人选作为工程经理。

给定条件：（必备条件）：BS 或 MS 学位，有管理经历。

（理想条件）：MESS 学位，当前不高于该职位的工资水平，在本公司有五年或五年以上的工作资历。

首先在雇员档案数据库中选择符合条件的雇员记录。打入：

SELECT EMPLOYEE WHOSE DEGREE IS BS* OR DEGREE IS MS*〈C/R〉

Data base EMPLOYEE 2820 Records

Busy

Total records in Result Set =152

这里须用两个 SELECT 语句，因为 AND 和 OR 不能在同一个命令行中。所以再打入：

SELECT RESULT WHOSE POSITION>1999 〈C/R〉

Data base RESULT 152 Records

Busy

Total records in Result Set = 51

找出雇员中雇员号大于或等于 2000 的人作为管理人员。注意，结果数据库既作为查找源，又作为查找结果写入结果数据库。

打入 SAVE POTENTL OK 〈C/R〉

将必备条件的结果保存起来。万一以后查找中产生的候选记录个数太少，可用来在放宽条件的情况下重新查找。

SELECT POTENTL WHOSE SALARY<40000〈C/R〉

Data base POTENTL 51 Records

Busy

Total records in Result Set = 30

工作结束，我们再来观察一下 MESS 情况。

SELECT RESULT WHOSE DEGREE IS MSEE〈C/R〉

Data base RESULT 30 Records

Busy

Total records in Result Set = 2

大概这是最好的两个候选人。让我们看一下他们是谁。

LIST RESULT BY NAME DEPT〈C/R〉

Data base RESULT 2 Records

NAME	DEPT
FRANKLIN.CARL	ENGR
STORER.ALFRED	R&D

再回去看看第二批候选人。

SELECT POTENTL WHOSE SALARY<40000 AND DEGREE IS BS?E〈C/R〉

Data base POTENTL 51 Records

Busy

Total records in Result Set = 22

SELECT RESULT WHOSE HIRS.DATE< 6/1/76〈C/R〉

Data base RESULT 22 Records

Busy

Total records in Result Set = 10

现列出第二批候选人。

LIST RESULT BY NAME DEPT DEGREE〈C/R〉

Data base RESULT 10 Records

NAME	DEPT	DEGREE
MORAN.GEORGE	ENGR	BSME
JONES.LORETTA	R&D	BESS
KUIPER.JAMES	ACCT	BSIE

第二节 报 表

本 rDBMS 系统可产生五种格式的报表：

（1）屏幕格式

该格式由 PRINT〈数据库名〉命令产生。例如：PRINT BUDGET。

（2）列格式

该格式由 PRINT〈数据库名〉 BY 〈数据项 1〉,〈数据项 2〉,……〈数据项32〉命令产生。例如：

PRINT BUDGET BY ACCOUNT AMOUNT

（3）带统计的列格式

该格式由 PRINT 〈数据库名〉 BY 〈数据项 1〉,〈数据项 2〉,……AND COMPUTE〈统计函数〉〈数据项 3〉,〈数据项 4〉,……命令产生。例如：

PRINT BUDGET BY PERIOD AND COMPUTE TOTAL SALES,COST, SUBTOTAL USING ACCOUNT

（4）汇总格式

该格式由 TABULATE 〈数据库名〉 BY 〈 数据项 1 〉, 〈 数据项 2 〉, …… {AND COMPUTE 〈统计函数〉〈数据项 3〉, 〈数据项 4〉, ……产生。例如：

TABULATE BUDGET BY PERIOD AND COMPUTE TOTAL SALES,COST, SUBTOTAL USING ACCOONT

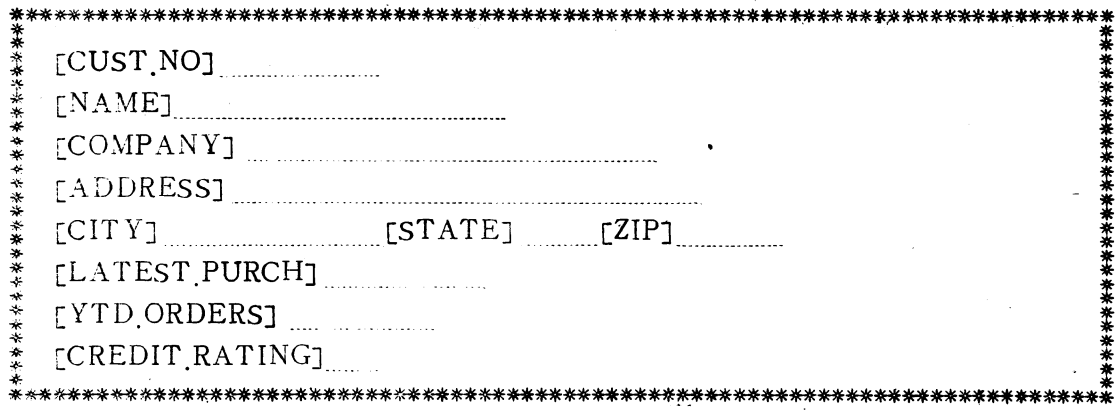
（5）统计报表格式

该格式由命令 STAX 〈数据库名〉 BY 〈数据项 1〉, 〈数据项 2〉,……产生。例如：

STAX BUDGET BY SALES,COSTS

第三节　屏幕格式报表

对于每个记录用多行信息的报表，屏幕格式报表是非常有用的。已知一个顾客数据库格式如下：



我们希望用它生成一个邮件并把它寄给自从3/1/81以后尚未订货的顾客。

定义该数据库。

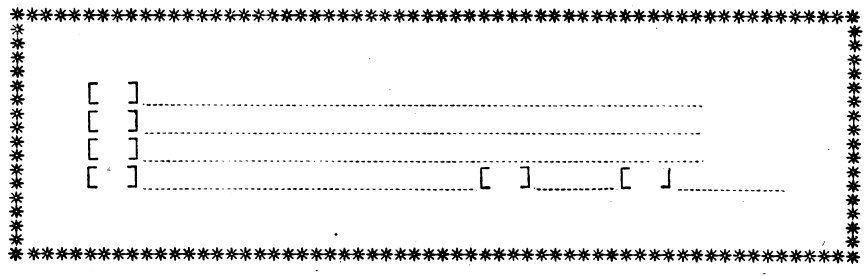
利用 COPY 命令得到一个修改的屏幕。

COPY MAILLABL.FRM = CUSTOMER.FRM〈C/R〉

现在用 FORMAT 命令编辑 MAILLABL.FRM 文件。格式编辑细则见第九章。

FORMAT MAILLABL〈C/R〉

编辑下面出现的格式。



把邮寄标签数据库录入数据词典。

ENTER DATA.DIC 〈C/R〉

对应于标题和格式文件处请录入 MAILLABL，对应于数据和定义文件处录入$$。(以后将用结果数据库打印其数据)其他的数据项空着留下。现在已完成了 MAILLABL数据库描述。下面的步骤说明如何选择和打印邮签。

获得报表信息。

为了获得报表的理想记录，录入如下命令语句。

SELECT CUSTOMER ST LATEST.PURCH LT 3/1/81〈C/R〉

选择的条件可随邮寄的目的而变化。下一步是根据邮签要在数据项上投影。打入：

PROJECT RESULT BY NAME*,COM*,ST*,ZIP〈C/R〉

注：被投影的数据项一定要与 MAILLABL 数据库中的数据项一样，而且要求投影的排列次序相同。

还需要按 ZIP 码的次序排列顾客记录：

打入 SORT RESULT BY ZIP 〈C/R〉

即用 ZIP 码对 RESULT 中的记录进行排序。

打入 TITLE 6,S,6〈C/R〉

每个邮签占六行，而在 MAILLABL 屏幕中占四行。我们使用的邮签要有 1 英寸的空余。以每英寸六行和两个邮签印出四行的格式打印，标签前放两个空格以使打印机对准标签。当执行 TITLE 命令时，打印机立即跳过二行。发出 TITLE 命令后，标签应该对齐。再印出邮签，即打入：

PRINT MAILLABL 〈C/R〉

第四节　列格式报表

顺序印出指定的数据项时，列报表格式很有用。引用以前邮签的例子，如果我们希望得到邮送给顾客的报表，我们就可以使用列报表格式。如：

SORT RESULT BY COMPANY 〈C/R〉

我们希望按照公司索引顺序报表。打入

TITLE T,'MAILING TO CUSTOMS WHO HAVE NOT ORDERED SINCE 3/1/81',S,S,'MAILING SENT',DATE,X,X,PAGE 〈C/R〉

PRINT RESULT BY COMPANY,NAME,ADDRESS,CITY,STATE,ZIP 〈C/R〉

时生成如下格式的报表：

MAILING TO CUSTOMERS WHO HAVE NOT ORDERED SINCE 3/1/81

MAILING SENT 6/1/81 Page 1

COMPANY	NAME	ADDRESS	CITY	STATE	ZIP
ABC	JONES,JL	34 CARGILL	READING	PA	23456
CONDOR	WILLAMS	PO BOX 8318	ANN ARBOR	MI	48107

第五节　带统计的列报表

有时我们非常希望报表末尾带上统计信息。在许多情况下，当一些关键字值改变时，小计和其他统计也很有用（常常把关键字值改变叫做级断开或控制断开）。这样的报表由打入 PRINT 命令产生。下面的例子用到四个数据库，它们是：

COSTOMER、PARTS、ORDERS 和 EXTENSION。

CUSTOMER 数据库结构如下(每个用户有一个记录)：

```
[CUST.NO]
[NAME]
[COMPANY]
[ADDRESS]
[CITY]          [STATE]    [ZIP]
[LATEST.PURCH]
[YTD.ORDERS]
[CREDIT.RATING]
```

PARTS 数据库中含有有关出售零件的信息：

```
[PART.NO]
[DESC]
[COST]
[PRICE]
```

ORDERS数据库，每个顾客订单有一个记录：

```
[CUST.NO]
[PART.NO]
[DATE]
[QTY]
```

EXTENSION 是只有一个记录的数据库，它用来选出为计算附加的价格和花费所需的数据项：

```
[PART.NO]
[EXT.COST]
[EXT.PRICE]
[PROFIT]
```

使用下面一组命令语句，可产生所希望的报表：

```
SORT ORDERS BY PART.NO<C/R>
JOIN ORDERS PARTS BY PART.NO<C/R>
SORT RESULT BY CUST.NO<C/R>
JOIN RESULT CUSTOMER BY CUST.NO<C/R>
JOIN RESULT EXTENSION BY PART.NO [D]<C/R>
COMPUTE RESULT ST EXT.PRICE = QTY*PRICE<C/R>
SORT RESULT BY COMPANY<C/R>
TITLE 'CUSTOMER ORDERS',D,X,P<C/R>
PRINT RESULT BY PART.NO AND COMPUTE TOTAL EXT.PRICE
SUBTOTAL BY COMPANY <C/R>
```

生成的一个报表如下：

```
CUSTOMER ORDERS 6/1/81 Page 1
PART.NO   EXT.PRICE   COMPANY
A12345      450.00    ABC ENTERPRISES
A45654     1500.00    ABC ENTERPRISES
  Subtotal
           1900.00
A12345     1400.00    CONDOR COMPUTER
B76554     1650.00    CONDOR COMPUTER
B65321      995.00    CONDOR COMPUTER
  Subtotal
           4045.00
Total      5995.00
```

第六节　汇总报表

用 TABULATE 命令可产生汇总报表，应用以前定义的数据库，生成一个由零件号表示的售货总结报表。例中的结果数据库来自以前例子中的结果数据库。

打入：

```
SORT RESULT BY PART.NO <C/R>
TITLE 'SALES BY PART NUMBER AS OF', DATE,X,X,P <C/R>
TABULATE RESULT BY PART.NO AND COMPUTE TOTAL QTY
EXT.PRICE [P] <C/R>
```

将产生如下形式的报表：

```
SALES BY NUMBER AS OF 6/1/81 Page 1
  PART.NO          QTY        EXT.PRICE

  A12345           12          1850.00
  A45654           10          1500.00
  B65321            1           995.00
```

B76554	15	1650.00
Total	38	5995.00

第七节　统计报表

统计报表不显示详细的信息数据，而显示诸如 TOTAL, MIN, MAX 和 AVERAGE 这样的统计信息。它也能用 COUNT 统计数据库的记录个数。下面的例子表明售货、价格的总计以及获得的利润。这里的数据库来自前边例子中的结果数据库。通过打入：

STAX RESULT BY EXT.PRICE EXT.COST PROFIT[T] [P] [C]〈C/R〉

可以得到报表：

EXT.PRICE	EXT.COST	PROFIT
Total		
5995.00	4563.46	1431.54

Count = 4

第八章　与外部程序的接口

rDBMS 提供了一些有用的功能块。进行数据录入、数据操作和报表都需要用到这些功能块。但对某些应用，需要把数据库中的数据提供给其他程序来使用。例如，你也许希望用词处理程序产生打字信件并把它传送给在数据库中存有信息的人。需要这种应用接口的其他实例有：用外部程序来做一个复杂的计算，或用非 rDBMS 所提供的格式生成报表。反之，你可以用外部程序生成放到你的数据库中的数据。

接口命令 READ 和 WRITE 可以使一个数据库与其他程序通讯。

本章使用的 rDBMS 命令有：

EMPTY　删除一个数据库中的全部记录，并将记录记数器置成零。

PRINT　按顺序打印数据库中的记录。

PROJECT　用一个数据库的指定数据项建立结果数据库。

READ　把一个 ASCII 文件记录读到数据库里。

SELECT　选择满足指定条件的数据库记录并用选出的记录生成结果数据库。

SORT　按照指定的数据项对数据库分类。

WRITE　利用数据库的数据创建一个 ASCII 文件。

第一节　词处理实例

从 CUSTOMER 数据库选出数据，产生一个打印文件。下面给出命令、动作和注解。

改变驱动器：打入 B ：〈C/R〉

将当前驱动器置成 B 。

SELECT：打入 SELECT CUSTOMER WHERE LAST.PURCHASE LT 1/1/81 〈C/R〉

选出自从1981年 1 月 1 号以来未购买过任何东西的顾客作邮信对象。

PROJECT：打入 PROJECT RESULT BY NAME,ADDRESS,CITY,STATE,ZIP 〈C/R〉

从 CUSTOMER 数据库中投影出 NAME、ADDRESS、CITY、STATE和ZIP 送到 RESULT 中，即在结果数据库的记录中，仅包含被投影的数据项。

SORT：打入 SORT RESULT BY ZIP 〈C/R〉

因为预先分类过的批信件邮资比较便宜，故信件可按 ZIP 号码顺序存放。

WRITE：打入 WRITE RESULT CUSTLIST.ASC 〈C/R〉

把结果数据库的内容写到 CUSTLIST.ASC 文件中去。这是一个 ASCII 文件，适合于词处理程序使用。词处理程序利用该文件产生打字信件。

第二节　计算程序实例

用另一个工程程序计算，数据首先写到 ASCII 文件中。该文件可由工程计算程序进行修改。经修改的数据再读回数据库中。

命令、动作和解释：

改变驱动器：打入 B: ‹C/R›

把当前驱动器置成 B 。

PROJECT：打入 PROJECT ENGRDATA BY MEAS.ID X1 Y1 X2 Y2 DISTANCE ‹C/R›

将六个数据字段投影到结果数据库中。在 ENGRDATA 数据库中， MEAS.ID 是唯一的关键字。X,Y座标和 DISTANCE 数据字段是计算程序需要的数据。

WRITE：打入 WRITE RESULT ‹C/R›

把结果数据库写到 ASCII 文件中。因为命令中没有指定 ASCII 文件名，故该文件名为 **$$.ASC**。

现在可执行名为 DISCALC 的工程计算程序。

打入 **$** DISCALC ‹C/R›

该程序是 rDBMS 之外的命令，故在命令表示中首先给出一个“**$**”号，标识该文件具有“.COM”扩展名。该程序计算（x_1,y_1）和（x_2,y_2）之间的距离，并把结果写回到 ASCII 文件中。

EMPTY：打入 EMPTY RESULT OK ‹C/R›

因为结果数据库仍包含原来的记录，所以需要把该库的数据置空。否则，在用READ命令时，就会产生重复的记录。

READ：打入 READ RESULT ‹C/R›

把名为 $$.ASC 文件中处理好的数据读回到结果数据库中。

POST：打入 POST ENGRDATA RESULT MATCHING ID REPLACE DISTANCE ‹C/R›

ENGRDATA 数据库中 DISTANCE 数据字段的数据用计算出的数据替换。

PRINT：打入 PRINT ENGRDATA BY MEAS.ID X1 Y1 X2 Y2 DISTANCE ‹C/R›

在系统打印机上打印数据。

第九章　FORMAT命令细则

本章讨论创建和编辑屏幕格式命令。用户可以用FORMAT命令创建和编辑屏幕和HELP文件。

为了创建一个新的屏幕格式文件（例如创建格式文件ORDERS.FRM），需要录入命令行：

FORMAT ORDERS 〈C/R〉或FORMAT ORDERS.FRM 〈C/R〉

该命令可以清除整个屏幕，而只留下屏幕底部的提示行。

FORMAT程序有两种选择方式：

替换方式（replace mode)和插入方式（insert-mode)。替换方式是常规方式。我们下面将逐步讨论这两种方式。

第一节　替换方式和插入方式

在替换方式下，用户可以在已有的字符或空格上打入字符，结果，新打入的字符替换了原有字符。通过使用控制键可使光标在屏幕的任何地方移动。

〈control-L〉　使光标向前（向右)移动；

〈control-H〉　使光标向后（向左）移动；

〈control-J〉　使光标向下移动；

〈control-K〉　使光标向上移动。

注意：如果光标键在你的终端上没做这些控制字符的定义，则必须使用控制键。

使用上述控制键，用户可把光标放在CRT上的任何位置。打入字母数字字符，在光标指定点录入该字符。数据项名（方括号中的）、短线和注解（非括号中的）都是以这种方式录入屏幕的。如果你希望在两个字符间加些字符，那么，就要用到插入方式。

插入方式可用来把字符加到光标指定的地方，而使原有的字符保留下来。当用替换方式时，打入〈control-A〉就可使该方式改成插入方式。同样，再打入〈control-A〉则改回替换方式。在插入方式下，用光标控制键可把光标放在CRT的需要插入字符的位置。打入一个字符，则该字符在光标点被录入，而那行中，插入点右边的字符串向右移动。

注意：若向右移动的字符串超出80列位置，则系统把超出的字符都删除掉。如果既要插入字符，又要保留那行中全部的原有字符，则打入一个〈C/R〉（相当于插入80个字符的位置），使屏幕的其余部分移到下一行中。

注意：用这样的方式插入一行，则会将底行上的字符（即23行）删除。

在插入方式下，打入〈control-D〉（或DEL，或RUBOUT），则可把光标所指的字符删除掉，而该行右边的字符串向左边移动一格。

打入〈control-X〉可以删除光标所在的行。如果该行右边具有80个空格，则将这些空格也删除。其后继行向上移动一行。

当完成屏幕格式时，返回替换方式，以便保存屏幕。在替换方式下，打入〈control-E〉

可结束并保存屏幕格式；若打入〈contrlo-C〉则结束，而不保存屏幕格式。

如果已经建立了一个新的屏幕格式，系统就问是否想要定义一个新的数据库，用户可以回答Y或N。如果用N响应，则FORMAT工作完成；如果用Y回答，系统就会显示数据字段名字，然后进行定义工作。DEFINE命令的细则参考第十章。

第二节　编辑现存的格式实例

在许多实际的应用中，可能需要编辑一个已经存在的屏幕格式。例如，使用FORMAT命令修改RESULT数据库的$$.FRM文件。由JOIN或者PROJECT这类命令产生的数据库RESULT，可以用$$·FRM文件名将其读出。

这可用FORMAT命令来做，如下所示：

假设有如下屏幕（来自PROJECT命令的结果）需要编辑，使每个数据项名都在单独的一行上。其过程如下：

1. FORMAT $$.FRM 〈C/R〉

a. 显示当前格式，准备在替换方式下进行修订

```
〔NAME〕.....〔STREET〕.....〔CITY〕.....
.....〔STATE〕.....
```

2. 打入〈control-A〉，改为插入方式

```
〔NAME〕.....〔STREET〕.....〔CITY〕.....
.....〔STATE〕.....
```

3. 用控制键将光标移动到第二个字段〔STREET〕的左括号下，然后打入一个〈C/R〉

a. 在第一个字段〔NAME〕之后插入一个空行并使表的剩余部分下移一行

```
〔NAME〕.....
.....〔STREET〕.....〔CITY〕.....
.....〔STATE〕.....
```

4. 使用控制键将光标移到屏幕中NAME左括号之下，打入〈control-D〉，直到字段〔STREET〕位于〔NAME〕字段之下。

```
〔NAME〕.....
〔STREET〕.....〔CITY〕.....
.....〔STATE〕.....
```

5．使用控制键移动光标到字段〔CITY〕的左括之号下，然后打入一个〈C/R〉

a．在第二个字段〔STREET〕之后插入一个空行，表的余留部分下移一行

```
*********************************************************************************
*                                                                               *
*    〔NAME〕 ____________________                                               *
*    〔STREET〕 ____________________                                             *
*                                        〔CITY〕 ____________________           *
*    ____________________〔STATE〕 ____________________                         *
*                                                                               *
*********************************************************************************
```

6．使用控制键将光标移到〔STREET〕的左括号之下，打入〈control-D〉直至〔CITY〕移到〔STREET〕字段之下

a．注意：第三个字段分裂成两个

```
*********************************************************************************
*                                                                               *
*    〔NAME〕 ____________________                                               *
*    〔STREET〕 ____________________                                             *
*    〔CITY〕 ____________                                                       *
*    ____________________〔STATE〕 ____________                                 *
*                                                                               *
*********************************************************************************
```

7．使用控制键将光标移到〔CITY〕字段划底线部分的第二部分开始，打入空格直到第二部分底线刚好超过CITY底线的第一部分并正好位于其下

```
*********************************************************************************
*                                                                               *
*    〔NAME〕 ____________                                                       *
*    〔STREET〕 ____________                                                     *
*    〔CITY〕 ____________                                                       *
*    ____________〔STATE〕 ____________                                          *
*                                                                               *
*********************************************************************************
```

8．使用控制键将光标移到刚好超过〔CITY〕字段底线的第一部分。打入〈control-X〉

a．删除一个空行，把字段的第二部分放好

```
*********************************************************************************
*                                                                               *
*    〔NAME〕 ____________                                                       *
*    〔STREET〕 ____________                                                     *
*    〔CITY〕 ____________〔STATE〕 ____________                                 *
*                                                                               *
*********************************************************************************
```

9．重复步骤3和4，把第四个字段移到〔CITY〕之下的位置

```
*********************************************************************************
*                                                                               *
*    〔NAME〕 ____________                                                       *
*    〔STREET〕 ____________                                                     *
*    〔CITY〕 ____________                                                       *
*    〔STATE〕 ____________                                                      *
*                                                                               *
*********************************************************************************
```

以上的实例会使你了解怎样用FORMAT命令有效地编辑屏幕。

第十章　DEFINE命令细则

本章讨论创建数据库用到的DEFINE命令。

DEFINE命令使用户能够创建和修改数据库定义，它也提供创建替换数据库（即用不同方法表示数据的数据库）的手段。

本章通过终端屏幕描述出DEFINE命令的各种功能。例子中，用户输入用醒目的形式给出。

第一节　修改定义

这一节描述怎样用DEFINE命令修改一个现存的数据库的数据定义。为此，要求数据库必须是空的。如果数据库中含有数据，有关修改定义的情况请参考第十一章。

第二节　编辑方式

当使用DEFINE重新定义数据库时，会将现存的定义以编辑方式显示出来。编辑方式每次显示一个数据项的现有定义，并把光标放在定义的右边。

为了改变定义，将光标移到要修改的项前并用新的信息代替（〈control-H〉使光标左移，而〈control-L〉使光标右移）。

数据项定义改变之后，系统统计出新的数据项的大小及编辑规范(MAX,MIN)。并用空格替换掉当前的字段大小和编辑。然后，退回到数据项类型并打入〈C/R〉，新计算的值出现在屏幕上。

若要进行下一个数据项类型的修改，可打入〈C/R〉。

为了跳到一个指定的数据项类型处，打入一个“＊”号，系统产生提示符“>”，录入要修改的数据项名后跟一个〈C/R〉，则系统显示出所要修改的数据项的定义。移动光标到要修改的位置，打入新值。然后，系统出现提示符“>”，此时再打入一个“＊”号结束定义修改编辑。

下例将描述用户可以怎样修改一个缺省值和改变MAX,MIN值限。

```
B≫DEFINE TEST
This Data Base exists already
Choose Option: Describe(D), Re-define(R), End〈C/R〉?R
Enter data definitions in the following format:
FIELD-NAME:FIELD-TYPE,FIELD-SIZE,MIN-VALUE,MAX-VALUE,
  "DEFAULT-VALUE"
CHOICES:   (ANJ$R)   (1-127 bytes)   (1-10 digits)   (0-15Characters)
 1. NAME:  AN,       25, 0, 25,      "MISSING        "
 2. AGE:   N,        1, 18, 65,      "               "
```

```
{ {END} }
Definitions OK (Y/N)?Y
Busy
Do you want a printed copy of the data definitions (Y/N)?Y
Attribute summary of Data Base TEST
 1. NAME: AN,25,0,25,"  MISSING  "
 2. AGE: N,1,18,65,"
Record Size(Bytes) = 27
Total Records = 0
```

第三节　替换数据库描述

大多数数据库应用中，单个数据库描述已够用了，但是有些应用中，要求描述替换数据库。例如：用户可能希望在输入格式屏幕上加入注解，以帮助录入数据。而当数据库在列表时，又不希望把它们也列出来。这可通过在数据库词典中设两个数据库描述来实现。这两个数据库可以指定相同的数据文件和定义文件，而它们的格式文件不同。

下例中，创建了一个叫做TESTX的新数据库，它相应的数据库文件为TESTIN.FRM、TEST.DEF及TESTX.DAT

```
B≫DEFINE TESTX
Do you wish to create a new form(T/N)?N
TESX.FRM does not exist-enter different name or End Program〈C/R〉:TESTIN

Do you wish to create a new definition file(Y/N)?N

Enter Filename or End Program 〈C/R〉:TEST

Do you wish a printed copy of the data definitions(Y/N)?N

Attribute summery of Data Base TESTX

Number of Fields Defined = 2
Record Size (Bytes) = 27
Done
```

第四节　打印数据库定义

使用DEFINE命令可以做到显示数据库的定义而不改变它们。下例中表明怎样在打印机上获得数据库TEST的定义清单。

```
B≫DEFINE TEST

This Data Base exists already
Choose Option:Describe(D), Re-define (R), End<C/R>?
Send output to printer(Y/N)  ?Y

Attribute summary of Data Base TEST

 1. NAME:AN,250,25,"     MISSING     "
 2. AGE:N,1,18,65,  "                 "

Record Size (Bytes) = 27
Total Records = 0
```

第十一章　数据库的重组织

本章讨论修改数据库的结构。内容有：向数据库中加入新数据项，从数据库中删除数据项及改变数据项的定义。

本章用到的rDBNS 命令有：

DEFINE　修改数据库的定义文件。

FORMAT　修改数据库的格式文件。

LIST　显示一个数据库中的记录。

READ　把外部造好的数据读到一个以前写好的数据库上。

REORG　对一个数据库进行重组织。它允许加入或删除数据字段及重新安排数据 字段的次序。

RUN　启动一个带指示句的命令过程。

SAVE　保存结果数据库。

WRITE　保存数据库数据，以便重新定义数据字段。

第一节　REORG命令应用

用REORG命令增加、删除或重排数据库上的数据字段。

REORG命令可以重构一个含有数据的数据库。REORG命令用格式文件来确定要 增 加或删除哪些数据字段。重构一个数据库，第一步就是要拷贝需重构数据库的格式文件，在此拷贝上进行必要的修改。然后，用REORG命令按照修改的“.FRM”文件形式重构数据库。然后，提示用户定义新的数据项。

如果你想建立结果数据库而不去改变原始数据，可使用〔$ $〕任选。

下面例子说明怎样用一系列命令重组数据库：

COPY TEMP.FRM = BUDGET.FRM〈C/R〉

FORMAT TEMP.FRM 〈C/R〉

（这是用户修改格式）

REORG BUDGET TEMP.FRM 〔$ $〕〈C/R〉

这里对加入的数据项进行录入数据定义。

LIST RESULT 〈C/R〉

核实结果。

SAVE BUDGET 〈C/R〉

若结果是所希望的，则保存结果。

放在如下命令过程里的一系列命令，打入命令RUN REORG〈C/R〉，系统会自动完成对一个数据库的重组织。

ED REORG.CMD〈C/R〉

I〈C/R〉

```
;Syntax:RUN REORG database<C/R>
*MESSAGE THIS PROCEDURE REORGANIZES THE $1 DATABASE<C/R>
*MESSAGE<C/R>
*MESSAGE IT USES A FILE CALLED"TEMP.FRM" DURING THE
    REORGANIZATION<C/R>
*MESSAGE<C/R>
*MESSAGE IF YOU ARE NOT SATISFIED WITH THE REORGANIZA-
    TION,<C/R>
*MESSAGE REPLY"R"TO THE PROMPT AFTER THE SAVE COMMAND
    <C/R>.
*MESSAGE<C/R>
*MESSAGE ARE YOU READY TO PROCEED(Y/N)? <C/R>
*GET $2<C/R>
*IF $2< >Y<C/R>
    *IF $2 = N<C/R>
    *ABORT<C/R>
    *ENDIF<C/R>
    *MESSAGE RESPONSE MUST BE Y OR N<C/R>
    *ABORT<C/R>
*ENDIF<C/R>
COPY TEMP.FRM=$1.FRM<C/R>
FORMAT TEMP.FRM<C/R>
REORG $1 TEMP.FRM [$$]<C/R>
LIST RESULT<C/R>
SAVE $1<C/R>
<control-Z>
E<C/R>
```

注意：由于美元符“$”出现在括号[ ]中，所以不必重复它们。

第二节 READ和WRITE命令应用

用READ,WRITE和DEFINE 命令修改数据项的定义。

有人在已经定义了一个数据库后，希望改变某些数据项的定义。例如想要使两个数据库中不相同的某些数据字段发生联系；或者创建数据库以后，由于有些原因，希望改变数据项的大小和类型。

如果你还没有录入任何数据，使用DEFINE命令就可改变原数据库的定义（如第十章所述）。如果你的数据库中已具有数据，而你又不希望再重新录入，则可以使用WRITE命令把数据临时保存在一个ASCII文件中，在数据库改变完成后，再使用READ命令将ASCII文件中保存的数据写到数据库的数据文件中。

下面的一系列命令将描述怎样完成重组织一个含有数据的数据库。

首先需要产生一个需要重组织的数据库的拷贝。然后使用WRITE命令将数据保存在一个ASCII文件中。此时，你可以使用DEFINE命令改变数据库中指定数据项的定义，在定义修改完毕后，再使用READ命令将数据写到临时拷贝数据库中。如果重定义成功，则将临时数据库的数据拷贝到重定义的数据库中。最后使用DESTROY命令清除READ和WRITE命令所用的临时数据库及ASCII文件。

```
COPY TEMP = BUDGET〈C/R〉
WRITE TEMP〈C/R〉
DEFINE TEMP〈C/R〉
```

根据要求修改数据定义。

```
READ TEMP〈C/R〉
```

注意：如果新的定义与旧的定义不兼容，则可能引起错误信息。如果修改了别的内容而没有改变“A”类与“AN”类数据项的大小，也可能发生这种情况。如果这种情况发生，你必须使用DEFINE命令来改正这些定义。

```
LIST TEMP〈C/R〉
```

按需要核实结果。

```
COPY BUDGET = TEMP〈C/R〉
```

若不是所希望的，用N响应提示。

```
DESTROY TEMP OK〈C/R〉
DESTROY $$$$.ASC〈C/R〉
```

将完成上述过程所用到的一系列命令放在一个命令过程文件REDEFINE.CMD中。如：

```
ED REDEFINE.CMD〈C/R〉
I〈C/R〉
;Syntax:RUN REDEFINE database〈C/R〉
*MESSAGE THIS PROCEDURE REDEFINES THE $1 DATABASE〈C/R〉
*MESSAGE 〈C/R〉
*MESSAGE IT USES A DATABASE CALLED "TEMP" DURING THE
   REDEFINITION〈C/R〉
*MESSAGE〈C/R〉
*MESSAGE IF YOU ARE NOT SATISFIED WITH THE REDEFINITION,
   〈C/R〉
*MESSAGE REPLY "N" TO THE PROMPT THE COPY COMMAND〈C/R〉
*MESSAGE〈C/R〉
*MESSAGE ARE YOU READY TO PROCEED (Y/N)? 〈C/R〉
*GET $2〈C/R〉
*IF $2〈 〉Y〈C/R〉
     *IF $2 = N〈C/R〉
        *ABORT〈C/R〉
     *ENDIF〈C/R〉
```

```
        *MESSAGE RESPONSE MUST BE Y  OR N<C/R>
        *ABORT<C/R>
    *ENDIF<C/R>
    COPY TEMP.FRM = $1.FRM<C/R>
    WRITE TEMP<C/R>
    DEFINE TEMP<C/R>
    EMPTY TEMP OK<C/R>
    READ TEMP<C/R>
    LIST TEMP<C/R>
    COPY $1 = TEMP<C/R>
    DESTROY TEMP OK<C/R>
    DESTROY $$$$.ASC OK<C/R>
    <control-Z>
    E<C/R>
```

录入下面命令即可执行此文件：

```
RUN REDEFINE <C/R>
```

注意：临时文件$$.ASC在命令文件中叫做$$$$. ASC。这是因为命令处理 程 序把一个“$”字符看成一个参数符号，除非$$.ASC写成$$$$.ASC，否则会产生错误。

第十二章　RUN命令处理程序细则

RUN命令处理程序处理批命令的执行。运行该程序可通过录入一个命令行，或在HELP屏幕中选择任一项完成。

本章用到的rDBMS命令有：

RESTART　继续处理一个被中断了的命令过程。

RUN　处理一个带有指示句的命令过程。

第一节　命令过程

命令过程允许在一个命令过程文件中分组存放许多命令。以后，通过打入RUN〈命令文件名〉.CMD可以检索和处理这个文件。另外，通过HELP屏幕也可以选择具有命令过程文件的项。

RUN命令处理程序中的指示句在命令过程处理中有极大的灵活性。其指示句描述如下：

*IF　根据符号参数的值，选择满足条件的一组命令。

*ENDIF　一组条件执行命令的结束标志。

*MESSAGE　在视频终端上显示信息。

*GET　在命令过程序处理期间允许录入符号参数值。

*LET　在命令过程语句中对符号参数进行赋值。

*ECHO　在命令过程处理期间禁止来自命令过程的信息。

*END　结束过程的处理。在*END后不再执行任何其他的语句。它与正常的程序或过程结束有一样的效果。

*ABORT　指示过程中止，不再处理任何命令。

RUN命令过程的执行，是由打入RUN〈过程名〉〈C/R〉实现的，当打入上述命令以后，计算机将检索名为〈过程名〉.CMD的过程文件。并根据文件中的指示句处理指定的命令。

这里引用的述语“处理”与执行的含义不同。处理过程是：命令处理程序创建一个名叫$$.DBM的工作文件。它将过程中的可执行命令写到工作文件中。执行时，逐条执行工作文件中的这些命令。命令处理程序扫描过程文件，同时，在扫描时完成下面的功能：

1．用RUN命令行中给出的实际值去替换RUN过程文件中的形式参数(符号参数)$$_1$，$$_2$，……$$_9$。该值是通过 *GET 或 *LET 指示句赋给的。启动时，用当前日期作为形参$TODAY的值。

2．当遇到*IF指示句时，测试指定的条件，并决定是否将*IF 和*ENDIF之间的命令行写入工作文件。仅当条件满足时，才能执行被写入工作文件的命令。

3．当命令处理程序遇到*MESSAGE指示句时，把信息显示在屏幕上。

4．当命令处理程序遇到*GET语句时，在显示屏幕上出现提示符：。在你录入参数值前，程序处理一直处于等待状态。此时你录入的值对所有以后程序中遇到的该参数都有效，

直到以后碰到指定同一个参数号的另一个*GET指示句。

5. 当命令处理程序碰到*LET指示句时，等号左边的参数被赋以等号右边的值。若参数出现在等号右边，也同样可以赋值。对于置入参数的缺省值，*LET指示句是很有用的。

6. 当命令处理程序遇到*END指示句时，处理结束，不再处理以后的命令行，而*END之前的命令被写入工作文件中，并被执行。

7. 当命令处理程序遇到*ABORT指示句时，处理也告结束，不把命令写入工作文件中，也不执行任何命令。

8. *ECHO指示句有在工作文件第一行写上SET ECHO的效果。如果要处理几个*ECHO指示句，*ECHO状态则是上次扫描过程文件处理的最后一个*ECHO状态。

9. 注释行以分号“；”开始。它不出现在工作文件中。

10. 处理程序的指示句不被写入工作文件中，因此在工作文件执行期间不产生条件测试。

第二节　命令过程实例

```
1)  ;Syntax:RUN SALEREPT start-date end-date
2)  ;     $1 IS THE BEGINNING DATE OF THE REPORT PERIOD
3)  ;     $2 IS THE ENDING DATE OF THE REPORT
4)  *LET$4 = N
5)  *MESSAGE THE REPORT WILL SHOW SALES FROM $1 TO $2
6)  *MESSAGE ARE THESE THE DATES YOU WANT TO USE(Y/N)?
7)  *GET $3
8)  *IF $3 NE Y
9)       *ABORT
10) *ENDIF
11) SELECT SALEDATA ST DATE GE $1 AND DATE LE $2
12) *MESSAGE DO YOU WANT TOTALS(Y/N)?
13) *GET $3
14) *IF $3 = Y
15)    *MESSAGE DO YOU WANT SUBTOTALS(Y/N)?
16)    *GET $4 = Y
17)    *IF $4
18)         *MESSAGE ENTER SUBTOTAL DATA ITEM
19)         *GET $5
20)         SORT RESULT BY $5
21)    *ENDIF
22) *ENDIF
23) *IF $3 = Y  AND $4 = Y
24)    PRINT RESULT BY REGION TOT SALES SUBTOTAL BY $5
```

```
25)       *END
26)    *ENDIF
27)    *IF $3 = Y
28)       PRINT RESULT BY REGION TOTAL SALES
29)       *END
30)    *ENDIF
31)    PRINT RESULT BY REGION SALES
```

第三节　实例分析

命令过程的第1行是注释行，它们不被写入工作文件中。第4行把参数置成N。如不给$4一个值，则第23行的*IF指示句将导致错误。在第16行中可以改变$4的值。

第5行和第6行显示两个日期信息。这两个日期用命令RUN SALEREPT 1/1/81,12/31/81格式录入。

第7行要求确认报表日期。

第8～10行检查用户的响应，如果响应不是Y，则本过程中止。

第11行把SELECT命令写入工作文件。

第12～13行决定报表中是否需要总计。注意这里再次用到参数$3。

如果需要做总计，在第15～16行中询问用户是否还需要小计。如果不需要总计，则命令处理程序跳到第31行，也确定了第14、23、27行上*IF指示句上的总计是不希望做的。下面二条命令被写入工作文件并执行之。

```
SELECT SALEDATA ST DATE GE 1/1/81 AND DATE  LE 6/1/81 〈C/〉
PRINT RESULT BY REGION SALES
```

如果，对13行上做总计的答复是Y，则在第15～16行上问用户是否要做小计。

如果需要做小计，则要满足第17行上的*IF指示句并且在第18～19行上对要求的数据项做小计（Subtot）。

一旦录入小计，则在第20行替换SORT命令中的数据项名“$5”，并把该指示句写入工作文件中。

如果第23行上的*IF指示句既要求总计，又要求小计，在做小计（SUBTOTAL）的数据项“$5”被替换以后，将第24行上的PRINT语句写入工作文件中。由于有第25行上的*END语句，而不再对命令文件做进一步的处理。

如果只需要做总计，则第23行上的*IF指示句不能满足，于是跳过第24～26行。若要满足第27行上的*IF指示句，则把28行上的PRINT语句写入工作文件。由于第29行上出现了*END指示句，则不再把31行上的PRINT写入工作文件。

注意：这里的行号仅为说明而引用，在命令过程文件中是不允许出现的。

第四节　命令过程的执行

命令过程处理完成后，开始执行写入工作文件中的每条命令。在工作文件中，没有指示句，也不允许出现某些条件执行的命令，只作有条件的处理。出现在工作文件中的命令都要被执行，除非是因为某种原因命令过程异常结束。这种异常结束可能由于下列原因产生：

1．在操作中打入〈control-C〉；

2．命令异常中止，操作员打入N确认删除数据库；

3．电源故障或发生硬件复位；

4．遇到无效命令。

第五节　命令过程的重启

如果发生了命令过程的异常中止，则在工作文件中保留被中止的那条命令以后的所有命令。这时，应检查一下命令过程，并标识那条作废的命令。然后用户决定是否要重做那条作废的命令。但重复执行APPEND，POST和READ命令可能引起非预期的结果。一旦决定了该过程的状态，则可用RESTART令重新使命令过程继续执行。如果工作文件不在当前驱动器上，则录入命令形如RESTART〈驱动器〉：，如果当前驱动器已被命令过程复位，则当前驱动器与命令过程作废时的驱动器不同。

第十三章　实用程序

实用程序在这里是不能归为任何其他类型的一些命令。这些实用程序的提供使得用户使用CONDOR SERIES/20 rDBMS更加方便。本rDBMS系统提供了四组实用程序：

1．数据库和文件实用程序

COPY　　拷贝数据库或文件。

RENAME 改变一个数据库文件的名字。

SAVE　　保存结果数据库。

2．数据库和文件目录实用程序

DIC　显示数据词典的登记项。

DIR　显示磁盘目录表中的文件清单。

3．系统启动实用程序

DATE　显示或录入日期。

TERM　设置视频终端特性参数。

4．系统操作实用程序

LOGDISK　登记更换的新磁盘。

SET　　设置rDBMS操作参数。

SYS　　退出rDBMS环境。

在附录A中，将要对每个实用程序详细讨论。

附录A　rDBMS 命令说明

一、命令说明和使用方法

数据库管理系统rDBMS是在操作系统支持下运行的，它由打入DBMS命令进行引导，将数据库管理程序装入内存。这个管理程序一旦被装入就一直驻留在内存中与磁盘操作系统不同的区域，对此管理程序所进行的初始化，还包括“记录启动日期（使用DATE命令）”及“设置终端工作方式和标识（使用TERM命令）”。

当你进入数据库环境时，你会看到CRT上显示出“X≫”，其中X指定为当前驱动器名（可以是A～H）中的任一个。例如，若提示为A≫时，表示当前驱动器为A驱动器，打入命令X：〈C/R〉，可使你随意选择某一驱动器作为当前驱动器。

二、命令句法及语法规则

1．{ }　花括号中的内容表示为任选项，它可以是字符、词或词组。当实际使用此任选时，只写括号中的内容而不写“{ }”。

2．〔 〕　方括号中的内容表示为任选项，可以是字符或词。当使用此任选时，必须连同〔 〕一起送入。

3．( )　圆括号中的内容只作为说明，不必送入。

4. / 斜杠表示列出的数据项只选择其中之一进入。但在J类数据项中与在COMPUTE命令中使用时例外。

5. UPPER CASE BOLD 大写黑体字按原样送入。

6. lower case bold 小写黑体字代表变量名，在实际使用时，由用户代换。

7. "BY"、"MATCHING"、"ST(使得)"、"USING"、"WHERE"和"WHOSE" 用于方便阅读，并不是必须的，但如果送入，必须按原样写出。

8. 某些命令行中，可以具有多达32个操作数。操作数可以是数据项名（用$d_1, d_2 \cdots d_{32}$表示），也可以是常数、定界符、逻辑运算符及算术运算符（如∧、∨、¬、+、-、*、/），而命令名、文件名不计为操作数。如：

SORT CLIENTS BY NAME, CITY, …～d_{32}

9. 数据项名必须与数据库格式文件中方括号[ ]的内容完全一致，并且可以缩写，可以带有"*"及"?"这些符号。但在使用COMPUTE命令时，必须将这些数据项完整地拼写出来，以免和算术运算符相混淆。

10. 命令行中定义的操作数的长度必须少于16个字符。中间嵌有空格的操作数，必须放在单引号或双引号内，以便作为一个操作数来翻译，在这种情况下，允许它的长度超过16个字符。例如"JONE SMITH"可作为一个操作数，但是，这个规则对ENTER或UPDATE命令是不适用的。另外，数据库标题（TITLE）的长度不得超过8个字符。

11. 就一条命令而言，其长度不得超过127个字符。

12. 大写、小写字母对于数据库的查找和比较操作都是相同的。

13. 当建立记录工作集和结果集（RESULT）文件时，需要带有扩展名$$.ext，如$$.DAT、$$.DBM、$$.DEF、$$.ASC、$$.FRM等。这些文件都建立在当前驱动器上。根据需要，可以选择其他驱动器作为当前驱动器。在命令过程中，名为$$.ext的格式文件名必须换成$$$$.ext，这样就不会把"$"符号解释为批处理参数。

14. 下列文件将作为系统文件出现，对这些文件的检索仅限于在A驱动器上进行，而不考虑当前驱动器。这些文件是：

DEF.DEF、DEF.CVL、DEF.FRM、DIC.DEF、DIC.FRM、DIR.DIR、DIR.FRM和DIR.DEF。以上这些文件总是驻留在A驱动器上。

命令语法

1. ABORT 停止命令SUBMIT和RUN的执行。
2. APPEND 将一个数据库的记录添加到另一个数据库上。
3. CHANGE 改变一个数据库的数据项值。
4. COMBINE 连接两个数据库的记录，并建立结果集。
5. COMPARE 根据指定条件（NOT）MATCHING，比较两个数据库的数据项值，并建立结果集。
6. COMPUTE 计算数据库数据项的值。
7. COPY 产生一个数据库或数据库文件的付本。
8. DATE 显示或录入日期。
9. DBMS 装入CONDOR SERIES/20 rDBMS系统。

10. DEFINE　建立或重定义或描述一个数据库。
11. DELETE　根据给定条件删除数据库的记录。
12. DESTROY　清除一个数据库或文件。
13. DIC　显示数据词典中的登记项。
14. DIR　显示指定驱动器中目录表文件清单。
15. DISPLAY　显示指定条件的数据库的记录。
16. EMPTY　清除一个数据库的所有记录。
17. ENTER　往一个定义好的数据库中写记录。
18. FORMAT　建立或修改一个格式屏幕（form screen）及辅助屏幕（help screen）。
19. HELP　辅助操作员选择过程。
20. JOIN　对两个数据库的数据项进行等值联接（自然联接）。
21. LIST　显示顺序存放的数据库记录。
22. LOG　在计算机上登记一个新磁盘。
23. POST　修改一个数据库中的数据项值。
24. PRINT　打印顺序存放的数据库记录。
25. PROJECT　建立结果数据库，其记录来自数据库中被指定的数据项。
26. READ　将一个 ASCII 文件传送到已经存在的数据库中。
27. RENAME　改变一个数据库或文件的名字。
28. REORG　对一个数据库的结构进行重组织。
29. RESTART　继续执行一个被中断的命令过程。
30. RUN　启动一个具有指示句的命令过程。
31. SAVE　保存结果数据库。
32. SELECT　根据指定条件筛选出满足条件的数据库记录，并建立一个结果数据库。
33. SET　设置rDBMS系统操作参数。
34. SORT　按照数据项值（可按升序排列或按降序排列）对数据库记录进行分类。
35. STAX　显示或打印输出数据项值的统计信息。
36. SUBMIT　启动并处理一个命令过程。
37. SYS　退出 rDBMS 环境。
38. TABULATE　汇总及打印指定的数据项。
39. TERM　指定系统终端标识。
40. TITLE　打印报表题头。
41. UPDATE　按照给定的条件改变数据库中满足条件的数据项值。
42. WRITE　把数据库的记录传送到 ASCII 文件中。

rDBMS 命令语法

ABORT
APPEND database1 database2
CHANGE database {ST d1 IS value1 AND d2 IS value2……}
COMBINE database1 database2
COMPARE database1 database2 {NOT} MATCHING d1,d2,……d32
COMPUTE database ST dr = d1 opr d2,…d31
COPY destination-database = source-datalase {OK}
COPY drive: = source-database/file {OK}
DATE {mm/dd/yy}
DBMS {license nnnnnn}
DEFINE database {*}
DELETE database {WHERE condition}
DESTROY database/file {OK}
DIC {drv:}

DIR {drv:}
DISPLAY database {WHERE condition}
EMPTY database {OK}
ENTER database [option]
FORMAT formfile {.FRM/.HLP}
HELP helpfile{.HLP}
JOIN database1 database2 MATCHING d1,d2,…d32 [option]
LIST database {BY d1,d2,…{AND COMPUTE statistic d3,d4,…}} [option]
POST database1 database2 MATCHING d1,d2,…opr d3,d4,…
PRINT datadase {BY d1,d2,…{AND COMPUTE statistic d3,d4,…}} [option]
PROJECT database BY d1,d2,…d32
READ database {filename.ASC}
RENAME newname oldname {OK}
REORG database {formfile.FRM} [option]
RESTART {drv:}
RUN procedure {.CMD} {param1,param2,...param9}
SAVE database/file {OK}

SELECT database {WHERE condition}
SET {function status}
SORT database BY d1,d2,...d32
STAX database BY d1,d2,..d32 [option]

SUBMIT procedure {.SUB} {param1,param2,...param9}
SYS or SYSTEM
TABULATE database {BY d1, d2,...{AND COMPUTE statistic d3, d4,...}}
[option]
TERM {id}
TITLE {'text' or "text",PAGE,DATE,SKIP,LINE,TOP,nn}
UPDATE database {WHERE condition}
WRITE database {filename.ASC}

CONDOR 命令详解

1. ABORT命令

功能：终止SUBMIT和RUN命令的执行。

格式：ABORT

使用说明：

此命令用来结束使用SUBMIT，HELP和RUN命令运行的应用程序。

2. APPEND命令

功能：将一个数据库的记录添加到另一个数据库上。

格式：APPEND database1/file1 database2/file2

使用举例：

```
APPEND B:PARENT C:DALLYINT
APPEND B:DEESET.DEF B:PARTDEF.DEF
APPEND CLIENTS.DAT $$.DAT
```

使用说明：

1）操作结果，database2的记录被添加到database1的记录上，database2本身没有改变。

2）做添加的两个数据库（或文件）必须具有相同的记录长度（字节数/记录）及它们在各自目录表中占有数据字段（field）的个数，否则，添加失败。

3）带有扩展名".DEF"、".DIC"和".DAT"的两个文件也可以进行添加，若没有扩展名，则认为该名就是数据库标题，并从数据词典中读取相应的数据词典文件。

4）在添加时，若出现磁盘空间不足或其他的磁盘故障时，将产生一个错误信息。同时APPEND执行中止。在database1中，没有被添加的记录。

5）如果要进行的操作不在当前驱动器上，必须在命令行中指明驱动器名。

3. CHANGE 命令

功能：改变一个数据库中数据项的值。

格式：CHANGE database {ST d1 IS value1 AND d2 IS value2...}

其中，value1,value2可以是常数或其他数据项值。

使用举例：

```
CHANGE CLIENTS ST DATE = 01/06/79
CHANGE CLIENTS ST STATUS IS 1 AND DATE IS 06/01/81
```

```
CHANGE ORDERS.ST COMPANY IS 'CONDOR CORP'
CHANGE CLIENTS ST STATUS2 = @STATUS1
```

使用说明：

1）按照命令行中给出的新值，改变数据库某个记录中一个或多个数据项的值。命令中的“IS”与“=”叫做动词，动词左边为欲改变的数据项（因变量），动词的右边为新的数据项值，在使用动词“IS”、“=”时两边要用空格隔开。

2）数据项可用同一记录内的其他数据项值来取代，这就要求动词右边数据项的第一个字符前面带一个“@”号，例4中动词右边出现“@”号用于区别数据项和常数。

例如：

```
CHANGE CLIENTS ST STATUS2 = @STATUS1
```

这个命令的意思为：将STATUS1的值赋给数据库CLIENTS每个记录中的STATUS2的值。在这里，要求STATUS1与STATUS2的数据定义必须相同（即要求它们是同长和同类的）。

3）按照给定的数据项名和值的列表，使用此命令可以一次完成改变16个数据项的值。两个数据项和值之间用“AND”分开。

4）使用该命令时必须小心，因为对于新值的上、下界是否合适并不进行编辑校验。例如：如果数据项COMPANY是4个字节字母数据字段，指定的改变量是值COMPANY IS CONDOR，则只有4个字符COND有效，且不显示出错信息。

4. COMBINE命令

功能：组合两个数据库的记录，并建立一个结果（RESULT）集。

格式：COMBINE database1/file1 database2/file2

结果数据库格式：$$.DAT、$$.DEF、$$.FRM

使用举例：

```
COMBINE B:CLIENTS C:CLIENTS
COMBINE B:CLIENTS.DAT C:CLIENTS.DAT
```

使用说明：

1）此命令产生两个数据库记录集合的并集，即建立一个包含database1和database2记录的新集，并且将database1中的格式文件及定义文件拷贝到结果集中。

2）在组合时，两数据库或文件必须具有相同的记录型（即数据定义相同，顺序相同）。否则将显示出错误信息，程序执行中止。

3）两个具有扩展名“.DEF”、“DIC”和“.DAT”的文件，只要它们具有相同的数据定义，就可以进行组合，只是组合的结果并不产生格式文件“.FRM”或定义文件“.DEF”。

4）一般地说，若在databasel中含有i个活动的记录，而在database2中含有j个活动的记录，组合的结果，将产生（i+j）个活动的记录。并将结果送到结果集上。那些不活动的记录则不能送到结果集上。

5）在命令执行期间，若发现目录表满或者磁盘空间不够，则在CRT上出现信息，同时停止命令的执行，没有记录写到结果集上。

5. COMPARE命令

功能：根据给定的条件，比较两数据库中指定数据项的值，并建立结果数据库。

格式：COMPARE databasel datalase2 {NOT} MATCHING d1,d2,...d32

结果集格式：$$.DAT、$$.DEF、$$.FRM

使用举例：

```
COMPARE ORDERS TRANSACT MATCHING ACCOUNT
COMPARE ORDERS SHIPPING USING ACCOUNT AND SHIPPER
COMPARE ORDERS OLDORDER NOT MATCHING ACCOUNT
COMPARE ORDERS OLDORDER ACCOUNT
```

使用说明：

1）此命令根据条件 MATCHING/NOT MATCHING 将 database1 的数据项与 database2相应数据项进行比较，并产生结果数据库。在语法上，“USING”、“BY”与“MATCHING”是同义的。

2)使用MATCHING条件时，产生databasel记录的结果数据库。其记录是在database1上指定的项与database2相匹配的记录。使用 NOT MATCHING 条件虽然也产生database1 记录的结果数据库，但其记录是在database1上指定的 项 与database2 不匹配的记录。

3）在例 1 中，根据条件MATCHING对数据库ORDER和TRANSACT上数据项名 ACCOUNT的值进行比较。选择满足条件MATCHING的那些ORDER的记 录，并把它们写到结果数据库中；在例 3 中，情况则相反，它选择满足条 件 NOT MATCHING 的那些 ORDER的记录，并把它们写到结果数据库中。注意：进行比较的数据项，必须具有相同的数据定义（即名字相同，长度相同，数据字段类型相同）。

4）对于命令：

```
COMPARE ACCOUNTS TRANSACT USING ACCOUNT
```

与

```
COMPARE TRANSACT ACCOUNTS USING ACCOUNT
```

执行的结果不一样。在第一条命令中，是将ACCOUNTS中与TRANSACT 中在ACCOUNT上相匹配的ACCOUNTS的记录写到结果数据库中；在第二条命 令中，是将TRANSCT中与 ACCOUNTS 中在ACCOUNT 上相匹配的TRANSACT 记录写到结果数据库上。

6. COMPUTE命令

功能：计算数据库中给定数据项的值。

格式：

```
COMPUTE database ST dr  =  d1 opr d2.....d31
COMPUTE database ST d1 opr d2.....d31 = dr
```

其中：$d_1, d_2, \ldots d_{32}$为操作数，它们可以是整常数或者是数据项。

dr为数据项名。

opr为算术运算符，它们可以是“+、-、*、/”之一。

在命令行中可以有32个数据项，包括结果数据项。

使用举例：

```
COMPUTE ORDERS ST SUBTOTAL  =  QUANTITY*PRICE
COMPUTE ORDERS ST PROFIT  =  PRICE-COST
COMPUTE ORDERS ST DISCOUNT  =  PRICE/4+12.00
```

使用说明：

此命令用来计算数据库中某数据项的值。并将结果存在数据库中每个记录的数值数据字段中。由于该命令会对整个数据库的记录产生影响，所以在使用此命令前，你应该尽量熟悉它的语法。这样，你才能收到良好结果。

1）计算的顺序

表达式左端是结果数据项名，并且在定义文件中已被定义过。表达式右端可以是常数、运算符和其他数据项名。对等号右端所进行的计算，是按照从左至右这一顺序进行的，它没有算符的优先级之分。因此，表达式右端的括号也可以去掉。如下的几个表达式计算结果是相同的：

```
PRICE=7-4*AMOUNT+2              (1)
PRICE=(7-4)*AMOUNT+2            (2)
PRICE=7-(4*AMOUNT)+2            (3)
PRICE=7-(4*AMOUNT+2)            (4)
PRICE=7-4*(AMOUNT+2             (5)
```

2）项的标识

每个表达式的项（term）由它们的第一个字符进行标识。若第一个字符是数字（0～9），则该项被认为是常数，并解释为整型量；若该项第一个字符为字母（A～Z，a～z）的话，则该项被认为是数据项，其值为整型量；若“*”作为字符出现在表达式的项中，则要求在数据库出现的地方要完整地将名字拼写出来；若在数据项名中嵌有空格或字符（+、-、=、/），则要求该数据项名必须用单引号(′)或双(引号″)括起来，作为单个数据项名。例如ENROLLED_DATE必须写成“ENROLLED-DATE”或‘ENROLLED-DATE’的形式。否则，系统会认为是两个数据项作减运算。通常，一个合法的表达式中应至少含有三个操作数、两个运算符号（其中包括等号）。

3）数据类型

在表达式中用于计算的数据项必须是“N、J、$”这三类的。其他的，如“A、AN”两类数据项则不能出现在COMPUTE命令中。

4）数字转换及溢出

本命令只限于进行整数运算，在进行除运算时，其小数部分被舍掉，只取其整数部分。如 9/10=0，　7/3=2

在正常情况下，参加运算的操作数都被规范化成四个字节，这就是说，不同大小的数据可以做算术运算。例如，一个两字节长的数可与一个一字节长的数相乘。另外，对乘除法运算有个限制，就是两个操作数之间做相乘或相除运算时，其中间结果不能大于四个字节。例如：在表达式A*B/C=D中，A*B必须小于等于四个字节，并要求A*B/C产生的结果仍不大于四个字节。如果计算结果比原先定义的结果要长的话，就会产生溢出，且不能将其结果保存起来。

5）日期类数据项

两个J类数据项可以进行相减运算，如给定两个日期期间的天数，你可以写：

```
COMPUTE TIME ST DAYS = FINISH-START
```

一个J类数据项也可以减去或加上一个天数。例如：

COMPUTE TIME ST FINISH = START+30

日期格式的数据不能作为常数录入，因为所有常数均假定为整数。如日期1/1/80是以J型数据存放的，写成010180也不能使该数据项用于计算。

6）美元类数据项

美元型数据项按美元值的100倍存放（即按美分存放）。它在整数后面跟有两位小数。例如对"$123.00"来说，应写成123.00而不能是123。所以，用户使用COMPUTE命令时要特别小心。如下的例子说明$类，N类数据项的正确使用。

$ = $ + $

$ = $ - $

$ = $ * $/1000

$ = $/$ *100

$ = $ *n

n= $ * $/1000（此种情况经常产生溢出）

n= $/$

7）COMPUTE命令的交互使用方式

COMPUTE 命令也可以采用交互方式，即使用时先打入：

COMPUTE database〈C/R〉

此时，系统会出现提示，要求你进入要计算的等式。在你输入等式后打入一个〈C/R〉，计算机自动计算输入的等式，并将结果保存起来。并再显示出提示，问你要下一个等式。如果你想结束计算，可打入E退出。

7．COPY命令

功能：拷贝一个数据库或文件。

格式：COPY destination-database/file = source-database/file {OK}

COPY driver: = database/file {OK}

使用举例：

COPY B:RESULT = C:RESULT

COPY B: = C:RESULT

COPY B:NEW.DAT C:CLIENTS.DAT

COPY DUPLCATE.DAT = CLIENTS.DAT

使用说明：

1）此命令既可以将数据库或文件从一个驱动器上复制到另一个驱动器上，也可以在同一个驱动器上用不同的名字产生源数据库或文件的付本。文件名和数据库名的区别在于它们是否有扩展名。

2）若某个数据库的标题存在，则它的格式文件、定义文件和数据文件均可做为源文件而产生它们的付本。对于源名与目的名相同时或是没有目的名（隐含与源名相同）的情况，拷贝结果在数据词典上产生一个与源文件同名的词典文件，不然的话，拷贝的文件名取目的文件名，但要求带有相应的扩展名。

3）若在目的驱动器上有一个数据词典的话，则录入一个相应于拷贝的数据库的文件名和标题的记录。若不存在这样的数据词典，则在拷贝时，自动建立一个数据词典文件(DATA.

DIC），并录入相应的新记录。

若在源驱动器不存在数据词典 DATA.DIC 时，则所有在命令行上出现的名字均认为是普通文件名。

4）若在目的驱动器上已经存在与被拷贝的数据库或文件同名的数据库或文件，除非用户给定认可“OK”请求删除，否则，原先存在的数据库或文件均被保留。

5）使用 COPY 命令时，不能使用表示任意字的通配符号。如“*”或“?”。

8．DATE 命令

功能：显示及录入日期。

格式：DATE {current-date}

使用举例：

```
DATE
DATE  4/1/79
DATE  4-1-79
DATE  040179
```

使用说明：

1）在第一种情况下，单纯打入 DATE 命令，系统可显示当前的日期，并提示你是否要进入新的日期，此时若单纯打入一个〈C/R〉则表示原先的日期不变（也即是当前日期）。若打入新值时表示代替原先的作为当前日期，这与在命令行中直接写上日期值是一样的，只是系统不产生提示。日期数据项的格式可以是 mm/dd/yy、mm-dd-yy 或 mmddyy。每一个 J 类数据项共占三个字节。

2）除 mmddyy 这种格式外，mm、dd 前面的零不用打入。

3）当一个新文件生成时，将把当前的日期存储在它的目录表中，你可使用文件目录列表（即 LIST CLIENT.DIR）显示或打印出数据库建立的日期。

9．DBMS 命令

功能：引导 CONDOR SERIES 20/rDBMS 系统，把 DBMS 装入内存。

格式：DBMS { LICENSE nnnnnn }

DBMS { LIC nnnnnn }

使用举例：

```
DBMS
DBMS  LICENSE  123456
DBMS  LIC  123456
```

使用说明：

1）此命令作为本数据库系统的主体，为保证 SERIES 20/rDBMS 命令的操作，它必须常驻在内存中，并且要占用很大的内存空间。有时用户在打入 DBMS 命令后，在CRT上出现指示内存空间不足的信息。因此，要运行此系统，计算机必须具有大于48K的内存（其中44K 为 DBMS 使用，6K 为支持它的操作系统使用）。

2）在装入 DBMS 以后，系统会出现提示，请求你打入一个 License Number（此号来源于 CONDOR 公司）。这时用户必须写入正确的号码，然后打入一个〈C/R〉。若单纯打入一个〈C/R〉，系统会认为无此许可号，在这种情况下，系统将限制用户的使用。在以后的

数据库操作中，既不能执行多于50个记录的操作，也不能删除数据库及其记录。除非你给出正确的License Number。例如：你在启动时没有给出正确的Licen-se Number，当你要打印一个具有120个记录的数据库文件时，系统会提示你录入此号，此时若给出正确的号码，程序则继续处理，否则，执行中止。当正确的号码给出后，它在下次装入DBMS以前一直有效。

3）本rDBMS系统的另一个特点，就是许多非rDBMS程序（实用程序及应用程序）都可以在数据库环境下运行。它不需要退回到操作系统环境中去执行。但这些程序只限于带有扩展名".COM"的应用程序。使用时，必须在程序名前加上一个前缀"$"。如$PIP、$DUMP等。前面说到，rDBMS系统要常驻内存大约为44K，在这44K中，有近32K内存是用来装入程序和数据的。如果实用程序本身超过32K，就必须退回到CP/M环境去执行。退出rDBMS环境时，打入SYS (SYSTEM)，而此后你若想重新进入rDBMS环境，你必须再次打入rDBMS命令进行系统装入。

4）有一种可在CP/M环境下进行启动、自动装入rDBMS系统以及进行初始化的小过程，名为START.SUB。其中包含有DBMS、DATE、TERM命令，使用SUBMIT.COM来执行这个文件。这个过程一旦被启动，就会自动地逐条执行其中的命令。

10. DEFINE命令

功能：建立一个新的数据库，重新定义一个数据库和描述一个数据库。

格式：DEFINE database {*}

使用举例：

```
DEFINE  CLIENTS
DEFINE  INVENTERY
DEFINE  QUICKDB  *
```

使用说明：

1）此命令建立一个数据库。包括建立一个定义文件、一个数据文件和一个格式文件，并将这些文件名，做为一个新记录写到数据词典中。

2）对于一个定义好的数据库，你可以在任何时刻修改编辑规范。如果数据库中没有记录存在，则可以使用重定义任选，否则，必须在定义文件中使用UPDATE命令。例如：

```
UPDATE CLIENTS.DEF WHERE FIELD IS 3 4 5
```

在执行时，仅修改它的最大值、最小值及缺省值。

3）被定义的数据项可以在任何时刻使用（D）任选项显示出来。此时，系统还提示结果是否需要打印。一个数据库要显示其数据定义时，并不要求该数据库的记录一定是空的。

4）下面详细说明此命令的使用：

打入DEFINE B:CLIENTS ‹C/R›

系统首先检索B驱动器是否有名为CLIENTS.FRM的文件。若没有找到这样的格式文件，则会产生两种提示信息。第一种信息提示你是否希望建立一个新的格式文件。回答Y时，系统就会自动装入FORMAT程序。回答N时；系统则请求你录入一个原先定义好的具有格式文件的数据库名（对于这个回答，要求不带扩展名）。第二种提示信息询问你是否希望建立一个定义文件。回答Y时，系统要求你提供数据定义说明；回答N时，系统请求你录入一个具有定义文件名的数据库名(不要带扩展名）。在这里，要求提供的数据库格式文件与定

义文件中数据项个数必须是相同的。否则出现错误信息。

如果你在命令行中使用了“*”号，则隐含着所有数据项均被定义为AN类。不过，在写定义文件之前，用户可以根据需要把它们改成其他的类型。有效的数据类型有：

A　字母文字类。如NAME

AN　字母数字类。如CP31

N　数值数字类。如957

$　美元类。如$123.00

J　日期类。如10/15/79

打入以上类型的一个代理字母，然后打入一个〈C/R〉，可完成对某个数据项的数据定义，并且系统会自动提供一下个要定义的数据项。当最后一数据项定义完成后，系统会出现{{END}}，表示数据项定义结束，并询问以上的定义是否正确，是否要修改；打入Y表示以上定义正确，将其写入定义文件中；打入N时表示允许编辑定义。在这种情况下，首先显示出原先对每个数据项的定义；选择哪些是可满足的，哪些不是。对那些满足的定义，只须打入一个〈C/R〉跳过去，系统会继续提供下一项。如你觉得该项不满意，可使用〈control-H〉将光标左移到需要被编辑的地方。打入新值，然后打入一个〈C/R〉。重复执行上述操作，直至{{END}}再次出现。此时你可打入Y表示定义OK，并将该定义文件存储在当前驱动器上。

在定义被存后不久，系统显示出提示信息，询问是否需要对被描述的数据库进行说明。打入Y表示需要，接着提示你是否将这些定义和说明打印出来，并给出任选（Y）——送交打印机打印或（N）——送交CRT上显示。

5）一个没有记录的空数据库可以使用重新定义任选进行重定义。若一个有记录的数据库需要重新定义，则可使用WRITE命令将其记录保存起来，等到重定义完成后，再使用READ命令恢复原来的记录。若你想在一个数据库中增加或删除某些数据字段，而又不改变原来字段的定义，则可使用REORG命令。

6）使用DEFINE命令和选择重定义任选，都可以对一个数据库进行重定义。在定义方式下，首先显示以前的定义，然后允许你们对它们进行编辑。为了编辑指定的数据项，需要先录入一个“}”（花括号）。此时，系统将出现提示符“>”，录入要改变的数据项名，后跟一个〈C/R〉。打入另一个花括号“}”，结束这种编辑方式。

11. DELETE命令

功能：根据给定的条件，清除数据库中符合条件的那些记录。

格式：DELETE database { WHERE condition }

其中：condition是由一个或多个由AND及OR连接起来的比较词。注意，在一个命令行上，AND及OR不能同时出现在一起。否则认为它具有二义性。一个比较词是一个用一个或多个数据项值进行比较的短语。例如：

（数据项）	（比较动词）	值
data-item	comparision-verb	value1, value2……
STATE	IS	MI, IN

使用举例：

```
DELETE  CLIENTS  WHERE DATE IS 1/1/79
DELETE  CLIENTS  ST DATE IS GT 5/1/78
```

DELETE CLIENTS ST ACTIVITY IS TERMINATED

比较动词将给出的值（一个或多个，用逗号隔开）与原来的数据项的值进行比较，检查该条件是否被满足。下面给出本 rDBMS 使用的比较动词（comparision-verb）：

(VERB) 动词	(SYNONYM) 同义词
=	EQ, IS, EQUALS, ARE
<>	NE, IS NOT, ARE NOT
<	LT, IS LT, IS NOT GE
>	GT, IS GT, IS NOT LE
<=	LE, IS LE, IS NOT GT
>=	GE, IS GE, IS NOT LT

比较所用的值可以是常数或其他数据项的值。在数据项名前放一个@（at sign）号，表示是数据项值而不是数据项值。例如：

SHIP.ADDRESS IS @BILL.ADDRESS

另外，在命令行中使用逗号所划分的数值，与使用短语“OR data-item IS”效果相同。例如：

SELECT CLIENTS WHERE STATUS IS MI, IN

与

SELECT CLIENTS WHERE STATUS IS MI OR STATUS IS IN

执行结果是一样的。

使用说明：

全局影响：

当删除一个记录时，则该记录在文件中处于不活动状态。在用户看来仿佛不存在于文件之中。实际上，删除一个记录并不改变文件目录表中记录的计数。这个记录个数表示活动记录个数与不活动记录个数的和。如果你要清除这些不活动的记录，可使用 SORT 命令。

12. DESTROY 命令

功能：清除一个数据库或文件。

格式：DESTROY database/file { OK }

使用举例：

```
DESTROY  CLIENTS
DESTROY  PARTS OK
DESTROY  PARTS.DEF
```

使用说明：

1）此命令用来请求永久地清除一个数据库的所有文件以及它们在数据词典中的标题。若数据库的标题与组成数据库的文件名不同（如数据文件名、定义文件名或格式文件名），则系统出现提示，询问你是否同意删除。如果你回答 Y，该数据库或文件就被删除。这与在命令行中给定确认 OK 效果相同，只是不出现提示信息。

2）使用 DESTROY 命令，可以在 rDBMS 环境下删除一些其他文件（要写出扩展名），它与 CP/M 的 ERA 命令相当。如果你打入 DESTROY *.* 〈C/R〉，则当前驱动器上所有文件都会被删除。注意：对于一个被删除了的数据库或文件，其数据无法恢复。如果

在指定的驱动器上没有数据词典，则文件名作为自变量，而忽略扩展名。

13. DIC 命令

功能：显示在数据词典中的登记项。

格式：DIC {drv:}

使用举例：

```
DIC
DIC B:
DIC C:
```

使用说明：

此命令将列出任何一个驱动器上数据词典的登记项。若没有指定驱动器名，则假定为当前驱动器。列表将以四列形式出现，即：

标题	数据集	定义集	格式集
TITLE	database-set	definition-set	form-set

而且，数据词典的格式也可以使用命令

```
LIST DATA.DIC BY TITLE, DATA, DEFINITION, FORM
```

来显示。

14. DIR 命令

功能：列出磁盘目录表中的文件。

格式：DIR {drv:}

使用举例：

```
DIR
DIR B:
DIR C:*.DAT
DIR CLIENTS.*
```

使用说明：

DIR 命令可以显示指定驱动器的全部文件目录的列表，也可以显示与某些名字相符的文件。这里可使用“*”及“?”这两个通配符。如果用“*”作为通配符，指定名后面跟以任意字母的名均符合受限条件。如果用“?”作通配符，则只在“?”位置为任意字符的名才符合受限条件。如打入：

DIR CL?ENTS.*，将显示出：

```
CLIENTS.DAT
CLIENTS.DEF
CLYENTS.DAT
CLYENTS.DEF
```

15. DISPLAY 命令

功能：显示一个数据库中满足指定条件的记录。

格式：DISPLAY database { WHERE condition }

使用举例：

```
DISPLAY CLIENTS WHERE NAME IS SMI*
```

```
DISPLAY CLIENTS WHERE STATE IS MI
DISPLAY DEFSET.DEF ST FIELD IS 1 2 4
DISPLAY DATA.DIC WHERE TITLE IS CLIENTS
DISPLAY CLIENTS ST ENROLL.DATE GT @TERM.DATE
```

其中，条件 condition 是由一个或多个由 AND 或 OR 连接起来的比较词（在一个命令行中 AND 与 OR 不能同时出现，否则认为具有二义性）。一个比较词用一个或多个值对一个指定数据项进行比较，例如：

数据项	比较动词	值
(data-item)	(comparision-verb)	(value1, value2, ……
实例：STATE	IS	MI, IN

比较动词用一个或多个数值对数据项进行比较，检查所给的条件是否得到满足，本 rDBMS 系统给出的比较动词为：

动词	同义词
=	EQ,IS,ARE,EQUALS
<>	NE,IS NOT,ARE NOT
<	LT,IS LT,IS NOT GE
>	GT,IS GT,IS NOT LE
<=	LE,IS LE,IS NOT GT
>=	GE,IS GE,IS NOT LT

比较所用的值可以是常数，也可以是其他数据项的值。数据项名前放一个@号，表示它是数据项值而不是数据项名。例如：

SHIP. ADDRESS = @ BILL. ADDRESS。另外，在命令行中使用逗号所划分的数值与使用短语 “OR data-item IS” 的效果是一样的。例如：

```
DISPLAY CLIENTS WHERE STATE IS MI, IN
```

与

```
DISPLAY CLIENTS WHERE STATE IS MI OR STATE IS IN
```

执行结果是一样的。

使用说明：

1）通常，使用 DISPLAY 命令来查询数据库的某一个记录，或查询满足给定条件的一组记录。例如，你在命令行中指定一定的条件，计算机根据条件检索数据库的所有记录，若找到满足条件的记录，就将其显示在控制台屏幕上。若想打印，可以使用〔P〕任选项。用户可以给定条件继续检索其他记录或停止检索。但你却不能用 DISPLAY 命令修改记录。

2）在一个命令行关系句中，可以指定多个检索条件，并且，一个比较词中可以具有多个值，这些值可以使用逗号或空格隔开。它与短语 “OR data-item IS” 的功能是一样的。例如：比较词：AGE IS 10, 12, 14 或 AGE IS 10 12 14，它可以写成：

```
AGE IS 10 OR AGE IS 12 OR AGE IS 14
```

在同一个命令行中同时出现 OR 与 AND，将被认为具有二义性。例如子句：

AGE IS 10 20 AND SEX IS MALE 是二义的，它可解释为：检索10岁的男性或20岁的男性；也可以按另一种方法解释，即检索10岁的人（不论性别）或20岁的男性。

使用 DISPLAY 进行显示的也可以是数据库定义文件和数据词典文件以及数据文件的目录。但要显示的文件必须要有一个扩展名。如“.DIR”、“.DIC”、“.DEF。”

16. EMPTY 命令

功能：清除一个数据库中的所有记录。

格式：EMPTY database {OK}

使用举例：

```
EMPTY TRANSACT
EMPTY B:CLIENTS OK
```

使用说明：

1）此命令清除一个数据库的所有记录，并将目录表的记录记数器置为零。被清除的记录是不可恢复的。用这条命令既可以清除指定数据库名的记录，又可以清除指定文件名的记录。若要清除的数据库或文件不在当前驱动器上，必须在命令行上指定驱动器名和数据库名。

2）当你打入 EMPTY database/file〈C/R〉时，系统会询问你是否同意清除其中的记录，打入 Y 表示同意。同样，你也可以在批处理过程中在数据库名后面加上OK，以确认删除。

17. ENTER 命令

功能：录入数据到一个新定义的数据库，或在一个已有数据的数据库中插入记录。

格式：ENTER database 〔option〕

其中任选项分别表示为：

〔R〕 表示设置自动重复方式

〔D〕 表示设置自动缺省方式

使用举例：

```
ENTER CLIENTS
ENTER CLIENTS. DEF 〔R〕
ENTER DATA. DIC
```

使用说明：

1）一旦数据录入到数据字段内，就打入一个〈C/R〉，则光标指向下一个数据字段的第一个字符的位置。若录入数据的长度大于或等于该字段的长度，光标会自动指向下一个数据段字的位置。即系统提供一个自动〈C/R〉功能，用户此时不必再打入〈C/R〉。录入〈control-H〉或〈control-J〉可使光标往回移动，而录入〈control-L〉或〈control-K〉可使光标向前移动。在录入方式下，系统要对一致性进行检查，即检查录入的数据与数据定义文件中所规定的编辑标准是否一致？当你录入的值不在给定的定义范围或与指定的数据类型不符的话，控制台将自动地在屏幕上显示出错误信息。同时光标自动恢复到这个数据段的第一个字符的位置。此时用户必须改正错误，然后才能继续录入其他的数据。对于范围出错，可在打入〈C/R〉之前打入一个〈control-O〉跳过此数据段。在标准数据录入方式下（即不设置选择开关），若你在当前的字段上，只打入〈C/R〉，并且此时在数据定义文件中缺省值非空时，则缺省值会自动录入到该字段上，同时光标指向下一个字段的位置。

2）除标准录入方式以外，还有两种数据录入方式，即自动重复方式（auto-repeat mode）和自动缺省方式（auto-default mode）。当设置自动缺省方式时，光标移到要录入

数据的位置。打入〈C/R〉，以前在定义文件中预定义的缺省值会自动地被录入，不需操作员输入（假定缺省值非空），光标会自动移到下一个字段的位置。如果缺省值为空，打入〈C/R〉后光标仍保持在原来的位置上。打入〈control-H〉可使光标往回移动到前面要修改的数据字段。当设置自动重复方式时，允许前一个记录中录入的值在后一个记录中使用，而不用重新打入。在选择继续工作任选 C 时，不清除输入格式屏幕。

3）用户可以通过控制键〈control-D〉设置自动缺省方式 ON 或 OFF；通过〈control-R〉设置自动重复方式 ON 或 OFF。这些方式也可以通过调用 ENTER 程序，然后设置任选开关为 ON。

4）用户若想立即结束当前数据录入状态，且不需保留当前录入的数据记录，可打入〈control-C〉。在这种情况下，会使当前录入的记录丢失，但在此之前录入的记录均被保留下来。通常用户可选择 SAVE-END(E)任选作为 ENTER 子程序的出口。

5）若你录入数据后出现软盘上没有可用空间或数据库满时，应该在一张新盘上开始录入新数据库。为使记录型与已满的数据库一致，可使用 DEFINE 命令定义一个新数据库，其格式文件及定义文件都与以前定义的数据库相同。

6）当给出“NO SPACE LEFT ON DISKETTE”信息时，最后录入的记录已经丢失。因而首先应该将该记录写到新软盘所定义的数据库上。对于一个数据库来说，最多能容纳 32767 个记录。

18. FORMAT 命令

功能：建立或修改一个格式文件或辅助屏幕文件。

格式：FORMAT filename {.FRM}or

FORMAT filename {.HLP}

使用举例：

```
FORMAT NEWFORM. FRM
FORMAT NEWFORM
FORMAT C:OLDFORM. FRM
FORMAT HELPFLE. HLP
```

使用说明：

1）为了检索数据和录入，FORMAT 命令允许用户建立和修改一个已经存在的 FORMAT 格式，FORMAT 命令与 DEFINE 命令合用，可完成对一个数据库的定义。

2）FORMAT 程序具有两种编辑方式：即替换方式(replace mode)和插入方式(insert mode)。对于替换方式来说是缺省方式，它允许用户进行满屏编辑输入(与光标配合使用)。

3）替换方式及插入方式之间可以通过〈control-A〉进行转换。另外，使用光标控制键〈control-H〉、〈control-J〉、〈control-K〉和〈control-L〉可使光标分别向左、向下、向上和向右移动。

4）在插入方式下，打入的任何字符被插入到光标所在的位置上。该行右端的符号串向右移动，且光标向右移动。打入〈control-D〉（或打入 DEL 或 RUBORT），则可以删除光标所在位置上具有的字符。并且，该行右端的符号串向左移动。打入〈C/R〉则插入一新行，打入〈control-X〉则删除光标所在的行。

5）在替换方式及插入方式中，还有一些任选项和提示，它们的功能和使用方法用户可

在实际使用中去体会。

6）若指定的文件不存在，则在屏幕中出现信息“NEW FILE”，否则，对现存的文件进行编辑。用户在FORMAT的处理期间，随时都可决定是否保留原来的结构不被修改。若你还想保留修改前的内容，可使用COPY命令，并使用一个新的名字产生它的一个副本。

7）当你使用FORMAT要对一个新文件进行编辑时，除在屏幕的底端有一行提示行外，整个屏幕为空。你可以根据意愿在屏幕上与光标合用进行满屏编辑。这里注意：数据项名必须括在方括号内，后跟若干条短线。如〔ACCOUNT〕----，注解行及信息可以在屏幕上显示。这些信息可根据需要写出，但不可用方括号括起来。

8）如果要对某一格式进行修改，只需将光标移到相应的位置上，打入新的字符（或空格）去覆盖原先的字符信息（或空格）。

9）当一个编辑好的格式文件被存起来后，系统会提示用户是否希望定义一个新的数据库。回答Y时，系统会自动调用并运行DEFINE程序。

10）使用FORMAT命令可以建立一个辅助文件（HELP文件），方法与上述定义格式文件相同。只是文件名后必须跟有扩展名“.HLP”。

11）使用FORMAT程序建立的文件不能带有〈CR-LF〉字符。

注意：如不指明驱动器，则新文件将建立在当前驱动器上。FORMAT命令只能对带有扩展名“.FRM”及“.HLP”的文件进行编辑。若这些文件不在当前驱动器，必须指定驱动器名。

使用UPDATE命令可以改变数据文件。

19. HELP命令

功能：辅助操作员选择过程。

格式：HELP {helpfile} {.HLP}

使用举例：

```
HELP
HELP BILLING
```

使用说明：

1）此命令帮助用户从控制台的菜单上选择并启动一个过程或程序。此命令对于那些不了解数据库命令语法规则、数据库名及数据项名的用户来说，是很有用的。

2）功能上，HELP命令是一个带有扩展名“HLP”的辅助文件，并将其显示在控制台上，若仅使用HELP而不指定操作数，则隐含指出系统缺省文件名SYSTEM.HLP。

3）一个HELP文件是使用文本编辑程序（ED.COM）或数据库的FORMAT.DBM程序进行编辑的。如果使用FORMAT程序，它一定要使用扩展名“.HLP”。HELP文件的格式为：

```
Description1〔Command Line1〕
Description 2〔Command Line2〕
.....
Description n〔Command Linen〕
```

注意：命令行是括在方括号中的，在使用时并不显示出来，显示的只是这个文件的说明

部分。打入数字 n（n≥1），则系统将选择第 n 组中相应的语句，并根据给出的实际命令语法启动方括号内的命令语句。在一个 HELP 文件中，可以选择127个语句（包括说明部分及命令部分）。而且一个 HELP 程序可以调用另一个 HELP 程序或命令过程。并且可以调用具有扩展名“.COM”及“.DBM”的程序（对带有扩展名“COM”的程序，调用时须使用“$”作为前缀。如 HELP $ED〈C/R〉。注意：HELP 命令不可以调用 rDBMS 内部命令，如 DIC、DIR、LOG、SET、RESTART 和 SYS。但可调用含有这些命令的过程。

4）以下例作为 HELP 命令处理财会的实例，其中调用文件名为 ACCOUNT. HLP

HELP ACCOUNTS

①Review of accounts on file 〔PRINT ACCOUNT BY NAME,ADDRESS〕

②Accounts receivable 〔RUN ACCREC〕

③Accounts payable 〔HELP PAYABLES〕

④Summary of Accounts by state 〔TABULATE ACCOUNT BY STATE〕

注意：这个 HELP 文件能够调用另外一个 HELP 文件，如可以调用 PAYABLES.HLP 文件。

5）在 HELP 程序中的命令行上，可以使用 RUN 命令，利用这一特性，用户可以很方便地执行所扩充的命令过程。全部参数可以在命令过程中赋值，并可以建立 HELP 屏幕链。

20. JOIN 命令

功能：按照指定的数据项值联接两个数据库的数据项。

格式：JOIN database1 database2 MATCHING d1,d2,……d32 〔option〕

任选中，若 database1 中设有相应的匹配记录时，将 database2 的缺省值存入结果数据库中。

结果数据库格式：$$.DAT、$$.DEF、$$.FRM。

使用举例：

```
JOIN ACCOUNTS MATCHING ACCOUNT-NO
JOIN ACCOUNTS BY ACCOUNT-NO 〔D〕
JOIN PRODUCTS PRICES BY ITEM-NUMBER
```

使用说明：

1）此命令建立一个新的数据库，它包括两个数据库（database1，database2）的数据项。每当 database1 中指定的数据项与 database 2 的数据项值相匹配时，就产生一个结果记录，并将其存于 database1 中。进行匹配的特定数据项是在命令行中指定的，它们通常被称为是“公共数据项（common data-item）”。要求联接的两数据库中的公共数据项必须有相同的定义。即两个数据项必须具有相同的属性、长度和类型。

2）两个数据库联接的结果，产生一个结果集，连同结果集中的一个格式文件和一个定义文件。定义文件中具有两个数据库的标号和数据定义。结果集记录的数据项顺序为：来自 database1 的数据项在前；来自 database 2 的数据项在后。由于公共数据项对 database2 来说是冗余的，且它在匹配时已经被合并了，于是结果集中每个记录应具有 d_1+d_2-K 个数据项和 $1+R_1+R_2-C$ 个字节。这里 d_1，d_2 分别是 database 1 和 database2 数据项的总数，K 是在命令行中指定的公共数据项个数，R_1 和 R_2 分别是 databasl 和 database 2 数据字段的总长

度，C 是公共数据项数据字段的总长度(字节数)。所以以上这类联接也被称为“自然联接”，也就是进行数据项的等值联接，清除冗余的数据值。一般地说，若在 database1 中出现的公共数据项有 n 个值，在 database2 中有 m 个值，那么，联接的结果将产生 m*n 个结果记录。

3）在命令行中使用〔D〕任选项，可以产生一类特殊的联接。在这种情况下，如果在两个数据库的任何记录之间都没有匹配的数据项出现，则强行把 database2 缺省值中不匹配的 database1 的记录写到结果集中去。

4）JOIN 命令限制结果记录最大长度不超过1024个字节，最多127个数据项、32767个记录以及有一个 CRT 屏幕。结果集的格式可以自动地被压缩和编辑。使得它只包含标号及输入格式下的短线（详见 PROJECT 命令）。

21. LIST 命令

功能：顺序显示一个数据库中的记录。

格式：LIST database/file {BY d1,d2,…{AND COMPUTE statistic d3, d4…}} 〔option〕

其中，任选项可是〔A〕,〔P〕,〔X〕；统计值 statistic 可是 TOTAL(TOT),SUBTOTAL (SUBTOT),MIX,MAX, AVERAGE(AVE、AVG), STAX 等。另外，compute可以简写为“{”或“@”。任选项分别表示为：

〔A〕 累加小计

〔P〕 送交打印机打印

〔X〕 在列格式中，清除标记头

使用举例：

```
LIST CLIENTS
LIST CLIENTS BY NAME,AND CITY AND STATE
LIST CLIENTS NAME,AND CITY AND STATE
LIST CLIENTS BY NAME AND COMPUTE TOTAL WAGE,HOURS
LIST CLIENTS NAME { TOT WAGE SUBTOTAL USING DEPT
LIST CLIENTS.DEF
LIST CLIENTS.FRM
LIST REPORT.SUB
```

使用说明：

使用 LIST 命令可以产生多种输出格式。在命令行中，数据项可以使用逗号、一个或多个空格、或 AND（两边用空格隔开）作为分隔。使用任选项〔P〕可以在打印机上获得所需要的信息（详见 PRINT 命令）。

LIST 命令可有四种输出格式，即：屏幕格式、列格式、带统计的列格式和ASCII码格式。

1）屏幕格式

这种格式可由 LIST database 语句产生。它从该数据库的数据文件的第一个记录起每次显示一个记录。你可以根据提示的任选，继续选择下一个记录或前一个记录。另外，你可以使用 TITLE 命令设置空间（详见 TITLE 命令）。你也可以使用 LIST 命令显示或打印数据词典、数据库目录和数据定义文件。这些文件要求录入它们的扩展名。如：

```
LIST DATA.DIC
```

LIST CLIENTS.DIR
LIST CLIENTS.DEF

2）列格式

这种格式产生一个仅包含指定数据项记录的列表。对于命令

LIST CLIENTS BY NAME STATE AND PHONE

来说将产生三列的记录列表。每一列的标题(数据项名)分别使用NAME,STATE和PHONE。其中标号名必须与数据库格式文件中的数据项名一致。使用〔X〕任选项，可以清除标题。用户可以按任意顺序指定多达32个数据项。每列赋予打印字符数等于1加上定义中数据项后跟短线的数目。或者加上标号长度(取其中之较大者)。例如：如果属性名后面具有23条短线，则可以指定24个打印位置；如果某属性后仅具有2条短线，则可以指定6个字符的打印位置。这是标号本身的长度。

3）带统计的列格式

这种格式能提供某些数据项的统计信息。如（TOTAL），它们被显示或打印在列表的最下边，并且还有在水平间断位置上给出小计（SUBTOTAL）。对于那个做小计的数据项，事先必须经过分类（分类可按某项值的升序或降序排列）。如果没有经过分类而求其小计，则会给出错误信息，指出数据库没按逻辑顺序排列。在这种情况下，必须使用SORT命令进行分类（见SORT命令）。用户在命令行中可以对每个统计项指定多个数据项或指定多个统计项。但是一个统计项后面不能立即跟有另一个统计项。如：

COMPUTE TOTAL WAGE,HOURS AND AVE WEEKS,HOURS 是可接受的，而

COMPUTE TOTAL, AVERAGE HOURS 却是错误的。数据项的打印字符数是在格式2中决定的。对带有TOTAL的列，至少要给出13个字符的打印位置。在格式1，2，3（屏幕格式、列格式及带统计的列格式）中，数字、美元及日期类型的数据项，在显示（或打印）时是按右端对齐的，左边用空格补充，而字母和字母数字类型的数据项是按左边对齐，右边用空格补充。

若当屏幕中出现“……”和”＊＊＊＊”这样的信息，说明输出有错误，前一种说明数据项后跟短线数不够；后一种指出用TOTAL、SOBTOTAL及AVG（或AVE）这样的统计任选时，产生了溢出（规定最大允许10位数字）。

4）ASCII码格式

这种格式列出一个ASCII文件（例如正文文件、命令文件、格式文件及HELP文件）。在这种情况下，不作解释，但对标记字符作扩展。

LIST MENU.HLP
LIST START.SUB
LIST LEDGER.CMD
LIST LEDGER.FRM

除带扩展名“.COM”、“.CVL”、“.DBM”、“.DEF”、“.DIR”、“.DAT”或“.IDX”的文件外，所有其他的文件均可做为ASCII列表文件。

22. LOG命令

功能：在计算机上登记一个新磁盘。

格式：LOGDISK 或
LOG DISK

使用举列：LOGDISK
LOG DISK

使用说明：

1）每当在一个驱动器上更换一个新磁盘时，必须打入 LOGDISK 命令。它的功能等效于 CP/M 中的磁盘复位功能（〈control-C〉）。如果你在某个驱动器上更换了一张磁盘，并且没有对该磁盘进行登记，则这张盘是只读的。如果企图对这种未登记的新磁盘进行写操作，就会发生写错误。

2）在 rDBMS 环境下，打入〈control-C〉不能对磁盘进行复位。

23. POST 命令

功能：用一个数据库的数据项修改另一个数据库的数据项。

格式：POST database1 database2 MATCHING d1, d2, …{opr d3, d4, …opr d5, d6, …d32}

其中opr 可以是以下列出操作符之一：
REP,REPLACE,ADD,SUM,SUB,SUBTRACT

结果集格式：$$.DAT、$$.DEF、$$.FRM

使用举例：

POST INVENTRY TRANSACT MATCHING PARTNO AND ADD QUANTITY AND REP DATE

POST ACCOUNTS RECEIPTS USING ACCTNO, ADD AMOUNT, QUANTITY

POST ACCOUNTS RECEIPTS ACCTNO ADD ORDERS SUB STOCK REP DATE

使用说明：

1）在命令行中，可以引用多达32个数据项，只要这些数据项存在命令行中所指出的两个数据库中。如果引用这些数据项，还要求它们具有相同的定义（即在两个数库中有相同的数据字段名、相同的类型及相同的长度等）。在一个命令行中，对每一个操作符“opr”后所列出的数据项都执行同样的操作，直到遇到一个新的操作符为止。例如命令：

POST ACCOUNTS RECEIPTS BY PARTNO AND REPLACE DATE, PAYMENT

与

POST ACCOUNTS RECEIPTS BY PARTNO AND REPLACE DATE, REPLACE PAYMENT

执行后结果是相同的，也就是说，所使用的逗号及词“AND”是任选的。在命令行中可以指定任何属于两个数据库的数据项。但是只对于 N 类及 $ 类数据项才可以在前面带有算术运算符（如 ADD）。REPLACE 操作符对所有类型的数据项都适用。如果对非数值类的数据项进行了无效操作，就不作任何更新，同时程序执行失败。做更新的记录是由数据库之间所规定的匹配关系所决定的。每当出现一个匹配的记录时，就将 database1 中的那个记录拷贝

到结果集的 $$.DAT 文件中去。然后，按照命令行中指定操作符去更新 database1 中的匹配记录。在操作时，如果任何一个字段产生溢出，则不更新 database1 的记录，也不写记录到 $$.DAT 中去，而在控制台屏幕上显示一个溢出信息，请求操作员干预。操作员可选择中止该程序操作或是跳过不对该记录的更新操作。如果选择了中止程序执行任选，则不产生结果集。

注意：如果以前所作的更新没有错误，数据库 database1 中匹配部分会进行更新。

2）若在命令行中没有指定数据项或是操作符时，则执行一个完全替换操作。它是一个缺省操作。例如：对于

POST ACCOUNTS RECEIPTS USING ACCTNO

会产生缺省操作。

AND REPLACE d_1, d_2, …

其中 d_1, d_2…为 database2 的全部数据项。

注意：对于以上操作，要求两个数据库的数据项之间必须满足这样的条件。即是：对 database2中所有数据项名均必须在 database1中有同名的数据项。如果该条件不真，则程序中止。

3）结果数据库对维护账底清单及核对 database2 中全部更新了的记录提供了一种简便的方法。这里结果数据库的 $$.DEF、$$.FRM 与 database1 中的 .DEF、.FRM 文件完全一致。

4）如果按 POST 命令中指定的匹配数据项预先使用 SORT 命令分别对两数据库进行分类，则 POST 命令可在最短的时间内执行完毕。但对一个只有很少几个记录不按顺序排列的数据库，例如：在一个已经分类的数据库中增加了几个无序记录，这种情况再对该数据库进行分类以图加速 POST 命令的执行，其效果并不是很明显的。

24. PRINT 命令

功能：打印数据库中的记录。

格式：

PRINT database/file {BY d1,d2,…{AND COMPUTE statistic d3,d4,…}} [option]

其中：统计值 statistic 可为下列词之一：

TOTAL (TOT),SUBTOTAL (SUBTOT),MIN,MAX,AVERAGE (AVE,AVG), STAX;

任选项可以是[A]或[X]；

COMPUTE 可以用符号@缩写；

STAX 用于指明 TOTAL, MIN, MAX,AVE。

使用举例：

```
PRINT CLIENTS
PRINT CLIENTS BY NAME,CITY AND STATE
PRINT CLIENTS NAME,AND  CITY AND STATE
PRINT CLIENTS BY NAME AND COMPUTE TOTAL WAGE,HOURS
PRINT CLIENTS NAME  @  TOT WAGE SUBTOTAL USING DEPT
```

使用说明：

此命令可以产生几种输出格式。在命令行中可以使用逗号、一个或多个空格符及AND（两边用空格隔开）作为分隔符。用它划分命令行上的数据项。

本命令有四种输出格式，即：屏幕格式、列格式、带统计的列格式和ASCII码格式。

1）屏幕格式：

这种格式由打入命令PRINT database产生。在这种情况下，可使用TITLE命令设置空间（详见TITLE命令）。数据库的定义文件、数据词典文件及目录表都可以使用PRINT命令打印出来。但对这类文件，必须写出其扩展名。如：

```
PRINT DATA.DIC
PRINT CLIENTS.DIR
PRINT CLIENTS.DEF
```

2）列格式

与LIST命令相似，这类输出仅产生指定数据项的列表，例如：使用命令

PRINT CLIENTS BY NAME, STATE AND PHONE

将产生有三列的记录列表。每列的标号分别为NAME, STATE及PHONE。注意：在命令行中的标号名必须与格式文件中的数据项名相一致。否则，将产生出错信息。使用[X]任选项，可以清除列表的标号头。在一个命令行中，用户可以按任意顺序指定多达32个数据项，而在打印时，每一列指定一个打印位置数，它等于1加上数据项名后跟的短线数或取标号的长度(取其中较大者)。例如：若某数据项具有23条短线，则指定24个打印字符；若在数据项后面仅跟有2条短线，则它具有6个打印字符空间，这是因为标号本身定义了这个长度。

3）带统计的列格式

这类输出除了产生列表清单外，还提供一组统计信息，如TOTAL、SUBTOTAL和AVG等。在这种格式中，每当部分值改变时，就打印出一个SUBTOTAL来。注意：对数据库按SUBTOTAL进行分类是必要的（可按升序或降序）。若这些数据项没有被分类则打印出错误信息，指明数据库未按逻辑顺序排列。如发生此错误，数据库必须用SORT命令分类（详见SORT命令）。

用户在命令行中，可以同时指定多个数据项及多个数据项的统计信息，但不能在一个统计之后立即跟以另一个统计。如：

COMPUTE TOT WAGE, HOURS AND AVE WEEKS, HOURS是可以接受的。而

COMPUTE TOT, AVE HOURS却是非法的。

数据项的打印字符数目是在格式2中决定的。但是对带TOTAL的列，至少要给出13个打印字符位置。对格式2、3中的数字、美元和日期类的数据项，在打印时是以右为准对齐的，左边用空格补充；而对字母数字或字母类型的数据项，是以左为准对齐的，右边用空格补充。

若当屏幕上出现了“……”和“＊＊＊＊＊”这些的信息时，说明输出有错误。前一种现象说明数据项后跟短线数太少，后一种现象说明在使用TOTAL、SUBTOTAL和AVE（AVG）统计任选时，产生了溢出，而按规定最多只允许10位数字。

4）ASCII码格式

这类输出将打印一个ASCII文件（例如：命令文件、正文文件、格式文件或HELP文件

等)。这种情况下，不作解释，但须记住除了扩展名为".COM", ".CVL", ".DBM" , ".DEF" , ".DIR", ".DAT" 及 ".IDX" 以外的任何其他文件均可作为ASCII 文件打印。

```
PRINT MENU.HLP
PRINT START.SUB
PRINT LEDGER.FRM
PRINT LEDGER.CMD
```

25. PROJECT 命令

功能：从一个数据库中选择一些数据项，建立一个结果数据库。

格式：PROJECT database BY d1, d2, d3…d32

结果集格式：$$.DAT, $$.DEF, $$.FRM

使用举列：

```
PROJECT CLIENTS BY NAME CITY STATE
PROJECT EMPLOYEE USING NAME AND SALARY
PROJECT DEPT NAME, DEPTNO
```

使用说明：

1）此命令选择数据库中指定的数据项，并将这些数据项的值投影到一个叫做结果集的数据库中。在结果集中，除保留原数据库中有关数据项的格式定义和数据定义以外，还保留它们的编辑标准。

使用 PROJECT 命令产生的格式文件中，具有原数据库格式投影下来的数据项，其后跟用来分配数据值的短线。在结果集中，记录的长度等于被投影的每个数据项字段长度之和。

2）此命令允许将同一个数据项投影多次，一组这样的数据项可在格式文件中出现多次。PROJECT 程序为数据项组的每一次重复出现建立一个记录，而对格式文件中非重复出现的数据项作为重复次数为 1 的数据项组来处理。在格式文件中，重复出现的数据项必须具有相同的定义(即类型相同、长度相同)。例如：在格式文件中，有一个数据项QVANTITY出现了六次，那么，这个重复出现的 QUANTITY 必须具有相同的定义。若在格式文件中有重复的数据项组，则所有重复的数据项重复次数必须相等。例如：在命令

PROJECT ORDER BY ORDER.NO QUANTITY PRICE中，数据项 QUANTITY 及 PRICE 是重复的数据项组，并且 QUANTITY 在格式文件中出现六次。那么，PRICE 也必须在格式文件中出现六次。而ORDER.NO 只出现一次。

通常，在具有 n 个记录、r 个数据项组的数据库中，经过投影，可在结果集中产生 n * r 个记录。其中包括原数据库中不活动的那些记录。

26. READ 命令

功能：将一个 ASCII 文件的数据传送到一个已经存在的数据库中。

格式：READ database {input-filename}

使用举例：

```
READ CLIENTS
READ CLIENTS APPLCANT.ASC
```

使用说明：

1）此命令可作为本数据库数据输入的一种手段，它可以将一个 ASCII 文件转换成一

个数据库的数据文件，这里的数据库必须是以前定义好的，在ASCII文件中数据可以是定长的，也可以是变长的，其格式描述如下：

①定长格式

不间断的字符串流、字符或记录之间没有任何分隔符。

②定长格式

字符串用〈C/R〉或〈CR〉-〈LF〉来划分记录，字段之间没有分隔符。

③定长或变长格式

字符串括在“ ”之内，两个字段之间用逗号分隔，两个记录之间用〈C/R〉或〈CR〉-〈LF〉来分隔。

④变长格式

字符串之间使用TAB(ASCII码的09)来分隔，记录由〈CR〉-〈LF〉来分隔。

这里要求所有的ASCII码数据文件必须由一个〈control-Z〉(ASCII码的01AH)来指定文件的结束。

2）通过数据格式文件和定义文件对所有的输入记录进行格式化和编码。输入记录的格式化，按照在格式文件中每个数据项名后跟短线的个数进行。例如：若数据项名是NAME，它在格式文件中具有12条短线，则用这个长度读入字符串。若输入字符串的个数不是12个字符，则剩余部分用空格补充。若输入字符串大于12个字节，就会在控制台上显示错误信息，指出哪个ASCII记录有错，并且自动跳过这个记录而不把它读入数据库中。若在输入记录中出现二进制数据或使用引号代替字段或记录分隔符的话，也会产生错误信息，并跳过该记录。在读入数据时，要按照定义文件中的说明，将每个记录的字段进行编码，并检验与说明的字段类型是否一致。若不一致，则给出错误信息，并跳过那条错误的记录。

27. RENAME命令

功能：更换一个数据库或文件的名字。

格式：RENAME newname = oldname

使用举例：

```
RENAME CLIENTS = APPLCNTS
RENAME NEW.DAT = OLD.DAT
RENAME D:NEW = D:OLD
```

使用说明：

此命令可以改变指定数据库标题连同相应数据库文件（包括.FRM、.DEF、.DAT文件）的名字。这种改名只限于在同一个驱动器上进行。数据库名和文件名的区别在于它们是否具有扩展名。如CLIENTS及CLIENTS.DAT。若在进行RENAME操作的驱动器上没有数据字典时，系统则认为要改的名字为文件名。如果一个数据库名或文件名已经存在的话，你则不能将其他数据库或文件改换成这个名字。若须要这样做的话，必须将该数据库名或文件名换成另外的名字或者将其删除掉。

28. REORG命令

功能：对于数据库的结构进行重组织，增加或删除数据项。

格式：REORG database {formname.FRM} [option]

其中任选为[S S]—表示创建结果集。

结果集格式：$$.DAT，$$.DEF，$$.FRM

使用举例：

```
REORG CLIENTS
REORG CLIENTS NEWFORM.FRM
REORG CLIENTS NEWFORM.FRM [$$]
```

使用说明：

1）此命令允许用户改变数据库中数据项的次序，增加或删除某些数据项。数据库经过重组织以后，它的记录将按新定义格式中所给出的数据项重新安排。如果在命令行中没有给出格式文件名，则REORG程序将出现提示信息，请求你录入一个要重组织的格式文件名。新格式是事先使用FORMAT建立的。在这个格式文件中，被保留下来的数据项必须要有同原先的数据库中数据项相同的名字及相同数量的短线。在设计好新格式以后，打入〈control-E〉结束FORMAT程序，此时不能去定义该格式文件中的数据项。在格式完成之后，使用REORG命令。对数据库的重组织除按照新设计的格式进行外，还在执行过程中建立一个结果集。若在新格式中有新的数据项名，系统将提示用户对这些数据项进行定义。新定义的数据项初始值为空。若原数据库的某些数据项已被删除，则系统提示用户确认删除这些项。另外，系统还会提示用户选择是否用结果集代替原来的数据库。

2）在批处理操作中，若用户在命令行中提供了新的格式名，则除非用户需要对其中的某些数据项名进行定义，否则系统不出现提示信息。

3）在批操作过程中，如果用户在命令行中没有指定任选项〔$$〕，则使用结果集代替原来的数据库。

29. RESTART 命令

功能：继续处理一个被中断了的过程。

格式：RESTART {drv:}

使用举例：

```
RESTART
RESTART B:
```

使用说明：

每当一个正在运行的命令过程异常中止时，可以使用RESTART命令重新启动它。在许多情况下，如系统错误、操作员打入〈control-C〉硬件复位或掉电等一些硬件故障，都可能引起过程运行中止。由于命令过程执行前已将所有可执行命令写入工作文件$$.DBM中，当执行异常中止时，那些在$$.DBM中未执行的命令被保存起来。RESTART命令可恢复这些命令的执行。若$$.DBM文件不在当前驱动器上，则要在命令行中指明驱动器名(如RESTART B:)。注意：用户必须登记操作中止时的当前驱动器名，然后打入RESTART {DRV:}命令。对于那些由硬件复位或其他故障引起的操作中止，使用RESTART命令可能会产生奇怪的结果。对于这种情况，应当确定发出RESTART命令继续进行批处理操作产生的结果。在命令行分析程序中已经设计好，使得〈control-C〉仅在批处理过程中一条命令执行完毕时起作用。故使用RESTART命令可以重新启动它。但是，LIST和PRINT命令可能在执行期间中止，而且不能使用RESTART命令来重启动。

30. RUN命令

功能：启动一个带有指示句的命令过程。

格式：RUN batch {.CMD} {param1, param2, …param9}

使用举例：

```
RUN GLEDGER
RUN REPORT JUNE AUG
RUN ORDERS
```

使用说明：

1）使用RUN命令可以用来处理命令过程文件中的一系列命令。其命令过程文件可由系统编辑程序（CP/M的ED.COM）来建立。命令文件所产生的输出是一个名为 $$.DBM 的工作文件。该文件一经启动就一直执行下去，直到终止或有错误出现。命令文件由系统编辑程序建立，必须跟有一个扩展名 .CMD，其文件格式如下：

```
COMMAND LINE1……
COMMAND LINE2……
……
COMMAND LINE127……
```

2）在每一个命令行中，可以包含可执行命令（象 rDBMS 命令）和不可执行命令（象RUN指示句），其指示句的格式如下：

```
*IF condition
*ENDIF
*ECHO ON
*ECHO OFF
*MESSAGE message line
*GET param1, param2, …param9
*LET Param = value
*END
*ABORT
```

在命令文件中，当第一个字符为分号“；”时，就作为注解行。注解行只用来解释程序，不写入$$.DBM工作文件。

3）在命令文件中，若第一个字符为星号“*”，则认为星号后面所跟的关键字为指示句，其格式如上所示。它们的功能如下：

*IF指示句：用来测试一定的条件，尤其是用在有些参数要测试，要判别是否执行一段命令这种情况。*IF句中指出的条件是任意的逻辑条件。如果条件为真的话，它将执行由*IF和*ENDIF指示句所包括的语句。如：

```
*IF $1<40
   SELECT CLIENTS WHERE STATE IS MI, IN
*ENDIF
```

若条件$1<40为真，则执行由*IF 及*ENDIF 括住的语句 SELECT CLIENTS WHERE STATE IS MI, IN。若条件不真，则跳过*IF和*ENDIF括住的语句。在一个RUN命令文件中，*IF和*ENDIF指示句可以嵌套七层。

*GET指示句：

允许用户在命令文件执行期间通过键盘写入参数值。例如：如果在运行的文件中出现指示句＊GET $4，系统会用一个冒号":"提示用户，此时你必须录入一个值及一个〈C/R〉，系统会自动将这个值赋给$4。

在一个＊GET指示句中，允许同时出现多个参数。参数之间用逗号或空格符隔开。在录入参数值时，要使值的个数与参数的个数（带有空格或逗号的参数必须放在引号中）相同。例如＊GET $3，$5指示句在运行时会提示出":"，打入500,"LOS Angelos"分别对$3和$5赋值。并且，＊GET 指示句给出的参数可以在 RUN 命令行中指定其值，或是使用＊LET指示句对这些参数进行赋值。例如：用户打入RUN REPORT JUNE AUG 〈C/R〉。当命令过程文件在执行时出现指示句＊GET $2,而后无论操作员打入何值来对$2响应，$2的值均由"AUG"取代。这与指示句＊LET $2=AUG是一样的。$1的值仍旧为"JUNE"。

＊MESSAGE指示句：

＊MESSAGE(或*MSG, *MESS)指示句在运行时将信息显示在控制台屏幕上。如果无信息行，则仅作为一次回车换行处理。在＊IF和＊ENDIF指示句间，可以使用＊MESSAGE指示句帮助操作员选择适当的条件。

＊LET指示句

允许在命令文件执行中对由＊GET指示句给定的参数进行赋值。〔注意〕等号两端必须使用空格隔开。如：

＊LET $4 = N (1)

＊LET $5 = "Los Angelos" (2)

＊LFT $5 = $4 (3)

对于最后一句即＊LET $5 = $4,参数$4必须是常数，或必须事先使用＊GET和＊LET指示句对其进行赋值。

＊ECHO指示句：

＊ECHO OFF 指示句，可以在打印之前，关闭批处理作业流的打印输出。在通常的情况下,当出现错误或批操作中止时,总是将＊ECHO置成ON。并且,在命令过程正常结束或遇到一个HELP命令时，它也被置成ON。在一个批处理文件中,允许使用9个符号参数,用$1～$9表示。它们的值可以由命令 RUN 后跟操作数指定，也可以由＊LET指示句指定。在命令行中出现的参数个数必须与命令文件中出现参数的个数相同。若不相同，就会出现错误信息。并使RUN执行失败。另外，$TODAY可以作为命令过程中一个特殊的参数，它的值指定为当前日期。

＊ABORT，＊END指示句：

＊ABORT指示句用来中止RUN程序的执行，并将控制权交给操作员。在这种情况下，不对命令文件作任何处理。

＊END指示句后面不再跟任何语句。一旦遇到＊END指示句，就启动作业流。在命令文件中，＊END指示句是任选的。

31．SAVE命令

功能：保存一个结果数据库。

格式：SAVE database/file {OK}

使用举例：

SAVE CLIENTS OK
SAVE THIS
SAVE DATAONLY.DAT

使用说明：

此命令产生结果集文件($$.DAT，$$.DEF，$$.FRM)的拷贝，并在命令行中给这些文件指定新名（通常，新名与新数据库的标题相同），同时还要将这个标题写到数据词典中。若这个数据库已经存在，则系统要求确认许可删除该数据库或者用新的名字代替它，确认可以在命令行末尾以批方式给出OK。文件可以独立存放，这时不录入词典记录。文件名与数据库名区别就在于它们是否具有扩展名。

例如命令：

SAVE CLIENTS.DAT

将产生$$.DAT文件的拷贝CLIENTS.DAT。

32. SELECT命令

功能：选择出满足条件的记录，并建立结果数据库

格式：SELECT database {WHERE condition}

使用举例：

SELECT CLIENTS WHERE DATE IS LT 1/1/79
SELECT CLIENTS ST ACTIVITY IS TERMINATED
SELECT CLIENTS WHERE STATE IS MIOR STATE IS IN

其中条件部分详见DELETE命令中的描述。

使用说明：

使用该命令选出满足条件的主数据库记录，并将它们写入数据文件$$.DAT中。复制主数据库的格式和定义文件并分别写到$$.FRM和$$.DEF文件中。新的数据文件（$$.DAT）包括主数据库中所有的数据项。

注意：选择时，只输出活动的记录，因此不用进行分类就可用该命令删除一个不活动的记录。在这种情况下，可以根据用户选择，决定在选择时，将结果集重命名为原记录集后，是否要删除原来的记录集。为了使用选择后的数据库，用户必须把用来调用这些输出文件名的记录登入数据词典，或者通过重命名将它们拷贝到一个新磁盘上（这个新磁盘上应含有数据字典登记项）。通常，用户可以使用RESULT作为标题保留一个数据词典登记项并使用$$来指定它的数据文件、定义文件及格式文件。而这些文件，可以建立在当前驱动器上。

33. SET命令

功能：设置rDBMS系统操作参数。

格式：SET {function status}

使用举例：

SET AUTOSEL OFF
SET ECHO OFF
SET MASTER B:
SET ABORT OFF
SET DATE DMY

使用说明：

此命令用来设置系统某些功能的状态，这些功能包括：

ABORT ON/OFF

AUTOSEL ON/OFF

DATE MDY/DMY/YMD

ECHO ON/OFF

MASTER {DRV:}

1）ABORT ON/OFF（缺省值为ON）

在ABORT参数设置为ON的情况下，批处理操作如果出现错误，就会自动结束操作。相反，如果ABORT参数设置为OFF，一旦批处理操作发生错误，并不终止批操作，而是跳过这一段。批处理只能由使用〈control-C〉或机器复位来停止。因此，在这种状态下所进行的批处理操作，有可能引起错误的连锁反应，这对于批处理是不利的。因而在正常情况下，总是把状态ABORT设置成ON。

2）AUTOSEL ON/OFF

当AUTOSEL设置为ON时，系统可以在当前驱动器中未找到录入的命令时自动地到主驱动器（缺省值为A驱动器）中去查找。若AUTOSEL设置为OFF时，对录入命令的查找只限于在当前驱动器上。

3）ECHO ON/OFF

当ECHO设置为OFF时，主要用来在批操作中禁止命令行和把信息回送到控制台上。每当命令过程执行完毕、发生系统错误或显示HELP屏幕时，都会自动将ECHO设置为ON。在本系统的数据库管理程序初始装入内存时，AUTOSEL、ABORT、ECHO都设置为ON，MASTER设置为A驱动器。

4）SET DATE

系统允许下列三种格式作为用户显示日期及录入日期的格式。它们是：

MDY⟹mm/dd/yy

DMY⟹ dd.mm.yy

YMD⟹ yy-mm-dd

这里，使用“/”、“.”、“-”作为这类数据的分隔符。在上述之一的格式中，不允许同时出现二种分隔符。

上述DATE格式，仅作为一项登记和显示参数格式，它并不影响日期值。因此，用户可以选择上述SET DATE命令中的一种格式进行录入，而使用另一种格式进行显示。

34. SORT命令

功能：按照给定的数据项名，对数据库的记录进行分类。

格式：SORT database BY d1, d2, …d32 〔option〕

其中任选〔D〕表示记录按降序排列。

结果集：其中的文件被重写。

使用举例：

SORT CLIENTS BY NAME

SORT CLIENTS BY CITY STATE AND ZIP-CODE

使用说明：

1）此命令把d_1作为主分类元素，然后顺序把d_2，d_3，…d_{32}称作是次分类元素。在命令行中，可以指定多达32个数据项名。这些数据项之间使用逗号，AND或空格作为分隔符。若在命令行中指定任选〔D〕，则文件记录按照降序排列。

2）由于在分类过程中要重写所有的分类项和数据文件，因此在当前驱动器上会产生一个临时工作文件（$$.SRT）。这个文件在分类工作完成之后自动被删除。如果当前驱动器没有足够的空间来容纳这个文件的话，可在另一个驱动器上放上一张空盘，并对它进行注册，然后对数据库文件进行分类。这样能以保证拥有最大容量的空间来完成这项工作，也就是说，分类操作需要有一倍于原文件大小的可用空间或一张分类磁盘作为工作文件空间。否则，分类操作会中止。

3）利用分类操作也可以清除文件中不活动的记录。对于分类程序来说，允许被分类的文件长度最大不超过128K字节。若被分类的文件很大的话，可以首先使用SELECT命令将其划分成许多小的段，然后再逐段进行分类。最后，使用APPEND或COMBINE命令进行合并。

35. STAX命令

功能：显示或打印数据项的统计值。

格式：STAX database BY $d_1,d_2,\cdots d_{32}$ 〔option〕

其中任选项分别表示为：

〔A〕 仅显示平均值

〔C〕 仅显示计数

〔M〕 仅显示最小值及最大值

〔T〕 仅显示总计值

〔P〕 打印输出

使用举例：

```
STAX EMPLOYEE BY AGE
STAX EMPLOYEE BY WAGE AND AGE 〔P〕
STAX EMPLOYEE WAGE 〔T〕 〔M〕 〔P〕
```

使用说明：

下例中，你可以确定数据库ACCOUNTS的销售费用和成本（SALES & COSTS）。打入命令

```
STAX ACCOUNTS BY SALES AND COSTS
```

可显示这两个数据项的统计值。如：

	SALES	COST
Minimum		
	3.00	4.00
Maximum		
	55.00	220.00
Total		
	10345.00	5435.00

```
Average
                    41.38          21.74
Records Count = 250
```

在STAX命令行中，最多可以指定32个数据项。注意：对A类及AN类数据项来说，不能使用STAX命令作统计。另外若平均值及总计值产生大于四个字节的数值，就会溢出。在这种情况下，会在屏幕上出现一串星号。

36.SUBMIT命令

功能：启动并处理一个批处理命令过程。

格式：SUBMIT procedure {.SUB} {param1,param2,…param9}

使用举例：

```
SUBMIT GLEDGER
SUBMIT REPORT JUNE AUG
SUBMIT ORDERS
```

使用说明：

1）SUBMIT命令可处理一个带有扩展名为.SUB的文件，该文件中含有一系列可执行命令（如rDBMS命令、CP/M命令）及不可执行命令（如注释语句、RUN的指示句）。这个文件由系统编辑程序（如CP/M中的ED.COM）来建立。执行时，打入SUBMIT〈过程名〉〈C/R〉。文件执行时，产生一个名为$$.DBM的工作文件（该文件只含有可执行命令）。一旦一个过程被启动，它就一直执行到底，除非执行时有错误出现。命令过程文件在编辑时必须带有扩展名.SUB。其格式为：

```
COMMAND LINE1……
COMMAND LINE2……
        ……
COMMAND LINEn……
```

文件中可包含注释行，注释行用一个分号“；”作为首字符，其后的内容作为给程序员的注解。注释行仅用来说明，它不出现在工作文件$$.DBM中。

传递给命令文件的参数是$1～$9。在命令行上，跟在批处理文件名后的参数值，将顺序代替文件执行时的参数。若命令行上出现的参数值的个数与命令文件中参数的个数不一致，则出现错误信息，并导致SUBMIT过程执行失败。另外，在命令文件中，允许出现表示当前日期的参数$TODAY，在执行时，自动地被赋值为当前日期值。

37. SYS命令

功能：退出rDBMS环境。

格式：SYS或SYSTEM

使用举例：

```
SYS
SYSTEM
```

使用说明：

使用SYS命令，用户可以退出数据库管理系统环境，进入操作系统环境，并将控制权还给CP/M。此时所有的地址及存储空间都恢复到原始状态。

38. TABULATE命令

功能：汇总和打印指定的数据项。

格式：TABULATE database BY $d_1, d_2 \cdots$ {AND COMPUTE statistic $d_3, d_4 \cdots d_{32}$} [option]

其中，统计值statistic可以是：

TOTAL (ADD, TOT, SUM)，MIN, MAX, AVERAGE (AVE, AVG)。词COMPUTE可以简写成“{”，或者@。

任选项分别表示为：

[A] 累加部分和

[P] 送交打印机打印输出

[S] 建立结果集

结果集：$$.DAT, $$.DEF, $$.FRM (若任选[S]时)

使用举例：

```
TABULATE CLIENTS BY STATE
TABULATE CLIENTS BY STATE AND COMPUTE TOTAL WAGE[P]
TABULATE CLIENTS BY STATE { TOT WAGE, AVERAGE HOURS
```

使用说明：

在使用此命令之前，要将所有要统计计算的数据项进行排序（使用SORT命令）。不然的话，将会产生错信息及程序中止。

通常，所生成的结果输出带有计数、总计、最小值、最大值及平均值的统计报告。

在第一个例子中，打印输出如下：

SEX	{COUNT}
female	20
male	25
Total	45

在第三个例子中，打印输出如下：

SEX	WAGE {subtotal}	HOURS {average}
felmale	2301.03	38
male	2968.07	49
Total	4999.10	

注意：总计只能在小计（subtotal）和计数（count）的列上进行累计。列的空间为1加上格式文件中定义的那个项的标号字符个数或在数据项标记后跟的短线的个数。这里加上1，是为了使在打印格式中列与列之间至少有一个空格加以分隔。对于计算谓语中的数据项，如果其长度（或短线个数）不足12个，则分配12 + 1 = 13个打印字符空间。空字段计数总是具有13个打印字符空间。

39. TERM命令

功能：定义系统终端。

格式：TERM {id}

使用举例：

```
TERM
TERM = LEAR
TERM = ADDS
TERM  MISC
```

使用说明：

通过打入四个字符的代码，用户可以选择八种不同类型的终端。与CONDOR SERIES 20 rDBMS相兼容的终端类型有：

ADDS-ADDS REGENT 20 25 40 60 100 200

BEEH-BEEHIVE MICRO - B_1 MACRO -B_2 B_{100} B_{150} B_{550}

HAZL-HAZELTINE 1500 1510 1520

INFO-INFOTON (GTC) 100 101 200 400

LEAR-LEAR SEIGLER ADM - 1A ADM-2 ADM-3A ADM-31 ADM42

PERK-PERKIN ELMER FOX 1100 OWL 1200

SORC-SOROC IQ-120 IQ-140

TRS2-PIckles AND TROUT CP/M FOR TRS80-Ⅱ

任何模拟以上类型的终端，具有屏幕清除、游标控制等功能。如果你选择了正确的终端类型，那么在一个刷新的屏幕中央会出现一个“READY”，而在屏幕的四个内角出现星号“*”。

如果用户选择了TERM MISC，那么，用户可以把他的屏幕功能码提供给终端编辑程序，并且用户还可以定义一个具有四个字符的终端标识符（定义的终端标识符不能是上述八个标识之一）。详见附录B。

40. TITLE命令

功能：打印报表标题。

格式：TITLE {'text',or"text",PAGE,DATE,TOP,SKIP,LINE, nn} 标题从句是用以下定义的词和字符组成的。

P或者PAGE　打印第n页，n指当前页。

L或者LINE　回车换行。

D或者DATE　打印当前注册日期。

T或者TOP　打印nn行后走纸。

S或者SKIP　跳过一行。

nn　标题与标题之间的行数（2≤nn≤99，缺省值为66）。

使用举例：

```
TITLE 'FINANCIAL REPOPT FOR ABC COMPANY AS OF',
DATE
TITLE LINE, LINE, 'TWO LINE', LINE, 'FOR A TWO LINE
TITLE'
TITLE TOP, 44
```

以上标题，每44行换一页。

使用说明：

在打印出第nn行之后，该命令就把报表标题打印出来。这里，nn 可以是2～99之间的任何数（缺省值为66）。若在命令行上出现P或者D，则在报表的每一页上打印出页号和注册日期，它的这置在标题行中给出，例如：

打入TITLE 'THIS IS',PAGE,'AND TODAY IS',DATE<C/R>

则打印出：

THIS IS PAGE1 AND TODAY IS 4/23/81

使用S可以在打印nn行后插入若干个空行。例如：

TITLE SKIP, 2

则每打印二行插入一个空行。

注意每个TITLE命令清除以前的TITLE命令。

41. UPDATE命令

功能：改变一个数据库中的数据项值。

格式：UPDATE database {WHERE condition}

使用举例：

```
UPDATE CLIENTS WHERE NAME IS SMI*
UPDATE CLIENTS NAME IS 'DOE JANE'
UPDATE DEFSET.DEF_ST FIELD IS 1 2 4
UPDATE DATA.DIC WHERE TITLE IS CLIENTS
UPDATE CLIENTS.DIR
```

其中，条件部分详见DELETE命令的有关条件部分。

使用说明：

该命令类似于DISPLAY命令，所不同的是它可以改变一个数据库中的记录。当做完某项的更新时，用户可以根据屏幕底端给出的提示，选择适当的控制字符或继续修改或退回系统。每个修改好的记录都被写回文件中。

42. WRITE命令

功能：把数据库记录传送到一个ASCII 文件中。

格式：WRITE database {output-filename} [option]

选择项为：

无——定长的连续的ASCII 数据流，数据项之间无分隔符。数值字段向右对齐。

[A]——节略的变长数据流。数据项之间用 TAB（ASCII 的09）分隔、记录之间用<CR>—<LF>分隔。数据项后的所有空格都被略去。

[B]——基本的变长数据流，其中数据项之间用引号" "括住，两个数据项之间用逗号分隔，记录之间用<CR>—<LF>分隔。

[R]——PRG定长数据流，数据项之间无分隔符，记录之间用<CR>—<LF>划分。数值字段向右对齐。

使用举例：

```
WRITE CLIENTS
WRITE CLIENTS REPORT
```

WRITE CLIENTS REPORT〔B〕

注意：所有的ASCII文件都用〈control-Z〉（ASCII 01AH）表示文件结束。如果在命令行中没有指定输出文件名，则将输出写到文件$$.ASC中。当你需要将SERIES/20数据库转换成一个ASCII数据文件，并要其他的应用程序和报表生成程序来读这个ASCII文件时，这条命令是非常有用的。这里所指的应用程序是指利用主语言，例如BASIC、FORTRAN所写的程序。你可根据应用程序中数据格式的要求，选择以上四种格式之一，生成你的ASCII文件。

注意：格式〔B〕与MBASIC及BASIC相兼容。

附录B　TERM命令与MISC选择

用TERM命令的MISC选择可给计算机定义屏幕功能码和四个字符的终端标识码(ID)，还可把这些定义保存起来。一旦提供了功能码和终端标识码（ID），只须录入这个ID码，就可以选择CRT终端类型。该选择用于满足CONDOR SERIES /20rDBMS的工作要求但尚未在附录A的TERM命令中列出的终端。

按如下步骤设置功能码和终端类型码：

打入A:TERM 〈C/R〉

系统提示出 Terminal=

打入MISC 〈C/R〉响应。

计算机用如下显示的指令、语法、描述及提示响应（划线部分是用户对提示的响应）。

****

A》TERM〈C/R〉

Terminal=MISC〈C/R〉

Specify CRT screen control codes by typing the decimal value of the code

Parentheses( )indicate the number of codes or range expected

Bracket〔 〕indicate the current default values

Entering only a〈C/R〉cause the default values to be used

Enter screen clear sequence (1-2)(26)••27 42〈C/R〉

Screen clear delay in nulls (0-99)(0)••6〈C/R〉

Enter home cursor sequence(1-2)(30)〈C/R〉

Enter cursor addressing sequence(2)(27 61)〈C/R〉

Line or column sent first(1)(L)? 〈C/R〉

Line or column offsets(2)(32 32)〈C/R〉

Cursor addressing delay in nulls (0-99)(0)6〈C/R〉

Enter back cursor sequence (1-2) (8) 〈C/R〉

Forward cursor sequence (1-2) (12) 〈C/R〉

Up cursor sequence (1-2) (11) 〈C/R〉

```
  Down cursor sequence (1-2)  (10) <C/R>
Enter Initialization sequence ( 0 -8) [ 0 ]
Parameters OK (Y/N)? Y
Enter Terminal ID ( 4 ) [MINE]: IQ20<C/R>

II READY II
A»
```

****

如果查看了你的终端码之后，发现需要修改它们，则打入 TERM <C/R> 和 MISC<C/R>。屏幕上所看到的缺省值是以前录入的代码。若只打入<C/R>，则保留以前正确的代码；若打入新的代码，则改变它们。例如：你若希望改变空值的个数，可打入如下命令：

****

```
A»TERM<C/R>
Terminal = MISC<C/R>
Specify CRT screen control codes by typing the decimal value of the code
Parentheses () indicate the number of codes or range expected
Bracket[ ]indicate the current default values
Entering only a <C/R> causes the default values to be used

Enter screen clear sequence (1-2)  (27 42) <C/R>
  Screen clear delay in nulls (0-99)  (6):8<C/R>
Enter home cursor sequence (1-2)  (30) <C/R>
Enter cursor addressing sequence ( 2 )  (27 61) <C/R>
    Line or column sent first (1)  (L) ? <C/R>
    Line and column offset (2)  (32 32) <C/R>
    Cursor addressing delay in nulls (0-99)  (6):8<C/R>
Enter back cursor sequence (1-2)  (8) <C/R>
    Forward cursor sequence (1-2)  (12) <C/R>
    Up cursor sequence (1-2)  (11) <C/R>
    Down cursor sequence (1-2)  (10) <C/R>
Enter initialization sequence (0-8) [ 0 ]

Parameters OK (Y/N) ? Y
Enter terminal ID (4) [IQ20]<C/R>

II READY II
A»
```

****

注意：这时缺省值是以前定义的值。

附录C　分类总帐系统数据定义

下面是分类总账数据库和日记账数据库的数据定义：

DATABASE: GENERAL LEDGER DATABASE

DATA ITEM NAME	TYPE	NUMBER OF BYTES
ACCOUNT	AN-Alphanumeric	6
ASSET	AN-Alphanumeric	1
LIABILITY	AN-Alphanumeric	1
NAME	AN-Alphanumeric	18
DEBIT	N-Numeric	4
CREDIT	N-Numeric	4
YTDAMT	N-Numeric	4
MONTH	AN-Alphanumeric	3

DATABASE: JOURNAL

DATA ITEM NAME	TYPE	NUMBER OF BYTES
ACCOUNT	AN-Alphanumeric	6
DESC	AN-Alphanumeric	18
DEBIT	N-Numeric	4
CREDIT	N-Numeric	4
MONTH	AN-Alphanumeric	3
DATE	J-Julian	3

参加本册编译校对的有：刘运基、郑远新、沈江、周毅、焦敏。