微型计算机软件资料汇编

第五册

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

机械工业部仪表局情报室
《仪表工业》编辑部

CP/M

微型计算机软件资料汇编（第五册）

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
北京市机电研究院科益印刷厂排版印刷
北京市建华书刊发行社发行

*

1985年2月 北京
代号：8404

内 容 提 要

《微型计算机软件资料汇编》第五册包括 PASCAL/MT⁺、LINK-80、SID 符号指令调试程序、MAGIC WAND字处理程序和 WordStar 文件编辑系统的使用手册。

《PASCAL/MT⁺用户手册》介绍了在 CP/M 系统环境下 PASCAL 程序的编译、连接、运行以及反汇编和调试跟踪。对 PASCAL/MT⁺语言语句和过程、数据类型定义及内部构造函数等也作了详细地叙述。

《LINK-80用户指南》介绍了LINK-80连接编辑程序的功能和使用方法，它所处理的各种浮动模块、库文件的数据格式，用PL/I-80语言和 RMAC 浮动汇编程序的调用方法以及复盖处理使用等。

《SID 符号指令调试程序》介绍了在CP/M支持下的8080符号指令调试程序 SID 的功能以及用它进行程序开发的方法，对其各种命令及实用程序作了详细介绍。

《MAGIC WAND 字处理程序使用手册》和《WordStar 文件编辑系统》介绍了在 CP/M 操作系统下一个编辑和字处理系统所提供的编辑和打印功能及其使用。

本手册可作为微型机用户的使用手册，也可供计算机软件人员及大专院校有关专业学生学习参考。

微型计算机软件资料汇编

第 五 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。

目　录

PASCAL/MT⁺用户手册

LINK-80 用户手册

SID 符号指令调试程序使用手册

MAGIC WAND 字处理程序使用手册

Word Star 文件编辑系统

PASCAL/MT+用户手册

第一章 PASCAL/MT+导言和概述

第一节 如何使用本手册

1.使用编译程序

编译程序的选择开关和介绍见第二章第二节。大程序可分解为一些称为模块的小程序，见第三章第一节。在第二章及附录中介绍了有关运行时的错误处理。

2.使用连接程序

第二章第三节介绍了连接程序的操作。在熟悉了这些操作之后，可熟悉一下为大程序设置的复盖结构，见第三章第三节。

3.使用调试跟踪程序

调试跟踪程序指令见第二章第五节。

4.文件输入／输出（包括控制台和打印机的输入／输出）

附录A通过大量示例的详细说明，给出了PASCAL／MT+文件的读写方法。标准PASCAL以外的文件存取过程在第三章第四节及附录中均有讨论。第七章第十二节中也可找到一些关于文件存取的说明。快速读写处理在第三章第四节和附录A中描述。

5.实数

实数的格式见第五章第一节。AMD 9511硬件算术集成芯片的使用介绍见附录。FPREALS.ERL是软件浮点实数库，BCDREALS.ERL是BCD实数库。

6.递归

第二章2.4.10节和第三章第十节指出如何使用递归以及使用递归注意事项。递归对于语言各种性能的影响见索引。

7.把一个程序装入ROM中

有关目的码操作的信息见第六章。

8.汇编语言接口

用汇编语言编写并被编译成软件兼容文件的模块可以与PASCAL／MT+程序结合起来。如何实现这种结合见第三章第二节。

9.运行时错误处理

PASCAL／MT+运行时错误处理在第二章2.4.6节和第二章2.4.7节以及第四章中讨论。

第二节 系统概述

PASCAL／MT+系统包括编译程序、联接程序、反汇编程序、调试程序以及运行时子程序库和可选择编辑软件包。图1给出了这些程序间的关系图(任何CP／M@兼容的编辑程序，包括我们提供的语法扫描快速编辑程序都可以使用)。

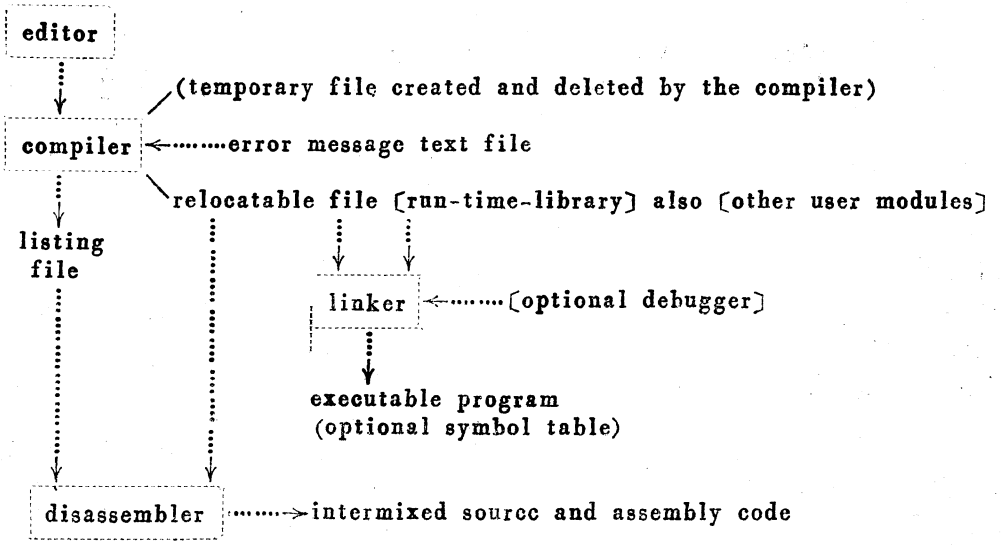


图1 PASCAL／MT⁺ 操作框图

用户执行编译程序，把源文件转换成浮动机器码。接着用户把此机器码与运行时子程序库连接起来，生成可执行的目的程序。如果需要，可使用反汇编，把编译程序生成的列表文件与浮动文件结合起来，形成源码与汇编码混合的文件。用户可以告诉编译程序，希望使用符号调试程序，这样编译程序和连接程序就会输出一个带有符号调试程序的可执行模块。使用符号调试程序软件包可以得到诸多动能，如变量显示、设断点、跟踪等。

第三节 系统要求

PASCAL／MT⁺ 系统要求 8080、8085 或 280 CPU 在 CP／M 操作系统（1.4 版本或更高的版本）下运行。与 CP／M 兼容的其他操作系统也可采用，不过这种兼容性要由用户自己负责确保。

在 CP／M 环境下，编译程序要求有 92K 同时联机的存储单元。PASCAL／MT⁺ 编译程序要求驻留内存 36 K。编译程序符号表占用 4 K 符号表空间（使用 $K 触发开关删去不用的符号，可以改变该空间大小），另外还要有些空间来存放用户符号。例如 PASCAL／MT⁺ 编译程序的最大模块要求 7 K 符号表空间。这意味着为了编译小程序（允许使用 3 K 用户符号表空间），必须的最小 TPA（暂态程序区）是 43 K。一个 48 K 的 CP／M 系统在大多数场合都提供了这种最小 TPA。如果 BDOS 入口高于 ABFF，则编译小程序就有了足够的空间，然而我们推荐 48 K TPA 作为最小的工作环境。如果有附加空间会被自动利用。

连接程序仅要求 11K 内存空间。因为在程序连接和运行的时候，子程序要用到空间，所以可用的内存空间量就限制了被连接程序的大小。对于 59 K 的 TPA（最小构成 64K 的 CP／M 中的最大可用空间）来说，在连接程序和连接程序数据装入之后，剩下的工作区空间大约是 49 K。这个工作区必须包括符号表和目的码。另外，如果没有使用／D 开关，该区域还须包含数据区。程序太大不能一次连接时，可采用复盖技术。

第四节　运行时要求

PASCAL／MT⁺ 系统产生的程序要用到从 PASLIB 运行时库和别的浮动模块中抽出来的各种运行时提供支持的子程序。这些运行时子过程处理很多，如“乘”、“除”以及与操作系统间的文件输入输出接口。使用随机存取时要求有 CP／M 2.0 或更高的版本。

在 CP／M 操作系统下运行的程序，其最小的运行时开销通常为 2K 到 6K。这包括用于整数、字符、字串的支持过程和正文文件读写过程。有些文件用到实数、非正文文件读写、超越过程等，这就要求包括一些附加模块。下面给出运行时库和支持模块目前的大致大小（以 1024 字节为单位）。这些数字有可能变动。如果你想知道运行时开销的确切值，请把每个文件连接一下，略去无定义考虑，注意连接终了时给出的数据和代码大小。

第五节　PASCAL/MT⁺ 配给磁盘信息

PASCAL／MT⁺ 软件包可配在与你计算机相兼容的介质上（磁带、磁盘等），该软件包由一组 CP／M 文件组成，包含目的文件、源文件和浮动文件。下面列出的是所有这些文件的名称和内容简述，并列出组成最小操作盘所要求的文件。随着软件新版本的出现和分配磁盘内容的改变，这些文件也会变动。

文　件	内　容
MTPUS.COM	PASCAL／MT⁺ 编译程序，可执行的 CP／M 文件（第0阶段）进行语法扫描和列表文件生成。还产生 PASCAL.TOK 临时文件，它的大小大约为源文件的1／3。同时把 MTERRS.TXT作为错误信息文件使用。
MTERRS.TXT	MTPLUS 所用错误信息的 ASCII 文件。
MTPLUS.000	调用复盖 001 到 006 的主控文件（将它换名成 .COM 文件，不能直接运行）。
MTPLUS.001	内部符号表初始化。
MTPLUS.002	构造用户符号表（第1阶段）。
MTPLUS.003	代码生成初始化。
MTPLUS.004	目标代码生成程序扫描（第2阶段）。
MTPLUS.005	代码生成结束，外部调用输出。
MTPLUS.006	调试符号表输出过程（实际上在.002 和.003复盖之间执行）。
LINKMT.COM	PASCAL／MT⁺ 连接程序。以.ERL 和.CMD 文件作为输入，选择地产生.COM、.HEX、.SYM 和 SYP 文件。
PASLIB.ERL	以浮动形式搜寻运行时子过程库。总是在最后连接。
FPREALS.ERL	软件浮点运算支持过程。该库包含软件浮点过程和REALIO.ERL。可以选择，必须在 PASLIB 以前连接。
TRANCEND.ERL	超越函数的支持过程（仅用于软件浮点运算）。
BCDREALS.ERL	商业运算支持过程（不包括平方根和超越函数）。

DEBUGGER.ERL	符号调试程序的浮动代码（使用时作为主程序连接）。
DEBUGGER.TXT	DEBUGGER 命令提要。
DIS8080.COM	反汇编程序，将 .PRN 和 .ERL 作为输入文件，将混合列表文件输出到磁盘文件上，或者输出到控制台、打印机上。也可以处理编译产生的Z80指令，这不是通用的反汇编，只能用于PASCAL程序。
LIBMT.COM	库程序。输入 .BCD 和 .ERL 文件，产生可搜寻的库文件或与 L80 兼容的模块。
FULLHEAP.ERL	用于完成 NEW 和 DISPOSE，带有浮动形式的废空间回收程序和堆管理程序。PASLIB 中有 UCSD 方式的堆栈管理程序。使用这些程序时，必须在 PASLIB 之前连接。
ROVLMGR.ERL	RELOAD 为真时，汇编出的复盖管理过程。
CALC.SRC	测试浮点的示例程序。使用 AMD9511 硬件时，用此程序来保证 9511 运算的操作正确。
REALIO.ERL	浮点读写过程，仅当使用 AMD 9511 硬件时需要。
TRAN 9511.ERL	包含了 9511 驱动器的模块。此种驱动器仅用于带有硬件浮动的超越函数。
FPRTNS.ERL	AMD 9511 硬件浮点支持程序。
AMDIO.SRC	包含了与 AMD9511 接口的模块。为了适应于指定的硬件要求，用户必须重新编辑和编译该模块。
AMD 9511.CMD	连接命令文件，包括 AMDIO, FPREALS, REALIO 和 TRAN9511。
UTILMOD.ERL	包含 KEYPRESSED, RENAME 和 EXTRACT 实用子过程的模块。
RANDOMIO.ERL	与顺序读写兼容的随机 I／O 过程。

以下是以源代码形式提供的运行时库的子过程。其中有些允许用户运行时不带操作系统（譬如 CP／M），而能很好地与硬件相交接。另外一部分子过程允许用户在 PASLIB 中的过程被执行时，改变过程的作用。这些过程可自成体系，或与 PASLIB 中的过程相兼容。几乎所有这些过程都以 PASCAL 模块的方式提供。这样在大多数情况下用户都不需要使用汇编去实现以 ROM 为基础的程序。

PINI.SRC	初始化过程。
CPMPD.SRC	用于读字串过程（@ RST）。 所以把它包括起来，是因为它使用了CP／M直接BDOS 调用。
HLT.SRC	“停止”过程。此过程调用 CP／M 。
CHN.MAC	链过程。修改后可做组之间的转换工作。这种情况在非 CP／M 的环境下用到。因为它太特殊，故以汇编码形式给出。注意，大多数基本的 ROM 系统不用链。所以用户没有必要拥有汇编程序去产生基本的 ROM 系统。
RNC.SRC	读下一个字符的过程。
WNC.SRC	写下一个字符的过程。
DIVMOD.MAC	DIV 和 MOD 过程的源程序。提供直接调用 CP／M，以得到除 0 的错误信息。

FIBDEF.LIB	PINI，RNC 和 WNC 文件。文件控制块定义。
OVLMGR.MAC	复盖管理程序。带有用户可选项，允许多个复盖和复盖间的调用。未经改动的版本在 PASLIB 中。
IOERR.SRC	用户处理 I／O 错误示例过程。
XBDOS.SRC	含有调用 IOERR 的 BDOS 过程。
IOCHK.BLD	用来建立 I／O 错误库的文件。
RST.MAC	读字串过程。
CWT.MAC	等待行结束过程。
GET.SRC	低级输入过程。
PUT.SRC	低级输出过程。
@RNB.SRC	读下一个字节过程。
@WNB.SRC	写下一个字节过程。

第六节　最基本构成说明

以下列出的文件是运行 PASCAL/MT+ 所需要的绝对最小构成。它们不一定要放在同一张盘上，但是必须全部联机。象连接程序和库文件这样的附加文件可以放在同一张磁盘上，不过对于运行编译程序来说并非一定如此。MTPLUS.006复盖仅在使用 DEBUGGER 时才必要，否则不需要联机。

文　件	大小	
MTPLUS.COM	35K	可执行的编译程序
MTPLUS.000	13K	主控程序，调用复盖
MTPLUS.001	11K	编译程序的复盖区（不可少）
MTPLUS.002	7K	
MTPLUS.003	8K	
MTPLUS.004	17K	
MTPLUS.005	8K	
MTPLUS.006	6K	仅当使用调试程序时才需要

需要的总空间是 92K，如果不使用调试程序可减为 86K。如果不能把以上所有文件放在同一张盘上，可以用一种方法使主控文件和复盖文件驻留到联机磁盘上。要做到这一点，可通过编译开关，告诉编译程序，主控程序和复盖在不同的磁盘上。见第二章 2.2 节。

连接一个程序需要的最少文件是：

LINKMT.COM	11K	连接程序
PASLIB.ERL	19K	运行时支持库

如果 PASCAL 程序中有对于某些.ERL 文件的调用，则这些.ERL 文件也不可缺少。参见关于连接程序的第二章第三节，那里有所提供的.ERL 文件清单，并说明为解决调用问题，哪些文件需要连接到你的程序中去。

如果空间允许，理想的构成是将所有程序外加 MTERRS.TXT （错误信息文本文件）放在同一张盘上。如果有空间，数据可以放在同一张磁盘上，否则再使用一张联机磁盘。

第二章　PASCAL/MT⁺系统操作方法

本章阐述如何使用配给磁盘上的主要的 PASCAL 程序。

第一节　编译并运行一个示例程序

在编译运行本章的示例程序以前，请把包括本软件版本的所有磁盘信息复制一份。

以下按步骤介绍 PASCAL/MT⁺ 系统的基本操作。先生成一个系统编译磁盘，在 CP／M 操作系统下编译、连接、运行示例程序，如果你在编译或连接方面存在问题，请在使用或写磁盘之前，阅读本章的其余部分，以得到进一步的帮助。

下面假定你准备执行 PASCAL/MT⁺ 的计算机有二张 8 寸单密度软盘，如果你没有采用此种系统结构，则要做相应的调整。在运行软件以前，请读懂以下各步骤，这样，可以了解你自己正在做的是什么事。

下列文件是编译 CALC 必须要用到的：

MTPLUS.COM	可执行的编译程序
MTPLUS.000	主控程序
MTPLUS.001	编译程序的复盖
MTPLUS.002	
MTPLUS.003	
MTPLUS.004	
MTPLUS.005	

MTERRS.TXT不是必须要有的，但很有用，它包含了错误信息，下面 4 个文件用于连接CALC.ERL（由编译程序产生），从而产生可执行文件CALC.COM。

LINKMT.COM	连接程序
TRANCEND.ERL	超越函数模块
FPREALS.ERL	浮点实数库
PASLIB.ERL	PASCAL运行时库

1. 建立一个磁盘，它含有 LINKMT.COM, CALC.SRC, FPREALS.ERL, TRANCEND.ERL 和 PASLIB.ERL。这就包括了连接 CALC 所必需的全部程序。

2. 把含有COMPILER和CP／M操作系统的磁盘插入A驱动器中。

3. 把含有 CALC.SRC, LINKMT.COM 和 .ERL 文件的磁盘插入B驱动器中。

4. 启动系统，保持A盘联机。

5. 键入以下命令（这里〈CR〉表示在键盘上按回车键）。

MTPLUS B：CALC 〈CR〉

在编译时CALC不应出现编译格式和语义错误。如果出现此种错误，可能是你用了一个坏拷贝，请重新拷贝。由于内存所限和硬件结构的缘故，可能会出现内存容限错误或写磁盘错误。对于内存容限错误，不是显示 'recursion stack full'（递归栈已满），就是符号表溢

出的407号错误。

第二章2.5节描述了为了减少此种错误而采取的步骤。如果第一遍扫描中出现诸如'ERROR WRITING PASTEMP.TOK DISK PROBABLY FULL'之类的写盘错误，请使用第二章2.2节中讨论的$T命令开关。

在编译时可能遇到的另一种错误是复盖管理程序找不到复盖中的过程或复盖本身。这会在丢失的过程或复盖名后面出现一个'?'。如丢失的是复盖，请检查一下名称是否有错误，复盖是否在磁盘上。如丢失的是过程，则包含此过程的文件就被破坏。参见第二章2.5节。

6.编译程序被装入并显示信息'PASCAL/MT+ V.XX'。这里V是版本号，XX是版本分号。编译程序在处理CALC程序时会显示如下信息（或者是相似信息）。

```
Pascal/MT+      Release V.xx
(c)1981 Digital Research, Inc.
CP/M-80 versien
+++++                          {syntax/token file generation}
Source lines:   109
V.xx Phase 1
Available Memory: nnnnn {total symbol table space}
User Table Space: nnnnn {after predefined symbols}
****                           {one * for each routine body}
Remaining Memory: nnnnn  {after user symbols}
V.xx phase 2
8080
SUBREAL 18
ADDREAL 54
TF      88                     {decimal offset from beginning}
CALC    161
MENU    1096
CALCULAT
Lines:          109
Errors:           0
Code:          2050
Date:            48
Pascal/MT+ V.xx Compilation Completed
```

7.编译结束以后，可键入"DIR B:CALC.ERL"，如果系统显示B:CALC.ERL，则表明编译程序正确地把CALC.ERL文件放在目的盘上了。

8.现在连接CALC。键入"B:"从而和B驱动器联机。出现提示B»。键入下列命令：

LINKMT CALC, TRANCEND, FPREALS, PASLIB/S〈CR〉连接程序不应在所连接的.ERL文件中发现错误。如出现如'Duplicated symbol'（重复符号）或者"Undefined Symbol"（未定义符号），则表明有些.ERL文件坏了，要重新从主盘上把所有文件拷贝一遍，再重新编译连接一次。如果出现写盘错误，很可能是因为目的盘上空间不够，如果一切正常，可以看到

以下输出：

```
LINK/MT+™ V.xx
Processing file-    CALC        .ERL
Processing file-    TRANCEND .ERL
Processing file-    FPREALS   .ERL
Processing file-    PASLIB      .ERL
Undefined Symbols:
No Undefined Symbols
nnnn (decimal) records written to CALC     .COM
Total Data:   nunnH bytes
Total Code:   nnnnH bytes
Remaining:   nnnnH bytes
Link/MT+ Processing completed
```

9.键入 "DIR CALC.COM" 验证连接程序是否把 CALC.COM 文件放在目的盘上了。CP/M 会响应：

```
B:CALC.COM
```

10.现在清运行程序。CALC的输出和本章描述的用户输入如下。用户输入的放在单引号内。CALC使用补缺ISO标准格式，你可以在自己的程序中借助格式输出语句进行修改。

```
'B:CALC'
ENTER FIRST OPERAND? '5.5<CR>'
R1= 5.50000E+00
ENTER SECOND OPERAND? '99.256<CR>'
R2= 9.92560E+01
ENTER OPERATOR
S:SIN C:COS A:ARCTAN L:LN E:EXP 1:SQR SSQRT
+,-,*,/ ARITHMETIC OPERATORS
M:NEGATE
=:EQUAL  N:NOT EQUAL
<:LESS THAN  >:GREATER THEN
Z:LESS THAN OR EQUAL TO
G:GREATER THAN OR EQUAL TO
4:TRUNC  5:ROUND
?'+'
   104.756
TYPE <ESCAPE> TO STOP
```

第二节　编译操作

2.1　请求调用和文件名

编译程序的名称是 MTPLUS.COM，它使用了主控程序MTPLUS.000和 5 个复盖，外加调试复盖。输入文件可以放在任意盘上，采用 CP/M 规定的标准名称。输入文件必须以回车、换行和 CTRL-Z 结束。文件名长度为1到8个字符，前面可冠以驱动器号（'A' … 'P'，或'@'，表示现行盘），文件名的扩展名任意。如果没指定扩展名（如 TEST1），编译程序就先查找扩展名为 SRC 的文件，继而查找扩展名为 .PAS 的文件。如果不匹配会显示错误信息：'Unable to open the input file'（不能打开输入文件）。MTPLUS 产生的浮动文件〈文件名〉.ERL 要通过 LINKMT+ 与运行时软件包连接起来。

执行 PASCAL/MT+编译时，键入

MTPLUS〈文件名〉（可选参数以'$'或'*'开头）

例如'MTPLUS CALC'表示把 CALC.ERL 放在补缺磁盘上，'MTPLUS CALC $RB'表示把 CALC.ERL 放在 B 盘上。连接程序的详细介绍见第二章第三节。

2.2 编译程序命令行开关

命令行上输入文件以后可跟一些可选开关，编译程序接受这些开关。选择开关是以'$'字符或'*'字符开始的字串形式，而且单字母后面可跟多个参数字符。参数字符串以 $ 开始到行结束，空格忽略。例如 $PXRB和 $PX RBL一样。下面给出这些开关及其补缺值。'd'参数指出输出文件被转向何处。如果'd'为'@'或'A' … 'O'，则输出文件送到标准的 CP／M 磁盘驱动器上。如果'd'为'X'，输出文件被转向控制台。

开关	含义	补缺值
Rd	把.ERL 文件放在'd：'上。d = @, A…O	把.ERL 和源文件放在一起
Od	把主控制程序 MTPLUS.000 和 MTPLUS.001 到 MTPLUS.006 放在'd：'上。d = @, A…O	主控程序和复盖程序放在补缺磁盘上
Pd	把列表文件.PRN 放在磁盘'd：'上。d = @, A…O, P, X	无.PRN 文件
X	产生包含反汇编记录的扩展.ERL 文件	不产生扩展 REL 文件
Ed	把 MTERRS.TXT 放在'd：'上 d = @, A…O	MTERRS 放在补缺盘上
D	产生目的码中的调试信息，把 PSY 文件写到由R开关指定的磁盘上去	既不产生调试程序信息，也不产生.PSY 文件
Td	把PASTEMP.TOK放在'd：'上。d = @, A…O	PASTEMP.TOK 放在补缺盘上
Q	去掉任何不必要的控制台信息	编译程序信息繁多
C	出错时继续进行。补缺时每出现一个错误，暂停一下，由操作人员处理错误	编译程序停下来，询问每一个错误
A	编译结束时自动调用连接程序。需要一个和输入程序同名的.CMD 文件。连接程序名只能是'LINKMT.COM'	编译程序不实现自动连接
B	使用 BCD 而不是浮点实数	浮点实数
Z	产生 Z80 优化码	只产生 8080 代码
V	把源文件中出现的每一个过程和函数的名字打印出来，帮助确定第 0 阶段中出错的位置	不打印过程、函数名
@	使字符'@'等同于字符'^'	两者不等同

下面给出一个例子：执行编译程序，从A驱动器盘上读源文件，把.ERL文件放在B：驱动器上，.PRN文件送控制台，并自动调用连接程序。

```
MTPLUS  A:TESTPROG  $RBPXA
```

2.3 编译用数据

PASCAL／MT+编译程序在编译前两阶段（0和1）中周期性输出字符，让用户知道程序正在工作。示例编译程序的输出见第二章第一节CALC的编译。

第0阶段中每完成16行源代码，语法扫描就把一个'+'送到控制台上。第1阶段开始时显示可用内存空间，这是生成符号表之前的内存字节数（十进制）。根据预先定义的标识符，符号表大约要占据4K空间，删去无用的内部子程序可以改变此空间大小，见第二章2.4节。此阶段结束时显示用户符号表空间（十进制），这部分空间是编译符号表被装入以后留给用户符号和编译程序递归堆栈用的。每发现一个过程或函数就向控制台送一个'*'号。第1阶段结束时显示内存中未用字节数（十进制）。

第2阶段产生代码。每遇到一个过程显示其名称，这样用户可以知道编译程序在程序编译时处理到哪里了。过程名之后给出该模块的起址的偏量。实际遇到过程体时才会有输出，例如A包含B，B包含C，则输出的先后秩序是C、B、A。连接程序开关／M将显示各模块中过程的绝对位置。遇到错误显示出错行。如果有MTERRS文件，还会在屏幕底部给出错误描述，否则只给出错误号，根据此错误号在附录中可查出错误含义。进一步信息见第二章2.5节。编译结束时显示如下几行信息：

Cines：被编译源程序的行数（十进制）

Errors：发现错误个数

Code：生成代码字节数（十进制）

data：保留数据字节数（十进制）

2.4 编译程序触发开关

编译程序触发开关告诉编译程序，用户希望启动或阻止某些可选项。触发开关的格式为（*$——*）或{$——}。其中的空格用触发开关填充。每行允许有一个触发开关。例如（*$E+*）合法，而（*$E+$S+*）不合法。编译程序不接受关键字母前的空格、尾部空格或名字内嵌入的空格，不过它可以跳过前面出现的空格。{$E+}和{$E+}一样，但对{$E+}就不予理睬了。

示例：
```
(*$E+*)
{$P}
{$I  D:USERFILE.LIB}
```

2.4.1 入口记录生成（E）

$E控制浮动文件入口记录的产生，使得全局变量和所有过程、函数作为入口使用，也就是说，在其他模块中，可以通过EXTERNAL宣称来调用。$E-结束入口记录生成，使变量过程和函数在逻辑上独享。补缺值$E+，该触发开关可以随意使用。

2.4.2 递归和堆栈结构分配（S）

$S可用于过程/函数参数和局部变量的堆栈结构分配。如果模块和程序中有递归的过程或函数，必须使用此开关。该触发开关必须在关键字PROGRAM或MODULE之前设置，在独立编译单元内不能撤去。不论是程序还是模块，全局变量总是静态分配，用到$S的模块可

以和没用到$S的模块联接起来，形成一个程序。

2.4.3 包含文件（I）

$I〈文件名〉产生对包含在PASCAL源语句序列中命名的文件的编译。文件名格式包括驱动器号和CP／M标准格式扩展名。若不指定扩展名，就假定从主文件名得出扩展名。在输入的文件结束时必须有回车、换行、CTRL-Z。嵌套的包含语句（被包括了的源文件中又包括了另一源文件）是不合法的。

2.4.4 设置堆栈指针<Z>

在非CP／M环境下，$Znnnn用于将堆栈指针初始化为nnnn。而在CP／M环境下，硬件堆栈的初始化工作是通过将绝对地址0006中的内容装入堆栈指针寄存器完成的。若使用$Z，那么CP／M环境下的初始化工作被强制进行。该开关必须设置在主程序中BEGIN的前面。它不必在各子模块中出现，在主程序中亦只可出现一次。

2.4.5 严格类型和可移植性检测/(T、W)

$T+，$T-，$W+及$W-对严格类型检测/非移植性提示进行控制。这些功能是成对相关的。即：严格类型检查隐含着警告非移植性的使用，反之亦然。其补缺状态为$T-($W-)，这时，类型检查被释放，而且警告信息提示亦不会产生。在检查状态下，源程序编译时遇到非移植情况，则会产生500号错误信息。这是因为字串数据类型是非标准类型，所以在W／T开关存在状态下，字串操作会导致上述错误的产生。

设计该开关还用于检查与ISO标准PASCAL的兼容性，它并不根据附录中所列的所有可移植性功能进行检查，这是因为附录中所列的大部份是相关实现的或者是软件过程实现。

该开关可根据用户的要求在源程序代码中设置或撤销。

2.4.6 运行时范围检查<R>

$R+和$R-控制编译程序生成对应的运行代码，这些代码将对数组下标进行范围检查，并将其存放在下标变量中。该开关的补缺状态是$R-。该开关可在整个源程序代码中按用户要求建立或撤销。详见第四章。

2.4.7 运行时的例外检查<X>

当整型数、实型数出现除零、字串上溢、实数上溢或下溢时，执行这种例外检测。$X+、$X-控制编译程序生成运行时检查错误，处理错误的运行代码。该开关的补缺状态为$X+，而在执行状态下，是不可撤销的。第四章中对此类运行错误处理及选择有较详细的论述。

2.4.8 列表控制<L、P>

$P，$L+及$L-控制编译的第一遍扫描所产生的表格。$P产生一个插到.PRN文件中去的格式馈给字符（CHR(12)）。$L+及$L-可控制在整个源程序中建立或撤销表格，并可以在任何需要的时候使用。

2.4.9 减缩空间：实数运算<C>

用户可在做实数运算时使用$Cn，达到减缩运行目的码时内存的需要量。在需要时，用户可以指定一个重起动指令号，编译程序把所有对@XOP过程、实数实用程序装入和实数存放过程的调用转向重起动指令，这样就使得所有三字节调用指令转化为单字节调用指令。用户指定'n'值（0～7），编译程序就会产生RSTn指令。在CP/M环境下，由于重起动入口是0和7，所以各个重起动入口无效。使用MP/M^{TM}者及其他用户应查阅有关硬件资料，以获得较详细的说明。

使用$C，要求$C不但要在主程序中出现，而且要在用到这一功能的模块中也出现。前者是为了使编译在调用@XOP时生成安装重起动向量的代码及重起动指令，而后者则是为了当模块实数运算时生成重起动指令。

2.4.10 减缩空间：递归<Qn>

$Qn与$Cn的情况相似，它对递归模块进行同样的控制。n的范围是0～7。对于用来安装和存储的过程@DYN的每个调用，均被转向重起动入口。例如，一旦给定（*$Q5*），则转向重起动入口号5。

使用$Q时，要求$Q不但在主程序中出现，而且要在用到这一功能的模块中出现。前者是为了使编译程序在调用@DYN时生成安装重起动向量的代码及重起动指令。而要求在使用递归的模块中出现是为了产生重起动指令。

2.4.11 符号表空间减缩<K>

$Kn用来将不用的内部函数定义从符号表中清除掉，以便为用户符号表腾出空间。对已清除的符号表进行查询或调用都会产生'undefined identifier'错误信息。这与连接程序的装载内容无关。n（0～15）值用于控制各组过程，它们可以任意组合，但必须出现在词PROGRAM或MODULE之前，否则无效。每个注释中只允许用一个$K。n值选择如下：

组	被移动过程
0	ROUND, TRUNC, EXP, IN ARCTAN SQRT, COS, SIN
1	COPY, INSERT, POS, DELETE LENGTH CONCAT
2	GNB, WNB, CLOSEDEL, OPENX, BLOCKREAD BLOCKWRITE
3	CLOSE, OPEN, PURGE, CHAIN, CREATE
4	WRD, HI, LO, SWAP, ADDR, SIZEOF, INLINE, EXIT, PACK, UNPACK
5	IORESULT, PAGE, NEW, DISPOSE
6	SUCC, PRED, EOF, EOLN
7	TSTBIT, CLRBIT, SETBIT, SHR, SHL
8	RESET, REWRITE, GET, PUT, ASSIGN, MOVELEFT, MOVERIGHT, FILLCHAR
9	READ, READLN
10	WRITE, WRITELN
11	<unuced>
12	MEMAVAIL, MAXAVAIL
13	SEEKREAD, SEEKWRITE
14	RIM85, SIM85, WAIT
15	READHEX, WRITEHEX

2.4.12 触发开关功能简述

编译开关		说明	补缺
$E	+／-	控制入口生成	$E+
$S	×／-	控制递归／静态变量	$S-
$I	<名>	将另一源程序引到输入数据中，例{$IXXX.LIB}	

$R	+／−	控制检查代码的范围	$R-
$T	+／−		$T-
$W	+／−	严格类型检测控制及生成警告信息控制	$W-
$X	+／−	控制特殊的检查代码	$X-
$P		在 .PRN 文件中引入一个格式馈送	
$L	+／−	控制原代码列表	$L-
$Kn		允许删除内部函数过程，为了节省符号表所占空间 (n = 0…7)	
$Z	$nnnn	将硬件堆栈预置为 nnnnH（开始执行时，补缺值为 0006单元内容）	
$Cn		在实数操作时使用 RSTn 指令（补缺方式是使用 CALL 指令）	
$Qn		在递归环境下用 RSTn 指令进行装载及存储（补缺方式是使用 CALL 指令）	

2.5 错误信息

编译错误编号与提示信息与 Jensen 和 Wirth写的'User Manual and Report' 一书中的相同。错误信息、简单的说明及导致错误的部分原因请参阅有关附录。

出错时，编译程序会询问你是中断编译还是继续编译（在编译命令行中，如使用了第二章 2.2 节介绍的 C 开关，则例外）。

编译程序如找不到复盖程序或复盖中的某过程，即会显示一个问号，紧跟着显示出未找到的复盖名或过程名。例如：

```
? A:  MTPLUS.003   {找不到文件的情况}
? "PH2TERM"   {指坏文件的情况}
```

未找到复盖文件，可检查一下文件名是否正确，或是否联机（在盘上）。这时可能需要 Od 开关（见第二章 2.2 节）。如果过程未找到，则说明你用了一个已坏的复盖文本。

下面这些过程当编译程序无法在复盖入口表中安装它们时，则被显示出来，名字前面的号是包含该过程的复盖组号。

```
001 INITIAL
002 PHASEI
003 PH21NIT
004 BLK
005 PH2TERM
006 DBGWRITE
```

错误 407，符号表上溢：发生在当符号表中没有足够的空间存放现行符号时，出现在编译第一阶段。这可通过下述方法加以解决：1) 使用 $K 开关；2) 将程序分成若干个模块。

递归堆栈上溢：发生在编译第二阶段。当编译程序的表达式求值堆栈与符号表发生冲突时，就会出现递归堆栈上溢出，解决的方法是：1) 简化表达式；2) 将 CASE 语句改为 IF-THEN-ELSE 语句。这是因为由 CASE 语句产生的跳转表可能很大。

第三节　连接程序操作

3.1　请求调用

键入　LINKMT、空格、以逗号隔开的主程序名及各模块名，即可实现对连接程序的调用。下面给出连接程序命令行的语法格式：

LINKMT　{<主程序>=}　<主程序>，<各模块>，PASLIB　{<各开关>}

被连接文件的扩展名必须是.REL（下一节要介绍.CMD 文件除外），但扩展名 .ERL 不须在命令行中出现。作为输入给连接程序的文件名至多不得超过 32 个，输出文件被直接放到主程序所指定的盘片上，如果用户在主程序前指定一个输出文件名再跟一等号，则输出文件被放到等号前主程序所指的盘上。每一个被连接的文件名前均可以指定盘号。

例如：

LINKMT　CALC,TRANCEND,FPREALS,PASLIB/S LINKMT B：CALC=CALC, B：TRANCEND, FPREALS, PASLIB/S（CALC·COM文件写到B盘上）

3.2　连接程序选择开关

连接程序允许用户在命令行中文件名后面放置一些开关。每个开关前面都使用“/”号，可用的单字母参量符号为 P、D、V、X、O，仅仅/S 开关对安放位置有要求。

3.2.1　运行时库搜索开关</S>

前一节的例子说明了／S 的用法。／S 开关使连接程序去查找并连接那些仅在运行中用到的 PASLIB 中的模块，／S 开关在命令行中有固定的位置：在连接命令的命令行中，它必须紧跟在运行库名的后面。PASLIB 和 FPREALS 是专门建立的库，所以是可查找的。而其他由系统提供的.ERL 文件除外部指定的外，都不能查找。用户建立的模块除了按第二章第六节描述的方法经 LIB／MT 处理过的以外，都不能查找。在库中，模块的秩序很重要每个可查找库必须以正确的秩序包含过程，最后跟有一个／S 来查找第二章第六节。

3.2.2　内存映象</M>

参数列表中最后一个文件名后面跟一个／M 开关，则产生映象到控制台。

3.2.3　装配映象</L>,</E>

最后一个模块名后面跟／L 开关，则使连接程序在连接各模块的过程中显示模块代码及数据位置，而跟／E 开关，则显示所有过程名，包括那些运行库过程名中以$、？、@开头的过程名。

3.2.4　程序</P>及数据</D>起址

为了提供浮动目的码及数据区，连接程序提供了／P 和／D 开关。／P 开关控制目的码的（ROM）位置，而／D开关则控制数据区（RAM）的位置。

格式为：／P：nnnn 或／D：nnnn

这里 nnnn 是 0～FFFF 之间的十六进制数。这两个开关用于主程序和复盖的连接。

如果使用／D 开关，在连接时还可以增加很多空间，这是因为在连接数据时，它不会与目的代码掺合在一起。使用该开关是解决内存限制的一种途径。这里请用户注意：在这种情况下，由于系统是在装配程序将数据区清零的情况下才使得该功能发挥作用，所以不可能得到局部文件操作。

采用／P 开关和/D 开关并不会使连接程序在·COM 文件开始处留下空余空间。其他连接程序（特别是 L80™）将会产生一些有效的磁盘空间，使程序在 CP／M 环境下被装配在适当的地址上。LINK／MT⁺的特点是：如果使用／P 开关，则说明用户确想将程序移到另一个系统下运行。

例如：如果用户给定／P：8000，那么.COM 文件的第一个字节将被置于地址 8000H 处，且该字节前面并不是 32K 零。

／P 及／D 在最后一个被装配的浮动文件后面给定，并且可以按任意顺序给定。

3.2.5 HEX 文件输出＜／H＞

／H：nnnn 开关允许连接程序产生一个.HEX 文件，而不是·COM 文件，nnnn 为十六进制数，并与补缺的重新装载值（100H）完全无关（亦可用／P 加以控制）。例如，用户可以将程序重定位在 1D00H 来执行，而产生的·HEX 文件地址起址于 8000H（这时用户应使用／P：1D00／H：8000）。

3.2.6 连接程序输入命令文件＜／F＞

在 CP／M 环境下；用户必须专门使用 SUBMIT，以便重复键入某些命令序列。如一再将多个文件连接在一起。

LINK／MT⁺连接程序允许用户将这些数据输入到一个文件中去，而使连接程序去处理该文件中的文件名。这种处理方式显然比 SUBMIT 要快。但是用户必须将这种文件的扩展名定为.CMD，并在该文件名后加上开关／F。如：

…CFILES／F

连接程序即从该文件中获得输入并处理这些文件名。这些文件名可以从终端输入，在同一行中用逗号隔开，也可以一行一个文件名或一行一组文件名。从文件被输入时起，从左到右直到／F 联在一块儿。连接程序的整个输入缓冲区可长达 256 字节。下面给出一个使用的.CMD文件将 CALC、TRANCEND、FPREALS 及 PASLIB 文件连成为.COM 文件的例子。连接命令为：

LINKMT CALC／F／L

这时在 PASLIB 中仅仅查找那些用到的模块，同时产生连接映象。其中 CALC 包括：A：CALC，D：TRANCEND，F：FPRELS，B：PASLIB／S。

3.2.7 复盖区开关

当连接程序处理采用了复盖技术的主控程序及处理某一复盖文件时，有三个开关供使用。在第三章第三节中对复盖有详细的讨论，这里仅叙述一下复盖开关的意思：

第一个：／Vm：nnnn 给定复盖区地址。

第二个：／X：nnnn 告诉连接程序，该复盖需要多少外加的数据空间。其值加到 SYSMEM——堆栈指针上去。

第三个：／O：n 指定该复盖块号，并指出前面的文件名为主控程序符号表中的 n 号。

3.2.8 连接程序开关简述

／S 把前面的文件名作为库文件名进行查找，仅将需要的过程选抽出来。

／L 将被连接的模块列表

／M 以表格形式将所有入口列表

／E 除其他入口外还要列出以$、？或@开头的入口

/P：nnnn　将目的代码装载在 nnnnH

/D：nnnn　将数据区装载在 nnnnH

/W　建立一个与 SID™ 兼容的.SYM 文件（与.COM 文件写在同一磁盘上）

/H：nnnn　以 HEX 格式建立.HEX 输出文件，其中 nnnnH 为起始地址，这与/P 开关无关（若使用该开关，则不会产生.COM 文件）

/F　将前面的文件名作为包含输入文件名字的.CMD 文件（每行输入一个文件名）

/Vm：nnnn　复盖区起始地址

/X：nnnn　把复盖静态变量空间加到 SYSMEM 上

/O：n　给复盖区编号，并将前面的文件名作为主控程序符号表。n 值补缺范围为 1～50，但可以通过复盖处理程序扩充至 1～256

3.3 需要哪些浮动文件

各配给盘上存有许多将要连接到用户程序中去的 .ERL 文件，将这些文件编成一列过程组，编译程序根据用户程序的内容来选择过程组。下面列出各文件及对应的过程组清单。如果你的程序中有未定义的参变量属于这些内容之中，那么可以连接对应的浮动文件。

BCDREALS　BCD 实数，@XOP，@RRL，@WRL。

FPREALS　浮点实数，@XOP，@RRL，@WRL（可查找的）。

AMD1D

FPRTNS

REAL1O　硬件实数（见 AMD9511.CMD）。

任何实数装入、存放、赋值或输入／输出时，必须将上述各组过程之一连接进去。

TRANCEND　提供 SIN, COS, ARCTAN, LN, EXP, SQR 等函数功能。

TRAN9511　以 9511 的超越函数代替 TRANCEND。

RANDOM1O　SEEKREAD 及 SEEKWRITE 在此解决。

DEBUGGER　@NLN，@EXT，@ENT；使用调试开关时，产生这三个过程。若@XOP 及 @WRL 未定义，请参看第二章第五节。

PASL1B　各种比较，I／O，运算功能等。

3.4 连接程序的错误信息

连接过程中的错误信息包括：

"Unable to open input file * * * * * * *" 及 "Duplicate symbol * * * * * * *"。"Duplicate symbol"是指运行过程或变量与用户的过程或变量采用了同样的名称。连接程序还可能输出"Duplicate symbol detected"，这是由于上述信息可能未进入屏幕。

要是在程序连接过程中超出内存界限，可以采用/D 开关将数据从代码区移出去。用户可以将 D 开关值设得较大一些，试连接一下，查看代码到底占用多少空间，然后将 D 开关的 nnnn 值设置得略大于最后代码地址（应留一点代码空间），再重新连接。还有，对于那些不必要的标识符，可以在编译时用 $E 开关将其取消，这样可以节省连接符号表的空间。

错误信息"initialization of DSEG not allowed" 说明连接程序在数据段中发现了 DB 或 DW。

"Incompatible relocatable file"是指.ERL 文件是个坏文件，不可以连接，或汇编语言文

件中存在某些不合法的内容。

连接复盖文件时，当主控程序代码文件有错时，则会产生“System not found in symfile”这样的信息。

在连接汇编语言模块时，只允许有 50 个外部地址及偏置地址被连接程序连入其偏量表。如果超过这个数目，则会出现错误信息“External offset table overflow”。

3.5 可连接模块的属性

LINK/MT+ 只对 PASCAL/MT+ 主程序，PASCAL/MT+ 模块和由 M80™ 和 RMAC™ 建立的汇编语言模块进行连接。LINK/MT+ 还提供 PASCAL/MT+ 所需的 Microsoft 浮动格式，但不包括：

1）COMMON

2）在 DATA 段中初始化数据区

3）请求库搜索

同时，LINK/MT+ 要求数据长度和程序长度记录在被装入的第一个数据字节之前。这是 PASCAL/MT+ 编译程序、M80 及 RMAC 的情况，而不是 FORTRAN 之类的编译程序的情况。

第四节 反汇编程序

PASCAL/MT+ 软件包的反汇编程序把编译程序第一阶段生成的.PRN 文件（用 P 开关生成）及最后一阶段生成的.ERL 文件（用 X 开关生成）连接成人们可读的文件。它包括内部插有 PASCAL/MT+ 语句的汇编语言，这有助于分析编译程序生成的机器码，并为在机器码这一层上跟踪程序提供了必要的信息。由于编译程序生成的绝大部分代码是 8080 代码，这里仅提供了 8080 的反汇编程序。

指令

反汇编程序是一个独立的程序。可通过下列格式调用，包括反汇编程序名、给定的.PRN、.ERL 文件名以及输出文件名。如：

DIS8080〈输入文件名〉 {〈目标文件名〉 {, L = nnnn} }

反汇编程序以〈输入文件名〉作为前缀去寻找相应的.ERL 及 .PRN 文件。该.ERL 文件必须是用 X 开关专门编译而成的。否则，整个程序都被反汇编成 DB 语句。而以 X 开关生成的操作码记录是为反汇编程序备用的。.ERL 及 .PRN 文件可存放在任意一张磁盘上，但二者必须在同一盘上。目标文件名可以是 CP/M 文件名，也可以是 PASCAL/MT+ 设备名。如 CON：，LST：等，补缺目标名是 CON：。L = nnn 参数，允许用户指定输出设备上每页的行数。对于 11 英寸打印纸来讲，补缺值为 66行/页，即每英寸 6 行。这对于使用如 T.I.810 这样的打印机来说是很有用的，T.I.810 有一个 6 行/英寸，8 行/英寸的转换开关。选用 8 行/英寸时，应当指明有 88 行，这样，可以节省大量纸张。选用 L =（选用值）时，用户必须给出〈目标名〉。

反汇编程序产生错误信息是由于在传送的数据流中发现了某些不希望有的东西，而继续做下去，随着序列的间断会产生更多的错误。连接成功的.ERL 文件是不应有错误的。要纠正它，就要重新进行编译，以生成.ERL 文件，同时应切记，你是在反汇编 PASCAL 代码。

第五节 调试跟踪程序

称为 DEBUGGER.ERL 的可重入程序——PASCAL/MT+ debugger 程序是 PASCAL/MT+系统的一部分。它与运行文件库一起被连接成为目的程序。调试程序可显示变量、设置符号断点、单步执行一个语句、显示入口、显示符号表以及从过程或函数中退出显示。当调试一个由模块组成的程序时，在跟踪状态下，能显示当前行号，但是在每一个模块中，行号是重复的。如果用户仅仅用<RETURN>回答调试程序对符号表文件（.SYP）名的请求时，则调试程序可在非 CP/M 环境下运行。这仅仅禁止了符号功能，但仍由寻址功能保持着显示。

以下叙述如何把调试程序连入目的程序以及怎样运行调试程序。

5.1 指命

要将调试程序的信息连入目的程序，用户必须对 MTPLUS.COM 指定 D 命令行开关。这样做，编译程序就把.PSY 文件写在.ERL 文件所在的盘上，并在每一过程的开头和结尾处生成附加代码，为调试程序设逻辑断点做准备。.PSY 文件中包含了每一个过程函数及程序中所宣称的变量的记录。这些项中的每个地址区都是与模块相关的，LINK/MT+ 处理这些.PSY 文件，并生成一个包含每一个过程、函数和变量的绝对地址的.SYP 文件。然后调试程序利用这一.SYP 文件来显示符号变量、符号断点等等。

LINK/MT+ 根据编译程序生成的.PSY、.ERL 文件生成（如前述）.COM 文件及.SYP 文件。用户必须把调试程序作为第一个模块进行连接，以使程序运行时，调试程序即开始执行。下面是一个对调试程序进行连接的命令行实例：

```
LINK/MT  USERPROG=DEBUGGER,USERPROG,PASLIB/S
```

其中"USERPROG="使连接程序生成的.COM 文件命名为 USERPROG，而非 DEBUGGER。

由于 USERPROG 是由调试程序进行控制的，故当运行 USERPROG 时，调试程序首先要询问符号表文件名是什么，用户应当以某个.SYP 文件名作为回答，或按<RETURN>键表示没有符号表文件。然后调试程序以"+>"符号作为响应。输入 BEGIN命令，用户即可输入任意的调试命令，以便对被调试程序进行检查。

当debugger 用于递归环境时，因为局部变量不是存于固定存储区而是存于栈中，所以这些局部变量不被显示。

如果象上例那样，把 debugger 连起来，将有两个未定义符号：'@OXP' 和'@ WRL'，它们是用来写实数的。因为 USERPROG 不用于实数，实数的运行时程序包未连上，故使得上述过程无法定义。当然，这在运行时并不产生任何问题，因为并没有实数要显示。如果想显示实数，会使程序返回到操作系统。

5.2 命令格式

任何时候，debugger 都可以将数据项转换成用户所希望的形式（如整数转换成十进制，逻辑量转换成真/假值等）。如果无法转换，则 debugger 将用 HEX 和 ASCII 码显示数据。命令描述如下：

<name>项可以是一个变量名、一个过程/函数名或一个带前辍变量名。名字由 1 到 8 个字符组成，并跟有 PASCAL 编译器的词法。容许有下划线但可以忽略不予考虑（例如 A—B 和 AB 完全相同）。带前辍变量名是以过程/函数名作为前辍的变量标识符。这用来表示非递

归过程中的局部变量和参数。如果两个过程都含有相同的局部过程名，那么只有第一个过程的符号在连接时，会被显示出来。

〈num〉项可以是一个十进制数，也可以是一个十六进制数（前面要有'$'号）。十进制数的范围是0～32767，十六进制数的范围是0～FFFF。

〈name〉 ::=〈过程标识符〉:〈变量标识符〉|〈变量标识符〉|
〈num〉

〈num〉 ::= $〈十六进制数〉|〈十进制数〉

显示变量命令跟有一个参数〈parm〉，它由〈name〉、〈num〉、偏差量和一个间接符号所组成。

〈parm〉::=〔〈name〉|〈num〉〕{∧}{〔+/-〕〈num〉}偏差量由'+'号（补缺值）或'-'号跟一个〈num〉组成。〈num〉的值表示一个字节数，或是将这一偏差量加到〈parm〉的地址上去，或是从〈parm〉的地址上减去。间接符号'∧'用于指针变量，一旦用上它，则调试程序显示指针所指的数据，而不是指针本身。〈parm〉的例子如下：

```
(* pascal declarations: *)
TYPE
  PAOC=ARRAY [1..40] OF CHAR;
VAR
  ABC:INTEGER;
  PTR:PAOC;
```

〈parm〉的例子：

ABC	整数
PTR	PTR的内容
PTR∧	整个数组
ABC+10	ABC位置后10字节
PTR∧+10	PTR[11]
ABC-3	ABC位置前3字节
PTR∧-3	PAOC数组前3字节
$3FFD	绝对地址
$423B∧	由$423B指向的32个字节
$3FFD+$5B	在$4058处的32个字节
$423B∧+49	由$423B+49的内容指向的32个字节
PROC1:I	局部变量
PROC2:J∧+9	局部指针的偏差量

改变内存地址内容的命令是'SE〈parm〉'。此命令以十进制数值显示该地址的内容。输入一个十进制数（或者$跟上一个十六进制数）后加上一个回车则使输入值取代原来的值。如果这个输入值太大，无法容纳在一个字节里（大于255），则只取最后两位。然后，下一个字节的地址和内容又被显示，你可以继续更改。若不想再更改，键入'.'跟一个回车，就回到命令状态。这和DDT或SID中的set命令是一样的。

命 令	意 义
DV〈name〉{∧}	显示变量——显示〈name〉代表的变量。如果这是一个指针型变量，则指针内容被显示，如果后面跟有符号'∧'，则指针所指的数据被显示。

如果不是显示符号，而是显示记录中的域或数组元素，可用下面的命令：

DI〈parm〉	显示整数
DC〈parm〉	显示字符
DL〈parm〉	显示逻辑值（布尔型）
DR〈parm〉	显示实数
DB〈parm〉	显示字节
DW〈parm〉	显示字
DX〈parm〉{num}	扩展显示（结构类型） 这通常以 HEE/ASCII 格式显示，在内存显示时，num 是字节数，其补缺值是 320。

下面是用户程序可用的控制命令：

TR 或 T	跟踪——执行一行并返回
T〈num〉	跟踪〈num〉行并返回
BE	开始执行（从头开始运行程序）
GO	从断点处继续执行
SB〈name〉	在〈name〉过程的开头设断点
RB〈name〉	将〈name〉过程的断点撤消
E＋	在执行期间，能显示每一个过程或函数的引入和退出（补缺值 E＋）
E－	禁止显示引入/退出
PN	显示.SYP 文件中的过程名
VN〈name〉	显示〈name〉过程中的变量名
SE〈parm〉	修改地址〈parm〉中的内容，此命令用句号'.'结束。
？？	请求帮助！列出 DBUGHELP.TXT 中的所有命令。

第六节　库管理程序 LIB/MT+

PASCAL/MT+ 库管理程序是个有双重目的的程序，它的基本目的是将.ERL 文件逻辑地连接在一起以构成一个可查询文件库，如 PASLIB。其次是将 PASCAL/MT+.ERL 文件转化为与 Microsoft 连接程序兼容的文件，象 L80 和 Link80 等。

6.1 请求调用和输入文件

可以这样调用库管理程序

LIBMT〈文件名〉

LIB/MT+运行时需要一个'.BLD'文件作为输入。但当调用 LIBMT 时，输入的文件名不能带扩展名'BLD'。这个.BLD 文件含有一个输出文件名，后面跟着一个输入文件名的列

表，每行一个名字。输入文件可以是单个的模块，也可以是模块库。注意：带'X'选择编译出来的 PASCAL 模块（用于反汇编）可能不被执行。

6.2 使用库管理程序

下面是一个'.BLD'文件生成 LINK/MT+ 兼容库的例子：

```
MYLIB.ERL
MYMOD1.ERL
MYMOD2.ERL
MYMOD3.ERL
```

当 LIB/MT+ 运行时，MYLIB.ERL 的所有旧内容将被删去，将 $MYMOD_1$.ERL、$MYMOD_2$.ERL和$MYMOD_3$.ERL 文件读入并连接，输出到文件 MYLIB.ERL 中去。PASCAL/MT+ 模块、文件库和有关的汇编语言模块都将作为有效输入。必须提供文件的扩展名，但不一定是'.ERL'。而如果要用 LINK/MT+ 运行，则输出文件类型必须是'.ERL'。

连接程序 LINK/MT+ 是一次扫描的连接程序。当用/S 可选项时，意味着一个文件就是一个库，连接程序仅仅把那些在前面模块中用到的模块装上。所以，你库中模块的顺序是很重要的。如果要连接模块的顺序是 A、B、C，那末，模块 B 和 C 不能包含对模块 A 的调用，除非模块 A 被装上时，它们已先装上了。然而，对上面的顺序，模块 A 可以调用 B 和（或）C，因为这正导致连接程序去装配它们。记住，连接程序仅将整个模块从库中抽出，而不能将单个的过程从一个模块中抽出。当使用开关 S/时，所有的入口，包括代码和数据，都将成为检索的对象。由于是整个模块装入，所以一个模块仅需要一个入口。

因为 PASLIB 具有特殊的结构，所以它不能用库管理程序来改变。在连 PASLIB 之前，应将替换模块连上来分解那些在 PASLIB 被检索以前调用到的过程。如果这些替换过程在一个库中，最好的方法是不要搜索这些过程，因为在 PASLIB 被处理以前是无法调用这些替换过程的。

第三章　PASCAL/MT⁺语言扩充

此章的目的是描述 PASCAL/MT⁺ 语言的扩充功能和用途。

第一节　模块编译

PASCAL/MT⁺ 提供了一个灵活的模块编译系统。与其他系统（例如 UNITS）不同，PASCAL/MT⁺ 不需要花费许多预处理工作，就能把较大的整块程序很容易地转换成模块化程序。程序可以写得很大，然后在编译时再分成许多模块。PASCAL/MT⁺ 模块编译系统允许在任何模块中对其他模块中的过程和变量进行操作。一个编译开关（E+/-）可供用户使用，以保护某些变量和过程变为用户私有。关于 $E 开关的讨论，参阅第二章 2.4 节。

模块的结构类似于程序的结构。它以保留字 MODULE 开始，后面跟一个标识符和一个分号，例如 MODULE TEST1；并且以保留字 MODEND 跟一个句号结束。在这两行之间，程序员可以象在程序中那样，宣称标号、常数、类型、变量、过程、函数等。然而，与程序不同的是，在过程和函数宣称之后，没有 Begin .. END 程序块，仅有保留字 MODEND 加上一个句号。例如：

```
MODULE MOD1;
〈label, const, type, var declarations〉
〈procedure/function declarations and bodies〉
MODEND.
```

为了存取其他模块（或主程序）中的变量、过程和函数，必须添加一个新的保留字 EXTERNAL。使用它有下列两个目的：

第一，EXTERNAL 可放在一个冒号之后、宣称的全称变量类型之前，以表示此变量表并不实际存在于此模块中，而存在于另一个模块之中，因此，不需要为这些变量预留空间。例如：

```
I,J,K:EXTERNAL INTEGER;  (* in another module *)
R:    EXTERNAL RECORD    (*again in another module *)
                ...      (*some fields *)
              END;
```

用户必须保证这些宣称互相一致，因为编译程序和连接程序没有类型检查的功能。

第二，EXTERNAL 保留字用来宣称存在于别的模块中的过程和函数。这些宣称必须出现在此模块/程序中第一个通常的过程或函数宣称之前。外部过程仅可在程序或模块的全称层（最外层）进行宣称。

与变量宣称一样，PASCAL/MT⁺ 系统要求用户保证参数的数目和类型完全匹配，函数返回值的类型完全匹配，因为编译程序和连接程序没有模块间类型检查功能。外部过程不能用过程和函数作为参数。

用户要注意：在 PASCAL/MT⁺ 中，外部过程名只有 7 个有效字符而不是 8 个。当与汇编程序通过接口连接时，标识符的长度由于汇编语言所限，应该限制为 6 个。

下面是一个主程序和一个模块的骨架。主程序引用了模块中的变量和子程序，而模块也

引用了主程序中的变量和子程序。主程序和模块的区别仅在于主程序开头的16个字节的标题和跟在过程和函数后面的主程序体。

Main Program Example:

```
PROGRAM EXTERNAL_DEMO;
<label, constant, type declarations>
VAR
    I, J : INTEGER;    (* AVAILABLE IN OTHER MODULES *)
    K,L : EXTERNAL INTEGER; (* LOCATED ELSEWHERE *)
EXTERNAL PROCEDURE SORT(VAR Q : LIST;LEN : INTEGER);
EXTERNAL FUNCTION  IOTEST : INTEGER;
PROCEDURE PROC1;
BEGIN
   IF  IOTEST  =  1 THEN
      (* CALL AN EXTERNAL FUNC NORMALLY *)
     ...
END;
BEGIN
SORT(....)
(* CALL AN EXTERNAL PROC NORMALLY *)
END.
```

Module Example: (Note these are separate files)

```
MODULE MODULE_DEMO;
<label, const, type declarations>
VAR
 I, J : EXTERNAL  INTEGER;  (* USE  THOSE  FROM  MAIN
                               PROGRAM *)
 K,L : INTEGER;             (* DEFINE THESE HERE *)
EXTERNAL PROCEDURE PROC1;(* USE THE ONE FROM MAIN
                               PROG *)
PROCEDURE SORT(...);       (* DEFINE SORT HERE *)
     ...
FUNCTION IOTEST : INTEGER;  (* DEFINE IOTEST HERE *)
     ...
<maybe other procedures and functions here>
MODEND.
```

第二节 PASCAL/MT+汇编接口

这一节供那些想从 PASCAL/MT+ 程序中调用汇编程序的用户参考，包括汇编程序

表、命名约定、变量存取、参数传递约定，并介绍能与LINK/MT+连接的汇编语言特性的限制。

2.1 汇编程序

在PASCAL/MT+上使用的汇编程序必须产生象编译程序产生的那种浮动目的码。PASCAL/MT+系统的8080和Z 80文本可产生Microsoft兼容浮动文件。这两个汇编程序都可产生部分运行时程序库。LINK/MT+可以在这个连接程序的控制之下，处理兼容汇编程序产生的文件，但是其他连接程序不一定能连接PASCAL/MT+编译程序产生的目的码。参阅关于库管理程序LIB/MT+的说明，以使文件与其他连接程序兼容。

2.2 命名考虑

汇编程序和PASCAL/MT+编译程序都以浮动文件形式生成入口和外部调用记录。这些记录包含外部符号名。Microsoft格式允许多至7个字符的名字，PASCAL/MT+编译程序使用7个字符的名字，而大多数的汇编程序仅产生6个字符的名字。这意味着，在PASCAL/MT+程序中的一个变量，如想在汇编程序中按此变量名调用，则用户必须将变量名限制在6个字符之内。

另外，M80允许符号以$开头，RMAC允许符号以？开头，而在PASCAL/MT+中，这都是不允许的。M80认为$是有效字符，而RMAC则相反。这意味着，在M80中，符号A$B将如实地放入浮动文件中，而在RMAC中，这个符号放入浮动文件时将变为AB。当使用RMAC时，'$'可象下划线（_）那样使用，但不能传给M80。

2.3 变量存取

PASCAL程序和汇编程序之间的变量存取操作十分简单。

2.3.1 从PASCAL程序而来的汇编程序入口

要想在汇编程序中使用PASCAL程序中的变量或子程序，它们必须在汇编语言模块中被宣称成PUBLIC，就象在PASCAL/MT+中宣称EXTERNAL一样。本章最后附上的汇编模块和PASCAL程序说明了这一点。

2.3.2 从汇编程序而来的PASCAL程序入口

要想在PASCAL/MT+中使用从汇编语言而来的全称变量和子程序，用户必须在汇编程序中将名字定义为EXTRN，这样，在PASCAL程序中就可简单地象一个全称变量或子程序那样去使用（注意，必须使用$E+开关！）例：

```
; ASSEMBLY LANGUAGE PROGRAM FRAGMENT
  EXTRN    PQR
  LXI      H,PQR     ;GET ADDR OF PASCAL VARIABLE
  .
  .
  .
  END
(* PASCAL PROGRAM FRAGMENT *)
VAR (* IN GLOBALS *)
  PQR : INTEGER;          (* ACCESSABLE BY ASM ROUTINE *)
```

2.4 数据分配

为了按名字对变量进行存取，用户必须知道变量在内存中是怎样存储的。第五章第一节讨论了每一个内部纯量数据类型的内存分配和格式。全称变量区中的变量是按宣称的顺序分配内存的，但是同一个类型的变量是按逆序排放的（例如 A，B，C：INTEGER 分配时先分配 C，后面跟 B，再跟 A）。在内存中变量是紧挨在一起的，两个宣称之间不留空间。例：

```
A       :INTEGER;
B       :CHAR;
I,J,K   :BYTE;
L       :INTEGER;
STORAGE LAYOUT;
        +0 A LSB
        +1 A MSB
        +2 B
        +3 K
        +4 J
        +5 I
        +6 L LSB
        +7 L MSB
```

结构型数据类型，象数组、记录和集合需要另加解释。数组是按行存放的，这意味着数组 A：ARRAY〔1,3,1,3〕of char 将存放成：

```
+0 A〔1,1〕
+1 A〔1,2〕
+2 A〔1,3〕
+3 A〔2,1〕
+4 A〔2,2〕
+5 A〔2,3〕
+6 A〔3,1〕
+7 A〔3,2〕
+8 A〔3,3〕
```

这在逻辑上是一个一维向量数组。在 PASCAL/MT⁺ 中，所有的数组在逻辑上都是某种类型的一维数组。

记录象全称变量一样存放。

集合总是存于连续的 32 个字节中，集合中的每个元素占一位，每 8 个元素占一个字节，每个字节的最低一位存放应该存在这个字节里的那 8 个元素的头一个元素。例如：集合 'A' …… 'Z' 就是象下面这样存放的：

```
字节顺序号  00 01 02 03 04 05 06 07 08  09  0A 0B 0C 0D 0E……1F
存储内容    00 00 00 00 00 00 00 00 FE FF FF 07  00 00 00……00
```

元素 A（其 ASCII 码是 65）存在第 65 位上，可在第 08 号字节的第一位上找到。最后一个元素是 Z（ASCII 码是 90），存在第 90 位上，可在第 0B 号字节的第二位上找到。注意

每个字节的最低位是第 0 位。

2.5 参数传递

当在 PASCAL 程序中调用汇编子程序，或在汇编程序中调用 PASCAL 子程序时，参数是通过堆栈来传递的。作为子程序入口的栈顶放置返回地址，在返回地址下面是与宣称顺序相反的参数表：(A, B: INTEGER; C : CHAR)在栈中排成的结果是 C 在 B 之上，B 又在 A 之上，每个参数至少需要 16 位，即一个字 (WORD) 的栈空间。一个字符或者一个布尔量也要用 16 位传递，这时高位字节为 00。变量参数按地址传递，该地址表示实际变量的最低存储地址。

非纯量参数(不包括集合)总是用地址传递的，如果该参数是一个值参，则其代码由编译程序产生并在 PASCAL 程序中调用@ MVL 去移动数据。集合参数是在堆栈上传递参数值，并用@ SS2 过程去存放它们。集合是以低字节存在上面(低地址)、高字节存在下面(变地址)的形式存在栈里的。

下面的例子给出了一个过程入口的参数表：

```
PROCEDURE DEMO(I,J : INTEGER;VAR Q : STRING;C,D : CHAR);
AT ENTRY STACK :   +0    RETURN ADDRESS
                   +1    RETURN ADDRESS
                   +2    D
                   +3    BYTE OF 00
                   +4    C
                   +5    BYTE OF 00
                   +6    ADDRESS OF ACTUAL STRING
                   +7    ADDRESS OF ACTUAL STRING
                   +8    J (LSB)
                   +9    J (MSB)
                   +10   I (same as J)
                   +11   I (same as J)
```

在返回调用过程前，汇编程序必须将堆栈中的所有参数移出。

函数值压入栈内，在返回之前，它们按逻辑顺序放在返回地址之下，于是在返回到主程序之后，它们恰好处于栈顶。汇编函数仅能返回纯量类型，象整数、实数、布尔量和字符等。请参阅第三章 2.7 节汇编函数和 PASCAL 程序的例子。

2.6 限制

要想使程序能用 LINK/MT+ 连接，用户必须了解对于程序的一些限制：

a) COMMON 不再提供使用。

b) 不能再在汇编语言子程序中的 DSEG 中使用 DB 和 DW 命令，对于有操作系统的应用，可在 CSEG 中使用 DB 和 DW 命令。

C) LINK/MT+ 不再提供请求库查询的功能。

2.7 汇编程序接口实例

下面是一个汇编模块和一个使用这个模块的 PASCAL 程序。这个汇编模块模拟 BASIC 中的 PEEK 和 POKE 过程。PEEK 返回输入地址中的字节，而 POKE 将输入值放入所指定

的字节之中。

PASCAL 程序（用到下面汇编程序模块 PEEK, POKE）

```
PROGRAM PEEK_POKE;
TYPE
        BYTEPTR = ^BYTE;
VAR
        ADDRESS : INTEGER;
        CHOICE : INTEGER;
        BBB : BYTE;
        PPP : BYTEPTR;
EXTERNAL PROCEDURE POKE (B : BYTE; P : BYTEPTR);
EXTERNAL FUNCTION PEEK (P : BYTEPTR) : BYTE;
BEGIN
   REPEAT
      WRITE('Address? (use hex for large numbers)');
      READLN(ADDRESS);
   PPP : = ADDRESS;   {ONLY ALLOWED IN PASCAL/MT + 8080}
   WRITE('1 (Peek OR  2) Poke');
   READLN(CHOICE);
   IF   CHOICE = 1 THEN
     WRITELN (ADDRESS,'contains', PEEK(PPP))
   ELSE
      IF   CHOICE = 2 THEN
         BEGIN
           WRITE('Enter byte of data : ');
           READLN(BBB);
           POKE(BBB, PPP)
         END
   UNTIL FALSE
END
```

程序 PEEK_POKE 所用的汇编模块（用 8080 助记码编写，用浮动汇编器汇编）

```
   PUBLIC PEEK
   PUBLIC POKE
 ;Peek returns the byte found in the address passed on the stack
 ;It is declared as an external in a Pascal program as :
 ;EXTERNAL FUNCTION PEEK(P : BYTEPTR) : BYTE
 PEEK :
   POP B     ;RETURN ADDRESS INTO BC
   POP H     ;POINTER TO BYTE INTO HL
```

```
  MOV E,M ;MOVE CONTENTS OF MEMORY POINTED TO BY HL
          INTO E
  MVI D,0 ;PUT A 00 INTO D
  PUSH D  ;RETURN FUNCTION VALUE
  PUSH B  ;PUT RETURN ADDRESS ON STACK
  RET     ;RETURN TO CALLER (NO PARAMETERS LEFT ON
          STACK)
;Poke places a byte into memory
;It is declared as an external in a Pascal program as:
;EXTERNAL PROCEDURE POKE(B:BYTE; P:BYTEPTR);
POKE:
  POP B   ;GET RETURN ADDRESS INTO BC
  POP H   ;P,THE BYTE POINTER IS PUT INTO HL
  POP D   ;REGISTER E GETS THE BYTE,D GETS THE EXTRA
          BYTE OF 00
  MOV M,E ;PUT E INTO MEMORY POINTED TO BY HL
  PUSH B  ;RETURN ADDRESS ON TOP OF STACK
  RET     ;RETURN TO CALLER (NO PARAMETERS LEFT ON
          STACK)
  END
```

第三节 PASCAL/MT+ 复盖功能

复盖系统允许用户编写若干个程序，并连接在一起，需要时各个程序可自动从盘上调入内存。使用复盖系统，程序不再需要整个地放在内存之中，那些不常使用的模块和不必驻留内存的模块可以存在盘上的“复盖区域”之中，仅仅在调用时再装入内存。

如果对 PASCAL/MT+ 不熟悉，建议开始编程时不要采用复盖技术。只有有经验的 PASCAL/MT+ 程序员才能得心应手地使用外部过程和变量，懂得怎样使用连接程序，以及熟悉怎样从手册中寻找所需要的内容。

PASCAL/MT+的复盖系统可以概括如下：

- 多至 255 个的复盖组
- 多至 16 个的复盖区域
- 一个复盖模块可调用另一个复盖模块
- 在同一个区域中，可嵌套调用复盖模块
- 仅在需要时才从磁盘上将复盖模块装入内存
- 复盖模块可由任意多个模块组成
- 将一个过程连接到一个复盖模块中是通过传递名字来实现的

• 可完成对主控程序中的过程、函数和变量（包括运行时过程软件包）的存取

• 实际运用时用户可指定含有复盖模块的驱动器

3.1 复盖的定义

下面先讨论复盖技术中经常使用的术语。

1、主控程序——总是以.COM 文件形式驻留在内存中的那一部分程序。它包括主程序、主程序需要 PASLIB 中的运行时过程和复盖所需要的运行时过程。

2、复盖模块组——一组可以是互不相干的模块，它们按指定的方法连在一起（见下文），并包含在一个带数字扩展名的文件之中（例如 PROG.001）。

3、复盖区域——内存中的一个区域，在此区域中，复盖管理程序在需要时装载复盖组，复盖组在内存的装载位置和所需空间必须由用户在连接时确定、计划好。

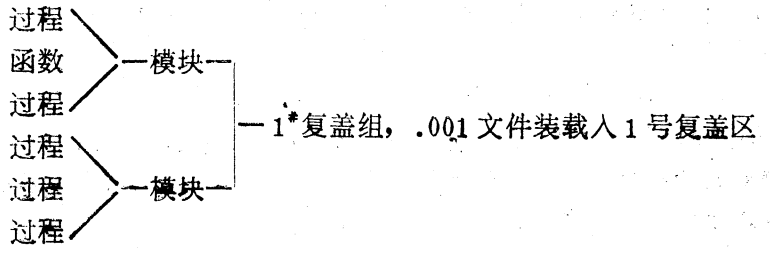
4、复盖模块静态变量——指在静态编译下被连接到某一个复盖内的模块中，模块的全程变量或过程的局部变量。递归程序可以减少静态数据所占的空间，但不一定会消除静态数据，因为通过复盖连接的运行目标代码可以含有静态数据（见/X开关）。

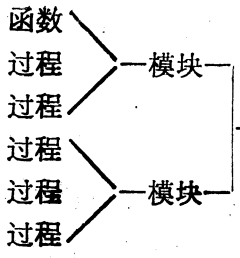
复盖系统允许多至 255 个复盖组，每个复盖组可包括一个或多个含有过程和函数的模块（模块可由 PASCAL 语言或汇编语言写成）。PASCAL/MT+ 复盖系统允许复盖组含有任意多个入口（模块中的一个过程或函数称为一个入口），所有这些入口都可以被主控程序和其他复盖模块所存取，每个入口都唯一地用其名字加以识别。由于连接程序和浮动格式的要求，复盖过程名、函数名的有效长度被限制在 7 个字符之内。

系统配给盘上的 PASCAL 复盖管理程序为 1 至 50 号复盖组开辟空间，这是通过在运行时保留 200 字节的空间来实现的。如果需要更多的空间以容纳复盖组，用户可以修改复盖管理程序的源程序，重新汇编并进行连接。复盖组的编号不必是连续的，例如，如果需要在三个不同的复盖区中使用三个复盖组，那么 1、17、33 或是其他的编号组合都可以将复盖组装入不同的复盖区中。

PASCAL/MT+ 系统的复盖组通常可在一个复盖区中操作，但若使用多个复盖区，则可以执行各种结构的程序。复盖系统提供多至 16 个复盖区，这些复盖区是通过复盖号来工作的。当确定了复盖号后，一个给定的复盖模块就被装载到该复盖区中。如果一个复盖区的地址未加说明，则补缺值为 1 号复盖区。1 至 16 号复盖模块组总是装载到 1 号复盖区中，17 至 32 号复盖模块组装载到 2 号复盖区中……等等，241 至 256 号复盖模块组装载到第 16号复盖区中。用户必须自己决定复盖区的地址和空间，并确保被装载的复盖模块组不会超过复盖区的空间。复盖装载程序以 128 字节为单位将复盖模块读入内存，因此，要为复盖模块组保留空间时，复盖区应有额外的空间。第 1 号复盖区必须加以说明，其余各复盖区是否说明由用户自定。

过程、函数、模块及复盖组之间的关系图示如下：





3.2 连接复盖组和主控程序

下面的三小节将指导用户如何使用连接程序。所描述的连接命令开关在使用时设置的次序无关紧要。第三章 3.2.4 节指导用户连接主控程序；3.2.5 节指导用户连接一个复盖模块组；3.2.7 节是上述几小节的总结，并给出连接一个使用复盖模块组的主控程序的步骤。下面的段落中，有些叙述可能会重复，但我们认为，充分的解释将有助于所有的用户理解复盖系统。

3.2.1 复盖区开关/V

/Vn：mmmm 开关通知复盖管理程序第 n 号复盖区起始于内存地址 mmmm，本开关只能用于连接主控程序。由于可使用多至 16 个复盖区，所以在连接主控程序时，本开关可使用多达 16 次。它至少要使用一次，以便对所有的复盖模块组都给出补缺的复盖区。

为确定/V 开关中的 mmmm 值，可先用 PASLIB 连接主控程序，如果使用了/D 开关，那么主控程序的目的代码所占空间加上 100H 就可作为一个复盖区的起始地址；如果不使用/D 开关，则除主控程序的目的代码所占空间加上 100H 以外，还要加上主控程序的数据空间才能作为 mmmm 的值。

3.2.2 复盖组和 SYM 开关/O

/O：n 开关通知连接程序，开关前的文件是一个 .SYM 文件，应被引入连接程序符号表作为一个入口。.SYM 文件是连接主控程序时产生的，它含有主控程序中的所有入口。由于是通过符号形式和主控程序相连接，所以如果某一个复盖模块组作了修改，只需重新连接该复盖模块组即可（除非该复盖模块组的数据区超过了连接主控程序时所作的限制）。/O：n 开关同时通知连接程序当前连接的是第 n 号复盖模块组。n 被用来构成文件名。本开关仅仅用于连接复盖模块组，第 1 至 16 号复盖模块组装入第 1 号复盖区，第 17～32 号复盖模块组装入第 2 号复盖区，余类推。

3.2.3 复盖局部区域开关/X

/X：nnnn 开关可用于连接主控程序和各复盖模块组。当用于连接主控程序时，/X：nnnn 开关通知连接程序要保留 nnnn 个字节的内存给复盖组作为局部区域使用。这个保留空间被加到@INT 中的整型变量 SYSTEM 中去，以保证 HEAP 指针（SYSTEM）的基址是正确的。用户必须确保这个局部区域足够使用，只要对每次连接复盖模块组时所需的数据空间加以检查，就可做到这一点。如果没有使用/X 开关，则复盖静态数据紧接在主控程序数据区后面开始存放，堆（HEAP）不能使用。

当用于连接复盖时，/X：nnnn 开关通知连接程序，在主控程序的数据区末尾和本复盖的静态数据存放首址之间保留 nnnn 个字节。例如，一个主控程序的两个复盖所需的数据区共需 500 个字节，第 1 个复盖使用 350 个字节，则可使用/X：0000 加以连接（也可以不用）。而第二个复盖可用/X：015E 加以连接，主控程序用/X：01F4 连接。

用户使用本开关可以获得一些额外的空间，以便当数据区的大小变化时，不需要全部重新连接。为得到一个复盖的数据空间，可将该复盖连接一次，连接程序结束时会显示数据区所占空间量。

3.2.4 连接主控程序

为连接主控程序，可键入命令行：

LINKMT ROOT,〈other root modules〉...,PASLIB/S/Vn：mmmm/D：oooo/X：pppp

（注意，使用/V 开关自动打开/E 和/W 开关，因而建立一个.SYM 文件）

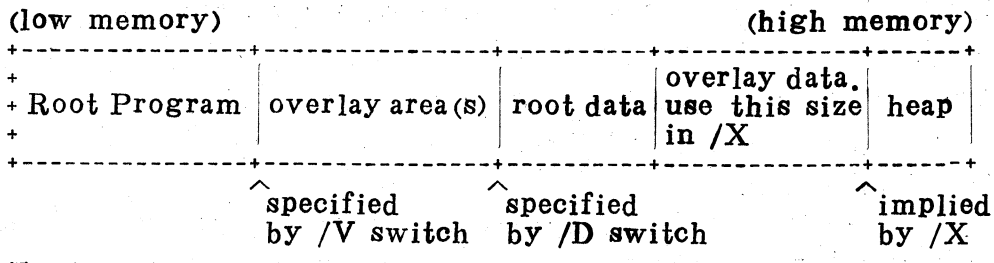
n——一个十六进制数（1,2,3…E,F,10）。

mmmm——0..FFFF 之间的一个十六进制数。

oooo——0..FFFF 之间的一个十六进制数。

pppp——0..FFFF 之间的一个十六进制数。

执行该命令，将建立 ROOT.COM 和 ROOT.SYM 文件。ROOT.SYM 用于连接随后的复盖。在复盖环境下连接主控程序必须使用/D 开关，复盖环境下的内存按排由下图所示：



如果用户要进行 HEAP 操作（即 NEW,DISPOSE 操作），复盖区必须位于主控程序数据区的左方（上图），/D 开关的参数必须包括主控程序目的代码空间和所有复盖区所占的空间。在程序研制过程中，用户可保留一些额外的空间以避免经常改变复盖区的大小。用户要牢记，系统装载复盖组进入内存是以 128 字节（1 扇面）为单位的，这意味着连接程序所显示的目标代码空间并非该复盖模块组的目标代码实际所占的空间。

/X 开关在连接主控程序时指定了复盖模块组所占用的静态空间，因而也定义了 HEAP 的下限，当程序有 HEAP 操作时，就使用到这个下限。如果没有使用/X 开关，连接程序自动地将复盖模块组的数据放在主控程序的数据区之后。

3.2.5 连接一个复盖模块组

连接一个复盖模块组涉及到下列步骤：读入构成该复盖模块组的各程序模块，并在 PASLIB 中扫描。必须使用/P：nnnn 开关以通知连接程序从何处装载复盖模块组，在同一复盖区中的诸复盖模块组所用的/P 开关参数 nnnn 应是相同的。除非在连接主控程序时只指定了一个复盖区，否则第 1 号和第 2 号复盖区中诸复盖模块组所用的/P 开关参数应不相同。详见下例。

/O：n 开关必须用来命名.SYM 文件，并将该复盖模块组和一个序号联系起来。

如果复盖有静态数据，则必须使用/X：nnnn 开关通知连接程序，该复盖的静态数据区始于距主控程序数据区末端 nnnn 字节处。如果不使用/X 开关，则各复盖的静态数据区都始于 SYSTEM＋2 处，从而导致所有的静态数据区都紧接着主控程序数据区。

应注意的是：任何已被装入主控程序中的运行时库程序，都不在复盖文件中重复出现，对于那些通过链接（chaining）方式构造的程序来说，这样可节约相当多的磁盘空间。

LINKMT ROOT=ROOT/O:2,MOD_1,MOD_2,...,PASLIB/S/P:4000

（注意，当在.SYM文件中找到该复盖时，其数据区自动地被置为SYSTEM+2的值。）

本命令建立ROOT.002文件，其中含有MOD_1.ERL,MOD_2.ERL和其他以'...'表示的所有模块中的过程和函数。如果命令行中没有'ROOT='，则该复盖模块组将被命名为MOD_1.002，复盖管理程序将找不到这个入口。所有的入口（包括过程和函数）都可由调用程序使用。注意，在编译时可使用$E开关使某一个过程或函数不进入复盖名称表，好像取消了入口记录一样。

主控程序（程序的常驻内存部分）和复盖模块组（程序的驻留磁盘部分）是分别进行连接操作的。和其他系统(尤其是数字研究公司的带PL/I-80语言的系统)不同，PASCAL/MT+系统允许用户修改复盖模块组并只需重新连接该复盖模块组。如果修改主控程序，则所有的模块（包括主控程序和复盖组）都必须重新连接。由于主控程序和复盖组之间是用符号来连接，所以仅仅在主控程序作出修改后，才有必要完全重新连接。

3.2.6 复盖文件名

复盖的基本文件名应和主控程序相同，扩展名说明复盖组号（例如，主控程序称为XYZ.COM，则第1号复盖就将称为XYZ.001）。扩展名总是以0开头，并尾随以十六进制的复盖组号（例如，第15号复盖组应在文件？？？？？？？？.00F中）。

3.2.7 连接一个带有复盖的主控程序

1）首先连接主控程序

在进行这一操作时，用户并不知道复盖组静态数据区的大小，即/X开关的参数值或/V、/D开关的参数值。本操作的目的是产生一个.SYM文件以便以后连接复盖组时使用，并获得主控程序在和所需的运行时库程序连接后的目标代码和数据区的空间。为了产生一个.SYM文件，可对/V开关设置一个伪参数或使用/E /W开关，记录下所需目标代码和数据空间，供后面步骤参考。

2）连接第一个复盖组

采用/O开关使用在1)中所产生的.SYM文件。复盖组号由用户确定，如果连接的是第1号复盖区的第1号复盖组，由于距主控程序数据区的偏差为0，所以可以不使用/X开关。有必要保留若干字节以便在程序研制过程中为主控程序数据区的变化提供空间。这个复盖区里所有的复盖组的装载起始地址是相同的，用作为/P开关的参数，这个地址至少应是主控程序目标代码空间+100H（因为主控程序从100H开始装载）+若干额外空间（供研制过程中修改程序用），记录下所需的目标代码和数据空间供后面参考。

3）连接第二个复盖组

采用/D开关使用在1)中产生的.SYM文件。复盖组序号由用户确定，但必须和2)中的序号不同；将在2)中所得到的数据区空间加上若干额外空间，以此作为/X开关的参数；/P开关的参数应是与本复盖组相联系的复盖区首址，它可以和2)中/P开关参数相同，也可以不同。记录下本复盖组所需的目标代码和数据区空间。

4）连接其余的复盖组

在这一步中，将以前连接的所有复盖组所需的数据区空间全部加起来，作为/X开关的参数；连接某一复盖区中的所有复盖组时都采用相同的/P开关参数；然后，将主控程序的目标代码空间+以前连接的所有复盖区的空间+一些额外空间，从而连接下一个复盖区中

的复盖组。基于程序研制的考虑，建议保留若干额外空间，而且应填满最后一个扇面，因为复盖管理程序是以 128 字节为单位进行装载的。

5）重新连接主控程序

至此，已经获得了最终连接主控程序所需的全部数据。将具有最高地址的复盖区加上该复盖区内最大的复盖组的目标代码空间，作为/D 开关的参数；同样，增加若干额外的空间以填满最后一个扇面。/V 开关的参数依赖于有多少个不同的复盖区以及这些复盖区是否起始于不同的地址。每一个不同的/P 开关的值都对应于一个主控程序中的/Vn，谨记 Vn 中的 n 是复盖区序号。此时，/X 开关参数应是所有复盖区中所有的复盖数据区的总和。

6）重新连接复盖组

每当.SYM 文件变化时，所有的复盖组都必须被再次连接。在第 1）步中/D 开关的参数可能是不正确的，5）中很可能对/D 开关的参数进行了修改，从而.SYM 文件也发生了变化。

3.3 复盖操作说明

3.3.1 宣称一个复盖过程

PASCAL/MT+ 编译程序接收外部宣称的扩展形式，如：

EXTERNAL { [<id> | <intconst>] } PROCEDURE/FUNCTION...

括号 { } 表示在宣称外部过程或函数时，是否采用复盖形式是可选的。对不采用复盖形式的程序来说也可套用上述宣称格式。id，intconst 必须是整型常数或整型文字常数，它向复盖管理程序说明了一个复盖组序号。

3.3.2 调用一个复盖过程

当一个复盖中的过程或函数被请求调用时，编译程序将调用复盖管理程序（@OVL），并将复盖组号和该过程或函数的字符串名称传递给@OVL。复盖管理程序将检查被调用的过程或函数是否已驻内存。在程序运行到调用发生时，仅仅在被调用的过程或函数不在所需的复盖区中时，才将调用的过程或函数从磁盘读入内存。然后，复盖管理程序在复盖入口表中检查被调用的过程或函数名，并通过入口表的地址进行调用。调用结束时，将返回请求调用的程序。

如果调用和被调用的程序在同一个复盖区内，应十分小心以确保在调用结束后将调用程序重新装载入该复盖区。调用程序的数据不能由被调用的程序所改变。

用户可调用一特殊的子程序（@OVL）来指定在哪一个驱动器上寻找复盖组。任何时候都可调用@OVS，将对应的复盖序号和盘号作为参数来传递。通常在调用位于另一磁盘上的复盖模块中的过程或函数以前调用@OVS，可将@OVS 宣称作为外部过程以备调用。

```
EXTERNAL PROCEDURE @OVS (overlay-number : INTEGER;drive : CHAR);
```

drive 以字符形式说明（只允许大写），其范围在'@' ... 'O' 之间，字符'@' 表示现行盘，'A' ... 'O' 都是允许的磁盘驱动器号，用户应保证所访问的磁盘驱动器已处于脱机状态。

3.3.3 复盖组之间的相互调用

如果复盖管理程序以补缺的非重新装载方式工作，则它从被调用的复盖程序返回时，并不将以前的复盖组重新装载进入该复盖区，因而不可能在同一复盖区中的两个复盖组中进行调用。由于非重新装载方式可以消除不必要的磁盘存取工作，因而大大提高了运行速度。复盖管理程序（overlay manager）的源文件中有一等式：'RELOAD' 可以置它为 TRUE 以便使复盖管理程序以重新装载方式工作，即从被调用的复盖程序返回时，将以前的复盖组重新

装入。配给盘上有一名为 ROVLMGR.ERL 的汇编文件，其中 RGLOAD 开关已被置为 TRUE，将此汇编文件连同运行时库程序 PASLIB 加以连接，即可使复盖管理程序以重新装载方式工作。

如果下面的二个条件得到满足，则复盖组之间可以相互调用。

1）如果请求调用的复盖程序使用了任何静态变量，而在调用过程中，一个或多个过程使用了复盖静态变量，就必须使用/X 开关来连接这些复盖组，以确保在调用过程中，程序的数据不会被破坏。

2）如果请求调用和被调用的复盖组在同一复盖区，则必须以重新装载方式工作（即置 RELOAD 为 TRUE）以便在调用结束时将请求调用的复盖组重新装载；如果被调用的复盖组位于另一个复盖区，RELOAD 可以不必为真，因为请求调用的复盖组仍然驻留在自己的复盖区中。

PASCAL/MT+ 程序都是纯码形式，而用户用汇编语言所写的程序模块则不是。用户应当知道，被调用的复盖组如果已在复盖区中，该复盖组可不必重新装入。因此每一次复盖组被调用时，汇编程序中'DB'所保留的变量区域并不被初始化，其中的内容已经被改变了。

3.3.4 存取主控程序变量及调用主控程序的过程

主控程序中的过程、函数、变量、运行时库子程序可用简单的外部宣称方式被复盖组的程序所存取或调用。

用户应注意，主控程序使用的运行时库子程序不能在复盖组中重复出现，仅由各个复盖组使用的运行时库子程序作为复盖文件存放在磁盘上。用户可将主控程序不使用的某些运行时库子程序（例如文件 I/O）连入主控程序，供复盖组使用，这样可节约空间。可将这些运行时库子程序宣称为外部子程序，然后使用 ADDR 函数，以便在浮动文件中产生一个调用。这样做是因为只需将一个运行时库子程序宣称为外部子程序而并不将它从 PASLIB 中装入内存。将 ADDR 函数的返回结束赋到一个假的整型变量中（甚至不必实际执行这个运行时库子程序，而让编译程序产生一个调用）。见下例：

```
EXTERNAL PROCEDURE @RIN;
PROCEDURE X;
VAR  P : ^INTEGER;
BEGIN
P := ADDR(@RIN); .... { causes @RIN to be loaded }
```

3.4 错误信息

复盖管理程序遇到两种错误时将向屏幕显示信息。一是它不能在编译程序要它装载的复盖组中找到所需的过程或函数，这种错误可能是由不正确的 EXTERNAL 宣称语句被引用，或是复盖组的序号错误。

第二种错误是复盖管理程序找不到所需的复盖组，这意味着该复盖组不在当前磁盘上。如果必要，可在程序中插入@OVS 调用以通知复盖管理程序在指定盘上寻找所需的复盖组。

3.5 实例

现给出一个简单的例子，假定主控程序的功能是要求用户从键盘输入字符，并根据该字符调用相应的小程序。这是一个以命令菜单驱动的商业软件包的简化例子。我们假定在实际操作中，要调用的子程序是相当大的，以致内存没有足够空间来存放两个子程序。因此，使

用复盖技术是再恰当不过了。配给盘上提供了这些实例程序的编译（一个主控程序和两个复盖组）。我们建议用户将这些程序编译和连接一遍（如下所示）以对全部的复盖操作有一感性认识。

First the main program:

```
PROGRAM DEMO_PROG;
VAR
  I  :INTEGER:(*TO BE ACCESSED BY THE OVERLAYS*)
  CH:CHAR:
EXTERNAL [1] PROCEDURE OVL1; (*COULD HAVE HAD
                                PARAMETERS*)
EXTERNAL [2] PROCEDURE OVL2; (*ALSO COULD HAVE HAD
                                PARAMETERS*)
(*EITHER COULD ALSO HAVE BEEN A FUNCTION IF DESIRED*)
BEGIN
  REPEAT
    WRITE ('Enter character, A/B/Q:');
    READ (CH);
    CASE CH OF
      'A','a':BEGIN
                I:=1; (*TO DEMONSTRATE ACCESS OF
                        GLOBALS*)
                OVL1  (*FROM AN OVERLAY*)
              END:
      'B','b':BEGIN
                I:=2;
                OVL2
              END
    ELSE
      IF NOT (CH IN ['Q','q']) THEN
        WRITELN ('Enter only A or B')
    END (*CASE*)
  UNTIL CH IN ['Q','q'];
  WRITELN ('End of program')
END.
```

overlay*1:

```
MODULE OVERLAY1;
```

```
VAR
  I :EXTERNAL INTEGER; (*LOCATED IN THE ROOT*)
PROCEDURE OVL1; (*ONE OF POSSIBLY MANY PROCEDURES
                  IN THIS MODULE*)
BEGIN
  WRITELN ('In overlay 1, I=',I)
END;
MODEND.
```

Overlay *2：

```
MODULE OVERLAY2;
VAR
  I :EXTERNAL INTEGER; (*LOCATED IN THE ROOT*)
PROCEDURE OVL2; (*ONE OF POSSIBLY MANY PROCEDURES
                  IN THIS MODULE*)
BEGIN
  WRITELN ('In overlay 2, I=', I)
END;
MODEND.
```

对这三个模块编译以后（PROG, MOD1, MOD2），必须将它们连接起来。

为连接主控程序，键入命令行：

```
LINKMT PROG, PASLIB/S/D:8000/VI:4000/X:40
```

- 主控程序数据区始于8000H（此地址是任意的）
- 复盖区（1至16号复盖组）始于4000H（此地址也是任意的）
- 保留64个（40H）字节作为额外空间

本命令建立PROG.COM和PROG.SYM文件。

现在连接1*复盖组，键入命令行：

```
LINKMT PROG=PROG/O:1, MOD1, PASLIB/S/P:4000/L
```

- /O:1意味着读入PROG.SYM，并说明当前连接的是1*复盖组
- 4000是本复盖组所属的复盖区首址
- 在PASLIB中扫描，将复盖组中使用的库子程序（未在主控程序中连入的）装载进内存（参考/L开关的功能）

本命令建立PROG.001文件。

最后连接2*复盖组：

```
LINKMT PROG=PROG/O:2, MOD2, PASLIB/S/P:4000/L
```

（说明同上，只是当前连接的是2*复盖组）。

因为没有使用局部数据，所以在连接复盖组时没有使用/X开关。在连接主控程序时用了/X开关，这样，为复盖组在将来加入局部数据提供了空间。因为复盖组之间不相互调用，RELOAD不必置为TRUE。

当 OVL_1 被调用时，PROC.001 被调入起始为 $4000 的内存空间中，然后扫描入口符号表；进而调用 OVL_1，一旦 OVL_1 运行结束，控制返回到主控程序，调用 OVL_2 的步骤相同。

现在，可以运行程序了。注意如果你不止一次地连续输入同一个字符（例如 A，A，A），则复盖组并不每次都装入内存，但当你交替输入两个不同字符时（例如 A，B，A），相应的复盖组将被装入内存以供调用。

第四节　内部建立的过程和函数

本节提供了 PASCAL/MT+ 的内部过程和函数的描述和应用实例。对每个过程和函数都加以功能描述及参数描述，并给出一个使用该过程或函数的实例程序。第三章 4.2.9 节中提供了所有内部过程和函数的速查表。

4.1　MOVE，MOVERIGHT，MOVELEFT

```
PROCEDURE MOVE (SOURCE, DESTINATION, NUM_BYTES)
PROCEDURE MOVERIGHT (SOURCE, DESTINATION, NUM_BYTES)
PROCEDURE MOVELEFT (SOURCE, DESTINANATION,
  NUM_BYTES)
```

上述三个过程从以 SOURCE 为起始地址的源地址，连续移动 NUM_BYTES 个字节到以 DESTINATION 为起始地址的目的区域中。MOVELEFT 将 NUM_BYTES 个字节从源区域的左端顺序移动到目的区域的左端。MOVE 具有和 MOVELEFT 完全相同的功能。而 MOVERIGHT 则是将 NUM_BYTES 个字节从源区域的右端依次移动到目的区域的右端（传递给 MOVERIGHT 的参数仍然是说明源区域和目的区域的左端起始地址的）。

SOURCE 和 DESTINATION 可以是任意类型的变量，而且两者的类型不必相同。它们可以是指针变量，也可以是用作指针的整型数，但不能是赋名常数或文字常数。NUM_BYTE 是一个 0～64k 之间的整数表达式。

使用这些过程要注意下列问题：

1）由于不检查字节数目（NUM_BYTES）是否大于目的区域的空间，因此如果目的区域不能容纳所要移动的字节数，则会溢出到与目的区域相邻的区域中。

2）NUM_BYTES 若为 0，不移动任何数据。

3）不进行类型检查，用户应对所使用的参数类型负责。

MOVELEFT 和 MOVERIGHT 可用于在不同的数据结构中移动数据，也可用于在相同的数据结构中移动数据。由于移动是以字节为单位进行的，所以可不考虑数据结构的类型。MOVERIGHT 适用于从一个数组的低端向其高端移动数据，如果不使用这个语句，就需要一个 FOR 循环语句将每个字符从低地址放到高地址中。MOVERIGHT 的执行速度很快，若在插入字符子程序中需要为缓冲区的字符开辟空间，则使用 MOVERIGHT 是很理想的。

MOVELEFT 适用于从一个数组向另一个数组移动字节，从缓冲区中擦除字符，或者将一个数据结构中的值移动到另一个数据结构中。例：

```
PROCEDURE MOVE_DEMO;
CONST
```

```
  STRINGSZ = 80;
VAR
  BUFFER : STRING[STRINGSZ];
  LINE : STRING;
PROCEDURE INSRT (VAR DEST : STRING; INDEX : INTEGER;
  VAR SOURCE : STRING);
BEGIN
  IF LENGTH (SOURCE) <= STRINGSZ - LENGTH (DEST) THEN
    BEGIN
      MOVERIGHT (DEST[INDEX], DEST[INDEX+LENGTH
        (SOURCE)], LENGTH (DEST) - INDEX + 1);
      MOVELEFT (SOURCE[1], DEST[INDEX], LENGTH(SOURCE));
      DEST[0] := CHR(ORD(DEST[0]) + LENGTH(SOURCE))
    END;
END;
BEGIN
  WRITELN ('MOVE_DEMO.....');
  BUFFER := 'Judy J. Smith/ 335 Drive/ Lovely, Ca. 95666';
  WRITELN (BUFFER);
  LINE := 'Roland';
  INSRT (BUFFER, POS ('5', BUFFER) + 2, LINE);
  WRITELN (BUFFER);
END;
THE OUTPUT FROM THIS PROCEDURE:
MOVE_DEMO.....
Judy J. Smith/ 355 Drive/ Lovely, Ca. 95666
Judy J. Smith/ 355 Roland Drive/ Lovely, Ca. 95666
```

4.2 EXIT

过程 **EXIT** 从当前正在运行的过程、函数或主程序中退出，如果在 **INTERRUPT** 子程序中使用 **EXIT**，则退出前将恢复寄存器的内容并触发中断使能开关。 **EXIT** 的功能与 **FORTRAN** 和 **BASIC** 语言中的 **RETURN** 语句相同。通常是在一判断语句后使用 **EXIT**。例：

```
PROCEDURE EXITTEST;
  EXIT THE CURRENT FUNCTION OR MAIN PROGRAM.
  PROCEDURE EXITPROC (BOOL : BOOLEAN);
  BEGIN
    IF BOOL THEN
      BEGIN
        WRITELN ('EXITING EXITPROC');
        EXIT;
```

```
    END;
   WRITELN('STILL IN EXITPROC, ABOUT TO LEAVE
    NORMALLY');
  END;
BEGIN
  WRITELN('EXITTEST.....');
  EXITPROC(TRUE);
  WRITELN('IN EXITTEST AFTER 1ST CALL TO EXITPROC');
  EXITPROC(FALSE);
  WRITELN('IN EXITTEST AFTER 2ND CALL TO EXITPROC');
  EXIT:
  WRITELN('THIS LINE WILL NEVER BE PRINTER');
END;
Output:
EXITTEST.....
EXITING EXITPROC
IN EXITTEST AFTER 1ST CALL TO EXITPROC
STILL IN EXITPROC, ABOUT TO LEAVE NORMALLY
IN EXITTEST AFTER 2ND CALL TO EXITPROC
```

4.3 @CMD

FUNCTION @CMD : POINTER TO STRING;

@CMD 允许用户存取命令行的参数部分（即除去命令名以外的内容），**@CMD** 函数从命令行缓冲区（**CP/M-80** 从 **80H** 单元起）取出信息送入指定字串中，并返回一个指针指向该字串。在 **CP/M-80** 中，命令行的参数部分是以一定格开始的，本函数的设计为那些使用各种方法来实现这种参数传递的操作系统提供方便。

必须在程序的一开始，任何文件被打开之前调用**@CMD**，且只能调用一次。例：

```
PROGRAM@CMD_DEMO;
TYPE
  PSTRG = ^STRING;
VAR S : STRING[16];
  PTR : PSTRG;
  F   : FILE OF INTEGER;
EXTERNAL FUNCTION @CMD : PSTRG;
BEGIN
  PTR := @CMD;
  S := PTR^;
  ASSIGN(F,S);
  RESET(F);
END.
```

4.4 TSTBIT, SETBIT, CLRBIT

```
FUNCTION TSTBIT   (      BASIC_VAR,BIT_NUM) : BOOLEAN;
PROCEDURE SETBIT (VAR BASIC_VAR,BIT_NUM);
PROCEDURE CLRBIT (VAR BASIC_VAR,BIT_NUM);
```

BASIC_VAR 是任何 8 位或 16 位变量，如整字符、字节字或布尔型。BIT_NUM 是 0到15 间 的整数，变量的最右一 位 应为0位。如果 BIT-NUM 超过 了范围，则结果不可预测，但程序照常运行。如果对一个 8 位变量企图进行第 10 位操作，虽不会出错，但没有结果。

如果 BASIC_VAR 中的指定位值 为 1，TSTBIT 返 回 TRUE，反 之 则 返 回 FALSE。SETBIT 将 BASIC-VAR 变量中的指定位置成 1，CLRBIT 将 BASIC-VAR 中的指定位置成0。

这些过程可用来产生一个循环等待程序段；或当需要时，通过改变某个位的状态来改变输入的数据；也可用于操作位映射屏幕。例：

```
PROCEDURE TST_SET_CLR_BITS;
VAR
  I : INTEGER;
BEGIN
  WRITELN ('TST_SET_CLR_BITS.....');
  I := 0;
  SETBIT (I,5);
  IF I = 32 THEN
    IF TSTBIT (I,5) THEN
      WRITELN ('I=',I);
  CLRBIT (I,5);
  IF I = 0 THEN
    IF NOT (TSTBIT (I,5)) THEN
      WRITELN ('I=',I);
  end;
Output:
TST_SET_CLR_BITS.....
I=32
I=0
```

4.5 SHR, SHL

```
FUNCTION SHR (BASIC-VAR,NUM) : INTEGER;
FUNCTION SHL (BASIC-VAR,NUM) : INTEGER;
```

BASIC-VAR 是一 8 位或 16 位变量，NUM 是一整数表达式，SHR 将 BASIC_VAR 向右移 NUM 位，并在左端补 0；SHL 将 BASIC-VAR 向左移 NUM 位，并在右端补 0。

SHR 和 SHL 的用途很清楚，假定一个 10bit 的数（二进制）可以从两个彼此独立 的 输入通道上获得，可使用下列语句将它读入：

```
X := SHL (INP[8] & $1F,3) ! (INP[9] & $1F) ;
```

在上面的语句中，从 8#通道读入一 8bit数据，取其低五位有效，并向左移 三 位，将 此

结果和9*通道上的8bit数据的低五位进行逻辑“或”操作。

下面的过程显示了调用这两个函数所期望得到的结果。例：

```
PROCEDURE SHIFT_DEMO;
VAR I : INTEGER;
BEGIN
  WRITELN ('SHIFT_DEMO.....');
  I := 4;
  WRITELN ('I=', I);
  WRITELN ('SHR (I, 2) =', SHR (I, 2));
  WRITELN ('SHL (I, 4) =', SHL (I, 4));
end;
```

Output：

```
SHIFT_DEMO.....
I=4
SHR (I, 2) =1
SHL (I, 4) =64
```

4.6 HI, LO, SWAP

```
FUNCTION HI (BASIC_VAR) : INTEGER
FUNCTION LO (BASIC_VAR) : INTEGER
FUNCTION SWAP (BASIC_VAR) : INTEGER
```

HI 函数返回 BASIC-VAR (8 位或 16 位变量)的高八 位 放在返回结果 的 低 八位 中(对16 变量而言)。

LO 函数返回 BASIC_VAR 的低八位，返回结果的高八位被置成0。(对 16 位变量而言) 。

SWAP 的返回结果是将 BASIC_VAR 的高八位 和低 八位对换(对 16 位变量而言) 。如果 BASIC_VAR 是一八位变量，则 HI 返回结果为0，LO 返回原来的变量值。

下面的例子表明如何使用这三个函数。例：

```
PROCEDURE HI_LO_SWAP;
VAR
  HL : INTEGER;
BEGIN
  WRITELN ('HI_LO_SWAP.....');
  HL := $ 104;
  WRITELN ('HL=', HL);
  IE HI (HL) = 1 THEN
    WRITELN ('HI (HL) =', HI (HL));
  IF LO (HL) = 4 THEN
    WRITELN ('LO (HL) =', LO (HL));
  IF SWAP (HL) = $ 0401 THEN
    WRITELN ('SWAP (HL) =', SWAP (HL));
```

```
END;
Output:
HI_LO_SWAP.....
HL=260
HI (HL)=1
LO (HL)=4
SWAP (HL)=1025
```

4.7 ADDR

FUNCTION ADDR (VARIABLE REFERENCE):INTEGER

ADDR 返回 **VARIABLE REFERENCE** 的地址，**VARIABLE REFERENCE** 可以是过程/函数名、下标变量和记录的某个段名（**fields**），也可以是外部变量（包括复盖组中的变量）。**VARIABLE REFERENCE** 必须是可见的，例如：从主控程序这一层上，一个位于第3层嵌套的子程序是不可见的，因此不可能得到这个子程序的地址。引用的变量 **VARIABLE REFERENCE** 不包括赋名常数，用户定义类型或其他不占目标代码空间或数据空间的变量。

这个函数具有下列用途：得到由 **INLINE** 所产生的编译时表的首地址；得到一个在 **MOVE** 语句中使用的数据结构的地址；等等。当递归开关打开时，**ADDR** 不能获得由 **INLINE** 产生的常数表的首址，这是因为子程序目标代码前保留了六个字节(若 $Q 开关打开，为5个字节)。例：

```
PROCEDURE ADDR_DEMO (PARAM : INTEGER);
VAR
  REC : RECORD
          J : INTEGER;
          BOOL : BOOLEAN;
        END;
  ADDRESS : INTEGER;
  R : REAL;
  S1 : ARRAY [1..10] OF CHAR;
BEGIN
  WRITELN ('ADDR_DEMO.....');
  WRITELN ('ADDR (ADDR_DEMO) =', ADDR_DEMO));
  WRITELN ('ADDR (PARAM) =', ADDR (PARAM));
  WRITELN ('ADDR (REC) =', ADDR (REC));
  WRITELN ('ADDR (REC.J)', ADDR (REC.J));
  WRITELN ('ADDR (ADDRESS) =', ADDR(ADDRESS));
  WRITELN ('ADDR (R) =', ADDR (R));
  WRITELN ('ADDR (S1) =', ADDR (S1));
end;
Output is system dependent
```

4.8 SIZEOF

FUNCTION SIZEOF (VARIABLE OR TYPE NAME) : INTEGER

参数可是任意变量，字符、数组、记录或其他用户定义的类型均可。SIZEOF 是一个编译时完成的函数调用。因此只有那些不生成目标代码就可计算其空间的变量才能作为SIZEOF的参数。SIZEOF 返回该变量所占空间的字节数，它可用在 MOVE 语句中来说明要移动的字节数。使用了 SIZEOF，程序员常常不必在研制程序的过程中不断改变常数。例：

```
PROCEDURE SIZE_DEMO;
VAR
  B : ARRAY〔1..10〕OF CHAR;
  A : ARRAY〔1..15〕OF CHAR;
BEGIN
  WRITELN ('SIZE_DEMO.......');
  A := '********';
  B := '0123456789';
  WRITELN('SIZEOF(A)=',SIZEOF(A);SIZEOF(B)=',SIZEOF(B));
  MOVE (B, A, SIZEOF (B));
  WRITELN ('A=', A);
end;
Output:
SIZEOF (A) = 15 SIZEOF (B) = 10
0123456789****
```

4.9 FILLCHAR

PROCEDURE FILLCHAR(DESTINATION, LENGTH, CHARACTER);

DESTINATION 是一个压缩字符数组，可以进行下标寻址。LENGTH 是一整数表达式，如果 LENGTH 大于 DESTINATION 的长度，则与之相邻的目标代码或数据将被复盖；而且，如果 LENGTH 为负，相邻的内存也被复盖。CHARACTER 可以是字符型变量，也可以是字符常量，本过程将 DESTINATION 中 LENGTH 个字符用 CHARACTER 来替换。

本过程向用户提供一种快速方法，以便向一个数据结构内填入相同的数据。例如，可使用本过程来初始化缓冲区。例：

```
PROCEDURE FILL_DEMO;
VAR
  BUFFER : PACKED ARRAY〔1..256〕OF CHAR;
BEGIN
  FILLCHAR (BUFFER, 256, '');          {BLANK THE BUFFER}
END;
```

4.10 LENGTH

FUNCTION LENGTH (STRING) : INTEGER;

返回字串长度的整数值。例：

```
PROCEDURE LENGTH_DEMO;
```

```
VAR
  S1 : STRING[40];
BEGIN
  S1 := 'This string is 33 characters long';
  WRITELN ('LENGTH OF',S1,'=',LENGTH (S1));
  WRITELN ('LENGTH OF EMPTY STRING = ',LENGTH (''));
end;
Output:
LENGTH OF This string is 33 characters long=33
LENGTH OF EMPTY STRING = 0
```

4.11 CONCAT

FUNCTION CONCAT (SOURCE1,SOURCE2,…,SOURCEn) : STRING;

返回一字串，此字串是参数表中所有源字串的联接。参数表中的参数可以是字串型变量、字串文字常量或字符，长度为零的源字串也可联接。如果所有源字串的总长度超过 255 个字节，则只截取前面的 255 个字节。有关使用 CONCAT 和 COPY 的限制参见下节 COPY 的注解。例：

```
PROCEDURE CONCAT_DEMO;
VAR
  S1,S2 : STRING;
BEGIN
  S1 := 'left iink, right link';
  S2 := 'root root root';
  WRITELN (S1,'/',S2);
  S1 := CONCAT (S1,' ',S2,'!!!!!!');
  WRITELN (S1);
end;
Output;
left link, right link/root root root
left link, right link root root root!!!!!!
```

4.12 COPY

FUNCTION COPY (SOURCE,LOCATION,NUM_BYTES) : STRING;

SOURCE 必须是字串型，LOCATION 与 NUM_BYTES 是整数型表达式。此函数返回一字串，它含有从 SOURCE 中 LOCATION 指定的位置起的 NUM_BYTES 个字符。如果 LOCATION 超出范围或是一负数，不会出现错误。如 NUM_BYTES 是负数或 NUM_BYTES 加 LOCATION 超出 SOURCE 字串的长度，则多余字节被截去。例：

```
PROCEDURE COPY_DEMO;
BEGIN
  LONG STR := 'Hi from Cardiff-by-the-sea";
  WRITELN (COPY (LONG_STR,9,LENGTH (LONG_STR)-9+1));
```

```
end;
Output:
Cardiff-by-the-sea
```

注：COPY 与 CONCAT 是“伪”字串返回函数，且只有一个静态缓冲区分配给返回值。因此，如果这些函数在同一表达式中被多次使用，那么这些函数每次返回的值是它最后一次所返回的值。例如：'IF((CONCAT(A,STRING$_1$)=CONCAT(A,STRING$_2$))'总是为真，因为 A 与 STRING$_1$ 联接的值被 A 与 STRING$_2$ 联接的值取代了。再如：WRITELN (COPY (STRING$_1$,1,4), COPY(STRING$_1$,5,4))，将两次写出同一字串，都是从第二组的 STRING$_1$ 中取出的 4 个字符（即第 5～8 个字符）。

4.13 POS

```
FUNCTION POS(PATTERN,SOURCE):INTEGER;
```

返回 PATTERN 在 SOURCE 中第一次出现的位置的整数值。如果 PATTERN 在 SOURCE 中没找到，则返回零值。SOURCE 是字串型，PATTERN 可以是字串型、字符型或文字常数。例：

```
PROCEDURE POS_DEMO;
VAR
  STR,PATTERN : STRING;
  CH : CHAR;
BEGIN
  STR := 'MT MicroSYSTEMS';
  PATTERN := 'croSY';
  CH := 'T';
  WRITELN ('pos of',' PATTERN,' in '.STR,' is', POS (PATTERN,
           STR));
  WRITELN ('pos of 'CH,' in ',STR,' is ', POS (CH,STR));
  WRITELN ('pos of' 'z' in', STR,' is', POS ('z', STR));
end;
Output:
pos of croSY in MT MicroSYSTEMS is 6
pos of T in MT MicroSYSTEMS is 2
pos of 'z' in MT MicroSYSTEMS is 0
```

4.14 DELETE

```
PROCEDURE DELETE (TARGET,INDEX,SIZE);
```

TARGET 是一字串，INDEX 与 SIZE 是整型表达式。此过程从 TARGET 中 INDEX 字节起删去 SIZE 个字符。如 SIZE 为零，则无任何操作。若 SIZE 为负，则严重出错。如 INDEX 加 SIZE 大于 TARGET 或 TARGET 为空串，则与数据邻近的内存空间可能被破坏。例：

```
PROCEDURE DELETE_DEMO;
VAR
  LONG_STR : STRING;
```

```
BEGIN
  LONG_STR : = '        get rid of the ieading bianks';
  WRITELN (LONG_STR);
  DELETE (LONG_STR, 1, POS ('g', LONG_STR) - 1) - ;
  WRITELN (LONG_STR);
END;
Output:
  get rid of the leading blanks
get rid of the leading blanks
```

4.15 INSERT

PROCEDURE INSERT (SOURCE, DESTINATION, INDEX);

DESTINATION 是一字串。SOURCE 可以是字符或字串、文字常量或变量。INDEX 是整型表达式。此过程将 SOURCE 插入 DESTINATION 中 INDEX 指定的位置。SOURCE 可以是空串。如INDEX 超出范围，或 DESTINATION 是空串，则数据将被破坏。如果将 SOURCE 插入 DESTINATION，导致它的长度大于允许值，则发生截取。例：

```
PROCEDURE INSERT_DEMO;
VAR
  LONG_STR : STRING;
  S1 : STRING〔10〕;
BEGIN
  LONG_STR : = 'Remember Luke';
  S1 : = 'the Force,';
  INSERT (S1, LONC_STR, 10);
  WRITELN (LONG_STR);
  INSERT ('to use', LONG_STR, 10);
  WRITELN (LONG_STR);
end;
Output:
Remember the Force, Luke
Remember to use the Force, Luke
```

4.16 ASSIGN

PROCEDURE ASSIGN (FILE, NAME);

此过程用在 RESET 或 REWRITE 前。它将一外部文件名赋给文件型变量。FILE 是一文件名，NAME 是字符常量或字串变量，其中含有将要建立的文件的名字。FILE 可以是任何类型的文件，但若要使用下面的特殊设备，则必须是 TEXT 类型的文件。

用户应注意到标准PASCAL对"局部"文件的定义。PASCAL/MT⁺采用PASTMP××.$$$暂存文件名来实现这一功能。其中××是序号，在每个程序开始时为零。如果一个外部文件没有先执行 ASSIGN，而对它进行 REWRITE 操作，那么在建立这一外部文件前也会把一个暂存文件名赋给它。在递归环境下局部宣称的文件不可用作暂存文件，除非用户执行了

ASSIGN(<file>,")操作初始化此文件。

下面是CP/M环境下提供的设备名。

设备名

CON：——用于输入时，将回波输入字符，对回车符回波回车/换行符。对退格（backspace）符回波退格、空格、退格。

用于输出时，对回车符回波回车/换行。CP/M将tab键扩展至8个字符间隔的位置上。不能输出换行符。

TRM：——CP/M控制台，仅作为输入设备使用。无转换（仅适用于CP/M_2·X）。

KBD：——CP/M控制台，仅作为输入设备使用。无回波，无转换，不能与CON：同时用作输入。

LST：——CP/M打印机，仅作为输出设备使用。无转换（也不扩展tab键）。

RDR：——CP/M读卡机，仅作为输入设备使用，通过BDOS用函数3调用BIOS中的辅助输入子程序。

PUN：——CP/M穿孔机，仅作为输出设备使用，通过BDOS用函数4调用BIOS中的辅助输出子程序。

这些设备名可用于ASSIGN过程。如下所示：

```
ASSIGN (CONIN,'CON:');
ASSIGN (KEYBOARD,'KBD:');
ASSIGN (CRT,'TRM:');
ASSIGN (PRINTFILE,'LST:');
```

由于实现函数调用的方法不同，在KBD：与CON：联用时有一个问题。为了实现control-s，在执行函数调用2（即写CON：）时，CP/M要检查键入的字符。如果用户键入一个字符不是control-s，则CP/M将此字符存放起来并期待后面出现的函数调用1。函数调用6（用于KBD：）总是直接进入BIOS进行输入而忽略任何内部缓冲区中已有的字符。所以，程序可能会漏掉实际上已由CP/M存放在内部的字符。

4.17 WNB,GNB

```
FUNCTION GNB (FILEVAR:FILE OF PAOC):CHAR;
FUNCTION WNB (FILEVAR:FILE OF CHAR;CH:CHAR):BOOLEAN;
```

这二个函数允许用户以高速方法按字节对文件进行存取。PAOC是任意类型的紧缩字符数组（Packed Array Of Char）。它的大小最好在128至4095的范围内。

GNB允许用户从一文件中每次读一个字节。这是一个返回值为字符的函数。当读到文件的物理结尾时，EOF函数为'真'，而不依赖于文件中的任何数据。读到超过文件结尾时便返回$FF。详见附录中文件一节。

WNB允许用户每次写一个字节到文件。这个函数需要一个文件和一个要写的字符作参数。它返回一布尔值。在写一个字节到文件时，若出错则为'真'值。它对被写的字符不作任何转换。

使用GNB和WNB是因为它们相当快（对F↑,GET/PUT而言）。这是由于有一个较大的缓冲区的缘故。

参见附录中文件一节的示范程序。

4.18 BLOCKREAD, BLOCKWRITE

BLOCKREAD(F : FILEVAR;BUF : ANY;VAR IOR : INTEGER;SZ, RB:INTEGER);

BLOCKWRITE(F : FILEVAR;BUF : ANY;VAR IOR:INTEGER;SZ, RB : INTEGER);

这二个过程用于直接 CP/M 磁盘存取。FILEVAR 是一非类型文件 (FILE)。BUF 是足以容纳数据的任意数组变量。IOR 是一整数，接收操作系统的返回值。SZ 是要传送的字节数，RB 是相对块号。在 CP/M 环境下 SZ 必须是 128 的整数倍。SZ 与 BUF 的大小有关。如果 BUF 是 128 字节，那么 SZ 必须是 128；如果 BUF 是 4096 字节，那么 SZ 最大可为 4096。RB 可以在 -1 至 32767 之间。当 RB 为 -1 时，运行时程序包即认为是顺序块传送；当 RB 大于 -1，则运行时程序包将计算文件的正确地址，并在需要时打开新的文件入口。

这二个过程能将数据从文件读出送到 BUF 中，或从 BUF 中取出送至文件中。

4.19 OPEN

PROCEDURE OPEN (FILE, TITLE, RESULT);

过程 OPEN 增加了 PASCAL/MT+ 的灵活性，并提供了与以前版本的 PASCAL/MT+ 的兼容性。FILE 是任何文件类型的变量。TITLE 是含有 CP/M 文件名的字符串。RESULT 是一个整型参数且返回与 IORESULT 相同的值。

过程 OPEN 实际上与顺序执行 ASSIGN(FILE, TITLE); RESET (FILE); RESULT : = IORESULT 一样。例：

OPEN(INFILE,'A : FNAME.DAT', RESULT);

4.20 CLOSE

PROCEDURE CLOSE (FILE, RESULT);

PROCEDURE CLOSEDEL (FILE, RESULT);

过程 CLOSE 与 CLOSEDEL 分别用于关闭和关闭并删除一个文件。为确保用任何一种方法写入文件的数据完全以文件缓冲区送到磁盘上，必须调用 CLOSE 过程。CLOSEDEL 通常用于暂存文件，使用后将其删除。FILE 与 RESULT 与在 OPEN 中的相同 (见第三章 4.19 节)。

当打开的文件被 RESET 时，文件就被隐含地关闭了。在同一时间可打开的文件个数与 CPU 有关。对 CP/M 来说，文件个数的限度仅取决于文件控制块使用的内存空间。

CLOSE 过程的使用见在附录中文件一节。

4.21 PURGE

PROCEDURE PURGE (FILE);

过程 PURGE 用于删除一个文件，其文件名放在一字符串中。用户必须先用 ASSIGN 将文件名赋给文件，然后执行 PURGE。例：

ASSIGN (F,'B : BADFILE.BAD');

PURGE (F);　　(*DELETE B : BADFILE, BAD*)

4.22 IORESULT

FUNCTION IORESULT : INTEGER;

在每次 I/O 操作后，运行时程序包中的例行程序都要置 IORESULT 函数一个值。一般地说，IORESULT 函数值与系统有关。任何时候都不要试图写 IORESULT，因为在每次 I/O 操作前它都被预置为0。

IORESULT 函数返回从 BDOS 带来的整数值，它在每次 CP/M 文件存取后都要被改变。

在CP/M环境下，一般规则是255表示出错，其他任何值表示成功。WRITE/PUT/WNB不属此例。在这些过程中，非零的IORESULT值表示出错。

下表是CP/M下的IORESULT值：

过程	返回值
CLOSE	255表示出错，其他均表示正确。
RESET	255表示出错，其他均表示正确。
REWRITE	255表示出错，其他均表示正确。
READ/READLN/GET	<>0表示文件结束，0表示正确。
PAGE/WRITE/WRITELN/PUT	<>0表示出错，0表示正确。
PURGE	总是返回0。
SEEKREAD/SEEKWRITE	0操作成功 1 读未写的数据 2 CP/M 错误 3 不能关闭当前文件入口 4 查找未写的文件入口 5 不能创建新的文件入口（仅对写） 6 查找超出磁盘的物理结束。

例：

```
ASSIGN (F,'C : HELLO');
RESET(F);
IF IORESULT = 255 THEN
WRITELN('C : HELLO IS NOT PRESENT'))
```

4.23 SEEKREAD,SEEKWRITE

```
PROCEDURE SEEKREAD(F : ANYFILE;RECORD-NUM : 0..MAXINT);
PROCEDURE SEEKWRITE (F : ANYFILE;RECORD-NUM : 0..MAXINT);
```

随机读写只能在CP/M2.0版本或更高的版本中才能实现。

随机读写过程把窗口变量用作I/O缓冲区，用户须在SEEKWRITE前赋值给窗口变量或在SEEKREAD后从窗口变量取值。在附录文件一节可找到使用这二个过程的示范程序。

```
F^ : = data;
SEEKWRITE(F, RECORD-NUM);
SEEKREAD(F, RECORD-NUM);
data : = F^;
```

不管记录的大小如何，用SEEKWRITE写的文件是连续的。这就意味着一个文件的读写既可以是顺序的也可以是随机的，但在改变读写方法前一定要执行CLOSE。

这二个过程比BLOCKREAD和BLOCKWRITE更有用。因为它们完成同BLOCKREAD和BLOCKWRITE相同的功能，但更为灵活，更易使用。

随机读写运行时的支持可在RANDOMIO.ERL中找到，它必须被连接，使得SEEKREAD和SEEKWRITE可以执行。

IORESULT值见第三章4.22节IORESULT。

如要了解这些例行程序的工作情况，详见附录的文件部分。

4.24 READHEX, WRITEHEX

```
PROCEDURE READHEX (VAR F : TEXT;VAR W : ANYTYPE;SIZE : 1..2);
PROCEDURE WRITEHEX(VAR F:TEXT;EXPRESSION:ANYTYPE;SIZE:1..2);
```

使用这二个过程可以完成类型为 WORD 的变量的 I/O，或者写一个十六进制数字。EXPRESSION 是一字节或二字节，SIZE 表示要写的字节数。

4.25 MEMAVAIL, MAXAVAIL

```
FUNCTION MEMAVAIL : INTEGER;
FUNCTION MAXAVAIL : INTEGER;
```

函数 MEMAVAIL 与 MAXAVAIL 和 NEW 与 DISPOSE 联用可管理 PASCAL/MT+ 中的 HEAP 内存区。MEMAVAIL 函数返回在某一时刻最大的、从未分配出去的可用内存量，不进行分散内存块的收集工作。MAXAVAIL 函数首先收集无用单元，合并相邻碎片，然后返回最大可用的内存量。注意，如果在显示时这些函数值为负数的话，则表示所余内存量大得不能用一个整数表示。返回值可以用 WRITEHEX 显示。在一个与时间有关的程序段前可用 MAXAVAIL 函数强行收集无用单元。

PASCAL/MT+ 系统充分支持标准 PASCAL 定义的 NEW 及 DISPOSE，支持 NEW 及 DISPOSE 实现的运行时例行程序可在 FULLHEAP.ERL 中找到。它必须在程序连接时被真正连接上。

参见附录有关动态内存利用的详细资料。

4.26 WAIT

```
PROCEDURE WAIT (PORTNUM, MASK, POLARITY);
```

PORTNUM 与 MASK 是文字常量或赋名常量。POLARITY 是一布尔常量。

WAIT 产生一简短的检查状态的循环等待代码段：

```
IN PORTNUM
ANI MASK
J?? $-4
```

其中?? 在 POLARITY 为'假'时是 Z，若 POLARITY 为'真'时则为 NZ。

WAIT 过程不产生这一循环代码段的嵌入代码。上述检查状态的循环代码段是建立在数据区，由 WAIT 的运行时子程序所调用的。这意味着此循环虽快，但调用并从此循环返回要增加少量的执行时间。因此如时间很重要的话，应使用 INLINE 嵌入。例：

```
PROCEDURE WAIT_DEMO;
CONST
  CONSPORT = $F7; (*for EXO NOBUS-Z  COMPUTER*)
  CONSMASK = $01;
BEGIN
  WRITELN ('WAIT_DEMO.....');
  WRITELN ('WAITING FOR A CHARACTER');
  WAIT (CONSPORT, CONSMASK, TRUE);
  WRIT ELN ('THANKS!');
```

end;

4.27 RIM85, SIM85

这二个例行程序使用了专门的8085指令RIM与SIM。含有此指令的过程产生一个调用。在CP/M环境中，HEAP区是从数据区尾向上增长，堆栈区(STACK)区是自顶向下增长。在CP/M中硬件堆栈寄存器被预置成地址0006中的内容，使用$Z开关可以改变此值。堆栈区从初始值以下512个字节处开始增长。详见第二章2.4.4节关于$Z开关部分。

4.28 对内部过程和函数的快速索引

以下每组都以字母为序：

字符数组操作例行程序：

```
PROCEDURE FILLCHAR  (DESTINATION,LENGTH,CHARACTER);
PROCEDURE MOVELEFT  (SOURCE, DESTINATION, NUM_BYTES);
PROCEDURE MOVERIGHT(SOURCE, DESTINATION,NUM_BYTES);
```

位与字节操作例行程序：

```
PROCEDURE CLRBIT(BASIC_VAR, BIT_NUM);
FUNCTION   HI     (BASIC_VAR )          : INTEGER;
FUNCTION   LO     (BASIC_VAR )          : INTEGER;
PROCEDURE SETBIT(BASIC_VAR, BIT_NUM);
FUNCTION   SHL    (BASIC_VAR, NUM)      : INTEGER;
FUNCTION   SHR    (BASIC_VAR, NUM)      : INTEGER;
FUNCTION   SWAP   (BASIC_VAR )          : INTEGER;
FUNCTION   TSTBIT(BASIC_VAR, BIT_NUM) : BOOLEAN;
```

字串处理例行程序：

```
FUNCTION CONCAT (SOURCE1,SOURCE2,...,SOURCEn) : STRING;
FUNCTION COPY    (SOURCE,LOCATION,NUM_BYTES) : STRING;
PROCEDURE DELETE (TARGET,INDEX,SIZE);
PROCEDURE INSERT  (SOURCE,DESTINATION,INDEX);
FUNCTION LENGTH (STRING)                    :INTEGER;
FUNCTION POS     (PATTERN,SOURCE)           :INTEGER;
```

文件处理例行程序：

```
PROCEDURE ASSIGN       (FILE, NAME);
PROCEDURE BLOCKREAD    (FILE, BUF,IOR,NUMBYTES,
                         RELBLK);
PROCEDURE BLOCKWRITE (FILE, BUF,IOR,NUMBYTES,
                         RELBLN);
PROCEDURE CLOSE        (FILE, RESULT);
PROCEDURE CLOSEDEL     (FILE, RESULT);
FUNCTION   GNB         (FILE )        :CHAR
PROCEDURE IORESULT                    :INTEGER;
PROCEDURE OPEN         (FILE, TITLE, RESULT );
```

```
PROCEDURE OPENX        (FILE, TITLE, RESULT, EXTENT);
PROCEDURE PURGE        (FILE );
PROCEDURE READHEX      (FILE, VAR, SIZE);
PROCEDURE SEEKREAD     (FILE, RECORD_NUMBER);
PROCEDURE SEEKWRITE    (FILE, RECORD_NUMBER);
FUNCTION  WNB          (FILE, CHAR) : BOOLEAN;
PROCEDURE WRITEHEX     (FILE, EXPRESSION, SIZE);
```

其他例行程序：

```
FUNCTION  @CMD : PTR_TO_STRING;
FUNCTION  ADDR (VARIABLE REFERENCE) : INTEGER;
PROCEDURE EXIT;
FUNCTION  MAXAVAIL : INTEGER;
FUNCTION  MEMAVAIL : INTEGER;
FUNCTION  PIM85 : BYTE;
PROCEDURE SIM85 (VAL : BYTE);
FUNCTION  SIZEOF (VARIABLE OR TYPE NAME) : INTEGER;
PROCEDURE WAIT (PORTNUM, MASK, POLARITY);
```

第五节 链 接

有时程序会超过可用内存空间，有时为了编译及维护等目的，需要将程序分段。PASCAL/MT+系统提供了一种“链接”机构，用这种方法可使程序的控制权从一个程序转到另一个程序。

用户必须宣称一个非类型文件（FILE），并用 ASSIGN 和 RESET 初始化此文件。然后用户可以调用 CHAIN 过程，将文件变量名作为参数传给 CHAIN。运行时库程序会将用 RESET 语句打开的文件装入。各个程序的大小没有关系，但当从小程序链接到大程序时，数据一定是在最大一个程序的上面。这意味着小程序可以链接到大程序而大程序也可链接到小程序。若用户希望在链接的程序间进行通讯，那么可采用两种方式：共享全局变量和绝对（ABSOLUTE）变量。

使用共享全局变量方法时，用户必须保证：至少全局变量的开始部分是用于通讯的数据区，并且在要通讯的两个程序中完全相同。其余全局变量无须相同，在全局变量段宣称的外部变量不会影响地址映射。除了有相同的宣称外，用户还必须用 LINKER 的/D 开关将所有要通讯的程序中的变量放在同一地址。

使用绝对变量方法时，用户通常是宣称一个用于通讯的记录，然后在每个模块中都将此记录定义在同一绝对地址上，这种方法不需要用/D 开关，但是需要知道程序和系统占有内存的情况。

为了在链接过程中维持 HEAP 区，可以宣称一个 EXTERNAL INTEGER 的 SYSTEM. SYSTEM 含有 HEAP 顶的地址。另外，若使用了 FULLHEAP，则 @EFL：INTEGER 与 @FRL：ARRAY〔1..4〕OF BYTE 含有 FULLHEAP 需要的信息。这些都要被保存在全局变

量数据区，并在被链接程序的开头重新设置这些值。为了把参加链接的所有程序中的全局变量放在同一地址上，需要用 LINKER 的/D 开关。

下面是两个用绝对变量方式通讯的示范程序。第一个程序链接到第二个程序，后者将打印出第一个程序运行的结果。

```
(*PROGRAM*1 IN CHAIN DEMONSTRATION*)
PROGRAM CHAIN1;
TYPE
  COMMAREA = RECORD
               I,J,K : INTEGER
             END;
VAR
  GLOBALS : ABSOLUTE [S8000] COMMAREA;
  CHAINFIL : FILE;
BEGIN (*MAIN PROGRAM*1*)
  WITH GLOBALS DO
    BEGIN
      I := 3;
      J := 3;
      K := I*J
    END;
  ASSIGN (CHAINFIL,'A:CHAIN2.COM');
  RESET (CHAINFIL);
  IF IORESULT = 255 THEN
    BEGIN
      WRITELN ('UNABLE TO OPEN CHAIN2.COM');
      EXIT
    END;
  CHAIN (CHAINFIL)
END.          (*END CHAIN1*)
(*PROGRAM*2 IN CHAIN DEMONSTRATION*)
PROGRAM CHAIN2;
TYPE
  COMMAREA = RECORD
               I,J,K : INTEGER
             END;
VAR
  GLOBALS : ABSOLUTE [S8000] COMMAREA;
BEGIN (*PROGRAM*2*)
  WITH GLOBALS DO
```

```
    WRITELN ('RESULT OF',I,'TIMES',J,'IS =',K)
END.            (*RETURNS TO OPERATING SYSTEM WHEN
                  COMPLETE*)
```

第六节　中断处理程序

在 PASCAL/MT+ 中可以实现一种特殊的过程：中断过程。用户可选择联接每个中断源的中断向量。过程宣称如下：

PROCEDURE INTERRUPT [<vecnum>]<identifier>

为了正确地装入中断向量，中断过程只能放在主程序中，不可以有参数表，但可以有局部变量，也可以读写全局变量。编译在程序开始时产生代码以便装入此过程地址的向量。对 8080/Z80 系统，<vecnum>可在 0 至 7 之间。对 Z80 方式 2 中断，用户必须宣称一个 **ABSOLUTE** 变量以存放一个中断表，然后用 **ADDR** 函数填写此表。在 Z80 环境下要用 **INLINE** 方法初始化 I-寄存器。

编译产生的代码在开始执行过程时将寄存器内容推入堆栈。在过程执行后从堆栈中推出寄存器内容并重新赋能中断。Z 开关不会产生 Z80 的 "RETI" 指令，因为在 Z80 上有多种中断方式。

用户应注意到系统不产生可重起动代码。典型的中断过程要置全局变量且不执行其他过程调用或输入/输出；避免集合、字串、过程调用、文件 I/O 和 CP/M 调用，以及任何含有数据区的运行时库中的例行程序。这是因为 8080 和 Z80 没有足够复杂的寻址方式对堆栈上的数据项进行操作，使得可以从堆栈中不断地移出返回地址并存放到静态分配的暂存单元中。CP/M 用户应注意到通过 BDOS 的 I/O 通常都重新开放中断。

为了禁止在 PASCAL 代码段前后中断，用 INLINE 与小汇编把 EI（使能中断）和 DI（禁止中断）指令放在代码段前后。

下面的示范程序等待四个开关中的一个发生中断，然后翻转连在此开关上的灯的状态。灯的 I/O 口是 0..3，开关的中断用 RESTART 2, 3, 4 和 5。

```
PROGRAM INT_DEMO;
CONST
  LIGHT1 = 0;          (*DEFINE I/O PORT CONSTANTS*)
  LIGHT2 = 1;
  LIGHT3 = 2;
  LIGHT4 = 3;
  SWITCH1 = 2;         (*DEFINE INTERRUPT VECTORS*)
  SWITCH2 = 3;
  SWITCH3 = 4;
  SWITCH4 = 5;
VAR
  LIGHT STATE : ARRAY [LIGHT1..LIGHT4] OF BOOLEAN;
  SWITCH_PUSH : ARRAY [LIGHT1..LIGHT4] OF BOOLEAN;
```

```
  I : LIGHT1..LIGHT4;
PROCEDURE INTERRUPT [SWITCH1] INT1;
BEGIN
  SWITCH_PUSH [LIGHT1] : = TRUE
END;
PROCEDURE INTERRUPT [SWITCH2] INT2;
BEGIN
  SWITCH_PUSH [LIGHT2] : = TRUE
END;
PROCEDURE INTERRUPT [SWITCH3] INT3;
BEGIN
  SWITCH_PUSH [LIGHT3] : = TRUE
END;
PROCEDURE INTERRUPT [SWITCH4] INT4;
BEGIN
  SWITCH_PUSH [LIGHT4] : = TRUE
END;
BEGIN (*MAIN PROGRAM*)
  (*INITIALIZE BOTH ARRAYS*)
  FOR I : = LIGHT1 TO LIGHT4 DO
    BEGIN
      LIGHT_STATE [I] : = FALSE;  (*ALL LIGHTS OFF*)
      SWITCH_PUSH[I] : = FALSE;  (*NO INTERRUPTS YET*)
    END;
  ENABLE; (*LET THE USERS HAVE AT IT!*)
  REPEAT
    REPEAT      (*UNTIL INTERRUPT*)
    UNTIL SWITCH_PUSH [LIGHT1] OR SWITCH_PUSH[LIGHT2]OR
          SWITCH_PUSH [LIGHT3] OF SWITCH_PUSH[LIGHT4];
    FOR I : = LIGHT1 TO LIGHT4 DO (* SWITCH LIGHTS *)
      IF SWITCH_PUSH [I] THEN
        BEGIN
          SWITCH_PUSH [I] : = FALSE;
          LIGHT_STATE [I] : = NOT LIGHT_STATE [I];
            (*TOGGLE IT*)
          OUT [I] : = LIGHT_STATE [I]
        END
      UNTIL FALSE; (*FOREVER DO THIS LOOP*)
    END. (*OF PROGRAM*)
```

第七节　非标准数据读写方式

7.1　INP 和 OUT 数组

PASCAL/MT+提供了直接对硬件输入输出口进行操作的功能。它提供了两个预先宣称过的字节型数组 INP[和 OUT，而且可以用口地址常数与表达式作下标。INP 数组只能用于表达式中，而 OUT 数组只能用于赋值语句的左边。如果把 INP 的值赋给类型为 INTEGER 的变量，则高位字节为00。

这二个数组可以用范围在 1 至 255 之间的整型表达式作下标。在此可有二种句法：如果使用常量作下标则产生嵌入代码；如使用表达式作下标则需要特殊的句法：将表达式括在方括号中的圆括弧内。用第二种方法时产生调用运行时子程序来处理 I/O 变量口。

如果这二个数组在递归环境中使用，则它们必须总是使用方括号中的圆括弧这种格式，而且不管下标是否是常量或表达式，都另外调用运行时库中适当的例行程序来处理 I/O 变量口。见下例：

```
OUT[(portnum+i)] := $88;  (用表达式与(或)递归)
OUT[0] := $88;  (用赋名常量或文字常量，无递归)
OUT[(0)] := $88;  (递归)
J := INP[(portnum)];  (用表达式与(或)递归)
```

编译并不总是能够判断出该用但没用圆括弧的情况。假如你的程序只是被编译而不运行，那么在 INP 或 OUT 语句中可能存在缺少圆括弧的情形。

7.2　重定向输入/输出方法

除标准 I/O 功能外，PASCAL/MT+ 还提供了另一种方法，用这种方法，程序员自己可用 PASCAL/MT+写一个对字符的 I/O 驱动程序。这种功能可以使准备放在 ROM 中的程序，与系统无关。它还允许用户利用系统已有的读/写转换程序来写一个字符输入输出的子程序，将数据写到字符串或 I/O 口中。

重定向 I/O 简单易用，用户只要简单地在 READ、WRITE 或 WRITELN 语句中将自己编写的取字符子程序（用于 READ）或写字符子程序（用于 WRITE）的地址（在方括号中）放在左圆括号后、参数表之前即可。例：

```
READ([ADDR(getch)],...);
WRITELN([ADDR(putch)],...);
```

“getch”与“putch”子程序可以用 PASCAL/MT+写，也可以用汇编写。它们需要的参数如下：

```
FUNCTION getch : CHAR;
PROCEDURE putch(outputch : CHAR);
```

这二个例行程序的宣称如上所示，名称不一定是 getch/putch，但取字符子程序一定不能有参数表，而写字符子程序则一定要有一个字符型参数。用户可以用 ADDR 函数把过程的地址赋给一个指针，然后说明此指针，即：READ([P],...)。它不节省执行时间，但确实节省了键入字符的时间。

注意：由于 EOLN 和 EOF 需要一个操作的文件，所以 READLN 和 EOF/EOLN 不能使

用重定向 I/O。因为 READLN 使用了 EOLN，因此读一字串变量（需用 READLN）是不允许的。这是因为@RST(读字串)子程序在不指定文件时就要直接读终端设备。用户可以重写一个@RST 子程序以便完成系统使用的控制台所要求的任何输入和编辑功能，这并不影响不使用重定向 I/O 的程序。

参阅附录 PASCAL 文件部分中重定向 I/O 示范程序。

7.3 绝对变量

〈绝对变量〉::=ABSOLUTE〔〈常数〉〕〈变量名〉

如果知道编译时的地址，用户可以宣称绝对（ABSOLUTE）变量。用户用专门的句法在 VAR 宣称中宣称一个变量是绝对地址的。在冒号后、变量类型前，用户放一关键字 ABSOLUTE，随后在方括号中放入变量地址(〔...〕)，编译不在用户的数据区中给绝对变量分配空间。所以用户有责任保证没有任何编译分配的变量与绝对变量冲突。注意：字串变量不是在任何地址都可以存在。在 8080 上，字串必须在 100H 至 FFFFH。这样做使得运行时子程序能将一个字串的地址与栈顶的字符区别开来。

```
I:          ABSOLUTE 〔S8000〕 INTEGER;
SCREEN : ABSOLUTE 〔SC000〕 ARRAY 〔0..15〕 OF ARRAY 〔0..63〕 OF
             CHAR;
```

第八节 INLINE 与小型汇编器

PASCAL /MT+具有一种非常有用的内部功能，称为 INLINE。INLINE 允许用户在 PASCAL /MT+ 的过程和函数中间插入数据。按此方式，小段机器码程序和常数表可以插到一段 PASCAL/MT+ 程序中，而无需使用外部汇编的子程序。

8.1 语法

INLINE 的句法特点类似于 PASCAL 中的过程调用句法。在 INLINE 这个字的右边加一个左括号“(”，后面可以跟任意个参数，参数之间用斜杠字符“／”分隔，然后用右括号“)”结尾。在斜杠之间的参数必须是常数，或者是具有常数值的变量名。这些常数可以是下列类型中的任意一种：字符型（char），字串型（string），布尔类型（boolean），整数型（integer），实数型（real）。用户应注意，在引号之间的“字串”不产生长度字节，而只是字串的全部数据。注意，对于堆栈结构寻址方式（例如使用 $S+ 或更复杂的 CPU）求出的变量值是作为数据段里的偏移量。对于使用动态寻址方式的 CPU（例如 8080，8085 和 Z80）来说，地址就是数据的绝对地址。

为了便于跳转，在 INLINE 性质的句法中，*+n 和*-n 也被作为合法的操作数（n 是一个整数）。例：

```
INLINE ("IN/$03/
        "ANI/$02/
        "INZ/*-4);
```

“*”表示在它之前的操作码，而不是“*”字符本身的地址。

整数型字母常数，若其值在 0 到 255 范围之内，将被分配一字节；但对于已命名的、已宣称的整数常数则总是分配二字节。

除常数数据之外，对于8080/8085 CPU，PASCAL/MT+系统还提供一个内部的小型汇编器。用户可在双引号之后紧跟汇编语言助记符，编译的第一遍扫描将把助记符翻译成合适的16进制值(例："MOV A,M 将翻译成"7E)。将来这种功能还可以扩展为能够处理其它处理机的汇编语言。可参看附录中的小型汇编器的操作码表。例：

```
INLINE ("LHLD  /        (*LHLD OPCODE FOR 8080*)
        VAR1   /        (*REFFRENCE VARIABLE*)
        "SHLD  /        (*SHLD OPCODE FOR 8080*)
        VAR2   );       (*REFERENCE VARIABLE*)
```

利用INLINE，可以在程序中插入机器码，或建立编译运行时表。以下两节将给出各种用法举例。

8.2 编码举例

下面这段编码是怎样用INLINE来写一个过程的示范，此过程调用CP/M，并返回一个值。这子程序包含在运行时软件包中，名为@BDOS。例：

```
FUNCTION @BDOS (FUNC : INTEGER;PARM : WORD) : INTEGER;
CONST
  CPMENTRYPOINT = 5;    (*SO IT ALLOCATES 2 BYTES*)
VAR
  RESULT : INTEGER;     (*SO WE CAN STORE IT HERE*)
BEGIN
 INLINE (S2A / PUNC /              (*LHLD FUNC     *)
         S4D /                     (*MOV C,L       *)
         S2A / PARM /              (*LHLD PARM     *)
         SEB /                     (*XCHG          *)
         SCD / CPMENTRYPOINT /     (*CALL BDOS     *)
         S6F /                     (*MOV L,A       *)
         S26 / $00 /               (*MVI H,0       *)
         S22 / RESULT);            (*SHLD RESULT   *)
 @BDOS := RESULT;                  (*SET FUNCTION VALUE*)
 END;
```

8.3 常量数据的生成

下面的一段程序是利用INLINE来构造一个编译运行时的表。例：

```
PROGRAM DEMO_INLINE;
TYPE
   IDFIELD = ARRAY [1..4] OF ARRAY [1..10] OF CHAR;
VAR
  TPTR : ^IDFIELD;
PROCEDURE TABLE;
BEGIN
  INLINE (   'MTMICROSYS' /
```

```
                'SOFTWARE  ' /
                'POWER     ' /
                'TOOLS.....' );
END;
BEGIN (*MAIN PROGRAM*)
  TPTR := ADDR(TABLE);
  WRITELN(TPTR^[3]);          (*SHOULD WRITE 'POWER  '*)
END.
```

如果INLINE程序段是在一个递归模块之内，则求TABLE的ADDR时，不给出表(TABLE)的地址。因为递归管理时产生了附加的代码，不使用$Q开关会额外产生6个字节的代码，使用$Q开关将产生5个字节的代码。

还有，表和求表(TABLE)的ADDR语句必须在同一模块中。

第九节　递归和非递归

用户如不请求，PASCAL/MT+不会产生利用递归的程序码。因为递归程序码产生后如不使用，则代码段占了一定的空间并且使执行速度下降。在编译开关一节里，第二章2.4.2，讲述了使用递归的方法。

所有过程的返回地址存放在硬件堆栈里。在补缺状态下，这个堆栈包含128个字节。如果递归嵌套很多层，这个堆栈的尺寸就不足以运行程序。在进行递归时，程序可能复盖局部数据或全部数据，解决方案是使用更大的硬件堆栈。在第五节里涉及怎样扩大硬件堆栈。

第四章　运行时错误处理

PASCAL/MT+系统在运行时提供三种类型的检查：界限、例外和输入/输出。

- 在第四章第一节里将讨论界限检查。
- 在第四章第二节里将讨论例外检查。
- 在第四章第三节里将叙述一种可由用户参与的处理错误的方法。
- 在第四章第四节里将讨论运行时 I/O 的检查。

系统的补缺形式是不做界限检查而做例外检查，也不做 I/O 检查。因为处理错误的子程序不是 PASLIB 的一部分，而是提供的源码，可由用户校正、修改以适合各自的需要。

处理出错的一般步骤是，错误检查和错误子程序置布尔标志。这些布尔标志和错误码一起被装入堆栈，然后在调用内部子程序@ERR 时使用这两个参数。

@ERR 子程序将测试布尔参数。布尔值为'假'时则未出错，@ERR 子程序将退回到被编译的代码段上继续执行。为'真'时@ERR 子程序按下面的叙述处理。

下面列出传递给@ERR 子的程序错误码：

值	意义
1	用 0 做除数
2	堆溢出（未用，见后）
3	字串溢出（未用，见后）
4	数组与子界出错
5	浮点数下溢
6	浮点数上溢
7	9511 超越错误

第一节　界限检查

界限检查的对象是数组下标和子界赋值。当读入一个子界变量时并不进行检查。当进行界限检查时，编译对于每个数组下标和子界赋值产生一个对@CHK 的调用。子程序@CHK 在堆栈中留有一个布尔值和错误码 4，在@CHK 被调用之后，编译对@ERR 作一次调用。如果界限检查是处在使能状态，@ERR 会打印一个信息，且询问是继续编译还是中止编译。如果界限检查未被使能，而数组下标又超出了允许的范围，则会产生不可预测的结果。对于子界赋值，它的值将以字节为单位被截断。

第二节　例外检查

例外检查的范围包括：用 0 除整数和实数，实数下溢和上溢，字串溢出。补缺状态是使能状态。在现行文本中，$ x - 不关闭例外检查。当例外检查被使能后，当需要的时候，编译会设置新需要的出错标志（零除数、字串溢出、实数下溢和上溢），在每个置标志的操作之

后，调用@ERR 子程序。如发生了浮点数下溢，返回 0.0，但@ERR 不打印信息；如发生浮点数上溢，则打印一个信息并返回一个较大的数值。用零除则返回一个最大值。对于堆溢出不做任何事，对于字串溢出则发生截断。

第三节　用户实现的处理程序

用户可以自己写一个 @ERR 子程序而不使用系统的子程序。用户应按如下方式宣称子程序：

```
PROCEDURE @ERR (ERROR : BOOLEAN;ERRNUM : INTEGER) ;
```

如上所述，子程序可以被调用。每次需要查错时，子程序应检查布尔变量 ERROR，如结果为'假'则退出；如为'真'，用户可决定做适当的事情。如要使用自己写的 @ERR 子程序，而不用 PASLIB 里的 @ERR，则要将子程序连接到 PASLIB 的前面以解决对它的调用。ERRNUM 的值同前面介绍的一样。

第四节　I/O 错误处理

编译运行时子程序@ BDOS，除了用户检查需要，在 IORESULT 中返回 CP/M 错误编码以外，不做任何 I/O 错误处理。在系统盘上的 XBDOS.SRC 文件中，可找到一个可更改的@ BDOS 子程序。当 XBDOS 程序调用 BDOS 时，将使用 CP/M的 I/O 系统调用，15，16，21，和22（分别为 OPEN 或 RESET，CLOSE，WRITE 或 PUT，以及 REWRITE）；它产生一个对 IOERR 的调用，并把 CP/M 系统调用号传给 IOERR。IOERR 子程序存在于 IOERR.SRC文件中，可以按用户需要做改动。

为使用 I/O 错误处理程序，要编译 IOERR.SRC 和 XBDOS.SRC 这二个程序。在配给盘上使用文件 IOCHK.BLD 作为 LIB/MT+的输入。这样，一些浮动文件就构成了程序库。这个生成的名为 IOCHK.ERL 的程序库必须连接到 PASLIB 之前，但在目录中查不到，因为所有对于@BDOS 的调用均由 PASLIB 产生。关于程序库的使用可看第二章有关章节。

由于对@BDOS 的调用均由 PASLIB 产生，所以@BDOS 不能宣称为外部子程序。同时，对 IOERR 的调用均由@BDOS 产生，所以 IOERR 也不能宣称为外部子程序。

第五章　一个PASCAL/MT+程序解剖

本章列出数据类型和与类型对应的存储方式，还详细介绍了字串的使用。
描述在CP/M的管理下，内存中PASCAL/MT+.COM命令文件的层次结构。

第一节　数据类型

本节描述在PASCAL1/MT+中，怎样执行标准的和扩展的PASCAL数据类型。下表列出各数据类型：

Date type	Size	Range
CHAR	1 8-bit-byte	0..255
BOOLEAN	1 8-bit-byte	false..true
INTEGER	1 8-bit-byte	0..255
INTEGER	2 8-bit-bytes	-32768..32767
BYTE	1 8-bit-byte	0..255
WORD	2 8-bit-bytes	0..65535
BCD REAL	10 8-bit-bytes	18 digits, 4 decimal
FLOATING REAL	4 8-bit-bytes	10E-17..10E+17
STRING	1..256 bytes	----------
SET	32 8-bit-bytes	0..255

1.1 字符

字符型数据操作时，每字符占用一个8位字节。保留字PACKED是对于字符数组而言的。字符变量的范围从CHR (0) 到CHR (255)。当字符变量进堆栈占16位时，高位字节放入00，这就使得ORD、ODD、CHR和WRD都可对字符进行运算。

1.2 布尔变量

布尔型数据操作时，每个布尔变量占用一个8位字节。当它进堆栈时，放入8位的0是为了和内部操作符及子程序一致。允许保留字PACKED，但不能把数据结构压缩到每个元素少于一个字节（这对于用不用压缩指令都是同样的）。ORD(TRUE)=0001和ORD(FALSE)=0000。布尔操作符AND, OR和NOT是单字节操作。用户使用&，！和～操作符（见第七章第八节作为16位操作符。

× × × × × × × 0/1

（×指无关的值）

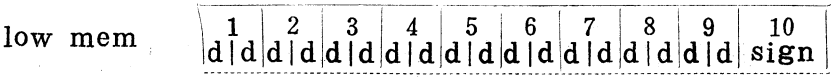
1.3 整数

整数型数据在操作时每个变量占用二个8位的字节。字节的顺序取决于CPU。在8080、8085、Z80、8086和8088中，低位字节放入低位地址，高8位放在高位地址之中。MAXINT = 32767，INTEGERS范围从－32768到32767。宣称一个值在0..255之内的整数子界，仅占一字节的存储单元而不是两个字节。整数常数可以是十六进制数，在十六进制数之前要加一个"$"符号（例$0F3B）。

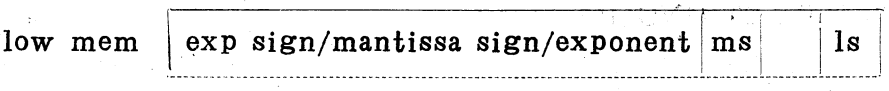
1.4 实数

在PASCAL/MT+中，实数型数据操作时有两种不同的方式，为两种不同的需要服务。

对于商业应用，实数型数据按二-十进制编码（BCD）操作，具有18位数、4位小数。在计算过程中，第4位小数之后的数自动进行四舍五入处理，如果需要格式化输出时，在指定的位置上也自动四舍五入。实数BCD数的格式是：1..9字节是经压缩的数字，一个字节两位数。第10字节包含符号：0表示正数，$FF表示负数。在编译时，必须使用开关B要求的编译器为每个实数保留10个字节，而不是补缺值4个字节。当使用BCD格式时，必须链接BCDREALS.ERL。对于这种格式，不提供超越子程序和平方根函数。



对于科学和工程应用领域，实数型数据采用二进制浮点数。在PASCAL/MT+中使用的浮点数与AMD9511硬件浮点数单元完全兼容。实现一个浮点数需要32位（4字节）。第一个字节内包含尾数符号、指数符号和指数，剩余三字节包含尾数。此种格式的精度大致为6.5(数位)有关二进制格式的详细介绍，读者可参阅AMD9511硬件手册。



ms = most significant bits
ls = least significant bits

当程序中使用实数时，编译使用浮点数格式作为补缺类型。

PASCAL/MT+在软件和硬件中均实现这种浮点数类型。标准浮点数组件与运行时软件（FPREALS.ERL）一起使用。AMDIO.SRC文件，可用来修改9511的口地址，以匹配装入用户系统的运行时软件包的文本。关于用构造软件来使用9511片子的详细介绍，见附录中9511接口说明。

1.5 字节

字节型数据占用一个单字节，它与整数和字符型数据均兼容。这对管理控制字符，处理字符运算等非常有用。字符和整数可向一个字节赋值。

1.6 字

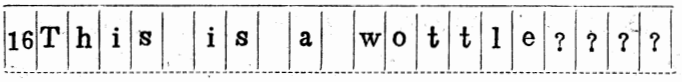
字是一个无符号的内部机器字。执行对于字类型数的表达式的运算都是无符号的。另外，所有的比较也是无符号的。字类型数据的设计使它和指针类型数据具有同样的尺寸。

1.7 字串

1.7.1 定义

预宣称类型字串象一个压缩的字符数组，它的零字节内含有字串的动态长度值，从1到n字节内含有字符。字串最长可达255个字符。字串长的补缺值是80字符，当宣称一个字串型变量时，可以更改这个长度值（见下例）。

字串"This is a wottle"长度为16个字符。下面的简图说明这些字符是如何存在一个宣称长度为20的字串里的。



如果在字串里的字符数目少于宣称的长度，那么在尾部的那些字节是未定义的。注意，字串长度值存放在头一个字节里，对于这个字串总共需要的字节数是17。例：

```
VAR
  LONG_STR :        STRING;      {This may contain up to 80
                                  characters}
  SHORT_STR :       STRING[10];  {This may contain up to 10
                                  characters}
  VERY_LONG_STR :   STRING[255]; {This may contain up to 255 char-
                                  acters, the maximum allowed.}
```

1.7.2 赋值

对字串变量赋值时通过赋值语句，使用READ或READLN读进一个字串变量，或使用预定义的字串函数和程序。例：

```
PROCEDURE ASSIGN;
VAR
  LONG_STR  :STRING;
  SHORT_STR :STRING[12];
  BEGIN
  LONG_STR := 'This string may contain as many as eighty characters';
  WRITELN(LONG_STR);
  WRITE('type in a string 10 characters or less : ');
  READLN(SHORT_STR);
  WRITELN(SHORT_STR);
  SHORT_STR := COPY(LONG_STR,1,11);
  WRITELN('COPY(LONG_STR..) =',SHORT_STR);
END;
OUTPUT :
This string may contain as many as eighty characters
type in a string 10 characters or less : {123456}  (USER INPUT)
123456
COPY(LONG_STR..) = This string m
```

存取字串变量的单个字符就象把字串当作字符数组。通过用常数、变量和表达式这些规范的数组下标，允许对字串内的单个字节进行到赋值和存取。在变量宣称的长度内存取字串是合法的，不产生运行时的错误。即使对字串的某一部分存取，而这部分超出现行的动态长度，仍然不出错。例如：如果字串实际上长为20字符，宣称长度为30，STRING[25]是可接受的。例：

```
PROCEDURE ACCESS;
VAR
  I:INTEGER;
BEGIN
  I:= 15;
  LONG_STR = '123456789abcdef';
  WRITELN(LONG_STR);
  WRITELN(LONG_STR[6], LONG_STR[ I-5 ]);
  LONG_STR[16]:='*';
  WRITELN(LONG_STR[ 16 ];
  WRITELN(LONG_STR); (*will still only write 15- characters*)
END;
```

Output:

```
123456789abcdef
6a
*
123456789abcdef
```

1.7.3 比较

在两个字串类型的变量之间(不考虑它们的长度),或者在一个字串变量和一个实际的字符串之间是可以进行比较的。实际的字符串是用引号标记括起来的字符序列。也可在一个字串和一个字符之间进行比较。使用 CONCAT 缓冲区,编译程序"迫使"字符成为字串,所以一个字符和 CONCAT 函数的结果进行比较是无意义的,因为比较结果总是相等。例:

```
PROCEDURE COMPARE;
VAR
  S1,S2:STRING[10];
  CH1  :CHAR;
BEGIN
  S1:= '012345678';
  S1:='222345678';
  IF S1 < S1 THEN
    WRITELN(S1,' is less than',S2);
  S1:='alpha beta';
  IF S1 = 'alpha beta 'THEN
    WRITELN('trailing blanks don't matter')
  ELSE
    WRITELN('trailing blanks count');
  IF S1 =  'alpha beta' THEN
    WRITELN('blanks in front don't matter')
  ELSE
```

```
    WRITELN('blanks in front do matter');
  IF S1 = 'alpha beta' THEN
    WRITELN(S1, ' = ',S1);
  S1 := 'Z';
  CH1 := 'Z';
  IF S1 = CH1 THEN
    WRITELN('strings and chars may be compared');
END;
```

Output:

```
012345678 <222345678
trailing blanks don't matter
blanks in front do matter
alpha beta = alpha beta
strings and chars may be compared
```

1.7.4 读，写字串

调用 WRITE 或 WRITELN 过程可将字串写入正文文件（text）。WRITELN将在字串后面加上回车和换行。因为一个字串总是用回车换行结尾，所以读字串总是通过 READLN 语句。而 READ 不能正确处理行结束字符，所以不能用。

Tab 键在读入一个字串数据类型时，被扩展成空格。

1.8 集合

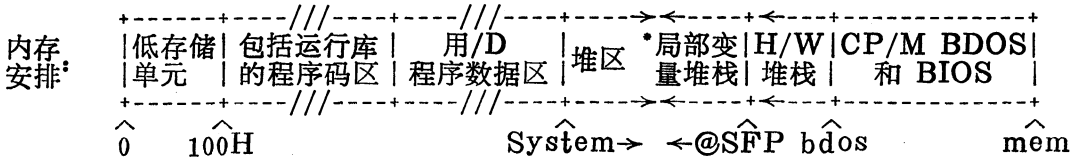
集合数据类型总是作为 32 字节的项来存储的。集合的每个元素按一位存储。每字节的最低位是集合中对应字节的第一位。下面所示是集合'A'..'Z'（65..122 位）

```
字节数 00 01 02 03 04 05 06 07  08 09 0A 0B 0C……1F
       -- -- -- -- -- -- -- --  -- -- -- -- --
内容   00 00 00 00 00 00 00 00  FE FF FF 07 00 ……00
```

第二节 执行时命令文件的结构

2.1 内存布局和系统变量

一个经过编译的程序，由 CP/M 连接并装入内存以便执行，其在内存的存储层次如下所示。如果用户在连接时没有指定/D 开关，那么在程序码区里同时含有程序码和数据。



*仅在递归情形中使用

符号///表示从连接程序输出的可变尺寸区域。堆区域向着高存储区增加，局部变量堆栈区向着低存储区增加。仅当程序内含 $S+开关（表示用递归）时，才使用局部变量堆栈。硬件堆栈内含有过程的返回地址和为了计算表达式而用的临时计算堆栈。局部变量堆栈内包

含传递参数和局部程序调用变量。外部整数（SYSTEM）指向堆栈的顶部，初始化时指向数据区域后的第一个位置。"SYSTEM"这个指针由 NEW 操作加以校正。外部的整数@SFP 指向局部变量堆栈的顶部，初始化时指向硬件堆栈顶再减去 128 个字节，由@LNK（配置堆栈结构）和@ULK（撤消堆栈结构）子程序加以校正。由（@SFP-SYSTEM）子程序来计算内部函数 MEMAVAIL(在不用 FULLHEAP 的系统中)。

2.2 硬件堆栈尺寸

硬件堆栈初始化为 128 个字节，但用户可以改变它的尺寸。用户可把 @SFP 作为外部整数，从中减去硬件堆栈所需要的附加空间（想要减小栈区时就做加法）。见下例：

```
VAR @SFP : EXTERNAL INTEGER;
(* 仅在主程序中! ! ! *)
@SFP := @SFP—MORE_HW_STACK_SPACE_IN_BYTES;
```

如果在中断驱动的系统下运行程序码，则经常需要增加硬件堆栈的尺寸。

2.3 程序结构

PASCAL/MT+ 编译器产生的程序模块具有非常简单的结构。在模块的首部设置一个跳转表，内含一个跳步操作，可跳到模块中每一个过程或函数。在跳转的首部为主程序的跳转指令保留了空间。当这个模块确是一个"模块"而不是一个"程序"时不用这一跳步。还有，在"程序"中，前面设有 16 个字节的头信息用于存放程序复盖信息（在非复盖程序中，放的是空操作 NOP）。在主程序的开始，编译程序产生代码，根据 CP/M 标准位置将单位 6 中的内容装入堆栈指针。目标代码放在 ROM 中使用的用户，由于 ROM 需要，必须使用 $Z开关，以建立初始堆栈指针。编译程序还产生一个对于 $INI 子程序的调用，这个子程序初始化"输入"和"输出"正文文件。当开关 $S+ 有效时，还要初始化所用的堆栈结构指针。还有，使用ROM 的用户将希望重写@INI 子程序以适合他们的需要。

第六章　PASCAL/MT⁺语言定义

第一节　引　　言

本章旨在定义 PASCAL/MT⁺ 语言的特性。在 PASCAL/MT⁺ 中所具有的而在 Jensen和 Wirth 的定义里面没有的那些特性要更详细地讨论。因此，如果你不懂 PACSAL，建议阅读本章时学习一下 Jensen 和Wirth 写的 'User Manual and Report' 一书。本章的标题与 'report' 中的一样。

第二节　PASCAL/MT⁺ 语言简述

ISO PASCAL 包括的数据类型有实数、整数、字符、布尔量、多维数组、用户定义的记录、指针、文件变量、用户定义的类型和常数、集合（本文本提供的是 256 个单字节元素的基本类型）、枚举型和子界类型。

另外 ISO PASCAL 还包括过程、函数及程序。过程和函数作为参数传递到一个 PASCAL 子程序中，是标准 ISO 的一部分。还包括一致型数组，即：相同下标类型和元素类型，但不同尺寸的数组能被传递到相同的过程中。在语法上，由 PROGRAM 语句支持外部参数。

象在标准 PASCAL 中 RESET、REWRIT、GET、PUT、READ、WRITE、READLN 和 WRITELN 所定义的那样，TYPED 和 TEXT文件被提供。补缺 I/O 文件 INPUT 和OUTPUT 也被定义。

提供所有 ISO 语句，这包括 WITH、REREAT…UNTIL、CASE、WHILE循环、FOR循环、IF..THEN..ELSE 和GOTO。

提供 PACK 和UNPACK，这并不影响程序的效率（在字节层，数据结构都被紧缩）。NEW 和 DISPOSE 也被提供，它们分配和释放 HEAP 空间。

模块编译是 MT⁺ 编译程序的一个扩充。变量和子程序可以是公共的也可以是局部的。任何其他模块或主程序都能调用它们。这个动能可以方便地将浮动汇编语言作为整个程序的一个单独模块。

各种 PASCAL编译文本都可实现扩展的数据类型STRING，BYTE 和WORD。STRINg类型含有一个表示长度的字节，其后跟随可能的最大字节数。还提供了一些子程序以便完全成如下一些操作：插入一个字符或字串，从一个字串中进行删除操作，寻找字符在字串中的位置，拷贝字串的一部分到另一个字串中，将两个或多个字串和/或字符连接成一个字串。BYTE 是一字节数据项表示 0～255 中的一个值。WORD是依赖于CPU的，对子位CPU来说用二个字节，16 位CPU 用一个字节。

其他管理磁盘文件的过程也被提供。磁盘上的一个文件是与一个内部文件相联系的，它能被关闭或删除。除了在 CP/M 1.4 以下运行或在兼容于 CP/M 1.4 的操作系统下运行，各种版本都允许随机文件的使用。

有关 BITS 操作和 BYTES 操作的子程序有：TEST, SET, CLEAR, SHIFTRIGHT 和 LEFT 返回 HI 或 LOW（一个变量的高字节或低字节），SWAP(交换一个变量的高低字节)。

程序中存取子程序有：返回一个变量或子程序的地址，返回一个变量或类型的尺寸，将一定数量的字节从内存某位置移到另一位置上，赋给数据项的某个字符。另外，在任何时间可对堆空间变量进行存取。提供对堆的废料回收。

允许对非布尔类型进行逻辑操作。

十六进制文字数值能用'$'符号作开始符来表示。

提供隐含文件。

提供在 CASE 语句中的 ELSE 子句。

提供 CHAINING 和 OVERLAYS 程序。所谓链是指一个程序的代码能完全由接着的下一个程序的代码所代替，但在跨 CHAING（链）时堆空间能被保留。第三章描述了复盖。

第三节　符号、术语和词汇

```
<letter>::= A | B | C | D | E | F | G | H | I | J |
            K | L | M | N | O | P | Q | R | S | T |
            U | V | W | X | Y | Z | a | b | c | d |
            e | f | q | h | i | j | k | l | m | n |
            n | o | p | q | r | s | t | u | v | w |
            x | y | z | @

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
            A | B | C | D | E | F  {only llowed in HEX numbers}

<special symbol>::= {reserved words are listed in the appendix}

            +  | -  | *  | /  | =  |<>| <  | >
            <= | >= | (  | )  | [  | ] |  {  | }  |
            := | .  | ,  | ;  | :  | ' | ^  |    |

             {the following are additional or substitutions:}
             (. | .) | ~ |    | ?  | !  | |  | $  | &  | *  |

             ¯, , and ? are synonyms  (see Section 7.8.1.1)

             |, and ¦are synonyms      (see Section 7.8.1.2)

             &                        (see Section 7.8.1.3)
```

第四节　标识符、数字和字串

```
<标识符>::=<字母>{<字母或数字或下划线>}
<字母或数字或下划线>::=<字母>|<数字>|
<数序序列>::=<数字>{<数字>}
<无符号整数>::= $<数字序列>  |
                *<数字序列>  |
                 <数字序列>  |
<无符号实数>::=<无符号整数>.<数字序列>|
               <无符号整数>.<数字序列>|
```

E〈比例因子〉

〈无符号整数〉E〈比例因子〉

〈无符号数〉::=〈无符号整数〉|〈无符号实数〉

〈比例因子〉::=〈无符号整数〉|〈符号〉〈无符号整数〉

〈符号〉 ::=+|-

〈串〉 ::='〈字符〉{〈字符〉}'|''

所有标识符有效字符为8个。外部标识符依赖于主系统为6个或7个字符有效。在标识符中的字母和数字之间有下划线'_'是合法的，并被编译程序所忽略，即：A-B和AB一样，aB和Ab一样，标识符可以用'@'开始，这就允许在PASCAL程序中宣称外部运行时子程序。用户应该避免使用@符号以防止运行时子程序名称发生冲突。ISO标准规定是'@'和'^'一样使用。一个编译开关'@'使得'@'字符不再起作用而仅仅当作指针符号使用。此时，'@'对标识符不再有效。

除了在'report'中列出的符号之外，'@'也是合法字母，因为用这个专门的字符作为运行时库子程序名字的第一个字母可避免与用户程序名发生冲突。允许用户调用运行时库子程序。

一个注释以'(*'开头的必须以'*)'结尾，以'{'开头的必须以'}'结尾。

〈注释〉::={〈任何字符〉}|

(*〈任何字符〉*)

为了允许嵌套注释，开始注释定义符必须与结束定义符相同，这样，在PASCAL/MT+中下列的注释为合法注释：

(*外部注释…{内部注释…}‥外部注释*)

数字可以是十进制，也可以是十六进制。整数之前加'$'号使编译将此数作为十六进制数处理。这时符号数字包括：'A'、'B'、'C'、'D'、'E'和'F'。这些符号可大写，也可小写。

第五节 常量定义

〈常量标识符〉::=〈标识符〉

〈常量〉::=〈无符号数〉|

〈符号〉〈无符号数〉|

〈常量标识符〉|

〈符号〉〈常量标识符〉|

〈串〉

〈常量定义〉::=〈标识符〉=〈常量〉

除了所有的标准PASCAL中常量定义外，PASCAL/MT+提供了空串常量的宣称：例：

```
nullstr = '';
```

第六节 数据类型定义

〈类型〉::=〈简单类型〉|〈构造类型〉|指针类型〉

〈类型定义〉::=〈标识符〉=〈类型〉

6.1 简单类型

〈简单类型〉::=〈纯量类型〉|〈子界类型〉|〈类型标识符〉

〈类型标识符〉::=〈标识符〉

6.1.1 纯量类型

〈纯量类型〉::=(〈标识符〉{,〈标识符〉})

6.1.2 子界类型

在 PASCAL/MT+ 中以下的类型是标准的：

INTEGER, REAL, BOOLEAN, CHAR, BYTE, WORD, STRING

在 PASCAL/MT+ 中还有以下附加的标准类型。关于所有标准类型和结构类型的表示和使用的说明见第五章第一节。

STRING：紧缩字符数组[0..n]：0 字节存放动态长度。字节1..n 存放字符。

BYTE：具有特定属性的子界 0..255，它与字符类型兼容。

WORD：无符号整数。

6.1.3 子界类型

〈子界类型〉::=〈常量..常量〉

6.2 结构类型

〈结构类型〉::=〈非紧缩结构类型〉|
PACKED〈非紧缩结构类型〉

〈非紧缩结构类型〉::=〈数组类型〉|
〈记录类型〉|
〈集合类型〉|
〈文件类型〉

可以使用保留字 PACKED，但无论 PACKED 保留字是否被使用，所有结构都以字节为单位被紧缩，故这个保留字不产生什么影响。同样保留字 UNPACKED 也不产生影响。

6.2.1 结构类型

〈数组类型〉::=〈常规数组〉|
〈串数组〉

〈串数组〉::=STRING〈最大长度〉

〈最大长度〉::=[〈整常量〉]|〈空〉

〈整常量〉::=〈无符号整数〉|〈整常量标识符〉

〈整常量标识符〉::=〈标识符〉

〈常规数组〉::=ARRAY[〈下标类型〉{,〈下标类型〉}]OF〈成份类型〉

〈下标类型〉::=〈简单类型〉

〈成份类型〉::=〈类型〉

对于 STRING 类型变量，其长度补缺值为 81 个字节（80 个数据字符）。用户要求的长度可以在 STRING 后面用方括号来规定。这个长度必须是一个文字常量或是一个宣称的常量，它表示数据部分的长度（系统将另外再分配一个字节用来存放该字串的长度），例如：STRING[5] 或STRING[XYZ]（这里 XYZ 是个等于 5 的常量）将占用 6 个字节的存储单元。

6.2.2 记录类型

〈记录类型〉::=RECORD〈域表〉END

〈域表〉::=〈固定部分〉|〈固定部分〉,〈变体部分〉|〈变体部分〉

〈固定部分〉::=〈记录段〉{;〈记录段〉}

〈记录段〉::=〈域标识符〉{,〈域标识符〉}:〈类型〉|〈空〉

〈变体部分〉::=CASE〈标志域〉〈类型标识符〉OF〈变体〉{;〈变体〉}

〈变体〉::=〈条件标号表〉:(〈域表〉)|〈空〉

〈条件标号表〉::=〈条件标号〉{,〈条件标号〉}

〈条件标号表〉::=〈常量〉

〈标志域〉::=〈标志符〉:|〈空〉

6.2.3 集合类型

〈集合类型〉::=SET OF〈基本类型〉

〈基本类型〉::=〈简单类型〉

一个基本类型的最大范围是0··255。例如，〔0··10000〕的集合或0··256的集合都是不合法的，但CHAR的集合或0··255的集合是合法的。

6.2.4 文件类型

〈文件类型〉::=文件{OF〈类型〉}

文件可以有类型定义，也可以没有。非类型的文件用于CHAING（见第三章第五节）中，也可以用于BLOCKREAD和BLOCKWRITE过程中（见第三章第四节）。用户必须小心地使用非类型文件。

当希望读ASCII字符文件和使用整数、实数的隐含转换时，用户应该使用预先定义的类型TEXT。TEXT并不完全与FILE OF CHAR相同，它有被隐含在READ和WRLTE过程中的转换，它也能用于READLN和WRITELN。TEXT类型的文件可以按以下方式宣称：'VAR F：TEXT'。而'VAR F：FIZE OF TEXT'是不正确的宣称方法。参阅PASCAL文件处理的附录。

6.3 指针类型

〈指针类型〉::=^〈类型标识符〉

当编译的RELAXED类型检查功能被赋能（补缺）时，存在弱类型检查。除此之外，指针类型与标准的相同。在这种情况下，用于指针的字和指针在各种情形下都是兼容的。另外允许指针运算。注意编译开@，它可代替字符'^'（在一个程序中，这两种字符能混合使用）。

6.4 类型和赋值兼容

'标准PASCAL'中最常见的问题是编译产生的类型冲突错误。如果变量被传递到一个VAR参数，那么类型必须一致。对于表达式和赋值语句，类型必须兼容。为了了解一致类型和兼容之间的差别，用户可以将类型看作指向编译时记录的指针。如果你宣称一个类型（例如：T=ARRAY〔1··10〕OF INTEGER），那么宣称为类型T的任何变量都真正地指向类型T所描述的记录。另一方面，如果你宣称了下面两个变量：

```
VAR A1 : ARRAY〔1··10〕 OF INTEGER;
    A2 : ARRAY〔1··10〕 OF INTEGER;
```

那么，它们不是一致的，因为编译对每一个类型都创造一个新记录，因此A_1和A_2在编译时并

不指向内存中的相同记录。一般的规则是：如果你不能找到归一到相同类型定义的路径，那么这些类型不是一致的。

下面是从美国国家标准委员会的 ISO 草案标准（DPS-7185）中摘录的一段关于兼容的 ISO 标准定义：

类型T_1和T_2被认为是兼容的，只要下面四个句子中的任一句成立：

(a) T_1 和 T_2 是相同的类型。

(b) T_1 是 T_2 的一个子界或 T_2 是 T_1 的一个子界，或者 T_1 和 T_2 两者都是相同宿主类型的子界。

(c) T_1 和 T_2 都被指派为紧缩的或者 T_1 和T_2 都没被指派为紧缩的。

(d) T_1 和 T_2 都是具有相同数量的成份的串类型*。

…赋值兼容性。如果下面五个条件中的任何一个成立，那么类型 T_2 的值将被规定可与类型 T_1 的赋值兼容。

(a) T_1 和 T_2 是相同的类型，但这个类型不能是文件类型，也不能是具有文件成份的结构类型（这个规则将递归地被解释）。

(b) T_1 是实型的而 T_2 是整型的。

(c) T_1 和 T_2 是可兼容的原始类型**，且T_2类型的值是在由类型T_1所规定的闭区间中。

(d) T_1 和 T_2 是可兼容的集合类型，并且类型 T_2 的所有成员都在由类型 T_1 所规定的闭区间中。

(e) T_1 和 T_2 是可兼容的字串类型*。

在任何使用赋值可兼容规则的地方：

(a) 如果 T_1 和T_2 是可兼容的原始类型并且类型 T_2的值不在类型 T_1 所规定的闭区间中，那将是错误的。

(b) 如果 T_1 和 T_2 是可兼容的集合类型并且类型 T_2 的成员不在由类型 T_1 的基本类型所规定的闭区间中，那将是错误的。

* 在 ISO 的PASCAL 中，字串类型是字符数组。

* * 顺序类型被称为枚举或数的子界。

CHR、ORD、ODD 和 WRD 是类型转换操作符，它不产生代码，但它告诉编译程序下面的 8 位或 16 位数据项将分别地被认为是 CHAR、INTEGER、BOOLEAN 或 WORD 类型。这些操作符可用于表达式中作为值参数但不是变量参数。

第七节　变量的宣称和表示

〈变量宣称〉::=〈变量〉：〈类型〉|〈空〉

〈变量〉::=〈变量 1〉|
　　〈外部变量 1〉|
　　〈绝对变量 1〉

〈外部变量1〉::=EXTERNAL〈变量 1〉

〈绝对变量 1〉::=ABSOLUTE〔〈常量〉〕
　　〈变量 1〉

〈变量 1〉::=〈整体变量〉|
〈成份变量〉|
〈参考变量〉

有关 ABSOLUTE 变量的更多信息可参阅第三章第七节。

7.1 整体变量

〈整体变量〉::=〈变量标识符〉

〈变量标识符〉::=〈标识符〉

7.2 成份变量

〈成份变量〉::=〈下标变量〉|
〈域指派符〉|
〈文件缓冲区〉

7.2.1 下标变量

〈下标变量〉::=〈数组变量〉[〈表达式〉{,〈表达式〉}]

〈数组变量〉::=〈变量〉

字串变量在进行下标处理时是作紧缩的字符数组结构，有效范围是0 ..max-length，max-length 的补缺值是 80。

7.2.2 域指派符

〈域指派符〉::=〈记录变量〉.〈域标识符〉

〈记录变量〉::=〈变量〉

〈域标识符〉::=〈标识符〉

7.2.3 文件缓冲区

〈文件缓冲区〉::=〈文件变量〉^

〈文件变量〉::=〈变量〉

7.3 参考变量

〈参考变量〉::=〈指针变量〉^

〈指针变量〉::=〈变量〉

第八节 表 达 式

〈无符号常量〉::=〈无符号数〉|
〈串〉|
NIL|〈常量标识符〉

〈因子〉::=〈变量〉|〈无符号常量〉
|〈函数指派符〉
(〈表达式〉)
〈逻辑非运算符〉〈因子〉|〈集合〉

〈集合〉::=[〈元素表〉]|〈空〉

〈元素表〉::=〈元素〉{,〈元素〉}|〈空〉

〈元素〉::=〈表达式〉|〈表达式〉..〈表达式〉

〈项〉::=〈因子〉〈乘操作符〉〈因子〉

〈简单表达式〉::=〈项〉|

〈简单表达式〉〈加操作符〉〈项〉|〈加操作符〉〈项〉

〈表达式〉::=〈简单表达式〉|

〈简单表达式〉〈关系操作符〉〈简单表达式〉

另一类16位变量操作符是&,!(或|)和^(或?),它们分别表示AND、OR和反码操作符NOT。它们与相应的布尔操作符有相同的优先级并接受小于等于二个字节的任意类型的操作数。

8.1 操作符

8.1.1 操作符NOT

〈逻辑非操作符〉::=NOT|～| |?|

～(与?同义)是非布尔型变量的NOT操作符。

8.1.2 乘操作符

〈乘操作符〉::=*|/|DIV|MOD|AND|&

&是对非布尔型变量的AND操作符。

8.1.3 加操作符

〈加操作符〉::=+|-|OR| | |!

! (与|同义)是对非布尔型变量的OR操作符。

8.1.4 关系操作符

〈关系操作符〉::= =|< >|<|<=|>|>=| IN

8.2 函数赋值符

〈函数赋值符〉::=〈函数标识符〉|

〈函数标识符〉(〈参数〉{,〈参数〉)

〈函数标识操〉::=〈标识操〉

第九节 语 句

〈语句〉::=〈标号〉:〈无标号语句〉|〈无标号语句〉

〈无标号语句〉::=〈简单语句〉|

〈结构语句〉

〈标号〉::=〈无符号整数〉

9.1 简单语句

〈简单语句〉::=〈赋值语句〉|

〈过程语句〉|〈goto语句〉|〈空语句〉

〈空语句〉::=〈空〉

9.1.1 赋值语句

〈赋值语句〉::=〈变量〉:〈表达式〉|

〈函数标识符〉::=〈表达式〉

赋值兼容性的规则，除了Jensen和Wirth所给的外，还包括如下两点：

1) CHAR 类型的表达式能被赋到 STRING 类型的变量中。

2) CHAR 类型和文字字符的变量能被赋到 BYTE 类型的变量中。

9.1.2 过程语句

〈过程语句〉::=〈过程标识符〉(〈参数〉{,〈参数〉})|〈过程标识符〉

〈过程标识符〉::=〈标识符〉

〈参数〉::=〈过程标识符〉|〈函数标识符〉|〈表达式〉|〈变量〉

9.1.3 GOTO 语句

〈GOTO 语句〉::=GOTO〈标号〉

从一个递归过程或一个递归调用序列中使用 GOTO 语句跳出是不允许的。

9.2 结构语句

〈结构语句〉::=〈重复语句〉|
〈条件语句〉|
〈复合语句〉|
〈开域语句〉|

9.2.1 复合语句

〈复合语句〉::=BEGIN〈语句〉{,〈语句〉}END

9.2.2 条件语句

〈条件语句〉::=〈分情形语句〉|〈如果语句〉

9.2.2.1 IF 语句

〈IF 语句〉::=IF〈表达式〉THEN〈语句〉ELSE〈语句〉|IF〈表达式〉THEN〈表达式〉

9.2.2.2 CASE 语句

〈CASE语句〉::=CASE〈表达式〉OF〈分情形表〉{;〈分情形表〉}{ELSE〈语句〉}END

〈分情形表〉::=〈标号表〉:〈语句〉|〈空〉

〈标号表〉::=〈分情形标号〉{,〈分情形标号〉}

〈分情形标号〉::=〈非实型短纯量常量〉

PASCAL/MT⁺中允许 ELSE 分句出现在CASE 语句中。此外，如果选择表达式不满足分情形选择符中的任何一个，那么程序执行时就将绕过这个 CASE 语句。这一点是不同于标准规定的，在标准 PASCAL 中这种情形将被认为是错误的。例如：

```
CASE CH OF
  'A' : WRITELN ('A');
  'Q' : WRITELN ('Q');  (* 分号可以省去 *)
  ELSE
    WRITELN ('NOT A OR Q')
END
```

9.2.3 重复语句

〈重复语句〉::=〈repeat 语句|
〈while 语句〉|〈for 语句〉

9.2.3.1 WHILE 语句

〈WHILE语句〉::=WHILE〈表达式〉DO〈语句〉

9.2.3.2 REPEAT 语句

〈RFPEAT语句〉::=REPEAT〈语句〉{;〈语句〉}UNTIL〈表达式〉

9.2.3.3 FOR语句

〈for 语句〉::=FOR〈控制变量〉:=〈for 表〉DO〈语句〉

〈for 表〉::=〈表达式〉DOWNTO〈表达式〉|〈表达式〉TO〈表达式〉

〈控制变量〉::=〈变量〉

9.2.4 WITH 语句

〈WITH 语句〉::=WITH〈记录变量表〉DO〈语句〉

〈记录变量表〉::=〈记录变量〉{,〈记录变量〉}

用户应该注意到 ISO 标准与 Jensen 和Wirth 规定的不同处。后者只有局部变量被允许做循环控制变量，这样防止了一些编程错误，例如：无意地使用某些全局变量作为 FOR 控制变量。

第十节 过程说明

〈过程宣称〉::=EXTERNAL〈过程首部〉|〈过程首部〉〈块〉

〈块〉::=〈标号宣称部分〉|
〈常量定义部分〉|
〈类型定义部分〉|
〈变量宣称部分〉|
〈过程及函数宣称部分〉|
〈语句部分〉

〈过程首部〉::=PROCEDURE〈标识符〉〈参数表〉;|
PROCEDURE〈标识符〉;|
PROCEDURE INTERRUPT[〈常量〉];

〈参数表〉::=(〈f 参数〉{,〈f 参数〉})

〈f 参数〉::=〈过程首部〉|
〈函数首部〉|
VAR〈参数组〉|
〈参数组〉

〈参数组〉::=〈标识符〉{,〈标识符〉}:〈类型标识符〉|
〈标识符〉{,〈标识符〉}:〈复合格式数组〉

〈复合格式数组〉::=VAR ARRAY[〈下标类型〉{;〈下标类型〉}]OF
〈复合数组 2〉

〈复合数组 2〉::=〈类型标识符〉|
〈复合格式数组〉

〈下标类型〉::=〈标识符〉..〈标识符〉:〈顺序类型标识符〉

〈顺序类型标识符〉::=〈纯量类型标识符〉|〈子界类型标识符〉

〈纯量类型标识符〉::=〈标识符〉

〈子界类型标识符〉::=〈标识符〉

〈标号宣称部分〉::=〈空〉|LABEL〈标号〉{,〈标号〉};

〈常量定义部分〉::=〈空〉

CONST

〈常量定义〉

{;〈常量定义〉};

〈类型定义部分〉::=〈空〉|

TYPE

〈类型定义〉

{;〈类型定义〉};

〈变量宣称部分〉::=〈空〉|

VAR

〈变量宣称〉

{;〈变量宣称〉};

〈过程及函数宣称部份〉::={〈过程或函数〉;}

〈过程或函数〉::=〈过程宣称〉|〈函数宣称〉

〈语句部分〉::=〈复合语句〉

PASCAL/MT+ 提供一个特别的过程类型——中断过程。用户选择与每个中断相关的向量，这个过程宣称形成如下：

PROCEDURE INTERRUPT〔向量号〕过程名。

关于使用 INTERRUPT 过程的详细说明参阅第三章第六节。

10.1 标准过程

以下是 PASCAL/MT+ 的内部过程名(除了将在本章10.1.1 节 中列出的 I/O 部分以外)有关参数和使用可见第三章第四节。这些过程是在程序以外预先宣称的，因而，任何具有相同名称的用户子程序将优先。

NEM	DISPOSE	EXIT	INSERT
DELETE	COPY	CONCAT	FILLCHAR
MOVELEFT	MOVERIGHT	CLRBIT	HI
LO	SETBIT	SHL	SHR
SWAP	TSTBIT	LENGTH	POS
ADDR	EXIT	MAXAVAIL	MEMAVAIL
SIZEOF	WAIT	LEAVE	

10.1.1 文件处理过程

包括所有标准文件处理过程。此外还包括过程 ASSIGN (f,string)，这里 f 是一个文件，string 是一个文字字串或变量字串。ASSIGN 将字串中的外部文件名赋予文件 f。在 RESET 和REWRIT 前使用它。更进一步的说明见 第三章4.15 节。

以下是文件处理过程名：

GET	PUT	RESET	REWRITE
ASSIGN	CLOSE	CLOSEDEL	PURGE

OPEN	BLOCKREAD	BLOCKWRITE	SEEKWRITE
CHAIN	PAGE	SEEKREAD	READ
GNB	WNB	IORESULT	
WRITE	READLN	WRITELN	

"局部"(有时称为暂存)文件不要求使用 ASSIGN 过程，在不同的 PASCAL 系统中这种文件的使用有不同的方式。在 PSCAL/MT+ 中一个文件被赋予一个系统所定义的文件名为 PASTMP??.$$$。REWRITE 子程序和连接程序接口允许这种方式。连接程序把数据区用 00 字节进行填充，以便 REWRITE 子程序能看到这个文件名是否长度为 0。在递归环境下，文件所占用的数据区在过程执行的开始并未被初始化。因此如果用户希望在一个递归环境下使用由一个局部变量存取的一个"暂存"文件时，必须在执行 REWRITE 前执行 ASSIGN (<file>,")。

10.1.2 动态分配过程

NEW DISPOSE

除了 NEW 和 DISPOSE 外，还包括 MEMAVAIL 和 MAXAVAL。

10.1.3 数据转换程序

PACK (A,I,E) UNPACK (E,A,I)

10.2 正向(FORWARD)过程说明

PASACL/MT+ 中实行正向过程宣称，建议除非要求严格 PASCAL 一致性才用这个特点。由于三遍扫描编译的性质所决定，不必正向宣称。

10.3 一致数组

用户应该注意到为了传递相似结构的数组 ISO 标准增加了一致性数组，即相同的维数、兼容的下标类型和相同的元素类型，但有不同的上下界。例如：用户能传递一个维数为 1..10 和一个维数为 2..50 的数组到一个期待数组做为参数的过程。用户定义这个数组为一个 VAR 参数(出错时产生错误 154：'actual paramenter must be a variable')，在宣称这个数组的过程时，用户还定义了这个数组的上下界变量。这些上下界项是在运行时由产生的代码来填充的。用户应该注意到为了以这种方式传递数组，下标类型必须是与数组上下界类型兼容的。

以下是一个传递两个数组到一个过程的例子，该过程显示文件 OUTPUT 中那二个数组的内容。

```
PROGRAM DEMOCON;
TYPE
  NATURAL=0..MAXINT; (*FOR USE IN CONFORMANT ARRAY
                       DECLARATION*)
VAR
  A1:ARRAY[1..10] OF INTEGER;
  A2:ARRAY[2..20] OF INTEGER;
PROCEDURE DISPLAYIT (
  VAR AR1:ARRAY[LOWBOUND..HIBOUND:NATURAL] OF
                  INTEGER);
(*THIS DECLARATION DEFINES THREE VARIABLES:
```

```
        AR1          :THE PASSED ARRAY
        LOWBOUND     :THE LOWER BOUND OF AR1 (PASSED AT
                      RUN—TIME)
        HIBOUND      :THE UPPER BOUND OF AR1 (PASSED AT
                      RUN—TIME)
*)
VAR
   I:NATURAL;
(*COMPATIBLE WITH THE INDEX TYPE OF THE CONFORMANT
 ARRAY*)
BEGIN
 FOR I:=LOWBOUND TO HIBOUND DO
  WRITELN('INPUT ARRAY(',I,')=',[I])
END;
BEGIN(*MAIN PROGRAM*)
 DISPLAYIT(A1); (*CALL DISPLAYIT AND PASS A1 EXPLICITLY
                  AND 1 AND 10 IMPLICITLY*)
 DISPLAYIT(A2)  (*CALL DISPLAYIT AND PASS A2 EXPLICITLY
                  AND 2 AND 20 IMPLICITLY*)
END
```

第十一节　函数说明

〈函数宣称〉::=**EXTERNAL**〈函数首部〉|〈函数首部〉

〈函数首部〉::=**FUNCTION**〈标识符〉〈参数表〉:〈结果类型〉;|

　　FUNCTION〈标识符〉:〈结果类型〉;

〈结果类型〉::=〈类型标识符〉

11.1　标准函数

以下是新提供的标准函数名称：

ABS	SQR	SIN	COS	SHORT
EXP	LN	SQRT	ARCTAN*	
ODD	TRUNC	ROUND	ORD	
WRD	CHR	SUCC	PRED	
EOLN	EOF	IORESULT	MEMAVAIL	
MAXAVAIL	ADDR	SIZEOF	POS	
LENGTH	LENGTH	LONG	ULONG	

*对于小于0的输入，**ARCTAN**返第四器象限的值。

11.1.1　算术函数

11.1.2　谓词

11.1.3 变换函数

WRD (X) ：X的值（X是一个变量或表达式）被变换为X的字值（这个字是无符号整数）。

其他变换函数在标准 PASCAL 中被定义。

11.1.4 标准函数的进一步说明

文件首部：（F是一个文件变量，见附录中文件部分）。

PUT (F)　　GET (F)　　RESET (F)　　REWRITE (F)

PAGE (F)　　EOF (F)　　EOLN (F)

因为 CP/M 文件结构不保存最后一个扇区的有效字节数，所以对 NON- TEXT 文件只有在最后一个扇区的全部内容都被读出后 EOF 才返回'真'值。用户必须检查伪结束记录或在另一个文件中保存这个文件所含有的记录个数。

动态分配：（T_n 是一个可变记录选字符，P 是一个指针）

NEW (P)　NEW (P, T_1, T_2, ···, Tn)　DISPOSE (P)

DISPOSE (P, T_1, T_2, ···, Tn)

数据变换过程：（更详细的描述，参阅 Jensen 和 Wirth 著文的第 106 页）。

PACK (A, I, Z)　UNPACK (Z, A, I)

算术运算：

ABS (X) 或 **ABS (I)** ——专门用于返回它的变量类型。

SQR (X) 或 **SQR (I)** ——如果送入整型数则返回整型数，等等。

以下仅适合于 PASCAL/MT+ 的浮点型变量而不适合于 BCD 型变量：

SIN (X)　COS (X)　ARCTAN (X)　EXP (X)　LN (X)　SQRT (X)

变换函数：（SC 是一个非实型短纯量）

以下的函数在编译时执行但不产生代码：

ODD (SC) : BOOLEAN　　**ORD (SC) : INTEGER**

CHA (SC) : CHAR　　**WRD (SC) : WORC**

以下的函数在运行时执行但不产生代码：

SUCC (〈除了实数类型以外的任何纯量类型〉)

PRED (〈除了实数类型以外的任何纯量类型〉)

第十二节　输入与输出

PASCAL/MT+ 提供了所有标准 PASCAL 的 I/O 功能。此外，PASCAL/MT+ 还增加了新的 I/O 功能，利用这些功能，PASCAL/MT+ 的程序员能写出他们自己的字符 I/O 驱动程序。这种功能允许在 ROM 环境下运行的程序独立于系统，并且允许用户将输入输出格式转换程序用于字串、I/O 口等。有关使用重定向 I/O 的进一步说明可参阅第三章第七节。

12.1 过程 READ

可以进行读入子界的操作，但不能进行界限检查（即使使用了界限检查开关也无效）。

12.2 过程 READLN

〈读调用〉::=〈读或读行〉｛〈｛〈文件变量〉,｝｛〈变量表〉｝）｝

〈读或读行〉::=READ|READLN

〈文件变量〉::=〈变量〉

〈变量表〉::=〈变量〉{,〈变量〉}

对字串类型的变量，仅仅 READLN 能被使用。不能在 EOF 后面读，通常这将导致程序以不可预算的方式停止执行。

12.3 过程 WRITE

12.4 过程 WRITELN

〈写调用〉::=〈写或写行〉{（{〈文件变量〉,}{表达式表}）}

〈写或写行〉::=WRITE|WRITELN

〈表达式表〉::=〈宽度表达式〉{,〈宽度表达式表〉}

〈宽度表达式〉::=〈表达式〉{:〈W 表达式〉}:〈十进表达式〉}}

〈W 表达式〉::=〈表达式〉

〈十进表达式〉::=〈表达式〉

不允许使用执行输入输出的函数作为 WRITE 或 WRITELN 语句的参数，例如，象 GNB 这样的存取子程序。"读程序"的执行（即输出所输入的文件内容）过程中文件指针被修改。

PASCAL/MT+ 系统不能写一个换行字符给 CON：，这是因为 CP/M 以这种方式处理控制台和磁盘文件的行终结符。如果想写换行，可使用 TRM：设备。

WRITE 或 WRITELN 允许格式化的数据输出。如前所述，这种格式化由写语句中的域宽度（〈W 表达式〉）和小数点位置（〈dec 表达式〉）所规定。域宽度包括十进制小数点、符号和指数值所占用的空间数。小数点位置数（只有实数才使用）告诉输出子程序小数部分的位数。所有数据项是右对齐。

实数的补缺输出格式是将该实数以 12 位的指数形式表示，其中五位为小数部分。如果数不能在规定的域宽度内被表达，那么输出总是以指数形式出现。

12.5 其他过程

参见第七章 10.1.1 节

标准过程 READ 和 WRITE 不允许字的输入和输出。READHEX 和 WRITEHXE 是两个新过程，它们允许任何一个或两个字节的变量类型（整数、字符、字节子界、枚举、字等）进行十六进制的 I/O。见第三章第四节中关于 PASCAL/MT+ 扩充部分。

第十三节 程 序

〈程序〉::=〈程序首部〉〈块〉.|

〈模块首部〉

〈标号宣称部分〉

〈常量定义部分〉

〈类型定义部分〉

〈变量宣称部分〉

〈函数及过程宣称部分〉

MODEND.

〈程序首部〉::=PROGRAM〈标识符〉｛（程序参数〉）｝；
〈模块首部〉::=MODULE〈标识符〉；
〈程序参数〉::=〈标识符〉｛，〈标识符〉｝

附录 A PASCAL/MT⁺ 文件输入/输出

1. 文件

文件就是按逻辑排列在同等大小元素中的数据，很类似于一个由指针存取的开关数组。数据的长短和排列方式由用户在他的程序中控制。文件通常被存于辅助存储器中。为了编制文件，假设辅助存储器为磁盘。你可用 PASCAL/MT⁺ 提供的文件操作过程将数据写入文件或从文件中读出数据。文件的数据可按顺序存取（记录 1 在记录 2 之前，记录 2 在记录 3 之前……）或随机存取（先取 3，然后取 1，再取 2……）。

文件名

文件名是指磁盘文件的文件名，即存储介质上的可按目录列表显示出来的名字。在 PASCAL/MT⁺ 中程序文件名是用一字串来表示的。例如："B：TEST.DAT" 是一个带有 'TEST' 文件名和 'DAT' 扩展名的在驱运器 B 上的文件（用文字字串格式表示）。

类型

文件类型规定了各个文件基本单元的大小和格式，基本单元是指一个文件的最小存取单位。例如：一个 INTEGER 类型的文件（2 个 8 位字节）可想象为：

00001000	00000000	00100001	00000001	00000001	00000000
record 0		record 1		record 2	

该文件包含整数 8，33 和 1（在此例子中是按反相存储的）。最小的可检索的单元是两个字节。如果你想把该文件当作非整数文件处理，请看非类型文件或字节文件的说明。文件可以分为标准的 PASCAL 标量类型：布尔型，整型，字符型或实型。文件也可以是结构型：字串，数组和记录。预先定义的类型 TEXT 用于 ASCII 码文件。正文文件与字符型文件相同，只是正文文件有分行的概念，写入正文文件的数被转换成 ASCII 码（可能被格式化）。从它们当中读出的数字被转换成二进制。行的定义是由行结束字符终止的一串字符。行结束字符通常是回车/换行字符。正文文件与字符文件不同，它把字符的压缩数组〔1…N〕或字符的数组（1…N〕（写一个非压缩数组不是 ISO 的标准）和字串作为数据接收。在写操作时，尔布值被转换成 ASCII 码字串"false"或"true"，但反之则不成立。有关类型文件和正文文件的进一步说明，请看操作部分。

一个与文件有关的非 ISO 标准的概念是非类型文件。该文件用于快速输入、输出数据块（整个扇区被读写），而不管文件包含什么样的数据。在第六节将进一步描述这种非类型文件。

文件信息块（FIB）

文件信息块包含运行时程序在磁盘文件上进行文件操作所需的信息。文件信息块包含的信息有：文件名、文件类型、存取类型（读或写）、文件结束和行结束标记、磁盘缓冲区（大

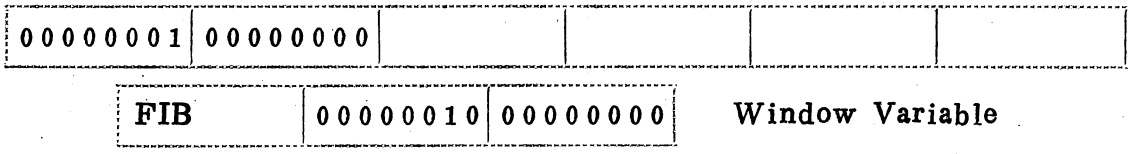
小与一个物理磁盘扇面一样)。有关 FIB 的完整的描述请看你的配给磁盘上 FIBDEF LIB 文件。此外，前面对文件变量的定义也包含 FIB 的进一步信息。

窗口变量或窗口指针

窗口变量是一个与文件基本单元大小相同的缓冲区，它位于 PASCAL/MT+ 中 FIB 旁边。可以这样来理解它：它沿着文件移动，并作为将被读或写的文件的基本单元的一个窗口。因此我们把它当作被存取的文件的基乐单元的指针。它是由 'F' 表示的。F 是文件变量的名字。你若想读文件，就应把被取的元素移到窗口变量的位置上。你若想写数据给文件，数据必须由窗口变量移到文件上。

文件变量

文件变量是由一个文件信息块和一个窗口变量组成的。它是由编译程序分配，并在 PASCAL 程序中被引用的实际数据项。通过一个实例将使你清楚文件变量 FIB 窗口变量是些什么东西。语句 'VAR F : FILF OF INTEGER' 使编译程序生成一个文件变量 F，在数据区占有一个文件信息块和一个窗口变量(2 个字节)来容纳一个 16 位的整数。窗口变量用 F↑表示。假设 I 是同一个程序中的一个整数，其值有 2，还假设文件已经包含了第一个单元中的值 1。见下图：



若要把 I 的内容写给文件，窗口变量必须包含 2(F↑ := I 将 I 的内容输给窗口变量)，其位置必须定在文件的第二个单元上。若执行指令 PUT (F) 数字 2 就被写给文件。

文件结束

必须用某种方式终止文件。大部分操作系统用一个控制字符表示文件结束。当系统阅读该字符时，标准的内部函数 EOF 返回"真"值。另有一些情况：例如数据没有填满文件中最后一个扇区，而有效数据在操作系统发出文件结束信号之前就结束了。在这种情况下，当数据切实结束时，即使你跟踪 EOF 也不会得到'真'值。用户必须使用伪记录作为最后的记录，或将记录数存在另一个文件中。

2. 基本文件操作

示范程序和解释说明了 PASCAL/MT+ 中文件操作过程用法。你将会看到文件怎样被打开、生成、删除、关闭和随机存取。示范程序和解释还说明了类型文件和正文文件的使用方法，介绍了文件状态函数 IORESULT、EOF 和EOLN 及怎样给窗口变量赋值。

图 1 列出了一个名字为 WRITE—READ—FILE—DEMO 的程序。它在磁盘上生成一个类型文件，将数据写给文件，关闭文件，然后重新打开文件，将数据读回。完成上述功能的过程有：ASSIGN, REWRITE, RESET, IORESUIT, PUT, GET 和 CLOSE。用 WRITE 在终端上显示结果。输出工作在 WRITEFILE 中完成。输入工作在 READFILE 中完成。文件的生成、打开、关闭工作在程序的主体中完成。行 37、43、46和49 上的WRITELN 语句将传递给它们的字串写给补缺OUTPUT 文件 (即控制台)。后面我们将在本节的正文文件里讨论此过程和 READLN。

首先要注意 OUTFILE 的宣称形式。OUTFILE 属于字符文件类型，在类型宣称章节中

(行 3,4) 被定义为 **FILE OF CHAR**。所以这样定义是因为文件作为一个参数转递给 **WRITEFILE** 过程和 **READFILE** 过程。一个参数表不能宣称一个新的类型。例如下列的参数宣称在 **PASCAL** 中是非法的，因为参数表中只允许用类型标识符：

```
PROCEDURE WRITEFILE (VAR F:FILE OF CHAR);
```

```
 1   0  PROGRAM WRITE_READ_FILE_DEMO;
 2   0
 3   0  TYPE
 4   1    CHFILE=FILE OF CHAR;
 5   1  VAR
 6   1    OUTFILE   :CHFILE;
 7   1    RESULT    :INTEGER;
 8   1    FILENAME:STRING[16];
 9   1
10   1    PROCEDURE WRITEFILE(VAR F:CHFILE);
11   1    VAR CH:CHAR;
12   2    BEGIN
13   2      FOR CH:='0' TO'9'DO
14   2        BEGIN
15   3          F^:=CH; {CHR(I+ORD('0'));}
16   3          PUT(F)
17   3        END;
18   2      END;
19   1
20   1      PROCEDURE READFILE(VAR F:CHFILE);
21   1      VAR I:INTEGER;
22   2        CH:CHAR;
23   2      BEGIN
24   2       FOR I:=0 TO 9 DO
25   2        BEGIN
26   3          CH:=F^;
27   3          GET(F);
28   3          WRITELN(CH);
29   3        END;
30   2     END;
31   1
32   1    BEGIN
33   1      FILENAME:='TESE.DAT';
34   1      ASSIGN(OUTFILE, FILENAME);
```

```
35   1     REWRITE(OUTEILE);
36   1     IF IORESULT = 255 THEN
37   1      WRITELN ('Error creating', FILENAME)
38   1     ELSE
39   1      BEGIN
40   2       WRITEFILE(OUTFILE);
41   2       CLOSE(OUTFILE, RESULT);
42   2       IF RESULT = 255 THEN
43   2         WRITELN('Error closing', FILENAME)
44   2       ELSE
45   2         BEGIN
46   3          WRITELN('Successful close of ', FILENANE);
47   3          RESET(OUTFILE);
48   3          IF IORESULT = 255 THEN
49   3           WRITELN('Cannot open', FILENAME)
50   3          ELSE
51   3           READFILE(OUTFILE)
52   3         END;
53   2      END;
54   1   END.
```

图 1　输入和输出文件

PROCEDURE ASSIGN(VAR F : FILE VARIABLE;STR : STRING);

目的：使文件变量 F 与磁盘上一个名字为 STR 的外部文件相连接。

ASSIGN 是行 34 中第一个被执行的文件操作。此过程使文件变量（OUTFILE）与磁盘上一个名为 TEST.DAT 的外部文件相连接。传递给 ASSIGN 的字串被放在文件信息控制块中，名字被注释。传给 ASSIGN 过程的文件变量总是与以名字参数命名的磁盘文件相连接，一直到另一个 ASSIGN 赋给文件变量为止。

PROCEDURE REWRITE (VAR F : FILE VARIABLE);

目的：用 FIB 中的名字在磁盘上生成一个文件（FIB 中的名字由 ASSIGN 语句预先填好，否则为空，一个暂存文件就产生了）。

REWRITE 过程在图 1 中的第 35 行中被调用。执行此过程就会生成一个名字由 F 指定在 FIB 中的文件。任何以这个名字命名的现在存文件都被删除，所以千万不要在一个含有有用数据的磁盘上使用 REWRITE。在此例中，磁盘文件被命名为 'TES DAT' 并被定在补缺磁盘上（因为在传送给 ASSIGN 的文件名字串中没有指定其他的磁盘）。

若预先没有执行 ASSIGN，FIB 的名字字段中为空。这样，一个名为 'DASTMDOO.$$$' 的暂存文件就生成了。暂存文件通常用作暂存在执行此过程后不需要的数据。名字中最后两位数字被用来给每一个暂存文件确定一个唯一的名字。这样用户可拥有多达 100 个暂存文件。

EOF 函数和 EOLN 函数之所以返回 '真' 值是因为 OUTFILE 是一个输出文件。打开

OUTFILE 只是为了按顺序写。OUTFILE 准备将接收数据放到其第一个单元中。参见图 4 中的文本并考虑执行 REWRITE 后的随机存取。若操作不成功，IORESULT 函数返回值 255（参见第 36 行）。

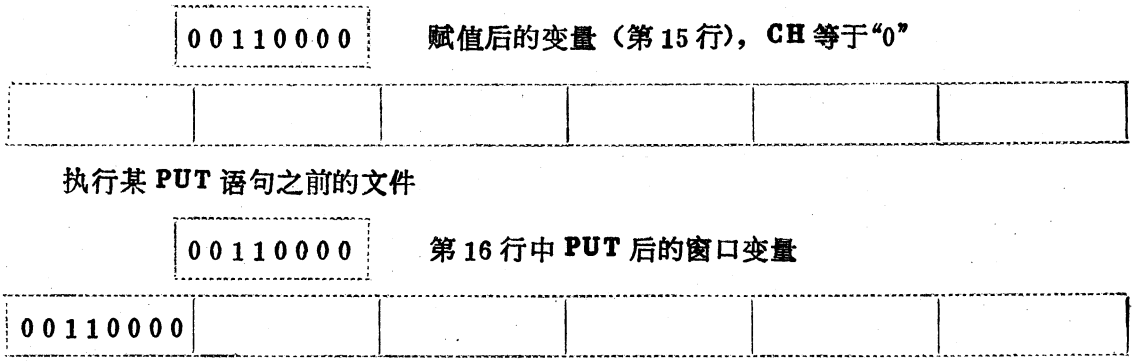
FUNCTION IORESULT : INTEGER;

目的：返回指示文件操作状态的整数值。此函数值在任何 I/O 操作后置入，并可在任何时候被检查。注意在图 1 中，此值在第 36、42 和 46 行中的每一个文件操作都要被调用。此值可用来在文件没有按照规定运行的情况下终止程序。注意不能 'WRITE (IORESULT)'，因为 IORESULT 在每一个 I/O 操作后被重新置 '0'。第三章描述了 IORESULT 返回值的意义。

PROCEDURE PUT (VAR F : FILE VARIABLE);

目的：将与 F 有关联的窗口变量的内容传送到文件中下一个变量记录中去。

过程 WRITEFILE 开始于图 1 中第九行，将字符"0"到"9"写给 TEST.DAT 文件。过程 PUT 的作用就是将数据写给文件，正象在第 15 行中一样，在执行一个 PUT 之前，总要赋值给窗口变量。下面是过程示意图：



在图 1 FOR LOOP 中第一个 PUT 被执行后的文件（从 13 行到 17 行）。

PROCEDURE WRITE;

PROCEDURE WRITE(expression,...,expression);

PROCEDURE WRITE (VAR F : FILE VARIABLE, expression,..., expression);

目的：是 'F↑ := data; put(F);' 的缩写。当 F 是一个正文文件时，还将数字转换成 ASCII 码。

表达式可包括变量、串、数组单元、常数和表达式。当没有指定文件变量时，OUTPUT 文件为补缺文件。在 15、16 行上，WRITE 过程对文件进行同样的操作。因为 ISO 标准的需要，所以提供 GET 和 PUT。在有些 PASCAL 文本中，例如 UCSD PASCAL，WRITE 只能变在正文文件上。

PROCEDURE CLOSE (VAR F : FILE VARIABLE; RESULT : INTEGER);

目的：刷新 FIB 中与 F 关联的缓冲区，以把所有的数据都写进磁盘。

从 WRITEFILE 返回后要执行第 41 行语句，在该行中文件被关闭。必须执行 CLOSE 来保证写给 'TEST.DAT' 的数据确实是存到磁盘上去了。要做到这一点，首先写入缓冲区，然后必须以刷新缓冲区来存入数据。操作系统检查关闭是否成功地确定了 RESULT 的返回值。

该返回值与IORESULT返回的值相同，可作为与编译程序的先前文本保持兼容的参数被调用。在这个程序中值255表示关闭文件不成功，其他任何值都表示成功。

PROCEDURE RESET (VAR F：FILE VARIABLE)；

目的：打开现有文件以便进行读操作。将窗口变量移到文件的起始位置。对随机存取操作而言，打开文件可进行读写操作。

RESULT检查以后，在第47行中调用RESET过程。RESET打开现有文件进行读操作，将窗口变量复位到文件的起始位置。将F的第一个单元赋给F^。如果F已经被打开，RESET就调用CLOSE。EOF和EOLN返回'假'值，如果在一个不存在的文件复位，IORESULT就为255了。IORESULT的其他所有的值都表明成功。在示范程序中由RESET过程打开OUTFILE，它便能被阅读了。下面是RESET在第47行中被执行过后文件和窗口变量的图表。

00110000

RESET执行47行后的窗口变量（OUTFILE^）。

00110000	00110001	00110010	00110011	00110100	00110101

注意：对于非控制台类型文件如OUTEILE，RESET包含执行了第一次GET，将窗口变量移向文件的第一个单元（0单元）内容。在控制台文件或非类型文件上，不执行第一次GET。因为用户总是要在其程序执行之前打印一个字符（因为GET正处于等待字符状态）。

PROCEDURE GET (VAR F：FILE VARIABLE)；

目的：将当前可存取的记录传递给窗口变量，再将窗口变量向前移。

RESET成功后，在第51行中调用READFILE。该过程读出对应文件的每个单元（在此例中单元是一个字符），并把读出的单元内容送给屏幕显示。下图描述了第26行和第27行中FOR循环内所发生的一切。

00110000

第26行执行后的窗口变量（OUTFILE）

00110000	001100011	00110010	00110011	00110100	00110101

CH在第26行执行后包含0的ASCII码（00110000）。窗口变量在执行第27行后，被移向前。

00110001

第27行中GE T执行后的窗口变量。

00110000	00110001	00110010	00110011	00110100	0011010···

第28行将CH的内容写给补缺输出文件（即控制台）。程序READFILE在控制台上将"0"到"9"的字符显示成一列。在顺序情况下，RESET执行结束以后，不需要CLOSE，因为文件在磁盘上没做任何改动。如果是随机存取OUTFILE，可能需要调用CLOSE。

PROCEDURE READ (data, data……data)；

PROCEDURE READ (VAR F：FILE VARIABLE, data, data……data)；

目的：当用户将该过程用于非控制台文件时，每读一项数项，需执行"data：=F^；GETCF)"；当用户将该过程用于控制台文件时，则执行"GET(F)；data：=F^"；若F没有

被指定，则 INPUT 为补缺文件。有关其他信息请参看正文文件的章节。READ 与 GET 的赋值和调用完全相同。如果在此例中用 READ 而不用 GET，那么 FOR 循环主体为：

```
FOR I:=0 TO 9 DO
  BEGIN
    READ (CH);
    WRITELN (CH)
  END;
```

3 正文文件

定义

正文文件是被分为行的 ASCII 字符文件。行由一系列字符组成，以一个不可显示的行结束指示字来终止（该指示字通常是回车/换行字符）。正文文件与字符文件相同，只是由正文文件读出的数字或写给正文文件的数字是自动进行转换的。也可以从正文文件中读出一个 STRING，并可将 BOOLEANS、STRINGS 和 DACKED ARRAYS 写给正文文件。正文文件的存取是通过下列过程进行的：

GET 和PUT（它们不进行转换工作）；READ 和 WRITE；READLN 和 WRITELN。

内存中正文文件的格式是一个 FIB 和一个字节的窗口变量。磁盘上的文件看起来与下面的示范文件一样。其中'>'表示回车，'/'表示换行，"*"表示文件结束。

```
+-----------------------------------------------------------------------+
| This is a line>/This is the next line>*/This is the last line>/* |
+-----------------------------------------------------------------------+
```

```
FUNCTION EOLN : BOOLEAN;
FUNCTION EOLN(VAR F : TEXT)BOOLEAN;
```

目的：仅当窗口变量处于行结束字符位置上时，通过返回"真"值指出文件的状态。若没有指定文件，就假设为补缺 INPUT 文件。

当 READ 语句读到一行中最后一个有效字符时，对于磁盘正文文件，函数将返回"真"值。因为对非控制台文件语句一次 READ 的顺序是 'CH:=F^; GET(F)'，所以 READ 语句读完最后一个字符后立刻将窗口变量置到文件结束字符的位置上。这样当 READ 语句读完最后一个字符时，对于非控制台正文文件 EOLN 将返回'真'值。另外，返回的是一个 BLANK 字符，而不是文件结束字符。对控制台文件，上列顺序要颠倒过来（READ 一开始就调用 GET，GET 后面紧接着的是来自窗口变量的赋值）。为此 EOLN 在使用 CONSOLE 文件时，在回车/换行字符被阅读后返回'真'值，而不是在磁盘文件上最后一个字符被读后返回'真'值。

```
FUNCTION EOF;
FUNCTION EOF (VAR F : FILE) : BOOLEAN;
```

目的：仅当窗口变量处于文件结束字符位置上时，返回"真"值，指出文件状态。若没有指定文件，就假设为补缺 INPUT 文件。当文件结束字符被读时，EOF 返回"真"值。对于非控制台文件 EOF 与 EOLN 相类似，因为最后被阅读的字符将 EOF 置为"真值"。对于控制台文件，只有文件结束指示字被输入时 EOF 才为真。在阅读控制台文件时不能读过文件结束字符，否则系统就要出错。同时，在阅读磁盘文件时也不能读过文件结束字符。当 EOF 为真值时，窗口变量送回一个空格。要注意：对于非正文文件，EOF 在有效数结束时，可能不会成为

'真'值，因为数据可能未将文件的最后一个扇区全部填满。

图 2 是一个程序。它将数据写给正文文件再读出，显示在输出装置上。WRITEDATA 过程实际上是将数据写给正文文件。过程 READDATA 被用来索取存在文件中的信息。该程序被分为一个主体和两个过程。将程序分成若干个具有独立功能的模块是很有用的，这使得程序的阅读和调试要容易得多。

第 3 行中宣称文件。注意宣称不能写成"VAR F：=FILE Of TEXT"。TEXT是作为FILE OF CHAR 的特别文本来处理的。

程序从第 25 行调用 ASSIGN 过程作为执行的开始。第 25 行到第 29 行生成一个名叫 TEXT·TST 的文件。

生成文件以后，示范数据在第 31，32 行中被预置。接下来调用第 33 行中的WRITEDATA 过程。WRITEDATA 使用了只与正文文件一起使用的 WRITELN 过程。

```
PROCEDURE  WRITE;
PROCEDURE  WRITELN;
PROCEDURE  WRITELN (expr, expr……expr);
PROCEDURE  WRITELN (F);
PROCEDURE  WRITELN (F, expr, expr……expr);
```

目的：将数据写到与 F 有关的文件中去，结束时，带有一个行结束字符。若没有指定文件，那么表达式就是写给 OUTPUT 文件的。一个没有表达式的 WRITELN仅仅输出回车/换行字符。WRITE 是作为转换时重新定义的而不是代替 PUT 的。

该过程将数据写给已命名的文件，并将文件结束字符置在最后写入的一项数据之后。若没有命名文件，就将数据写给补缺 OUTPUT 文件。数据可以是字母，已命名的常数、整数、实数，子界变量，枚举，布尔量，字串和压缩字符数组；但不可以是记录这一类的结构类型。数字数据被转换成 ASCII 码，字串被当作数组字符处理（字节长度不写给文件）。

输出格式

图 2 中构成 WRITEDATA 主体的 3 行（9、10、11）进行实际文件输出，第 9 行将字串变量 S 带着回车/换行字符写入正文文件 F。第 10 行用四个空格组成的段将 I 中的内容格式化，并将此格式化的数据输出给文件 F。第 11 行中的实数按字母被格式化成由九个空格组成的段，其中四个空格必须在数（十进制）位置的右边，然后将此格式化的数字写给文件F。段格式可以由任何数据类型指定。对非实数只指定段宽，不存在小数点后面格式化的问题。数据在段中是右对齐的。如一个数字的有效数位大于 6.5（小数点前面 6 位，后面 5 位），则输出就用指数形式来表示。同样，如段宽太小而不能表示数字，则也用指数形式来表示。格式输出的进一步介绍，请参阅 PASCAL 教科书，并亲自练习一下。

如果用下列方式写出WRITEDATA 的主体也可得到同样的结果。

```
WRITELN (F,S);
WRITELN (F,I：4,45.6789：9：4);
```

回到程序主体执行第 34 行。如 CLOSE 执行成功，第 39 行中 RESET 就打开文件 F（该文件仍然与磁盘的 'TEXT·TST' 相关联），并将窗口变量移到起始位置准备读文件 F 的数据。在 RESET 以后，调用 READDATA 程序读出 'TEXT·TST' 中的信息，并将该信息显示在控制台上。

```
 1   0  PROGRAM TEXTIO_DEMO;
 2   0
 3   0  VAR F : TEXT;
 4   1      I : INTEGER;
 5   1      S : STRING;
 6   1
 7   1    PROCEDURE WRITEDATA;
 8   1    BEGIN
 9   2      WRITELN(F, S);
10   2      WRITE (F,I : 4);
11   2      WRITELN(F, 45.6789 : 9 : 4);
12   2    END;
13   1
14   1    PROCEDURE READDATA;
15   1    VAR R : REAL;
16   2    BEGIN
17   2      READLN(F,S);
18   2      READ(F,I);
19   2      READ(F, R);
20   2      WRITELN(s);
21   2      WRITELN(I : 4,' ', R : 9 : 4);
22   2    END;
23   1
24   1  BEGIN
25   1    ASSIGN(F, 'TEXT.TST');
26   1    REWRITE(F);
27   1    IF IORESULT = 255 THEN
28   1      WRITELN('Error creating')
29   1    ELSE
30   1      BEGIN
31   2        I : = 35;
32   2        S : = 'THIS IS A STRING';
33   2        WRITEDATA;
34   2        CLOSE(F,I);
35   2        IF IORESULT = 255 THEN
36   2          WRITELN('Error closing')
37   2        ELSE
38   2          BEGIN
```

```
39    3            RESET(F);
40    3            IF IORESULT = 255 THEN
41    3              WRITELN('Error opening')
42    3            ELSE
43    3              READDATA;
44    3          END;
45    2      END;
46    1  END.
46    0  ------------------------------
46    0  Normal End of Input Reached
```

图 2　正文文件

procedure READ;

procedure READLN;

procedure READLN (F);

procedure READLN (F, variable, variable……variable);

目的：将数据从与 F 有关的文件读出。无论如何，都要读到行结束字符为止。跳过某个不可读的数据，移到下一行开始的位置上。用重定义 READ 来进行实数、布尔代数和整数的转换。

与 WRITELN 一样，READLN 可以把指定文件的变量和接收数据的一些变量作为参数。若指定文件的变量没有规定，就把补缺的 INPUT 文件（键盘）用作输入。参数表内的变量与从文件中读出的数据应是同一类型，但是变量不进行类型检查，这就要用户键入一个与用户文件格式相兼容的参数表。输入时，任何数字都被转换，但丢失了格式。必须用一个空格或回车换行字符将数字相互分开。

READLN 能识别但不能传送行结束字符。它的作用是连续读数据直到遇到行结束字符为止，然后将窗口变量移至下一行的开始位置上。下面是'TEXT • TST' 中的数据；

```
THIS IS A STRING>/
   35   45.6789/*
```

第 1 行中的字串被读以后，必须使用 READ 读整数 35，而不能使用 READLN。若这里使用 READLN，35 会被读出，因为第一个空格终止该数字，然而窗口变量将会经过实数被移到文件结束字符的位置上。此时，用户若试图再读实数，所得到的就是 EOF，因此根本无法知道那儿的实数究竟怎么了。

读字串必须用 READLN，因为它们都是以行结束字符终止的。写给文件的数据若是 'This is A STRING 35>/'，那么返回的 S 值就是包括 ASCII 码35 的一整行。

第 20 和 21 行按照数据在文件中的原格式写给控制台。

执行 READDATA 后，程序就结束了。这时不需要 CLOSE，因为文件数据未被修改过。

打印机输出

如图 3 所示，打印机输出非常简单。如下面程序中，第 5 行中文件变量被宣称为 TEXT 类型。在第 11 行中文件变量被赋给打印机。传给 ASSIGN 的文件名 'LST' 表示 F 将与列表设

备发生关联，这样写给 F 的所有数据就被传送给打印机。调用REWRITE 打开列表设备，准备写数据。注意：这里不需要 CLOSE，因为数据已写好，不必刷新缓冲区。第 23 和 25 行使用标准 PASCAL 格式指令。在第 23 行中 R 的输出格式为：共占 7 个字符，小数点右边占 3 个字符。

```
  1    0   PROGRAM PRINTER;
  2    0   (*WRITE DATA AND TEXT TO THE PRINTER*)
  3    0
  4    0   VAR
  5    1     F:TEXT;
  6    1     I:NTEGER;
  7    1     S:STRING;
  8    1     R:REAL;
  9    1
 10    1   BEGIN
 11    1     ASSIGN(F, 'LST:');
 12    1     REWRITE(F);
 13    1     IF IORESULT=255 THEN
 14    1       WRITELN('Error rewriting file')
 15    1     ELSE
 16    1       BEGIN
 17    2         S:='THIS LINE IS A STRING';
 18    2         I:=55;
 19    2         R:=3.141563;
 20    2         WRITE(F, S);
 21    2         WRITE(F,I);
 22    2         WRITELN(F);
 23    2         WRITELN(F,R:7:3);
 24    2         WRITE(F,I,R);
 25    2         WRITE(F,I:4, R:7:3);
 26    2         WRITELN(F);
 27    2         WRITELN(F,'THIS IS THE END.')
 28    2       END
 29    2   END.
 29    0   --------------------------------
 29    0   Normal End Of Input Reached
```

图 3　打印机输出及数字输出格式

4　随机文件 I/O

本节把过程 SEEKREAD和SEEKWRITE 与PASCAL/MT+随机存取文件放在一起讨论。

示范程序 RANDOM DEMO 说明了这些过程的使用，随机存取文件的宣称、生成和打开。程序中还说明了集合，CASE 语句中的ELSE，EXIT 语句，WRITE，WRITELN，READ 和READLN 的使用。

随机文件只是 PASCAL 的一类型文件。该文件通过 SEEKREAD 和 SEEKWRITE 随机存取。这样任何文件都可以是随机文件。在宣称中没有特殊的语法来区别随机文件和顺列文件。随机存取过程允许用户规定需要存取的相对记录号。随机存取与顺序存取不同，它可以在存取 RECORD 1 之前先存取 RECORD 0，等等。随机存取的记录号可以从 0 到 65536。

图 4a 和图 4b 的示范程序（RANDOM-DEMO）说明了随机文件存取的使用。该程序要么生成、要么使用现有类型为 PERSON 的记录文件。文件中的每一个记录包含两个字串：某人的名字和地址。该程序在第 69 至 80 行中是一个循环。允许用户通过 READRECS 读入记录或通过程序 WRITEREC 写出记录。首先看该程序的主体，然后看 READEC、WRITEREC 和 ERRCHK 在干些什么。

程序从第 59 行开始执行，首先显示一个提示符询问用户是否要生成一个文件或打开现行文件。无论是生成文件，还是打开文件，文件都在第 68 行被 RESET。这样就提出一个重要问题：对于随机文件，在 RESET 以后，它可能需要执行读出数据（用 SEEKREAD），也可能执行写入数据（用SEEKWRITE）。这一点与顺序文件不同（顺序文件在 RESET 后只能被读出）。还有，用 REWRITE 生成的一个新文件可以在数据写入文件以后再由 SEEKREAD 取出。重复循环允许读、写过程继续下去一直到一个 'Q' 输入而终止。若对R（EAD）、W（RITE）或 Q（UIT）选择的话，过程 READRECS 调用 SEEKREAD 来读记录。

```
 1    0   PROGRAM RANDOM_DEMO;
 2    0
 3    0   TYPE
 4    1     PERSON = RECORD
 5    1                NAME : STRING;
 6    1                ADDRESS : STRING;
 7    1              END;
 8    1
 9    1   VAR
10    1     BF : FILE OF PERSON;
11    1     S : STRING;
12    1     I : INTEGER;
13    1     ERROR : BOOLEAN;
14    1     CH : CHAR;
15    1
16    1   EXTERNAL PROCEDURE @HLT;
17    1
18    1   PROCEDURE ERRCHK;
19    1   BEGIN
```

```
20   2   ERROR := TRUE; (* DEFAULT *)
21   2   CASE IORESULT OF
22   2    0: BEGIN WRITELN('Succassful');ERROR := FALSE END;
23   3      1: WRITELN('Reading unwritten data');
24   3      2: WRITELN('CP /M error');
25   3      3: WRITELN('Seeking to unwritten extent');
26   3      4: WRITELN('CP/M error');
27   3      5: WRITELN('Seek past physical end of disk')
28   3    ELSE
29   3      WRITELN('Unrecognizable error code')
30   3    END;
31   2  END
32   1
33   1  PROCEDURE READRECS;
34   1  BEGIN
35   2    WRITE('Record number?');
36   2    READLN(I);
37   2    SEEKREAD(BF,I);
38   2    ERRCHK;
39   2    IF ERROR THEN
40   2      EXIT;
41   2    WRITELN(BF^.NAME,'/',BF^.ADDRESS);
42   2  END;
43   1
```

图 4-a 随机文件的输入和输出

```
44   1    PROCEDURE WRITERECS;
45   1    BEGIN
46   2      WRITE('Name?');
47   2      READLN(S);
48   2      BF^.NAME := S;
49   2      WRITE('Address?');
50   2      READLN(S);
51   2      BF^.ADDRESS := S;
52   2      WRITE('Record number?');
53   2      READLN(I);
54   2      SEEKWRITE(BF,I);
55   2      ERRCHK;
```

```
56   2   END;
57   1
58   1   BEGIN
59   1     WRITE('Create file?');
60   1     READLN(S);
61   1     IF S[1] IN['Y','y']THEN
62   1       BEGIN
63   2         ASSIGN(BF,'B:BIG.FIL');
64   2         REWRITE(BF);
65   2         CLOSE(BF, I);
66   2       END;
67   1     ASSIGN(BF,'B:BIG.BIL');
68   1     RESET(BF);
69   1     REPEAT
70   2       WRITE('R)ead,W)rite or Q)uit?');
71   2       READ(CH);
72   2       WRITELN;
73   2       CASE CH OF
74   2         'R','r':      READRECS;
75   3         'W','w':      WRITERECS;
76   3         'Q','q':      @HLT
77   3       ELSE
78   3        WRITELN('Enter R,W or Q only')
79   3       END
80   2     UNTIL FALSE;
81   1   END.
81   0   ---------------------------------
81   0   Normal End of Input Reached
```

图 4-b　随机文件的输入和输出

PROCEDURE SEEKREAD (VAR F : FILE; RECORD NUMBER : 0..largest word in system);

目的：将 RECORD-NUMBER 记录中的内容传给 F 的窗口变量。利用 CP/M 中的函数调用 33。

文件参数 F 是某个类型文件，可以通过在 RESET 和 REWRITE（如果需要读数据的话）之前的 ASSIGN 语句取得。RECORD-NUMBER 是相对记录。这里 0 为文件开始的记录。检索记录（若存在）被赋给窗口变量缓冲区；此记录若不存在，则是因为请求了超过文件结束的记录。

错误值由 IORESUIT 返回。要注意 ERRCHK 对 IORESUIT 进行检查，看有没有因 CP/M

系统而产生的错误。

READRECS 过程询问记录号，从文件中读出该记录，再将记录直接从窗口变量写给屏幕。第 37 行调用了 SEEKREAD，给出文件名和记录号。很显然，如果记录 0 和记录 2 包含数据，则也可偿试读记录 1，尽管记录 1 不包含任何数据。这样，用户必须对取到的记录进行分析，因为系统对取到没有数据的记录不作为错误来处理。检索到的信息在第 41 行写出。注意：窗口缓冲区是怎样被当作指针（好象宣称了一个指向记录类型的指针）使用的。如果用户想在别的地方存储数据，就要给予文件(在此例中是 TYPE PERSON)同一类型的数据结构赋值。

```
VAR  TEMP : PERSON;……
…TEMT : =BF↑;
```

注意：文件一旦由 SEEKREAD（或 SEEKWRITE）存取，此文件必须关闭。如果要按顺序方法存取时再重新打开。

PROCEDURE SEEKWRITE (VAR F : ANY_FILE; RECORD NUMBER : 0..largest word);

目的：将与 F 有关联的窗口变量的内容放到磁盘文件中 RECORD-NUMBER 指出的相对记录位置上。利用 CP/M 函数调用 34。

第 44 到 56 行中，SEEKWRITE 被用于过程 WRITECS。WRITECS 询问用户用来填满一个记录类型 PERSON（从 46 行到 51 行）的数据及被写的记录号(第 52 行到 53 行)，并调用 SEEKWRITE（第 54 行）将数据写给磁盘。ERRCHK 在第 55 行中被调用，窗口变量在第 48 行到 51 行中被赋值。下面是将数据写给记录 0、1 和 3 后的文件图表：

```
+-----------------+
| Name...         |
| Address...      |  representation of PERSON record
+-----------------+

+-----------------+-----------------+-----------------+-----------------+
| Gremlin B.H.    | Fred Gnome      | Unwritten dat   | MT Monster      |
| Nottingham      | Under the Hill  | Garbage.....    | Front Cover     |
+-----------------+-----------------+-----------------+-----------------+
   Record 0          Record 1          Record 2          Record 3
```

用SEEKWRITE 写成的文件内的记录邻接存储在磁盘上，不考虑记录所占有的扇区号。因此用 SEEKWRITE 生成的文件在 CLOSE 和 RESET 以后可用顺序存取方法存取。

5. 重新定向的 I/O

重新定向的 I/O 是运行时软件包中使用的 get-character 和 put character 子程序的另一种方法，并在用户不想使用正规 CP/M I/O 时特别有用。该动能还有利于数字和字串之间的转换。图 5 中的示范程序说明了这一功能的使用。

WIR 是 put-character 子程序，从第 18 行开始。它写给全局字串 CONV, GETCH。get-character 函数从第 25 行开始。该子程序从全局字串 CONV 获得一个输入字符。

测试程序段从第 39 行开始。前面的语句对 WIR 和GETCH 所需要的变量进行初始化。CONV-ERTING 是一个布尔变量，当 WIR 将一个数字写给 CONV 时为“真”值。CONM 被初始化为一个空串，这样该字串的字节长度为 0。测试变量 I 在第 42 行中被赋值为 2438，并在第 43 行中，通过执行 WRITELN 语句写给控制台。

示范程序的第 44 行为重新定向 I/O 的第一个例子。WIR 的地址被传送给 WRITELN 子

程序，这样，WIR 就被用于取代运行时软件包中的 put-character 子程序。数字 I 由运行时软件包转换成字符。这些字符被传送给 WIR，以便输给字串 CONV。按此方法，I 的内容被转换成字串。必须用 WRITELN 调用 WIR，同时，WIR 把回车字符作为数字结束的信号。

WIR 首先检验它接受的字符是否是换行字符（$0A），如果是换行字符，WIR 就存在。第 12 行检验查明数字转换是否已经开始。若数字转换已经开始，而且转换的字符不是表示数字结束的回车字符，那么新字符被集中到存放字符的 CONV 中去（第 14 行）。如果这是一个新数字，CONV 被置成字串（第 19 行）。在没有字串被传送的情况下，要检查回车字符。若有字符传送，该字符就被集中到字串中。CONVERTING 被置成"真"值。

从 WIR 返回后，I 被赋为 0 值（第 45 行）。这样当 GETCH 子程序被调用时，可以看出，旧值 2438 是从 CONV 中读出的。第 47 行写出 CONV 的内容为 2438。

最后第 48 行中的 GETCH 从 CONV 读到一个数字传给 I。GETCH 继续接收字符一直到 CONV 变空为止，如第 30 行所示。第 32 行表示把 CONV 中第一个字符赋给 GETCH，该字符从 CONV 中删除（第 33 行），这样，在下一次调用 GETCH 时，第一个新的字符被接收。当字串变空时，返回一个空格，系统要用它来表示读操作结束。

```
 1   0  PROGRAM CONV_DEMO;
 2   0
 3   0  VAR
 4   1    I : INTEGER;
 5   1    CONV : STRING;
 6   1    CONVERTING : BOOLEAN;
 7   1
 8   1  PROCEDURE WIR(CH : CHAR);
 9   1  BEGIN
10   2    IF CH = CHR ($0A) THEN (* Done, ignore linefeed *)
11   2      EXIT;
12   2    IF CONVERTING THNE
13   2        IF CH< >CHR ($0D) THEN (*Not at end of string*)
14   2          CONV : = CONCAT (CONV, CH)
15   2        ELSE
16   2          CONVERTING : = FALSE( * reached end - done * )
17   2    ELSE                              ( * First call, new string * )
18   2      BEGIN
19   3        CONV : = '';
20   3        IF CH< >CHR($0D)THEN
21   3          BEGIN
22   4            CONV : = CONCAT(CONV, CH);
23   4            CONVERTING : = THUE
24   4          END
```

```
25   4        END;
26   2    END;
27   1
28   1    FUNCTION GETCH:CHAR;
29   1    BFGIN
30   2      IFLENGTH(CONV)>0 THEN(*Somethingleft toconvert*)
31   2        BEGIN
32   3          GETCH:=CONV[1];
33   3          DELETE(CONV,1,1);
34   3        END
35   3      ELSE
36   2        GETCH:='';          (*Return blank-NO more chars*)
37   2    END;
38   1
39   1    BEGIN                    (*Marn program*)
40   1      CONVERTING:=FALSE;
41   1      CONV:='';
42   1      I:=2438;
43   1      WRITELN('I=',I);
44   1      WRITELN([ADDR(WIR)],I);(*Field width may be given*)
45   1      I:=0
46   1      WRITELN('I=',I);
47   1      WRITELN('CONV=',CONV);
48   1      READ([ADDR(GETCH)],I);(*READLN may notbeused*)
49   1      WRITELN('I=',I);
50   1    END.
50   0    ------------------------------
50   0    Normal End of Input Reached
```

图 5　重新定向的 I/O

6. 文件输入和输出方法

图 6～9 给出了一个例子，列出了一个名叫 TRANSFER 的文件传递的四种不同的执行方法。图 10 是调用传递程序的主程序。图 10 后面附有一列表，列出图 6～10 中所有程序的代码、数据大小和执行速度。

图 6 表示用 BLOCKREAD 和 BLOCKWRITE 操作。这里非类型文件用一个 2K 缓冲区传递数据。

```
PROCEDURE BLOCKREAD(F:FILE VARBLE; BUFFER:ANYTYPE; VAR IOR
                    :INTEGER; SZ, RB:INTEGER);
```

目的：将指定字节数的数据（SZ）传递给缓冲变量，字节始于相对块号（RB）的直接

CP/M磁盘存取，利用函数调用 21。

在 CP/M 上 RB 参数的范围是：-1……64K。如果需要，当 RB 大于 127 时'计算并打开目录入口。若 RB 是-1，就指定为顺序存取，下一个扇区 128-byte 被读。

图 6 中的代码总是用-1表示 RB，并按现行方式发生作用。

SZ 是被读入的字节数。至少128个字节，不能大于缓冲区。第25和28行指出可用 SIZEOF 函数计算 SZ 参数。

```
PROCEDURE BLOCKWRITE (F : FILE VARIABLE; BUFFER : ANYTYPE;
                      VAR IOR : INTEGER; SZ, RB : INTEGER);
```

目的：将 BUFFER 中的数据传送给文件 F。从文件中相对块号（RB）开始，共SZ 个字节使用 CP/M 磁盘存取（函数调用 21）。与在 BLOCKREAD 中一样，当 RB 是-1 时，规定为磁盘顺序存取。

变量'I' 被用来表示 RB 参数。第 29 行中指出，此参数按 16 递增，用一个缓冲区的字节数 2048 除以一个扇区的字节数 128。这就是说每次循环 16 个 128 字节，一个扇区从源文件被读入 BUF，并写到结果文件中去。

所有的数据被读后，RESULT 就为零。如果在下一个记录中没有数据存在，则 RESULT 不为零。当传送的文件大小是 2K 的偶数倍时，程序才开始写操作。例如：若源文件是 9K 字节那么最后 1K 就不会被写。因为 RESULT（第 25 行）继 BLOCKREAD 之后返回非零值。使用 128 字节的缓冲区能保证所有的数据被传递。

图 7 说明用 GNB 和 WNB 子程序执行同一文件的传递过程。此方法看起来好象是一次传递一个字节，但实际上是用了一个 2K 的缓冲区读、写字节。因此，此方法要比 GET 和 PUT 快得多。

```
FUNCTION GNB (F : FILE OF PAOC) : CHAR;
```

目的： PAOC 是一个字符型紧缩数组类型，可选范围为 128 到 4095。提供一种以字节存取的高速传递方法。

因为使用 GNB 的文件是个字符型压缩数组类型的文件，所以在 CP/M-80 上的边界界限条件可能会引起混乱。在这些系统中，运行时的例行程序磁盘文件上的数据读给 FIB 中的一个扇区的缓冲区，然后再把数据传递给窗口变量。若文件中数据的大小不是窗口变量容量的偶数倍，那么，要使窗口变量中的数据有效，就要及早在 FIB 中设置 EOF 标志。当数据全部传递完毕时，GNB 将获得一部分窗口变量的数据，其余都为 hex FF 字节。如果用户用 GNB 来处理 CP/M（8080 或 8086）上的正文文件，他可通过寻找 hex 1A 字符来确定数据是否结束。

```
FUNCTION WNB (F : FILE OF CHAR, CH : CHAR) : BOOLEAN;
```

目的：一次写给 F 一个字节，若在写数据时发生了错误，返回"真"值。

在 CP/M 上，当用 WNB 存取的文件被关闭时，窗口变量还没有满，这时，仍然将整个窗口变量的内容写给文件。在文件的结尾部分可能会包含无用的数据，因此建议在关闭文件之前，用 hex 1A（若文件是 ASCII）或 hex FT（若文件不是 ASCII），将窗口变量刷新一下。

图 8 给出了使用 SEEKREAD 和SEEKWRITE 传递文件的程序，因为这两个过程在前面已讨论过，所以列出此程序仅仅是为了比较。

注意：同 BLOCK I/O 一样，如果原文件中最后一部分数据没有填满扇面（在这种情况

下，表示文件 A 的窗口变量容量是 2K)，IORESULT 返回值 1，表示文件结束，但是不把最后 2 K 缓冲区填满是不会写到结果文件中去的。

图 9 使用 GET 和 PUT 传递文件，比其他任何一种方法都慢。有关 GET 和 PUT 的使用，请看前面的章节。

```
1   0   PROGRAM FILE_TRANSFER;
2   0
3   0   (*-----------------------------------------------*
4   0   (*Transfer file a to file b using BLOCKREAD and
        BLOCKWRITE*
5   0   (*-----------------------------------------------*
6   0
7   0   CONST
8   1     BUFSZ=2047;
9   1   TAPE
10  1     PAOC=ARRAY[1..BUFSZ]OF CHAR;
11  1     FYLE;
12  1
13  1   VAR
14  1     A, B:FYLE;
15  1     NAME:STRING;
16  1     BUF:PAOC;
17  1
18  1   PROCEDURE TRANSFER(VAR SRC:FYLE;VAR DEST:
                              FYLE);
19  1   VAR
20  2     RESULT, I:INTEGER;
21  2     QUIT:BOOLEAN;
22  2   BEGIN
23  2     I:=0;
24  2     REPEAT
25  3       BLOCKREAD(SRC,BUF,RESULT,SIZEOF(BUF),I);
26  3       IF RESULT=0 THEN
27  3        BEGIN
28  4         BLOCKWRITE(DEST,BUF,RESULT,SIZEOF(BUF),I)
29  4         I:=I+SIZEOF(BUF) DIV 128
30  4        END
31  4       ELSE
32  3        QUIT:=TRUE;
```

```
33  3    UNTIL QUIT;
34  2    CLOSE(DEST, RESULT);
35  2    IF RESULT=255 THEN
36  2      WRITELN('Error closing destination file')
37  2  END;
38  1                    (*MAIN PROGRAM IN FIGURE 10 *)
```

图6 用BLOCKREAD和BLOCKWRITE转换文件

```
1   0  PROGRAM FILE_TRANSFER;
2   0
3   0  (*-------------------------------------------------------*)
4   0  (*Transfer file a to file b using GNB and WNB          *)
5   0  (*-------------------------------------------------------*)
6   0
7   0  CONST
8   1    BUFSZ=2047;
9   1  TYPE
10  1    PAOC=ARRAY[1..BUFSZ] OF CHAR;
11  1    TFILE=FILE OF PAOC;
12  1    CHFILE=FILE OF CHAR;
13  1  VAR
14  1    A:TFILE;
15  1    B:CHFILE;
16  1    NAME:STRING;
17  1
18  1  PROCEDURE TRANSFER(VAR SRC:TFILE; VAR DEST:
                                    CHFILE);
19  1  VAR
20  2    CH:GHAR;
21  2    RESULT:INTECER;
22  2    ABORT:BOOLEAN;
23  2  BEGIN
24  2    ABORT:=FALSE;
25  2    WHILE(NOT BCF(SRC) AND (NOT ABORT)DO
26  2       BEGIN
27  3         CH:=GNB(SRC);
28  3         IF WNB(DEST, CH)THEN
29  3           BEGIN
30  4             WRITELN('Error writing character');
```

```
31  4            ABORT := TRUE;
32  4          END;
33  3        END;
34  2      CLOSE(DEST,RESULT);
35  2    IF RESULT=255 THEN
36  2      WRITELN('Error closing')
37  2  END;
38  1                  (*MAIN PROGRAM IN FIGURE 10 *)
```

图7 用GNB和WNB转换文件

```
1   0  PROGRAM FILE_TRANSFER;
2   0
3   0  (*-------------------------------------------------*)
4   0  (*Transfer file a to file b using SEEKREAD and
       SEEKWRITE*)
5   0  (*-------------------------------------------------*)
6   0
7   0  CONST
8   1    BUFSZ=2047;
9   1
10  1  TYPE
11  1    PAOC=ARRAY[0..BUFSZ] OF CHAR;
12  1    TFILE=FILE OF PAOC;
13  1    CHFILE=FILE OF PAOC;
14  1  VAR
15  1    A:TFILE;
16  1    B:TFILE;
17  1    NAME:STRING;
18  1  PROCEDURE TRANSFER(VAR SRC:TFILE;VAR DEST:
                                    TFILE);
19  1  VAR
20  2    CH:CHAR;
21  2    RESULT2,RESULT,I:INTEGER;
22  2    ABORT:BOOLEAN;
23  2  BEGIN
24  2    CH:='A';
25  2    RESULT:=0;
26  2    I:=0;
27  2  WHILE RESULT<>I DO
```

```
28  2      BEGIN
29  3        SEEKREAD(SRC,I);
30  3        RESULT:=IORESULT;
31  3        IF RESULT=0 THEN
32  3          BEGIN
33  4            DEST^:=SRC^;
34  4            SEEKWRITE(DEST,I);
35  4          END;
36  3        I:=I+1;
37  3      END;
38  2
39  2    CLOSE(DEST,RESULT);
40  2    IF RESULT=255 THEN
41  2      WRITELN('Error closing destination file')
42  2  END;
43  1              (*MAIN PROGRAM IN FIGURE 10*)
```

图 8　带有 SEEKREAD 和SEEKWRITE 的文件转换

```
1   0  PROGRAM FILE_TRANSFER;
2   0
3   0  (*-------------------------------------------------------*)
4   0  (*Transfer file a to file b using GET and PUT          *)
5   0  (*-------------------------------------------------------*)
6   0
7   0  TYPE
8   1    CHFILE=FILE OF CHAR;
9   1  VAR
10  1    A,B:CHFILE;
11  1    NAME:STRING;
12  1
13  1  PROCEDURE TRANSFER(VAR SRC:CHFILE;VARDEST:
                                        CHFILE);
14  1  VAR
15  2    RESULT:INTEGER;
16  2  BEGIN
17  2    WHILE NOT EOF(SRC)DO
18  2      BEGIN
19  3        DEST^:=SRC^;
20  3        PUT(DEST);
```

```
21    3          GET(SRC);
22    3        END;
23    2
24    2      CLOSE(DEST,RESULT);
25    2      IF RESULT=255 THEN
26    2        WRITELN('Error closing destination file')
27    2    END;
28    1            (*MAIN PROGRAM IN FIGURE 10*)
```

图 9　带有 GET 和 PUT 的文件转换

```
BEGIN
  WRITE('Source?');
  READLN(NAME);
  ASSIGN(A,NAME);
  RESET(A);
  IF IORESULT=255 THEN
    BEGIN
      WRITELN('Cannot open',NAME);
      EXIT
    END;
  WRITE('Destination?');
  READLN(NAME);
  ASSIGN(B,NAME);
  REWRITE(B);
  IF IORESULT=255 THEN
    BEGIN
      WRITELN('Cannot open,'NAME);
      EXIT
    END;
  TRANSFER(A,B)
END.
```

图 10　文件转换：图 6～9 中的文件主体

下列图表显示了各程序代码和数据的容量及上述各传递文件子程序在不带等待状态的 4MHZZ80 上和在单面单密度 8 寸软盘上的执行速度。容量用十进制字节数表示，速度用秒表示。传送的文件恰好是 8K 字节，对各种编译程序，这些数字各不相同。因此，对于你的文本对应不同的容量和速度（与图中列的都可能不一样），并不表示错误。下面只显示了四种文件存取方法的不同容量和速度。

Transfer Method	BLOCK I/O	GNB/WNB	SEEK I/O	GET/PUT
Compiled Code	520	519	530	477
Compiled Data	2532	2534	4584	482
Total Code	7317	7161	9243	6764
Total Data	3576	3577	5697	1494
Total Size	10893	10738	14940	8258
Speed	7.8	18.4	8.6	35.1

7. 其他有关文件的子程序

示范程序没有提供下面这些子程序，因为这些子程序的用途是很明显的，或若具有前面提到过的子程序所相同的功能。

PROCEDURE OPEN (F : FILE VARIABLE; TITLE; STRING; VAR RESULT : INTEGER);

目的：作用相当于'ASSIGN (F, TITLE); RESETCF';

PROCEDURE CLOSEDEL (F : FILE VARIABLE; VAR RESULT : INTEGER);

目的：关闭文件 F 并删除它。用作暂存文件，与 CLOSE 以后 PURGE 的作用相同。

PROCEDURE PURGE (F : FILE VAEIABLE);

目的：从磁盘上删去与 F 关联的文件。在调用 PURGE 的前面必须执行 ASSIGN，这样含有文件名的 F 所对应的文件控制块就被删除了。在有些操作系统上(不是CP/M60或86)此文件可能在 PURGE 执行以前需要先执行关闭。在这种情况中，CLOSEDEL 将是一个有用的程序。

附录 B　错误信息

递归堆栈溢出：求值堆栈与符号图表的冲突。通过减缩符号表的大小，简化表达式来纠错误信息。

1. 简单类型错误

   自解释。

2. 缺标识符

   自解释。

3. 缺'PROGRAM'

   自解释。

4. 缺')'

   自解释。

5. 缺':'

   可能在 VAR 说明中使用了一个'='。

6. 非法符号（可能在上面一行漏了';'）

   此时所遇的符号为语法不允许。

7. 参数表错误

   参数表说明中有语法错误。

8. 缺'OF'

自解释。

9. 缺'('

自解释。

10. 类型错误

TYPE 说明中有语法错误

11. 缺'['

自解释。

12. 缺']'

自解释。

13. 缺'END'

所有的过程、函数和语法模块必须有一个"END"。检查不匹配的BEGIN/END。

14. 缺';'（可能在上一行中）

表示要求语法分隔符。

15. 缺整数

自解释。

16. 缺'='

在 TYPE 或 CONST 说明中可能使用。

17. 缺'BEGIN'

自解释。

18. 宣称部分中有错误

典型的错误是非法向后调用一个指针说明中的一个类型。

19. 〈段列表〉中的错误

一个记录宣称中有语法错误。

20. 缺'.'

自解释。

21. 缺'*'

自解释。

50. 常数错误

文字常数中的语法错误。还有在使用递归时和不恰当地使 用 INP 和 OUT 时可能会出现常数错误。

51. 缺':='

自解释。

52. 缺'THEN'

自解释。

53. 缺'UNTIL'

可能由于 begin/end 不匹配而产生。

54. 缺'DO'

语法错误。

55. 在 FOR 语句中缺 'TO' 或 'DOWNTO'

自解释。

56. 缺 'IF'

自解释。

57. 缺 'FILE'

可能是 TYPE 说明中的错误。

58. 〈系数〉出错（坏表达式）

表达式中系数有语法错误。

59. 变量错误

表达式中变量有语法错误。

99. 缺 'MODEND'

每一个模块必须以 'MODEND.' 结尾。

101. 标识符宣称了两次

名字已存在于可见符号表中。

102. 低界超过高界

对子程序来说，下界必须<=高界。

103. 标识符使用不当

把一个变量名字当作一个类型使用，或把一个类型当作一个变量使用等等，都可能导致该错误。

104. 未宣称的标识符

指定的标识符不在可见符号表中。

105. 符号非法

在非整数/非实数常数中符号非法。

106. 缺数

该错误是在编译程序检查了数值的所有可能性之后，把表达完全弄错了而造成的。

107. 不兼容的子界类型

（例如：'A'……'Z' 与 0……9 不兼容）

108. 说明这里有些操作对文件不允许

这里不允许对文件进行比较和赋值。

109. 类型决不能是实数

自解释。

110. 〈tagfield〉类型必须是标量或子界

自解释。

111. 与〈tagfield〉部分不兼容

一个 CASE-变体记录中的选择与〈tagfield〉类型不兼容

112. 下标类型决不能是实数。

数组不能宣称用实数定维。

113. 下标类型必须是标量或子界

自解释。

114. 基数类型决不能是实数

一个集合的基数类型可以是标量或子界。

115. 基数类型必须是标量或子界

自解释。

116. 标准过程参数类型中的错误

自解释。

117. 不满足正向调用的要求

从未定义一个正向宣称的指针。

118. 变量宣称中向前调用类型标识符

用户试图宣称一个变量指向一个还未宣称的类型。

119. 为一个自解释的正向宣称过程重新指定参数不正常

自解释。

120. 函数结果必须是标量、子界或指针类型

一个函数已宣称用一个字串或其他非标量类型作为该函数的值，这是不允许的。

121. 不允许文件用值参

文件必须用 VAR 参数来传递。

122. 一个向前宣称的函数结果类型不能被重新指定

自解释。

123. 函数宣称中漏了结果类型

自解释。

125. 标准过程参数类型错误

该错误产生的原因可能是：内部过程调用中没设参数或企图读写指针、枚举类型等等。

126. 参数个数与宣称的不统一

自解释。

127. 非法参数代换

参数类型不完全与对应的形参匹配。

128. 结果类型与宣称的不统一

当赋值给函数结果时，类型必须兼容。

129. 操作数类型冲突

自解释。

130. 表达式是非集合类型

自解释。

131. 只允许等式检查

对不是等式的集合进行比较时发生该错误。

133. 不允许对文件比较

文件控制块不可以被比较，因为文件控制块可能包含许多对用户来说是没有

用的段域。

134. 非法操作数类型

操作数与操作符要求的操作数不匹配。

135. 操作数类型必须是布尔量

对于 AND、OR 和 NOT 操作，操作数必须是布尔量。

136. 集合元素类型必须是标量或子界

自解释。

137. 集合元素类型必须能兼容

自解释。

138. 变量类型不是数组

在一个非数组变量上指定下标。

139. 下标类型与宣称不兼容

一个数组的下标由一个错误类型的下标表达式来表示时，发生此错误。

140. 变量类型不是记录

企图用'dot'格式或'with'语句，对非记录型数据结构进行存取时发生。

141. 变量类型必须是文件或指针

当一个变量后面跟一个↑而不是指针或文件类型时，就会产生该错误。

142. 非法参数结果

自解释。

143. 循环控制变量类型非法

循环控制变量只能是局部非实数标量。

144. 非法表达式类型

在选择语句中作为选择表达式使用的表达式必须是非实数标量。

145. 类型冲突

Case 选择因子与选择表达式不是同一类型。

146. 不允许对文件赋值

自解释。

147. 标量类型与选择表达式不兼容

Case 选择因子与选择表达式不是同一类型。

148. 子界范围必须是标量

自解释。

149. 下标类型必须是整数

自解释。

150. 不允许对标准函数赋值

自解释。

151. 不允许对形参函数赋值

自解释。

152. 在指定记录里没有这样的段

自解释。

153. 同操作中类型错误

自解释。

154. 实参必须是变量

企图将一个表达式作为 VAR 参数传递。

155. 控制变量不能是形参或非局部变量

在 FOR 循环中的控制变量必须是局部变量。

156. 多次定义 Case 表

自解释。

157. 在选择语句中选择太多

在 Case 语句中，当生成的跳转表溢出其界限时，发生此错误。

158. 在这个记录里没有这样的变体

自解释。

159. 不允许实数或字串、变体域

自解释。

160. 前面的宣称是非正向的。

161. 正向宣称重复。

162. 参数范围必须是常数。

163. 宣称漏掉了变体

当使用 NEW/DISPOSE 和一个变体不存在时此错误就会发生。

164. 不允许取代标准过程函数。

165. 重定义的标号

用相同的标号对多个语句置标。

166. 多次宣称的标号

不止一次地宣称同一标号。

167. 未宣称的标号

语句的标号没有宣称。

168. 未定义的标号

不是用一个宣称过的标号对一个语句置标。

169. 基集中的错误。

170. 缺值参。

171. 标准文件被重新宣称。

172. 没有定义外部文件。

174. 缺 PASCAL 函数或过程

自解释。

183. 在嵌套层上不允许作外部宣称

自解释。

187. 打开库失败

自解释。

191. 没有局部宣称的文件

除了在一个程序或一个模块的全局变量段中，文件不能宣称，因为文件必须被静态地分配。

193. 此操作没有足够的空间

自解释。

194. 注释必须在程序的顶部。

201. 实数中的错误：缺数字

自解释。

202. 字串长度不能超过源行。

203. 整数常数超过范围

整数常数范围是-32768 到 32767

250. 嵌套标识符超界

编译时只允许 15 个嵌套层，这里包括 WITH 和过程嵌套。

251. 嵌套过程或函数太多

执行只允许 15 个嵌套层。当一个编译模块中的子程序超过 80 个时，就会出现该错误。

253. 过程（或程序主体）太长

一个程序生成的代码超过了内部程序缓冲区。减缩该程序的长度，再试试看（参看 CPU 应用介绍）。

259. 表达式太复杂

用户表达式太复杂（即编译一个表达式需要太多的递归调用），用户可用临时变量来简化编译。

397. 一个过程中有太多的 FOR 或 WITH 语句

在一个独立的程序中只允许 16 个 FOR 和/或 WITH 语句（仅对递归方法言）。

398. 执行限制

一般对于那些太大的数组和集合而言。

400. 文本中的非法字符

在一个引证字串外发现一个非 PASCAL 特殊字符。

401. 不期望输入结束

在返回外层之前遇到 'END.'。

402. 写代码文件时出错，没有足够的空间

自解释。

403. 读隐含文件时出错

自解释。

404. 写列表文件时出错，没有足够的空间

自解释。

405. 在独立的过程中不允许调用

自解释。

406. 隐含文件非法

自解释。

407. 符号表溢出。

496. INTEGER 有非法操作数

通常由于一个调用要求在运行时计算地址而引起出错。

497. 关闭代码文件时出错

当 ERL 文件被关闭时，会发生此错误。在结果盘上腾出更多的地方，再试试看。

999. 由于上述错误，编译程序发生混乱。做一些修改再试试看。如你的程序从语法上讲虽然正确，但存在一些语义错误，也可能使编译程序变得混乱。这时，编译程序可能会过早地异常结束。请注意编译中止在哪一行。

附录C 保留字

下面是 PASCAL/MT+ 中的保留字

AND	DOWNTO	FuNCTION	NIL	PROGRAM	TYPE
ARRAY	ELSE	GOTO	NOT	RECORD	uNTIL
BEGIN	END	IF	OF	REPEAT	VAR
CONST	FILE	IN	OR	SET	WHILE
CASE	FOR	LABEL	PACKED	THEN	WITH
DO	FORWARD	MOD	PROCEDURE	TO	

PASCAL/MT+ 还增加了保留字

ABSOLÚTE ExTERNAL

编译校对：张世和、梁洪枢、周琪琪、汪明霓
景 劲、唐晓林、蒋庆培、吴 一
许 斌、俞 鲁、李晓杰、张 渝

LINK-80 用户手册

第一章　LINK 连接编辑程序

LINK 是一个实用程序。它用来把各有关浮动目标模块 (relocatable object modules) 组成为一个准备在 CP/M 或 MP/M 下执行的绝对文件。浮动目标模块可以是两种类型，第一种具有的文件类型 (filetype) 是 REL，它由 PL/I-80、RMAC 或任何浮动目标模块为 Microsoft 格式的语言翻译程序所产生。第二种目标模块具有的文件类型是 IRL，由 CP/M 库程序 LIB 生成。IRL 文件所含信息与 REL 文件相同，另外还包括一个索引，这个索引使大型库文件连接得更快。

在完成连接任务的基础上，LINK 在控制台上打印符号表、未定义符号、内存图 (memory map) 以级使用因子 (the use factor)。内存图给出不同区段 (segmento) 的大小和位置，使用因子表示 LINK 用过的可用内存总量的十六进制百分比。LINK 把符号表写为 SYM 文件，以及使用 CP/M 符号指令调试程序 SID (the CP/M Symbolic Instruction-Debuggen)。为了在 CP/M 或 MP/M 下直接执行，LINK 还建立一个 COM 或 PRL 文件。

第一节　LINK 操作

调用 LINK 时，打入

LINK filename 1 {,filename 2,…,filename N}

其中，filename 1,…，filename N 是被连接的目标模块名。当不指定文件类型 (filetype) 时，LINK 假设为 REL。LINK 将产生两种文件：filename 1·COM 和 filename 1·SYM。如果要为这个 COM 和 SYM 文件指定其他的文件名，则可以用如下命令行：

LINK newfilename=filename 1 {,filename 2,…,filename N}

在连接 PL/I 程序时，LINK 将在补缺盘上自动检索运行时 (run-time) 库文件 PLILIB.IRL 以及 PL/I 程序使用的各子程序。

为连接操作的附加控制提供了一些选择开关，说明如下：

在连接处理期间，LINK 可以在补缺盘上建立多达八个临时文件，这些文件命名为：

XXABS$$$ XXPROG$$$ XXDATA$$$ XXCOMM$$$

YYABS$$$ YYPROG$$$ YYDATA$$$ YYCOMM$$$

当 LINK 正常结束时，这些文件被删除，但当 LINK 因某种错误条件而流产时，这些文件便保留在磁盘上。

第二节　LINK 开关

LINK 开关用来控制 LINK 和执行参数。这些开关用方括号括起紧跟在命令行中的一个或多个文件名后面，并用逗号分开。例：

LINK TEST[L4000],IOMOD,TESTLIB[S,NL,GSTART]

除 S 开关外，所有开关都可以出现在命令行中的任何文件后面。S 开关必须跟在引用 S

开关的文件后面。

2.1 附加内存开关（A）

当 LINK 的内存缓冲区不够时，开关 A 用来为 LINK 提供附加的符号表存储空间，这个开关仅在必要时才能使用。例如，在显示内存溢出错误 MEMORY OVERFLOW 时再使用。因为使用开关 A 使得内部缓冲区转到磁盘上，这样，便使连接处理变得相当慢。

2.2 数据起始开关（D）

开关 D 用来指定数据和公用段的起始地址。如果不用，则 LINK 把数据和公用段放在程序段的后面。开关 D 的格式是 Dnnnn，其中 nnnn 是以十六进制表示的数据起始地址。

2.3 执行开关（G）

当连接的程序不是从该程序段的第一个字节开始执行时，开关 G 用来指定程序执行的起始标号。LINK 将安置一条转移指令来转向指定标号的装入地址。开关 G 的格式是 G〈label〉，其中，label 是要执行的开始标号。

2.4 装入地址开关（L）

装入地址定义为由 LINK 生成的、COM 文件的基地址。通常装配地址为 100H，在标准 CP/M 系统中，这是暂驻程序区的基地址。开关 L 的格式是 Lnnnn，其中，nnnn 为十六进制数表示的装配地址。开关 L 也可以把程序的起始地址设置成 nnnn，另外，还可以用开关 P 来定义。

2.5 内存大小开关（M）

当建立在 MP/M 下执行的 PRL 文件时，为了正确地执行，要说明 PRL 程序所需的附加数据空间，可以用开关 M。开关 M 的格式是 Mnnnn，其中，nnnn 为所需的附加数据空间的总量，用十六进制数表示。

2.6 不列表开关（NL）

开关 NL 用来禁止在控制台上打印符号表。

2.7 不记录符号开关（NR）

开关 NR 用来禁止记录符号表文件。

2.8 输出 COM 文件开关（OC）

开关 OC 命令 LINK 产生 COM 文件。这对 LINK 是补缺条件（即，用户不加该开关时，LINK 自己也选择这个开关——译注）。

2.9 输出 PRL 文件开关（OP）

开关 OP 命令 LINK 产生一个为在 MP/M 下执行的页面浮动 PRL 文件（a page relocatable PRL file），而不是 COM 文件。有关建立 PRL 文件的详细说明见第三节。

2.10 程序起始开关（P）

开关 P 用来指定程序段起始地址。如不使用，LINK 将在装配地址处安放该程序段，装配地址在 100H，除非用开关 L 改变。开关 P 的格式是 Pnnnn，其中，nnnn 是十六进制数表示的程序起始地址。

2.11 问号“？”开关（Q）

在 PL/I 运行时程序库中的符号用问号“?”开头，以避免与用户符号冲突。通常 LINK 禁止打印和记录这些符号。开关 Q 使符号表中的这些符号在控制台上打印，并记录在磁盘上。

2.12 检索开关（S）

开关S表示把它前面的文件当作库文件，LINK将检索该文件，并从中找出已连接模块内引用的，但没有定义的那些符号的模块。

第三节　建立MP/M的PRL文件

汇编语言程序经常含有对基本页面（the base page）中的符号引用，例如BOOT、DFCB、BDOS以及DBUFF。为了在CP/M（或在MP/M下的COM文件）下正确运行，这些符号简单地用等价语句定义：见表1-1。

```
BOOT    EQU     0       ;JUMP TO WARM BOOT
BDOS    EQU     5       ;JUMP TO BDOS ENTRY POINT
DFCB    EQU     5CH     ;DEFAULT FILE CONTROL BLOCK
DBUFF   EQU     80H     ;DEFAULT I/O BUFFER
```

表　1-1

可是，由于PRL文件基本页面本身在装配时可以浮动，这样，LINK就必须知道在基本页面内固定位置处的这些符号也是浮动的，为此，在引用这些符号的模块中，应简单地把它们作为外部量来宣称一下，见表1-2，

```
EXTRN  BOOT, BDOS, DFCB, LBUFF
```

表　1-2

并且在另一个模块中连接。在这个模块中，符号是作为公共量来说明的。定义如下（表1-3）：

```
        PUBLIC  BOOT, BDOS, DFCB, DBUFF
BOOT    EQU     0       ;JUMP TO WARM BOOT
BDOS    EQU     5       ;JUMP TO BDOS ENTRY POINT
DFCB    EQU     5CH     ;DEFAULT FILE CONTROL BLOCK
DBUFF   EQU     80H     ;DEFAULT I/O BUFFER
        END
```

表　1-3

第四节　连接实例

表1-4给出了一个连接的实例。首先该实例程序GRADE.PLI被编译。

```
PL/I-80 V1.0, COMPILATION OF : GRADE
D : Disk Print
L : List Source Program
  NO ERROR(S) IN PASS 1
  NO ERROR(S) IN PASS 2
PL/I-80 V1.0, COMPILATION OF : GRADE
  1 a  0000    average :
```

```
 2 a  0006        proc options (main);
 3 a  0006        /* grade averaging program */
 4 a  0006
 5 c  0006        dcl
 6 c  000D          sysin file,
 7 c  000D          (grade, total, n) fixed;
 8 c  000D
 9 c  000D        on error (1)
10 c  0014          /* conversion */
11 d  0014          begin;
12 e  0017          put skip  list('Bad Value,Try Again.');
13 e  0033          get skip;
14 e  0044          go to retry;
15 d  0047          end;
16 d  0047
17 c  0047        on endfile  (sysin)
18 d  004F          begin;
19 d  0052          if n^=0 then n
20 e  005B            put skip list
21 e  008A              ('Average is', total/n);
22 e  008A          stop;
23 d  008D          end;
24 d  008D
25 c  008D        put skip list
26 c  00A9          ('Type a List of Grades, End with Ctl-z');
27 c  00A9        total=0;
28 c  00AF        n=0;
29 c  00B9
30 c  00B9         retry:
31 c  00B9        put skip;
32 c  00CA          do while('1'b);
33 c  00CA          get list(grade);
34 c  00E2          total=total+grade;
35 c  00ED          n=n+1;
36 c  00F7          end;
37 a  00F7        end average;
CODE SIZE=00F7
DATA AREA=004C
```

表 1-4

然后由 LINK 建立一个 COM 文件，LINK 自动为 GRADE 使用的子程序检索 PL/I 运行时程序库 PLILIB.IRL。开关 Q 使符号表列表清单（以及 SYM 文件）中包括了从 PLILIB.IRL 中取出的符号。符号表后面的内存图给出每个程序段的长度以及所分配的单元位置。使用因子 49 表示 49H%，或者说，至少有四分之一的内存可用空间为 LINK 用过。（见表 1-5）

（表 1-6 是执行 GRADE.COM 程序时在控制台显示的信息——译注）

```
B>link grade[q]
LINK V0.4
AVERAG  0100  /SYSIN/  1B77  ?START  1A08  ?ONCOP  18AE
?SYSPR  02C5  ?SKPOP   0430  ?SLCTS  1367  ?PNCOP  01FD
?QIOOP  1987  ?SYSIN   02C1  ?ID22N  13B3  ?QICOP  127E
?PNVOP  0221  ?STOPX   1B19  ?RECOV  1468  ?GNVOP  07D5
?QCIOP  11FB  /?FILAT/ 1B9C  /?FPB/  1BA5  ?PNBOP  01F7
?/PNCPR 04CF  ?IS22N   13F9  ?SIOOP  02CA  ?SIOPR  02E8
?FPBST/ 1BD3  /SYSPRI/ 1BE6  ?OIOOP  05A7  ?FPBIO  0758
?OIOPR  05C6  ?BSL16   131C  ?SIGNA  1626  ?SKPPR  0439
?GNCPR  094F  ?WRBYT   0E36  ?PAGOP  07C7  ?NSTOP  1322
?SMVCM  1390  ?SJSVM   132D  ?SSCFS  137A  ?OB08I  11E7
?OPNFI  0D13  /?FMTS/  1C0E  ?FPBOU  19DB  ?FPBIN  1993
?GNVPR  0812  ?RDBYT   0E23  ?RDBUF  0E5C  ?WRBUF  0E7F
?CLOSE  0F68  ?GETKY   0F99  ?SETKY  0FBF  ?PATH   0F4C
?BDOS   0005  ?DFCB0   005C  ?DFCB1  006C  ?DBUFF  0080
?ALLOP  14D2  ?FREOP   1568  ?ADDIO  1A64  ?SUBIO  1A7B
?WRCHR  19F1  ?RFSIZ   10C4  ?RRFCB  1136  ?RWFCB  113B
?QB16I  11EA  ?IN20    13F1  ?CNVER  1400  ?BSL08  1316
?SJSCM  132F  ?SJSTS   1341  ?SLVTS  1365  ?SMCCM  1394
?ID22   13CB  ?IN20N   13F1  ?ZEROD  1420  ?IS22   13F9
/?CONSP/ 1C16 ?OFCOP   14B2  ?RSBLK  1437  ?RECLS  1E79
?ERMSG  1B34  ?BEGIN   1E77  /?ONCOD/ 1C37 ?SIGOP  1616
?STACK  1E71  ?ONCPC   1948  ?REVOP  1903  /?CNCOL/ 1C3A
?BOOT   0000  ?CMEM    1B77  ?DMEM   1E7B
ABSOLUTE  0000
CODE  SIZE  1A77 (0100-1B76)
DATA  SIZE  023F (1C3C-1E7A)
COMMON  SIZE  00C5 (1B77-1C3B)
USE  FACTOR  49
```

表 1-5

```
A>b : grade
Type a List of Grades, End with Ctl-z
50, 75, 25
^z

Average is        50
End of Execution
A>b : grade
Type a List of Grades, End with Ctl-z
50
75
zot, 66
Bad Value, Try Again.
25
^z
Average is        50
End of Execution
A>b : grade
Type a List of Grades, End with Ctl-z
^z
End of Execution
```

表 1-6

第五节 错误信息

CANNOT CLOSE：输出文件不能关闭。软盘可能写保护。

COMMON ERROR：选择了一个无定义的公用块。

DIRECTORY FULL：没有目录空间给输出文件或中间文件使用。

DISK READ ERROR：文件不能正确地读。

DISK WRITE ERROR：文件不能正确地写。很可能由于盘满了。

FILE NAME ERROR：源文件名格式非法。

FIRST COMMON NOT LARGEST：后继的 COMMON 说明，大于指定块的第一个 COMMON 说明。检索正在连接的文件或库中的模块。

INDEX ERROR：某个 IRL 文件的索引有非法信息。

INSUFFICIENT MEMORY：没有足够的内存为 LINK 分配缓冲区。试用开关 A。

INVALID SYNTAX：调用 LINK 的命令行格式不正确。

MAIN MODULE ERROR：又遇到一个主模块。

MEMORY OVERFLOW：没有足够的内存来完成连接操作。试用开关 A。

MULTIPLE DEFINITION：指定的符号在被连接的模块中多次定义。

NO FILE：找不到指定的文件。

OVERLAPPING SEGMENTS：LINK 试图把一段写到已为另一段使用的内存中去。很可能是由于不正确地使用开关 P 及 / 或 D 引起。

UNDEFINED START SYMBOL：由开关 G 所指定的符号，在各连接的模块中都没有定义。

UNDEFINDE SYMBOLS：该信息后面的符号被引用但在各连接的模块中都没有定义。

UNRECOGNIZED ITEM：LINK 扫描(并略过)一个不能识别的位模式(bit pattern)。

第六节 REL文件的格式

REL 文件的信息是以一种位流 (bit stream) 方式来编码的，位流的具体解释如下：

1) 如果第一位是0，那么后面的 8 位按单元计数器的值装配。

2) 如果第一位是 1，那么后面的二位作如下解释：

00 特殊连接项〔见 3〕

01 程序相对。后面 16 位是减去该程序段起始地址值后装配的相对地址值。

10 数据相对。后面 16 位是减去该数据段起始地址值后装配的相对地址值。

11 公用相对。后面 16 位是减去当前选择的公用块起始地址值后装配的相对地址值。

3) 特殊项包括以下几部分：

- 4 位控制段，用来选择 16 种特殊连接项。具体在后面说明。
- 由一个 2 位地址类型段和一个 16 位地址段组成的可选值段。地址类型段解释如下：

  00 绝对

  01 程序相对

  10 数据相对

  11 公用相对
- 由一个 3 位名字计数后面跟以 ASCII 字符（每个字符为 8 位）表示的名字组成的可选名字段。

以下各项仅跟有名字段：

0000 入口符号，此名字段中表示的符号是在该模块中定义的。这样，当现行文件被检索时，该模块将被连接（如在用开关 S 时)。

0001 选择公用块。指示 LINK 使用与名字段中表示的公用块相联系的单元计数器。

0010 程序名，浮动模块的名字。LINK 检查每个模块的第一项，如果不是程序名，则发一个错误提示。

0011 不用

0100 不用

以下各项跟有一个值段和一个名字段。

0101 定义公用块大小。值段决定名字段中说明的公用块所保留的内存总量。分配给指定块的第一个尺寸必须大于或等于任何在其他连结模块后面定义的公用块。

0110 链外部。值段包含了一个链的头，链尾是绝对量0。链的每个元素是用来替换名

字段中说明的外部符号的值。

0111 定义入口点。名字段中的符号值由值段来定义。

1000 不用

以下各项仅跟有一个值段。

1001 外部增加位移量。所有链处理完后，必须把值段的值加到现行段（segment）的后两个字节上。

1010 定义数据大小。值段包含现行模块数据段的字节数。

1011 设置单元计数器。按值段中决定的值调整单位计数器。

1100 链地址。值段中包含用绝对0结尾的链头。链的每个元素用单元计数器的现行值来替换。

1101 定义程序大小。值段包含现行模块程序区段的字节数。

1110 结束模块。定义现行模块的结束。如果值段中的值不是绝对0，则该值用来作为连接后程序的开始地址。文件中下一项将在下一字节的边界端开始。

以下项没有值段或名字段。

1111 文件结束。跟在文件中最后模块的结束模块项之后。

第七节 IRL文件的格式

IRL文件由三部分组成：前导、索引和REL部分。

前导部分包含128字节，定义如下：

字节0 REL部分第一个记录的扩展数。

字节1 REL部分第一个记录的记录数。

字节2 127——目前不用。

索引部分由一组相应于REL部分的入口符号项的各索引项组成。这些索引项的格式如图1-1，

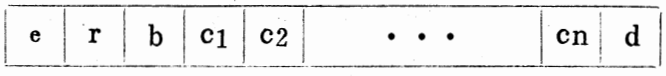


图 1-1

其中：

e为从REL部分的开始到模块开始的扩展位移。

r为从扩展的开始到模块开始的记录位移。

b为从记录的开始到模块开始的字节位移。

c1 – cn为符号名

d为符号结束的终止符（0FEH）

索引部分由C_1 = 0FFH项结束。包含终止项的该记录其余部分是不用的。

REL部分包含浮动目标代码，前节已说明。

第二章　RMAC 浮动宏汇编程序

CP/M 浮动宏汇编程序叫做 RMAC，它是 CP/M 宏汇编（MAC）的修订版本。RMAC产生浮动目标文件（REL），而不是绝对目标文件（HEX）。这种浮动目标文件可以与其他由RMAC或者诸如 PL/I-80 等一些语言翻译程序产生的模块相连接，得到一个准备执行的绝对文件。

RMAC 与 MAC 之间的不同之处在下面加以说明。有关汇编语言和宏功能的完整说明，见 CP/M MAC 宏汇编程序的语言手册和应用指南（CP/M MAC Macro Assembler：Language Manual and Application Guide）。

第一节　RMAC 操作

调用RMAC，打入

RMAC filename.filetype

后面跟可选汇编参数。其中，filename 为文件名，filetype 为文件类型。如不指定文件类型，则RMAC 假设为 ASM。RMAC 产生三种文件：清单文件（PRN），符号文件（SYM）以及浮动目标文件（REL）。除宏展开部分外，源文件中的小写字符，在清单文件中还是小写。

MAC 中汇编参数“H”是用来控制得到 HEX 文件的，在 RMAC 中已被取代为“R”，控制得到 REL 文件。把 REL 文件传给控制台或打印机是不允许的，因为 REL 文件不包含 ASCII 字符。例：

RMAC TEST $PX SB RB

指示 RMAC 汇编 TEST.ASM 文件，将 PRN 文件送到控制台，将符号文件（SYM）和浮动目标文件（REL）存到 B 驱动器。

第二节　表达式

一条语句的操作数段可以由一个复杂算术表达式组成。对表达式有如下限制：

1）在表达式 A + B 中，如果 A 等价于浮动值或外部量，则 B 必须是一个常数；

2）在表达式 A – B 中，如果 A 是外部量，则 B 必须是一个常数；

3）在表达式 A – B 中，如果 A 等价于一个浮动值，那么：

a）B 必须是常数，或

b）B 必须是一个重定位类型与 A 相同的浮动值(即，A、B 都必须在 CSEG、DSEG 或同一个 COMMON 块中出现）；

4）在所有其他的算术、逻辑运算中，两个操作数都必须是绝对量。

如果表达式不遵照以上限制，则会产生表达式错误（‘E’）。

第三节　汇编程序伪指令

为了支持模块的重定位（relocation）和连接，增加了以下汇编程序伪指令：

ASEG　用绝对单元计数器

CSEG　用代码单元计数器

DSEG　用数据单元计数器

COMMON　用公用单元计数器

PUBLIC　符号可以被另一模块所引用

EXTRN　符号在另一模块中定义

NAME　模块名字

伪指令 ASEG、CSEG、DSEG 和 COMMON 把程序模块分成绝对量、代码、数据和公用块区段。在连接时，这些区段根据需要可以在内存中重新安排。PUBLIC 和 EXTRN 伪指令提供了程序模块之间的符号引用。

注意：当符号名字多达 16 个字符时，在 PUBLIC、EXTRN 和 COMMON 语句中的所有符号的前 6 个字符必须是单义的，因为在目标模块中符号只截取前 6 个字符。

3.1　ASEG伪指令

ASEG语句的格式为

label　ASEG　；其中 label 是标号指示汇编程序采用绝对单元计数器，除非有其它说明。ASEG后面语句的物理内存单元，在汇编时由绝对单元计数器决定。这个计数器的补缺值是0，也可以通过在 ASEG语句以后的 ORG 语句重新设置别的值。

3.2　CSEG伪指令

CSEG语言的格式为

label　CSEG

指示汇编程序采用代码单元计数器，除非有其他说明。当 RMAC 开始汇编时，这是补缺条件。CSEG后面语句的物理内存单元在连接时决定。

3.3　DSEG伪指令

DSEG语句的格式为

label　DSEG

指示汇编程序用数据单元计数器，除非有其他说明。DSEG后面语句的物理内存单元在连接时决定。

3.4　COMMON 伪指令

COMMON 语句的格式为

COMMON /identifier/

其中，identifier 为标识符。指示汇编程序用公用单元计数器，除非有其它说明。COMMON 语句后面语句的物理内存单元在连接时决定。

3.5　PUBLIC 伪指令

PUBLIC 语句的格式为

PUBLIC label {,label,…, label}

其中，每个标号 label 都在该程序中定义。出现在 PUBLIC 语句中的标号，可以为用 LINK-80 连接的其他程序所引用。

3.6 EXTRN 伪指令

EXTRN 语句的格式为

EXTRN label {, label, …, label}

在EXTRN 语句中出现的标号 label 可以引用，并且在被汇编的程序中，不必定义这些标号。它们引用由 PUBLIC 宣称的其他程序的标号。

3.7 NAME伪指令

NAME语句的格式为

NAME 'text string'

NAME语句是可选的。它用来为 RMAC 产生浮动目标模块特定的名字，如果不用 NAME语句，则源文件的文件名用作目标模块的名字。

第三章　LIB 程序库

LIB 的功能是处理库，库是由许多浮动目标模块组成的文件。LIB 能把一组 REL 文件联成一个库，建立有索引的库文件（IRL），从库中选择模块，并且可以从库中打印模块名和 PUBLIC 的标号。

第一节　LIB 操作

调用 LIB，打入

LIB filename = filename 1, …, filename N

该命令将根据文件 filename 1.REL, …, filename N.REL 建立一个叫做 filename.REL的库。如文件类型没有，LIB 假设为 REL。

文件名后面可以跟有用括弧括起的一组模块名。在处理 LIB 功能时，只包括指定的模块。否则，包括该文件的所有模块。例：

LIB TEST = A(A_1, A_2), B, C(C_1-C_4, C_6)

这一命令将建立一个 TEST.REL 文件，它包括 A.REL 文件中的模块 A_1 和 A_2、文件B.REL 的全部模块以及文件 C.REL中的 C_1、C_2、C_3、C_4 和 C_6 模块。

在 LIB 命令行中，可以同时加几个可选择的开关。这些开关用方括号括起，并放在 LIB 命令行中第一个文件名之后。这些开关是：

I　建立索引库文件（IRL）

M　打印模块名

P　打印模块名和 PUBLIC

例：

LIB TEST = A, B, C

建立由 A.REL, B.REL 和 C.REL组成的文件 TEST.REL。

LIB TEST = TEST, D

把 D.REL扩充到 TEST.REL 文件的尾部。

LIB TEST〔I〕

根据 TEST.REL 建立一个索引库文件 TEST.IRL。

LIB TEST〔I〕= A, B, C, D

除不建立TEST.REL文件外，完成的功能与前面的 LIB 例一样。

LIB TEST〔P〕

打印 TEST.REL 中所有模块的名字以及 PUBILC 语句中的标号。

第二节　错误信息

CANNOT CLOSE：输出文件不能关闭。软盘可能写保护。

DIRECTORY FULL：没有输出文件的目录空间。

DISK READ ERROR：文件不能正确地读。

DISK WRITE ERROR：文件不能正确地写。很可能由于盘满了。

FILE NAME ERROR：源文件名格式非法。

NO FILE：找不到指定的文件。

NO MODULE：找不到指定的模块。

SYNTAX ERROR：调用 LIB 所使用的命令行格式不正确。

第四章　数据表示法和接口约定

本章说明各种 Digital Research 语言处理程序所采用的内存划分，以便于程序员能正确地处理汇编语言子程序与高级语言程序及 PL/I-80 运行时子程序库之间的接口关系。为了使程序员编制的程序与语言处理程序方便地联接，本章还给出一组标准子程序接口的约定。

第一节　数据元素的表示法

下面给出数据项的内部存储表示法。

1.1　指针、入口以及标号变量

提供对内存地址读写的变量，用两个相邻的字节来存储。内存中第一个字节存放低位字节，指针、入口和标号数据项如图4-1所示：

图4-1

其中，“LS”表示地址的低半字，“MS”表示地址的高半字。注意，MS 是“页面地址”，每个存储页面为 256 字节；LS 是页内地址。

1.2　定点二进制数据格式

简单的单字节和双字节符号整数值以定点二进制格式存放。两种方式的使用，取决于数据项的精度。精度为 1～7 的定点二进制值，以单字节值存放，而精度为 8～15 的数据项以全字单元（双字节）存储。

正如其他 8080、8085 以及 Z80 项一样，多字节存储的最低有效字节是内存的第一个字节所有定点二进制数都用 2 的补码形式表示。单字节值的允许范围是 - 128 至 + 127，全字的值在 - 32768 至 + 32767 范围内。数值0，1，－1 的表示如图4-2所示：

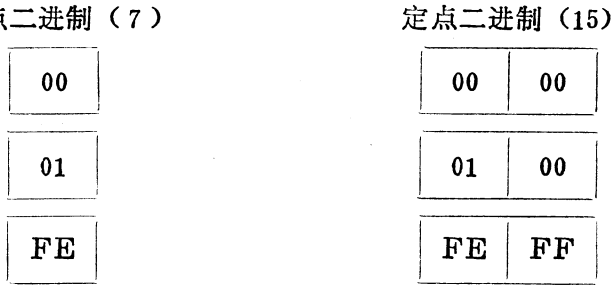


图　4-2

图中，每一方格表示一个字节，低字节在高字节的左面。

1.3　位数据表示法

位串数据象上面表示的定点二进制项一样，以两种形式表示，取决于宣称精度。长度

为1～8的位串以单字节存储，长度为9～16的位串占用一个全字（双字节）。当精度不是恰好为8或16位时，位值调整到字的左面，而“无关紧要”的位调整到右面。一个字的值的最低有效字节存放在内存的第一字节。位串常数值'1'b，'A0'b4以及'1234'b4，按以下方式存放(图4-3)。

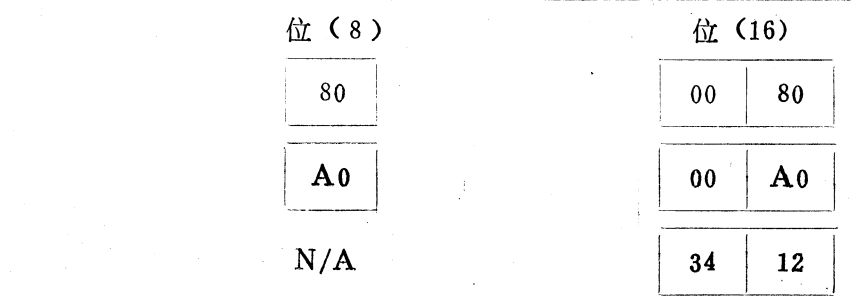


图 4-3

1.4 字符数据表示法

内存中存放两种形式的字符数据，具体取决于宣称。如固定字符串宣称为没有VARYING属性的CHAR(n)，则占有n个连续字节存储区，并且第一个串字符存放在内存的最低字节。用VARYING宣称的字符串，以字串长度作前缀，范围从0至254。用于CHAR(n) VARYING的保留区长度为n+1。注意，不论哪种情况，n不能超过254。字串常数'Walla Walla Wash'以CHAR(20)固定字符串方式存放时，如图4-4所示：

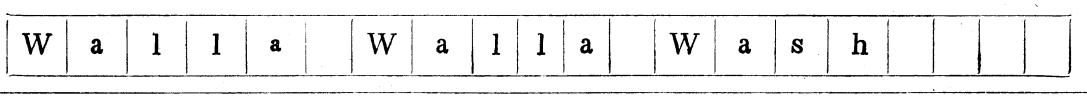


图 4-4

同样这个字串存放在CHAR(20) VARYING数据区时，如图4-5所示：

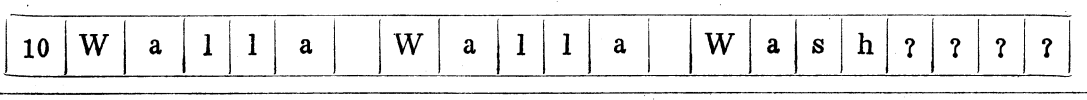


图 4-5

其中，“10”是字串长度（十六进制数），“？”表示无定义字符位置。

1.5 定点十进制数据表示法

十进制数据项以BCD形式存放，采用9的求补数据表示法。最低有效BCD对，存放在内存第一个字节中，并为符号位保留一个BCD位。正数具有0符号，而负数在最高符号位位置上有一个9。一个十进制数所占有的字节个数取决于它的宣称精度。给定一个精度为P的十进制数，所保留的字节个数为(P+2)/2的整数部分。其中P的变化范围在1至15之间，这样，一个十进制数最小占1个字节，最大占8个字节。给定一个精度5的十进制数域，数12345和－2的具体表示如图4-6所示：

图 4-6

1.6 浮点二进制表示法

浮点二进制数存放在四个相邻字节单元，与宣称精度无关。在内存中先存放24位尾数，后面跟8位阶码。按照数据存储约定，尾数的最低有效字节存放在内存第一字节。对非零数，应使尾数的最高有效位为"1"，以使浮点数是规格化的。阶码字节为00表示尾数为零。对非零值，因为尾数的最高有效位必须是"1"，这个位的位置由尾数的符号取代。为了用81表示阶码"1"，而7F表示阶码－1，二进制阶码字节以80（十六进制）为基准。浮点二进制值1.5的具体表示如图4-7所示

00	00	40	81

图 4-7

注意，上图中尾数的位流形式如下：

0100 0000 0000 0000 0000 0000

表示尾数符号位为正。假定把最高位值"1"，则得到尾数位流：

1100 0000 0000 0000 0000 0000

由于阶码81的基准是80，二进制阶码为1，故二进制值为

1.100 0000 0000 0000 0000 0000

或等价于十进制数1.5。

1.7 文件常数表示法

PL/I-80程序中，每个文件常数占32个相邻字节，另外还跟有0至14个字节的可变长度段。文件常数段是所有表示法的从属，并随时可能改变，不另行加注。

第二节 集中存储划分

PL/I-80数据项在内存中是连续存放的，没有空字节。存放的位数据总是没有调整好的。数组按主行顺序存放，这样，第一个下标量执行得最慢，最后的下标量执行得最快。语句RMAC COMMON用来共享PL/I-80程序中用EXTRNAL外部属性宣称的数据。下述（表4-1）中PL/I-80程序是供用EXTRNAL属性的例子：

```
declare
    a (10) bit(8) external,
    1 b external,
      2 c bit(8),
      2 d fixed binary(15),
      2 e (0:2,0:1) fixed;
```

表 4-1

下述（表4-2）RMAC COMMON区共享包含上面宣称程序的数据区。

```
        common /a/
x:      ds      1
        common /b/
c:      ds      1
d:      ds      2
e00:    ds      2
e01:    ds      2
e10:    ds      2
e11:    ds      2
e20:    ds      2
e21:    ds      2
```

表 4-2

其中，标号 e00，e01，…，e21 相应于 PL/I-80 下标变量单元 e(0,0)，e(0,1)，…，e(2,1)。

第三节 通用参数传递的约定

高级语言与汇编语言子程序之间的通讯，可以采用下面描述的 PL/I-80 通用参数传递机构来完成。具体讲，就是根据 PL/I-80 或汇编语言子程序入口，HL 寄存器给出那个入口指针的向量地址，返回时，带回实际参数。举例说明如下（图 4-8）：

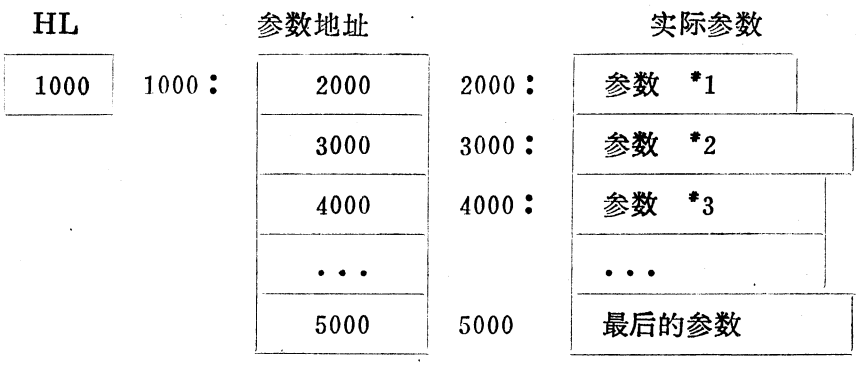


图 4-8

其中参数的个数以及参数长度和类型由调用程序和被调用子程序之间的协定隐含地决定。

为了举例说明，试考虑这样一种情况：设某个 PL/I-80 程序，使用浮点数做除法运算，其中每个除数都是 2 的权。又设：这个除法出现在速度临界状态的闭环中，因此需要用汇编语言子程序来完成这个除法运算。在这个除法运算中，汇编语言子程序只是简单地用每个除数的 2 的权，减去浮点数的二进制阶码，从而有效地实现了除法运算，避免了做一般除法运算要做的分离以及重新合并结果的辅助操作。可是在除法运算中，汇编语言子程序会产生下溢出（underflow），因此，当这种情况发生时，汇编语言子程序必须能够发出 UNDERFLOW 下溢出信号。

实现这种功能的程序由表 4-3、表 4-4 列出，表 4-5 是执行的结果。表 4-3 是测试除法运算的程序 DTEST。外部入口 DIV2 是做除法的汇编语言子程序，第 4 行定义并带有两个参数：fixed(7)和 floot 浮点二进制值，第 9 行的每次循环中，测试值 100 放入"f"，并且被传送给第 10 行的 DIV2 子程序。每次调用 DIV2，f 的值就改变为 f/(2**i)，并用 PUT 语句打印。在调用处，DIV2 接收两个地址的清单，这相应于在计算中使用的两个参数 i 和 f。

```
PL/I-80 V1.0, COMPILATION OF : DTEST
L : List Source Program
   NO ERROR(S) IN PASS 1
   NO ERROR(S) IN PASS 2
PL/I-80 V1.0, COMPILATION OF : DTEST
   1 a 0000 dtest :
   2 a 0006     proc options(main);
   3 c 0006     dcl
   4 c 0006         div2 entry(fixed(7),float),
   5 c 0006         i fixed(7),
   6 c 0006         f float;
   7 c 0006
   8 c 0006         doi =0 by 1;
   9 c 000A         f=100;
  10 c 0015         call div2(i,f);
  11 c 001B         put skip list('100/2 ** ',i,' =',f);
  12 c 0063         end;
  13 a 0063     end dtest;
CODE SIZE = 0063
DATA AREA = 0018
```

表 4-3

称为 DIV2 的汇编语言子程序，在表 4-4 中列出。在上面的入口 (div2) 中，i 的值装入累加器，HL 寄存器置成指向输入浮点数的阶码段的指针。如果阶码为零，DIV2 立即返回，因为结果值为零。否则，子程序在标号 "dby2" 处循环，这时，一边进行循环计数，一边对阶码按 2 的权递减，直到循环计数为零。在计数过程中，如果阶码达到零，则产生 UNDERFLOW 信号 (通过调用"? signal")。

DIV2 中 "? signal" 的调用表明汇编语言使用哪一个通用接口参数。?signal 子程序是 PL/I-80 子程序库 (PLILIB.IRL) 中的一部分。HL 寄存器被置成信号参数表指针，记为 "siglst"。返回时，信号参数表是一个指向信号代码"sigcode"、信号子代码"sigsub"、文件名指示字"sigfil"(这里不用)以及最后一个参数辅助信息 "sigaux" 的四地址向量。当错误发生时，辅助信息用来向操作员提供附加信息。信号子程序打印信息，直至字串长度减完(这里为 32) 或者在字串中遇到一个二进制数 00为止。

```
                        public  div2
                        extrn   ?signal
                ;       entry:
                ;               p1 >fixed(7) power of two
                ;               p2->floating point number
                ;       exit:
                ;               p1->(unchanged)
                ;               p2->p2/(* p1)
                div2:                   ; HL=.low(.p1)
0000 5E                 mov     e, m    ; low(.p1)
0001 23                 inx     h       ; HL=.high(.p1)
0002 56                 mov     d, m    ; DE=.p1
0003 23                 inx     h       ; HL=.low(p2)
0004 1A                 ldax    d       ; a=p1(power of two)
0005 5E                 mov     e, m    ; low(.p2)
0006 23                 inx     h       ; HL=.high(.p2)
0007 56                 mov     d, m    ; DE=.p2
0008 EB                 xchg            ; HL=.p2
                ;
                ;       A=power of 2, HL=.low byte of fp num
0009 23                 inx     h       ; to middle of mantissa
000A 23                 inx     h       ; to high byte of mantissa
000B 23                 inx     h       ; to exponent byte
000C 34                 inr     m
000D 35                 dcr     m       ; p2 alread zero?
000E C8                 rz              ; return if so
                dby2:   ; divide by two
000F B7                 ora     a       ; counted power of 2 to zero?
0010 C8                 rz              ; return if so
0011 3D                 dcr     a       ; count power of two down
0012 35                 dcr     m       ; count exponent down
0013 C20F00             jnz     dby2    ; loop aqain if no underflow
                ;
                ; underflow occurred, signal underflow condition
0016 210000             lxi     h,siglst; signal parameter list
0019 CD0000             call    ?signal ; signal underflow
001C C9                 ret             ; normally, no return
                ;
                        dseg
```

```
0000  0800      siglst:  dw      sigcod  ; address of signal code
0002  0900               dw      sigsub  ; address of subcde
0004  0A00               dw      sigfil  ; address of file code
0006  0C00               dw      sigaux  ; address of aux message
                ;        end of parameter vector, start of params
0008  03        sigcod:  db      3       ; 03=underflow
0009  80        sigsub:  db      128     ; arbitrary subcode for id
000A  0000      sigfil:  dw      0000    ; no associated file name
000C  0E00      sigaux:  dw      undmsg; 0000 if no aux message
000E  20556E6465undmsa:  db      32,'Underflow in Divide by Two',0
002A                     end
```

表 4-4

```
A>b:dtest
100/2 *          0=1.000000E+02
100/2 *          1=5.000000E+01
100/2 *          2=2.500000E+01
100/2 *          3=1.250000E+01
100/2 *          4=0.625000E+01
100/2 *          5=3.125000E+00
100/2 *          6=1.562500E+00
100/2 *          7=0.781250E+00
100/2 *          8=3.906250E-01
100/2 *          9=1.953125E-01
100/2 *         10=0.976562E-01
100/2 *         11=4.882812E-02
100/2 *         12=2.441406E-02
100/2 *         13=1.220703E-02
100/2 *         14=0.610351E-02
100/2 *         15=3.051757E-03
100/2 *         16=1.525878E-03
100/2 *         17=0.762939E-03
100/2 *         18=3.814697E-04
100/2 *         19=1.907348E-04
100/2 *         20=0.953674E-04
100/2 *         21=4.769371E-0
100/2 *         22=2.384185E
100/2 *         23=1.19・・・E-30
100/2 *         24=0・・・487F-31
```

```
100/2 *          25 =  .540743E-31
   ⋮                      ⋮
100/2 *         112 = 1.925929E-32
100/2 *         113 = 0.962964E-32
100/2 *         114 = 4.814824E-33
100/2 *         115 = 2.407412E-33
100/2 *         116 = 1.203706E-33
100/2 *         117 = 0.601853E-33
100/2 *         118 = 3.009265E-34
100/2 *         119 = 1.504632E-34
100/2 *         120 = 0.752316E-34
100/2 *         121 = 3.761581E-35
100/2 *         122 = 1.880790E-35
100/2 *         123 = 0.940395E-35
100/2 *         124 = 4.701977E-36
100/2 *         125 = 2.350988E-36
100/2 *         126 = 1.175494E-36
100/2 *         127 = 0.587747E-36
100/2 *        -128 = 2.938735E-37
100/2 *        -127 = 1.469367E-37
100/2 *        -126 = 0.734683E-37
100/2 *        -125 = 3.673419E-38
100/2 *        -124 = 1.836709E-38
100/2 *        -123 = 0.918354E-38
100/2 *        -122 = 4.591774E-39
UNDERFLOW (128), Underflow in Divide by Two
Traceback : 017F 011E
End of Execution
```

表 4-5

这个测试程序的输出，显示在汇编语言清单之后（表 4-5）。注意，当循环计数值 i 达到 128 时，i 变为负的。但在 DIV2 子程序处理中，把这个值看成是一个无符号小数，因此，当 i 达到 -123 时，发生下溢出。

第四节　函数的返回值

通过参数表返回值的变更在前一节中已作为一种方案说明了，子程序能够在寄存器中或堆栈上直接产生返回的函数值。本节说明返回数据为函数值的通用约定。

4.1 返回指针、入口以及标号变量

提供对内存地址存取的变量占一个全字值，如前节中描述的那样。在指针、入口和标号变量中，返回值在 HL 寄存器对中。如果返回的是一个能作为 GOTO 操作的目标的标号变量，当控制到达这一标号时，包含这一标号的子程序应把堆栈恢复到正确的级（level）。

4.2 返回定点二进制数据

返回定点二进制数据项的函数，将根据数据的精度，把结果存在 A 寄存器或 HL 寄存器对中。精度为 1～7 的定点二进制数从 A 中返回，精度为 8～15 的，从 HL 中返回。为了使寄存器 A 等于寄存器 L，返回值总是放在 HL 中，并且把 L 的内容拷贝到 A 寄存器中。

4.3 返回位串数据

与定点二进制数据项一样，位串数根据数据项的精度，从 A 寄存器或 HL 寄存器对 中返回。长度 1～8 的位串从 A 中返回，精度 9～16 的项从 HL 中返回。注意，位串在它们的段中是向左对齐的，这样，BIT(1)的“真”值从 A 寄存器中返回为 80（十六进制）。再者，HL 寄存器对中为返回保留的位值，在 A 寄存器中拷贝其高字节，以便在返回时 A 寄存器等于 H 寄存器。

4.4 返回字符数据

不论函数是否具有 VARYING 可变属性，字符数据项在堆栈中返回，A 寄存器为字串长度。

例如，字串

'Walla Walla Wash'

返回时如下图所示（图 4-9）：

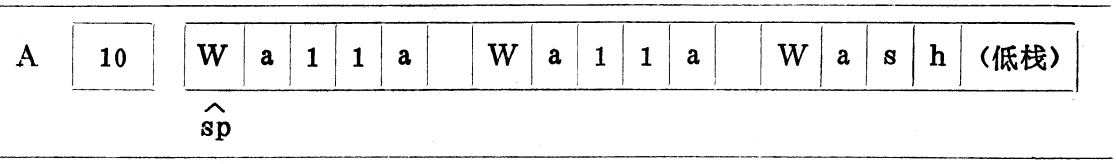


图 4-9

其中，A 寄存器所含字串长度 10（十六进制），栈指针（SP）按字串中第一个字符编址。

4.5 返回定点十进制数据

定点十进制数据总是在堆栈中返回一个 16 位的十进制值（8 个相邻字节）。低位的一对十进制数存在内存的最低字节上（位于堆栈“顶”），高位的一对数存放在内存的最高字 节 上。数按 9 的求补形式表示，最高数字位为符号位，正号记为 0，负号记为 9。例如返回的十 进制数 - 2 表示如下（图 4-10）：

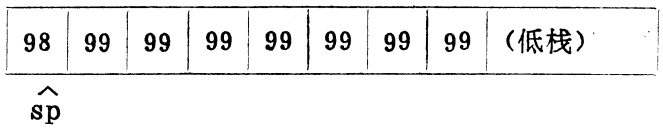


图 4-10

4.6 返回浮点数

返回的浮点数为栈顶的一个四字节序列，与宣 称精 度 无关。尾数的低字节在堆栈的顶

部，接着是中间字节，然后是高字节。第四个字节是这个数的阶码。在堆栈中返回值为 1.5 时的形式如图 4-11：

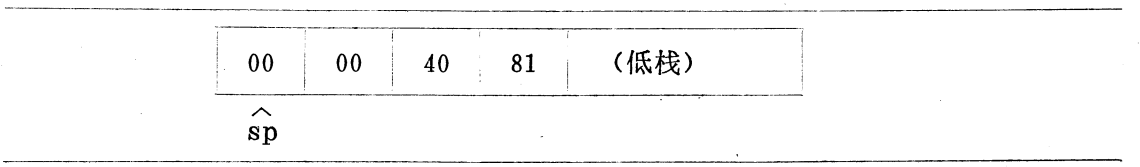


图 4-11

序列为

```
POP    D
POP    B
```

的操作，从堆栈中取出浮点值，B 中是阶码，24 位尾数在 C、D、和 E中。用序列

```
PUSH   B
PUSH   D
```

把结果放回堆栈中。

下面的两份程序清单（表 4-6）、（表 4-7）是说明返回函数值的例子。第一个程序(表 4-6) 叫 FDTEST，与前面的浮点除法测试相似，但所包含的入口定义为 FDIV2，这是个返回结果在堆栈中的汇编语言子程序。FDIV2 子程序的清单如表 4-7 所示，它很类似于以前的DIV2 程序，只是略有一点变化。首先，注意到浮点值被装入 BC、DE 寄存器，以便做不影响输入值的临时拷贝工作。寄存器 B 中的阶码段，按输入计数减少，并在 PC HL执行前，送回堆栈。

```
PL/I-80 V1.0, COMPILATION OF : FDTEST
L : List Source Program
   NO ERROR(S) IN PASS 1
   NO ERROR(S) IN PASS 2
PL/I-80 V1.0, COMPILATION OF : FDTEST
   1 a 0000 dtest :
   2 a 0006     proc options(main);
   3 c 0006     dcl
   4 c 0006          fdiv2 entry(fixed(7),float)
   5 c 0006                returns(float),
   6 c 0006          i fixed(7),
   7 c 0006          f float;
   8 c 0006
   9 c 0006          do i=0 by 1;
  10 c 000A          put skip list(' 100/2 *',i,' =',
  11 c 0055              fdiv2(i,100));
  12 c 0055          end;
  13 a 0055     end dtest;
```

CODE SIZE = 0055
DATA AREA = 0018

表 4-6

```
                        public  fdiv2
                        extrn   ?signal
                ;       entry:
                ;               p1->fixed(7) power of two
                ;               p2->floating point number
                ;       exit:
                ;               p1->(unchanged)
                ;               p2->(unchanged)
                ;       stack:  p2/(2 * p1)
                fdiv2:                  ;HL=.low(.p1)
0000  5E                mov     e,m     ;low(.p1)
0001  23                inx     h       ;HL=.high(.p1)
0002  56                mov     d,m     ;DE=.p1
0003  23                inx     h       ;HL=.low(p2)
0004  1A                ldax    d       ;a=p1 (power of two)
0005  5E                mov     e,m     ;low(.p2)
0006  23                inx     h       ;HL=.high(.p2)
0007  56                mov     d,m     ;DE=.p2
0008  EB                xchg            ;HL=.p2
                ;
                ;       A=power of 2,   HL=.low byte of fp num
0009  5E                mov     e,m     ;E=low mantissa
000A  23                inx     h       ;to middle of mantissa
000B  56                mov     d,m     ;D=middle mantissa
000C  23                inx     h       ;to high byte of mantissa
000D  4E                mov     c,m     ;C=high mantissa
000E  23                inx     h       ;to exponent byte
000F  46                mov     b,m     ;B=exponent
0010  04                inr     b       ;B=00?
0011  05                dcr     b       ;becomes 00 if so
0012  CA2A00            jz      fdret   ;to return from flcat div
                dby2:   ;divide by two
0015  B7                ora     a       ;counted power of 2 to zero?
0016  CA2A00            jz      fdret   ;return if so
0019  3D                dcr     a       ;count power of two down
```

```
001A 05                    dcr   b          ;count exponent down
001B C21500                jnz   dby2       ;loop agani if no underflow
                ;
                ; underflow occurred, signal underflow condition
001E 210000                lxi   h'siglst   ;signal parameter list
0021 CD0000                call  ?signal    ;signal underflow
0024 010000                lxi   b,0        ;clear to zero
0027 110000                lxi   d,0        ;for default return
                ;
002A E1            fdret:  pop   h          ;recall return address
002B C5                    pusn  b          ;save high order fp num
002C D5                    pusn  d          ;save low order fp num
002D E9                    pchl             ;return to calling routine
                ;
                           dseg
0000 0800          siglst: dw    sigcod     ;address of signal code
0002 0900                  dw    sigsub     ;address of subcode
0004 0A00                  dw    sigfil     ;address of file code
0006 0C00                  dw    sigaux     ;address of aux message
                ;          end of parameter vector, start of params
0008 03            sigcod: db    3          ;03 = underflow
0009 80            sigsub: db    128        ;arbitrary subcode for id
000A 0000          sigfil: dw    0000       ;no associated file name
000C 0E00          sigaux: dw    undmsg     ;0000 if no aux message
000E 20556E6465undmsg: db    32,'Underflow in Divide by Two',0
002A                       end
```

表 4-7

第五章　PL/I-80 运行时子程序

本部分讨论 PL/I-80 运行时子程序库(PLILIB·IRL)以及面向 CP/M 输入输出的可选子程序。这里给出的说明是把 PL/I-80 作为“系统语言”使用来介绍的，而不是应用语言，因为直接存取的实现取决于 CP/M 允许的功能。注意，这些特点使你的程序更依赖于机器和操作系统。

第一节　堆栈和动态存储子程序

PL/I-80 运行时程序库中，包括一些附属操作功能（implementation-dependent function），提供面向堆栈和动态存储结构的存取。下面讨论的功能，有实例程序说明它们的使用。堆栈放在代码和数据区的上面，下面是动态存储区。堆栈大小的补缺值为 512 字节，但可以在主程序过程开头的OPTION 部分中，用 STACK（n）选择来改变。通常，PL/I-80 动度存储机制有一份所有未分配存储空间的清单。针对每个存储请求，产生一个检索，寻找满足请求大小要求的第一个内存区段。如果找不到，则发出 ERROR (7)条件的信号（在控制台上产生的提示为 Free Space Exhausted）。否则，从自由区取出所请求的区段，并把剩余的部分送还给自由区清单。在 PL/I-80 的版本 1.0 中，存储器的动态分配，仅在进入“递归”过程时显式或隐式“打开”读写磁盘的文件，或在执行 ALLOCATE语句时进行。在任何情况下，不论请求什么尺寸，总按偶字节或全字长来分配。

1.1　TOTWDS 和 MAXWDS 函数

为了在一个实际程序执行的任意给定点上，得到存储可用空间的总量，这个函数是很有用的。TOTWDS（总字数）和 MAXWDS（最大字数）功能可以用来得到这个信息。在调用程序中，这个函数必须作以下的宣称：

```
dcl  totwds  returns (fixed (15));
dcl  totwds  returns (fixed (15));
```

调用时，TOTWDS 子程序扫描自由存储清单并返回该自由清单中总的可用字数。MAXWDS 子程序完成类似功能，但返回自由清单中的最大区段的大小（以双字节为单位）。只要后面指定区段大小的语句 ALLOCATE 不超出 MAXWDS，就不会产生 ERROR（7）、给出错误信号，因为存储区够用，注意，因为 TOTWDS和 MAXWDS 都以字（双字节）为单位计数，这个值能够用 FIXED BINARY (15) 计数器保存。如果在扫描自由内存时，遇到非法连接字（通常是由于下标越界或指针存储操作），TOTWDS 和 MAXWDS 的返回值为 −1。否则，返回值为非负整数。

1.2　ALLWDS 子程序

PL/I-80运行时程序库内有一个叫做ALLWDS的子程序库，在控制动态分配大小时很有用。这个子程序库在调用程序中必须宣称为：

```
dcl allwds entry (fixed(15))  returns (ptr);
```

ALLWDS 子程序库根据输入参数（以字或双字节为单位）给定的大小分配内存区段。如无可用区段分配，则 ERROR (7) 给

出错误信息。另外，输入值必须是非负整数。ALLWDS 返回一个指向已分配区段的指针。

使用TOTWDS，MAXWDS 和 ALLWDS 功能的例子在5-1中给出。表 5-2 是这个实例的交互操作过程。

1.3 STKSIZ函数

函数 STKSIZ（栈大小）无论何时调用，都返回一个以字节为单位的现行栈大小。这个函数在检查堆栈可能溢出或在程序测试时，在决定最大栈深度时，是很实用的。STKSIZ 函数在调用程序中必须宣称为：

```
dcl stksiz returns[fixed(15)];
```

STKSIZ函数的具体使用，体现在递归 Ackerman测试的程序清单中（表5-3）。在这个例子中，STKSIZ 功能用来测试在递归函数处理时的最大栈深度。表 5-4 是这个例子的交互操作过程。

```
PL/I-80V1.0, COMPILATION OF : ALLTST
L : List Source Program
  NO ERROR(S)  IN PASS 1
  NO ERROR(S)  IN PASS 2
PL/I-80 V1.0, COMPILATION OF : ALLTST
 1  a 0000  alltst :
 2  a 0006    Proc options(main);
 3  a 0006    /*assembly language interface to
 4  a 0006    dynamic storage allocation module*/
 5  c 0006    dcl
 6  c 0006      totwds  returns(fixed(15)),
 7  c 0006      maxwds returns(fixed(15)),
 8  c 0006      allwds  entry(fixed(15)) returns(ptr);
 9  c 0006
10  c 0006    dcl
11  c 0006      allreq fixed(15),
12  c 0006      memptr ptr,
13  c 0006      meminx fixed(15),
14  c 0006      memory (0 : 0) bit(16) based(memptr);
15  c 0006
16  c 0006      do while('1'b);
17  c 0006      put edit(totwds( ),'Total Words Available,
18  c 004F        maxwds( ),'Maximum Seament Size',
19  c 004F        'Allocation Size?')
20  c 004F        (2(skip,f(6),a),skip,a);
21  c 004F      get list(allreg);
22  c 0067      memptr = allwds(allreg);
```

```
 23  c 0070        put edit('Allocated',allreg,
 24  c 00B2          'Words at',unspec(memptr))
 25  c 00B2          (skip,a,f(6),a,b4);
 26  c 00B2
 27  c 00B2          /*clear memory as example*/
 28  c 00B2          do meminx=0 to allreg-1;
 29  c 00CC         memory(meminx)='0000'b4;
 30  c 00E7          end;
 31  c 00E7       end;
 32  a 00E7     end alltst;
CODE   SIZE=00E7
DATA AREA=0078
```

表 5-1

```
A>B:ALLTST
  25596 Total Words Available
  25596 Maximum.Segment Size
Allocation Size? 0
Allocated  0 Words at 250A
  25594 Total Words Available
  25594 Maximum Segment Size
Allocation Size? 100
Allocated 100 Words at 250E
  25492 Total Words Available
  25492 Maximum Segment Size
Allocation Size? 25000
Allocated 25000 Words at 25DA
    490 Total Words Available
    490 Maximum Segment Size
Allocation size? 490
Allocated 490 Words at E92E
      0 Total Words Available
      0 Maximum Segment Size
Allocation size? 1
ERROR(7),Free Space Exhausted
Traceback:016D
End of Execution
```

表 5-2

```
PL/I-80 V1.0,COMPILATION OF : ACKTST
L : List Source program
  NO ERROR(S) IN PASS 1
  NO ERROR(S) IN PASS 2
PL/I-80 V1.0,COMPILATION OF : ACKTST
 1  a 0000  ack :
 2  a 0006    procedure options(main,stack(2000));
 3  c 0006    dcl
 4  c 0006      (m,n)fixed,
 5  c 0006      (maxm,maxn) fixed,
 6  c 0006      ncalls decimal(6),
 7  c 0006      (curstack,stacksize) fixed,
 8  c 0006      stksiz entry returns(fixed);
 9  c 0006
10  c 0006    put skip list('Type max m,n :');
11  c 0022    get list(maxm,maxn);
12  c 0046      do m=0 to maxm;
13  c 005F        do n=0 to maxn;
14  c 0078        ncalls     =0;
15  c 0088        curstack   =0;
16  c 008E        stacksize  =0;
17  c 0091        put edit
18  c 012F          ('Ack(',m,',',n,')=',ackermann(m,n),
19  c 012F          ncalls,'Calls,',stacksize,'Stack Bytes'
20  c 012F          (skip,a,2(f(2),a),f(6),f(7),a,f(4),a);
21  c 012F        end;
22  c 012F      end;
23  c 012F    stop;
24  c 0132
25  c 0132    ackermann:
26  c 0132      procedure(m,n) returns(fixed) recursive;
27  e 0132      dcl
28  e 015C        (m,n) fixed;
29  e 015C      ncalls=ncalls+1;
30  e 0177      curstack=stksiz( );
31  e 017D      if curstack>stacksize then
32  e 018A        stacksize=curstack;
33  e 0190      if m=0 then
34  e 0199        return(n+1);
```

```
35  e 01A1      if n = 0then
36  e 01AA        return(ackermann(m - 1,1));
37  c 01BB      return(ackermann(m - 1,ackermann(m,n - 1)));
38  c 01DC      end ackermann;
39  a 01DC    end ack;
CODE SIZE = 01DC
DATA AREA = 0082
```

表 5-3

```
A>B : ACKTST
Type max m,n : 6,6
Ack(0,0) =     1          1 Calls,     4 Stack Bytes
Ack(0,1) =     2          1 Calls,     4 Stack Bytes
Ack(0,2) =     3          1 Calls,     4 Stack Bytes
Ack(0,3) =     4          1 Calls,     4 Stack Bytes
Ack(0,4) =     5          1 Calls,     4 stack Bytes
Ack(0,5) =     6          1 Calls,     4 stack Bytes
Ack(0,6) =     7          1 Calls,     4 Stack Bytes
Ack(1,0) =     2          2 Calls,     6 Stack Bytes
Ack(1,1) =     3          4 Calls,     8 Stack Bytes
Ack(1,2) =     4          6 Calls,    10 Stack Bytes
Ack(1,3) =     5          8 Calls,    12 Stack Bytes
Ack(1,4) =     6         10 Calls,    14 Stack Bytes
Ack(1,5) =     7         12 Calls,    16 Stack Bytes
Ack(1,6) =     8         14 Calls,    18 Stack Bytes
Ack(2,0) =     3          5 Calls,    10 Stack Bytes
Ack(2,1) =     5         14 Calls,    14 Stack Bytes
Ack(2,2) =     7         27 Calls,    18 Stack Bytes
Ack(2,3) =     9         44 Calls,    22 Stack Bytes
Ack(2,4) =    11         65 Calls,    26 Stack Bytes
Ack(2,5) =    13         90 Calls,    30 Stack Bytes
Ack(2,6) =    15        119 Calls,    34 Stack Bytes
Ack(3,0) =     5         15 Calls,    16 Stack Bytes
Ack(3,1) =    13        106 Calls,    32 Stack Bytes
Ack(3,2) =    29        541 Calls,    64 Stack Bytes
Ack(3,3) =    61       2432 Calls,   128 Stack Bytes
Ack(3,4) =   125      10307 Calls,   256 Stack Bytes
Ack(3,5) =
```

表 5-4

第二节　PL/I-80 运行时子程序入口

标准 PL/I-80 运行时子程序入口列表如下：

name	parameters		result	comment or definition		
im22n	DE	HL	HL	word * word integer multiply		
id22n	DE	HL	HL	word/word integer divide		
is22n	DE	HL	HL	word - word integer subtract		
in20n	HL		HL	- word		
fl40m	HL		ST	fp load	from M(HL)	to stack
fx44s	ST	HL	M(HL)	fp xfer	from stack	to M(HL)
fx44m	DE	HL	M(HL)	fp xfer	from M(HL)	to M(DE)
fa44s	ST	ST	ST	fp add	stack + stack	to stack
fa44m	DE	HL	ST	fp add	M(DE) + M(HL)	to stack
fa44l	ST	HL	ST	fp add	stack + M(HL)	to stack
fa44r	HL	ST	ST	fp add	M(HL) + stack	to stack
fs44s	ST	ST	ST	fp sub	stack - stack	to stack
fs44m	DE	HL	ST	fp sub	M(DE) - M(HL)	to stack
fs44l	ST	HL	ST	fp sub	stack - M(HL)	to stack
fs44r	HL	ST	ST	fp sub	M(HL) - stack	to stack
fm44s	ST	ST	ST	fp mul	stack * stack	to stack
fm44m	DE	HL	ST	fp mul	M(DE) * M(HL)	to stack
fm44l	ST	HL	ST	fp mul	stack * M(HL)	to stack
fm44r	HL	ST	ST	fp mul	M(HL) * stack	to stack
fd44s	ST	ST	ST	fp div	stack/STack	to stack
fd44m	DE	HL	ST	fp div	M(DE)/M(HL)	to stack
fd44l	ST	HL	ST	fp div	stack/M(HL)	to stack
fd44r	HL	ST	ST	fp div	M(HL)/STack	to stack
fc44s	ST	ST	ST	fp comp	stack : stack	to stack
fc44m	DE	HL	ST	fp comp	M(DE) : M(HL)	to stack
fc44l	ST	HL	ST	fp comp	stack : M(HL)	to stack
fc44r	HL	ST	ST	fp comp	M(HL) : stack	to stack
fn40s	ST		ST	fp negate stack		
fn40m	HL		ST	fp load from M(HL) and negate		
fe40s	ST		A	float p extract sign from stack		
fe40m	HL		A	float p extract sign from memory		

				1 => positive sign (non zero set)
				0 => zero result (zero flag set)
				-1 => negative sign (minus set)
fmodf	ST	ST	ST	floating point mod(x,y)
fabsf	ST		ST	floating point abs(x)
fmaxf	ST	ST	ST	floating point max(x,y)
fminf	ST	ST	ST	floating point min(x,y)

表 5-5-1

froun	ST A	ST		floating point round(x,k)
ftrnc	ST	ST		floating point trunc(x)
fflor	ST	ST		floating point floor(x)
fceil	ST	ST		floating point ceil(x)
fexop	ST A	ST		fp * k (k pos constant)
ffxop	ST ST	ST		x * y (exp(y * log(x))
bc12n	D	HL	HL	8/16 bit concatenate, where B = length of d, C = mask
bc22n	DE	HL	HL	16/16 bit concatenate, where B = length of d, C = mask
bs116	B	HL	HL	bit shift left 16, size in b
bs108	A	B	A	bit shift left 8, size in b
bst08	A B C	HL	M(HL)	bit substring store bit(8) in A to bit(8) in memory at HL, B = index, C = length
bst16	B C DE	HL	M(HL)	bit substring store bit(16) in DE to bit(16) in memory at HL
bix08	A B D	H	A/HL	bit index, A = source, B = search D = len(source), E = len(search)
bix16	B C DE	HL	A/HL	bit index, B = len(sonrce), C = len(search), DE = source, HL = search
boolf	B DE	HL	HL	bool (x,y,b), B = 4-bit mask x,y operands in DE and HL
ie12n	A		HL	sign extend A to HL
ie10n	A		A	integer extract sign(8-bit)
ie20n	HL		A	integer extract sign(16-bit)
imdop	DE	HL	HL	integer mod(x,y)
iab07	A		A	integer 7 abs(i)
iab1s	HL		HL	integer 15 abs(i)

imaxf	DE		HL	HL	integer max(x,y)
iminf	DE		HL	HL	integer min(x,y)
iroun	HL		A	HL	integer round(i,k)
iexop	HL		A	HL	integer * k (k pos constant)
slvts	HL			A	string load varying to stack A = length of string on return
slcts	A		HL		string load char to stack A = length of char string
ssvfs	A		B	HL	string store varying from stack A = current len, B = max length
sscfs	A		B	HL	string store char from stack
smvvm	A		DE	HL	string move vary to vary in memory A = max target len, DE = source, HL = target
smvcm	A		DE	HL	string move vary to char in memory A = target length
smcvm	A	B	DE	HL	string move char to vary in memory A = max target len, B = sounce len
smccm	A	B	DE	HL	A = target len, B = source len
sjsts	A		ST	ST'	string juxtapose (catenate) stack A = length of left, ST = chars of loft ST' = pushed psw with length of right followed by chars of right
sjscm	A		B	HL	string juxtaposestack with char memory A = stacked len, B = char len, HL = .char

表 5-5-2

sjsvm	A			HL	string juxtapose stack with vary memory
savvm	A		B	HL	string append vary to vary in memory A = char len, B = max target length
sasvm	A		B	HL	string append stack to vary in memory A = stacked length, B = max target length
sacvm	A		B	HL	string append char to vary in memory A = char len, B = max target length
scccm	A	B	DE	HL	string compare char to char in memory A = len right, B = len left, DE = .char left, HL = .char right
sccvm		B	DE	HL	string compare char to vary in memory B = len left, DE = .char, HL = .vary
scvcm	A		DE	HL	string compare vary to char in memory

					A = len right char, DE = .vary, HL = .char
scvvm			DE	HL	string compare vary to vary in memory
					DE = .vary left, HL = .vary right
scscm	A	B		HL	string compare stack to char in memory
					A = len stk, B = len char, HL = .char
scsvm	A			HL	string compare stack to vary in memory
					A = len stk, HL = .vary
sccms	A	B		HL	string compare char in mem to stack
					A = len stk, B = len char, HL = .char
scvms	A			HL	string compare vary in mem to stack
					A = len stk, HL = .vary
scsts	A				string compare stack to stack
					A = len right element on stack,
					ST is stack right string, next is
					pushed psw with len left string,
					followed by left string, result:
					sign value & cond if l<r,
					zero value & cond if l=r,
					pos value & cond if l>=r,
					nzer value & cond if l>r.
cs2ad	A		E	HL	char substr (ex, ei) address
					A = length, E = ei, HL = ex
					A = result length on return
cs3ad	A	C	E	HL	char substr(ex, ei, el)address
					C = el
					A = result length on return
vs2ad			E	HL	vary substr(ex, ei)address
					E = ei, HL = ex
					A = result length on return
vs3ad		C	E	HL	vary substr(ex, ei, el)address
					C = el
					A = result length on return
cxccm	A	B	DE	A/HL	str index char to char in memory
					A = len right, B = len left,
					DE = .char left, HL = .char right
cxcvm		B	DE	A/HL	str index char to vary in memory
					B = len left, DE = .char, HL = .vary
cxvcm	A		DE	A/HL	str index vary to char in memory
					A = len right char, DE = .vary, HL = .char

cxvvm			DE	A/HL	str index vary to vary in memory DE = .vary left, HL = .vary right
cxscm	A	B			A/HL str index stack to char in memory A = len stk, B = len char, HL = .char

表 5-5-3

cxsvm	A			A/HL	str index stack to vary in memory A = len stk, HL = .vary
cxcms	A	B		A/HL	str index char in mem te stack A = len stk, B = len char, HL = .char
cxvms	A			A/HL	str index vary in mem to stack A = len stk, HL = .vary
cxsts	A				str index stack to stack A = len right element on stack, ST is stack right string, next is pushed psw with len left string, followed by left string, result: A/HL = 0 if right not found in left, otherwise index returned
verop	A	ST	ST	A/A/HL	verify (s, c), A = len (c), st has chars (c), len (s), chars (s)
colop				A/ST	collate ( ), A = 128, stack has
x12op	A	ST	ST	A/ST	translate (s, t), A = len (t), stack has chars (t), s
x13op	A	ST	ST ST	A/ST	translate (s, t, x) A = len (x), stack has chars (x), t, s 0, 1, ..., 127 (ascii chars)
dldop	A		HL	ST	decimal load to stack, A = prec
dasop	A		ST	HL	decimal assign, stack to memory
dadop	ST		ST	ST	decimal add to stack
dsuop	ST		ST	ST	decimal subtract to stack
dngop	ST			ST	decimal negate to stack
dcmop	ST			A	decimal compare operator
dexop	ST		ST	ST	decimal exponentiate to stack
dmuop	ST		ST	ST	decimal multiply to stack
ddvop	ST		ST	ST	decimal divide to stack
dsiop	ST			A	decimal sign extract
dmodf	ST		ST	ST	decimal mod (x, y)
dabsf	ST			ST	decimal abs (x)

dmaxf	ST	ST	ST	decimal max (x, y)
dminf	ST	ST	ST	decimal min (x, y)
droun	ST	A	ST	decimal round (x, k)
dtrnc	ST		ST	decimal trunc (x)
dflor	ST		ST	decimal floor (x)
dceil	ST		ST	decimal ceil (x)
dexop	ST	A	ST	decimal * k (k pos constant)
qcdop	A B	ST	ST	convert character to decimal A = string length, B = scale ST = character string, returns ST = decimal number
qddsl	A	ST	ST	decimal/decimal left shift A = shift count
qddsr	A	ST	ST	decimal/decimal right shift A = shift count
qicop	A		HL	convert integer to char in stack A = string size, HL = integer value
qvcop	A/ST		A/ST	convert varying to char
qi07d	A		ST	convert fix (7) to decimal
qi15d	HL		ST	convert fix (15) to decimal
qi07f	A		ST	convert fix (7) to float

表 5-5-4

qi15f	HL		ST	convert fix(15) to float
qfi07	ST		A	convert float to fix (7)
qfi15	ST		HL	convert float to fix (15)
qfcss	A	ST	A/ST	convert float-char stack to stack A = target length, ST = fp number
qfcms	A	M(HL)	A/ST	convert float-char memory to stack
qb08c	A	B	ST	convert bit (8) in a, to string in stack, with precision b
qb16c	HL	B	ST	convert bit (16) in HL to string
qb08i	A	B	HL	convert bit (8) in A to fixed with precision B in HL
qb16i	HL	B	HL	convert bit (16) to fixed
qi07b	A	B	A	convert fix (8) <to bit (8) fixed precision in b
qi15b	HL	B	HL	convert fix (<16) to bit (16)

qdi07	ST		A	convert dec in stack to fix (7)
qdi15	ST		HL	convert dec in stack to fix (15)
qciop	A/ST		HL	convert char in stack to integer
qcfop	A/ST		ST	convert char in stack to float
qccop	A	B ST	A/ST	convert char to char on stack A = len (s), B = converted length return A = b, ST trunc or extend
nstop	BC	DE HL	M(HL)	non-computational store, move M(DE) to M(HL) for BC bytes
nc22n	DE	HL	A	double byte non-computational compare : zero flag set if DE = HL, non-zero otherwise
ncomp	BC	DE HL	M(HL)	non-computational compare, M(DE)-M(HL), set flags

表 5-5-5

表 5-5 中，子程序的入口名在左边，接着是输入值寄存器和结果寄存器，右边是解释部分。注意，这份表不包括环境或 I/O 操作，因为这些入口可以从一种版本变换成另一种版本。另外，表中的定义说明只是针对通用信息而言，随时可以改变，不另行说明。寄存器名用大写字母给出，M (r) 表示寄存器对 r 的内存编址，ST 表示堆栈的值。

第三节　面向 CP/M 功能的调用

所有对 CP/M 版本 1 和版本 2 的功能，以及对等价的 MP/M 调用的存取，都是通过 PLIDIO.ASM 中的可选子程序来完成的。PLIDIO.ASM 清单在附录A中，也包括在PL/I-8 软盘上。

PLIDIO.ASM 子程序不是标准 PLILIB.IRL文件的一部分，因为具体的应用可能要求对面向 CP/M 的功能作各种改变。例如：为减少空间的转存操作，为特定环境的接口作相应的改变。注意，如果接口改变，最好也改变一下入口的名字，以免别的程序员读此程序时发生误会。

浮动文件 PLIDIO.RE L，通过用RMAC 汇编这个源程序建立：

```
rmac  plidio  $pz+s
```

($pz+s选择不产生列表和符号文件)。

给定一个 PL/I-80 程序 DIOCOPY.PLI，且它已在磁盘上，DIOCOPY.REL 文件可打入命令：

```
pli diocopy
```

(DIOCOPY 程序清单在附录 C)。这两个程序与 PLILIB.IRL 连接，可打入命令：

```
Link  diocopy, plidio
```

以文件 DIOCPY.COM 为结果的程序，可在 CP/M 下直接执行。

文件 DIOMOD.DCL 是一个包含的标准 PLIDIO 入口宣称的源文件，这使它们在编

译时，可以用“include”语句

```
% include  'x; diomod.dcl'
```

将其方便地拷贝到源程序中。其中，可选项“x：”是磁盘驱动器前缀符，表示包含DIOMOD.DCL文件的驱动器名（从A：到P：）。如果，DIOMOD.DCL文件与PLI源文件在同一盘上，则不需要驱动器前缀。DIOMOD.DCL文件的内容表示如下（表5-6）：

```
dcl
    memptr   entry              returns (ptr),
    memsiz   entry              returns (fixed (15)),
    memwds   entry              returns (fixed (15)),
    dfcb0    entry              returns (ptr),
    dfcb1    entry              returns (ptr),
    dbuff    entry              returns (ptr),
    reboot   entry,
    recon    entry              returns (char (1)),
    wrcon    entry              (char (1)),
    rdrdr    entry              returns (char (1)),
    wrpun    entry              (char (1)),
    wrlst    entry              (char (1)),
    coninp   entry              returns (char (1)),
    conout   entry              (char (1)),
    rdstat   entry              returns (bit (1)),
    getio    entry              returns (bit (8)),
    setio    entry              (bit (8)),
    wrstr    entry              (ptr),
    rdbuf    entry              (ptr),
    break    entry              returns (bit (1)),
    vers     entry              returns (bit (16)),
    reset    entry,
    select   entry              (fixed (7)),
    open     entry    (ptr)     returns (fixed (7)),
    close    entry    (ptr)     returns (fixed (7)),
    sear     entry    (ptr)     returns (fixed (7)),
    searn    entry              returns (fixed (7)),
    delete   entry    (ptr),
    rdseg    entry    (ptr)     returns (fixed (7)),
    wrseg    entry    (ptr)     returns (fixed (7)),
    make     entry    (ptr)     returns (fixed (7)),
    rename   entry    (ptr),
```

```
logvec    entry              returns (bit (16)),
curdsk    entry              returns (fixed (7)),
setdma    entry              (ptr),
allvec    entry              returns (ptr),
wpdisk    entry,
rovec     entry              returns (bit (16)),
filatt    entry              (ptr),
getdpb    entry              returns (ptr),
getusr    entry              returns (fixed (7)),
setusr    entry    (fixed (7)),
rdran     entry    (ptr)  returns (fixed (7)),
wrran     entry    (ptr)  returns (fixed (7)),
filsiz    entry    (ptr),
setrec    entry    (ptr),
resdrv    entry              (bit (16)),
wrranz    entry    (ptr)  returns (fixed (7));
```

表 5-6

附录 复盖与文件定位的控制

本附录介绍版本在1.0以后的LINK-80和LIB-80中汇集的几个附加特点，包括处理运行时复盖的扩充，以及源文件、中间文件和目标文件定位的控制。包括自动PL/I-80程序库检索“请求项”的使用和新命令行错误报告格式的说明。还包括了附加的LIB-80工具，为删除或替换子程序库中各模块提供方便。

1.0 复盖

LINK可以用来产生一个简单的的树复盖结构，如图1：

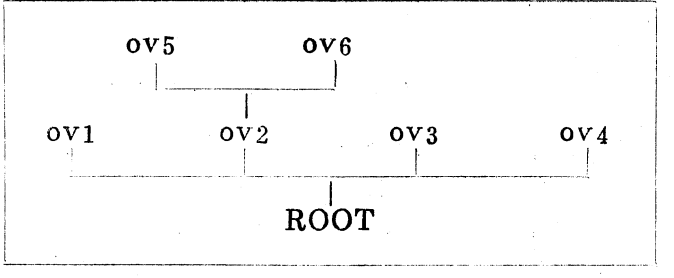


图 1

除产生ROOT.COM和ROOT.SYM文件外，LINK能为命令行中指定的每个复盖产生一个OVL文件和一个SYM文件。OVL文件由一个包含装配地址和复盖长度的256字节的前导，以及跟在前导后面的绝对目标代码组成。一个复盖的起始点是它下面的模块的最高地址，在“树”上刚好达到下一个128字节的边界。PL/I程序的堆栈和自由空间被定位在连

接的最高复盖的顶端，刚好达到下一个128字节的边界。这个地址在整个LINK完成时被写到控制台上，并且被修补到单元'?MEMRY'的引导模块中。SYM文件只包括那些低层树中模块未被宣称的符号。

当用PL/I-80和LINK产生一个复盖系统时，必须遵守以下限制：

每个复盖有一个进入该复盖的入口。该入口由复盖管理程序假设为复盖的基(装入地址)。

不允许一个模块对树上的高层模块入口作向上引用(upward references)，这与所描述的复盖主入口不同。对树上的低层模块入口或根模块作向下引用(downward references)是允许的。

复盖不是浮动的，所以根模块必须是COM文件。

一个模块中宣称的公用块（用PL/I外部宣称）不可由树中高层模块来初始化。任何这类企图LINK都将忽略。

复盖可以嵌套的深度为5层。

定位在80H的补缺缓冲区，由复盖管理程序使用，这样，用户程序就不能依靠这个缓冲区中存放的数据。

1.1 PL/I程序中使用的复盖

在PL/1程序中，有两种方法使用复盖。第一种方法很直接方便，且能满足大多数应用。但是，有一定限制：所有复盖必须在补缺盘上，运行时不能决定复盖的名字。第二种方法没有这些限制，但要用稍微复杂的调用序列。

在使用第一种方法时，一个复盖作为被引用模块中的一个入口常数简单宣称。作为一个入口常数，可以有某个参数表中宣称的参数。复盖本身是一个或一组PL/1过程。例如，下列程序是有一个复盖的根模块（表1）：

```
root: procedure options (main);
    declare ovl entry (char (15));
    put skip list ('root');
    call ovl ('overlay 1');
    end root;
```

表 1

复盖OV1.PLI如表2：

```
ovl: procedure (c);
    declare c char (15);
    put skip list (c);
    end ovl;
```

表 2

注意，当参数传递给一个复盖时，程序员要确保参数的数目和类型在调用程序和复盖本身中相同。

为了把这两个程序连接成一个复盖系统，可使用下述连接命令：

LINK ROOT (OV1)

(连接复盖的命令行语法在下一节中详细说明。) 根据这个命令，LINK 将产生四种文件：

ROOT.COM, ROOT.SYM, OV1.OVL 以及 OV1.SYM。

文件 ROOT.COM 执行时，首先在控制台上输出信息 'root'。语句 'call ov1' 使控制转向复盖管理程序。复盖管理程序把补缺驱动器中的文件 OV1.OVL 装在ROOT.COM之上的适当位置，并且把控制转移到这里，以正常方式传递 char(15)参数；然后执行复盖，在控制台上产生信息 'overlay1'，再直接返回到 root.pli 中 'call ov1' 后面的语句，并从那里继续执行。

用这种方法，当复盖管理程序决定的被请求复盖已经在内存中时，在控制转移到这里前，复盖将不能重新装入。关于这第一种复盖方法有几个重要的注意事项：

• 调用入口语句中与复盖相关联的名字，是复盖管理程序装入的 OVL 文件的实际名字，这两个名字必须一致。由于在PL/I80产生的REL文件中，符号各截取到6个字符，所以OVL文件的名字必须限制在6个字符内。

• 指向某复盖入口的名字(过程的名字)不必与调用序列使用的名字一致。为了避免混淆，应该用相同的名字。

• 复盖管理程序只装入补缺驱动器上的复盖(引导模块执行时的驱动器是补缺驱动器)。

• 复盖的名字是固定的。要改变复盖名，只有重新编译的重新连接源程序。

• 不得用非标准 PL/I 语句（程序可以输送给其他系统）。

在有些应用中，复盖有更大的灵活性，例如，有从不同的驱动器中装入复盖的能力，或可在运行时决定某复盖的名字。这些是用第二种方法完成的。

这时，指向复盖管理程序的显式入口必须在 PL/I 程序中宣称如下：

```
declare ?ovlay entry (char(10),fixed(1));
```

第一个参数是字符串，用来指定装入复盖的名字以及标准 CP/M 格式 'd:filename' 中的可选驱动器代码。第二个参数是装入标志。当装入标志为 1 时，复盖管理程序将装入指定的复盖，不管它是否已在内存中。如装入标志为0，则只有当复盖已经不在内存中时才装入。

如果需要，'call ?ovlag' 语句告诉复盖管理程序装入请求的复盖。复盖管理程序返回调用程序，然后调用程序必须完成一个哑元调用，来执行复盖管理程序刚处理的复盖。这允许把一个参数表传给该复盖。

以第一种方法说明的例子为例，上部分描述参见表3

```
root:procedure options (main);
    declare ?ovlay entry (char (10), flxed (1));
    declare dummy entry (char (15);
    declare name char (10);
    put skip list ('root');
    name = 'OV1';
    call ?ovlay (name, 0);
    call dummy ('overlay 1');
    end root;
```

表 3

OV1.PLI与以前相同。运行时，复盖管理程序将从补缺驱动器中装入 OV1.OVL，由于是变量'name'的现行值，所以返回到调用程序(这时是 root)。在这一点上，根据 PL/I-80 参数传递约定，自变量'overlay1'应该建立。'call dummy'把控制转向复盖管理程序，复盖管理程序将简单地把控制转向刚处理过名字的复盖的基地址。当 OV1 结束时，它返回到'call dummy'语句后的语句。注意，当在上例中，'name'用赋值语句置给'OV1'时，复盖名已经能由别的资源中得到的字符串来提供，例如操作员的键盘。在使用第二种复盖技术时，必须遵守几个要点：

• 可以指定某驱动器代码，这样，复盖可以从补缺驱动器以外的驱动器装入。如不指定驱动器，则与方法1说明的一样，用补缺驱动器。

• 因为复盖名用字符串指定（不是用入口符号），所以可达 8 个字符。

• 如果 'call ?ovlay' 后的哑元调用有一些参数，那么，它们必须在数目和类型方面与复盖过程宣称的参数一致。

1.2 在命令行中指定复盖

指定复盖的语法，除每个复盖说明用括弧括起外，与无复盖的连接相同。复盖说明可以有如下几种形式：

```
link  root (ov1)
link  root (ov1, part2, part3)
link  root (ov1 = part1, part2, part3)
```

第一个命令由文件OV1.REL产生文件 OV1.OVL，第二个命令根据 OV1.REL，part 2.REL，part 3.REL产生 OV1.OVL，最后一种情况，OV1.OVL 文件是根据 part 1，part 2，part 3这三个 REL 文件产生的。

注意，左括号表示新复盖说明的开始，也表示它前面组的结束。换言之，下面的命令行非法，并将发出错误信息：

```
LINK  ROOT (OV1) , MOREROOT
```

包括在'树'上任何点的所有文件都必须一起出现，没有插入任何复盖说明。因此，下述命令行合法：

```
LINK  ROOT, MOREROOT (OV1)
```

命令行中的任何文件，可以跟有一些用方括号括起的连接开关，如 LINK-80 操作员指南所描述的那样。注意，复盖说明不能与根模块分开，也不能用逗号隔开，空格可用来改进易读性。

复盖的嵌套在命令行中用嵌套的括号表示。下述命令行中用来描述连接图 1 中描述的复盖系统：

```
LINK ROOT (OV1) [OV2(OV5)(OV6)] (OV3) (OV4)
```

1.3 连接执行实例

在表 4 的连接操作实例中，请注意 OV1 被作为无定义符号标出。LINK 简单地指出 OV1 在现行模块中还没有定义，这样，它被假设为某个复盖的名字，或者指向某个复盖的哑元入口。当连接复盖时，引用复盖的每个入口变量（实际名字或哑元入口）将作为无定义符号表现出来。除这些实际名或哑元复盖入口外，没有无定义符号。

```
A>LINK ROOT(OV1)
LINK1.1
PLILIB RQST ROOT   0100   /SYSIN/ 1A15 /SYSPRI/ 1A3A
UNDEFINED SYMBOLS:
OV1
ABSOLUTE        0000
CODE SIZE       18BC  (0100 -19BB)
DATA SIZE       02A9  (1A90-1D38)
COMMON SIZE     00D4  (19BC-1A8F)
USE FACTOR        4E
LINKING OV1.OVL
PLILIB RQST
ABSOLUTE        0000
CODE SIZE       0024  (1D80 -1DA3)
DATA SIZE       0002  (1DA4-1DA5)
COMMON SIZE     0000
USE FACTOR        09
MODULE TOP      1E00
A>ROOT
root
overlay 1
End of Execution
A>
```

表 4

1.4 运行时错误信息

复盖管理程序可能产生以下错误信息：

FERROR (8) OVER LAY, NOFILE d：filename.OVL

指定文件找不到。

FERROR (9) OVER LAY, DERIVE d：filename.OVL

非法驱动器代码作为参数传给了 ?ovlay。

ERROR (10) OVER LAY, SIZE d：filename.OVL

指定复盖将会在写操作时，超出堆栈及／或在它装入后，超出自由空间。

ERROR (11) OVER LAY, N：STING d：filename.OVL

装入的指定复盖将会超出最大嵌套深度。

ERROR (12) OVER LAY, READ d：filename.OVL

在复盖装入期间，磁盘读错误，也许由过早的EOF所产生。

1.5 其他复盖系统

复盖系统也可以不是树结构，但仍须有一些独立的复盖区，如下图所示（图-2）：

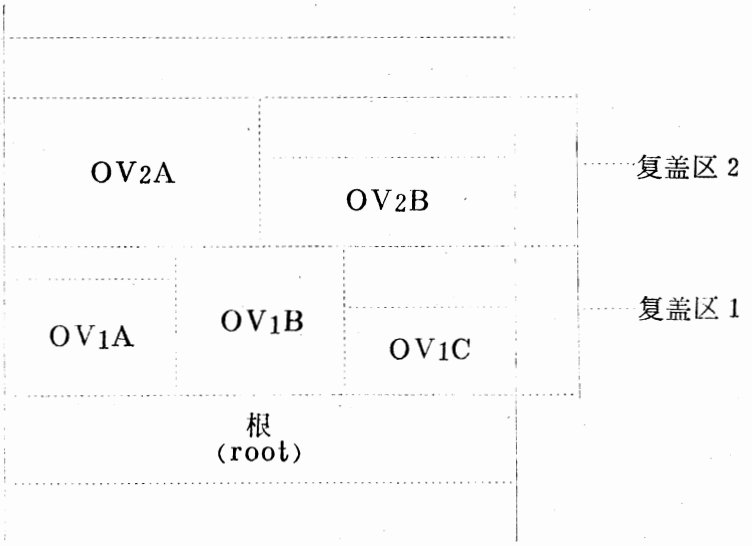


图 2

在这一系统中，根模块可引用任何复盖。某一复盖可以引用根模块的入口，或引用不在同一复盖区的任何模块的主入口。

连接上图所示的复盖系统有几步工作要做。每个复盖区必须完成一个连接，因为在连接下一个更高的复盖区时，必须向 LINK 提供该复盖区顶的地址。例如，命令

LINK ROOT (OV1A)(OV1B)(OV1C)

在复盖区 1 产生三个复盖，并指出模块的顶地址。这个地址是下一个命令

LINK ROOT (OV2A[Lmod top])(OV2B[Lmod top])

的装入地址。这个命令在适当的地址上建立复盖区 2 的复盖。注意，内存的最高复盖区应该最后连接，因为在连接结束时，模块顶地址总是被写进根模块。

在整个系统连接后的有些地方，可以要求只重新连接一个复盖，这个复盖可以不在顶上的复盖区。为了防止含有错误 ? MEMRY 值的根模块产生，可以用 $OZ 开关来做这件事。

保证既没有复盖重叠，又没有企图引用同一复盖区另一个复盖的复盖，是程序员的责任。

1.6 LINK-80 “$” 开关

开关 “$” 用来控制 LINK-80 之下的源设备和目的设备。开关的一般形式是：

$ td

其中，‘t’ 是类型，‘d’ 是驱动器说明符。有五种类型：

C—控制台

I—中间

L—库

O—目标

S—符号

驱动器说明符可以是范围从‘A’到‘P’中的一个字母，对应十六个逻辑驱动器之一；或者是以下的一个特定字符：

X—控制台

Y—打印机

Z—字符桶 (byte bucket)

$cd—控制台

通常出现在控制台上的信息可以定向到列表打印设备($CY) 或禁止显示 ($CY)。一旦$CY 或$CZ指定后，在命令行后面可以用$CX把控制台信息重定向到控制台设备。

$Id—中间

由 LINK 产生的中间文件通常放在补缺驱动器上。$I开关允许用户指定另一个驱动器为 LINK 存放中间文件。

$Ld—库

LINK 通常根据REL文件中的请求项，在补缺盘上检索自动连接的库文件，$L开关命令 LINK 在指定的驱动器上检索这些库文件。

$od—目标

LINK 通常在命令行中第一个 REL文件所在的驱动器上产生目标文件，除非命令中包括显式驱动器的输出文件。$O开关命令 LINK把目标文件放在由$O后面的字符指定的驱动器上，或者当$O后面的字符是'Z' 时，禁止目标文件的产生。

$Sd—符号

LINK 通常在命令行中第一个 REL文件所在的驱动器上产生符号文件，除非命令中包括了带有显式驱动器的输出文件。$S开关命令 LINK 把符号文件放在由$S后面的字符指定的驱动器上，或者当$S后面的字符是'Z' 时，禁止符号文件产生。

'td' 字符对后面的'$' 必须用逗号分开。$ 开关的整体部分与其他开关用逗号隔开。表示如下：

LINK PART1〔$SZ, $OD, $LB, Q〕, PART2

KINK PART1〔$SZODLB, Q〕, PART2

LINK PAKT1〔$SZ OD LB〕, PART2〔Q〕

以上三个命令是等价的。

在整个连接操作中，$I开关指定中间文件所使用的驱动器。命令行中其他'$' 开关可以被改变。当命令行从左向右处理时，'$' 开关的值将一直保留到它改变为止。这一般只在连接复盖时用。例如：

LINK ROOT〔OV1($SZCZ)〕(OV2)(OV3)(OV4〔$SACX〕)

将禁止 SYM 文件并且当 OV1，OV2 和 OV3 连接时产生控制台输出。当 OV4 连接时，SYM 文件将被放在驱动器 A：上，并且控制台输出将被送往控制台设备。

LINK 1.0 中使用的 NR和 NL开关，LINK 1.1 不认识，但能用$SZ 和$CZ 来完成这些功能，

1.7 请求项

PL/I-80 版本 1.1 通过请求项 (REL文件中一种特定的位模式) 向 LINK 表示要检索 PLILIB。这也是 Microsoft 编译连接它们的运行时程序库的方法。当 LIKN 处理一个库请求时，首先检索指定文件名的 IRL 文件。如果没有 IRL 文件，它再检索这个文件名的REL文件。两个检索都失败则发错误信息：

NO FILE：filename.REL

并使 LINK流产。以这种方式请求库，将在控制台上打印的符号表中出现，并具有'R QST'值

1.8 命令行错误

错误信息'FILE NAME ERROR'(文件名错)和 'INVALID SYNTAX' (非法语法) 不是全部信息都产生。在检查出任何命令行错误时，代替回送的是从命令尾部开始到出现错误的地方为止的符号，并跟上一个问号。例如：

```
LINK A, B, C; D
A, B, C; ?
LINK LONGFILENAME
LONG FILEN
```

1.9 附加的 LIB-80 工具

库中的模块可以用单个命令删除或代替。受影响的模块名用尖括号括起来，紧跟在后面的是包括这个模块的源文件名。下述例子说明这一特点的使用：

```
lid newlib=oldlib<mod 1>
lib newlib=oldlib<mod 1=file 1>
lib newlib=oldlib<mod 1=>
lib newlib=oldlib<mod 1,mod 2=file 2,mod 3=>
```

在第一种情况中，除模块MOD1 由文件 MOD 1 REL的内容代替外，建立的新库NEW-LIB.REL与文件 OLDLIB.REL 相同。当被代替的模块名与代替这个模块的 REL 文件的文件名相同时，可以用这种形式。

在第二种情况中，模块MOD1 被新库 NEWLIB.RELL 中的文件 FILE1.REL 所代替。这种形式用来代替模块名与代替它的文件名不同的模块。注意，如果文件名大于 6 个字符，就必须使用这种形式，因为 REL 文件中的模块名只截取到 6 个字符。

在使用第三种命令时，NEWLIB.REL 根据不带模块 MOD1 的文件OLDLIB.REL 来建立。最后的命令形式说明在尖括号中可以包括一组取代及／或删除的指令。

2.0 多行命令

如果一个命令在一行 (126 字符)上放不下，则可以通过用符号(&)终止的命令行来扩展该命令：符号 (&) 可以出现在任何命令字符的后面，但符号 (&) 的后面不能是文件名。LINK-80 在下一行星号(*)响应。这时，命令行可以继续，可以输入任意多个用符号(&)结束的行。命令的最后一行用回车结束。注意，XSUB 可用来提交多行 LINK-80命令(表 5)。

```
A>link main, iomod1, iomod2, iomod3, iomod4, iompd5, &
LINK 1.3
*lib1[s], lib2[s], lib3[s], lib4&
*[s], lastmod lp2000&
*,d2001
(. . .symbol table and memory map. . .)
A>
```

表 5

(编译校对 匡鲁江 康兴鹤)

SID 符号指令调试程序使用手册

引　言

SID 是 CP/M 的符号调试程序。它扩展了“CP/M 动态调试程序（DDT）用户指南”中所描述的 CP/M 标准调试程序，大大增强了汇编级程序检查的能力。具体地说，SID 包括了实时断点，完全地监控执行，符号反汇编，汇编及内存显示和填入功能。另外，包含实用程序功能的 SID 操作能被动态的装入并提供向后跟踪和直方图的特性。

手册第一章的第一节描述的是启动 SID 的命令形式和引用 SID 程序作用的命令行。第二节描述了 SID 通过符号表达式引用绝对机器地址的能力。第二章的第一节叙述了引用调试过程的命令；第二节叙述由 SID 实用程序所提供的附加调试特性。第三节介绍了 SID 调试对话的几个实例。

第一章　SID 使用方法及规定

第一节　在CP/M下的SID操作

1.1 启动 SID

打入下列命令之一就能启动 SID 程序。

(a)	SID	
(b)	SID	X·Y
(c)	SID	X·HEX
(d)	SID	X·UTL
(e)	SID	X·YU.V
(f)	SID	*U·V

在每一情况中，SID 装入到暂驻程序区（TPA），把它再定位到 TPA 的顶部，并覆盖 CP/M 控制台命令处理部分。图 1—1 示出了在 SID 装入之前的内存分布，而图 1—2 表示 SID 被装入和重定位之后存储器的分布。由于重定位处理，SID 与在特定计算机结构中 CP/M 管理的存储器的大小尺寸无关。

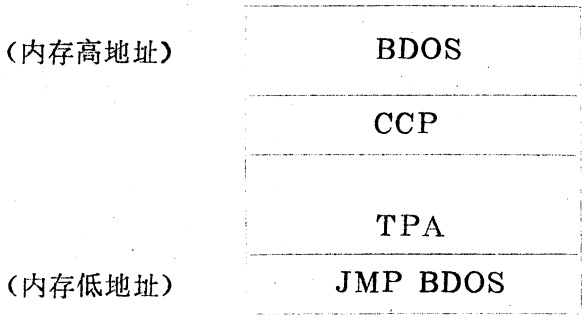


图 1-1　在 SID 装入前内存的分布

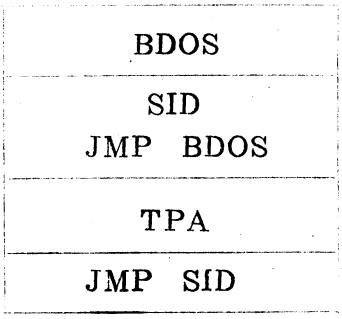


图 1-2　在 SID 装入后内存的分布

在装入和重定位后，SID 改变了 BDOS 的入口地址，反映了存储尺寸的缩小，如图 1-2 所示。而 TPA 下面的部分空间是用来存放测试程序的。值得指出的是在操作的时候，虽然

SID 仅占有上面 6K 的内存空间，但自身重定位处理至少需要一个 20K 的 CP/M 系统作为初始化设置。而给测试程序剩下大约仅 10K 空间。

上述命令形式（a）是装入并执行 SID，而无需装入测试程序到 TPA 中。这种形式用来检验内存或写入，并用 SID 内部的汇编功能测试简单的程序。

形式（b）和（a）相似。不同点在于（b）形式中为了接着进行测试，由 X.Y 所给出的文件被自动装入。注意，虽然 X.Y 被装入到 TPA 中，但并不执行，直到使用了下述命令之一：C（调用）、G（转向）、T(跟踪）或 U（不跟踪）之后，SID 把程序控制转到测试程序时才开始执行。你的责任是和调试检查程序一样应保证在 TPA 中有足够的空间来容纳测试程序，如果 X.Y 程序在软磁盘上不存在或不能被装入，SID 发出标准的错误提示信息“？”。如果没有发生装入错误，SID 作如下应答提示：

```
NEXT    PC    END
nnnn    pppp  eeee
```

在这里 nnnn，pppp 和 eeee 都是十六进制值，这些值分别指出了被装入程序之后紧接着的下一个地址空间、程序计数器的初始值和 TPA 的逻辑终点。因此，nnnn 一般是测试程序数据区的开始，pppp 是程序计数器的始端（置到 TPA 的开始），而 eeee 是测试程序可用的最终存储单元，如图 1—3 所示。虽然 X.Y 通常含有机器码，但操作员能命名一个 ASCII 文件，在这种情况下，这些程序的地址没有多大意义。

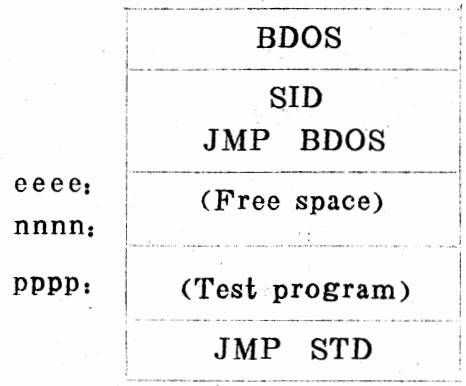


图 1-3　测试程序装入内存后的结构

命令形式(c)同形式(b)是相同的，不同处在于测试程序采用 INTEL “HEX” 格式，正象直接由 ASM（8080 汇编程序）或 MAC（8080 宏汇编程序）所产生的一样。在该情况下，从十六进制文件的终止记录获得程序计数器的初始值，除非这个值是零，而在此情况下程序计数器是置到 TPA 的开始部分。正如 ASM 和 MAC 手册所讨论的那样，程序计数器的值可以在源程序的 “END” 语句中给出。再次指出，你要负责保证十六进制记录不与 SID 调试程序或 CP/M 操作系统部分重叠。在十六进制格式中，如果十六进制文件没有生成或出现错误，SID 发出 “？” 提示。另外，前段中给出的程序基本单元被列表在控制台上。

当 SID 中包含实用功能时，就利用命令（d)。在该情况下，和上面一样 SID 首先被装入和再定位，然后实用功能被装入到 TPA 中。实用功能也自定位并立即传送到 TPA 的顶部，直接放置它们自己在 SID 程序下面。BDOS 入口地址的改变，反映了 TPA 的缩小，如图 1—4 所示。通常实用程序打印符号开始（Sign-on)信息或能不能从控制台输入的提示信息。

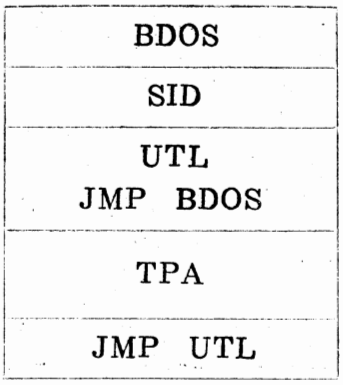


图 1-4　实用程序装入后的内存结构

命令（e）和（c）相似，不同处在于由 U.V 给出的符号表和程序 X.Y 一起被装入。符号信息被装入到现行 TPA 的顶部，测试程序移到下面，如图 1—5 所示。

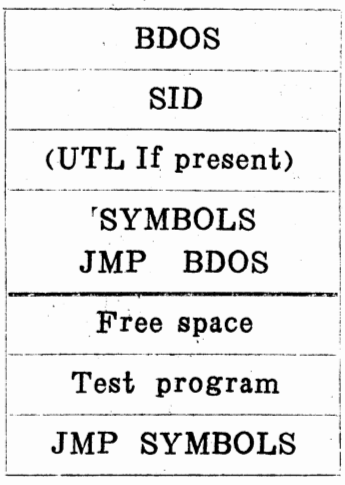


图 1-5　符号装入后的内存结构

在这种格式中，CP/M 宏汇编产生符号表。符号表必须是一个地址与符号名一一对应的序列，在这里地址由 4 位十六进制数组成，它和从这个地址单元中取出的符号用一个空格隔开。符号最多用 15 个 ASCII 字符码组成，被一个或多个标记（↑I）或一个回车换行所终止。注意，只要按下述格式，就能用 CP/M 编辑建立和修改符号表。

程序装入后的响应如前面的命令形式（b）所述。当 SID 开始装入符号表时，它就显示如下信息：

SYMBOLS

这个信息指出，任何后续的错误都是由于符号表装入处理引起的。特别是，在 SYMBOLS 响应后出现的“？”错误是由于非法的或不正确的格式符号文件所造成的。

命令形式（f）的不同点在于，符号文件 U.V 装入时程序没有被装入。（f）和命令（e）是相同的。

启动 SID 程序命令的典型例子如下所示：

(a)　SID

(b)　SID　DUMP.COM

(b) SID DUMP.ASM
(c) SID SAMPLE.HEX
(c) SID DUMP.HEX
(d) SID TRACE.UTL
(d) SID HISP.UTL
(e) SID DUMP.COM DUMP.SYM
(e) SID DUMP.HEX DUMP.SYM
(e) SID TEST.COM TEST.ZOT
(f) SID *DUMP.SYM

1.2 SID 命令输入

输入到 SID 的命令由控制 SID 程序的一系列命令行 (Command lines) 所组成。这些命令允许对内存和 CPU 寄存器内容进行显示，并且在测试程序调试期间能直接执行和进行断点操作。

当 SID 准备好接受下一个命令时，在控制台上就显示一个“*”。每条命令都是单个字母，后面跟着选择参数并用一个回车结束。注意，CP/M 的所有标准行编辑特性都是有效的，最大为 64 个命令字符。CP/M 标准行编辑功能列表如下：

表 1-1 CP/M 行编辑控制

控制字符	功能
↑C	CP/M 系统重新引导，返回到 CCP
↑E	实际的行结束
↑H	删去最后一个字符，光标退回一格
↑P	打印控制台输出，(乒乓开关)
↑R	重新打印当前的输入行
↑S	停止/启动控制台输出
↑U	删除当前的输入行
↑X	和↑U 一样
擦掉	删去并回送最后一个字符

↑符号表示在按下这个实际功能键的同时，必须同时按住 control 键。注意，↑R、↑U 和↑X 键引起 CP/M 在结束这一行时打印一个“*”，表示这一行被除去。

SID 的各种命令在控制台上产生长输出（见 D 命令显示内存例）。在这种情况下，按控制台上的任一键，都能在完成打印前停止打印输出（都有返回能力）。

控制 SID 作用的单一字母命令，在命令行开始时被打印，可以用大写或小写字母输入命令。表 1—2 概括了这些有效命令。

每一个命令的详细说明在下一节中给出，几乎所有的命令都紧靠有控制命令作用的参数。这些参数可以是计数值或内存地址，既能以文字形式出现又可以符号形式出现，但最终都折算到 0～65535 的范围内（4位十六进制数）。

表1-2 字母命令

字　母	意　义
A	从指定内存单元开始汇编
C	由SID 调用指定内存单元的程序
D	用 HEX 和 ASCII 显示内存
F	用常数值填入内存
G	转向执行测试程序
H	十六进制运算
I	输入 CCP 命令行
L	用 8080 助记符指令列表
M	移动内存字块
P	遍数指针置位，复位，显示
R	读测试程序和符号表
S	内存置数
T	跟踪测试程序的执行
U	不跟踪（监督）测试程序
X	检查 CPU 状态寄存器

举一个“显示内存”命令的例子，取如下形式：

D ssss, eeee

其中 D 是命令字母，ssss 和 eeee 是“命令参数”，分别给出了显示内存的起始和终止地址。在它们最简单的形式中，ssss 和 eeee 可以是文字十六进制值，如下所示：

D 100, 300

指示 SID 用十六进制和 ASCII 码打印这些值，而这些值被包含在 0100H 到 0300H 之中。

即使能够引用程序列表得到机器的绝对地址，但对通过程序符号快速访问机器地址来说，SID 支持更广泛的结构。能由符号表达式组成的命令参量在下节中详细叙述。

第二节　SID 符号表达式

SID 有一个十分方便灵活的功能，即能通过符号表达式引用绝对机器地址。符号表达式用到的名字能从测试程序中获得，而测试程序包含在由 CP/M 宏汇编产生的“SYM”文件中。符号表达式也可由十六进制、十进制或 ASCII 码字符串形式的文字值组成，而这些值能同各种运算符组合以提供数据或程序区的索引或间接地址。这一节叙述的符号表达式使你能够用下节各命令中的参数进行综合处理。

2.1 文字十六进制数

SID 通常接收和显示十六进制值，而这个十六进制数是由十进制数 0 到 9 同十六进制数 A、B、C、D、E 和 F 一道组成，A～F 分别对应十进制的 10 ～ 15。

在 SID 中的一个文字十六进制数是由一个或多个相连的十六进制数字组成。如果打印一

个四位数，最左边的数是最高有效位，最右边的数是最低有效位。如果一个数字大于四位，只取右边四位为有效位并保留，最左边的位数就被丢掉。下面所示的例子表示出与一些所给输入值对应的十六进制和十进制值：

输入值	十六进制值	十进制值
1	0 0 0 1	1
1 0 0	0 1 0 0	2 5 6
f f f e	F F F E	6 5 5 3 4
1 0 0 0 0	0 0 0 0	0
3 8 0 0 1	8 0 0 1	3 2 7 6 9

2.2 文字十进制数

虽然 SID 的标准数是十六进制数，但也能输入上述十进制数来取代，这时在十进制数前加“#”号。在这种情况下，这个数必须由一个或多个十进制数字（0到9）所组成，左边是最高有效数位而右边是最低有效数位。在把十进制数转换到等效的十六进制数时，按前述十六进制数的规则，对十进制数进行填写或取舍。

下面这个例子中的左方是输入值，它产生的内部十六进制值示于右方：

输入值	十六进制值
#9	0 0 0 9
# 1 0	0 0 0 A
# 2 5 6	0 1 0 0
# 6 5 5 3 5	F F F F
# 6 5 5 4 5	0 0 0 9

2.3 文字字符值

为操作员方便，SID 也可以接收由一个或多个用文字表示的 ASCII 码字符，它是由撇号（'）括起来的字符串。在表达式中当做文字值看待，在两个撇号中间所保留的字符和输入的一样（也就是没有发生转换）。同时最左边的字符是最高有效位，而最右边的字符是最低有效位。类似于十六进制数，对一个字符长度的字符串来说在左边填零，而对两个以上的字符串只保留最右边的两个字符，最左边的字符被丢掉。

注意，封闭的撇号不包含在字符串中，也不包含在字符计数中，仅有一种情况例外。为了写带撇号的字符串，相邻的一对撇号可以简化为1个单一的撇号而包含在字符串中，作为一个标准的图形字符。

下面示出输入的字符串（左边）所产生的十六进制值（右边）。（对这些例子，ASCII 码大写字母被编码为十六进制值是从 41 开始，小写字母从 61 开始，一个空格十六进制值是 20，一个撇号的十六进制编码是 27）。

输入字符串	十六进制值
'A'	0 0 4 1
'A B'	4 1 4 2
'A B C'	4 2 4 3
'a A'	6 1 4 1
''	0 0 2 7

″ ″ ″	2 7 2 7
′ A′	2 0 4 1
′A ′	4 1 2 0

2.4 符号引用

在SID调试会话期间提供符号表，通过下面符号引用的三种方式可以引用与符号有关的值。

(a) .S

(b) @S

(c) =S

这里S表示1到15个字符序列，这个字符序列与表中的符号相匹配。

形式（a）产生相对于符号S的地址值（也就是说，这个值与表中的符号相关）。形式（b）产生由.S给出的包含在二个内存单元中的16位“字”值。而形式（c）由.S给出的内存单元中取出一个8位字节值。例如，假设输入的符号表包含2个符号，并且表示如下：

0100 GAMMA 0102 DELTA

另外，假设内存从0100开始，包含有如下的字节数据值：

0 1 0 0 : 0 2

0 1 0 1 : 3 E

0 1 0 2 : 4 D

0 1 0 3 : 2 2

以这个符号表和这些内存值为基础，符号引用表示在左下方，所产生的十六进制值表示在右下方：

符号引用	十六进制值
.GAMMA	0 1 0 0
.DELTA	0 1 0 2
@GAMMA	3 E 0 2
@DELTA	2 2 4 D
=GAMMA	0 0 0 2
=DELTA	0 0 4 D

二次读出被存储在8080内存中的16位值，且第一个数是低位字节，从而得到在0100和0102中一个字的值分别为3E02和224D。

2.5 限定符号

值得注意，由于在各自独立的程序块中对不同子程序或数据区使用了相同名字，而这些程序块又是分别汇编和编译的，因此在符号表中会产生重复的符号。此外，程序块结构语言，如PL/M，允许相同定义名的嵌套而不会冲突。因此，SID允许引用如下形式的限定符号

$S_1/S_2/\cdots/Sn$

在这里S_1到Sn表示在特定的对话期间存在于表中的符号。

SID总是从符号表中的第一个到最末一个符号进行检索，符号按顺序出现在符号文件中；对限定符号SID开始匹配第一个S_1符号，然后对与S_2相匹配的符号进行扫描，一直扫描到与符号Sn相匹配。如果这个检索和匹配过程没有成功，SID将在控制台上打印“?”。

例如，假设符号表显示为：

0100 A 0300 B 0200 A 3E00 C 20F0 A 0102 A

在这个符号文件中，预置内存的初值如前节所示，则非限定和限定符号引用表示在左下方，其产生的十六进制值表示在右下方：

符号引用	十六进制值
.A	0100
@A	3E02
.A/A	0200
.C/A/A	0102
=C/A/A	0004D
.B/A/A	20F0

（原文.B 为@B—译者）

2.6 符号操作符

利用一元的和二元的“+”和“-”运算符能把文字数值、文字串和符号引用组合成符号表达式。这完整的数、符号和操作符的序列，写入时不能嵌入空格。此外，从左向右求这个序列的值，在每一步计算中都产生一个 4 位的十六进制数值。在这个求值过程中上溢和下溢都是不考虑的，而最后的值成为命令参数，它们的解释说明取决于在它们之前的特定命令字母。

在两个操作数之间放置一个“+”号表示第二个操作数加前面被累加的值，这个和成为在求值中这一步的新的累加值。如果表达式开始就用一元的“+”，那么符号表达式直接取前面的（已完成的）值作为起始累加值（在 SID 启动时，假设累加值为零）。例如：

D FE00+*128, +5

最前面的表达式“FE00+*128”表示 FE00 和一个十进制数 128 相加得 FE80 作为显示命令的起始值。第二个操作数“+5”开头部分是一个一元“+”，表示以前面的表达式值（FE80）作为符号表达式的基数生成的值，FE85 作为显示操作的结果值。因此，上述命令相当于：

D FE80, FE85

在符号表达式中，“-”号导致 SID 从当前 16 位累加值中减去文字数值或符号引用。如果表达式开始是一个负号，那么起始累加值取零，即

-X 如同 0-X

在这里 X 是任一有效符号表达式。例如下述命令：

D FF00-200, -*512

相当于这样一个简单的命令：

D FD00, FE00

一个特殊的向上箭头“^”表示的操作符在测试程序时表示取栈顶值。通常在测试程序中 n 个向上箭头的操作符序列表示取出测试程序中第 n 个堆栈顶，但不改变测试程序的栈内容或栈指示器。这个特定操作符经常同 G（GO）命令一起使用，即在测试期间内设置从子程序返回点，这在 G 命令中将被说明。

2.7 标准的符号表达式

在程序测试中，SID 符号表达式的组成，常与程序结构密切相关。假设你需要调试一个

分类程序，这个程序包含的数据项列表如下：

LIST 是分类的字节值符号名表，假设表中元素小于255个，用LIST(0)，LIST(1)，…，LIST(254)表示。

N 是一个可变字节，它指出在LIST中的有效项数。N值小于256。分类项被存储在LIST(0)到LIST(N－1)中。

I 是字节的下标，指出在分类处理比较中的下一项，也就是LIST(I)是紧跟在序列中放置的下一项。I在0到N－1的范围内。

已知数据范围，命令

D. LIST, +#254

为了分类，整个区域被保留

LIST(0), LIST(1), …, LIST(254)

命令

D. LIST, +=I

表示LIST向量达到并且包含了下一分类项

LIST(0), LIST(1), …LIST(I)

命令

D. LIST+=I, +0

仅表示LIST(I)。最后，命令

D. LIST, +=N－1

表示LIST中容纳的分类有效项范围仅是：

LIST(0), LIST(1), …, LIST(N－1)

在确定的命令方式中，SID用的符号表达式取决于所发出的各种命令，这些命令的细节说明见下章。

第二章　SID使用命令及实例

第一节　SID命令

在控制台上开始启动SID命令后，紧跟着就输出“*”。命令控制调试过程不但控制测试程序的执行，而且还可以改变和显示CPU各寄存器和内存的内容。

下面叙述SID所接受的命令。

1.1 汇编（A）命令

A命令可以把标准的Intel助记符以及符号引用的操作数用8080机器码和操作数插入到当前内存映象中。A命令的形式为：

(a)　As

(b)　A

(c)　-A

s表示任意有效的符号表达式。形式(a)是在s所给的地址上开始一行一行地串行汇编，每条地址逐一显示出来，直到打印一空行(即一个单一的回车)为止。形式(b)与(a)相当，只是汇编的起始地址取自上次被汇编、被列表或被跟踪的地址（参看“L”、“T”和“U”命令)。下面的指令序列是汇编一个短程序进入暂驻程序区(注意，结束每一个命令行都必须用一个回车)的例子：

```
A100                          在0100处开始汇编
0100  MVI  A,10               用十六进制数10装入A
0102  DCR  A                  A寄存器减1
0103  JEZ  102                循环直到零
0106  RST  7                  返回到调试程序
0107                          单个回车
```

由于每一行地址逐次被显示，因此，可以在这一行送入一条助记符指令，或送一个单回车返回到SID命令方式（同“S”命令一样也接受单一的“.”终止串行汇编)。

在助记符和操作数之间允许的分界符，包括空格或指定序列。

对无效的助记符或作法形成的操作码产生“？”错误。在此情况下，控制返回到命令方式。可以继续进行另一个命令行或仅打印单一的“A”返回到汇编方式，而设定的起始地址能自动取自上次被汇编的地址。

SID汇编/反汇编部分是一个独立的程序块，并且被删除以增加有效的调试空间。因此，形式(c)是删除该模块回收大约$1\frac{1}{2}$K字节的空间。由于整个的SID调试程序需要大约6K字节，这样SID就只需要大约$4\frac{1}{2}$K字节了。在此方式中，当汇编/反汇编程序块被删除时，命令A和L实际上也被删除。此外，跟踪和不跟踪功能仅显示十六进制码，而且有效跟踪记忆仅显示十六进制地址。任何现行的符号信息，在这时同样被除去，尽管这个信息能被再装入（看“I”和“R”命令)。

一个有效汇编命令的例子如下所示：

A100

A*100

A.CRLF+S

A@GAMMA+@X-=I

A+30

假定命令 A100 已被送入，在 SID 和操作员之间发生的相互作用如下：

SID 提示	操作员输入
0100	MVI C,.A-.B
0102	LXI H,.SOURCE
0105	LIX D,+100
0108	MOV A,M
0109	INX H
010A	STAX D
010B	INX D
010C	DCR C
010D	JNZ 108
011O	("return" only)

A、B、和 SOURCE 都是出现在符号表中的符号。在这种情况下，SID 计算 A 和 B 地址之差，作为 MVI 指令的操作数。LXI H 的操作数变成 SOURCE 的地址，而 LXI D 指令接收的操作数值为.SOURCE+100，因为.SOURCE 已直接转换为符号表达式中的值。这个特定的程序段是移动由相应符号地址值所确定的内存块。

1.2 调用（C）命令

C 命令用来执行对内存中某一绝对地址单元的调用，测试程序的寄存器状态不受干扰。C 命令取形式如下：

(a) Cs

(b) Cs,b

(C) Cs,b,d

虽然 C 命令被设计成供 SID 实用程序用，但它也能调用测试程序的子程序以执行程序预置，或在测试程序执行前，使 CP/M BDOS 调用各种系统参数的预置程序。

上述形式 (a) 执行绝对地址单元 S 的调用，在这里 S 是一个符号表达式。在此情况下，在调入时寄存器 BC = 0000 和 DE = 0000 通常从子程序退出，执行一条 RET 指令，把控制返回给 SID，紧接着显示正常的 SID 提示。

上述形式 (b) 和 (a) 是等效的，不同点只在于 BC 这一对寄存器被置为表达式 b 的值，而 DE 置为0000。

形式 (c) 和 (b) 相似，BC 寄存器对被置为 b 值，而 DE 对被置成 d 的值。有效 C 命令的几个例子如下所示。也可参照在讨论 SID 实用程序时的有关通用初始化、数据采集和显示功能中 C 命令的例子。

C1000

C*4096

C.DISPLAY
C@JMPVEC + = ×
C.CRLF, #34
C.CRLF, @×, + = ×

1.3 显示内存（D）命令

D 命令用于显示所指定内存区内的字串（8 位）或字（16 位）的内容。D 命令在输出时，既能以十六进制方式显示，也能以 ASCII 码方式显示。D 命令取如下形式：

(a) Ds
(b) Ds, f
(c) D
(d) D, f
(e) DWs
(f) DWs, f
(g) DW
(h) DW, f

(a) 到 (b) 显示内存字节格式，(e) 到 (h) 显示内存字格式。显示字节格式表示为：

aaaa bb bb bb···bb cc···cc

在这里 aaaa 是显示行的基地址，后面的 16 个 bb 序列是表示存储在以 aaaa 为起始地址中的十六进制数的值。序列 c 表示相同显示区中可能的ASCII 码格式。句号（·）是在没有数据项的时候，看做为一个固定数位，相当于一个用文字表示的字符被显示。当数据项没有图形字符相对应时，显示句号（·）占据此位置。

字节方式显示是被规格化的，它以 16 的倍数作为地址界限。也就是说，如果起始地址 aaaa 不是 16 的倍数，那么，显示行就显示下一个以 16 为倍数的界限地址。每一个显示行后都包含 16 个数据元素，直到碰到最后一个显示行为止。

命令形式 (e) 到 (h) 是按字方式显示，与上面所描述的字节方式显示相类似，不同点在于数据元素是打印双字节格式：

aaaa wwww wwww···wwww cc···cc

此处 aaaa 是显示行的起始地址，后面 8 个 wwww 序列表示被存储在起始地址为 aaaa 内存中的数据项。类似于字节方式显示，序列 c 表示在起始地址 aaaa 的内存中的 ASCII 码字符与在字节方式显示一样，句点用来占据一个固定数位，这时该位置上的字符没有图形符号。

和字节方式显示相反，在字方式显示中，没有规格化的模为 16 这一地址界限。重复调出 8080 以低字节在前的双倍字长，因而字方式显示颠倒每一个字节对，致使各数据项被显示成 4 个十六进制数字，最高有效数位是在高地址上。

命令形式 (a) 是在字节格式中从起始地址单元 s 显示内存的内容，占用标准 CRT 屏幕的 1/2 (12 行)。当需要从某一位置开始检查内存而又没有一个明确的结束地址时，这个命令形式是有用的。

命令形式 (b) 与 (a) 相似，但有明确的结束地址。在该情况下，其初始地址为 s，并连续显示到地址 f。对一个非常长的打印输出，只要按任一个字符键就能停止显示，例如按回车键。

形式（c）与（a）和（b）相类似。不同点在于显示的起始地址是取自上一次被显示的地址或取自内存地址指针寄存器（HL）的值。在这种情况下，在这之前上一个断点的显示没有发生。它常常是很方便的。例如，用形式（a）显示一个内存区，跟着用 D 命令的形式（c）继续显示。每一个 D 命令都显示内存另外 1/2 的屏幕。

命令形式（d）和（b）相似，不同处是起始地址按照上述形式（c）自动产生。

作为一个例子，假设十进制值 1 到 255 存储在起始地址为十六进制 0100 的内存单元中。这个命令为：

```
D100, 12A
```

其扩展形式在屏幕上显示如下，

```
0100 01 02 03 04 (etc.) 0E 0F 10.. (etc.) ..
0110 11 12 13 14 (etc.) 1E 1F20.. (etc.) .
0120 21 22 23 24 (etc.) 2q 2A 2B! "*$%&' ( )* +
```

命令形式（e）到（h）的类似之处已在（a）到(b)字节显示中给出，不同点是以字格式来显示。形式（e）在字格式中是从单元 s 按 1/2屏幕显示，而形式（f）是从单元 s 到单元 f 显示。形式（g）从上一个显示单元显示或从 HL 指针处开始显示，只要紧接着前面已经有一个未显示过的断点。形式（h）和（g）相类似，但它显示到单元 f。这个命令：

```
DW100, 128
```

则产生下述行。其扩展输出方式如下：

```
0100 0201 0403 (etc.) 0E0D 100F.. (etc.) ..
0110 1211 1413 (etc.) 1E1D 201F.. (etc.) .
0120 2221 2423 (etc.) 2928 2B2A! "*$%&' ( )* +
```

下面是有效 D 命令的例子：

```
DF  3F
D#100, #200
D.GAMMA, .DELTA+#30
D.GAMMA
DW @ALPHA, +#100
```

1.4 常数填入内存（F）命令

F 命令是用一个常数字节值填入内存，其形式如下：

```
F,s, f, d
```

s 是填入的起始地址，f 是填入的结束地址，而 d 是存储在 s 到 f 中的 8 位数据项。你的责任是不要填入到 SID 保留区内 CP/M 常驻部分所占有的内存单元中去。下面是有效 F 命令的例子：

```
F100   3FF, FF
F.GAMMA, +#100, #23
F@ALPHA, +=I, =X
```

1.5 转向（G）命令

G 命令是通过程序控制转向被测试的程序，测试程序是从 G 命令所指定的地址处开始执行。也就是说，G 命令把处理控制从 SID 中释放后交给被测试的程序。这样，在执行达

到断点或扫描遍数以前，不会返回到 SID（扫描遍数的讨论见 P 命令）。然而操作员能使用强制手段使其返回到 SID。为此由测试程序提供的一条 RSTT 指令，来中断处理执行而返回到 SID，或用外部硬件强制返回，例如用前面板控制开关（如果有的话）。

G 命令取形式如下：

(a) G
(b) Gp
(c) G, a
(d) Gp, a
(e) G, a, b
(f) Gp, a, b
(g) -G
(h) -Gp
(i) -G, a
(j) -Gp, a
(k) -G, a, b
(l) -G, p, a, b

形式 (a) 到 (f) 是用被使能的符号特性启动测试程序的执行。而形式 (g) 到 (l) 在功能方面是相同的，但 SID 的符号特性不适用。具体说，形式 (a) 是从测试程序的机器状态所给出的程序计数器 (PC) 值开始执行测试程序的（按机器状态显示见 X 命令）。在此情况下，测试程序中不用设置断点。形式 (b) 与 (a) 相似，只是在开始执行前预置测试程序的 PC 为 P，这时在测试程序中也不用设置断点。和 (a) 相类似，形式 (c) 也是按现行 PC 的值开始执行，但要在单元地址 a 上置一断点。测试程序接受控制并实时运行。直到 PC 内容等于 a 为止。还需指出，在达到扫描遍数或 RSTT 时如上所述控制也返回到 SID。

当达到断点地址 a 时，在控制台上按如下形式后可打印出断点地址：

*a .s

在这里 s 是与地址 a 相当的符号表中的第一个符号（如果有符号的话）。注意，当 SID 返回命令方式时，地址 a 处的临时断点自动被清除（对永久断点看“P”命令）。

形式 (d) 兼有 (b) 和 (c) 的功能，测试程序的 PC 置入地址 P 并把一个临时断点置在单元地址a上。如上所述，在控制 SID 返回时断点被清除。应该指出的是，如果 P = a，断点总是立即发生。但是如果这是不希望的，则要用跟踪功能单步地通过现行地址紧跟着再用 G 命令（关于跟踪功能的作用见 T 命令）。

形式 (e) 允许扩充到两个临时断点，断点地址为 a 和 b。从现行 PC 处开始执行程序，并且一直延续执行，直到碰到地址 a 或 b 为止。当SID 返回到命令方式时，这两个临时断开的地址都被清除。形式 (f) 和 (e) 相似，只是在启动测试程序前 PC 中的初始值被置到单元 P。

值得注意，当你使用 G 命令时，在断点地址处指令不被执行。例如，假设被命名为 TYPEOUT 的子程序，在测试程序中被设置在地址 0302 处，其包含的机器码为：

```
TYPEOUT:
    0302    MOV  E, A
```

```
0303    MVI  C, 2
0305    JMP  0005
```

进一步假设，当正在测试一个使用了 TYPEOUT 子程序的程序时，在那里断点的地址是已被设置的。输入命令：

```
G, .TYPEOUT
```

测试程序从当前 PC 值开始执行，当到达 TYPEOUT 子程序时停止执行，用断点信息：

```
*0302.TYPEOUT
```

指明控制从测试程序返回 SID。此时测试程序的程序计数器中为单元 0302（也就是 .TYPEOUT），并且在这个单元中的指令还没有被执行。你可以利用任意的 G，T 或 U 命令去执行 TYPEOUT 子程序。在此情况下，用下面的命令是有效的。

```
G, ^
```

这个命令继续从 0302 执行程序，并且设置一个栈顶元素的断点（由"^"给出）。因为栈顶单元必须是子程序的返回地址，所以这个特定 G 命令执行 TYPEOUT 子程序后，以一个断点返回到调用子程序 TYPEOUT 指令的下一条指令。

需要注意，G 命令和 U(非跟踪)命令之间的主要区别是用 G 命令时程序实时而不被监控地继续执行，而用 U 命令时每一条指令都是在单步方式下被执行。在 U 命令下执行完全被监督的优点在于，你能在测试程序执行中的任一点收回控制。而在非跟踪方式中，测试程序的执行时间被大大地加长了，这是由于紧跟着每一条指令后断点被自动置位和清除所致。

下面是有效 G 命令的例子：

```
G100
G100, 103
G.CRLF,.PRINT, *1024
GaJMPVEC+=I, .ENDC, .ERRC
G, .ERRSUB
G, .ERRSUB, +30
-G100, +10, +10
```

1.6 十六进制值（H）命令

H 命令执行包括数制转换的操作的十六进制计算。H 命令形式如下：

```
(a)   Ha, b
(b)   Ha
(c)   H
```

形式（a）计算两个十六进制操作数的和与差，显示结果

```
ssss  dddd
```

其中 ssss 是 a+b 的和，而 dddd 是 a-b 的差，在此忽略了上溢和下溢

形式（b）执行数和字符的转换，a 是符号表达式，在此情况下，显示格式为：

```
hhhh  *dddd'c'.s
```

此处 hhhh 是 a 的 4 位十六进制值；*dddd 是（相当于）a 的 5 位十进制值；c 是当 a 是图形时 a 的 ASCII 码值，而 s 是在符号表中与 a 值相对应的第一个符号)如果该符号存在的话)。例如，在前面的例子中，假设的 GAMMA 的符号被存放在地址 0100 单元中，H 命令

表示在下面的左方，结果显示在下面的右方：

命　令	显示结果
H0，1	0001　FFFF
H41	0041 *65 'A'
H100	0100 *256.GAMMA
H.GAMMA	0100 *256.GAMMA
H=GAMMA	0001 *1
H@GAMMA	0201 *513
HFF+=GAMMA	0100 *256 .GAMMA
H'A'	0041 *65 'A'
H'A'+=GAMMA	0042 *66 'B'

命令形式（c）打印列表符号和它们对应的全部地址值。列表按输入的符号从头到尾进行打印，在打印输出中按下任一个字符键都能中止打印。

1.7 输入行（I）命令

当测试程序在CP/M环境中运行时，常用来模拟CCP的命令行，通常是为程序装入做准备。I命令形式如下：

I　ccccc···ccc

序列c表示ASCII码字符，而这字符在CCP命令行中通常跟在测试程序名后。例如，通常在CCP命令方式中，CP/M"DUMP"程序是由打印如下内容来启动的：

DUMP　X.COM

它使CCP去查找并装入DUMP.COM文件，并把"X.COM"作为参数传送给DUMP程序。特别是，CCP预置两个默认文件控制块以及包含DUMP命令字符的默认命令行。

为了跟踪和调试一个程序，如DUMP程序，则要求SID程序用下列命令：

SID　DUMP.COM

该命令装入含有DUMP机器码的命令文件。如果符号表有效，则SID援引：

SID　DUMP.COM　DUMP.SYM

在这两种情况下，SID都装入DUMP程序并用命令提醒控制台。准备换模拟CCP的命令行，就打印命令：

IX.COM

此处I表示输入命令，后面是模拟命令行。程序员可以开始调试运行，同时默认区也就相应地建立。

I命令逐一地在内存低地址中预置默认文件控制块，通常标号DFCB 1被定位在005C。由I命令完成预置的文件控制块，这意味着程序可以简单地访问DFCB 1并执行，还可以在没有更多预置工作时做打开、建立和删除操作。为便利起见，第二个文件名被预置在元单DFCB 2，它是在地址DFCB 1+0010处（十六进制）。

因为在DFCB 2上的16个字节区能被任意文件操作所改写，所以在DFCB 1上执行文件操作前，你的任务是转移第二个驱动号、文件名和文件类型到另一内存区。另外补缺缓冲区，其标号为DBUFF，它能容纳整个命令行，用存有命令行长度的缓冲区中第一个字节进行预置。在一个标准的CP/M系统中，DBUFF区设在起始单元0080到结束单元00FF中。然而需指

出，I命令把模拟CCP命令行限制在63个字符，这是因为在模拟时使用了SID的行缓冲区。

下面给出一个I命令的形式：

I d1:f1.t1 d2:f2.t1

这里d1:和d2:是（任选项）驱动标识符，f1和f2是（相当8个字符）文件名，t1和t2（三个字符任选项）是文件类型。两个补缺文件控制块名按下面形式配备：

DFCB1:d1′ f1′ t1′ 00 00 00 00

DFCB2:d2′ f2′ t2′ 00 00 00 00

00 （当前记录字段）

如果在初始命令行中无d1:，那么d1′=00（表示自动选择补缺驱动器），在其他情况下，如果d1=A，B，C或D，则各自相应地预置d1′=01，02，03或04到文件控制块，以自动选择磁盘。字段f1′用f1提供的ASCII码文件名预置，用ASCII码的空格符填满8个字符的字段。同样，文件类型t1′也是用ASCII码预置，用空格符填满3个字符的字段。

在f1和t1中的小写字母转换为大写字母送到各自的f1′和t1′中，超过各字段长度的名字从右被截去，最后把伸长的字段长度减为零，为BDOS请求打开或建立文件做准备。

由d2，f2，和t2给出的第二个补缺文件控制块，用相同的方法产生并存储在006C起始的单元之中。注意，在007C单元中的当前记录字段也被预置为00，如果f1，t1，f2和t2中的任一个字段不包含在命令行中，那么它们在补缺文件控制块中相应的字段用空白填入。

用“*”或“?”符号表示的多义性标记与CCP中的处理方法相同：当“*”号在名字或类型字段中出现时，用“?”字符填入整个字段。下面所示的输入命令行说明了各种非多义性和多义性文件名默认区的预置，这个区显示在右边，其内容是从标号地址开始的十六进制值。在这里，ASCII码，A，B，C和D各自对应十六进制值41，42，43和44。此外，专用符“:”，“.”，“*”和“?”的ASCII码的空格对应十六进制值20。

命令行	补缺数据区的预置
I	DFCB1: 00
	20 20 20 20 20 20 20 20
	20 20 20 00 00 00 00
	DFCB2: 00
	20 20 20 20 20 20 20 20
	20 20 20 00 00 00 00
	00
	00
	DBUFF: 00 00
IA.B	DFCB1: 00
	41 20 20 20 20 20 20 20
	42 20 20 00 00 00 00
	DFCB2: 00
	20 20 20 20 20 20 20 20
	20 20 20 00 00 00 00
	00
	00

```
                    DBUFF: 04  20  41  2E  42  00
IA:B.C  b:d.e       DFCB 1: 01
                            42  20  20  20  20  20  20  20
                            43  20  20  00  00  00  00
                    DFCB 2: 02
                             44  20  20  20  20  20  20  20
                            45  20  20  00  00  00  00
                            00
                            00
                    DBUFF: 0B 41  3A  42  2E  43  20
                              42  3A  44  2E  45  00
IAA*.B7C D:         DFCB 1: 00
                              41  41  3F  3F  3F  3F  3F  3F
                              42  3F  43  00  00  00  00
                    DFCB 2: 04
                              20  20  20  20  20  20  20  20
                              20  20  20  00  00  00  00
                              00
                              00
                    DBUFF: 0B 20  41  41  2A  2E  42
                              3F  43  20  44  3A  00
```

需要指出，I 命令和 R 命令一起也被用在 SID 已启动装入后的程序文件和符号表的读入中。在这种情况下，I 使用的细节由下面的 R 命令提供。

下面给出了附加的有效 I 命令：

```
I  x .dat
Ix .inp  Y .out
Ia:x .inq  b:Y .out  $-p
I  TEST .COM
I  TEST .HEX  TEST .SYM
```

1.8 列表（L）命令

L 命令是对计算机内存中的机器码反汇编，即把符号标号和操作数安置在适当的可能字段上。L 命令形式如下：

(a) Ls
(b) Ls, f
(c) L
(d) -Ls
(e) -Ls, f
(f) -L

形式 (a) 是列出从符号单元 S 开始的 12 行（1/2 CRT 屏幕）被反汇编的机器码。形式

(b)对反汇编指定一个确切的区域，s为被指定的起始单元，f是反汇编的最终单元。形式(c)和(a)相似，但反汇编是从上次被列表、汇编（见命令A）、跟踪（见T和U命令）或间断的地址（见G和P命令）开始。由于形式（c）也是在1/2 CRT屏幕上列表，因此它经常跟踪在形式（a）后使用，继续反汇编过程到另一个程序段。形式（d）到（f）类似于（a）到(c)，但SID的符号性能不适用。具体说，在反汇编期间，负字首制止任何符号查阅操作。

L命令输出取如下形式：

```
sssss：
aaaa  opcode  operand  .ttttt
```

此处"sssss："表示由十六进制值地址aaaa提供的程序地址的标识符（当该符号存在时）。"opcode"字段给出在地址单元aaaa中指令的8080助记符，而"operand"字段是给出内存中跟着操作码后面的十六进制值（当存在该值时）。".ttttt"是指令引用了一个和符号表中符号相匹配的内存地址时被打印出的内容。

当在列表地址处的操作码不是8080的有效助记符时，输出形式是：

```
? ? =hh
```

在这里hh是无效操作码的十六进制值。

几个有效L命令列表如下：

```
L100
L*1024, *1034
L.CRLF
L@ICALL, +30
-L.PRBUFF+=I, +'A'
```

1.9 移动内存（M）命令

M命令能把数据块从内存的一个区移到另一个区。M命令形式如下：

```
Ms, h, d
```

这里s是移动操作的起始地址；h是移动的高(最后的)地址，而d是开始接受数据的目的地址。SID在起始地址到目的地址一次移动一个字节，每当一个字节值被移动时，起始和目的地址就分别加1。当起始地址增加至超过最终地址h时(原文为f—译者注)，传送过程结束。例如：命令：

```
M100, 1FF, 3000
```

是把0100到01FF整个内存块复制到内存3000到30FF目的区。0100到01FF的数据块仍保持完好。

注意，在传送过程中数据区可以重叠，如命令：

```
M100, 1FF, 101
```

在这个例子中，从单元0100开始的值被传送到从0101到0200整个程序块。

一些有效的M命令列表如下：

```
M-100, FFD0, 100
M.X, +=Z, .Y
M.GAMMA, +FF, .DELTA
M@ALPHA+=X, +*50, +100
```

1.10 扫描遍数计数器（P）命令

P 命令允许在程序测试过程中置位和清除“扫描遍数指针”（“pass points”）和“扫描遍数计数”（“pass counts”）。P 命令取如下形式：

(a) Pp

(b) Pp, C

(c) P

(d) -Pp

(e) -P

“扫描遍数指针”是一个监视测试程序执行的程序单元，类似于临时断头（看 G 命令），每当达到有效扫描遍数时，它将引起 SID 停止测试程序的执行，但它和临时断点不同，即在执行期间当到达扫描遍数时扫描遍数指针值不会自动清除。此外和临时断点的不同还在于扫描遍数间断发生在作为扫描遍数指针地址的指令执行完之后。由此可见，在 G 命令控制过程中，你能单纯地连续执行测试程序，直到下一个扫描遍数有效时为止，或到达临时断点为止。

每一个扫描遍数都能任意选择一个补缺值为 1 的“扫描遍数计数”，这个扫描遍数计数允许在实际间断发生前多次地扫描到扫描遍数指针，从而增强了灵活性。具体说，在 1～FF（十进制 1 到 255）范围内与特定的扫描遍数相联系，每当在扫描遍数内指令被执行一次时，对应这个减 1 过程一直进行到扫描遍数计数到 1 为止。在这同时打印间断地址和停止执行测试程序。当扫描遍数计数到达 1 时，扫描遍数指针成为一个每当此指令执行完时都停止执行的永久间断地址。需要指出，扫描遍数计数一旦达到 1 时，它就不能再改变，在任一特定的时间里八个不同的扫描遍数指针都能有效地置位。

形式（a）是在地址 P 处置扫描遍数，同时扫描遍数计数为 1，致使地址 P 变成一个永久断点。形式（b）类似，所不同的是扫描遍数计数被预置为 C。形式（c）用于显示这些有效的扫描遍数点，用这种格式：

```
cc  pppp  .sssss
```

其中 cc 是扫描遍数计数的十六进制值，它与当前的扫描遍数指针地址 pppp 相关，sssss 是与地址 pppp 相匹配的符号（如果符号存在的话）。

形式（d）清除地址 p 的扫描遍数指针，而形式（e）清除所有有效扫描遍数指针。注意，命令：

```
Pp, O
```

是和形式（d）等效的。

每当碰到扫描遍数指针时，SID按如下格式打印扫描遍数信息：

```
cc  PASS  pppp  .sssss
```

cc 是在扫描遍数指针 pppp（当大于 1 时，cc 就减 1）上的当前扫描遍数计数。如上所述，sssss 是在可能情况下与被打印的地址 pppp 对应的符号。

专用命令形式“-G”和“U”如同计数器被减到 1 一样禁止中间的扫描遍数跟踪。例如，假设 TYPEOUT 子程序是在测试下的部分程序，如上述 G 命令所示，发出命令：

```
P  .TYPEOUT, #30
```

P 命令把扫描遍数指针置在以“TYPEOUT”为标号的单元上，“TYPEOUT”被假设已存在于符号表中。扫描遍数计数置为十进制 30，也就是在产生间断前允许程序执行到扫描遍

数指针 30 次。给出扫描遍数指针在 TYPEOUT 上有效后命令：

```
G
```

开始执行没有临时断点的测试程序。扫描遍数指针每执行一次，紧接着跟踪就打印一次。

```
1E  PASS  0302  ·TYPEOUT
 (寄存器跟踪)
1D  PASS  0302  ·TYPEOUT
 (寄存器跟踪)
1C  PASS  0302  ·TYPEOUT
 (寄存器跟踪)
    · · ·
01  PASS  0302  ·TYPEOUT
 (寄存器跟踪)
  * 303
```

“寄存器跟踪”表示在 TYPEOUT 地址上的“MOVE, A”指令执行前 CPU 寄存器的状态（寄存器显示格式见 X 命令）。需指出，最后的断点地址是 0303，它是跟在扫描遍数指针地址 0302 的“MOV”指令后的一个地址。在跟踪扫描遍数指针时，按下任一符号键，SID 立即停止执行接在扫描遍数指针地址后面的指令。如果发出的命令是：

```
-G
```

在控制台上，中间的扫描遍数指针跟踪显示不出现，个别情况下，仅最后的跟踪

```
01  PASS  0302  ·TYPEOUT
 (寄存器跟踪)
  * 303
```

被打印。虽然中间的跟踪扫描遍数指针不被显示，但也能用按字符键的方法停止执行。如果中间扫描遍数指针与禁止跟踪相遇，则 SID 停止执行并且控制返回到键盘。

当扫描遍数指针有效时，临时断点也能被设置。也就是说，这样一类的命令：

```
Ga, b,
Ga, b, c
G, b
G, b, c
```

会同 P 命令置位的永久断点发生混杂。然而值得指出，在永久断点和临时断点出现在相同地址的时候，永久断点优先于由 b 和 c 指定的临时断点。另外，当永久断点有效时，T 和 U 命令能跟踪测试程序段。在该情况下，扫描遍数计数象上面所说的递减，当计数到 1 时，同时引起间断。

有效 P 命令如下所示：

```
P 100, FF
P.BDOS
P@ICALL+30, *20
-P.CRLF
```

1.11 读代码/符号（R）命令

R 命令连同 I命令一起读程序段、符号表和实用功能程序到暂驻程序存储区中。R 命令形式如下：

(a) R

(b) Rd

I 命令建立起的文件名将被读操作使用。形式 (a) 是读取 I命令提供的程序和/或符号表，这时装入地址没有使用偏移。形式 (b) 是加偏移值 d 到每一个程序的装入地址和/或符号表地址。注意，这个加法设有溢出检验发生以便能应用负偏移值。作为一个简单的情况，通常启动 SID：

A>SID X .COM

能用下面命令序列替换：

SID	没有测试程序下启动 SID
IX.COM	预置输入行
R	读内存测试程序

在这种情况下，SID 的响应正好和正常预定的相同，即给出前面第一节叙述的"NEXT PC END" 信息。

每个程序和符号文件都能用一个先于 R 命令的 I 命令形式读出

IX.Y U.V

X.Y 是被装入的程序，U.V 是符号表文件。注意，Y 通常显示"COM"；X 通常和U一样，并且 V 通常显示 "SYM"。于是一个典型的等效命令形式可显示如下：

I DVMP.COM DUMP.SYM

R

这个命令序列读 DUMP.COM 程序文件进入暂驻程序区，并且把由 DUMP.SYM 提供的信息装入符号表中。做为 "HEX" 文件类型的程序则装入到由 Inter 十六进制格式记录指定的单元中。而作为 "UTL" 文件类型的 SID 实用功能程序其装入和再定位是自动的，所有其他的文件类型作为绝对代码程序从暂驻区基底开始装入。当实用功能程序再定位时，自动移走任何存在的符号信息，但在其他情况下，符号的装入操作是渐增的，特别是这种专门的输入形式：

I* U.V

R

因为在程序名处有一个星号表示没有程序装入，装入的仅是符号文件。因此，上述形式的命令序列可以装入符号表，此符号表是在小模块上被最初产生的一个长程序的选择部分。

例如，假设用 MAC 已产生一个报表生成程序，由下述模块组成：

IOMOD.ASM	I/O
SORT.ASM	文件分类模块
MERGE.ASM	文件归并模块
FORMAT.ASM	报表格式模块
MAIN.ASM	主程序模块
DATA.ASM	公用数据定义

进一步假设，每一个模块都已用 MAC 单独汇编，结果形成几个相应于各个程序段的"HEX"

和“SYM”文件，这个已被装配在一起的程序段由 SID 形成一个内存映象被打印成命令序列：

```
SID                    启动 SID 程序
I  IOMOD.HEX           预置 I/O 模块
R                      读 I/O 模块
I  SORT .HEX           预置分类模块
R                      读分类模块
I  MERGE .HEX          预置归并模块
R                      读归并模块
I  FORMAT .HEX         预置格式模块
R                      读格式模块
I  MAIN .HEX           预置主模块
R                      读主模块
I  DATA .HEX           预置数据区模块
R                      读被预置的数据
```

按照这个序列，暂驻程序区包含有完全的报表生成程序的内存映象，假设接着上一个 R 命令打印出信息：

```
NEXT  PC  END
1B3E  0100  8E00
```

表明内存高地址是 1B3E，使用 H 命令：

```
H1B
```

你会发现 1B（十六进制值）页面和 27（十进制）页面一样：在这一点无论用 control-c（热起动）还是用“GO”命令都能返回到 CCP 方式，而留下一个完整的内存映象。然后发出命令：

```
SAVE  27  REPORT  .COM
```

在磁盘上建立起一个内存映象文件，接着利用下述命令重新进入 SID：

```
SID  REPORT  .COM
```

以装入完整的测试模块，报表生成程序的特定部分能调用原先模块所装入的符号表中的符号。例如，MAIN 和 SORT 模块能够被其后装入的相应符号信息来调试：

```
I* MAIN .SYM
R
I* SORT .SYM
R
```

这些符号信息是为今后的调试做准备的。若在 SORT 模块中发现错误，则 SORT .ASM文件被编辑而做必要的修改，并且用 MAC 模块重汇编，结果仅对SORT模块产生新的“HEX”和“SYM”文件。若在 SORT 模块下面提供了充分的扩展区后重新启动 SID，而 SORT 模块包含：

```
SID  REPORT .COM
I  SORT .HEX  SORT .SYM
R
```

在初始报表生成程序内存映象中，则由改变的SORT模块所复盖。这时你可用打印I和R命令装入附加的符号表中，例如：

```
I* MAIN .SYM
R
I* DATA .SYM
R
```

以便在SORT、MAIN和DATA模块中存取这些符号。需要指出，各符号表文件都能在SID调用前利用PIP程序（关于PIP操作看CP/M特性和设备手册）连接起来。例如PIP命令：

```
PIP NOBUGS .SYM=IOMOD .SMY,SORT .SYM, MERGE .SYM,
    FORMAT .SYM
```

生成一个叫做NOBUGS .SYM的文件，而NOBUGS .SYE是用来保存IOMOD、SORT、MERGE和FORMAT的符号。SID命令：

```
SID REPORT .COM NOBUGS .SYM
```

装入报表生成程序的内存映象和这些特定模块的符号表，在这之后附加的符号文件能利用I和R命令选择装入。于是能用PIP命令构成完整的内存映象的符号文件：

```
PIP REPORT .SYM=NOBUGS .SYM, MAIN .SYM, DATA .SYM
```

这时你可打印：

```
SID REPORT .COM REPORT .SYM
```

以装入报表生成程序的内存映象和全部符号表。可是，由于符号表总是按装入顺序进行检索的，这样对在二个模块中相同的符号名则必须用限定符号名来区别（见第一节）。如上所述，形式（b）给出被加到每一个程序地址和符号值上的偏移量d，然而当程序是一个SID实用功能程序时，位移量无效（文件类型"UTL"）。例如命令：

```
IDUMP .HEX DUMP .SYM
R 1000
```

是把DUMP程序装入到起点比规定值高1000（十六进制）的单元中，同时符号地址适当地被调整。注意，任一个偏差值都可能是一个符号表达式，于是命令

```
R-200
```

首先产生一个负数（2的补码），这个负数加到每一个地址中。鉴于从16位计数器中的溢出是不予考虑的，所以R命令把程序装入在规定地址以下200（十六进制）的单元中，同时符号地址也偏移相同的数值。

在执行R命令期间，报告出错限定用规定的"？"应答，表明程序或文件不存在，或者程序或符号文件是错误格式。同SID启动信息类似，在程序装入后，符号装入前发出应答：

```
SYMBOLS
```

因此在应答SYMBOLS之前的出错符"？"表明错误出现在程序装入期间，而在应答SYMBOLS信息之后的"？"，表明错误出现在符号装入操作期间。符号文件出错的准确位置能被后面接着使用的H命令找到。H命令是用于检查实际上已被装入的这段符号表。

1.12 置内存（S）命令

S命令使数据能送入主存，S命令的形式是：

(a)　　Ss

(b)　　SWs

形式（a）使数据能在单元S处以字节（8位）或字符串方式送入，而形式（b）是以数据项为字（16位）方式存储。不论发生那种情况，SID程序都用从单元S开始的连续地址及被定位在该对应地址的数据项提示给控制台。当每行提示出现时，你可以按一单个回车或符号表达式（后面跟一回车），符号表达式被求值后使该单元中换成了新的数据项。如果单打印回车，那么该单元中的数据元素保持不变。每当检测到一个无效数据项或单打印一个“.”后面跟一个回车时，S命令终止。形式（a）允许单一字节数，因而当高位字节为非零的一个双倍字节值被送入时产生标准的“？”。此外，形式（a）也允许送入长的ASCII码串数据。用格式：

ccccc . . . ccccc

在这里序列c（用回车结束）表示：图形符号的ASCII字符送入到被提示的单元中。在输入期间不产生从小写到大写的转换。此外，下一个被提示的单元自动地设置到输入字符串后第一个未填入的单元。

跟在命令：

S 100

后的一个有效输入序列表示如下。在此，SID的提示在左边给出，而操作员的输入行显示在右边，“cr”表示回车键。

SID	提示	操作员输入
0100	C 3	34 cr
0101	24	*254 cr
0102	CF	cr
0103	4 B	“ASCII cr
0108	6 E	=X+5 cr
0109	E 2	‘%’ cr
010A	D 4	.cr

跟在命令

SW.X+*30

后的一个有效双字节输入序列如下所示：

SID	提示	操作员输入
2300	D 06D	44 F cr
2302	4 F 32	@GAMMA cr
2304	33 E 2	cr
2306	FF 11	.X+=I-*20 cr
2308	348 F	.cr

1.13 跟踪方式（T）命令

T命令对一个测试程序做单步或多步操作，这时CPU各寄存器的内容也随着它们而变化。此外，可以用带有SID实用程序的T命令收集测试程序的数据，作为后面显示之用（见“SID实用程序”一节）。T命令形式如下：

(a)　Tn
(b)　T
(c)　Tn, C
(d)　T, C
(e)　-T（同a-d任选项）
(f)　TW（同a-d任选项）
(g)　-TW（同a-d任选项）

形式（a）从程序计数器PC的当前值执行跟踪程序（PC值和CPU状态的显示格式相同。见X 命令）。形式（b）是（a）的一般情况，即采用n=1单步计数的（a）命令。不论是哪种情况，SID程序都在每条指令执行前显示寄存器状态及被译码的指令（假设"-A"是无效的）。例如，命令：

T 4

跟踪程序4步，产生的格式如下：

（寄存器状态1）　操作码1
标号：
（寄存器状态2）　操作码2
标号：
（寄存器状态3）　操作码3
标号：
（寄存器状态4）　操作码4• bbbb

各相应的操作码被执行前显示寄存器状态，各操作码写入的格式和L、X命令一样。只要有可能'被引入的符号操作数也被译码。此外，访问内存指令，如INR　M同内存操作数一起被列表如下：

opcode　M=hh

"opcode"是访问内存的指令操作码，hh是包含在操作发生前由HL寄存器提供的内存地址中的十六进制值。当在执行程序的流程中出现被引入的标号时，它指出程序的地址。被上面"*bbbb"指示的最终间断地址表示操作码4被执行后程序计数器的值。在此，你能用打印单一字符X命令来显示CPU的状态。

形式（c）和（d）仅仅是和SID的实用程序一齐使用，并且在每一条指令执行以后自动地完成CALL　C，值C是一个数据收集实用程序入口地址。下面一节将详细叙述这些形式。然而，要注意形式（d）相当于（c）在计数n=1单步操作的情况，

（e）提供的形式类似于（a）到（d），但对上述的负号命令，SID的符号特性不适用。具体指符号操作数和符号标号在跟踪过程中都不被译码。这种方案可稍许提高在出现长符号表时SID的跟踪操作。

由（f）提供的形式类似（a）到（d），但是要执行"不请求跟踪"的功能它常是有用的。例如，在只跟踪主入程序而不跟踪在执行主程序时被调用的子程序时，TW命令是如此完成功能的：即每当进入子程序时，实时运行测试程序，一旦返回到当前子程序级就立即回到全跟踪方式。如果返回操作发生在当前级（即在全跟踪方式下执行RET），那么跟踪仍保留在完成的子程序或主程序级处。例如，假设主程序和子程序结构如下所示存在于一个特定

程序中：

```
主程序          子程序1          子程序2    . . .    子程序n
 . . .          S1:MOV A,C       S2:MOV A,D          Sn:MOV A,L
CALL S1         . . .            . . .               . . .
MOV B,C         CALL S2          . . .               . . .
MOV C,D         MOV C,E          CALL S3    . . .    MOV C,L
. . .           MOV D,E          MOV D,H             MOV D,L
. . .           . . .            . . .               . . .
JMP 0000        RET              RET                 RET
```

进一步假定在调用子程序S2前，在子程序S1内停止这一测试程序的执行时，有命令：

T*100

跟踪从S1到S2，S3，等等直到碰到Sn级为止。虽然，这种跟踪方式可能是有用的，但仅对一个特定子程序级跟踪。仅观察上面S1子程序的功能，使人更清楚一些。在这种方式下，由于对不必要的（次要的）程序流程不进行跟踪，这样错误的子程序就很容易被发现。如果在S1子程序处打印如下命令：

TW*100

那么S2以下各级子程序都将实时地被执行，如同在S1中每次到CALL处都已执行了一次"G"命令。一旦在S1内执行RET指令，跟踪就回到主程序级，在主程序级CALL S1以后的任何子程序调用都不再继续跟踪。

由（g）提供的形式和（a）到（d）相似，只是在和（e）形式相同的方式中SID的符号特性不适用。

注意，SID允许跟踪达到只读存储器（ROM）中的程序代码。在进入ROM时，SID停止跟踪操作，实时运行ROM程序，而在ROM程序出口处自动设置一个中止程序控制的断点。然而，假设ROM机器码是经由子程序调用(CALL或RST n)进入而不是经PCHL或JMP指令进入的，无论如何，在ROM程序执行后的返回地址总是当成测试程序堆栈中的临时地址。

此外还要注意，SID不能跟踪调入BDOS程序的执行，因为这些操作往往很长，而且偶尔还需要执行各种磁盘功能的实时操作。因此，BDOS的进入被SID中断，然后紧接着在完成BDOS功能之后重新开始。要在任意时间停止跟踪，可按一个字符键，而不用RST指令去终止跟踪功能。

有效的跟踪命令表示如下：

```
T 100
T*30, .COLLECT
-TW=I, 3E03
```

1.14 不跟踪方式（U）命令

U命令和上面给出的T命令相似，所不同在于其每一步不是显示CPU的寄存器状态。而测试程序在全监控下运行，以后可以在任何时间停止程序的执行，或对SID实用功能程序用作为数据的采集。U命令形式与T命令相似：

(a) Un
(b) U
(c) Un, c
(d) U, c
(e) -U (用a-d任选项)
(f) Uw (用a-d任选项)
(g) -Uw (用a-d任选项)

形式(a)到(d)执行和T命令形式(a)到(d)相似的功能，每一步都不显示寄存器状态。由(e)提供的形式和T命令不同，但仍不能执行符号功能，下面的命令形式：

-Un
-U
-Un, c
-U, c

禁止中间扫描遍数指针值的显示(看"P"命令)，直到相应的扫描遍数计数达到1为止。

由(f)提供的形式除了禁止跟踪外恰好和"T"命令相当。在这种情况下，当前子程序级在完全监控下运行，但较高的子程序级在实时下运行。

由(g)提供的形式和(f)相似，但和上面讨论的一样，禁止对扫描遍数指针值显示。

可以按下任一个字符键来终止不跟踪方式的执行。这个间断地址被显示而控制返回到SID命令方式。

有效的U命令如下：

UFFFF
U*1000,.COLLECT
Uw=CAMMA, .COLLECT

1.15 检查CPU状态(X)命令

X命令允许检查和修改测试程序执行中CPU的状态。X命令形式如下：

(a) X
(b) Xf
(c) Xr

形式(a)显示整个CPU状态，格式如下：

CZMEI A=aa B=bbbb D=dddd H=hhhh S=ssss P=pppp OP SYM

这里C、Z、M、E和I分别表示CPU的进位、全零、负、奇偶校验和交错进位的真假状态。如果在位置上包含一个"—"，那么相应的标志就是为假，而别的标志字母被打印出来。字节值aa是A寄存器的值而双字节值bbbb、dddd、hhhh、ssss和pppp分别是BC、DE、HL、SP和PC给出的16位值。被记作"OP"的字段给出了在pppp单元上被译码的助记符指令。在这种情况下，打印出操作码的十六进制值，除了"-A"是有效的外，在可能的情况下，"SYM"字段包含一个被译码的操作数。关于符号指令译码格式，参考L命令。单个字母"X"的命令可产生下面格式的显示：

C-M--A=03 B=34EF D=2000 H=334E S=4323 P=0100 LDA0223.Q

上例指出进位和负号标志为真，而零、奇偶校验和交错，进位标志为假。此外，A寄存器中存

有 03，而 B、C、D、E、H 和 L 寄存器中分别存的是十六进制值 34、EF、20、00、33 和 4E。堆栈指示器 SP 中为 4323，而程序计数器 PC 是指在 0100 单元，在 0100 单元上是下一条要执行的指令取 0233 中内容存入累加器（LDA）。另外，Q 是与地址 0233 在符号表中相匹配的第一个符号。

形式（b）允许改变 CPU 标志状态，在这种情况下，f 必须是状态码字母：C、Z、M、E 或 I 中之一。不论标志字母是真还是"—"（如是假），现在的状态标志均被显示。你可打一个单个回车，以保留现有的标志状态；也可以打一个 1，置标志为真；或打一个 0，使标志复位为假。如已给出进位标志为真，则打命令：

```
XC
```

产生 SID 回答：

```
C
```

后面跟一个空格，表示进位标志目前被置位，等待一个可能的变化。打入一个回车以保留这个标志或打入 0 就把进位标志复为假。同样地，如零标志为假，打命令：

```
XZ
```

产生 SID 回答：

```
----
```

指出零标志为假。如果要保持状态不变就打回车或打 1，以置零标志为真。

形式（c）允许代换单个 CPU 寄存器的内容，这里的 r 是寄存器名字 A、B、D、H、S 或 P 中的一个。在此情况下，寄存器当前内容被显示，同时控制台上提醒用户输入。如打入单个回车则寄存器数据值保持不变，否则，符号表达式被求值，并改变寄存器为新值。当用 XA 形式时仅是字节值可能被接收，而用其他形式时接收双字节值。需要指出，BC、DE 和 HL 必须作为寄存器对来改变，当 A 寄存器内容被修改时，典型的 SID 变换表示如下：

```
XA
A=03  45  cr
```

这里你打入"XA"，SID 印出 A 寄存器的值为 03，接着你打入"45"以代换 A 寄存器的值。本例和下例中 cr 都表示回车键。用上述的格式再举几个用 SID 提供的寄存器代换的例子：

```
XB
B=34 EF   cr（数据保持不变）
XD
D=2000  2300  cr（D 变换为 23）
XH
H=334 E .GAMMA  cr
XS
S=4323  @STKPTR+*100  cr
```

第二节　SID 实用程序

SID 实用程序是用 SID 操作的专用程序，以便提供附加的调试手段。象第一章第一节所讲的那样，打入下面命令就可装入 SID 实用程序

SID X .UTL

这里X是一个将要叙述的实用程序的名字。在启动时，装入实用程序，再定位并用某些需要的参数提示控制台，然后收集需要的程序测试数据（用U或T命令），再调用实用显示子程序来显示信息。关于系统初始化、数据收集和显示的结果在下面详细给出。

2.1 实用程序操作

一个专门的SID实用程序和常规的测试程序一样装入到内存中。但正如前面所述，实用程序自动地移动它自己到内存的高地址，恰好占据了SID程序下面的区域。用上面给出的SID命令，仅打入实用程序名，就可完成实用程序装入操作。也可以按I和R命令所述的方法，在SID执行期间装入实用程序。但当实用程序装入时，所有存在的符号信息都被移走。因此，在需要调试运行时必须进行重新预置。

通常，一个SID实用程序有三个主入口：实用程序初始化入口INITIAL、数据收集入口COLLECT和数据显示入口DISPLAY。装入实用程序后，在符号表中建立了这些符号，并以如下格式打印出入口地址：

.INITIAL = iiii
.COLLECT = cccc
.DISPLAY = dddd

这里iiii、cccc和dddd是三个入口的十六进制地址。但要指出，这三个符号名和这三个地址是等效的。

继初始信号后，实用程序可以用附加的调试参数提示控制台。在交互作用完成后，你可以用I和R命令装入测试程序和符号表，以便开始进行调试对话。

在调试运行期间，由于用U或T命令在监控方式下运行测试程序因而产生数据收集。下面任何一个命令

UFFFF, .COLLECT
UFFFF, CCCC

都控制SID程序从当前程序计数器值开始去运行测试程序达最大65535(十六进制FFFF)步，随后转到实用程序的数据收集入口。每一个程序断点把信息送给实用程序，在实用程序中为下面的显示进行制表。注意，在这个特定情况下，可以在连续的65535步未做完前按回车键来停止不跟踪方式。

下面是一组数据收集操作，送入任何一个调用实用程序DISPLAY入口的命令：

C. DISPLAY
C dddd

都可打印输出积累存储的数据，而紧跟在附加的显示操作之后，重新开始如上所述的数据收集过程。

在任一点，为了重新初始化程序可以打入任何一个下述命令：

C.INITIAL
C iiii

它产生实用程序符号表的重新预置，而为了完成重新初始化实用程序产生对附加参数的提示。

注意，在调试对话期间装入和执行超过一个实用程序的功能时，会产生无法预料的结果。

本节下一部分说明SID实用程序的功能。

2.2 HIST 实用程序

HIST 实用程序产生一个在测试程序被选择的程序段上执行时相对出现次数的直方图（柱状图形），HIST 实用程序使你能够监视测试程序时的“过热点”（“hot spots”）在那里程序执行最频繁。

和上节叙述的一样，在初始信号之后，HIST 实用程序提醒控制台输入。

```
TYPE HISTOGRAM BOUNDS
```

你必须回答两个用逗号隔开的符号表达式：

```
llll, hhhh
```

这里 llll 是监视的最低地址，而 hhhh 是最高地址。为了收集直方图信息，必须使用下面的命令形式之一：

```
Tn,c  T,c  TWn,c  TW,c  -Tn,c  -T,c  -TWn,c  -TW,c
Un,c  U,c  UWn,c  UW,c  -Un,c  -U,c  -UWn,c  UW,c
```

这里 c 不是.COLLECT，就是相应于.COLLECT 的入口地址。虽然这些命令中的任何一个都是可用的，但形式

```
Un, .COLLECT
```

是最经常用的，因为它的跟踪输出被禁止，测试程序完全被监控，以及在每一个程序步都产生数据的收集。

下面是一组数据收集的操作，打印：

```
C.DISPLAY or Cdddd
```

显示直方图，接着按下列格式打印出直方图：

```
直方图
  地址    相对出现次数，最大值 = mmmm
  aaaa     * * * * *
  bbbb     * * * * * * *
  cccc     * * * * * * * * *
  . . .
  xxxx     * * * * * * * * * * *
  yyyy     * * * * * * * * * * * * * * * * * * * * * * * *
  zzzz     * * * * * *
```

这里地址 aaaa 到 zzzz 的变化范围是在 HIST 预置时给出的从低地址到高地址的范围。最大值 mmmm 是在任一显示地址上被累加的指令最大数，而星号代表了指令相对出现次数的直方图。根据最大值 mmmm 决定的比例尺度，地址范围被自动地分度出 64 个不同的插入地址（如上所示的 aaaa,bbbb,. . .,zzzz），在任何一个实际的直方图形中的星号最多为 64 个。

上面显示出的“过热点”（“hot spot”）在地址范围 xxxx 到 zzzz 前后，这时可打印下面任何一个命令，以重新初始化 HIST 实用程序：

```
C. INITIAL
C iiii
```

HIST 初始化后做如下提示和回答：

TYPE HISTOGRAM BOUNDS XXXX ZZZZ

接着你可用下面命令重新运行测试程序：

UFFFF, .COLLECT

在测试程序达到稳定状态并有足够的时间之后，在被监控执行的期间内按一个回车能中断程序的执行。然后对 XXXX 到 ZZZZ 之间的扩展区重新请求显示功能，使得频繁执行区 的观察得到更精细的结果。

随后用 L 命令可以精确测定最经常被执行的一些指令。如果必要，该指令序列也可稍微做一些改进，作为程序执行时一般的修正。

2.3 TRACE实用程序

TRACE 实用程序在获得测试程序时产生一实际间断点指令的向后跟踪。例如程序可有一个这样的错误状态，它是从正常测试下很难找到的指令序列中产生的，在这种情况下，TRACE 可以在执行测试程序时收集程序地址，并当你请求时按最近到最后的新次序显示这些地址和指令。为了用 TRACE 实用程序请求 SID，打入下列命令：

SID TRACE .UTL

实用程序回答如下：

INITIAL = iiii
COLLECT = cccc
DISPLAY = dddd

在此情况下，TRACE 实用程序也打印信息：

READY FOR SYMBOLIC BACKTRACE

这表明 SID汇编/反汇编的任选项被提供，且当向后跟踪被请求时将对指令反汇编。

接着你可以用任选项的符号表装入一个测试程序。例如，可用打印如下命令装入DUMP程序：

IDUMP .COM DUMP .SYM
R

通常回答：

"NEXT PC END"

表明测试程序已被装入。这时，SID 调试程序连同 TRACE 实用程序和测试程序的符号，在内存高地址执行，而在内存低地址的测试程序为执行做好了准备。

为了获得最简单的向后跟踪，打入 HIST 实用程序中的 U 或T命令形式。具体地说，一个 U 命令形式：

U*500, .COLLECT

执行程序 500（十进制）步，然后自动停止。打印下面命令获得对此停止的向后跟踪：

C .DISPLAY

这个命令引起 TRACE 用如下形式显示标号、地址和助记符信息：

```
label—255:
  addr—255  opcode—255  sym—255
label—254:
  addr—254  opcode—254  sym—254
```

```
label—253:
  addr—253  opcoda—253  sym—253
  . . .       . . .        . . .
label—000:
  addr—000  opcode—000  sym—000
```

这里 label—255 向下到 label—000 代表由 addr—255 向下到 addr—000 给出的被译码的符号标号相应的地址（如符号标号存在的话）。opcode—255 向下到 opcode—000 代表与向后跟踪地址相对应的助记符操作码，而 smy—255 向下到 sym—000 相应于最靠近操作码符号的符号操作数(如果有的话)。用命令列表的相同格式显示操作码。注意，在这显示中，必须最先是在单元 addr—255 上被执行的指令，而最后是在 addr—000 上被执行的指令。TRACE 在 T 或 O 方式中共占有积累的 256 条指令，被积累的指令不受 DISPLAY 功能的影响，但是能被调用下述初始化而清除

C. INITIAL

TRACE 和扫描遍数指针共同使用时，TRACE 实用程序是十分有用的（见 P 命令），尤其是扫描遍数指针首先设置在对向后跟踪感兴趣的程序单元时。随着程序被运行到测试开始的中间单元，在这个中间测试点用 U 命令执行全监控方式下的测试程序，同时在 TRACE 的 COLLECT 入口点处收集数据。当碰到 U 方式中的扫描遍数指针之一时，中断程序的执行，并且在 SID 命令方式中可以收回控制。然后请求 TRACE 的 DISPLAY 功能，以获得所需要的向后跟踪信息。

作为处理过程的一个例子，假设如上所述，DUMP 程序和 TRACE 实用程序都在内存中；进一步假设你希望观察在首次调用 BDOS 时 DUMP 程序的作用（通过 0005 单元，从DNMP 到 CP/M 基本磁盘操作系统的首次调用）；假定符号表已被装入，打印命令：

P.BDOS

在相当于扫描遍数计数 = 1 的 BDOS 入口置扫描遍数，然后在监控方式下执行 DUMP。在每条指令上收集数据：

UFFFF,.COLLECT

当然，在此情况下，不跟踪指令计数达 FFFF（65535）条是太多了，但假设在指令计数超出前，DUMP 程序就停在 BDOS 的调用处（假如不是这样，可按任一个字符键，以强迫程序间断）。在这种情况下，DUMP 程序在 BDOS 调用前仅执行了少数几条指令。于是，在间断处产生如下信息：

```
01  PASS  0005 .BDOS
 -ZEI A=80 B=0014 C=005C H=0000 S=0249 P=0005 JMP CCDF
 *CCDF
```

这表示扫描遍数计数为 1，扫描遍数地址为 0005，符号单元为 BDOS，寄存器状态以及间断地址。由于执行到的这一点是被监控的和数据是被收集的，因此请求 TRACE 功能：

C.DISPLAY

在这里显示结果：

```
BDOS:
  0005  JMP  CCDF
```

```
01CA  CALL  0005 .BDOS
01C8  MVI   C,OF
01C5  LXI   D,005C .FCB
01C2  STA   007C  .FCBCR
SETUP:
01C1  XRA   A
010A  CALL  01C1  .SETUP
0107  LXI   SP,0257  .STKTOP
0104  SHLD  0215  .OLDSP
0103  DAD   SP
0100  LXI   H,0000
```

注意，在这特定情况中，BDOS 在调用前仅执行了 11 条指令，因此全部 256 条指令容量并没有超过。事实上，上面所示的向后跟踪给出了从地址为 0100 的第一条指令执行 DUMP 的全部过程。要继续执行 DUMP 程序，只需打印：

```
UFFFF,.COLLECT
```

在下面调用 BDOS 时，每次都产生一个断点，而被给出的程序执行到调用 BDOS 20 次时停止。为此，打入扫描遍数点命令：

```
P BDOS,*20
```

置扫描遍数计数为 20（十进制），再送入命令：

```
UFFFF,.COLLECT
```

若中间扫描遍数指针被跟踪，相反地打入命令：

```
-UFFFF,.COLLECT
```

禁止中间跟踪。在任何一种情况下，在调用 BDOS 第 20 次时停止执行，并可送入显示命令：

```
C .DISPLAY
```

以观察跟踪这个特定的 BDOS 调用的情况。

在显示期间，按任一字符键都会终止打印输出。ctl-s 键是为在输出期间暂停输出用的。最后，你可发出"C .DISPLAY"命令任意次，以重复产生向后跟踪，因为这个命令不清除 TRACE 缓存器。

当没有反汇编模块时，也能使用 TRACE 实用程序。在此情况下，在跟踪时指令地址被列表，而不包含助记符。例如，下面所示的命令序列装入了 TRACE 实用程序，但没有反汇编模块。后面装入 DUMP 程序，而没有符号表：

```
SID                  装入 SID 程序
-A                   消除反汇编
I TRACE.UTL          TRACE 实用程序准备就绪
R                    读 TRACE 实用程序
IDUMP. COM           装入 DUMP 程序
```

在这种情况下，TRACE 实用程序打印出下面的符号信息：

```
"-A" IN EFFECT, ADDRES BACKTRACE
```

向后跟踪信息接着按如下格式被显示：

```
addr—255 addr—254 addr—253 . . . addr—248
addr—247 addr—246 addr—245 . . . addr—240
          . . .
addr—007 addr—006 addr—005 . . . addr—000
```

第三节　SID调试对话实例

这一节包括了SID调试会话的几个实例。这些实例以一个字节值列表的"上推分类法"为基础。这个上推分类法程序首先在它的非调试方式中被列表。一系列的测试、编辑和重汇编过程的结果在最后的调试程序中被显示出来。在每种情况下，操作员随着CP/M、ED、MAC或SID的相互作用规定的符号显示。过程的每一个注释用斜体字并排地给出：

下面的对话包括如下操作序列：

(1) TYPE SORT .PRN	列表初始分类程序
(2) TYPE SORT .SYM	显示分类符号表
(3) TYPE SORT .HEX	显示分类HEX文件
(4) SID SORT.HEX SORT.SYM	第一次调试对话
(5) ED SORT.ASM	分类程序的第一次重编辑
(6) MAC SORT	分类程序的第一次重汇编
(7) TYPE SORT.SYM	显示新的符号表
(8) SID SORT.HEX SORT.SYM	第二次调试对话
(9) ED SORT.ASM	分类程序的第二次重编辑
(10) MAC SORT	分类程序的第二次重汇编
(11) SID SORT.HEX SORT.SYM	第三次调试对话
(12) ED SORT.ASM	分类程序的第三次重编辑
(13) MAC SORT	分类程序的第三次重汇编
(14) LOAD SORT	为分类程序生成COM文件
(15) SID SORT.COM SORT.SYM	第四次调试对话
(16) SID SORT.COM SORT.SYM	为调试重进入SID
(17) SID SORT.COM SORT.SYM	为调试重进入SID
(18) SID SORT.COM SORT.SYM	为调试重进入SID
(19) ED SORT.ASM	分类程序的第四次重编辑
(20) MAC SORT	分类程序的第四次重汇编
(21) SID SORT.HEX SORT.SYM	第五次调试对话
(22) ED SORT.ASM	分类程序的第五次重编辑
(23) MAC SORT	分类程序的第五次重汇编
(24) SID SORT.HEX SORT.SYM	第六次调试对话
(25) ED SORT.ASM	分类程序的第六次(最后一次)重编辑
(26) MAC SORT $+S	第六次（最后一次）重汇编

下面调试对话给了最终调整好的分类程序。

三个独立的调试对话用来表示用 HIST 和 TRACE 实用程序监控被测试的分类程序的运行。它包含如下的操作：

```
(27) SID HIST.UTL          装入 HIST 实用程序
(28) SID TRACE.UTL         装入 TRACE 实用程序
(29) SID                   装入 SID 程序，继续跟踪
```

下面的例子是在一次调试对话之后，列表调用 BDOS 的简单程序。在调用 CP/M BDOS 之后，这个例子表示当子程序被跟踪时 SID 的作用。这时的操作如下：

```
(30) TYPE IO.RRN           列表 I/O 程序
(31) SID IO.HEX IO.SYM     进入 SID 调试
```

（编译校对　高忠麟　康兴钨）

MAGIC WAND 字处理程序使用手册

第一章 引 言

MAGIC WAND 是一个功能很强的字处理系统，它在CP／M 操作系统下运行，可提供丰富、灵活的编辑特性和打印特性，使计算机成为一台文字处理机。

在CP／M下使用 MAGIC WAND的步骤如下：

首先，制备 MAGIC WAND的工作盘：

1．用 FORMAT 程序对空盘进行格式化。

2．用 PIP 或 DUP 程序将 MAGIC WAND 程序拷贝到经过格式化的空盘中。

3．用 SYSGEN 程序作系统生成，制备可启动的工作软盘。

然后，对 MAGIC WAND 进行用户化，即按照硬件配置，用 CONFIGUR 程序对系统配置参数进行修改。

提供的 MAGIC WAND 盘包含下列程序：

EDIT.COM	(1)
PRINT.COM	(1)
CHANGE.COM	(1)
PRNT 1640.HEX	(1)
PRNT 630.HEX	(1)
REPLIES.GET	(2)
SALUTES	(2)
SAMPLE1～SAMPLE8	(2)

为防止出现磁盘满的错误，可将上述程序分别拷贝到“程序盘”（包含上述标识为1的程序）及“文件盘”（包含上述标识为2的程序）中。完成上述工作之后，便可以开始运行 MAGIC WAND 程序。

第二章 编辑特性

第一节 装入编辑程序和文本文件

1. 装入编辑程序和文本文件的完整格式为：

d：EDIT d：Filename {d：Newfilename}

其中 d ：为驱动器名，缺省为当前联机磁盘。Filename为源文件名（缺省为源文件同名）。例如，从当前磁盘装入SAMPLE1文件，编辑后保存在B驱动器上，可使用如下命令格式 EDIT SAMPLE B：。

2. 装入EDIT之后打印出注册信息屏幕，包含诸如版本号和所使用系统的系列号。

3. 打印注册屏幕之后，在继续运行之前可以更换磁盘。也就是说用户在编辑期间无需再从盘上装入EDIT。

4. 当已装入正确的盘后，按RETURN键。

5. 若在这时需要中止编辑，例如新文件名打错，可不按RETURN而按ESCAPE键返回CP／M。

6. 按RETURN键后，驱动器上的盘可以读写。

因此在以后切勿改变驱动器上计划保存经编辑文件的磁盘。例如，已编辑文件保存在驱动器B上，可以更换驱动器A上的磁盘任意次，但不能更换驱动器B上的磁盘。

7. 如果“文件名”不在指定的驱动器上，则系统询问是否为新文件。

8. 如果“文件名”是新文件，按Y键，计算机为其在磁盘目录表中开辟一位置。

9. 若“文件名”不是新文件，按N键，计算机中止编辑返回CP／M。

第二节 文件的命名

文件名由基本名和文件类型组成。基本名可达8字符。除空白、星号、问号或句号之外的任何字符均可作为文件名的一部分。文件名应是有意义的。因此文件名中不要使用除大、小写字母，数字和几种标点符号（如斜杠、连字符、圆括弧和方括弧）之外的其他字符。

文件类型指定是任选的，可达3个字符。可使用与文件名相同的字符，如果有文件类型指定，则基本名后跟以句号，然后是文件类型。例如：INQUIR.LTR、BAK和$ $ $是保留文件类型，不能用在EDIT中作为输出文件类型。若要编辑BAK和$ $ $文件，必须将编辑文件以另一文件名保存起来，或在编辑前重新命名该文件。

第三节 光标移动

1. 在文本屏幕上可控制UP(向上)、DOWN(向下)、LEFT(向左)和RIGHT(向右)键

来移动光标。

2.当按其中的一个键时，光标位置改变但文本不改变。

3.按光标移动控制键，沿指定方向将光标移动一格。但下列情况除外：

a.光标位于屏幕顶行，按 UP 键使文本向后移一行。如果光标已在文本开始行，则重写当前屏幕，但并无改变。

b.光标位于屏幕底行，按 DOWN 键使文本向前移一行。

c.下移光标只能移到超出文本结尾一行，但不会超过该点。

d.光标位于一行的末端时，按 RIGHT 键使光标移到下一行的第一位置。

e.光标位于屏幕末行最后位置上时，按 RIGHT 键除同 d 之外，还使文本向前移一行，除非光标已超出文本末端。

f.光标位于一行的开始，按 LEFT 键使光标移至前一行的最后位置。

g.这时若光标位于屏幕顶行，还将文本后移一行，除非光标已在文本的第一行上。

4.按 TAB 控制键，光标向前跳至下一标记位置。若光标超过一行的最后标记，则跳至下一行的第一位置。

5.按 HOME 键，光标移至所在行的始端。若原光标在一行始端，则跳至屏幕首行的第一个位置；若光标已在顶行的第一位置，则按 HOME 键无效。

第四节　文本移动(滚动)

1.文本文件向前或向后的移动称为滚动。滚动改变文本在屏幕上的位置而不改变光标在屏幕上的位置。

2.FORWARD LINE SCROLL(向前滚动一行)控制键将当前文本在屏幕上向上移一行，并将文本下一行写到屏幕最后一行。光标超过文本末端时该命令无效。

3.BACKMARD LINE SCROLL(向后滚动一行) 控制键将屏幕上的当前文本向下移一行并在屏幕顶加一新行。当文本的第一行已在屏幕顶上时，按该控制键则计算机重写屏幕，但不改变任何内容。

4.FORWARD PAGE SCROLL (向前滚动一页) 控制键清除屏幕并从前一屏幕末行开始重写屏幕。如果文本文件剩余不到一个屏幕，文本最后一行置于光标所在行，重写最后屏幕。

5.BACKWARD PAGE SCROLL(向后滚动一页)控制键清除屏幕并重写文本，使前一屏幕上的第一行成为新屏幕的最后一行。若文本文件开始处剩余不到一个屏幕，计算机自动写入文件开始第一个屏幕的文本。

6.按 TOP (翻页至顶) 控制键显示文本的第一个屏幕，这时光标位于第一行的第一位置上。

7.按 BOTTOM (翻页至底) 控制键显示文本的最后屏幕，这时光标位于文本末后面一行上。

第五节　插入(字符插入和全插入)

1.将字符插入文本有两种方法：字符插入(CHARACTER INSERT)用于插入字符，全插入(FULL INSERT)用于较长的插入。

2.为了使用两种插入控制键，必须先把光标移至要插入字符的前面。例如要将a插入bet使成为beat，将光标移至t上。

3.当光标位于正确位置时，按字符插入键，并键入字符。每次按键，将光标到该行末的所有字符右移一格。这时再按字符插入键，光标停留在同一字符位置上。按其他任何控制键，如光标移动、滚动、后退、清除等键之前，当键入字符时，插入仍有效。

4.插入字符时可将字推向下一行。

5.当要作更长的插入时可使用全插入键。按键前将光标置于要插入的字符前。完全插入清除光标至屏幕底处的屏幕，并将光标后文本行显示在屏幕底的信息行上以供参考。现在即可以键入所要插入的内容。

6.若按SEARCH(检索)、TOP、BOTTOM或ESCAPE控制键，计算机在执行命前结束全插入。

7.不能利用DOWN、FORWARD、LINE、SCROLL或FORWARD、PAGE、SCROLL控制键使光标超过最后插入行的下一行。

8.完成插入之后按END INSERT(结束插入)控制键。结束全插入时，计算机将后面文本填入屏幕的其余部分。

9.若插入量超过屏幕容纳量时，文本向上滚动。这时，计算机取消屏幕底参考行，以免产生混淆。

第六节　删除(字符和行)

1.为了删除单个字符，将光标移至该字符上并按字符删除(CHARACTER DELETE)键。

2.删除字符后，把光标右边该行的所有字符向左移一格(光标不动)。

3.可利用字符删除键删除任何字符，包括控制字符。但仅通过行删除(LINE DELETE)键才能删除回车。

4.按行删除键时，删除自光标至该行末所有字符，并将文本的其余部分向上移至光标处。若连续两次按行删除键，仅删除一行。若光标位于其行的回车符右边，上述两个删除命令均无效。

5.若删除某行的最后一个单词与下行第一个单词之间的空格，则两个单词变成一个单词，这时可能在该行放不下。发生这种情况时，将合并形成的单词写在下一行的开始位置。

第七节　检　索

1.按检索(SEARCH)控制键可以寻找当前光标位置之后的字符串。按SEARCH键时，光标跳至屏幕底信息行上并显示冒号(：)，等待键入检索字符串。

2.键入检索字符串，并按 RETURN 键。

3.计算机检索文本文件，直至找到检索字符串的第一次出现，并重新显示屏幕。这时光标位于该字符串的第一个字符上。

4.重新显示屏幕时，光标位于按检索键前所在屏幕的同一行。例如，开始检索时光标位于第 3 行，计算机向前滚动文本并显示屏幕，使第一个检索字符串出现在第 3 行上。

5.计算机仅能检索光标与文本末端之间的检索字符串。若检索字符串不在光标与文本末端之间，则无法检索到该字符串。

6.除键盘字符外，字符串可以包括 PAGE FEED（换页）、BLOCK MARKER(块标志）和 RETURN 控制字符

7.因为按 RETURN 键开始检索，若检索字符串包含回车时，可用 TAB 键代替，作为字符串的一部分。例如为了检索下一回车，可按 TAB 键，接着按RETURN 键启动检索。

8.若要改正检索字符串，可利用BACK SPACE(退格)控制键将光标左移，重新键入。

9.要中止检索命令，可按 ESCAPE 键。

10.检索并不仅限于一个完整单字，并且检索时有大小写的区分。例如，检索字符串为 the 时，会检索到 there 和 other，但不会检索到 The。

11.要保证字符串是唯一的。如上例中要检索 the，则应置字符串为 “the”，并前后用空格分开。

第八节　检索与替换

1.使用检索与替换 (SEARCH AND REPLACE)命令，检索光标与文本末端之间的检索字符串并以替换字符串取代之。字符串定义同上节所述。

2.键入检索字符串之后不按RETURN键而再按 SEARCH键。这时在信息行检索字符串之后显示第二个冒号（：），并等待键入替换字符。

3.若要删除检索字符串，则按RETURN键而不键入替换字符串。

4.将光标返回至第二个冒号时就停止替换命令。若要重新启动替换命令，须再按 SEARCH 键并再键入替换字符。

5.当检索字符串及替换字符串均正确时，按RETURN键启动检索和替换。

6.接到检索替换命令后，显示屏幕，并置光标于替换字符串之后。

第九节　多重检索与替换

1.多重检索替换（MULTIPLE SEARCH AND REPLACE）完成与检索与替换相同的功能，但可以给定替换次数或指明进行全部替换。字符串定义同上。

2.键入替换字符串后，再按 SEARCH 键而不是RETURN键。这时在替换字符串的信息行上显示第三个冒号。

3.键入检索字符串被替换的次数，并按RETURN键。若要将光标与文本末尾间全部检索字符串均作替换，则按RETURN键而不键入次数。

4.计算机重新显示屏幕，这时光标位于最后被替换的字符串之后。

5.若替换全部检索字符串，在信息行上给出找到的出现次数。当出现数目小于规定的替换数目时，计算机仍打出其出现数目。

6.也可不指定替换次数，而每次只进行一个检索字符串的替换。这须要使用重复检索功能（重复检索控制键）。

第十节　重复检索

按REPEAT SEARCH（重复检索）键使计算机重复进行检索和替换，完成前面所述的检索与替换命令。

若按REPEAT SEARCH 控制键而以前未定义检索字符串，则自动寻找第一次出现的惊叹号（！）。可以通过将惊叹号放在文本中利用 REPEAT SEARCH控制键帮助写出标准化信件（见后面所述）。

第十一节　换页与换行

按换页控制键（PAGE FEED）时，在屏幕上出现'^'，代表控制字符。以后用 ENIT 打印文件时如遇到该控制字符，则打印机向前移至下页的第一行。打印机接收到这一控制字符时，完全如新页命令一样处理（见后面“打印机特性”一章）。

PAGE FEED 控制键还可将文件编排为 INCLUDE 文件格式（见下面“计入命令”一节）。

按 LINE FEED（换行）控制键，屏幕显示'1'，代表控制字符。除使用 EDIT 编写 MICROSOFT BASIC 程序外，无需用换行控制键。

若是这种情况，可以结束程序行而不必结束物理行。为此，按 LINE FEED 控制键并立即跟以回车。

当用户进行程序列表时，回车如通常那样起回车作用；但当执行程序时，LINE FEED 控制字符通知计算机用户不结束当前程序行。

PAGE FEED和 LINE FEED 控制字符可由字符删除控制键来删除。

第十二节　内存溢出

当再输入约 300 个字符就要填满内存时，计算机发出声响警报，在信息行指出内存接近存满。这时，为了继续编辑，需要向磁盘写入一些文件（见后面“写入命令”一节）。当内存接近存满到这个程度时，每次移动光标均重复发出警报和信息。

如果发生内存溢出，只有按 RESET键，重新开始。因为内存溢出是不可恢复的。

第十三节　命令屏幕概述

1.编辑是从命令屏幕开始的。从命令屏幕进入文本屏幕可以按 RETURN 键（不打入命令）。

2.从文本屏幕返回命令屏幕，按 ESCAPE 键。

3.要从命令屏幕给出命令，在键入命令后按 RETURN 键。命令使用大写字母，系统可将小写字母转变为大写字母。

4.若要修改已键入命令，可利用 LEFT、CHARACTER DELETE 或 BACK SPACE 控制键返回光标进行修改。若要取消已键入的命令行，可按 ESCAPE键。

第十四节　命令屏幕状态特性

命令屏幕保持正在编辑的文件状态。在顶行列出所用的所有文件及其状态。

每次在屏幕上可有四种不同的文件：读入（输入）文件、写出（输出）文件、打印文件（用于后台打印）和计入文件（用于并入正在编辑的文件）。命令屏幕列出各种文件名并指出是否是现行的、非现行的或已完成的。

其次，命令屏幕列出在工作区（文本缓冲区）中的文本行数和字符数，以及装入 CP/M 和 EDIT 以后的剩余空间。

在下一行上，命令屏幕给出文本屏幕的当前方式——文本方式、程序方式或特殊方式。然后是文本方式使用的当前行长度。最后，命令屏幕给出 TAB 标记的当前位置。

第十五节　读命令

1.当装入 EDIT时，程序自动地将“文件名”指定的文件读入内存，至少保留 50% 的有效内存用于文本的编辑附加内容。若文件大于有效内存的 50%，其中一部分在第一遍不装入。

2.该文件全部装入之后，计算机在命令屏幕上靠近该输入文件名处显示出“input finished”。

3.若要将该文件多装入一些，可键入R（Read）。最多可装至剩余内存的50%（有效内存的 25%）。每执行一次 READ 命令，为附加内容留下的空间就减少一些。若所需的空间多于附加读入之后留下的空间，或者文本文件大于有效内存，必须利用读命令和写命令的结合（见下面“写命令”一节）。

4.执行 Rn 可指定读入内存的逻辑行数，n是任意整数。注意：实际上内存并不完全满，仍有约 250 个字符的空间，这些空间足够用于存储少量的附加内容（见上面“内存溢出”一节）。

例如，需要对 40000 个字符的文件进行小量修改。装入 CP/M 和 EDIT 之后，仍具有 46000 个字符的有效内存。如果只键入 R 命令，就必须重复该命令数次，才能将全部文件读入内存，若执行 R9999（假设文件不多于 9999 逻辑行），可用一个命令装入全部文件。

第十六节　写命令

1.当给出写（WRITE）命令时将内存中的全部或部分文本写入磁盘上的输出文件。当文本写入磁盘后，从内存中清除。

2.键入 W，从内存中把整个文件写入磁盘。命令 Wn 仅写入文件开始的 n 逻辑行。

3.键入 Wc，从文本的始端写至当前光标的位置，若光标在行中间，则只写到前一行的末端。

实例：

a．假设用户具有40页的编辑文件。

b．装入EDIT 时装入文本文件的开始部分。

c．命令屏幕指出文本文件的输入未完成。

d．进入文本屏幕并对这部分文件进行改正。

e．返回命令屏幕并键入W。

f．计算机将这部分文本写入磁盘并清除内存。

g．键入R。

h．计算机将文本的下一部分读入内存。

i．进行d～h的步骤直到做完全部修改。

j．返回命令屏幕并键入END。

k．计算机将内存中文本写入磁盘，将留在输入文件的任何文本变换至输出文件，并结束编辑返回CP/M。

第十七节　行长度

利用 EDIT 时，行长度（LINE LENGTH）命令在屏幕或打印机上设置可以打印的每行的最大字符数，行长度命令是为在使用 EDIT 时方便而设的，当文件保存在磁盘中或用 PRINT 程序打印时，行长度命令无效。

当用户不指定行长度时，计算机使用用户终端每行最多的字符数（通常为80个字符）。行长度命令格式为在命令屏幕键入Ln，其中n为整数，从2至用户终端每行最大字符数。例如，为了置行长度为50个字符，键入L50。行长度可以改变任意次数，而不影响文本文件。

注：自动字分裂有效时，行的最后字符只能是空格、回车、换页或换行。由于 PRINT 在文件行末端除去空格，行长度为 60 个字符的文本被 PRINT 打印和被 EDIT 打印时，两者看起来不完全相同。

第十八节　标记设置

制表标记有助于在文本屏幕上移动光标而不影响文本文件。用户最多可指定 16 个制表标记。标记最大可置为终端的最大行长度，但是不能超过当前行长度。

命令屏幕指出置标记的列。最初这些列为 1、9、17、25、33、41、49、57、65 和73（注：CP/M 的标准间隔为8）。

改变标记间隔的命令格式为Tn，n为1至80间的整数。例如T5置标记于1、6、11、16等列。

用户可以在指定列设置标记，命令格式为Tn，n1，n2，nx，0；其中n至nx为整数值。例如T5，20，33，50，0将标记置于1、5、20、33和50列。

左页边为第1列，若要在相隔左页边为5和20的空格处置标记，键入T6，21，0命令。

若要将标记置于靠近行始端指定列中，在此后按一定间隔后置时，命令格式为Tn，n1，…

UX。其中Ωx小于或等于前面的数。例如，为将标记置于列6和21，然后相隔10个空格设标记，可键入以下命令：T6，21，10。现在标记置于列1、6、21、31、41、51、61、71。

第十九节　块标志

块标志(BLOCK MARKER)以'-'表示，用于标识被处理的文本块。为了处理文本块，在文本中必须同时设置两个块标志，如果多或少于两个标志，计算机在屏幕上打印错误信息。

设置块标志的方法是将光标移到文本块的第一个字符，并按BLOCK MARKER控制键，然后光标移至文本块后的第一个字符，并再按BLOCK MARKER控制键。

若文本块以回车结束，用户要把回车包括在文本块中，可将标志置在回车之后下一行的第一个字符上。

可用字符删除控制键删除每个块标志。若要删除文件中所有块标志，可进入命令屏幕并键入BK（BLOCK KILL）命令。

块标志不保存在文件内，计算机在将文件写入磁盘时，将消除文件中的任何块标志。

第二十节　字块拷贝

字块拷贝（BLOCK COPY）命令可将文本块拷贝到文件其他位置，留下同一字块的两个拷贝。

首先要用BLOCK MARKER命令标识要拷贝的字块（设立块标志）。再将光标移至要放置拷贝的地方（注：不能将字块拷贝到同一块内，如果光标在字块内将产生错误信息)。然后，进入命令屏幕并键入BC（BLOCK COPY）命令，进行拷贝。当返至文本屏幕时，光标置于拷贝的第一个字符上。字块拷贝的次数受有效内存的限制。当不再需要标识原字块时，用BK命令删除块标记。

第二十一节　字块移动

字块移动命令（BLOCK MOVE）可将文本块移到文件内的另一位置。同样也要首先标识文本块。然后将光标移到要放置移动字块的位置。同上节相同，字块移动不可移到同一块内，若光标在文本块内则产生错误信息。

进入命令屏幕并键入BM（BLOCK MOVE）命令。

当返至文本屏幕时，光标位于移动字块的第一位置上。

由于字块移动取消了块标志，故无需再删除它们。

第二十二节　字块删除

字块删除（BLOCK DELETE）命令可以从文件上删除文本块。首先要为删除文本块设立标志，要保证光标没有留在要删除的文本块内。

进入命令屏幕，键入BD（BLOCK DELETE)命令。这时计算机打印出字块中的字

符数并请求确认用户要执行的字块删除（这可以防止不小心删除）。若要删除则键入Y，否则键入N。当返回文本屏幕时，字块和字块标志已被删除。用户还可利用字块删除功能来确定文本块的字符数。当请求确认时只要回答N即可。

第二十三节　显示命令

使用显示（DISPLAY）命令可以不退出编辑程序而检查磁盘上任何ASCII码文件的内容，它的执行对编辑过程不产生影响。

为显示一个文件，可进入命令屏幕并键入D filename命令。filename必须与盘上的文件名完全相同。若磁盘不在当前驱动器上，命令格式如下：D d：filename，其中d：为filename所在的驱动器名。

若文件不是纯ASCII码文件，则产生错误信息。显示文件时，使用终端的最大行长度进行显示。这时，由于自动字分裂功能不起作用，所以一行可以在字中间结束。

计算机在屏幕底信息行上提示文件名。可以按RETURN键显示文本的下一页，或按ESCAPE键返回至命令屏幕。当显示到文件末端时，按RETURN键返至命令屏幕。

第二十四节　计入命令

计入（INCLUDE）命令可将磁盘上全部或部分ASCII文件包括到正在编辑的文件中。

1.建立INCLUDE命令

a.在命令屏幕键入I filename。计算机重新显示命令屏幕，指出filename为现行计入文件。

b.每次仅有一个文件可以是计入文件。

c.在建立另一个计入文件或退出EDIL之前，filename指定的文件一直作为计入文件。

2.如何计入

a.将光标移到当前编辑文件中要放置其他文件的地方。

b.进入命令屏幕，并键入I启动INCLUDE序列。

c.计算机显示计入文件的第一个屏幕。

d.在信息行上提示按Y以计入当前屏幕，或在不需计入当前屏幕时按RETURN键。

e.计算机然后提示按RETURN键以继续至文本的下一屏幕；按ESCAPE键返回命令屏幕；或者键入节名，接着是RETURN（见下面）。

f.重复这一步骤直到计入文本结束并返回命令屏幕。

g.计入一节之前，计算机先将该节试装入内存。可计入的文本数量仅受有效内存的限制。若不能把文本节送入内存（即可用内存放不下该节文本），计算机通知用户。如果存储器已满，必须在继续执行计入序列之前将文件中一部分写入磁盘。

3.重置计入文件

a.如果文件的待计入部分在屏幕上出现的文本前面，则必须将文件复位至开始处。

b.为此可在命令屏幕键入I@命令。它将文件重置到开始处，并准备另一计入序列。

4.一次计入多节

a.在一个计入序列中可以计入多个文本屏幕。各节应按计入顺序放在文本文件中，第一节自光标处开始，此后立即跟以相继各节。

b.如果中断计入序列，例如重置文件，则要计入的下一节写在光标位置上，放在最后计入序列计入的文本前面。

c.若要该节跟在已计入的最后一节之后，必须进入文本屏幕并移动光标。例如：有一个由10节组成的计入文件，要按次序计入节8和节3。计入节8（方法如上述）之后，通过键入I@重置计入文件。进入文本屏幕并移动光标，使光标跟在节8之后，然后用通常方法计入节3。若不先移动光标，节3被计入节8之前。

d.若以错误的次序计入文本，可以利用字块移动（BM）命令容易地改正文件。

5.利用节或段编排文件格式：

a.利用换页（PAGE FEED）控制键帮助编排文件格式以分成合适的屏幕。在计入序列中换页当作节的标记处理，使每一屏幕（除第一屏幕外）均以换页开始，遇到下一个屏幕结束。

b.若一节超过一个屏幕，则只显示出该节的第一个屏幕。

c.可以在每节的第一行上置以标题以供识别和参考。标题以换页命令开始，以回车结束。

d.在计入序列中可规定一个标题。计算机进行检索，直至找到在换页符后的该标题。

e.当开始INCLUDE序列时可通过键入“I@标题”来指定标题。这时直接进入指定的节。

f.若需要重置文件，必须在键入命令检索指定的标题之前首先键入I@命令。

g.若已经按名检索到某一节，则在计入该节时不包含标题。

第二十五节　从辑编中打印

MAGIC WAND提供了在编辑时打印全部或部分文件的拷贝功能。除了执行换页命令和看不到回车外，打印在内存中的文本与屏幕上出现的完全相同。计算机打印所有嵌入的用于打印的命令，但不执行这些命令。键入P命令可以打印内存中的全部文本。如打印文本块，先在文本块两端设立标志，然后进入命令屏幕键入PB（PRINT BLOCK）命令。按CONTROLS可以中断当前打印。按其他任何键即可恢复打印。按ESCAPE键为完全停止打印。

注：许多打印机内部都有一个用于打印的小缓冲区，由于在应答停止命令前要清空其缓冲区，所以要用几秒时间才能停止打印。

第二十六节　后台打印

在编辑一个文件时可以在后台打印另一个文件（即所谓假脱机打印）。其步骤如下：

1.在命令屏幕键入S filename命令。若文件未在磁盘上或文件不是纯ASCII码文

件，则在屏幕上出现错误信息。注：若后台文件是由 PRINT 程序格式编排的文件，可以允许有非 ASCII 字符，只要按 RETURN 键即可继续进行。任何包含非 ASCII 字符的其他文件均不能打印。

2.计算机重写命令屏幕，指出打印对于该文件是可行的。

3.在命令屏幕键入 S 开始打印。然后进入文本屏幕可继续进行文件编辑。

4.若在打印文件中间停止打印，可进入命令屏幕键入 SX 命令。再键入 S 命令即可从断点开始恢复打印。

虽然在给定时间只能有一个现行的打印文件，但在一次会话期间可以建立多个文件作为后台打印文件。

后台打印有若干限制，这些限制降低了其使用能力。

a.计算机不能编排被打印文件的格式，所以文件必须有其行间断，例如程序列表或由 PRINT 程序产生的输出文件。

b.由于打印机和终端争用计算机时间，因而键盘响应缓慢。用户可以发现如果作长的附加内容会漏失字符。

c.退出编辑程序时必须终止后台打印。

第二十七节　列出文件

列出文件(FILES)命令可以在不退出编辑条件下检查磁盘的目录。其格式为在命令屏幕键入 Fd。其中 d 为指定驱动器，缺省时为当前驱动器。如果指定驱动器名有效。则列出该磁盘上的文件。按 RETURN 键可返回命令屏幕。

第二十八节　特殊格式特性

EDIT 有三种编排屏幕格式：文本、程序和特殊方式。各方式间的差异是一些格式编排特性：在字尾的自动行截断（字分裂）、空白压缩和回车符（～）在屏幕上显示等。在文本方式中，所有三个特性均起作用。在程序方式中，自动字分裂和回车显示不起作用。其他特性组合为特殊方式，例如自动字分裂无效但显示回车等。除输出文件类型为 ASC、ASM、BAS、COB、FOR、MAC或PRN外，EDIT以文本方式开始。在上述文件类型需要时以程序方式开始。

在命令屏幕键入 MP(Mode Program)命令即可变为程序方式。而键入 MT (Mode Text) 命令可变为文本方式。

当自动字分裂无效时，只有当文件行超过指定行长度或遇到回车时才断开文件行。

在命令屏幕键入 MAN (Mode Automatic Truncation, No)命令可以取消自动字分裂功能，而命令 MAY 可以恢复自动字分裂功能。

空白压缩特性取一系列空白并当穿过 CP／M标记点时用标记符号代替之。当写入有不同标记点的 MICROSOFT BASIC 程序或建立固定长度的数据文件时，不需要该特性。

要取消空白压缩特性，可进入命令屏幕并键入 MBN (Mode Block Packing, No) 命令。要恢复该功能，可键入 MBY 命令。

要取消回车特性，可进入命令屏幕键入 MCN (Mode Carriage Return, No)命令。为恢复此特性，可键入 MCY 命令。

第二十九节　退出编辑程序

正常退出编辑程序的方法是进入命令屏幕并键入 END命令。这时 EDIT以装入时用户指定的名字保存编辑文件。如果编辑文件不是以新名保存或存于不同盘上，则原来文件改名为 BAK 文件型。这时原来的 BAK 文件被抹掉。

注：计算机不能区别由同名文件产生的 BAK 文件。例如，有两个文件 REPLIES.PRO 和 REPLIES.CON，若 REPLIES.PRO 有 BAK 文件，则当编辑完REPLIES.CON 后，现存的 BAK文件将被抹掉，并以新的 REPLIES.BAK 代替之。

在需要退出编辑而不保存所编辑的文件时，可进入命令屏幕并键入 QUIT 命令。

有时用户可能要在编辑时保存当前文件而不改变原文件或其后备文件。假如启动 EDIT 时未为输出文件指定新名，则这时可以用 $ $ $ 文件型（输出目的文件）来保存。方法是进入命令屏幕并键入 W 命令（将编辑文件写入磁盘），该命令执行完后键入 QUITX 命令。

第三十节　磁盘满错误恢复

当执行 W 或 END 命令时，若磁盘空间已满，则在屏幕上给出 DISK IS FULL(磁盘满）信息。

这时用户可通过在命令屏幕执行 Kfilename命令删除盘上不必要的文件。若盘上存在用户指定的文件则询问用户是否要删除。若回答 Y，则执行删除指定文件功能；若回答N，则删除命令中止。在盘上有了足够空间时可再给出 W 或 END 命令来完成保存编辑文件的功能。

第三章　编辑特性扩充

在这以前所描述的为MAGIC WAND V1.0 编辑的基本特性。这一章将给出MAGIC WAND V1.1所扩充的编辑特性部分。有些特性在MAGIC WAND V1.0中也可以使用，但在手册中未作解释。

在MAGIC WAND V1.1中编辑特性中，全插入(FULL INSERT) 控制键现在既作为全插入键又作为结束插入 (END INSERT) 控制键。

其他功能扩充包括：(1)处理大型文件；(2)改进的文件操作；(3) 字处理功能；(4) 从编辑程序打印。

第一节　处理大型文件

MAGIC WAND V1.1已作了修改，使文件第一次读时装入文本至内存容量剩余约2000个字符。当处理大型文件时，每次将其中一部分送入内存。如果首次编辑文件时该文件太大以致不能装入内存，在命令屏幕上将显示信息：

More text on file—type R to read as needed

这表明内存中只有用户文件的一部分。但通常用户编辑整个文件。对于大型文件，可能就需要读出几次才能完成对全部文件的编辑。这时需要将内存当前内容写入磁盘，并读入下一篇文本，直到用户所要编辑的文本部分进入内存。

第二节　WR 和 WCR 命令

WR (Write and Read) 和WCR (Write to Cursor and Read) 两个命令可以完成将内存信息传入磁盘的功能。

WR 命令将内存全部内容写入磁盘，然后再装入同量的文本到内存。内存可以重新填入至离完全填满剩余约2000个字符，并且显示屏幕将再次提示用户是否还要再读文本。

WCR 命令同样完成写磁盘功能，但只是写入从内存中至转向命令屏幕之前光标所在位置之间的文本部分。一旦文件存入盘内，则必须以 END 命令退出编辑，并通过再进入编辑而返回到文件顶部。

第三节　新文件操作

MAGIC WAND 增加了下述三个新的文件处理特性。

1.提取和保存功能

在工作期间可能要输入某些感兴趣和有用的内容，如经常使用的短语或说明，这些内容可能以后还要用到。用户可以用提取和保存 (X) 功能将该项输入而不中断紧接的任务。可以对文件进行命名和存储，然后取回使用或供以后进行编辑。有时因为某种原因需要中断当

前工作，因此用户要会用保存功能来防止遗漏文本和损失时间。当通过提取从内存将文件保存在磁盘上时，MAGIC WAND 为之命名为 SAVE。用户也可以选择其他名字，因为每一次提取都将以前的保存文件删除。

(1)在命令屏幕上键入 X 命令，它将当前内存文件以 SAVE 为名保存在磁盘中。完成写盘之后返回命令屏幕，用户可继续编辑。

(2) 现在 SAVE 文件中包含了存盘时内存工作区的全部内容，以前未存于别处的保存内容均被抹掉。如果用户编辑时被意外中断，则可以通过对 SAVE 进行编辑和计入返回执行 SAVE 时的最后一个断点，继续用户编辑。

(3) 在用 W 或 END 命令将工作文件存入磁盘后，文件 SAVE 被抹掉。

(4) 在命令屏幕键入 X = filename 命令将当前内存内容以指定文件名 filename 保存。如果盘上已有名为 filename 的文件，则询问用户是否覆盖它。Y 抹掉以前文件的内容；N 则取消该请求。

2.用 XB 命令提取文本块

(1) 在文本中利用块标志设置文本块可以提取内存中的部分文件。这可以通过在命令屏幕键入 XB = filename 来完成。这时以文件名 filename 保存这部分文件。

(2) 在键入 XB 命令之前，通过更换磁盘可以将文件存到新磁盘上。

3.改变输出文件名

我们假定用户使用一个样板文件 FUZZY.CON 来建立类似的文本文件。键入 EDIT FUZZY.CON，修改后键入 END = IRONCIAD.CON，这时以新名保存修改好的文件。

注意，MAGIC WAND 使用 CP/M 改名功能来实现这一改名过程，因此改名后文件只能保存到由用户提示指定的缺省磁盘上（例如，如果用户是在 CP/M 提示符 D：下进入 EDIT，则输出文件产生在驱动器 D：上）。

第四节　改变输入文件名

在建立文件时，可能要从另一个文件合并附加文本，这种合并可通过将另一个文件连接至当前“写激活（Write active)”文件的末端来完成。

1.在命令屏幕键入 Cfilename 完成将另一个文件连接到已在内存的文本之后。

2.即使已改变了“读激活”（输入）文件的文件名，输出文件名仍保持与以前相同。

3.被连接的文件可以在不同的磁盘上，只要在键入 ‘C’ 之前改换磁盘，并记住在键入 ‘END’ 之前更换原来的输出磁盘，即要写入现行文件的磁盘。在结束编辑之前用户可以任意改变连接文件。

4.如在内存某个位置而不是文本末加入文本，可使用 ‘I’ 命令。

5.在连接新文件之后，可以通过 R 命令将其写入内存。

第五节　更换磁盘

编辑时可以在任何时间更换磁盘。可以利用 ‘X’ 功能从内存提取文本或利用 ‘C’ 或 ‘I’ 命令选择新的输入文件。

EDIT 可以防止用户写到错误的磁盘上。当用户键入 END 时，计算机在目录表中查找用户输出文件。如果不存在，则给出信息“filename .$$$must be mounted on drive (system drive)”。这时用户需要换入正确的盘并重新键入命令。

但在各种情况下，均可能由于写至错误的磁盘引起相当损坏，故在换盘时要十分小心。

第六节　字功能

1.字计数：每次显示命令屏幕时，均计算和显示文本中当前的单词数。带连字符的单词计作2。

2.字删除：当按 WORD DELETE 键时，若光标位于一个‘字’字符（单词）上，则删至下一个字的开始处。若光标位于“非字”，字符上，删至下一个字的末端。

3.字标记：在命令屏幕打入 TW 命令后开始字标记状态。当返回文本屏幕时，TAB 使光标跳至其右边一个单词。当该行上不再有单词时，光标返至常规标记参数。若已在文本末端，则返回到文本最后行的开始处。按 ESCAPE 键到命令屏幕并键入 T 命令恢复以前的列标记。

第七节　从 EDIT 打印

MAGIC WAND 可以用来输入各种文件，也可用作具有记忆功能的打字机以编辑和打印无需保存作以后调用的文件。为此，EDIT 中包括了一些打印命令，可以不退出编辑，进行文件输入和打印。

1.插入 MAGIC WAND 盘并键入 EDIT 可进入无文件名编辑状态。

2.这时建立的文件名为 WORK。若需要以后以另一个名保存这一文件，必须打入 END = filename 命令。注：不能用 END = 命令来更换磁盘驱动器。若 WORK 在驱动器 C 上，则 filename 要在驱动器 C 上。例如，END = D：LELLER1 在原来输出文件在驱动器 D 上时是合法的，D 为系统驱动器。

3.当调入文本屏幕时将为空屏幕，这时可以键入文件，并以 ESCAPE 退出文本屏幕至命令屏幕。

4.在命令屏幕键入 P = 命令设置打印参数。这时提示用户开始打印参数。

5.然后 P、P！ 和 PB 命令根据指定参数打印内存的内容。PB 命令仅打印包括在文件块标志内的内存内容。P！ 命令使打印机跳到新页并开始打印。

第四章　打印特性

第一节　装入打印程序和文本文件

1.装入打印程序 PRINT 和文本文件的完整格式是：(d：) PRINT (d：) filename，这里 d： 为驱动器名，缺省为当前驱动器。如果磁盘不在当前驱动器上，则必须指定驱动器名。

2.打印提示屏幕，并暂停等待用户换盘和（或）在处理文件前输入命令。

3.按 RETURN 键时，计算机重置驱动器(因换盘后 CP/M 将该驱动器临时置为只读)以便用户向任何盘进行写操作。

4.重置磁盘之后，检索用户指定的文件。若用户指定的文件不在指定磁盘上（或用户在以前未输入文件名），打印出信息并允许用户输入不同的文件名 filename。如果需要可在输入新名字之前改换磁盘，因为按 RETURN 键时计算机立即重置磁盘。

5.在磁盘上找到指定文件后，等待在按 RETURN 键之前打入命令。

6.若输入命令则执行之，在屏幕上打印倒斜杠（\）并等待其他命令。若未给出命令，则开始处理文件。

第二节　从键盘输入命令

绝大部分打印命令可在运行时直接从键盘输入。在下列四种情况下允许从键盘输入命令：a.首次装入 PRINT 时；b.遇到 TEXT 命令时；c.中断文件处理时（通过人工或嵌入 WAIT 命令）；d.遇到不适当命令时。

在键盘上按任何键（除 CONTROL 或 SHIFT 键之外）可人工中断文件处理。一旦计算机接收到一个字符即认为用户想要中断。遇到这种情况则在屏幕上打印倒斜杠（\），停止文件处理，并等待输入命令。如要输入命令则在键入命令后按 RETURN 键。计算机按照命令键入的顺序进行处理。如果计算机接收命令，则在屏幕上再打印出一个倒斜杠(\)准备接收另一条命令。若因某种原因拒绝接收命令，则在打印倒斜杠（\）前在屏幕上显示错误信息。在同一命令行上可输入多余命令，命令之间用逗号分隔。如果同一行上的某一命令不正确，则该命令之后的任何命令均无效。这时必须重新输入不正确命令之后的命令并改正错误。当输入完所有命令之后应按 RETURN 键。注：从键盘输入命令时，命令行不能长于屏幕宽度（通常为 80 字符）。

第三节　在文本文件中嵌入命令

用户可以在文本文件中嵌入任何打印命令。这些命令在文本中用倒斜杠（\）来标识。嵌入命令的格式与从键盘输入命令的格式完全相同，且这些命令用大小写表示均可以。每一

行可嵌入多余命令，每条命令间用逗号分开。单命令行包含的字符不超过 256 个。在完成命令行后按 RETURN 键。若要把命令作为文件行的一部分，可通过键入第二个倒斜杠(\)代替回车来结束命令。

下面两个例子的处理和打印结果完全相同：

例1：\LM10
\RM60
\Lincoln' s…

例2：\LM10，RM60\Lincoln' s…

若用倒斜杠（\）后跟回车来结束命令行，回车将成为文本的一部分并被打印机执行。

第四节　处理嵌入命令

当计算机遇到文本文件中的倒斜杠时，把倒斜杠之后的内容作为命令解释，直到遇到另一个倒斜杠或回车为止。嵌入命令对于打印机是不可见的，虽然计算机遇到每条命令都进行处理，但不打印命令本身，命令在打印纸上不占据空间。计算机接收命令后就给予处理，所以如果有一些命令必须按照特定顺序执行时，一定要保证输入顺序正确。若遇到不正确的命令时，就中断文件的处理，在屏幕上打印错误信息并给予机会从键盘输入正确的命令。如果键盘输入命令正确，则可继续处理。

第五节　文本定位命令

1. LM—LEFT MARGIN（左页边）

（1）LM 命令确定打印行从打印机左边偏移的字符数。其格式是 LMn。n 是零至打印机最大页宽之间的整数。例如 LM10 将该行偏移 10 个字符。

（2）LM 命令仅影响行在页上的位置而不影响行长度。计算机通过将空格插入行的始端来确定行在页上的位置。由于打印机对每个偏移字符都要输出一个空格，故若左页边较宽，则花费的打印时间也就长。

（3）LM 命令建立最小左页边；若要在最小左页边的右边建立临时左页边，可利用 IN 命令（见下面）。

（4）若键入 LM+n 或 LM−n，则增加或减小的偏移字符数为 n。例如，若原左页边偏移 10 个字符，键入 LM+5 命令时，左页边增至 15 个字符（注：也可通过键入 LM15来完成）。如果再执行 LM−5（或 LM10）则将左页边置回为 10 个字符。

2. RM—RIGHT MARGIN（右页边）

（1）RM 命令建立一行的字符数也即长度（缺省时为 60 字符）。其格式为 RMn，n 是整数。例如 RM50 建立了自左页边起 50 个字符的最大行长度。所置的行长度不超过打印机最大宽度减去左页边的偏移。例如，打印机最大宽度为 80 个字符，用户建立了 10 个字符的左页边，则 RM 命令的最大数为 70。所置的行也不得小于缩进值（IN），例如已建立了 20 个字符的临时左页边，不能改变行长度使之少于 20 个字符。

（2）可通过键入新长度或自当前长度加上或减去字符以增大或减小行长度。例如为了

将60个字符行长度改变为50个字符，可以键入RM55或RM－5命令。

3. IN—INDENTATION（缩进）

（1）IN命令建立临时左页边。其格式为INn，n为零至当前行长度之间的整数。例如RM为60时，最大可能的缩进值为60字符。

（2）与RM和LM一样，可通过键入新缩进值或改变前一缩进值来改变IN（IN＋n或IN－n）。IN0将左页边返回到由LM命令建立的左页边。

4. PI—PARAGRAPH INDENTATION（段缩进）

（1）PI命令建立逻辑行或段开始的缩进。缩进可以是正的或负的。其格式对于正缩进值为PIn，对于负缩进值为PI－n。

（2）正缩进限制为当前行长度减去临时缩进。负缩进不得大于当前临时缩进。例如：若IN是5个字符，PI－5为允许的最大负缩进。

（3）PI命令可以是负值，因此必须键入实际缩进值。

5. PL—PAGE LENGTH（页长）

（1）PL命令建立每打印页的单间隔行数（缺省值为66）。其格式为PLn，n为0至255之间的任意整数。页长必须大于或等于顶页边加底页边的长度。

（2）通过键入PL＋n或PL－n，可增大或减少页长。若所置页长不是通常的页长，则需要取消自动换页特性。

6. TM—TOP MARGIN（顶页边）

（1）TM命令建立打印文本体前保留在页顶部作为页边的行数。其格式为TMn，n是零至当前页长度减去当前底页边之间的任意整数。

（2）通过键入TM＋n或TM－n，可增大或减小顶页边。

（3）用户可通过TM命令给出标题（如有的话）和文本体之间的适当空间。

7. BM—BOTTOM MARGIN（底页边）

（1）BM命令建立从页底至文本下沿的行数，缺省值为6。其格式为BMn，n为零至当前页长减去当前顶页边之间的任意整数。

（2）通过键入BM＋n或BM－n，可增大或减小底页边。

第六节　格式控制命令

1. LEFT—齐平左边

（1）齐平左边是标准打印格式，左页边是平的，右页边不规则。齐平左边是PRINT的缺省格式。

（2）键入LEFT命令可以变为齐平左边。在齐平左边格式中，仅以完整的单词或连字符结束一行。

（3）打印时将该行末尾空格取消，使全部60个字符能放在60个字符行内。

（4）如果一个单词长度长于当前最大行长度，则将该单词在中间拆开。例如，要在15个字符行上打印20个字符的字，则在该行打印开始的15个字符，再在下一行打印5个字符。

2. RIGHT—齐平右边

(1) 齐平右边与齐平左边相反，即右边是平的，而左边不规则。变为齐平右边可键入命令 RIGHT。

(2) 在打印齐平右边时，在行的开始处插入将该行移至右齐平所需的空格数。

(3) 最常用的齐平右边格式是标题和底脚，例如打印日期或页号。

3. JUST—对齐

(1) 当左和右页边都齐平时，文本被对齐。计算机是通过将空格插入文本行填充至当前最大行长度（RM 减 IN 减 PI）建立齐平的页边。在标准打印机上，通过在文件行字间插入字符大小的空格来建立齐平的页边。

(2) 由于在文件行间插入大量空格，而在文本中产生了空间。计算机在文本行右、左或中间部分插入空格，且尽可能使文件行上的空格均匀分布。

(3) 若因为以回车结束，使文本行小于当前最大长度，则以左边对齐打印该行而不插入任何空格。

4. CENTER—中央格式

(1) 中央格式在当前最大行长的中央打印每行正文（LM 减 IN）。

(2) 键入 CENTER 命令使所有相继行置于中央，直到改变格式为止。

(3) 利用中央格式时，不在字末端分裂行。为使命令行置于中央，行长度必须短于最大行长度。打印的行长度大于当前最大行长度时，以文字格式打印。

5. CTR—中心行

(1) 为了将一单行置于中心，在该行始端键入 CTR（或在该行前的命令行上）。这时将以中央格式打印下一行。

(2) 当给出 CTR 命令时，以给出该命令时有效的格式打印相继行。例如当对齐左边格式时，给出将行置于中心的命令，将该行置于中心后就返至对齐左边格式。

(3) 所有适用于中央格式的规则也适用于中心行命令。

6. LIT—文字格式

(1) 在文字格式中，计算机按照所找到的文本格式打印之，超过最大当前行长度时才分裂行。

(2) 在文字格式中，执行除 PI 命令之外的全部格式控制命令。

(3) 通常程序清单需要以文字方式打印。注：若要在程序列表中使用 PRINT，一定要在开始列表之前使各种识别符无效（见下面）。可从键盘，或根据嵌在 REM 语句及其他相等语句中的命令给出这些打印命令。

第七节　行间隔命令(SP)

SP 命令置打印机行的间隔，其格式为 SPn。n 是 0 至 6 之间的整数，例如 SP2 使之置为双倍行间隔。SP0 停止打印机的自动换行，使用户可打印特殊的格式。例如，要打印一封信的标题，顾客姓名左边对齐而日期右边对齐，可给出下列命令：

\SP0, LEFT\American Widget Co.

\SP1, RIGHT\October 23, 1927

注：若要影响行末端的间隔，必须在回车前恢复或改变间隔。

第八节　识别字符

识别字符是文本中的字符。它能触发计算机的一个动作，即计算机能识别但不能打印它。识别字符仅在当作文本的一部分时起作用，而在是命令行或变量值时不起作用。每次可有7个识别字符：命令标记、重象连字符、底线、黑体下标和上标以及忽略字符等。用户可以定义任意字符作为识别字符。但对于给定功能只能定义一个识别字符，只要对该功能定义了新字符，前面的字符就失去其定义，即不再是识别字符。每次一个字符仅可作为一个功能的识别字符。PRINT对除IGNORE外的所有识别字符都有缺省值。一般不应改变这些字符，除非需要打印其中的某个字符。

8.1 CMD—COMMAND MARKER（命令标记）

键入CMDc命令可以改变命令标记（缺省为＼字符），c是任意字符。该命令可以从文本或从键盘给出。

若在文本中给出命令，必须首先用当前命令标记来标记该命令。若要用命令标记而不是回车结束命令行，必须使用旧的命令标记而不是新的（这是因为该命令被执行前命令标记不改变）。例如，若要从倒斜杠变成方括弧，应键入＼CMDI＼。只键入CMD意为不把任何字符作命令标记符，即不在文本中嵌入命令。但可以中断，从键盘给出命令。当准备好从键盘给出命令时，在屏幕上打印命令标记。若未定义命令标记，屏幕不打印任何内容，仍可如通常那样给出命令。

若必须改变命令标记，应在文本开始改变命令标记并在整个文本中保持一致，以免混淆。在多遍扫描的文件中，改变命令标记会产生混乱，因为在第二遍时不能识别改变了命令标记的原来命令（因＼不再是命令标记）。若在多遍文件中用嵌入命令来改变命令标记，必须在文件末端将标记变回到倒斜杠。也可以在开始处理文件之前，从键盘改变命令标记，这种改变命令标记的命令不是文件的一部分。

8.2 HY—重象连字符

缺省重象连字符是&符。若定义新的重象连字符，可键入HYc命令，c是任意字符。

当一个单词不能放在一行中时，则在允许放连字符处放置一个重象连字识别符。计算机检查单词中是否包含重象连字符，如果包含，则检查由识别符标记的单词片断是否在该行中。若在，则打印该片断，用实际的连字符代替重象连字符，并把该单词的其余部分移向下一行的始端。若该单词含有不止一个重象连字符，则打印能放在文件行上的最大片断。

重象连字符不在屏幕上显示。若一个字不需要用连字符连接，则放过识别字符；若需要连接，则用其连字符代替识别字符。

计算机仅在重象连字符处分裂字，要让一个用连字符连接的单词必要时可被断开，必须首先插入重象连字符。例如，Mother-in-low应成为Mother-&in-&low。若重象连字符没跟着实际连字符，则只打印一个连字符。

重象连字符除在作连字符判定时外均被忽略，故可用重象连字符识别符作为第二个IGNORE字符，只要其置于字末端和字之间即可（见下面省略字符一节）。

8.3 UN—划底线

可用底线识别字符来标记要划底线的字符（缺省为—符）。命令格式为UNc，其中c为

任意字符。将识别字符直接置于第一个字符之前，并直接跟在要划底线的最后一个字符之后。若要在划底线的字符之后直接跟以回车，则无需键入第二个识别字符。

在打之印一行后，打印机返回，并在已标记为要划底线的字符下划线。可以自动地在文本的任何点任何字符下划线，而不管正在干什么。例如，连字、置中心和对齐等。

如果设置的区距大于CPI10，则底线可以不全连在一起。底线可以连接或断开。连接线在两个标记之间的每个字符下都划线；断开线在字母和数字下划线，但在空白或标点符号下不划线。划连接底线，可使用命令 UNS；划断开底线，键入 UNB 命令。为了打印连接底线，可不定义底线字符，可以象打印机那样键入该行，或者在适当空格两端置两个底线标记，要保证计算机置于连接底线方式。如：

（\UN\—\UN_\或\UNS_ _\UNB\）

8.4 IGNORE—忽略

忽略命令格式为 IGNOREc。其中 c 为打印时被忽略的字符。打印时打印机放过字符 c 不打印。忽略字符用于 SKIP TO 命令，使跳至文本行中间，或跳至命令行的始端。忽略命令也可使其他识别字符无效。

8.5 OUT—输出

OUT 命令可向打印机发送一系列数，通知打印什么字符、移动多少等以驱动打印机。这些数是根据文本字符 ASCII 值和打印机的某些约定产生的。这意味着有可能利用打印机上不由PRINT支持的特殊特性。

OUT 命令格式为OUTn1, n2, ···, nx，其中数在 0 到255之间。计算机仅检查n是可接收的数，这些数的意义由用户负责保证。计算机只要遇到命令就将数送至打印机。与识别字符不同，计算机直到文本字符送至打印机之后才等待。注：此命令主要是给程序员使用的，若不熟悉打印机的工作，一般不要使用该命令。

第九节　多页和多次扫描命令

1. DRAFT —草拟

DRAFT 命令可以断开或省略与任何专用打印机有关的命令（黑体字、变量节距、下标和上标等）。这样可以在较快的标准打印机上打印为专用打印机准备的文件。打印这种文件前必须给出 DRAFT 命令。命令可以嵌入文件或在运行时从键盘给出。给出命令之后不要改变 DRAFT 状态。

2. FORM—页格式

任何多页或多次扫描文件需要不止一页打印纸。在不给出 FORM 命令指定时，缺省为连续的纸张。若不使用连续形式纸张，必须在页间暂停以插入新纸。使页间暂停的命令格式为 FORMS。一旦给出此命令，打印机在每页之后暂停以便装入新纸。装入打印纸后，按 RETURN 键可恢复打印。若要在页末端终止打印，按 ESCAPE 而不是 RETURN 键。如键入 FORMC 命令可以恢复连续形式打印。

3. LINE—移行

LINE 命令允许在一页上移动指定行。其格式为 LINEn，n 是当前行和当前最大行数之间的数。n超出范围时命令无效。

LINE 命令同 TAB 命令纵向移动是相等的。此命令允许移至指定行数而与已经打印完的行数无关。LINE 命令允许移动至指定行，与使用的行间隔无关。例如，有从第 1 行开始的双行间隔的文件，通常不能移至第 10 行。键入 LINE10 即可跳至第 10 行。

4. TAB—列跳动

TAB 命令允许跳至指定列。其格式是 TABn，n 是当前列和最大列数之间的数。

若要将一列标记移至当前列的左面，该命令无效。若要将一列标记移至当前最大列的右边，则拒绝接收该命令并在屏幕上打印错误信息。要在标记到某一列打印字符时，键入 TAB "c"n。c 是任意字符。例如要在标记 40 列时打印句号，键入命令 TAB "."40。

5. NP—新页

(1) NP 命令建立新页。若文件中有任何标题和底脚，计算机在 NP命令后执行它们。

(2) 处理嵌入 EDIT 中的换页字符与处理 NP 命令完全相同。在文本开始之前给出的 NP 命令并不实际建立新页或改变页计数，但产生在第一页上打印的文件的标题。在任何其他时间给出的 NP 命令进行实际换页，增加页计数，并重置行计数为零。回车是 NP 命令的一部分。若键入命令序列＼CTR，NP＼，下一行将不置于中心，因为 NP 命令中的回车断开置中心的开关。

注：NP 命令在 SETUP 和 TEXT命令间的初始段中不起作用。

6. NL—新行

NL 命令与处理 RETURN 相等。当需要建立新行但不能用回车时，可使用 NL 命令。例如在这行时从键盘或在条件打印语句中使用 NL命令。

例：若正在打印一段文件，在该段中间有条件语句。若语句为真，在该点结束此段并跳至其余部分 (NL，SKIP)；若语句为假，打印该段的其余部分，通常以回车结束。

7. FF—换页

若要强行换页而不修改页面或行的计数，可使用 FF 命令。键入 FF 与从打印机按换页键完全相同。FF 命令的主要用途是在打印之前从控制台重置打印纸。

注：由于 FF命令不计算页数或行数，在文本文件中使用 FF 命令会产生混淆。FF 和 NP 命令不相同。在打印文本文件时要换页应使用 NP 命令。

8. FORMFEED—换页开关

当已达页末或遇到 NP命令时，可利用打印机中换页硬件。若打印时，在打印机上置的是页而不是长度，但不希望按这种设置打印，可使用换页开关。例如，若打印机被置为每页 66 行而文件要求 22 行的页长度时，打印机在页末尾跳过其余 44 行。若要断开硬件换页，可键入 FORMFEED OFF 命令。若要再换页可键入 FORMFEED ON命令。要注意在许多打印机上不具有机械换页和逐行换页的差别。

9. PG —置页数

PG 命令允许改变 % PAGE 系统变量的当前值（见下面“系统变量”一节）。其格式为 PGn，n是 0 至32767 之间的任意数。这是改变 % PAGE 值的唯一方法。注：PG 命令仅改变页计数的方法，而不影响文件的打印。例如：已准备好打印报告的第二部分，第一部分在第 20 页结束。要使该部分在第 21 页开始，可键入 PG21 命令。

10. COPY—拷贝

如要进行文件的多份拷贝，可使用 COPY命令。其格式为 COPYn，n是任意数。当打

印到文件末端时就自动地开始文件的下一次拷贝。不给出 COPY 命令时缺省为n = 1。计算机不能自动至新页或重置页计数器，若要这样做就必须嵌入命令（见 NP 和PG命令）。打印机完成打印各次扫描文件直到打印完指定份数为止。键入 COPY0是特殊情况。每次扫描完毕暂停并询问是否要新的拷贝。回答Y就进行下一拷贝，回答N就停止文件的处理。

11. SETUP／TEXT—置位／文本命令

SETUP 和TEXT命令标记仅在第一次扫描时执行的文件节。

SETUP 应是文件中最先的命令之一。在 SETUP 节中包括所有外部数据文件的定义、标题和底脚秩序以及仅在第一次起作用（如打印日期）和仅需在第一次完成的功能。计算机执行文件第一次扫描中该开始节的所有命令。在相继的各次扫描中，计算机放过该节。开始节仅用于命令，计算机忽略该节中的任何文本字符。由于 NP 命令产生文本（即回车），计算机忽略 SETUP 节中的任何 NP 命令。

TEXT 命令标记文本的开始，并应出现在该文件中用于打印的文本之前。TEXT命令必须是命令行中的唯一命令。在第一次扫描上，当计算机在 SETUP 命令之后遇到 TEXT 命令时就暂停处理文件，显示状态屏幕。这时可以从键盘打入命令。在以后的各次扫描不暂停。若已在初始节中定义了数据文件，直到在 TEXT命令暂停时才打开该文件。若文 件没有 TEXT命令，则打印机打印文件而在文本始端不停止。

12. START／STOP—启动和停止

START和STOP命令允许指定外部数据文件中要启动和停止处理的记录。当处理特定记录时键入命令 STARTn，n 为记录号；若要停止处理指定记录时 键 入 命 令 STOPn。START 和 STOP 命令彼此无关。若给出 START 命令而未给出 STOP 命令，则从指出的记录开始并继续进行直到文件结束为止；若给出 STOP 命令而未 给出 START 命令，则从第一个记录开始并继续进行，直到已打印出 STOP 记录为止。计算机 在 STOP 记 录点终止处理过程。文件设有外部数据文件时，START 和 STOP 命令确定开始或结束处理的页号。

13. END—结束

END 命令允许指定结束文件的某次扫描。此命令通常用来建立数据文 件 的 新 记 录。END 命令必须是命令行上最后一个命令。END 命令可以节省时间，可以跳过不须打 印 的文本部分而不做处理。若在处理标题和底脚时给出 END 命令，则结束标题或底脚处理而返回文本体。

14. QUIT—退出

QUIT 命令无条件立即终止文本文件的全部处理并结束打印。可在文件 中 嵌 入QUIT命令，但更多的是利用此命令从键盘停止打印程序。和 END 命令一样，QUIT命令必须是一行上的最后一条命令。

第十节　变量概述

1. 变量是可以具有多于一个值的量。变量名可达7个字符。它可以是数值和大写字母的任意组合，只要第一个字符为字母即可。

2. 变量当作命令处理，并仅出现在命令行上。

3.同其他命令一样，把变量中的小写字母当作大写字母处理。对于不适当的变量名不予接收，并在屏幕中打印错误信息。

4.变量值是可达55个字符的字符串。字符串可包含任意字符，如小写字母、标点符号、空格等。

5.无需重新定义识别字符以将其包含在字符串中。与其他命令一样，当识别字符是变量值的一部分时就不起作用。

6.若字符串超过55个字符，则只前55个字符有效。

7.若要指定字符串开始的字符数，键入V（n），V是任意变量，n是1～55间的字符数。

8.在一个文件中不得超过128个变量。一个文件中允许的变量缺省值是32，但是可以用VSIZE命令增加或减少这个数（见下面"改变内存缓冲区"一节）。

9.可通过变量引用或作为IF语句中的一部分来建立一个变量，但只能用GET、SET和DATA三个命令中的一个给变量赋值。

10.一旦建立了变量名，即使未使用变量，只要是在打印阶段，变量就存在。

10.1 GET—运行时间赋值

GET命令允许在运行时间从键盘定义变量的值。其格式是GET V，V是任意合法变量名。当计算机遇到GET命令时就停止处理文件，并在屏幕上打印EnterV。例如，若键入GET NAME，则在屏幕上打印出Enter NAME。

键入变量值并按RETURN键，计算机将键入的内容（开始的55个字符）赋为变量值，并恢复处理文件。若要限制变量长度小于55字符，可键入GET V（n）。例如，GET NAME（25）将所有多于25个字符的变量名缩写为25个字符。由于在定义变量名前已指定限制范围，计算机仅在内存中存储指定的字符数。按RETURN键而不输入任何内容，使变量值不变。若要消去变量值，在按RETURN键前按入空格。

若需要，可定义打印在屏幕上的是提示语句而不是变量名。为此键入GET V ="提示语句"。例如，GET NAME="FULL name (last name first)?"。现在打印语句Full name (last name first)?，而不是变量名Enter name。可从键盘直接给出GET命令作为给变量赋值的方法，但给变量赋值的方法不止此一种。

10.2 SET—直接赋值

SET命令可直接给变量赋值而不反映到屏幕上。SET命令还可从键盘赋值而不利用GET命令。与GET命令一样，可利用SET命令将字符串值赋给变量。为此键入SET V="S"。此命令赋字符串S为变量V值。

与GET命令不同，SET命令能将一个变量值置为另一个变量值。为此键入SETV=V1，此命令置V值等于V1值。例如，SET NAME1=:NAME。

注：当需要存储即将改变为其他内容的当前变量时，使用SET命令。可利用SET命令存储变量C的截断值。例如，SET NAME1（25）=:NAME。还可利用SET命令来增加或减少（加上或减去）数值变量，可加上或减去2个或多个数值变量（见"数值变量"一节）。不能使用SET命令将字符串加在一起。

10.3 FILE—文件语句

用户可从外部数据文件即从不是正在处理的文件将一值赋给变量。一个数据文件划为若干记录。计算机在每遍文件扫描开始从数据文件读出一个完整记录。一个记录可分成若干元

素，可将元素赋作变量值。

有两类文件可以存取：文本文件和固定文件。文本文件中的元素是以回车结束的，具有不定数目的字符。文本文件中每个记录包含固定的元素（回车），在固定文件中的每个元素和记录中的每个元素包含固定的字符数。若要定义文本数据文件，可键入 FILE Tn，n 是每个记录中的元素（回车）数。例如，文件的每个记录有12个元素，键入 FILE T_{12}。

若定义固定数据文件，键入 FILE Fn，n是每个记录的字符数，但不得超过 128 个。

通过键入逗号并跟以完整的文件名，可以在 FILE 中计入数据文件名。例如，FILE T_{10}，CUSTOMER.IN0或 FILE F_{128}，B：NAMELIST。用户可以利用变量提供文件名。例如：键入 GET，FILENAME，FILE T_{12}，：FILENAME(注：冒号表示 FILENAME是变量，没有冒号时则 FILENAME 作为文件名）。

若未给文本文件命名，或计算机未找到已命名的文件，则在屏幕上给出错误信息。

可以利用 FNAME 命令从键盘定义或改变数据文件名。其格式是 FNAME，FILENAME。命令 FNAME 可在打开文件前的任何时间改变数据文件名。在计算机准备好读入数据文件的首记录时，才打开数据文件。注意，应该总是将 FILE 命令置于 SETUP 和 TEXT 命令之间的文件初始节。当到达 TEXT 命令时，计算机显示系统状态，并允许从键盘输入命令。这时可用 FNAME 命令，因为当一允许处理文件，计算机就要打开文件。在给定的文本文件处理中只存取一个数据文件。若已打开数据文件，则忽略所有附加的 FILE 或 FNAME 命令。FILE 命令只告诉计算机每个记录中包含多少数据元素，而不建立任何变量或对变量赋值。由于 FILE 语句具有几个部分，此语句应是文件行上最后的、也许是唯一的命令。

10.4 DATA—数据语句

利用 DATA 语句建立和定义数据变量。DATA 语句确定如何将记录的每个元素赋值给变量。若数据文件是文本文件，DATA 语句的格式为 DATA V1, V2, V3 等。V1，V2 和V3是变量名。在此命令之后，计算机将记录的第一个元素赋值给第一个变量，将记录的第二个元素赋值给第二个变量等。若数据文件是固定文件，DATA 语句的格式是 DATA V1 (n1), V2(n2) 等。V1 和 V2 是变量名，n1 和 n2 是变量中的字符数。在此命令之后，计算机将记录中的第一个n1字符赋给V1，将以下的n2个字符赋给V2等。

注：若要限制来自文本文件的变量长度，可以在 DATA 语句中的变量名之后包含变量长度。

无需将记录中的每个元素赋给变量，但须指出每个元素在 DATA 语句中的位置。来自文本文件的不可赋值的元素在 DATA 语句中用额外的逗号表示。例如有6个元素的记录，要赋第二个元素为 NAME、第四个元素为 PHONE、第五个元素为 REGION。键入 DATA，NAME，,PHONE,REGION，命令。最后的逗号任选，因为计算机在所有变量被定义之后忽略留下的元素。

若要跳过部分固定记录，必须指出跳过的字符数。例如，DATA NAME (30)，15，ADDR1 (20)，ADDR2 (20)，5，SALUTE (15)。计算机忽略留在记录末端的任何未赋值的字符。只要赋值的元素或字符不超过在 FILE 语句中建立的元素和字符，可以使用不止一个 DATA 语句来给记录中的元素赋值。若要赋值的元素或字符多于记录中定义的元素或字符，计算机拒绝该命令，并在屏幕上打印错误信息。

每个DATA语句均作为第一个DATA语句的延续来处理，例如下列命令：

\DATA NAME

\DATA ADDR1

\DATA ADDR2

与\DATA NAME，ADDR1，ADDR2等价。

仅能对包含纯ASCII字符的变量元素赋值，来自数据文件未赋值的元素可包含非ASCII字符。例如，若有由BASIC程序产生的固定文件，该文件包含用非ASCII格式存储的数，只要放过所有的非ASCII字符，仍可将其当作数据文件使用。一个文本数据记录包含的元素不得超过128个。文本数据记录中的一个元素可包含任意数目的字符，但只能对元素中开始的55个字符赋值。一个固定数据包含的字符不得超过32768个。对于某固定数据记录的变量赋值的字符不得超过55个。每次最多可跳过固定记录中的255个字符。

注：由于可用任选方法划分固定记录，在一排中可通过给字符赋值而分开变量，使得赋值的字符可超过55个；类似地，通过跳过相邻的元素可使跳过的字符超过255个。

第十一节　变量使用

用户可用GET、SET或DATA命令将值赋给变量。利用变量或IF语句可建立变量而不予以赋值。变量必须是命令行的一部分。为了指明变量，用下列几种符号的一种作变量名前缀：冒号（：）、等号（=）、美元号（$）、数号（#）、&符号或百分比号（%）。除指出变量名之外，这些符号还告诉计算机如何解释变量。若未指明正在使用变量，计算机把变量作命令处理。这时，通常在屏幕上打印错误信息。但若变量名与命令相同，则执行该命令。

注：在GET、SET、DATA或IF语句中无需给变量加前缀，除非要用特殊方法解释变量。

第十二节　字符串变量

12.1　：VARIABLE—冒号变量

计算机为已建立的每个变量当前值分配55个字符的内存。虽然可用不同方法解释变量值，但总是将内存中的值作字符串存储。若变量值小于55个字符，则在最后字符之后加入空白。变量前的冒号表示在使用其值时去除所有结尾空白。若将变量值作为文本的一部分打印，键入\：V\，V是任意变量名。当遇到该命令时就打印V的当前值，去除所有结尾空白。若要在行上打印变量本身，必须在回车之前用命令标记跟在变量名之后，或用NL命令跟在变量之后。例如\：NAMA\～或\：NAME，NL。若以回车直接跟在变量之后，这时回车解释为命令行的结束，而不作为文本行的结束。在大多数情况下，当打印变量或引用变量为命令的一部分时，应使用冒号变量。

12.2　=VARIABLE—等号变量

将等号放在变量前就可返至变量的未截断值，即可能包含具有结尾空白的值。当用GET、SET或DATA命令给变量赋值但不规定变量长度时，计算机将实际长度值存于内

存中。这时由于不存储任何结尾空白，冒号变量和等号变量是等价的。

如果在给变量赋值时规定长度，计算机把该长度作为变量值长度存储，并将变量值缩短或将空白加至变量值之上，使之有规定的长度。例如：若键入 SET DATE(30) = "May1, 1950"，这时在键入的11个字符之后加19个空格，使恰好为30个字符。

对于大多数情况来说，由于不截断字符会造成在文本中放入不需要的空白，因此应该使用冒号变量，尽量不要用等号变量。等号变量虽然不常用，但其变化的格式即 = "字符串" 却十分重要和普遍，可用来打印文字（字符串）语句。="（等于引号）命令允许在运行时从键盘将文本插入打印文件。还允许将文本打印为命令行的一部分，通常仅是条件语句的一部分（见"条件语句"一节）。若已从键盘插入文本，则字符串的最大长度为屏幕的宽度。例如屏幕宽度为80个字符，则字符串不超过78个字符（两个字符用于="）若当输入字符串时超过最大长度，计算机拒绝执行任何附加字符，并立即打印到目前为止已输入的内容。若要插入的文本超过一个屏幕行，通过给出附加="命令就可在被中断处继续进行。在前一文本之后立即打印每一相继行而不中断。

当使用="命令时，利用 NL 命令将回车和换行插入文本。注：若已输入的内容未充填打印页上的一行，则将字符串存于内存，直到行的其余部分从文件或屏幕填入为止，或直至接收到回车或 NL 命令为止。

若正使用="命令作为嵌入文本中命令行的一部分，字符串不得使命令行超过最大的255个字符。例如，若在30个字符命令前面使用="命令，字符串最多可为226个字符长。

12.3 $VARIABLE—美元变量

变量名之前的美元符（$）表明要以逗号和小数点编排变量值的格式。在小数点左边可有12个字符，右边可有2个位置。如999，999，999，999，99。美元变量的主要用途是将数据文件中未处理的数按其为美元数量来打印。只在变量名前有美元字符时不将美元符号加至打印数之前。想要在打印值前面出现美元符号，除在变量名之前加一美元符号之外，还要在文本中放一美元符号。

当将变量值变换为美元格式时，计算机忽略除了数、句号、加号和减号以外的所有字符。这意味着经过格式化的数可作为无逗号或美元符号的数来处理。计算机将其在变量值中找到的第一个句号作为小数点处理，并忽略任何以后的句号。若在字符串中无句号，仅打印整数而无小数点和两个小数数位。若找到一个句号，在句号的右边打印两个字符。若在句号之后找到的数少于两个就填入零。若在句号之后找到的字符多于两个，仅打印开始的两个字符。例：

10000成为10,000
10000.成为10,000.00
10000.1成为10,000.10
10000.102成为10,000.10

在变量值中存在负号（或破折号），使变量值作为负数处理（除非因某些原因，在以后字符中有加号）。当打印负数时，负号跟在最后字符之后，即×××.××-。

可将美元符号置于任何变量之前而不得到错误信息。若字符串中不包含任何数，则美元值为零。

注：可在条件（IF）命令中使用美元变量来比较两个数，这是比较很大的数、小数或

负数的唯一方法。

如果要将美国格式变为国标格式，即用句号代替逗号和变小数点为逗号，可以键入DECIMALC，要改回至美国格式时键入DECIMALP。

第十三节　数值变量

数值变量与字符串变量不同，其值作为数而不是作为字符串处理。数和字符串的主要差别在于数可用于算术运算，而字符串仅可被打印或比较。任何字符串都可有数值。可用数值变量（或数值表达式）取代任何接收数的命令（例如左页边、页长等）中的数。在数的位置不能放变量的唯一命令是BF，它只允许BF1至BF9。

可利用SET命令将数值变量加上或减去常数，或其他数值变量。例如，SET*TIMES=*TIMES+1，SET*LEFT=*LEFT-1，SET*TOTAL=*PAID+*FREE-*NOSHOW。数值变量必须是整数。包含小数的被舍入至最靠近的整数。例如，10.5成为10，1.99成为1。数值变量不得大于32767或小于零。小于零的数值作零处理。计算机拒绝大于32767的数和值小于零或大于32767的数值表达式，并打印错误信息。

13.1 *VARIABLE—数变量

在变量前置数符号时，该变量的字符串值作为数值处理。将字符串换为数时，计算机逐字符读出字符串，将1至9的字符作数的一部分处理，直至遇到1至9以外的字符为止。

小数点和负号不是数，因此，不能识别小数或负数。若数超过32767则拒绝接收该命令。若变量值是空白或以非数值字符开头，则作为零处理。

下例中左边数字是字符串值，右边是变换为数的字符串：

“12345”	12345	“123,456”	123
“123.45”	123	“123Main”	123
“-12345”	0	“123456”	ERROR

数变量可以加上或减去任何数值变量或常数，称为数值表达式。数值表达式中可意有任数目的数值变量或常数。数值表达式可看作为一个数，计算结果是表达式的值。若表达式值大于32767或小于零，则拒绝接收该命令并打印错误信息。在任何接收数值参数的命令中可使用数变量或表达式。

例：若打印一个文件但事先不知道开始页数，可在文件的SETUP节放入下列命令：

```
\GET PAGE= “Starting page number? ”
\PG*PAGE
```

要给数变量一个负值，必须在它前面放一个负号，如PI-*V，K-*V。还可将数字变量和IF语句中其他数值变量、表达式或常数进行比较（见“数值比较”一节）

利用SET命令可将变量的数值设置为任何其他数值变量、表达式或常数。注：当设置数变量时，切勿将引号（“）放在值的前后，因为引号是用来指出字符串而不是数的。例：为了设置变量等于常数，键入SET*V=3。利用SET命令可建立文件中的计数器，记下指定功能已执行多少次。例：为了设置一个变量，记下文件某一部分的打印次数，可键入命令SET*TIMES=*TIMES+1。注：此命令将*TIMES的值增加1，这样做是可以的，因为在执行此命令后才改变*TIMES的值。还可利用SET命令储存一些其他数值变量的

当前值。例如，设置一个变量等于当前页数，可键入 SET*PAGE = %PAGE。注：所有长度（&）变量和系统（%）变量均是数值变量。不能置字符串值等于数值变量，例如以下命令为非法的：SET PAGE = %PAGE。虽然作为数将值赋给变量，但将其值作为字符串存储，可如同其他字符串值一样处理。由于值作为字符串存储，因此当要使用变量的数值时，必须把数符号放在变量名之前，即使原来已赋给它一个数值。注：虽然空白变量的数值被解释为零，但若实际将零值赋给变量，则必须利用 SET 命令。两者的区别在于空白值无长度，而零值长度为 1。

13.2 &VARIABLE—长度变量

将&符号放在变量名之前时，计算机返回该变量字符串值的长度变量，截断至最后一个非空白字符处。长度变量具有数值，可像其他任何数值变量(如数值表达式、命令参数等)那样使用。长度变量的一个重要用途是测试字符串变量的长度，以保证它可装入指定的地方。例：下列命令从键盘得到名字，测试其长度。若其大于30个字符，截断至30个字符：

```
\GET NAME
\IF&NAME<=30, SKIP3
\SET NAME=NAME(30)
\SHOW "Name must be shortened to",:NAME
\GET NAME="If OK press RETURN, else reenter name"
```

13.3 %VARIABLE—系统变量

系统变量为数值变量，它由计算机自动维护的记录文件处理。系统变量可以在任何能使用数值变量处使用。不能用 SET、GET或DATA 命令改变系统变量。

系统变量是：%PAGE，%PASS，%REC，%LINE，%LINES，%COL，%EOF。只有系统变量可取消百分比符号。若在任何其他变量名前加百分比符号则打印错误信息。可以有与系统变量同名的变量，例如 PAGE 或 REC，但其值与系统对应值无关。

1．%PAGE

%PAGE 表示正在打印文件的当前页数。每次当打印从一页底部至下一页首部之间换页时，%PAGE 增1。与其他系统变量不同，用户可以改变%PAGE 值(使用 PG 命令)。

注：计算机在文本文件扫描结束时不重置%PAGE，若要页计数器重置为1（或其他数），必须用 PG 命令。

2．%PASS

%PASS 告诉当前扫描次数，即包括当前这一次在内已扫描了多少次。从外部数据文件打印文件时经常使用%PASS。

3．%REC

%REC 告诉用户外部数据文件中被处理的当前记录号。若记录从 1 开始，%REC 同%PASS 相同。若不从记录1开始，则 %REC 和 %PASS 不同。例如若从记录 5 开始（键入START5），则 %REC 为 5 而 %PASS 仍是 1。

4．%LINE

%LINE 表示出当前行数。行数从页顶（包括顶页边）算起。例如若顶页边为3行(TM3)，文本体从第4行开始。行数为小数时舍入至最靠近的整数。每次进入新页时，计算机将%LINE 复位为零。

5. %LINES

%LINES 告诉用户当前页文本体留有多少行（包括当前行）。例：若现在位于第50行（每页66行），有6行底页边，则%LINES 为11（50至60是11行）。当不要用断页末尾部分的某些行时使用%LINES。例：现有占8行的数字表格，要保证不在一页的中间结束该 页，可在数字表格前面的命令行上键入＼IF%LINES<8，NP。这样可在打印表格之前，检查是否有足够的空间来打印该表格，若没有足够空间就建立新页。

6. %COL

%COL 表示出打印行上当前列的位置。利用%COL可帮助返回至处理文件前不能确定的指定列。例：若正在打印文件时想在磁盘上存储某种信息而不丢失文件中的位置，可以断开打印机并接通磁盘，置行间隔为零并将% COL 赋给变量。当完成写入磁盘后，再接通打印机，给以前保存的列打标记，并恢复断开处的文件打印。

7. %EOF

%EOF告诉用户是否已处理完数据文件的最后记录。当到达数据文件末端时，%EOF 等于1，其他时间%EOF等于零。

当到达数据文件末端时，在终止处理之前检查在 FOOT 程序中是否存在任何特殊命令。这表明可利用 FOOT 程序建立仅当到达文件结束时才作处理的文本。例：若文件 含有3个不同的字母，这些字母的打印取决于数据文件的变量。已在文件中嵌入数变量作为计数器以记下每类字母打印的次数。计数器命名为 LTR_1、LTR_2 和 LTR_3。还有一个计数器称为 LTR_0，用于没有被打印字母的记录。处理完最后一个记录后，要让计算机跳至新页并摘要打印出每个字母打印了多少次。为此用户可建立底脚记录：

```
＼FOOT7
＼IF%EOF = 0, END
＼FF, CTR＼Recap of processing for＼ : DATE＼, NL, NL
Letter one＼TAB20, : LTR1＼Records, NL
Letter two＼TAB20, : LTR2＼Records, NL
Letter three＼TAB20, : LTR3＼Records, NL
No letter＼TAB20, : LTR0＼Records, NL
＼TAB10＼TOTAL＼TAB20, %PASS＼Records processed
```

第十四节　条件命令

14.1 IF语句

条件命令仅当条件为真时才执行。而条件必须是两个表达式之间的比较。表达式则是变量、常数或两者结合。由于已知常数比较的结果，第一个表达式必须包含一个变量。可以进行测试，判断一个表达式是否大于（>）、小于（<）或等于（=）其他表达式。还可以同时测试判断任何两个关系，即大于或等于（>=）、小于或等于（<=）及不等于(<>)。

条件由 IF语句建立。其格式为：

IF 〈第一表达式〉比较〈第二表达式〉

如需要可使用 NOT 与 IF 结合来判断条件是否为真。例如 IF NOT*TIMES<5（如

果 TIMES 的数值不小于 5)。若变量名以 'NOT' 开始，必须在其前加一冒号，否则按 IF NOT 解释。IF 语句后面可跟以任何合法命令或命令系列，用逗号分开。当条件为真，则执行命令行的其余命令。当条件为假则跳过命令行中其余的命令。

14.2 比较字符串

用户可将字符串与任何其他文字字符串或字符串变量进行比较。不必在第一个变量名之前加冒号或等号（虽然需要时也可以加），因为一般已假定（除非另有指定）变量名是字符串。

若正在比较变量名与文字字符串，必须将文字包括在引号内（单引号或双引号）。

比较字符串变量与常数的格式是：

IF〔或 IF NOT〕变量 = "常数"

第二个引号是需要的，否则认为后面的命令是常数的一部分。

比较两个字符串变量值的格式是：

IF〔或 IF NOT〕变量1 = 变量2

若两个字符串以相同次序包含相同字符，则认为两个字符串是相等的（大写字母与小写字母不相等）。

进行相等比较测试时，计算机不计及结尾空白，即两个字符串除结尾空白之外若相等则认为两字符串是相等的。由于这个原因，未赋给字符的变量等于全部空白组成的变量或字符串。但是，在前面的空白要予以考虑。例如 "Mr. Brown" 不等于 "Mr.Brown"。当比较字符串时，要逐个字符进行对比。即一个字符串的第一个字符与另一个字符串的第一个字符进行比较，第二个字符与第二个字符进行比较等。因此，除少数情况外，不应是判断字符串值是否彼此大于或小于。进行数的比较时，必须使用数值比较或美元比较。如果两者字数相同，可以测试判断一串数是否大于另一串数，例如邮区编码。由于邮区码可超过32767'不能依靠数值比较，需要进行字符串比较。还可利用小于或大于的比较判断字符串字母顺序。例：若要判断某个姓名按字母顺序是否出现在M的前面，可键入 IF NAME<"M" (记住仅对第一个字符进行比较)。

比较按字符的 ASCII值进行。这意味着空白小于数字、数字小于大写字母、大写字母小于小写字母。各种标点符号分散在其他字符之中。

如只需要比较变量的一部分，可将进行比较的字符数包括在变量名之后的括弧中，例：若要判断名字的第一个字符是否是M，可键入 IF NAME (1) = "M"。若第一字符是M则表达式为真，而不管其后跟什么字符。如要比较两个变量的固定字符数，必须在变量名中包括字符数，如 IF NAME (10) = NAME1 (10)。

14.3 数值比较

当数值变量作 IF 语句的开始部分时，可以利用数值比较。在数值比较中仅包括数值变量、表达式和常数。切勿将数值比较中的常数放在引号内。适于数值变量的限制也适于数值比较，即数值必须是0至32767之间的正整数。大于32767或小于零的数当零处理。比较的两边可以是数值表达式而不是简单变量或常数。例如：

IF•TIMES + 3 > %PAGE

IF &NAME < •LENGTH - 5

必须利用数值比较来比较长度变量（&VARIABLE）或系统变量（%VARIABLE）。

若将加数或减数作为比较的一部分，必须利用数值比较。

若只简单比较两数，应使用美元比较（也称为十进制比较）以避开数值变量的限制。

14.4 美元比较（十进制比较）

利用美元比较来对比变量数值与其他变量或常数。在美元比较中，直接跟在IF后的表达式必须是美元变量。它允许小数点左边有12位，右边有2位。还可用正数或负数，但不能使用数值表达式，如%PAGE+2、*TIMES-1等等。

若比较变量与常数，可将常数放在引号中。常数中可包含加号、减号、逗号、小数点和美元符号，或者只包含一系列数。若比较两个变量，也可以用美元格式置第二个变量。当进行美元比较时，计算机自动地将比较的两边置为美元格式，必要时填入零。例：10000.1成为000,000,010,000.10和-303,030.3成为000，000，303，030.30-等等。

在两边置为美元格式后，计算机对两边进行字符串比较。由于美元格式以固定格式置数，与比较字符串有关的问题已予以限制，因此可利用美元比较来判断各种关系（大于、小于和等于）。美元比较是对比大于32767或小于零的数以及小数的唯一方法。

14.5 SKIP—转跳命令

SKIP命令可以跳过部分文件而不予处理。SKIP命令本身很少使用，而是与IF语句结合使用，成为安排文件的一种有效方法。SKIP命令的格式为SKIPn，n是要跳过的文本中回车符数。遇到SKIP命令就跳过该文件，忽略命令和文本，直到遇到第n个回车符为止。所有回车均计入n内，不管它是用于结束文本行、命令行还是空白行。计算机在第n次回车后立即开始处理过程。若包含SKIP命令的命令行以回车结束，该回车不计入n内。若命令行以命令标记结束，下一回车计作为跳过的一行。若在SKIP命令中未指定数，则跳过一个回车。

可以跳至指定字符的下一次出现而不是跳过回车数。该命令格式为SKIP TOc，c是除命令标记之外的任意字符。当计算机遇到SKIP TO命令时，就跳过文本文件直到下一个c为止。计算机从c开始处理过程。

若不要打印c，应将c作为IGNORE字符建立（见“忽略字符”一节）。若c是命令行的一部分，计算机打印命令行而不予执行，因为放过了命令标记。若要跳至命令行，必须将c置于命令标记的前面。

SKIP和SKIP TO命令必须都是命令行的最后命令，因为其后的任何命令均被放过。

14.6 多重条件

每次测试不止一个条件时，需建立多重条件。多重条件有两种：AND条件和OR条件。

AND条件要求所有条件对于被执行的命令均为真。OR条件要求至少有一个条件对于执行命令为真。计算机不能识别AND或OR，但能容易地建立多重条件。要建立AND条件，可将同一命令行的所有IF语句放在条件命令前面。若IF语句中任何一个不为真，则跳过命令行的其余部分。因而为了执行该命令，所有IF语句必须是真，AND条件才被建立。

OR条件的建立比AND条件更有技巧性。有两种建立方法，可将每一条件放在不同命令行中，每行末端重复条件命令。例：若命令名为CODE的变量取值A或B，则建立新页。应键入：

\IF CODE = “A”，NP

\IF CODE = “B”，NP

由于CODE不能同时是A和B两者，所以仅执行NP命令一次。也可以建立一个OR条件，其中同时可有不止一个条件为真。这时要防止计算机执行命令多于一次。例：若要在CODE等于A或B，或者当前页数小于2时建立新页，应键入：

\IF CODE = “A”，NP，SKIP2

\IF CODE = “B”，NP，SKIP

\IF%PAGE<2，NP

若不包含SKIP命令，则若CODE等于A或B且当前页数是1，应该建立两个新页。

可将条件组合到一个AND语句中，而不写入不同的命令行。为此，通过IF后跟NOT，再跟以SKIP来变换条件。例：可将上述命令重写成：

\IF NOT CODE = “A”，IF NOT

CODE = “B”，IF NOT%PAGE<2，SKIP

\NP

若CODE等于A或B，或%PAGE等于1，则条件为假，计算机应跳至包含NP的命令行。否则，执行SKIP命令并跳过NP命令。

第十五节　标　　题

标题是一系列文本和（或）命令行。在多页文件中，该命令行在每页的开始处被处理；在多遍文件中，该命令行在每遍开始处被处理。标题不需要包含页面上打印的任何内容。可使用标题节在屏幕上打印信息、打印到盘上、修改计数器或实现其他非打印功能。

除在TEXT命令之后立即给出NP命令外，计算机不处理文件第一页或文件第一遍扫描上的标题节。若在文件开始处给出NP命令，计算机不跳至实际新页，不增加页计数。

标题的指定通过键入\HEADn命令，n是HEAD命令后的行数（以回车结束）。HEAD命令是标题的一部分。若HEAD命令必须单独在一行上，则该行末的回车不计入n内。下面是两行标题：

\HEAD2

\LEFT，SPO，：COMPANY\

\RIGHT\Page\%PAGE\

打印标题从页的第一行开始。要保证调整顶页边为标题留出足够空间。若顶页边不够标题用，则机器取得标题所需的行数打印标题并在标题后立即打印文本体。标题中的打印参数无需与文本体或底脚中的相同。必须将与文本体不同的打印参数放在标题节。完成标题后，计算机自动返回至前页结束时有效的参数。要在处理全部标题行和返回文本体之前结束标题，可键入END。

要在文本的某处改变标题，可给出另一条HEAD命令。改变标题时，前一个标题被清除。

标题存入内存中。通常分配1000个字符的存储单元用于标题。可使用HSIZE命令改变计算机保留用于存放标题的存储单元数量。

第十六节　底　　脚

底脚与标题相似，只是它在页底部。除下面指出的以外，所有适用于标题的规则也适用于底脚。

可通过键入 FOOTn 指定底脚，n 是底脚中的行数（回车数），当达到任何页（包括第一页）的页底边或给出 NP 命令时就进入底脚程序。若底脚产生任何被打印的文本，则在文本体后底页边第一行上开始打印。若要在文本和底脚之间留下一些空间，必须在底脚中包含附加行。由于底页边往往超出页的实际底边，因此必须保证为底脚留有足够的空间。当到达文件末端时，也进入了底脚程序。文件末端可以是单遍文件或外部数据文件的末端。

第十七节　打印至磁盘

用户可将全部或部分处理文件打印到磁盘上以代替在打印机上打印，也可另外在打印机上打印。开始打印至磁盘可用 DISK ON 命令；停止打印磁盘使用 DISK OFF 命令。当接收 DISK ON 命令时，就立即打开与该文本文件在同一磁盘上的具有相同文件名的 PRN 型文件。例如，若文本文件是 REPLIES.GET，则输出文件为 REPLIFS.PRN。

若打开不同名字的输出文件，可键入 DISK filename 而不是 DISK ON。这时打开指定盘上名为 filename 的文件。

若计算机发现指定名的文件已存在，就在屏幕上打印错误信息，这时可重新指定文件名。

在给定文本文件的每次处理时打开的磁盘输出文件不得超过一个。任何打开第二个文件的企图作 DISK ON 处理。

用户可以任意次地关闭和重打开相同文件。若关闭或重打开磁盘输出文件，则在中断的地方恢复文件的打印。

若需要，可以打印至磁盘而不在打印机上打印。为此必须首先键入 PRINT OFF 命令以断开打印机。恢复打印可用 PRINT ON 命令。

打印到磁盘的文件被全格式化以用于打印。当编辑另一个文件时，可在后台打印。若磁盘输出文件不包含非 ASCII 字符，还可用编辑程序进行编辑。

即使断开打印机，计算机仍将产生的文本作为打印文件的一部分来处理，并修改行和页的计数器。如果不希望把打印到磁盘的那部分文件计算到被打印页数中，必须在接通磁盘前键入 SP0 停止换行并在断开磁盘后恢复换行。若要在断开打印机时文件行的同一点上恢复打印，必须储存中断处的列数（%COL），然后标记该列（见“%COL”一节）。通过有选择地接通和断开磁盘输出文件可以建立各种文件。

第十八节　调节内部缓冲区

在装入 PRINT 后，要为三个不同的缓冲区（变量缓冲区、标题缓冲区和底脚缓冲区）保留一些剩余内存。在已装入 CP/M 和 PRINT 以及为这些缓冲区保留空间之后，剩余的内存可用来存储文本文件（注：可用 DB 命令确定三个缓冲区的当前大小和留给文本文件的

内存容量）。

在处理文本文件的第一遍时，每次将文件的一个记录装入内存。一个记录包含128个字符（也称为扇区）。如果整个文件装入内存，则不必在下次扫描时从磁盘上读文件，这样可以节省大量时间。若没有足够内存用于装入整个文件，则应尽可能多装入，在以后各次扫描时首先处理内存中那部分文件，然后再切换到磁盘。在一个文本文件中只能对缓冲区大小调整一次，并且应在文件开始的SETUP命令之后立即进行。

18.1 VSIZE—确定变量数

启动PRINT时，允许用户定义32个变量。计算机为允许的每个变量保留64个字符（7个字符用于变量名，55个字符用于变量值，名和值的实际长度各占1个字符）。也就是说，最初在内存中为变量保留2000个字符。

可使用VSIZE命令增加或减少允许的变量数（因而也就相应地改变了变量缓冲区的大小）。其格式为VSIZEn，n是0至128之间的一个数。

如果试图定义的变量数超过了允许的最大变量数，则计算机拒绝接受，并在屏幕上打印出错误信息。如果要定义的变量多于32个，必须先给出VSIZE命令，且最好在文件的最开始处给出。

18.2 HSIZE，FSIZE—确定标题和底脚缓冲区

当用HEAD或FOOT命令建立标题或底脚时，计算机将指定的各行存入标题或底脚缓冲区的内存中。最初为每个缓冲区在内存中分配1000个字符。如果试图定义大于当前缓冲区大小的标题或底脚，则在屏幕上打印错误信息。要增加或减少这些缓冲区的大小，可用HSIZE或FSIZE命令。

命令HSIZEn可置标题缓冲区大小，n是缓冲区中的字符数，其值从零到可用内存。命令FSIZEn置底脚缓冲区大小。HSIZE或FSIZE命令必须在执行HEAD或FOOT命令之前给出，最好在文件开始处给出。

第十九节　屏幕命令

MAGIC WAND提供了一些打印至屏幕的命令。用户可以使用这些命令从系统得到有关信息，写入用户的信息并对目前正处理的内容有所了解。所有这些命令可以嵌入文本中，或在运行期间从键盘给出。

19.1 *—内部注释

星号表示对所编辑者的内部注释。计算机在命令行遇到星号就忽略该命令行的其余命令，因而星号命令必须是命令行的最后命令。它可用在文本文件中的任何地方。

19.2 NOTE—屏幕信息的注释

NOTE命令将信息打印至屏幕。命令行中NOTE右边的全部内容均在屏幕上打印（用户信息不需要置于引号中）。与星号命令一样，NOTE命令必须是命令行的最后一条命令。对于占多行的信息，必须利用多条NOTE命令。

19.3 WAIT—等待

WAIT命令停止文件处理并进入等待命令屏幕。可将信息包括在WAIT命令中，使在屏幕上显示出停止原因。例如，Change print wheel。信息要写在WAIT命令的右

边，如NOTE命令一样。

计算机在停止前在屏幕上打印 WAIT 命令右边的所有内容。WAIT 命令必须是命令行的最后一个命令。对于占用不止一行的信息，可用一系列 NOTE 命令的写入信息，后跟 WAIT 命令以停止处理。

19.4 SHOW—显示

可使用 SHOW 命令在屏幕上打印变量当前值。还可用 SHOW 命令打印文字文本。必须将文字文本括在单引号（‘’）或双引号（“ ”）内。每节文字文本和每个变量均用逗号分开。

可打印各种变量：冒号（：）、数（＊）、美元（$）、长度（&）或系统（%）。还可计入 NL 命令以帮助编排屏幕上打印内容的格式。必须将空格作为文字文本包括在内，即用引号括起来。例如，NAME 的当前值是 Johe Doe，当前记录数为5。若键入 SHOW “Record no.”，%REC，” ”，：NAME，则将显示出 Record no.5 Johe Doe。

19.5 CLS—清屏幕

CLS 命令清除屏幕并将光标置于第一行第一个位置上。利用 CLS 命令并与 NOTE、WAIT和 SHOW命令相结合可简化屏幕。

19.6 DS—显示状态

当给出 DS 命令时，则清除屏幕并打印状态屏幕。状态屏幕给出所有主要形状和格式参数的当前值，并给出现在正处于文本中的位置——扫描次数、页数、行数和列数。指出正在使用的文件——文本、数据和磁盘输出。

状态屏幕还指出迄今为止文本中已定义的变量数，给出当前识别字符及当前行的源点，即是否来自文本、标题或底脚。

19.7 DV—显示变量

当给出 DV 命令时，清除屏幕并打印已在文本中定义的所有变量的名字、当前长度和值。DV命令仅显示变量的字符串值。若要以不同格式显示值，或要显示系统变量，必须利用 SHOW命令。如果你觉得可能已错误地键入某变量名，可用 DV 命令帮助找到错误，因为将会有未赋值的变量名。

19.8 DF—显示文件变量

当给出 DF命令时，清除屏幕并打印在 DATA 命令中定义的所有变量的名字、当前长度和值，还可指出当前正在处理的记录号。在文件中嵌入 DF 命令是了解数据文件被处理的当前记录的最容易方法。

19.9 DB—显示缓冲区

当给出 DB命令时，在屏幕上打印变量的当前大小、标题和底脚缓冲区以及为文本文件留下的空间，包括允许的变量数、用字符表示的标题和底脚缓冲区的大小以及可放入内存中的文本文件的记录数。

为了计算变量缓冲区中的字符数，将变量乘以64。计算可装入内存的文本文件字符数，将记录数乘以128。例如，若内存中有 100 个记录的空间，则它可以容纳包含 12800 个字符的文本文件。

计算机不指出实际上使用了多少缓冲区。

第四章　打印特性扩充

MAGIC WAND V1.1增加了一些新的打印特性，这些特性可在编辑时嵌入文本，或在打印文件时产生。

第一节　行操作

1.1　单行右平齐

RF 打印一文本行，该行与右页边对齐。文本必须跟在同一行上的 RF 命令之后。如果以前是左边对齐格式，则打印机在下一行返回到左平齐格式。此命令类似于 CTR命令。

RF 命令对于打印日期和页数很有用。例如，以左边齐平打印一信件并要以右边齐平打印日期，可键入＼RF＼Date：05/23/81＼，免得在日期前首先打入 RIGHT、在日期后再打入 LEFT 命令。

1.2　打印托架倒转

LINE－n命令可使打印机托架倒转至指定行n。这对打印文本的列很有用。例如 LINE－－3不是使打印机托架倒退三行，而是将该页定位在第 3 行。

第二节　屏幕命令

2.1　格式化文本显示

在 PRINT 程序中可使用屏幕命令将格式化文本送至终端显示。利用 PRINT 和 SCREEN ON 打印文本后，键入的命令将不影响已输入的原来文本。为永久性改变命令，必须用 EDIT 再输入文本。

用 SCREEN ON显示文本的速度，可通过键入数字9～0来选择。9的速度最慢，0的速度最快。

按 ESC 键可停止文本显示以便输入命令。这时将停止显示，在屏幕上打印命令标记并等待命令。

屏幕的开始4列用于行编号和行间隔码。文本行在屏幕上的打印与在页面上的打印一样。若行太长屏幕上放不下，则继续放在下面的行上，行号的旁边放一个加号表示续行。

用横过屏幕的连字符行指示分页。

行间隔由行号表示。例如双间隔行编号为01，03，05 等等。划底线通过打印空白行并在要划底线的文本行适当位置上放置底线字符来表示。用于底线行的行号与要划底线的文本行行号相同。

由于很多原因，左页边总是在屏幕的第5行，因而改变 LM 将改变行长度，但不改变左页边在屏幕上显示的式样。可在屏幕上正确地表示缩进。

第三节　条件新页(CNP)

有时需要改进文本的外貌，可使用 CNP 命令。在打印机打印完一段之后，如果在最后打印行和底页边之间留的行数小于 n，而下一段要打印的行数多于 n，则执行 CNPn 命令跳至新页。例如，若将 CNP4 置于文件顶部，则在每个打印段的末端，程序将计算其所取的行数以打印下一段，并与留在底页边的行数进行比较。若放不下，程序自动地跳至新页以打印该文件段。

第五章　程序员使用说明

第一节　文件定义

EDIT可用于遵照CP/M规则建立的纯ASCII文件，包括数据文件和源程序。下列说明主要用于这些非文本文件。

1．EDIT使用ASCII字符32～126（即从空格符到“～”符）、回车（13）、换行（10）和制表标记（9）。由CTRL－Z（26）或实际文件尾确定文件结束。

2．按照CP/M的标准用法，在输入时，用一个或多个空格替换制表标记符，直至下一个第8列。在保存EDIT文件时，应压缩空格以节省磁盘空间。

3．制表标记（或空格压缩）的使用与TAB键和TAB设置无关。

4．PRINT模块忽略换行，EDIT允许换行，这与标准用法是一致的。

5．在输入文件时，EDIT去除CRLF（回车换行）中的LF，而在输出经编辑的文件时恢复之。

6．EDIT接受换行符作为无特殊意义的ASCII字符，并允许换行符跟以回车符以适合Microsoft BASIC的规则。

7．回车换行对消去通常在输出文件中为每个回车增加的尾随换行符。

8．EDIT忽略每个输入字符的高位位以及输入文件中的空字符，并用空格取代任何其他非标准字符。

9．在EDIT中，每当打开文件时（不管是用于输入、显示、计入还是打印），计算机扫视第一个扇区，查找除空字符（NULL）之外的其他非标准字符，如果发现任何非标准字符，则发出警告信息。

第二节　方式设置

EDIT有三种定义操作方式的任选项。在编辑期间可在任何时候设置或重置。其中两种最常用的组合是文本方式和程序方式，还有一种是特殊方式。

EDIT选择的初始方式设置基于输出文件的类型。对于文件型为ASM、MAC、COB、FOR、BAS和PRN时，自动调用程序方式；文件型ASC调用特殊方式。如果文件型不是上述这几种，则进入文本方式。

第三节　Microsoft BASIC

与CP/M兼容的大多数语言都可直接从EDIT接受文本文件，但MBASIC需要作一些特殊考虑，因为它比CP/M出现得要早。

1．如果想使用自动行编号和重编号特性，大部分程序应该在BASIC内建立。要对以

BASIC 写的程序作编辑，首先必须以 ASCII 格式保存（例如 SAVE "PROG NAME.ASC",A)。注意，文件型 ASC 被 EDIT 认为是特殊方式，因而可与组合 BAS 文件型区分开来，由 ASC 调用的特殊方式是空格压缩任选断开的程序方式。

2．在建立文件时，应避免使用制表标记符，因为 BASIC 假定标记符在每个 第10列，而不是第8列。

3．标记处理上的差别并不影响程序的运行，但可导致程序列表时的间距错误。

4．如果在编辑 BASIC 文件时，在输出文件中使用了制表标记，可运行一个 伪编辑，将空格压缩特性断开，将标记变换为空格。

5．在 BASIC 中，单独进入的换行，存放时作为 LFCR，以区别于 CRLF，后者指明逻辑行结束。对这种情况，换行控制键由 EDIT 识别。注意，必须保证在换行符后立即 跟 以回车符，以保持和 BASIC 使用的兼容。

6．EDIT 不能避免重复计入、顺序不对和无编号行，如果出现这些错误，程序装入 可能不正常。这时，可列出成功装入的程序代码，看看在什么地方出错，重进入编辑，修改程序。

第四节　数据文件

可以使用 EDIT 建立和修改文本格式的数据文件，也可以编辑只包含 ASCII 字符的定长记录文件。

1．对定长记录文件，应设置行长度为记录长度的偶数因子，例如128字节记 录占2 行，每行64字符。

2．对定长记录文件的方式设置与文件型 ASC——MAN、MBN 和 MCN 相同。

3．由于 EDIT 不限制文本行长度，因此可建立设有单个回车的定长文件。注：如果包含回车，计算机在回车符后插入换行符（除非前面已加换行符），因而在定长记 录 中占2字节。

第五节　其他考虑

1．如果想查找某行开始处的一个字符串，例如行号或变量名，必须在该字符串前面 加回车符（即按 TAB 键）。例如：序列（SEARCH、TAB、A、B、C、RETURN 键）查找定义ABC的程序行，并将光标定位于该行始端。

2．可以使用 INCLUDE 命令将一个程序的某一部分与另一个程序合并。如果未将程序文件格式化为 INCLUDE 文件，则必须逐个屏幕翻页，直至找到希望合并的程序段。

3．如果有一些例程经常在程序中用到，应该建立例行程序文件库，在每个例程开 始 插入标识，以便可以按名将例行程序包含到应用程序中。

（编译校对　沈　江　刘运基）

Word Star 文件编辑系统

第一章 引 言

在使用 Word Star 之前，如果你不是使用 Diablb 打印机，则首先应调整磁盘以适配你的设备。请参阅第十三章“装配”。

Word Star 很容易学。备考信息系统可随时给予提示。第七、八、十和十一章可增强你对 Word Star 的了解。第八章给出成组命令，第十二章列出了错误信息。

Word Star 有丰富的命令集，其中许多命令可以分别由几个更基本的功能组合实现。开始时，你可先使用第六章介绍的命令集。

注：操作 Word Star 时，首先使用备考信息系统。菜单给出了每一个命令的说明和一些可被调到屏幕上的信息的标题。

第二章　Word Star 简介

Word Star 是一个与CP/M兼容，面向屏幕并带有集成打印的文本处理系统。文本的编辑修改可即时显示或打印出来。系统能自动处理由于长单词而引起的换行问题。大多数格式化功能出现在打印输出期间，而并写邮件（Mail-Merge）功能将为书写成批信件提供许多方便。

第一节　编辑功能

1．屏幕编辑：待编辑文本的一部分显示在屏幕上，增删或修改是立即显示的。调一篇文章仅需打入名字即可，各种编辑功能都直观地反映在屏幕上。

2．自动磁盘缓冲：文本的大小不受内存而仅受磁盘容量的限制，文本自动输入内存。

3．卷字：当一个字超过右页边时，Word Star 能自动地把这个字传送到下一行，并重新调整本行内名字的间隙。回车键仅用来控制一段的结束、空白行或其他断点。

4．自动留边、调整、对中：卷字功能已把每一整行自动地进行调整，左右对齐，留边，全行排满等。用适当的命令可以把靠边的短行调到中间。

5．重新组段：从头标位置到本段结束之间的文字可以重新组为一段。

6．高级编辑命令：除基本功能以外，Word Star 还有跳格存储（设立/清除）、块操作（复制/删除）、查找和替换（从辅助文件里读写）以及设置标记（设置/返回）等高级功能。

7．查找和替换命令：查找、替换操作可以做一次或多次。整篇字替换可忽略大小写。

8．备考信息系统：编辑时，命令菜单出现在屏幕的顶部，也可以压缩菜单而将空间让给编辑文本。命令的详细解释可以查阅备考信息系统菜单。

9．动态分页显示：编辑时，页面分隔会自行显示，并随着文本的增删自动调整。

10．文本格式的精确控制：Word Star 记住空格和回车中哪些是操作者打入的，哪些是卷字或重新组段时插入的。自动格式处理产生的结果可以用另外的命令来取消。

11．连字符：Word Star 能够让你确定将一个字分入两行时是否要连字符及连字符的位置在何处。如果页面的格式调整把这个字移到一行的中间，连字符将会自动地撤去。

第二节　并写邮件

并写邮件（Mail-Merge）是 Word Star 的一个独特的工作状态。在此状态，上节所述的打印功能仍然存在。但在打印的同时不能对文件进行编辑。同时，在这个状态中，又有一些新的“并打命令”。

1．合并：有时需将同样内容的一封信打印多份，填上不同的姓名和地址，这就叫“并写邮件”。姓名和地址可存放在一个“数据文件”中，也可以临时由操作者提供。填入信息的位置和次数可以在文本的开头设置。

2．嵌套和链接打印：在打印一个文本的过程中可以调用另一个文本来打印。同样一段文章可以被多次调用；还可以专门建立一种控制文件，用许多命令把要打的各段文件连起来，打印时只须给出一个指令就行了。

3．反复打印：同一个文件的打印可重复多次，重复命令可含于文本之中，也可由操作者在开始打印时输入。

4．打印格式整理：编辑时规定的文本格式在打印时仍然有效。同时，页边空白将会根据并写时插入的信息重新得到调整。

第三节　附加功能

1．键盘缓冲：屏幕响应跟不上按键速度时，它会完整地保存已键入的字符，让计算机按顺序处理。但在编辑、更名、删除、打印等功能初始化时，须稍加等候。

2．补缺规则：Word Star 里有丰富的可供用户定义的性能参数。在系统开启时，各种性能参数都以最常用的值为补缺值。

第四节　兼容性

Word Star 与许多硬件系统兼容，而且能和一些软件系统实现文件共享。

1．文件：Word Star 使用标准的 CP/M 文件。但由于它还具有能编辑多种语言源程序的特殊能力，所以也可看做是一个通用文本编辑的软件。

2．终端：Word Star 屏幕编辑在H89或 Z89 终端上运行。屏幕每帧为25行，每行 80 字符。命令菜单是白底黑字，以和正文相区别。

3．打印机：Word Star 可驱动几乎所有种类的打印机。既可用 CP/M 命令，也可用 Word Star 的指令来执行打印过程。如使用菊花轮打印机或其他增量打印机，则可自由设置行间距（1×1/48″到127×1/48″）和字间距（1×1/120″到127×1/120″），而且系统还会自动进行微量的调整。如使用其他类型的打印机，则只提供两种字间间距和有限的格式控制。上、下角码，黑体字等在不同的打印机上有各自的打印形式。

第三章　计算机、操作系统和文件

第一节　终　　端

屏幕一般分为16×64字符或24×80字符两种，最左边的字符位置称之为第一列。Word Star可以通过相应的控制命令使屏幕显示上下翻卷、快速换页。键盘一般采用标准键盘，CTRL键的用法就象SHIFT键一样，它用来和字母键一道组成控制字。控制字表示成↑A、↑B等，Word Star都是用控制字作命令。

第二节　磁盘驱动器

除终端外，系统还必须带一个或几个磁盘驱动器以存放文件。注意各种驱动器的使用方法。运行Word Star时，换盘时是有规定的（参阅本章第十节），不可随意换盘，否则会造成系统或文件的损坏。

第三节　操作系统

WordStar在CP/M操作系统下运行。WordStar依靠操作系统对磁盘文件进行存取，用户也可直接调用操作系统提供许多功能。这些功能命令很多，与Word Star有关的，有：

1．确定磁盘上的自由空间（在多数系统中是STAT命令）；

2．确定某个文件的大小（STAT文件名）；

3．拷贝文件（PIP命令），这个命令既可以用来拷贝文件，也可以用来对一块盘片进行初始化（见本章第九节）。在Word Star中，也有用来进行文件拷贝的命令。

第四节　文　　件

文件是存在磁盘上的一些字符的有序集合。文件中很重要的一类是正文文件，或称为文本，例如信件、书籍、账单等。输入文本的时候，键入的字符顺序地记载在磁盘上，包括所有的空格和每行后面的回车。注意，回车后面自动跟有一个换行，但在少数地方也有例外，这在后面将要提到。

第五节　文件命名

文件通过文件名来调用、编辑、存取和打印。文件名是在建立文件时给出的。文件名可以用CP/M系统命令（REN）或Word Star中的改名命令进行更改。

文件名由三部分组成：驱动器名、文件名和扩展名。如果驱动器名省略，则为现行驱动器。例如下列文件名是合法文件名：

A：ABC.D　　B：FOO　　B：ABRACADA.BRA
　　ABC.D　　B.FOO　　RKT.TAK

第六节　联机驱动器

联机驱动器（有时也称为现行驱动器）的驱动器名在文件名中不必出现。当CP/M操作系统启动时，A驱动器被定义为联机驱动器。联机驱动器也可定义成其他驱动器，这只需在CP/M命令状态下打下驱动器名和一个冒号（如B：），或用Word Star命令（后文叙述）去实现。Word Star显示磁盘文件目录时，仅显示联机驱动盘。因此，如果想知道B盘上的文件目录，必须先将B驱动器定义为联机驱动器。

第七节　磁盘容量和文件大小

磁盘容量一般以字节为单位。一张8英寸单密度盘的容量大约为241K，而5.25英寸盘的容量约为90K。磁盘上除了存储正文文件外，还有许多系统文件，例如Word Star、PIP、STAT等。磁盘一旦满了，就不能再存入新的内容。所以，当你键入一个新文本或在旧文本上增加内容时，先检查一下磁盘上的自由空间（用STAT命令）。如果所剩空间不多，应启用一块新盘。

第八节　后备文件

为了防止意外事故，所有的文件都应有拷贝的副本。这种副本最好是拷贝在另一张磁盘上，并加以妥善的保管。拷贝命令一般是PIP，在Word Star中，也有专门的拷贝命令。还有一个拷贝命令DUP，它是将一张盘上的全部内容拷贝到另一张盘上去。

第九节　工作盘生成

使用之前，应先拷贝一张Word Star的工作盘，而将源盘妥善地保存起来。拷贝命令是DUP、COM或DISKCOPY.COM。

Word Star程序必须在CP/M系统支持下运行，所以在工作盘上还应进行系统生成（CP/M），用命令SYSGEN完成，可参阅CP/M手册。Word Star的所有源文件分布在两张5.25英寸的源盘上，或一张8英寸的源盘上。对于5.25英寸源盘来说，两张盘如下：

A盘

WS.COM　Word Star主文件，在现行驱动器上将其调入内存以后，可对文本进行插入、删除和其他编辑操作。当硬件设置有所变动时，它必须作相应的改动。

WSMSGS.OVR　包含Word Star的所有命令菜单和错误信息。

WSOVLYI.OVR　这是一个Word Star的支持文件，它使用覆盖技术分块调入Word Star程序。

B盘

INSTALL.COM　这个文件将指导你使 Word Star 系统适配硬件环境。

EXAMPLE.TXT　这是一个文本的例子，有助于你对 Word Star 的了解。

使用 Word Star 时，必须使用两种工作盘：一种是系统盘，一般只一张；另一种是文本盘，可以有多张。

系统盘一般在驱动器A中，它上面应该有下列文件：

1. CP/M操作系统，用 SYSGEN 命令生成。

2. Word Star 系统程序。1° WS.COM、2° WSMSGS.OV R、3° WSOVLYI.OVR、4° MERGPRIN.OVR，此程序可选，只在需要 Word Star 的并写邮件功能时，此程序才必须处于系统盘上。

3. 其他CP/M的辅助操作命令文件，例如 STAT.COM。

文本盘一般放在驱动器B中，它专用来存放用户输入的文本。

在 Word Star 运行的时候，联机驱动器中必须始终有盘。

第十节　何时换盘

在 Word Star 运行过程中，当非文件菜单显示在屏幕上，而不再进行编辑或打印时，也可以换盘。这时不需按↑Z。可以用内存中的 Word Star 程序对不同的文本盘进行处理，即使这些文本盘上并没有 Word Star 程序。这样可以节省许多磁盘存储空间。

在编辑和打印过程中是不能换盘的。所以，在调用 Word Star 之前，必须检查是否有充足的磁盘空间，供你的文本编辑之用。如空间不足，应先换盘再调用 Word Star。

第十一节　文本编辑时的文件变换

无论是对已有的文件进行编辑，还是欲建立一个新文件，都必须在调用 Word Star 的同时给一个文件名。

当对已有的文件进行编辑修改时，修改后的文本暂时存在一个“工作文本”之中，一直到给出 SAVE命令后，这个工作文本才转为正式文件，而原来文件则转化为 BAK 文件。但新键入的文件执行 SAVE 操作后，不会有 BAK 文件生成。

工作文本并不久留于磁盘上，所以，应及时地给出 SAVE 命令。Word Star 提供了一条“存盘并重编辑”命令，将编辑和存盘两个操作合为一体，使用起来比较方便。

如果你对所进行的编辑修改工作不满意，可以用 ABANDON 命令取消工作文本。如果此时工作文本已存成正式文件，原来的文件已经改成 BAK文件，则可用 CP/M 系统命令或 Word Star 中的命令将其转化为正式文件。

第四章　文本格式介绍

第一节　行的形成及有关概念

1．页边：左、右页边决定了每一行的长度，每一行长度的补缺值是65，可以通过命令来改变。

2．字间空隙：两个字之间的空隙可以是一个字符位置，也可以是两个或多个，补缺值是一个。

3．右页边的对齐或散乱方式：行的右端可以上下对齐，在一些行的字间适当插入些空格，这叫“页边调整”。也可以不进行页边调整而任其散乱。这两种方式中，页边调整是补缺形式。

4．行的形成：

（1）在左边插入空格，使各行左对齐。

（2）输入足够的字填满一行，如果有多余，则暂存起来以放入下一行。

（3）进行左页边调整，在字间插入必要的空格。

（4）在右端加上一个回车，如果采用字间双空格，则加上两个回车，以此类推。

5．卷字：Word Star的卷字功能给用户提供了极大的方便。操作者在一段文章未打完以前不需要打入回车。当某一个字在行右端打不下时，Word Star会自动将这个字调到下一行的开头，而对这一行进行右调整。这就是卷字功能。

6．格式重整：文本格式经常需要重新调整，它是通过反复使用上述的几种操作而进行的。原先附加的空格和回车必须移动到新的适当位置上。

7．硬回车：这是由操作者打入的回车，一般用于一段的结束和表格标题的换行。如要在一段后面加一空行，需要在回车后面再打一个回车。硬回车的位置不随格式调整而变化，屏幕上显示文本时，凡含硬回车的行的最右端有一个“>”作为标志符号。

8．软回车：这是由Word Star进行格式处理时加入的回车。这种回车在格式重整时会调整位置，屏幕上显示文本时，含软回车的行的最右端是空格。

9．硬空格：操作者打入的空格，不随文本格式的变化而变化。

10．软空格：在格式调整时自动添加的空格。如果文本的格式发生变化，这种空格会增加或减少。

11．连字：Word Star具有连字功能，即把一个本行放不下的长字分成两段，放一段到下一行去，而在前一段最后加连字符。连字符的位置可自由选择。

第二节　打印格式

1．页面：分页可由操作者控制，如不加控制，则一页打满才换一页。每页有一个页码。

2．上、下角码，黑体，下划线等：这些特殊打印是通过特殊控制字符来实现的，要特

殊打印的部分用控制字符圈起。段中的任何部位都可以采用这些特殊的打印形式。

3．打印机功能和打印命令：字高、字宽和色带变换等打印机功能和打印的格式都可以由文本中的嵌入指令来控制。Word Star有两种打印指令：打印控制字符和点命令。

第三节　打印控制字符

打印控制字符用来指示打下划线的文字的起始和末尾、一些特殊符号的打印位置和色带的变换。例如：↑S可以用来指示带下划线文字的始末。要打印：The word underline is underlined.应键入：The word ↑S underline↑S is underlined.又如：↑H可以将其后的一个字符打在前一个字符的同一位置上。要打印：derrière 应键入：derrie↑H're

注意，所有这些打印控制字符都仅在↑P前缀下才有效。

第四节　点命令

点命令是送入文件的特殊的命令行，它用来确定打印位置、设置页号以及开始新的页面等。点命令控制打印过程，它本身并不打印出来，且都有补缺形式。

点命令必须紧接着回车键入，它以句号打头，后面紧跟着两个大写字符指明命令功能，再后面是空格，空格后跟着数字或其他参量，这要依据命令来定。点命令最后用回车结束，回车以前还可以有注释，这些注释和点命令本身一样，不被打印。例如：

·HT 5　　（每页顶都留5行位置的页边）

·HE section II（"section II"字样打印在每页的顶部，一直到下一条·HE命令为止）

·CP 12　　（如果本页剩下不到12空行就换页）

点命令象文本的其他部分一样，可以被显示和编辑，但由于打印时不出现，故它并不影响页面的划分。

错误的点命令被当作注释，不予理睬。而在这一行的最右边特征字符栏中出现一个问号。

由于点命令混杂在文本之中，所以必须注意，不要使文本段落开始（即上段的回车之后）的第一个字符是句号。

第五节　动态分页显示

在文本显示时，页的页面分隔处将出现一条横杠，后面跟一个P字：----------P

分页分无条件和有条件两种，分别由点命令（·PA）和（·CP）控制。

影响动态分页显示的点命令还有：·LH（行宽）、·PL（页面长度）、·MT（顶部页边宽度）和·MB（底部页边宽度）。这些点命令仅在文件开头打入才有效。

如果点命令的位置错了，屏幕上将会显示错误信息（见第十二章）。这时，打印时的分页结果可能与命令动态分页不一致。

第五章　Word Star 的调用和非文件命令

第一节　启动 Word Star

在启动 Word Star 之前，必须做好下列准备工作：

1．检查工作盘上的文件是否齐备（参阅第三章第十节）；

2．检查 Word Star 是否适配你的终端和打印机等硬件设备（参阅第十三章）；

3．学会引导（冷启动）CP/M 操作系统。CP/M 命令状态的提示符是“A>”，在提示符后键入适当的命令，以调用 Word Star：

方法一：在A>后键入 WS (CR)（基本方法），这里 (CR) 是回车。打了回车几秒钟后，屏幕上将出现一个非文件菜单（见下节）。

方法二：直接进入文本编辑。例如：

A>WS LETTER.DOC (CR) A>WS B：ABC.XYZ (CR) 这里 WS 空一格，然后跟上文件名。键入回车后，Word Star 将首先调入内存，并立即开始对指定的文件进行编辑。

方法三：用两张盘进行编辑。这是编辑特别长的文件时用的，例如：

A>WS A：BOOK.DOC B：(CR)

使用了这个命令后，Word Star 首先被调入内存，然后对 A 盘上的文件 BOOK.DOC 进行编辑，新的文本则存放到B盘上去。编辑结束时，A盘上的原文件将改为 BOOK.BAK，而B盘上的新文件称为BOOK.DOC。如果这时还给出“存盘并连续编辑命令”，那么则倒过来，从 B 盘上调文件进行编辑，而存到A盘上去，这个过程可以一直这样倒换。

第二节　非文件命令

用上节第一种办法，即不带文件名调用 Word Star，或当一文件编辑完毕，Word Star 将进入所谓 “非文件命令状态”，“editing no file” 的信息将显示在屏幕上方，下面是“非文件菜单”，即列出了在这个状态可以键入一系列命令，要键入某一条命令，只需键入等号前的一个字母即可（大、小写一样），而且不需回车，Word Star 立即开始执行这条命令，同时，这个命令被显示在左上角。见表 5-1。

第三节　备考信息层次

在使用 Word Star 的过程中，屏幕上方总是提供命令菜单及命令的解释，给用户以很大方便。然而，缺点是屏幕上被占去了很大一块地方，供待编文件显示的区域太小了。

当用户有了相当的经验对命令比较熟悉之后，一定想扩大文件显示区域，这就要重新设置“备考信息层次”，简称备考层。备考层可为 0、1、2 和 3，其中第 3 层为补缺，信息最详细，从 2 到 1 到 0，信息越来越简单。

表 5-1　　非文件命令

命令	功　能	说　明
D	编辑文本	要求键入一个文件名，然后开始编辑。已有的或新建立的文件均可。
N	编辑非文本	同上，但编辑的是“非文本文件”，有动态页面功能，其他补缺功能也不一样。
X	退到CP/M系统	从Word Star中退回到CP/M系统状态。
H	设置备考层次	菜单罗列信息可简可繁，也就是备考信息的详细与否分成几个层次。H命令可用来设置备考层次。
Y	删除文件	要求键入文件名，并删除之。同CP/M中的ERA命令。
L	变换联机驱动器	显示现行联机驱动器名(见第三章第七节)，并请求键入新驱动器名。
F	文件目录显示功能开/闭	按F，显示功能打开，再按，则关闭，如此反复。
P	打印文件或暂停打印或恢复打印	如F命令，按P，开始打印，再按，暂停打印，再按，恢复打印，如此反复。详见第十一章。
M	文件合并打印	从一个数据文件中取数据，并到另一个文件中去打印，以形成信件之类的文稿。使用M命令时，A驱动器盘上必须载有文件MERGPRIN.OVR，否则出错。
R	运行程序	无需退到CP/M系统状态，使用R命令就可以运行程序。按入R以后，屏幕出现：COMMAND? 这时如按入STAT .COM，就可以得到磁盘自由空间的数量。
O	拷贝文件	按O以后，屏幕出现： NAME OF FILE TO COPY FROM? NAME OF FILE TO COPY TO? 如果分别键入A：FIRST.DOC和B：，则A盘上的文件FIRST.DOC就被拷贝到B盘之上。
E	文件更名	按E以后，屏幕出现： Enter NAME OF FILE TO RENAME? NEW NAME? 如果分别键入A：FIRST.DOC和SECOND.DOC，则A盘上的文件FIRST.DOC就被改名为SECOND.DOC。

设置备考层的是非文件命令 H，或者编辑命令 ↑JH（见第八章第十一节）。一键入命令，屏幕则显示：

```
H                    editing no file
  HELP LEVELS
    3 all menus and explanations displayed
    2 main editing menu(1—control—char commands)suppressed
    1 prefix menus(2—character commands)also suppressed
    0 command explanations(including this) also suppressed
  CURRENT HELP LEVEL IS 3
  ENTER space OR NEW HELP LEVEL(0, 1, 2, OR 3):■
partial DIRECTORY of disk A: ↑Z=scroll up
  CHAPTR1.DOC CHAPTR1.BAK CHAPTR2.DOC CHAPTR2.BAK
  CONTENTS    FILE1.DOC   FILE1.BAK   FILE2.DOC
  LETTER.DOC  LETTER.BAK  TEST.DOC
```

我们可以看到各备考层的定义：

3——列出所有命令和解释。

2——主编辑命令菜单（单字符命令）被压缩。

1——前缀命令菜单（双字符命令）被压缩。

0——所有备考信息被压缩。

在光标处键入希望的层次，再按入回车，就得到新层次的备考信息。

第六章　如何使用编辑程序

本章介绍文件编辑的全貌，为进一步学习打下基础。这里介绍了许多命令和使用这些命令的环境。

我们建议打开你的计算机，启动 Word Star，输入一个 D 和一个文件名称，来开始文件的编辑，试试这里所讲的每一条命令，这样将使你较快地熟悉 Word Star 的功能。

当开始文件编辑时，例如，用非文件菜单（见第五章第二节）中的 D 命令，如果文件不在软盘上，Word Star 首先在几秒内显示信息：NEW FILE。如果这个信息不出现，那么该文件已存在于软盘中。

接着，Word Star 进入文件编辑状态，对于一个新的文件并处于 3 级备考层时，屏幕显示如下：

```
A：TEST.DOC     PAGE 1 LINE 1 COL 1          INSERT ON
CURSOR：↑A＝left word          ↑S＝left char     ↑D＝right char
                                                 ↑F＝right word
          ↑E＝up line          ↑X＝down line
SCROLL：↑Z＝up line            ↑W＝down line     ↑C＝up screen
                                                 ↑R＝down screen
DELETE：DEL＝char left         ↑G＝char right    ↑T＝word right
                                                 ↑Y＝entire line
OTHER：↑V＝insert off/on       ↑I＝tab          RETURN＝end para
                                                 ↑U＝stop
          ↑N＝insert a RETURN  ↑B＝reform to end para
                                          ↑L＝find/replace again
HELP：↑J displays menu of information command
PREFIX KEYS ↑Q ↑J ↑K ↑O ↑P display menus of additional commands
  L----! ----! ----! ----! ----! ----! ----! ----! ----! ----! --------R
```

屏幕的顶部是状态行（STATUS LINE），A：TEST.DOC 是正在被编辑的文件名称，页、行和列将随着你输入文件或移动光标而变化。

假定你不改变备考层（3 级），那么主菜单（MAIN MENU）占据屏幕上面几行。这个主菜单提供各种命令键的功能概貌。

在菜单下面是尺线（RULER LINE）：

```
L--------! --------! --------! --------! --------! --------! --------R
```

L 表示左页边，R 表示右页边，而 ! 是跳格停留位置。

菜单和尺线是白底黑字，以便与被编辑文件相区别。

尺线下面是文件显示区域（FILE DISPLAY AREA）。对于新的文件这里是空白，对于已有的文件，则显示开头的几行。

要输入正文，只要按键即可，除了控制字之外，键入的所有字符都会如实地进入你的

文件。

如果你打入一个字超出右页边时，可以注意到 Word Star 将把这个字移到下一行，光标停在这个字面上，让你继续打入，这叫卷字（WORD WRAP）。你还可以注意到，Word Star 还在上一行的名字之间加上适当的空格，使各行的右边对齐，这叫调整（JUSTIFICATION）。

当一段没有打完时，不能用 RETURN 键——让卷字功能去自动换行。行宽设置、页边调整功能屏蔽、字间双空隔等特殊要求可参阅第八章第六节。

要改字或增加字，只需把光标移到所需要的位置打入所需要的内容即可。

要移动光标，可使用菜单上列出的光标上、下、左和右的命令。需要移动多个字符的位置时，使用左字和右字命令比左/右字符命令来得更快。↑H 和 BACKSPACE 键也能使光标左移一个字符。

如果光标停在正文中的某个位置，这时打入新字符，就可以看到新字符插在光标之前，而光标之后的原文被向后挤开，这就是插入（insertion）。编辑一开始的状态就是插入状态，插入状态可以通过打入↑V而退出，也可以通过↑V重新进入。退出插入状态时，打入的字符将代换原来的字符，这就是置换。状态行指出了系统是否处于插入状态。

在一段中间进行了插入或置换操作后，光标后面的部分如果不再需要，可以用↑G命令来进行删除，按一次，删除一个字符，同时后面的内容依次前推。

如果你打入文本时，发现了错误，可以用 DELETE 键。每按一次 DELETE 键，就删除光标左边的一个字符，光标随之右移。接着你可以打入正确的字符。

其他的删除命令列在菜单中，并将在第八章第四节中叙述。

在做完一段文本的修改后，右页边可能不整齐了。其中一些行可能太短，而另一些行可能太长，以至于需要两行来显示（此时，第一行的最右边有一个“+”来标明）。为了使右页边排列整齐，可用↑B命令：把光标移至第一个不规范的行当中或前面，打入↑B，Word Star 就开始整理工作，一直到这一段结束（即 RETURN，在屏幕的最右边有一个<号）。

当段重整理命令（↑B）正在工作时，它可能遇到一个长字，这个字无法写在一行当中，得用连接符连接。当这种情况出现时，Word Star 将停下来显示一个提示信息，让你按“-”键（如果你想用连接符连接这个字的话）。如不用连接符连接，可按↑B键来继续进行整理。这个“连字”（Hyphen-Help）的性能将在第八章第六节中详述。

整理一个长段要花费几秒钟，在这期间，“↑B”出现在屏幕的左上角，仅在整理完成后或遇到一个长字需要用连接符连接而停下时，整理过的文本才显示出来。

如果你想插入一个新段，就先让插入位置上空出几行，然后在空行上插入文本。这种作法，显然比较方便。插空行的命令是↑N。

当屏幕打满后，光标每换一行时，屏幕将卷上一行，如果你打得快，屏幕翻卷可能跟不上，但你停下来时，屏幕继续翻卷，一直把你打入的命令执行完为止。

偶尔，字 WAIT（等待）可能在状态行中出现，通常还伴随有磁盘驱动器发出的咯哒咯哒的声音，当这种情况发生时，最好停止打字或缓慢地打，等待字 WAIT 消失。

当你已输满一页（补缺行数）时，屏幕上将出现由破折号“——”组成的分隔行，行的最右边有个P字：

----------P

已告诉你，打印时将从这里分段。

这就是动态分页显示（DYNAMIC PAGE BREAK DISPLAY）。在第十章中，将讨论用“点命令”来随时另起一页（尽管当前的页未满），或者来设置页的各种格式，例如行数。

为了查看不在屏幕上的文本部分，可用菜单中的“卷上一行”（“scroll up line”）、“卷下一行”（“scroll down line”）、“卷上一帧”（“scroll up screen”）和“卷下一帧”（“scroll down screen”）等命令。还有，当光标已在屏幕的底部时，使用“光标下移”（“cursor down”）命令可卷上一行。而当光标在屏幕的顶部时，使用“光标上移”（“cursor up”）命令，则可卷下一行。

象“scroll down line”这样的命令，可立即使屏幕内容向下移动。你不必等待上一条翻卷命令执行完毕再打入新的命令。如果你想向下卷动5行，可快速地打入5个↑W命令，而机器会尽快地一个不漏地执行。

你将注意到，编辑命令将影响文本的存储形式，例如，如果把光标不断向右移动，当光标移到一行的末端后，它将移到下一行的左端。这是因为文本中包括一行行从左到右顺序排列的字符，其中有“回车”字符和下一行的字符，“回车”字符是在上一行的最右端和下一行的最左端之间。

如果你向右移动光标，直到它要移动到下一行时，给一个“删除右字符”（↑G）命令，回车就被删除，屏幕上下一行的头与这一行的尾将连在一起。可见，回车符象一般字符一样能被编辑。

前缀（prefixes）：除了单个控制字命令之外，Word Star还有许多双字符命令。其中第一个字符则称为前缀，双字符命令以↑O、↑J、↑K、↑Q或↑P开头，这5个前缀列在主菜单的底部。

前缀菜单（prefix menus）：如果你打入一个前缀，一秒钟之后，带这个前缀的所有双字符命令在屏幕上被列出。这样做稍微慢一点，如果你知道双字符命令本身，可以快速地打入两个键，这时将直接出现前缀命令菜单。

如果打入一个前缀，又不想要其菜单上的任一命令，可打入一个空格或其他在菜单上未定义的键。这时主菜单将再次出现。用这种方法，可调阅所有的前缀菜单来检索任一命令。

双字符命令的第二个字符可以带CTRL键，也可以不带，字母既可小写，也可大写。

例如按↑Q键可建立↑Q前缀菜单：

```
↑Q  A:TEST.DOC PAGE 1 LINE 3 COL 19
  ↑Q PREFIX                (to cancel prefix, hit SPACE bar)
CURSOR: S=left side screen E=top screen X=bottom D=right end line
        N=beginning file C=end file  0-9, B, K, V, P=to marker
SCROLL:        T=continuous up       Q=continuous down
DELETE TO END LINE:  DEL=left    I=right
FIND,REPLACE:      F=find a string  A=find and substitute
REPEAT NEXT COMMAND: O=repeat until key hit
L----! ----! ----! ----! ----! ----! ----! ----! ----! --------R
This is text entered by the user
```

The quick brown fox jumped over the lazy dog!
abcdefghijklmnop

你能看到新的光标移动命令：移到顶行，移到文件末尾，删除一行的左部分和右部分的命令，查找和置换命令以及一些解释不大明显的命令。所有这些命令将在第八章详述。

↑K前缀菜单包括一些很重要的命令，用于存盘和打印：

```
    ↑K PREFIX                    (to cancel prefix, hit SPACE bar)
END EDIT/SAVE :    D = done edit   X = done, exit   S = save reedit
                   Q = abandon
MARK BLOCK :       B = mark start  K = mark end     H = hide/display
BLOCK OPERATIONS : V = move block  C = copy block   Y = delete block
                   W = write
ADDITIONAL FILES   R = read file   W = write block  J = delete file
& PRINTING :       O = copy file   E = rename file  P = print a file
DISK & DIRECTORY : L = change logged disk          F = file directory
                                                       on (OFF)
PLACE MARKERS :    0-9 = set/hide place marker 0-9
```

注：本菜单略去了状态行、尺线和文件名。

保存（saving）：正如在第三章第十一节中曾说的那样，在编辑期间，被输入和被变更的文本是一个过渡性的工作文件（working document）。如果要把文件留下备用，必须发出保存（save）命令。基本的保存命令是↑KD。它将工作文件以编辑启动时选取的文件名保存下来。接着系统返回第五章第二节中所述的非文件菜单。

在↑K前缀菜单上的↑KP是打印命令，用于编辑过程中打印的启动、停止和继续（不在编辑状态时，非文件菜单上的P命令能准确地执行↑KP的同样功能）。只有被保存的文件（不是正在编辑的工作文件）才能被打印，所以，通常打印出来的文件与正在编辑的文件不同。

打印文件（printing document）：打印功能将在第十一章详述，基本的打印步骤如下：首先，你想打印的文件如已输入，则须存盘（↑KD）；其次，准备好你的打印机和打印纸；然后打入P命令（如正在进行编辑，则用↑KP命令）。此时，Word Star将回答：

NAME OF FILE TO PRINT;

打入文件名，并按RETURN键。每当按一次回车，都将回答若干个问题，然后，打印开始，同时非文件菜单（或正在被编辑的文件）将回到屏幕上。当打印正在进行时，可以给出其他命令。

当一个文件正在打印时，你能编辑另一个文件，但是，键盘响应将变慢，所以，我们建议：打印时，只可查看文件，偶尔进行慢速修改等编辑工作。

↑O前缀菜单包含许多命令，这些命令允许采用字间双空隔、变换页边和跳格间距、屏蔽卷字功能以及执行与屏幕格式有关的其他功能：

```
        ↑O PREFIX : on-screen formatting commands
S = set line spacing  C = center cursor line  F = margins/tabs from file line
L = set left margin   X = margin release      E = soft hyphen-entry off(ON)
```

R = set right margin W = word wrap off(ON)
D = –, print ctrl dspy off(ON)
I = set tab stop J = justification on(OFF)
P = page break display off(ON)
N = clear tab stop V = variable tabs off(ON) T = ruler display off(ON)
G = paragraph tab H = hyphen-help off(ON) SPACE = cancel prefix

另外，↑O菜单表明：在相应命令后的括号内，指出了命令，诸如卷字、行调整等的开或关，在上例中，仅有行调整是关（OFF），其他所有命令都是开（ON）。

为了输入打印控制字符，可利用↑P前缀菜单命令：

↑P PREFIX : put control character in file
V = subscript begin/end T = superscript begin/end
Y = ribbon color change
S = underscore begin/end B = boldface begin/end
D = double strike begin/end
A = alternate pitch N = standard pitch
X = strikeout begin/end
O = non-break space F = phantom space G = ph.rubout (see manual)
C = pause when printing H = overprint next character
RETURN = overprint next line
Q, O, E, R = user printing controls SPACE = cancel prefix

为了调菜单方便，↑J前缀菜单展示了各备考层的控制命令，参考信息随时供查阅：

↑J PREFIX help commands
H = display and set help level M = margins and tabs
F = flags in right screen column S = status line
I = command index : entering text R = ruler line
B = paragraph reform(↑B command) V = moving text
D = dot commands print controls P = place markers
SPACE = cancel prefix

↑JH和非文件菜单上的H命令一样用来显示、说明和设置备考层，其余的命令（指↑J前缀清单上的命令）显示各菜单的注释。例如，↑JV说明怎样移动文件块，↑JI帮助查找各种功能的命令，↑JD概括打印指令。其中一些备考命令的信息需几个屏幕来显示，此时，要打一个键，让Word Star换屏幕。

备考层（help levels）：如果用↑JH命令把备考层从3变到2，则主菜单将不显示。在备考层2中打双字符命令时，如果打了第一个控制字，再在打第二个字符之前暂停一下，那么在这段暂停的时间间隔中，前缀菜单将出现。如果把备考层变到1，那么前缀菜单也不再显示。更进一步把备考层变到0，则各种单个的命令，如非文件D命令或备考层命令的注释也都不再显示。

Word Star的所有编辑命令将在第八章中详细叙述并被列成“编辑命令”表。第七章详述屏幕显示。

第七章　屏幕显示

在文件编辑期间，屏幕显示包括状态行、菜单、文件目录（如果允许的话）、尺线和文件显示区域。

第一节　状态行

在文件编辑期间，处于屏幕顶部的状态行通常包括：

正在输入或执行的命令（假如有的话）；

正在输入的文件名；

页号；

行号；

列号；

插入开（如果插入是开的话）及其他有关短语。

执行中的命令（假如有的话）显示在屏幕的左上角，例外的是：快速单字符命令，如象“光标下移一行”命令，则不予显示。

其次，三状态行的顶通常是页nn、行nn、列nn，它们分别是打印页数、在一页上的打印行数和在光标位置上字符的打印列数，它们随着光标的移动而不断更新。

假定文件用的页数是从1递增被打印，页是印出的页数，如果通过“点命令”把不同的印出页数列成清单，则这些页数将不在状态行映出。行是印出的行，并不是计数点命令的行，也不是计数比屏幕宽度更长的文件行在屏幕上的续行。

列是印出的列，不是屏幕上的列。当行包含一非打印字符，如打印控制字符（例如↑S来产生在文本下边加横线）时，或当文件行比显示屏幕的宽度更长而在屏幕上显示二行或更多行时，或当标志信号出现在显示行时，将印出不同于屏幕上的列。

如果用“编辑一个非文件”（“edit a non-document”）命令（即N命令）曾启动过编辑，或当页分段显示被禁止（用↑OP命令）时，则用FC = nnnnn，FL = nnnn来代替页和行。其中，FC = nnnnn给出了文件字符数，或光标和文件开始之间的字符数加1，这是计数被存储在文件中的所有字符（字节），其中包括回车、换行等。FL = nnnn给出了文件行数，或光标和文件开始之间的文件行数（包括点命令行）加1。

所以，通过↑QC命令，把光标移到文件的末尾并读出FC = nnnnn项，你能确定以字符为单位的文件的大小。

在状态行中的剩余项表示下列短语：

WAIT 表示 Word Star 正在对软盘读或写，这时需停止打入或用很缓慢的速度打入，否则可能会遗漏字符。

MAR REL 表示取消页边控制时，出现此项信息。

decimal 表示只要十进制数是向右对齐，机器对十进制数加上标记停。

INSERT ON 表示插入打开，即打入的字符插入在文件中，而不是置换文件中的字符。插入的开和关是通过打入↑V命令来实现的。

LINE SPACING 表示除了当单空隔是有效外，行空隔能用↑QS命令来设置。

PRINT PAUSED 表示打印暂停。

REPLACE (Y/N) 通过置换命令，是否置换显示在状态行。

注意：当上述几个短语出现时，则LINE SPACING项从屏幕的右侧消失是正常的。

第二节 菜 单

菜单可在状态行下边出现，菜单列出了各种编辑命令。

主菜单在补缺备考层是3的条件下列出了单控制字符（非前缀）编辑命令，当备考层是2或1和前缀键（如主菜单所示）按下时，在一短时间暂停之后，带有前缀的命令菜单被显示。菜单和状态行、尺线一样都是以白底黑字显示，从而区别于文件文本的显示。

在某些条件下，一定数量的解释性和警告性的信息在菜单的上下显示出来，例如：

TYPE ↑KP TO CONTINUE PRINT

（解释，打印暂停时出现）

* * * WARNING : WORD TOO LONG TO FIT MARGINS

（警告，当一行太长而又没有相应的处理命令时出现）

这些信息，在按下一个键时会消失。产生这些信息的条件详见第十二章。

第三节 文件索引

一般在编辑期间，文件的目录不显示。如果需要，可以用↑KF命令来显示目录（当输入一个文件名称时，打入控制字F，可以临时显示文件目录），文件目录显示在菜单和尺线之间。

通常，当一个文件在编辑时，Word Star仅仅显示部分目录，以便留出屏幕上的更多空间来显示文件。主编辑菜单上的↑Z和↑W命令可将屏幕上的目录上下翻卷，以便于观看所有文件的名称。

文件名是按字母排列的，后备文件（·BAK）放在相应的正式文件之后，非文本文件如·COM和·INT放在最后。临时文件（·$$$）也放在目录的后面。但是，如果目录很长，那么对它不按字母顺序显示也是正常的。

当菜单，或由各种命令引起的问题，或其他信息也都占用了屏幕上许多行时，文件目录（偶尔也包括尺线）将暂时不出现，一旦屏幕空间许可，目录将再出现。

第四节 尺 线

菜单和文件目录（当显示时）下边是尺线：

L----------! ----------! ----------! ----------!----------!----------R

尺线指示当前左页边（L）、右页边（R）和可变化的跳格停留位置（!）。十进制数格

式的跳格停留位置是（*）（见第八章第六节），页边之间的非跳格列以“-”来显示。如果L或R的设置出现在跳格列，则将以！或*来表示。

如果用↑OG命令（见第八章第六节）将左页边移动，则L不移动，但临时页边的左边部分将以黑底白字显示。

当页边控制被屏蔽（↑OX命令，第八章第六节）或卷字功能关闭（↑OW命令，第八章第六节）时，跳格标记可停在页边之外。如果所置右页边比屏幕还宽，将出现一行或多行尺线，可以用↑OT命令来消除和恢复尺线显示。如尺线显示被关闭和目录显示被打开，那么显示目录时，目录和文本之间是用白底黑字的“=”组成的横线隔开。

第五节　文件显示区域

屏幕的下部显示编辑文件的文本。屏幕光标总与文件光标的位置相一致。

屏幕上的最右列是空白或显示一个“特征”字符来指明特殊类型的行或--个“hard”回车符，以此来说明以下区段。

特殊行：文件显示区域中的各行通常将打印在文件的一行上；例外的如下：

延续行：如果文件中的一行比屏幕上一行要长，则这样的文件行被延续在屏幕的下一行上，并用“+”插在第一行的最右列为标志，如果这样的行很长，它可以在屏幕上显示出三行、四行或更多行；则除掉最后一行外，所有行将在最右列插上“+”特征位。当光标在屏幕上任一延续行时，在状态行中的列数映出了印出的列。

重复打印行：为了特殊用途，它能够把文件中的二行或更多行，一个接一个地打印在同一行上，从而形成一个特殊的图案，在屏幕上，凡特征位是“-”的，则指明下一行将重复打印在本行上。“-”的形成见下章。

分页行：文本分页时，屏幕上显示：

----------P

注意：文本本身并无此行，它仅为显示用。用↑OP命令能关闭和打开分页行显示。

文件行的结束：当文件在屏幕的底部之前结束时，用句号（·）作为特征位的空白行填满剩余的屏幕。对于一个新文件开始也用上述空白行填满屏幕。

文件行的开始：如果显示文件向下移动到文件的开始所在显示区域的顶部之下，那么在文件开始行之上的屏幕部分用冒号（：）作为特征位的空白行填满。

普通行：文件显示区域中的普通行表示它既不比屏幕上的行长，也不是由下一行重复打印过的行，普通行能以“硬”（“hard”）或“软”（“soft”）回车字符作为结束。“hard”回车符用“<”特征记号来指明，“soft”回车符用“空白”特征记号来指明。

下例表示输入二段文章和一个表之后屏幕的外貌，它直观地说明“hard”和“soft”回车符的显示。非菜单表示通过备考层被置为2或小于2的情况。记号■代表光标。

```
                A:FILE 1 DOC PAGE 1 LINE 13 COL 1
L----!----!----!----!----!----!----!----!----!----!----!----R
This is paragraph of text entered without using the RETURN
key;word wrap formed the lines,The line breaks will be
moved if the user invokes reformation,Note that the flag
```

```
character column is blank, except on the last line, where a
indicates that RETURN key was used to end the paragraph.    <
                                                            <
The following list was entered using the RETURN key between
lines, because the line breaks were desired in these
positions. Note the<'s in the rightmost column:              <
     bread                                                  <
     hamburger                                              <
     orange juice                                           <
     yogurt                                                 <
■                                                           <
```

下面的例子表示利用一些文本输入后给出屏幕的外貌。注意：状态行映出光标在第一行的98列上，因为这一行比屏幕宽度还长，所以在屏幕上利用一个延续行。文件行的开始、一个分页行和文件行的结束也都表示在屏幕上。

```
  A:FILE2 DOC  PAGE 1 LINE 1 COL 98          INSERT ON
L----!----!----!----!----!----!----!----!----!----!----!----!----R
                                                               :
This is the first line of a short file.It is a very long line,  +
intended to be printed on a wide printer.  ■                    <
This is the 2nd line.It is short enough to fit 1 screen line.   <
PA go to a new page here                                        <
-----------------------------------------------------------------P
This is the last of the file                                    <
```

文件字符显示：由 Word Star 编辑过的文件可以包括除去控制码—Z（1AH）之外的所有ASCII码（码值为0—7FH）。大部分终端仅仅显示打印字符（码值为20—7EH），Word Star利用字符的组合来显示其他代码。

表 7-1 不可显示字符的显示

字符	显示为
Tab (consrol-I：09H)	输入若干个空格来把光标移到后面8的倍数列上。注意：这个字符是非文件中的公共字符，但它不用来作 Word Star 变量的停止标记。
Other control characters	∧和字母或标点字符（除了1FH和1EH之外，它们在内部被用来代表“soft”连字符并以白底黑字的“-”来显示）。
Delete (7EH)	～（否定号，7EH）在文件中出现的可能性很小。

控制字母是文件中的公共字符，用来控制打印效果。如在打印字符下面划线，显示为∧和一个字母。虽然在屏幕上出现二个字符，但仅有一个字符在屏幕中并作为单一的字符被编辑，例如，一个“删除字符”（“delete character”）命令将导致屏幕上∧和一个字母都消失。

第六节　标志符

大部分特征字符能出现在文件显示区域的最右列中，表 7-2 概括说明了所有的特征字符。

表7-2 特征字符

字　符	意　　义
(blank)	表示以“soft” 回车字符为行的结束。这个行的回车位置通过后来的卷字或段落调整操作可以被改变。
<	表示以“hard” 回车字符为行的结束。这个行的回车位置不随后来的卷字或段落调整操作而变化。
+	表示屏幕上的下一行是这个文件行的续行。
-	表示下一行将重复打印在这一行上。
•	表示这一行在文件的结尾之后。
:	表示这一行在文件的开始之前。
P	表示下一行将作为新页面的开始，当分页显示是打开的（↑OP），它才出现。
?	表示本行包含一个不认识的或可能是错误的“点命令”，当点命令打入时，它也出现，但在非文本编辑时，它不出现。
J	表示没有“回车” 的“换行”，这是标准的 Word Star 中从不使用的非标准文件格式。
M	表示本行包含一个 “并打点命令”(Merge-print dot command)。见第十六章。

第七节　屏幕更新

在每一个命令的结束，或每一个被输入文本的字符处理完之后，屏幕进行更新；否则屏幕显示不被更新，必要时，等待（WAIT）出现在屏幕上。在屏幕更新过程中，如果你打入一个字符，则屏幕更新暂停，直到处理完打入的字符后再继续进行更新。

例外情况是：↑Z和↑W命令（移上和移下一行）有时需打入多次，这时，可以使用键盘上的重复（REPEAT）键。

提示：所有终端设备都具有文本上卷的功能，但在这些设备上若没有“行删除”（“Line delete”)代码，那么只有在文件显示区域之上无菜单时，Word Star 才卷动而不是重新显示。如通过减少备考层的层数来禁止菜单显示，则用户的不灵活（dumb）终端将得到更快的上卷功能（例如用↑Z命令）。

第八章　编辑命令

本章分类列出和解释编辑命令，如有的命令可以归入几类，则在每一类中都列出。

第一节　光标移动

光标移动命令可以列成下表：

命令	功　能	说　明
↑S或↑H或BACKSPACE	光标左移一格	注意，S、D、E、X四个键恰好排成左、右、上、下的位置。
↑D	光标右移一格	
↑E	光标上移一行	
↑X	光标下移一行	
↑QE	光标移至顶行	注意，在上面四条命令中加上Q，就得到这四条命令。
↑QX	光标移至底行	
↑QS	光标移至行首端	
↑QD	光标移至行末端（不一定在屏幕右边缘）	
↑A	光标左移一个字	参阅下文“字”的定义。
↑F	光标右移一个字	
↑QR	光标移至文件首	如果光标处于一个长文件的后部，则用↑KS命令更快。
↑QC	光标移至文件尾	
↑Q0、↑Q1直到↑Q9	移光标设标记	详见第八节。
↑QB	移光标至块首	这里“块”指作上特殊记号的部分，详见第九节。
↑QK	移光标至块尾	
↑QP	将光标移到上一命令执行之前光标原所在位置	例如，在存盘命令后，打入↑QP，则光标回到刚才正在编辑的地方。又如在↑B（段格式重整）之后，↑QP也使光标移为原处。
↑QV	将光标移到上一寻找或替换命令执行之前光标原所在位置	此命令通常在寻找、替换命令执行中发生“未找到”(NOT FOUND)信息时使用。详见第七节和第八节。
↑QF	寻找	详见第七节。
↑I	跳格（TAB）	详见第三节。
RETURN	回车并换行	

一、“字”(WORD) 的定义

被空格、逗号或其他标点隔开的一串字符称为一个“字”。单独的一个回车也被当作是一个“字”。

二、光标移动的限制和克服办法

用编辑命令无法使光标移到文本显示实际区域之外，例如一般最后一行只有几个字，光标就不会移到回车之后。解决的办法就是不用编辑命令，而用空格键或 TAB键来右移光标。注意，这时插入功能应关闭 (OFF)。

在↑D、↑H 这样的带控制的字符显示时，光标总是停在 D、H 上，而不停在符号↑上。

第二节　屏幕翻卷

翻 卷 命 令

命令	功 能	说 明
↑Z	卷上一行	光标总是停留在某字符上，跟着一起上卷，直移到屏幕顶部为止。在屏幕上有文件目录显示时，按↑Z命令使目录上卷一行，文本显示不动（见第十二节）。
↑W	卷下一行	
↑C	卷上一帧	实际是上卷一帧的3/4。
↑R	卷下一帧	
↑QZ	连续上卷	按命令时，屏幕显示 TYPE 1～9 TO VARY SPEED, SPACE TO STOP 意思是可按 1～9 等数字以确定翻卷速度，1 最快，9 最慢，不按则约定为 3，按其他任何键则翻卷停止。
↑QW	连续下卷	

第三节　正文输入

有关输入的基本命令

命令	功 能	说 明
↑V	插入的开/关	按↑V 可以使编辑在插入和替换两种状态中来回转换，在状态行中有明确指示。
return	结束一段“硬”回车	在插入状态，按 return 键即按入一个“硬”回车，在替换状态，则可将“软”回车换成“硬”回车。
↑N	插入“硬”回车	在一行中插入“硬”回车，光标不动，光标之后部分移至下一行。一般在文本输入时用 return，而修改文本时用 ↑N 比较方便。
↑I	跳格	对于插入状态，光标跳过的部分均作为空格，对于替换状态，光标跳过的字符不变。
↑OI	设置不等距跳格	常用于列表格，见第六节。
↑ON	清除所设不等距跳格	见第六节。

↑OF	在尺线上设置跳格和页边	见第六节。
↑P（X）	在文件中插入控制字符	这里（X）代表要插入或要替换的控制字符，例如↑PS将在文件中插入↑S。在设置打印控制字符，象打印下横杠、黑体字、角码时需用此命令。

第四节　正文删除

删除命令

命令	功能	说明
↑G	删除光标所指字符	如果光标处在行的末尾之后，则删掉回车。
delete	删除光标左边字符	如果光标处于行的首端，则删掉上一行末的回车。
↑T	删除右边的字	删除光标右边的整个字和所跟空格。如果光标处于字的中间，则仅删除右半个字及所跟空格。如果光标处于两字之间，则仅删去两字之间的空格。
↑Y	删除光标所在的一行	注意，这里的一行指的是打印的行，而不是屏幕的行。
↑QY	删除一行中光标后面的部分	行末尾的回车仍保留。
↑Q delete	删除一行中光标前面的部分	
↑KY	删除一块	见第八节。

第五节　存盘和废弃

存盘和废弃命令

命令	功能	说明
↑KD	文件编完，存盘	回到非文件菜单。
↑KX	存盘，并结束编辑	回到CP/M系统状态。
↑KS	存盘，并继续编辑	回到编辑主菜单，如再按↑QP，则光标会跳到刚才存盘前所处的位置。在文本输入过程中，应注意要经常存盘。 ↑KS也是使光标迅速回到文件开头的办法。
↑KQ	编辑废弃	所作的编辑工作被废弃，不存盘。回到非文件菜单。

第六节　屏幕文本格式

屏幕文本格式与有关命令

命令	功　能	说　明
↑OC	行移中	光标所在行移至屏幕中间。此行前后的空格不予理睬。
↑OL	置左页边	请求输入一个1～240的数以确定左页边，然后按回车(RETURN)键；或者按ESCAPE键使用当前列数（即状态行中COL项显示的当前列数）。
↑OR	置右页边	输入最右列数，或按ESCAPE键使用光标所在的当前列数。
↑OF	根据现有文件行格式设置页边和跳格位置	如果某一段落中行的页边和跳格位置已合乎需要，则可以用↑OF命令将其借来用于当前文件的编辑。
↑OG	利用跳格停留位置来设置左页边	例如，第一个跳格停留位置是第6列，按一次↑OG命令则使左页边移至第6列，按两次↑OG命令则使左页边移至第二个跳格停留位置。↑OG命令的功能仅限于一段之中，遇到RETURN后消失。
↑B	段落重整	将光标所在行移到本段结束之前的部分进行格式调整。在页边改变、字间空格改变或文本增删之后，往往需要用这个命令。
↑QP	把光标移到前一个命令执行所在位置	参阅第一节。
↑OX	页边释放／关闭	按下↑OX命令，右页边释放，此时键入字符可超出右页边之外，就象卷字功能消失一样。再按一次↑OX命令，页边又恢复正常。
↑OW	卷字功能关闭／打开	按一次功能关闭，再按一次功能恢复。
↑OJ	行调整关闭／打开	打开时，“软”空格自动插入各字间，右页边为上下对齐方式。关闭时，不插入“软”空格，连原有的也会撤去，右页边采用散乱形式。
↑OI或↑Otub	设置跳格停止位置	可键入一个列数使每隔几列设置一个跳格停止位置，新的设置立即在尺线上反映出来。
↑ON	清除跳格停止位置	可键入要清除跳格的停止位置的列数，如键入“A”则全部清除。
↑I或tub	跳格	按尺线上的“↓”所示位置跳。参阅第三节。十进制数跳格方式可参阅本节后部。

↑OV	可见跳格功能 关闭／打开	打开时，跳格之间的位置都以空格形式进入文件。关闭时的情形参阅第九章第六节。在通常情况下，应使功能打开。
↑OT	尺线显示／消失	
↑OP	页面显示功能 关闭／打开	详见第七章第一节。
↑OD	打印控制字符显示与否的控制	文本中有许多打印控制字符，从而影响文本的精确面貌，观察时可用↑OD使这些控制字符隐去。编辑时又可使用↑OD让它们重现。
↑OH	连字功能关闭／打开	打开时，操作者移动光标至所需连字符位置，然后按入“-”键。
↑OE	“软”连字符输入的开／关	详见本节后部：连字符功能。
↑P-	输入“硬”连字符	详见本节后部：连字符功能。
↑PO	输入一个“固定”空格	输入一个↑O进入文件，代表一个空格，这个空格既不被卷字所清除，也不允许行调整时增加其他的空格。
↑PH	退格打印	显示为↑H，打印时使后一个字符打在↑H的前一个字符之上。
↑P return	两行重叠打印	这里的“回车”不带“换行”，显示时，前一行右端有特征字符“-”，打印时，后面所跟的一行重叠打印在前一行之上，以形成特殊的打印效果。
↑O	状态显示	↑O菜单显示下列功能的开关状态： 连字功能 可变跳格 页面显示 卷字 行调整 打印控制字符的显示 “软”连字符的输入 尺线的显示

屏幕文本格式参数的补缺值

项　目	文本编辑	非文本编辑(N命令见第九章第五节)
左页边	第一列	同左
右页边	64字符宽屏幕：第60列	同左
	80字符宽屏幕：第65列	同左
可变跳格停止位置	间格5直到56列	同左
可变跳格功能	开	关

卷字	开	关
行调整	开	关
尺线显示	开	关
打印控制字符的显示	开	开
软连字符输入	关	关
分页显示	开	不可操作
连字功能	开	关

注：无论你是否已换了编辑的文件，任何参数的改变将保持下去，除非已用了 N 命令进行了非文本编辑。反过来也是一样。

下面简要说明上列命令的实际使用方法。

一、使用卷字功能输入文本

如上表所示，行的宽度、字间空格等格式参数都有补缺值，所以可直接输入文本。如果想改变这些参数，则可使用相应的命令。

在输入文本时，最重要的一点是在一段中间不要按回车键。回车键仅用来结束一段，或产生空行（连按两次回车）。

要将某一较短的行（例如文章标题）调至屏幕中间，则应把光标移至这一行的任一字符上，再按命令↑OC。

二、段落的调整

1.一段当中的修改

在一段当中进行了插入或字符替换以后，要按↑B 对这一段进行重新调整，同时注意这时的连字功能最好是关闭的，不然，段落重整过程会经常停下来，而要求插入连字符。

在输入文本时如发现输入的内容有错，可以停止输入，将光标移上去修改；修改完毕后，按↑B，再按↑D，则光标回到刚才离开的地方，让你继续输入。

在删除操作的时候，往往连段落最后的回车也删去了，这一点应注意避免。每一个段落的结尾都应该有一个“硬”回车，即在屏幕右边可以看到一个标志符<，如果发现某一段结尾行右边没有这个标志符，说明这是用了“软”回车结尾，应该替换为“硬”回车。

2.在两段当中插入一段

将光标移到后段首行的左端，按↑N，则插入了一个“硬”回车，且光标处于回车之前，这时可将要插入的段落键入。要在两段当中插入空行，也用↑N。

3.将一段分为两段

将光标移到分段的地方，按↑N，则将一段分成了两段，然后分别对上一段和下一段进行调整（↑B）。

4.将两段连结为一段

将光标移到上一段的末尾，按↑G，删去两段之间的回车，然后按↑B 重整段落。

三、行格式的调整

行格式调整主要有以下几个方面：

左右页边的改变，右页边采取对齐形式还是散乱形式，字间空格是一个还是多个等。这些可以分别用命令来设置，然后用↑B 对段落进行重整。例如，我们希望某一段的某几行排窄一点，让右边空出一块地方画图，就可以用↑OR 置右页边，例如 40，再用↑B 命令，则

光标的下部分全变成左半幅状态；然后移动光标到下面适当部位，再用↑OR 命令和↑B 命令将右页边恢复原状。

注意，↑B 命令总是处理光标右边的内容，因此，在使用↑B 命令以前，首先要把光标移到需要调整的段落之前。

四、段落序号的处理

使用命令↑OG，可以很方便地使段落序号醒目地排列在正文左边。按下序号 1 和一个句号，再按↑OG 命令，光标就跳到第 6 格位置，等待输入正文，一直到这整个段落结束，每一行都是从第 6 格开始排列。当然，也可以一开始就将左页边定义为从第 6 格开始，然后在键序号前，释放左页边（用↑OX 命令），这样，序号就可以从第 1 格键入。

五、表格式文本

表格式文本要求的是每一栏上下对齐，因而不能使用卷字和段落重整等操作。最方便的方法是使用↑AB 键，让各栏分别从某一跳格停止位置开始排列，特别注意每行都要有一个硬回车，这样就能保证各栏上下对齐，而且整个表格不受段落调整的影响。

使用点命令.UJ 也能避免表格受到破坏，参阅第十章第二节。

六、利用尺线来设置左右页边和跳格停止位置

例如，要使每一行从第 5 格开始，到 50 格终止，跳格停止位置在第 10、20、30 格，那么可以在文本中输入一根尺线：

```
---------! ---------! ---------! ---------! ---------! ----------
```

让光标处于尺线上，再按命令↑OF，就得到需要的行格式。很明显，这对于设置表格非常有用。

为了使上面这根尺线不致于在打印时也打进文本当中，可以在尺线上再加一行（仅有两点），变成：

```
..
---------! ---------! ---------! ---------! ---------! ----------
```

然后让光标处于尺线之上，再按↑OF。

七、连字符功能

连字符功能有助于文本显示格式。在“连字符功能开关”开时，↑B 命令将寻找出在一行末尾放不下的长字，让光标停在适当的地方，等待操作员回答是否要连字。操作员还可以选择适当的地方，这是为了使连字符合语法规则。

此时输入的连字符是“软”的；如果以后被调整到一行中间则不会显示和打印，但是它们仍然留在文件中；如果再置于行尾又会显示出来。

在使用连字符功能时，一定要把开关打开（可以通过↑O 命令检查），确定页边和字间间隔，把光标置于一段的起始，然后使用↑B 命令。此时光标位于 Word Star 认为应该插入“-”号的地方，敲“-”键则将连字符插入。如果想改变连字符的位置，则把光标向左右移动，然后按“-”键。如果不想用连字符，按↑B，调整将继续进行，只不过整个字将放到下一行去。

上面这样插入的连字符称为“软”连字符。对应地，如果在键入正文时键入了一个连字符，则称为“硬”连字符。硬连字符无论在一行的什么位置，都会显示出来。“软”连字符是白底黑线，而硬连字符是正常的黑底白线，可以清楚地区分开。

注意，在早期的 Word Star 版本（释放级 2.0）中，软连字符处于一行中间时，也会被打印出来。

八、十进制数跳格制表规则

在打印统计表的时候，我们希望十进制数是按小数点上下对齐，而不象字串那样左对齐。为了做到这一点，应该采用另一种跳格方式，即十进制跳格方式，这时，尺线上惊叹号表示的跳格位置应换成 * 号。

设置十进制跳格方式前，应检查可变跳格功能（variable labbing）是不是开的(可用 ↑O 检查菜单)，如不是，应该按命令 ↑OV。

按入命令 ↑OI，再按 * 号，再按入列号或 ESCAPE键，这样就可以设置十进制跳格停止位置；也可以设置一根尺线，在相应的跳格停止位置按上 * 号，然后用 OF ！来承认。惊叹号！和 * 号也可混用。

在键入十进制数的时候，可以发现，第一个数字出现在 * 号所指位置上，整数部分的其他数字，每键入一个，都出现在 * 号所指位置，而把已有的数字往左边挤，直到键入小数点，这个过程才结束；随后键入的小数部分和通常的情形一样，逐个往后排，而小数字一直处于 * 号所指位置上。

要键入第二栏数字，应按 TAB 键；一行数字键入完毕，应按回车键，再按下一行。

九、比屏幕宽的文本

在宽行打印机上，一行所能打的字符往往比屏幕上一行多得多。Word Star 可以在一行内设置多至 240 个字符，以满足宽行打印机的需要。

屏幕显示的时候，一个宽行显示成两或三行，最后一行的右边缘特征字符是空格，而未结束的行的右边缘特征字符是 + 号。

十、在同一位置打印多个字符

有时需要在同一位置打印多个字符以形成特征记号，这可用 ↑PH 命令实现。例如，光标记号▀可以由字符 E、I、N、Z打在同一位置而得，即键入：

E ↑PHI ↑PHN ↑PHZ

注意，屏幕显示为：

E ↑HZ ↑HN ↑HZ

十一、重打一行 (OVERPRINT UNES)

几行文字有时需要打印在同一行上。例如，可以在同一行上打印分别由 I、N、Z 和H 等字母组成的四行字符，形成了一个光标行。

如一重打行跟在点命令后，它被认为是点命令的一部分而不会被打印，这个特点可用来在文件中保留一个控制行，此行不会被打印但可为 ↑OF 命令服务，进行设置页边和跳格停止位置的工作。

重打行是这样输入的：先输入第一行，然后在行尾打 ↑P.RETURN键，接着输入要重打的行。如果还想再重打一行，再打 ↑P.RETURN 键，输入重打行的内容。↑P.RETURN 的作用是只输入一个回车字符到文件中去，去掉了换行字符。在屏幕中，如果某行将被其下一行重叠，则右边缘特征字符是“-”。

第七节　查找和代换

本节详细地介绍了查找与代换功能。

简述：

查找（↑OF）功能使你能方便地把光标移到文件中的某个位置，而不必在屏幕上通篇去找。它同“继续查找／代换”命令↑L联合使用，能方便地找到文件中出现的所有给定的字、短语和其他字符串。使用查找命令找到第一个出现的位置，在屏幕上观察且做了相应的改动后，按↑L去找下一个出现的位置。

代换（↑OA），该命令能方便地用另一个字、短语或字符串等来代换现有字、短语或其他字符串，而无需专门去找，再删除老的和换上新的。选用“全部”（global）选择项，就可用一个代换命令将文件中所有匹配串都代换掉。在做每个代换之前，该命令在屏幕上显示匹配到的内容并且询问在这种情况下是否要做代换，如果不需要询问，可以进行快速代换。

继续查找（或代换）命令，允许在两次代换之间使用其他编辑命令，而不需要重新键入↑OF或↑OA命令。

代换还有许多有用的功能，如插入一个常用字串到文件中等。

一、查找命令的基本用法

键入↑OF启动查找命令，然后Word Star询问：

FIND?

你可在此询问后面键入所要找的任何字符序列，然后按回车键。以后，我们将把你在这里键入的字串称为查找字串。查找字可以是一个字、一个短语或其他不超过30个字符的任意字符串。键入查找字串后，Word Star又询问：

OPTION (? FOR INFO)

在此，你先只按回车键，现在讨论这个问题。

Word Star然后就在文件中找字串下一次出现的位置（注意，从目前光标位置找起），一旦找到，光标就停在找到的字串的最后一个字符的后边，屏幕重新显示同时结束查找命令；如果没有找到查找字串，光标就停在文件的末尾并显示“未找到”的信息：

＊＊＊NOT FOUND＊＊＊（查找字串）＊＊＊按FSC键＊＊＊

例如，假定你刚开始编辑一个文件，此时光标在文件的开头，要把光标移到第一次出现“section II”的位置，键入↑OF，然后回答：

FIND? section (CR) OPTION? (? FOR INFO) (CR)

(CR)指按回车键，黑体字表示是你键入的内容。

注意，基本的查找命令将准确地匹配你的查找字串。比如，上面的例子中不会找到“section␣␣II”（字之间有两个空格）或“section”。另一方面，即使它隐含在其他的字当中也可以找到。在上述的例子中，如果“section III”出现在“sectionII”之前，找到的将是“sectionIII”。如果“page”、“agent”在“age”之前出现，在找“age”时也将找到它们中的“age”。处理这种不希望的匹配很简单，只需在查找字串的两端各加上一个空格。

查找命令不区分软（soft）硬（hard）空格、软硬回车。

如果查找字串是空串（null）（只按回车），光标将不动，这可让你在回答询问时取消不

需要的查找命令。

二、代换命令的基本用法

代换命令首先进行查找，然后用操作员指定的字串来代换查找字串。

按↑OA 启动代换命令，Word Star 同查找命令一样询问“FIND？”(找什么？)，你键入查找字串后又询问：

REPLACE WITH？（用什么代换？）

输入你用来代换查找字串的字串。然后，Word Star 又询问“OPTIONS？”(选择项？)，在此你暂且只按回车键。

此时 Word Star 开始搜寻查找字串。如果没找到，提示“未找到”(NOT FOUND)信息并将光标置于文件的末尾；如果找到了，则找到字串的上下文部分显示在屏幕上，并且状态行中显示：

“REPLACE (Y/N)”

此时光标在找到的字串前后跳来跳去，如果你想做代换，就按 Y，否则可按任一个别的键。如果你想继续往下进行同样的代换，按 YL 就可从现行的光标位置开始继续执行。

三、查找与代换选择项

在执行查找和代换命令时的询问“OPTION？”能够用一个或多个字母如 G、N、W、U 或 B（大小写均可）以及数字（1到65535）来回答。也可键入问号（?）让选择项菜单显示出来，再回答询问，最后打一回车。

各个选择项的作用如下：

数字：指进行查找或代换的次数。例如，在用查找命令时，它的意思是要找到查找字串第几次(比如第4次)出现的位置。当用在代换命令时，意思是要代换下面找到的4个匹配字串。数字可以不只一位数。如果找不到那么多的匹配字串，将提示没找到(NOT FOUND)错误信息。

G：在整个文件范围内代换。先把光标移到文件的首部，然后重复代换（每次都询问），直到文件结束。只当一个匹配字串也没有时才提示“未找到”（NOT FOUND）错误信息。执行完代换命令后，光标停在文件的末尾。如果查找命令用上 G 选择项，意思是找文件中最后一个匹配字串的位置。

N：代换时不询问。如果你确信不会有不希望的匹配发生，则可将 N 与 G 连用。

B：从光标所在的位置倒向搜寻，如提示了没找到（NOT FOUND）错误信息，则这时光标停在文件的开头。如与 G 连用，查找和代换都是从文件末尾开始直到文件开头。

U：找匹配字串时不区分大小写。使得“age”也能匹配到“AGE”或“AGe”等。该选择项在查找与代换时都能用。

W：只匹配独立的字串。避免“age”匹配“page”或“agent”的情况发生。

注意：W 找不到刚好出现在文件首尾的匹配字串。也就是说，在匹配字串前后至少要有一个字符（注意字符，包括空格与回车等）。

另外，关于选择项要注意的是：如果你用 ESC 键，而不用回车键结束你对查找命令中 FIND?询问的回答（或代换命令中 REPLACE WITH 的回答）就不会再向你提出选择项询问。

在下面的查找和代换选择项的例子中，操作员键入内容均用黑体字表示。

FIND? JONG (CR) OPTION? (? FOR INFO) 13 (CR)

寻找光标位置以后第13次出现的“Jones”。

FIND? section (CR) OPTION? (? FOR INFO) wub (CR)

找前一个（b）以单个字出现的(W)字串“section”，不分大小写(u)，执行该命令将找到Section，“section”，“SECTION”，但不会找到“dissection”或“sectional”以及”sub-section”。

FIND? John Jones (CR) REPLACE WITH? Jomes D. Smith (CR)

OPTION? (?FOR INFO) gn (CR)

把文件中所有的（g）“John Jones”都换成“Jomes D.Smith”，每次代换不向用户询问（N）。

FIND? carraige (CR) REPLACE WITH? carriage (CR)

OPTION? (?FOR INFO) gn (CR)

同上，校正文件中所有拼错的词“carraige”。

四、查找字串中的特殊字符

在响应询问FIND? 时，可以使用下列的控制字符以匹配某一类字符：

↑A：匹配任何单个出现的字符（如果查找字串本身含有↑A，则应按↑P↑A）。

↑S：匹配任何非字母和数字的字符。注意，既然↑S有从正在输入的字串中删掉一个字符的作用，因此，当查找字串中本身含有↑S时，必须按↑P↑S。

↑OX：匹配除了X以外的任何其他字符。这里X是紧挨着↑O输入的任意字符。

↑N：匹配后面的回车键，换行符（通常隐含在两行之间）。注意，对软、硬回车不予区分。↑N也可以用来响应询问REPLACE WITH?（用什么代换），此时，则产生一个硬回车作为查找字串。

此外，也可以使用第九章叙述的响应询问(question-response)的特殊字符——↑S、↑D和↑T纠正按键错误；↑P把控制字符输入到字串中；↑R用来恢复先前的响应等等。

几个用来说明上述特殊字符用法的例子：

FIND？↑N sectionII (CR) OPTION? (?FOR INFO) (CR)

找到只出现在行首的“sectionII”。

FIND？↑N (CR) OPTION? (FOR INFO) 50 (CR)

找到光标所在的行以后的第50个回车，即光标向下移动50行。如果在给出命令时光标在文件的开始，则找到了第51行。

FIND？↑N↑N (CR) OPTION? (CR)

找到后面最先碰到的一个空行（因为是两个回车相邻，结果光标停在空行的下一行）。

FIND？X↑A↑AX (CR) OPTION? (? FOR INFO) (CR)

找到两个X，它们之间有两个任意字符，比如：“XABX”，“X_1，X_2”，“X+X”等等。

FIND？↑N (CR) REPLACE WITH? ¦↑N¦ (CR) OPTION? 13n (CR)

把下面出现的13个回车改为¦回车¦，在改的过程中不询问，也就是要当前的行及下面的12行的末尾加¦，在下面的13行的开头插入一个¦，这种命令用在给表格的四周形成一个框。

注意：在查找字串中使用特殊字符时，一定要清楚，通常认为的两行之间的“回车”实际上是两个字符：一个回车，一个换行符。这个事实会影响↑A和↑S在查找字串中的使用。

五、查找和代换过程中的软连字符（soft hyphens）

"软连字符"是一个特殊字符，一个字可以从它所在的位置分开到两行，软连字符不能打印，除非通过使用卷字(word wrap)或段调整（paragraph reform）（↑B）把它移到行的末尾。

如果你打开软连字符输入开关(↑OE)在响应询问FIND?(找什么？）或REPLACE WITH?（用什么代换？）时，按"-"键，将在响应字串中加入一个软连字符，这个用法可以用来找软连字符，用代换命令在文件中加入软连字符。查找命令对所有的连字符同样对待（无论能否打印），用代换命令加入到文件中的软连字符一般不打印出来，除非在段落重整中它正好处于行的末尾。

当软连字符输入开关打开时，如先按↑P再按"-"，则这样输入的连字符是"硬"连字符，总是可以打印的。

六、查找和代换用法提示

1.查找长字串：通常只需用长串的一部分（5～10个字符)来响应FIND?(找什么？），如果这样找到是错的，按↑L继续找，出现"没找到"(NOT FOUND)错误以后，按↑QV使光标恢复到起点。

2.全部代换的两个方法：1)可用选择项G，如果处处都代换还可以与N连用。2)按↑QR使光标回到文件开头，输入只代换一个的代换命令，它将代换掉第一次出现的匹配字串；然后再按↑L，代换下面的。后一种方法使你能够在两次代换之间使用其他编辑命令，但进行下一次代换时只需按↑L。

3.处理大文件时的注意事项：如同在其他章节（如第九章第二节)所叙述的一样，使光标倒向移动很长距离的命令，在大文件中使用时要格外谨慎。这是由于处理的时间太长，而且由于占用临时的文件空间，从而导致出现"磁盘满"(DISK FULL)错误。建议在要把光标移到文件开始时，先用↑KS重存文件，这用在选择项是G的时候。如果光标远离文件开头，在做全部代换时要先按↑KS，同样在使用命令↑OV时，如果光标远离起点，也要先用↑KS。

4.从现行的光标位置开始全部代换的办法：用一个足够大的计数值如9999来响应选择项询问，所有从光标当前位置以后的显示都被代换掉，然后将会出现错误"没找到"(NOT FOUND)。使用大的执行次数与选用G选择项的区别在于前者不必把光标移到文件的首部。

5.撤消一个不希望的查找代换命令：下面两种方法都行，1)只用回车键响应询问，2)按中断命令↑U。

6.停止正在执行中的查找或代换命令：可按中断命令↑U，即出现"中断"(INTERRUPTED)错误，这时光标的位置不定。

7.加速全部代换命令的执行：正常情况下，即使用了选择项G、N或重复次数，代换命令也将显示每次代换，屏幕的变化反映了代换命令的执行时间。如果你在这种命令执行的过程中按任何一个无关紧要的键（如光标向上），每个代换执行完后就不再显示，这样代换命令的执行就快多了。在命令结束后可以看到全部代换的结果。

8.停止全部代换命令时应注意：如上所述，在加速代换时（在全部或重复计数代换过程中按任意一个键），将不显示正在进行这样一个命令。如发现它会对文件产生不良后果，就用中断键↑L停止它的执行。

9.常用的插入操作：用空串（只按回车键）作为代换命令中的查找字串，要插入的字串作为代换字串并选用N选择项，这样就把代换字串插入到了光标所在的位置。以后每按

一个↑L，代换字串都被插入到现行的光标位置。

10.查找和删除字串：使用代换命令时，如果代换字串是空串，该命令将找到查找字串并用空串来代换掉它（即删除它），按↑L键继续执行删除。

11.把其他文件变成Word Star文件来进行段调整：为了转换一个非Word Star文件以便对它的各段进行调整，必须把每段中的回车都变成空格，然后再进行调整，先把光标移到要调整的第一段的开始，键入↑OA，屏幕上显示：

FIND? ↑N (CR) REPLACE WITH␣ (CR) OPTION? N (CR)

注意：要用一个空格来回答询问REPLACE WITH?（用什么代换?），执行这个命令将把第一个回车换成空格，把第一、第二行连起来并把光标移到原先第二行的开头。按↑L继续代换直到这段中所有的回车都用空格换掉（注意：最后一个回车保留）。这时，该段变成了文件的一行，在屏幕上显示成若干行，最右列显示'+'表示两行是连着的。把光标移到本段开始，然后按↑B，就得到一个格式化了的段，此时光标停在段的末尾。

把光标移到下一段重复上面的处理过程（如果段之间没有空行，光标已经在下一段的开头了），直到整个文件结束。

12.使用响应前一次询问的键入内容：假定你想重复刚才的查找命令，但要用不同的选择项，如果查找字串很长重新搞很烦，就可以按↑OF，然后用↑R来响应询问（FIND?），先前的回答就会显示出来。你可以按回车键保留原先的回答，也可对显示的内容作修改再按回车。接着照常回答选择项询问（OPTION?）。

这种用以前的回答来响应同一询问的方法，也适于询问REPLACE?（用什么代换?），OPTION?（选择项是什么?）以及Word Star所有以问号结尾的提问。第九章第一节的询问－响应（question-response）控制字符给出了更多的解释。

查找代换及有关命令

命令	功能	说明
↑OF	查找字串	询问要找的字串和查找选择项，如果无选择项就把光标移到下一个查找字串出现的位置。
↑OA	查找并代换	询问查找字串、代换字串得用选择项，如果没有选择项，屏幕上显示位于光标现行位置以后出现的第一个匹配字串。然后询问REPLACE(Y/N)?（代换否），如果按Y，就执行代换。
↑L	重复查找和代换命令	重复上一次的（即最近的）查找或代换命令。
↑OV	光标回到起始点（见正文）	查找、代换或↑L（继续）命令执行完后，用该命令把光标恢复到执行命令前的位置，或者命令中的选择项是重复次数值或G，那么光标就移到了这一连串查找或代换操作之前所处的位置，在出现（NOT FOUND）"没找到"错误后，该命令尤为有用。 注意：如果其间已经使用了某些其他命令，执行↑OV就会有不同的含义。参阅第一节所述。

第八节　位　　标

位标是便于文件编辑的一个手段。使用位标记住文件的一个或多个位置，光标随后就可移到这些位置中的任一个。位标并不存在文件中，只在文件的同一次编辑中有效。

位标共有十个，为0～9。开始时，位标均未设置。如把光标移到未置的位标处，则产生错误信息。置位标时，先按下↑K键，接着按一个0～9中的数字，位标就置在现行光标处。然后，位标在置位处醒目地显示为〈n〉，n即键入的数字。能显示的〈n〉实际上并不在文件中，光标移动时会跳过它。

按下↑O和一个数字，就可将光标移到该位标处。在下章第二节中，介绍了长文件中光标移到很远处时的注意事项。

位标可以这样隐藏（即从屏幕上消失）：当光标在位标处时，按↑Q及该位标数字。隐藏后，该处位标仍被记住；当用↑Q指令访问时，它会重新显示。

以类似的方式，块起始标志〈B〉和结束标志〈K〉也可被设置、隐藏、访问（这时按B或K而不是数字）。在下一节，将对这些标志作进一步的讨论。

位　标　指　令

指　令	功　能	注　释（DESCRIPTION）
↑K0～↑K9	设置/隐藏位标	指定的标志（0～9）被置于光标处。如此处位标已置过且正在显示，则显示消失，否则显示为〈n〉。
↑Q0～↑Q9	光标移至位标	光标被移到指定标志（0～9）处，如位标隐藏则重新显示；如设置该位标，则出现错误信息。

第九节　块操作

块的移动、拷贝、删除指令，使Word Star功能更为强大和灵活。我们通过两个例子介绍这些块指令。

例1：假设你想把文本的某一块从一处移到另一处，可进行如下操作：将光标置于要移动的块的开始处，按↑KB，B在该处显示。然后将光标移到该块最后，键入↑KK，K在此处显示。这一块的内容将以白底黑字的形式显示。再把光标移到适当的地方，键入↑KY，则标好的段连同始末标志，都被移到该处，并显示于屏幕上。

例2：假设你想把一段中的一条语句移到另一位置，操作与上类似。移动语句后，通常需要↑B指令来对段落进行调整。

块拷贝指令与上类似，只是拷贝的块仍存在于原来的地方。块删除指令删去整个标志好的块。块写指令，将标记好的块写入另一文件。相关指令包括块隐藏／显示指令，该指令消除或重新显示屏幕上块标志或白底黑字显示的块。如果在块拷贝或块移动指令后，执行↑QV指令，光标就移回到块原来所处的地方。

编辑时，按↑K键，显示基本块指令的菜单。而求助键↑JV，可显示块移动操作的过程。

现在更详细地描述上面介绍的指令。

一、设块标志

始标和终标的设置顺序是任意的。设标过程中，可以插入任何其他的指令。在文件中别的地方使用↑KB和↑KK，建立新标志，以前所设的标志则失效。此外，还可以先标定块，然后在该块内编辑。

如果块中最后一行是一整行，建议把终标设在下一行的开始，这样，就可把最后一行的回车包括在标记块内。

始标和终标之间的内容，是整体的一个块，因此，不可能仅标记和移动表格中的某一竖列。

为了区别始标和终标指令，只须记住单词BLOCK，首字符为B，末字符为K。

前面已经说过，始标和终标之间的内容用白底黑字显示。终标在该块最后一行的回车之后，该行后的空格和标志符（如硬回车<）才会用白底黑字显示。

↑KH命令可以使标定的块从白底黑字转为正常的黑底白字状态，也可以使其又重新显示为白底黑字。但是在块处于黑底白字状态时，不能进行块操作，否则就出错。这有助于你防止大意的指令，如出现这样的操作错误，只要按↑KH并重复那条指令即可。

二、块移动

块移动指令是（↑KY），它把标记的文本块移到光标所在处，并将该块文本从原位置上删去。移动后，光标处于移动块的开头。

始标和终标随块移动，并继续显示。检查了移动结果后，我们建议按↑KH将标记块隐藏起来。这样，既从屏幕上消去了不顺眼的东西，又防止了偶然的↑KY指令。如以后仍想使用该标记块，再按↑KH即可。

块移动指令只移动标志的所有字符，并不自动进行重格式化。因此，移动后，常需重整文本格式（↑B）。

块移动后，↑QV指令可将光标移到块移出的地方。我们建议这里也要检查，如果此处也留下了太多的空格、回车，也需要重整格式。

注意：标记块中的0～9等位标并不随块移动，它们留在原处。

三、块拷贝

块拷贝指令（↑KC）与块移动指令类似。块拷贝并不从原位置删去标记块，只把标记块拷贝到光标所在处。光标留在拷贝文本新的开始处，块标记随文本移动。

标记块如需多次拷贝，可用连续相间的↑KC完成。在每一次拷贝后移动光标，再按↑KC就能在新的光标处进行一次新的拷贝。

四、块删除

使用块删除指令（↑KY），可以方便地删去文件中的无用部分。但这也很危险。我们强调每次块操作完成后就使用块隐藏（↑KH），其原因之一就是怕误用该删除指令。

五、块写入文件

块写入文件指令需要文件名，然后将标记块写入该文件中。该指令可把一段文本从一个文件移到另一个文件，也可在大文件内将块进行转移。这些内容将在第十节中作进一步讨论。

块写指令既不移动光标，也不改变现行编辑文件的文本。如给出的文件名已存在，WordStar便询问：

〈文件名〉已存在，——重写吗？（Y/N）

如按"Y"，则Word Star进行块写操作，并删除该文件以前的所有内容；如按"N"键，Word Star则重新请求文件名。

六、块长度限制

一次拷贝或移动的块的大小受到机器内存容量的限制，32K内存的计算机中，块最大约为500字符；64K的约为数千个字符。如果块长度超过限制，拷贝或移动时，会出现"块太长"错误信息。这时，可将标记块分成几个小块，分别进行操作。

七、长文件注意事项

块的移动、拷贝、删除或块写指令执行时，光标总先移到标记块处，然后回到原位置。如光标离标记块很远，则块指令的执行就非常慢。也可能由于工作文件所用的空间太大而产生"磁盘满"(DISK FULL)错误，见下章第二节。因此，块拷贝和移动指令时，如原位置和目的处相距太远，建议先将块写入文件，然后再从文件中读出。

块　指　令

指　令	功　能	说　明
↑KB	置标记块始标	设置块始标〈B〉到光标位置。如已在光标处置位且已显示，则该标志隐藏。
↑KK	置块终标	设置或隐藏终标〈K〉。
↑KV	块移动	将标记块移到位置。块标志随文本移动。
↑KC	块拷贝	将标记块拷贝到光标位置，原来文本并不改变，块标志随文本移动。
↑KY	块删除	删去标记块。
↑KW	将块写入附加文件	标记块写入该文件。文件中的编辑文本并不改变。见第十节。
↑KH	隐藏/重显示标记块	将标记块由白底黑字转换成黑底白字，或反过来，由黑底白字转换成白底黑字。
↑QB	光标移至块始	将光标移到块始处。如始标已隐藏，则显示。
↑QK	光标移至块结束处	与上类似。
↑QV	光标移至块原来位置	在块移动、拷贝、删除或块写指令各使用中，将光标移至块原来始标处。可用于检查块来源处的文本。

第十节　附加文件命令

附加文件(Additional File)指除正在编辑的文件以外的文件。有关附加文件的命令有合并文件，从编辑文件中抽出一段形成文件，或者用许多段装配成新文件，还有在一个文体当中进行剪贴等。

附加文件命令

命　令	功　能	说　明
↑KW	将块写入文件	请求文件名，将标记块写入该文件（见上节）。现行编辑的文件并不改变。
↑KR	读附带文件	请求文件名，然后将该文件全部内容插入到编辑文件

		的光标处。文件可以是块写指令建立的，也可以是正常编辑所建立的。
↑KJ	删除文件	请求文件名，然后删除该文件。由该文件占据的空间可以重新使用。该指令与系统状态ERA命令等效。
↑KL	改变现行联机驱动器	参见第三章第七节和第八章第十二节。
↑KF	目录显示开／关	参见第七章第三节和第八章第十二节。
↑KP	打印文件	参见第十一章。
↑KO	拷贝文件	将一个文件拷贝到另一个文件上。其使用与非文件菜单中的命令类似。见第五章第二节。
↑KE	文件更名	同非文件菜单中的E命令。见第五章第二节。

一、输入文件名

文件名在提问"NAME OF FILE? "之后输入。提问之上有菜单：

↑S = 删去字符　　↑Y = 删去输入项

↑F = 文件目录　　↑D = 重现字符

↑R = 重现输入项　　↑U = 取消

附加文件指令总需要一个文件名。为了不致和已有的文件名相重，可在出现"NAME OF FILE?"问句时，键入↑F显示文件目录，还可用↑Z和↑W将文件目录向上或向下卷动，以便看到所有文件名（用指令↑KF，可使目录在指令执行完仍显示）。

在"NAME OF FILE? "后，若想使用上一条附加文件命令使用的文件名，可键↑R，则上次使用的文件名会显示在屏幕上，按RETURN键即可使用。这在块写、读和删除指令均对同一文件进行操作时很方便，既减少了键盘次数，又省去了记住文件名的麻烦。

二、附加文件命令的用法

1.有些很常用的文稿段落，可以作为标准段块存成一些小文件。需要时，可以用↑KR命令很方便地将这些标准段块拷贝到正在编辑的文件中来。这也可用点命令.FI来实现。详见第十六章。

将标准段块拷贝到编辑文件后，可用替换命令来替换其中的某些项，如姓名、地址等。合并打印指令具有更灵活和更强大的插入功能。详见第十六章。

2.将某一段块从一个文件移到另一个文件：编辑第一个文件，标出要转移的段块(开始处用↑KB，结束处用↑KK)，使用块写↑KW命令将这个块存成文件，然后结束该文件的编辑（如没做删除和修改，使用↑KQ结束最快)；接着编辑第二个文件，将光标置于适当处，用块读命令↑KR，并给出前面↑KW时所使用的文件名，即可使刚刚存作文件的那段块插入第二个文件。

这里介绍的方法也适用将某一段块从同一个文本的一处移到另一处，对于长的文本和很长的移动距离，这样做可能较快。

在文件移动完毕以后，生成的临时文件应及时删去，以腾出磁盘空间。

第十一节　备考信息

备　考　信　息

命　令	功　能	说　明
↑JH	设置备考信息层 1	显示现行备考层，请求新的备考层 1。详见第五章第三节。
↑JD	打印命令	点命令和其他打印控制。
↑JI	命令索引，输入文本	
↑JM	页边，字间间隔，行调整，跳格停止位置	
↑JB	段落重整	
↑JP	置位标	
↑JV	移动文本	
↑JR	尺线	
↑JS	状态行	
↑JF	特征字符	
↑P	打印控制字符	
↑O	格式控制选择项的状态（例如卷字功能开/关等）	

第十二节　其他命令

其　他　命　令

命　令	功　能	说　明
↑QQ	重复下条命令	反复执行后面紧跟的那条命令，速度可由用户控制。数字 1～9 可用于控制执行速度，1 速度最快，9 速度最慢，补缺速度是 3。按任何其他键都终止重复。例如：↑QQC连续满屏幕地显示文本，直到结束。
↑U	中断	停止现行命令的执行。执行中的命令中断较慢。中断后，光标位置不定。如果键入了几条命令，而这几条命令没执行时键入 ↑U，则这些命令均被清除。出现“NAME OF FILE?”或“FIND? ”一类问句时，也可用↑U来中断。如果不对任何命令中断，键入↑U，则显示： ＊＊＊INTERRUPTED＊＊＊ 要求操作员按ESCAPE键。
↑PX	输入控制字符	X是除空格外的任意字符。该命令把该字符作为控制字输到文本中。详见第八章第三节。

↑JH	设置备考层	显示现行备考层，并请求新的备考层。详见第五章第三节和本章第六节。
↑KL	改变现行联机磁盘驱动器	显示现行联机驱动器名，并请求新的联机驱动器名（只按RETURN键，则不变）。其功能与非文件菜单L命令一致。详见第三章第五节和第五章。
↑KF	文件目录显示开/关	通常只显示部分文件目录，可用↑Z和↑W上下卷动以显示所有的文件名。如将命令压缩为↑F，则目录的显示只暂时保留。
↑KP	打印文件	请求文件名和选择项（选择项可用RETURN补缺），启动打印指定的文件。打印过程中，可以继续编辑其他文件（打印着的文件除外）。详见第十一章。
↑KP	停止打印	当一文件正在打印时，该指令暂停打印过程。
↑KP	继续打印	文件打印过程暂停后，用↑KP便可恢复。

第九章　附加编辑功能

第一节　问题的应答

Word Star的许多命令要向用户提出询问，一些询问要求以单字符应答，而另一些询问允许由回车符作为终止的输入行应答。每一类询问具有若干个公共字符，这些字符将在这里叙述。在所有场合，都可键入中断字↑U来中断询问。

单字符应答：要求以一个字符作为应答的问题由冒号(：)结尾。对非等待(RETURN)的问题可以键入下一个键作为应答。这种应答的例子即置备考命令问题。

是—否问题：某些问题要求一个是或否作为应答；这些问题有形如"…（Y/N）："的提示。这些问题接收Y、y或↑Y作为"是"，而任何其他的键都作为"否"。也可任意地用RETURN、空格或任何手动键作为"否"的应答。

行输入问题：这些问题结尾以"?"提示。它们包括全部文件名的问题(如D、P和↑KR命令等)、页号和停列标记问题以及若干其他问题。当回答一个行输入问题时，错误的键盘输入能够被校正，这个"PREVIOUS RESPONSE TO THE SAME QUESTION"，或它的一部分，若不再记入时，它们能够被恢复。文件目录能被显示并可上下任意滚动。

对行输入问题可用RETURN、LINE FEED或ESCAPE键加以终止。ESCAPE键在打印（P或↑KP）、查找（↑QF）和替换（↑QA)中具有特殊含义：对页边和跳所格停止问题（↑OL、↑OR、↑OI和↑ON），ESCAPE意味着使用光标列（如状态行示），而RETURN意味着跳过该命令。在其他情况下，ESCAPE等价于RETURN，LINE FEED等价于RETURN。

当回答一个行输入问题时，可选用表9-1所示的特殊字符以便校正键入的错误（在按下返回键之前）、显示目录和其他一些功能。注意：↑X意味着控制X，即按下CTRL键，同时键入字母X。

在第2、3级备考状态下，许多公用特殊字符后的注释出现在绝大部分行输入问题上，对在每一备考状态和以"?"作为结尾的所有问题中，这些特殊字符都是有效的：

↑S = delete character　↑Y = delete entry　↑F = file directory

↑D = restore character　↑R = restore entry　↑U = cancel

表9-1　回答行输入问题时的特殊字符

字符	功能
↑X or ↑Y	擦掉全部回答。
↑S、↑H BACKSPACE or DELETE	擦除一个字符。
↑D	光标右移：恢复一个字符；从前面被取消的回答中或从上一次具体被询问所给出的回答中恢复一个字符。
↑R	恢复全部被擦掉的回答或之前的应答。为了使用如同上一次的相同应答(例如，

你想打印同一文件的第二个副本），可直接输入↑R和返回。

↑E　在命令执行期间显示存入磁盘驱动器中的文件目录，如果文件目录已在屏幕上，则无效。

↑Z、↑W　如部分文件目录被显示，把文件目录向上（↑Z）或向下（↑W）滚动来使下余的文件名进入屏幕。

↑P　逐次使用下一个键打作为回答部分，不论它是通常具有特殊功能的控制字符或是终止字符之一，即使软连字符输入（↑OE）有效，也可以用↑P－来输入一个硬连字符（规定为－）。

－　如果软连字符已被打开（↑OE），则输入一个软连字符。允许在查找和替换命令中使用软连字符。利用↑P来强制输入一个硬连字符（即－）。

↑N、↑S　它们仅在查找命令（↑QF）和置换命令（↑QA）中引出的“FIND？”的问
↑A、↑O　题中才有特征含义。

↑U　在中断和终止命令的执行中，清除前面键入的任何命令或文本。当在回答一个问题时，和大多数其他上下文中，↑U也具有这种作用。

第二节　长文本处理

Word Star能编辑存满一张软盘那样的大文件，它对中等大小文件的编辑则更快，更容易和更灵活。我们建议：最好把大文件分解成多个子文件。但下列情况不能分解：当一个分页是文件之间的一个强制性的结果时，或当一个自动的页编号仅在文件内部工作时（即使可以用一个“点命令”来指定每一文件的开始页数），要打印多个文件就必须给出多个打印命令。

文件大小的确定：通过非文件清单中的R命令，或CP/M操作系统的STAT命令，能确定以字节（字符）为单位的一个文件大小。当编辑时，把光标移到文件的结尾（↑QC命令），同时输入↑OP来关闭页显示。这样，就可以检查文件的大小。当页显示关闭时，文件中的字节（字符）数以FC＝nnnnn形式显示在状态中。

文件多大才算大文件：这问题依赖于所使用系统的内存容量的大小而定。当一个文件能全部置入内存时，将光标从文件的结尾移到文件的开始（↑QR命令），若用几秒钟，则这个文件仍然算小文件；若用几分钟，则这个文件就算是大的了。遇到这种情况，希望仔细阅读本章的其余部分，或把文件分成二部分（见第八章第十节的↑KW部分）。

对大文件的软盘空间的考虑：当编辑一个文件时，Word Star总是在磁盘上产生一个新的文本，当文件中的字符数量比用来存放它的RAM的容量更大时，一旦光标回移到超出RAM容量之外的字，即会产生切换。Word Star对由此而产生的临时文件的编辑，还要利用附加空间，所以，这将导致下面二个问题。第一个问题是：光标在倒转（指自下而上移动）期间所用的临时文件，使磁盘容量不够用，这时若回移光标，则可能出现“disk full”（磁盘满）错误（见第十二章），但它仍将可能把光标自上而下移动并保存；在编辑时，只将光标下移，可以把这个问题所造成的影响减到最小。第二个问题是：有效磁盘存储空间容纳不下新的文本，应注意，绝不能让这个问题出现！这需要考察你的文件大小和软盘有效存储空间的大小（用STAT命令），如必要时，可把文件移到不同的软盘上，以确保有足够的有效存储空间。

最终，一个文件可能做得很大，以致于不可能把二个文本装在一张软盘上。这时，文件

就应每次编辑到不同的软盘上。在D或N命令中或调用 Word Star 的系统终端命令行中，可在文件名之后输入一个空格及第二个磁盘驱动器名加以实现（如第五章所述）。在这种方式下，每一个"SAVE"将在驱动器之间交替执行，把长文件分解为多个文件。文件大小的绝对限制是软盘容量。

长文件的编辑：编辑长文件时，要考虑二点：第一，要使临时工作文件的磁盘空间减到最少和避免 "disk full" 错误。第二，光标自下而上长距离移动，是很慢的。为避免自下而上长距离移动，当文件产生切换时，对超出RAM容量的字符继续使光标自上而下通过全部文件，使得所有超出 RAM 容量的字符按预计的顺序排列好；再把光标从靠近文件尾部的地方返回到文件的开头，用 "SAVE"（↑KS）而不用↑QR来保存文件。这样，不仅使文件的编辑加快，而且也避免了所有超出 RAM容量的字符的存储可能出现的故障。

类似地，当需要全局性"替换"时，首先要保存，以便从文件的开头处开始进行替换，如源点与目标点相距较远，要避免用块移动↑KW，代之以移动光标（若光标的目的位置在原位置之前，应先保存一下），然后进行附加文件读↑KR。

第三节　关于更换磁盘的附加说明

本节重述第三章第十节中关于更换软盘所给的规定以及为安全更换软盘附加的一些特殊条件：

1.在系统提示符 A>状态下，可安全地更换软盘，换后应立即键入 ↑C。

2.当 Word Star 处于非文件清单（不进行编辑）状态，且对磁盘文件不打印时，亦可更换软盘。

3.驱动器中如无软盘时，亦可在驱动器中插入一个软盘。

4.磁盘处于只读状态，可用↑KR（附加文件读）或打印（第十一章）命令来更换软盘。但要确定，在此之前是否已使用了可对新插入磁盘执行的写操作命令。

第四节　文本和文件的兼容性

Word Star 的文本文件（用 D 命令来编辑的文件，或用于处理"卷字"的文件或用 ↑B 命令来处理的文件）可能与一些别的程序或语言不兼容，但这对字处理应用没有影响，如想以 Word Star 作为通用编辑程序，用来编制其他用途的文件，如数据文件、源程序文件等，请细读此节。

Word Star 的文本文件和一个完全标准的 CP/M 文件之间的差别是：在 Word Star 文件中，字节的高位被置成确定值，而在某些情况下，标准的CP/M文件，这位总置"0"。

为了产生一个与其他程序兼容的文件，用↑N命令来调入 Word Star 进行编辑，同时不用卷字或↑B命令。用手工输入所需的所有回车符和所有行的格式。

另一方面，用 Word Star 时，通过 PIP 中使用〔Z〕来复制文件或通过用 CP/M 的编辑程序来编辑文件，则可屏蔽掉高阶位。当然，这将对下一段文件的编辑产生影响。如需要，可用Word Star 对新近的文件进行重编来保存一份未被屏蔽的文本。

如以后想用 Word Star 文件，那么Word Star 不应由其他编辑程序编辑。否则会引起

许多错误。

第五节　非文本编辑命令

对字处理，用户不需了解它。

“非文件”编辑命令(在非文件菜单上的N)有双重用途：为文件的相容性来禁止动态加页码和为方便提供许多不同的补缺标志。偶尔可注意到：用非文件清单上的 D 命令或用形如«WS file name”的系统命令开始标准文件编辑和用N命令开始“非文件”编辑之间的差别。下面是完整的清单。

1）动态加页码，这使高阶位的利用被阻止。分页（↑OP）无效。

2）状态行表示文件字符和文件行而不是页面和行或页面（第七章第一节）。

3）在编辑期间，“点命令”无效。

a.非法点命令没有“？”标志。

b.不出现“POT AT FILE……LAY”信息。

4）卷字、调整、有效的跳格和尺线的补缺显示为 OFF，而不是 ON。如需要，这些功能用第八章第六节的常用命令实现。

注意：卷字或分段重调整〔↑B〕的作用是把“高阶位”放入文件的特征位，这可能导致与某些外部程序不兼容。

第六节　固定跳格方式

为标准字处理用，让变化跳格方式有效，如用包含跳格字符的现存文件工作，或希望用固定跳格，请看本节。

当可变跳格（↑I，09H）无效时，利用文件中的跳格字符可显示 8 个空格，这与通过变化跳格方式而输入到文件中的多个空格是不同的。这类跳格与通过CP/M编辑程序和 Micropro Word Master 所用的跳格是兼容的。这样的跳格常用在 CP/M 的开发程序中。

因为跳格符是一单字符，所以固定跳格符编辑不同于变化跳格符编辑。在固定跳格符编辑时，光标不能被放在屏幕上代表跳格符的空格内，但是，当光标跨越跳格符时，跳格符“pops”（退栈），即一个跳格符退栈，好似删除一个单字符。一个单字符将重复打入一个跳格符。当在一个跳格符前插入时，跳格符后的文本仍保持在屏幕的原位置上，直到插入足够多的文本，才把跳格符“push”（下推）到下一个跳格停位置。在此情况下，跳格符向右退栈到以后 8 的倍数列上。

跳格方式与跳格符的输入有关。当变化跳格方式打开时，已在文件中的跳格符将有相同的显示（看来象空格）并以标准固定跳格方式（不完全象前段空格）编辑。当变化跳格方式打开，用跳格键把光标移在现存文本之上（插入关），而且文本包含跳格符时，光标可达到下一个跳格停的位置而停止。

为编辑一个已包含固定跳格符的文件，不需要设置固定跳格符，除非你需用↑I（跳格）键，并把固定跳格符输入到文件中。注意：PIP 版本为一些系统提供了一个把跳格符扩展到具有每n列跳格停的多个空格的选择项，可用它把一个文件从固定跳格变换到变化跳格。

第十章　打印指令

两种打印指令："dot commands"（点命令）和"print controls"（打印控制）。

第一节　打印控制字符

"print controls"是将单控制字符输入到文件中（编辑时），在输出打印期间产生下划线、粗体字和下标功能。在编辑期间，打印控制为键入↑P后跟一个所需的字母，该字母可以是小写或大写，或借助CTRL键输入，可在打印文件的中间段中任意增加下划线和粗体字；这些控制字符，如没有特殊考虑，一般用来中止"卷字"和"段变换"操作。

通常，含有打印控制字符的行，在屏幕上显示的长度要比打印出来的更长，这是因为Word Star在行里放入一些用于控制打印而又不被打印输出的附加字符。表10-1叙述了所有打印控制字符；↑P菜单给出简单的说明。

表10-1　打印控制字符

字符	功　能
↑S	下划线。它放在要被划线的短语前后，仅适用于非空字符。
↑B	粗体字。借助菊花轮和其他打印机的错位重选强力重复打印。在"Teletypa_like"（类似电传）打印机上重复打印每个字符。
↑D	双打控制。每个字符连续打印两次，这比用粗体字控制打出的字迹更清楚。
↑X	擦除显示控制。在打印字符上面，打印连接号（即----）来指出一个被修改文件中被删除的部分。
↑V	下标控制。它以下标形式来打印被封闭的字符。字符下移的补缺形式为3/48英寸，还可用·SR点命令来改变（见第二节）。若打印机没有下移功能，那么下标打印在下一空行上，否则打印在同一行上。
↑T	上标控制。以上标形式打印被封闭的字符。
↑Y	色带颜色控制。对菊花轮打印机和其他打印机，它对色带颜色进行选择，以特定的颜色打印被封闭的文本。
↑C	中止打印。它中止打印直到操作员重新启动。它可用来允许对打印轮和色带进行更换而要求打印机中止打印。如需要，可在中间行和/或在同一行中使用。当暂停发生时，状态行将显示"print paused"。可用P命令（非文件清单）或↑KP命令（编辑时）重新启动打印。 注意：对一般的双向打印机，Word Star总是从左到右打印含有↑C的一行。
↑A	选择变化的字符间隙。对菊花轮打印机，每英寸可达12个字符，可用·CW点命令来改变（见第三节）。通常在非菊花轮机上不起作用。
↑N	返回标准的字符间隙。对菊花轮打印机，每英寸选择10个字符，或用·CW命令来改变。

↑K	左右头标/脚标控制。该字符用于头标和脚标点命令产生头标、页号等等，分别打印在偶页面的左边和奇页面的右边。当在正文主题中使用·HE或·FO点命令时，如页号是偶数，它的作用是抑制打印后面的空格，直到下一个非空为止。
↑F	"虚拟空格"。借助菊花轮打印机把代码"20H"打印为字符。这个代码打印为一个"百分号"；对不同类型的打印机，则打印不同的字图。
↑G	"虚拟擦除"。借助菊花轮打印机把代码"7FH"打印为一个"非符号"或"双下划线"，也可由"96－"字符的菊花轮打印机打印其他图形。
↑O	非断裂空格。打印空格，但在格式化期间，行中止或行调整时不作空格处理。它用在需要单空格显示的地方。在调整或连到下一行时，该位置的空格数不允许再扩大。
↑H	退格。使下一字符添印到本行中被处理的字符上。通过打印多个字符在字母上方置重音符产生特殊符号是有用的。它可用在"格式"文本中——"字包"和"段的重格式"；为退格留下了适当的余量。在调整文本中，定位不是十分准确的，在非退格打印机上（类似电传），退格通过相同的列最多不能超过8次。
↑Q	用户打印机功能 1
↑W	用户打印机功能 2
↑E	用户打印机功能 3
↑R	用户打印机功能 4

用户打印机功能控制字符是用来实现进入Word Star不另外支持的特殊打印机功能，如改变铅字类型或赋能走纸装置。

其他控制字符：在上面表格中没有定义的控制字符将以一个↑和一个字母的形式打印，正如它们所显示的那样，只有以下例外：

Form Feed（↑L）产生一个页面中止。

"Tab"（↑I）以8的倍数，成倍地前移为打印和显示提供足够多的空格。除在固定跳格方式中，Word Star通常不把这个字符输入到文件中。

↑M类似回车，若没有换行而输入一个回车，通过按↑PM或↑P返回将产生一个重复打印行。

↑J同"Line Feed"字符一样。这个字符在应用中通常不明显地使用。

第二节　点命令

点命令输入到文件中的特殊行，用来设置页面的长度或规定一页的头标。所有的点命令对正常使用时都有相应的补缺值。点命令本身不被打印，它控制打印操作。动态页面中止显示也对某些点命令进行应答。

一般来说，一个点命令由第一列中的句号、两字母代码和（对某些命令）文本的行号或其他变元组成的。

在编辑期间，以相同方式把点命令输入到文件中，并在编辑时显示在屏幕上。当把一个

句号输入到第一列时（或已在一列中），为方便起见，自动缩进到左边界被禁止。

由打印功能，将解释放在文本中任何地方的所有有效点命令行；可变的顶限和底限以及行的高度都可任意改变，直到打印输出被影响为止。但是，当点命令都被放在文件的开始，即所有正文之前时，动态页面中止显示将仅对·LH、·PL、·MT和·MR命令进行应答。

表10-2将叙述点命令。表10-3总结点命令以及列出了它们的补缺值。为了在编辑时能即时参考，↑JD命令显示出点命令的简要说明。

表 10-2　　点 命 令

命令		说　　明
纵页的布局		
·LHn	行的高度	在菊花轮打印机上设置行高度为1/48（英寸），·LH8每英寸产生6行（补缺值）；·LH6每英寸产生8行。·LH12在一个打印机上给予相同的间隔是1—1/2行间距等等，表10-4表示了·LH命令用于改变每寸行数。当文本行格式化时，这个·LH命令对由↑OS命令控制的单、双、三间隔提供变换或补充。当在一个无增空能力的打印机上打印时，如用·LH命令，则页面中止仍象命令生效一样被确定。注意：为建立新的行高度，必须对其后若干行进行解释。原先为单位设置的数值（顶限、页面长度等等）仍保持不变。
·PLn	纸长度	n是整个一页的行数（上一个·LH所设置的行数；如不给·LH的话，按每英寸6行），页包括顶限和底限。纸的高度必须和使用的形式相匹配。补缺值是66个，补缺行高为11英寸。
·MTn	顶限	从页顶到该页正文开始之间的行数。注意：如使用头标就打印在头限所表示的范围内。
·MBn	底限	在页的底部不为正文所使用的行数。页号或脚标打印在底部范围之内。注意：页面中正文所能使用的最大行数是纸的长度减去顶限和底限。上述命令（·LH、·PL、·MT和·MB）在编辑期间，如它们出现在文件的开始处，则它们仅仅根据页面中止显示被正确解释。但是，它们可以用在文件中任何地方来改变所需要的竖直页面格式，直到影响到打印输出为止。
·HMn	头标边界	设置页面头标（如用，见·HE命令）和正文之间的空行数。注意：头标和头标边界必须位于顶限空间内。如n值大于或等于顶限值，则取顶限值减1，当用顶限值减1，则补缺值为2行。
·FMn	脚标边界	确定页面中正文可使用的底端和页号或脚标（见·FO）之间的行数。注意：脚标边界和页号或脚标出现在底限空间内。补缺值为2行。

表 10-2（续）

水平页的布局		

大部分水平格式，象页边和文本的对齐是在编辑期间完成的，其中不包含点命令。下面

提供的点命令是利用打印功能对水平格式进行控制的。

·PCn　页号列　当不用脚标命令（见后面·FO），且·OP（在下面）无影响时，页号所在的列，可设在（可通过在·HE和·FO命令中的“*”实现。说明在下面）页底的左边、右边或中心等位置。当给出·PC命令时，就象由↑A、↑N和/或·CW命令所确定那样，列宽受到字符宽度的影响。补缺值是编辑功能右边补缺值的1/2，从而在补缺宽度下的正文中页号处在中心位置。

·POn　页位移　整个文件从打印机的左边右移。其中除所有缩进文件中之外的列（目前宽度）数，允许从左边纸的牵引输送孔偏移文本，并允许把窄纸放在打印机支架的中心位置附近。补缺值为8列。

标记页数

·PA　页　无条件开始新的一页。

·CPn　条件换页　如在目前页面上剩下不到n个空行（目前行高度），则开始新的一页。应用此命令使文件整块地保持在一起，防止在一个标题后或在一个表格的中间就分页。

页顶标、页下标和页号

·HE text　定页顶标　当正文中一旦出现·HE，则在下一次出现·HE之前的诸页均依此作为页顶标。页顶标可任意地频繁改换。最初的（或补缺的）页顶标是空白，对非文本标题可用·HE来改回为空白。在开始页上打印标题时，一个·HE命令须在所有文件的文本之前。当在屏幕上构成页顶标（或页底标）时，考虑到不打印字符“·HE”，这样，顶标将出现在屏幕上“·HE”出现位置左边第4列中。仅在页顶标命令（·HE）和页底标命令（·FO）中解释下列特殊字符：

*作为目前页号打印，它可被安置在页的顶部或底部所需要的任何地方。

＼若没有特殊解释，它打印下一个字符——用＼*在标题或结束时来打印一个*，用＼＼来打印一个＼。↑K，如页号是偶数，则忽略下面空格，直到下一个非空格字符为止；对出现在页面距连接部分最远的角落上，可以用它来产生页号或其他文本的页顶标或页底标。注意：键入↑P可把↑K输入到文件中。K用于所有打印控制。

·FO text　页底标　在一行中，·FO之后部分将被用作当前页和以下诸页面的页底标。·FO可以根据所需任意给出；当每页的底部发生冲突时，就用最新给定的页底标。

在页底标文本中，字符*、＼和↑K的解释如同上面刚讲过的·HE命令中所描述的一样。当没有·FO命令或在·FO命令后面是非文本时，Word Star将在用·PC命令所指定列的页底标行中打印页号。当被用户指定的页底标有效时，如你希望编号的页在页头标或页底标所需位置上包含一个*，那么页号就不会自动被打印。

·OP　省略页号　如页底标没有给出，它就阻止页底标行中页号的打印。如页底标指定，

·OP 也不影响在页头标或页底标出现会打印出 * 号页编号。补缺状态有效。

·PN　页编号　在·OP 命令后若没有设置页号，不带有数字 n 的·PN 命令则进行页编号。如先用了·FO 命令，·PN 将无效。

·PNn　页号　带有数字 n 的·PN 命令打 * 页编号（如已用·OP 命令，则页编号撤消）并在当前页的底部和/或在遇到 * 的地方设置页号，页号连同页头标或页底标被打印。起始值的补缺值是 1。

其他的点命令

·CWn　字符宽度　该命令在菊花轮打印机上以 1/120 英寸来设置字符的宽度，这宽度既适合标准字符间距，也适合变化字符间距（分别由↑N 或↑A 规定了的）。但在非菊花轮打印机上无效。允许对特殊字符调节间距——如为突出标题使字符的间隔比正常的大得多。标准间距补缺值为每英寸 10 个间距（等价于·CW12），可变间距补缺值为每间距 1/12 英寸宽（CW10），表 10-5 表明，用·CW 命令是为了各种各样的间距（即每英寸间距数）。注意：改变字符宽度或间距（用·CW、↑A 或↑N）并不影响以前确定的页位移（·PO）和页号应处在列（·PC）的值。也不影响以前确定的页头标（·HE）或页尾标（·FO）的文本。当碰到·PO、·PC、·HE 或·FO 时，将按实际上字符和间距宽度来解释。

·SRn　下标/上标滚动　在菊花轮打印机上打印上标或下标之前，打印纸以 1/48 英寸滚动。补缺值为 3/48 英寸。

·UJ off　微调整对齐撤消（或·UJ0）。

·UJ on　微调整对齐有效（或·UJ1）。

当行微对齐是撤消时，所有空格（包括“软”空格）出现在文件中时，它们都被均匀地打印。当把空格加到屏幕上行对齐时，它们按出现在屏幕上的位置被打印。而不是被均匀地分布在菊花轮打印机上以 1/120 英寸的间隔来打印。微对齐通常是处于有效状态的，将微对齐转为撤消状态有时也很有用处。例如，为了对一表格的格式进行打印，通过变换（↑B）或字包，把表中的列在屏幕上排成行，即使“软”空格也允许排进表行中。

·BP off　双向打印撤消（或·BP0）。

·BP on　双向打印有效（或·BP1）。

菊花轮打印机的双向打印（左右交替地打印各行）通常是有效的，当打印的字符有一点不整齐或纸的传送有一点草率时，把双向打印撤消可以提高输出质量。

·IG text　忽略
..text　行的剩余部分中任何一个注释都不打印。由于打印功能，可把任何一个未定义的点命令也看作是一个注释。但是，编辑功能对未定义的点命令在屏幕最右列上显示一个“?”提示符号。

表 10-3　点命令和补缺值汇总

命令	功能	单位	补缺值
.LH	行的高度	1/48英寸	8=6行（英寸）
.PL	纸的长度	行	66个补缺行=11英寸
.MT	顶部页边	行	3个补缺行=1/2英寸
.MB	底部页边	行	8个补缺行=$1^1/_3$英寸
.HM	页头标	行	2个补缺行=1/3英寸
.FM	页底标	行	2个补缺行=1/3英寸
	（页*页边）		
.PC	页*所在列定位	列	从右页边数1/2
.PO	页横移	列	8个补缺列=4/5英寸
.PA	新页		
.CP	条件换页	行	
.HE	页头标（信息的）		空白
.FO	页底标（信息的）		在.PC列的页号
.OP	省略页号		
.PN	页号		1
.CW	字符宽度	1/120英寸	12个标准间距/英寸
			10个变化间距/英寸
.SR	下标的滚动	1/48英寸	3
.UJ	微对齐调整	撤消（0）有效(1)	有效（1）
.BP	双向打印	撤消（0）有效(1)	有效（1）
.IG	注释		
..	注释		

表 10-4　字符间隙

间隙（每英寸字符数）	点命令
5	.CW24
6	.CW20
7	.CW17
8	.CW15
10（补缺值）	.CW12
12	.CW10
15	.CW 8
20	.CW 6
24	.CW 5
30	.CW 4

表 10-5　行的高度

每英寸行数	点命令
2.0	.LH24
2.4	.LH20
2.6	.LH18
3.0	.LH16
4.0	.LH12
4.8	.LH10
5.3	.LH 9
6.0	.LH 8（补缺值）
6.8	.LH 7
8.0	.LH 6
9.6	.LH 5

第三节　点命令的使用

页面头标（Page heading）：.HE 命令可在每一页的顶部确定一文字行。例如：

```
.HE Part II
.HE 10.3                                    Using Dot Commands
```

在屏幕上构成该命令时，应记住：".HE␣" 所占的开头四列不被打印，这样，在打印时，".HE␣" 后面紧跟的部分全部被打印，并且向右移动四格，得到一个经过调整的头标。如下所示：

```
.HE                                                    Section 5
```

若希望在页面顶部打出页号，则在.HE命令字后面相应地方给一个 "＊"。例如，要在页面顶部"Page" 后面打出页号，则用命令：

```
.HE Page＊
```

当头标中含有一个"＊" 时，页号所占的列数取决于页号的数字位数。若只想在顶部打印页号，则用.OP 命令禁止 Word Star 再次在底部打印。

头标可根据需要加以改变。给一个空头标就可清除头标：

```
.HE
```

.HE 命令的结果会影响以后各页面。如在当前页面被打印之前，该命令已有效，在该页也将有效。

头标与正文之间的间隔可用.HM（Heading Margin）命令改变。该命令只能移头标，而不能修改页长或页面上正文的位置，且头标不能超过页面顶界。

当页面顶界（.MT）置为 0 时，头标无法打出。

头标（或下标）中可以使用由打印控制字符产生的下划线、粗体字等特殊字符。.HE（或.FO）命令中的打印控制不会对正文产生影响，正文内部也不可能产生头标（或下标）。如要改变字符的宽度（下面将叙述），则在给出.HE（或.FO）命令以后，头标（或下标）

以宽字符打印。如果在头标（或下标）中使用↑N或↑A，则在打印的任何时刻打出相应的字符间隔。

页面下标：.FO 命令在每一页的底部确定需要打印的文字行。用法类似于.HE 命令。每页中打出的下标都是最新确定的。

若含有“*”，则在“*”处打出页号。

.FM 命令用以确定正文区与下标之间的间隔，不得超出底界范围，其变化仅为下标的上下移动，不改变正文区的大小。当正文底限（.MB）置为0时，下标无法打出。

页号是下标的补缺形式：若无有效的下标（未给出.FO 命令或在最近出现的.FO 命令中不包含任何信息），Word Star 将在下标行中打出页号，.OP 命令可以使之消除。页号可由.PC 命令置于中间规定的列上，其补缺形式是第33列（若 Edit 规定只用64列，则为第30列）。

页面编号：打某一文件时，Word Star 以补缺形式从1开始进行编页。也可用.PN 命令改变。如某一文件是文件的第二章，且第一章共有33页，则在打印该文件时先设置：

.PN 34

打出该文件时，将从第34页开始编号。注意：.PN 仅在打印输出时有效，编辑时，状态行上的页号显示总是从1开始，且逐页加1。

Word Star 所允许出现的最大页号为65533（对有些点命令，最大为255）。

页号的打印可由.OP 命令抑制，又可用.PN 命令使之继续（除要重新置页号外，.PN 后面无需任何数字）。

.PCn 命令可以改变页号位置。例如：欲置页号于行宽为80列的中间，可这样实现：

.PC 40

这与下面的下标命令等效：

.FO

置左－右页号：为便于装订，.HE 或.FO 命令行中，通过特定的字符↑K，Word Star 在右端打出奇数页号，而偶数页号在左端打印。对于偶数页号，↑K 会禁止其后空格的输出。例如：

.FO ↑K 10- *

对应于偶数页面，等效于：

.FO 10- *

而对于奇数页面，则等效于：

.FO 10- *

分页控制：页面中止控制有两个目的：使下文从下页页顶开始，以防在一新页中出现不适当的空格。这可使用户有目的地在段与段之间空一、二行，或避免某一段开始于页面最底部。

对于第一种情况，可使用页面命令：

.PA

第二种情况常用于当前页面剩余行数小于一定数值时，进行页面中止。用.CP 命令测试当前页面上的剩余行数，若很少，则使页面中止。例如，若有一个10行的表格待打印，在其上面置：

.CP 10

若当前页面剩余行数大于10，该表格就在当前页面打出，否则在下一页顶部打出。

以.CP代替.PA可以减少页面的检查。修改文本时，必须退出点命令。例如，要在表格的中间分页，只需在它上面置上.PA命令即可。以后应找到.PA，并将其擦掉，.PA在任何地方都会进行分页。另一方面，若.CP10命令字保留在剩余行数大于10的页面中，则不会产生任何影响。

此外，还习惯于在每个标题下面置上.CP5，当页面中剩余行数小于5时，会产生一新页面。

纸的长度：通常，Word Star规定每页纸长度为66行，即11英寸，每英寸6行。也可用.PL命令在每个COMPLETE PIECE OF PAPER上改变纸长。请注意它与正文区长度的区别。

如使用的非菊花轮打印机每英寸打印的行数不为6，也可使用.PL命令。例如，打印机每英寸打印的行数为8，仍希望纸长为11英寸，则应在文件开始时置.PL88。

页顶限和页底限：Word Star允许用命令.MT在每页的顶上设置一定数量的空行，用命令.MB在页面底部置一空区域。当.PA或.CP命令产生非页中断时，每页打印的正文行数与页面长度（.PL）相等，而小于页顶限与页底限之间的行数。若想改变页面中正文区的大小，只需相应地增减页顶限和页底限的大小。

页面正文定位：正文可通过改变页顶限（.MT）或页底限（.MB）在页面中上下移动，或直接在打印机上移动打印纸。当打印头在页顶下二、三行处开始打印时，其补缺形式约为××××××的中间位置。

通过改变页面偏置，能使正文在水平方向移动。通过补缺，Word Star允许在每行的开始留8个空格，否则要用命令改变。例如：

.PO 0

动态页面中断显示的相互影响：文件开始时使用.PL、.MT、.MB和.LH命令，在编辑期间，动态页面中断显示将准确地出现在屏幕上。

改变行的高度：在菊花轮打印机上，.LH将行的高度规定为每英寸行数的1/48。例如，用每英寸8行取代通常的6行，应使用命令：

.LH 6

表10-5列出了各种不同的行高。

使用.LH命令时，须注意.LH和另一些纵向格式命令的先后顺序。例如，若.LH6后面接着是.MT4，则产生1/2英寸的页顶限。反过来，.LH6跟在.MT4之后，则产生2/3英寸的页顶限。

改变字符间隙有两种实现的途径：使用可被嵌入中线且允许在两字符宽度内改变的打印控制字符↑A和↑N；（在菊花轮打印机上）可使用允许设置任意字符宽度，但仅对行之间而言的.CW命令。

确定了页面偏置（.PC）、页号所在列（.PC）、头标（.HE）或下标（.FO），其字符间隙就是现行值，以后变化时也就无需再改变以上四种参数的字符空间。

非菊花轮打印机上是通过可变间隙（↑A）和标准间隙（↑N）等打印控制字符控制字符间隙的。菊花轮打印机上改变字符间隙是由↑A、↑N和.CW命令完成的。

↑A　字符间隙可变，补缺值为 12 字符/英寸，可用.CW 命令加以修改。

↑N　标准的字符间隙，补缺值为 10 字符/英寸，也可用.CW 命令修改。

.CWn　置字符宽度为 n 个 1/120 英寸的标准形式或可变的间隙。表 10-4 给出了.CW 命令得到的各种值。

示例：

↑N The Word"↑A crammed" ↑N is printed in alternate pitch

若.CW 命令未作用，字"crammed" 的间隙为 12，其余均为 10。

↑A (CR)

.CW20 (CR)

↑N (CR)

.CW8 (CR)

narrow↑C↑A wide ↑C↑Nnarrow

"narrow"将以 8/120 英寸的字符宽度打印(15 字符/英寸)，"wide"以 20/120 英寸的宽度打印(6 字符/英寸)，下一个"narrow"又以 8/120 的宽度打印。↑C 是控制打印机暂停的，可让操作员更换不同的字轮。↑A、↑N 后面加一个回车换行会在正文上面打印两个空行。

第十一章　打印功能

打印功能可打印一编辑文件或磁盘文件。不加特定的抑制，页顶限、页底限、头标和下标将随正文一起打印。

Word Star 可配置菊花轮的、针式的或其他具有增定能力的打印机，它还能对打印文本进行微空间调整，取消在屏幕上用于调整行格式的“软”空间，且在字与字之间（也可能是字符之间）增加 1/120 英寸的空隙，不产生字与字之间的大空隙，而将该行排满。各种字符的宽度也因字而异。

微空间调整仅用在字包或重新格式的段落的“被格式”行的左右限之间，因此，界线及以外的正文不改变。字包或重新格式的段落的非“被格式”的行不需进行微空间调整，这样表中的列准线则被明确地显示（为在打印时进行微空间调整，则输入段落的最后一行必须通过字段重格式）。

Word Star 仅能打印存在磁盘上的文件。

打印功能可以在编辑的同时进行，而不需返回主菜单。利用非文件菜单中的 P 命令或编辑过程中的 ↑KP 命令控制打印、暂停、继续。

在打印功能作用期间，除内存容量极小的计算机外，都可开始或继续编辑，且其他大多数命令可以执行。打印时，对键盘的响应稍有延迟。若不以高速输入正文，则可同时进行编辑和打印，用以对文本进行检查和少量的修改，但不能输入正文，否则要暂时中止打印。

非法的打印控制字符及点命令将不予执行。在磁盘上生成一输出文件时，将解释所包含的点命令，保留大多数打印控制字符。未使用打印控制字符的文件也可退出 Word Star 进行打印。

第一节　启动打印

在非打印继续或打印中止状态给一打印命令（非文件菜单中的 P 或 M，或编辑过程中的 ↑KP），就意味着“启动打印”。Word Star 将提出：

NAME OF FILE TO PRINT ? ■

打印机准备就绪后，输入文件名和一个 ESCAPE 键，所有的操作都以其补缺方式开始打印。若输入文件名后，打入 RETURN 键，在开始打印之前，Word Star 会提出一些有关打印操作的问题。如果文件未找到，则显示相应信息并重新提问。

当文件名是以 RETURN 结束的，在开始打印之前，Word Star 将提出下列问题：

DISK FILE OUTPUT (Y/N) :
START AT PAGE NUMBER (RETURN for beginning) ?
STOP AFTER PAGE NUMBER (RETURN for end) ?
USE FORM FEEDS (Y/N) :
SUPPRESS PAGE FORMATTING (Y/N) :
PAUSE FOR PAPER CHANGE BETWEEN PAGES (Y/N) :

Ready printer, press RETURN：

所有 Y/N 问题都可用一个字符来回答：Y、y 或↑y 表示“yes”，其他字符均表示“No”。RETURN 或 ESCAPE 可用作“No”或其他补缺值。因此，文件名后打一个 ESCAPE 和连续七个 RETURN 效果一样。可用↑U 取消打印命令。

下面对七个打印命令进行解释：

DISK FILE OUTPUT (Y/N) ：

“No”响应使之在打印机上打出所希望的格式。

“yes”响应后，Word Star 提出：“OUTPUT FILE NAME？”。该文件的打印影像与输入的格式不同。扩展的点命令可用来适当延迟向打印机的传递。

START AT PAGE NUMBER (RETURN for beginning)？

指定开始打印的页号，页号后随一个 ESCAPE 或 RETURN键。

用 0 或 1 回答，则从文件的开头开始打印。若以非数字、非空字符回答，则重新提问，直到给出正确的回答。

STOP AFTER PAGE NUMBER (RETURN for end)？

打入页号和 RETURN 或 ESCAPE，完成给定页面的打印之后，停止打印，一个 0 响应还可使之继续打印，直到文件结束。

USE FORM FEEDS (Y/N) ：

以“yes”回答，Word Star 将送一个“走纸”符 (OCH) 代替页面中若干个换行，在页首也一样。但打印机必须具备响应换行功能，打印纸必须装在所规定的位置。

SUPPRESS PAGE FORMATTING (Y/N) ：

以“yes”回答，Word Star 将打印文件中的点命令，然后再对它们进行说明。页面格式直接由控制产生，也可用其补缺形式，即不执行点命令。若未设置分页功能，输出结果将在打印纸的一边。但文件大多数打印控制字符的解释与该问题的回答无关。

该操作会产生一个内容正确而格式不同的输出文件。它允许对点命令进行校验。也可用来打印那些已进行分页，但不是用 Word Star 建立的文件，以及用打印功能 DISK FILE OUTPUT 选择建立的磁盘文件。

PAUSE FOR PAPER CHANGE BETWEEN PAGES (Y/N) ：

装入类似信笺的单张纸时，应以 “yes” 回答。打完一页后，Word Star 使打印暂停，“PRINT PAUSED” 将显示在状态行上；更换了新的打印纸后，键入一个 P（非文件状态）或↑KP（编辑状态）使打印继续。

以“No”响应，则使打印连续地进行。

Ready printer, press RETURN：

先使打印机处于准备就绪状态，除选择了 USE FORM FEEDS 外，应使打印纸置第一页的顶上，该位置决定了以后各页的起始位置。

第二节　暂停和异常打印中止

打印过程中，若再给一个打印命令，会使打印暂停，显示被打印的文件名，并显示下面的信息：

TYPE "Y" TO ABANDON PRINT, "N" TO RESUME, ↑U TO HOLD :

回答"Y"，使打印异常中止。

回答"N"，继续打印。

一个中断控制字符（↑U）使它返回到非文件菜单或编辑状态，使打印暂停，下一个打印命令会使它继续打印。

第三节　打印中继

打印暂停时，可用一个打印命令使之继续。

打印会因下列几种原因而暂停：

1. 选择了"PAUSE BETWEEN PAGES"，且完成一页的打印。
2. 在文件中发现打印控制字符↑C。
3. 操作员通过打印命令使之暂停，然后对RESUME问题以↑U回答。
4. 选择DISK FILE OUTPUT时，磁盘满了。

打印暂停时，状态行显示"PRINT PAUSE"；编辑时，信息"TYPE ↑KP TO CONTINUE PRINT"显示在主文件菜单的上方。

第十二章　错误及错误信息

本章主要解释 Word Star 运行中出现的错误及其警告性提示信息。其他在装配（installing）Word Star 时出现的错误信息参阅第十三章。

第一节　磁盘正文文件

Word Star的许多信息，当需要时，是从磁盘文件WSMSGS.OVR读进来或显示出来的。其中包括大多数的错误信息、清单及许多解释信息。如果不存在信息文本文件，就会出现下列信息：

@@@@
FILE WSMSGS.OVR　NOT FOUND menus &
messages will display as @@@@ only

Word Star 将继续运行，但许多信息被@@@@取代；特殊情况下，只出现象 @@@@ 一样的菜单。所以，如果 WSMSGS.OVR 不存在，显示的信息和菜单就会减少到最低限度。

第二节　退出错误状态的键

许多编辑时出现的错误，在显示了错误信息后，需要用户按 ESCAPE 键。大多数这类错误信息有如下形式：

＊＊＊ERROR n：（相应的错误信息）＊＊＊press ESCAPE key

当有这样的信息显示时，要求按 ESCAPE 键，继续编辑。通常引起错误的命令不引起文件改动，也不移动光标。

当磁盘信息文本文件不存在时，将显示下列错误信息：

＊＊＊ERROR n：@@@@＊＊＊＊press ESCAPE key

@@@@显示出来代替了文本。如果此文本不存在，就会从 WSMSGS.OVR 取到。数值（n）仍然有相同的意义。

第三节　编辑时出现的错误信息

＊＊＊INTERRUPTED＊＊＊　press ESCAPE key

根据需要按一个中断键（↑U），就会出现上面的信息，来中止命令的执行。按 ESCAPE 键并继续编辑。

＊＊＊NOT FOUND：string＊＊＊press ESCAPE key

当用命令寻找（↑QF）、替换（↑QA）或者同时寻找替换（↑L）时，光标留在结尾（或开始）的左端。

* * * ERROR E5 : THAT PLACE MARKER NOT SET * * * press ESCAPE key

编辑文件时，出现这类错误，就得用“移动光标置标志”命令去置尚未设置的标志。按ESCAPE键，从错误信息中退出，继续编辑。

* * * ERROR E6 : BLOCK BEGINNING NOT MARKED (OR MARKER IS UNDISPLAYED) * * * press ESCAPE key

出现此类错误的原因是：没有首先置文件开始标志就执行下列命令：

成块搬家（↑KV），成块拷贝（↑KC），成块删除（↑KY），成块写（↑KW）。

执行这些命令要带有置块开始的命令（↑KB）。

要告诉Word Star从哪里开始块移动、块删除等，首先要将光标置于这一块的第一个字符处，并按↑KB。置好了开始和结束标志以后，再执行上述命令。

用↑KB（光标已置标志）以及用↑KH（屏幕/显示交替进行）屏蔽了块开始以后，上述错误（ERROR E6）也可能发生。需用↑KH来显示。

* * * ERROR E7 : BLOCK END NOT MARKED (OR MARKED IS UNDISPLAYED) * * * press ESCAPE key

与ERRORE6相似，只是END设置标志。告诉Word Star执行文件从哪里结束，要把光标放到最后一个字符，以便搬家、删除等等，并且按↑KK键。在置了块尾标志后，再执行以前的命令。

* * * ERROR E8 : BLOCK END MARKER BEFORE BLOCK BEGINNING MARKER * * * press ESCAPE key

在置开始标志之前先置了结束标志，Word Star将不知道究竟哪块要搬家、拷贝、删除或写。更正后，再执行以前的命令。

* * * ERROR E9 : BLOCK TOO LONG MOVE OR COPY IN TWO SMALLER BLOCKS * * * press ESCAPE key

文件块长度超过了Word Star成块搬家和拷贝的限度，执行时要分成二个较小的块处理。

块写命令却不受块大小的限制。

* * * ERROR E11 : THAT FILE EXISTS ON DESTINATION DISK DELETE EXISTING FILE FIRST OR USE A DIEFERENT DISKETTE * * * press ESCAPE key

原因：指定一个文件在某一磁盘上编辑，而此版本放在不同的磁盘上。这个信息表明同样的文件名和类型已存在目的驱动器中。如继续运行，目的盘上对应的文件不能丢失。

按ESCAPE键以后，Word Star进入非文件菜单，你可删除存在的文件。然后，再借助于D命令。有两种选择：一是插入不同的盘接受目的文件，二是回到操作系统状态，用REN（换文件名）更换某一文件的名字和类型。

can't edit a file of type .BAK or .$ $ $ rename or copy the file before editing

在非文件菜单状态，选用D、N命令时，如果文件以.BAK和$ $ $结尾，则会显示上述错误信息。如果想要编辑文件，需重新命名文件（用E命令）为其他类型。

ALLOW PRINT TO FINISH BEFORE EDITING A FILE

YOUR SYSTEM DOES NOT HAVE ENOUGH MEMORY TO PERMIT SIMULTANEOUS EDITING AND PRINTING

产生的原因：从非文件菜单发出 D 或 N 命令——如果用了打印功能，由于系统内存较少或操作系统设置重定位使得现有的内存都被 Word Star 占用。

* * * ERROR E12 : DISK FULL * * * press ESCAPE key

盘满了。当移动光标到一个长文件的开始（见第九章），在这种情况下，仍可以移动光标到文件的结束，但如果向前移动光标或存文件时就出现此类错误。切切注意：不要使你的磁盘空间满了。

从 DISK FULL 错误命令恢复时的注意事项：

1.如果光标正向大文件开始端移动，这时仍有可能移到最后。在这种情况下，用↑KS 通过保存使光标回到开头，然后移到所需要的位置。

2.如果光标向文件结束移动，或者当存文件时出现 DISK FULL 错误，首先用删除命令（↑K）删去不需要的文件，继续进行编辑。

3.如果已经做了大量的更动或增加，请不要贸然中止编辑，如删除文件仍不能得到足够的空间完成编辑，可采用下面一些解决方法：

a.如果在另一驱动器中磁盘有多余的空间，可将更换的一部分文件以块写形式放到该盘上，在安排好足够的磁盘空间后一块一块地重新结合。

b.删除没变动的文件，直到存文件时不产生 DISK FULL 为止。最后再想法恢复这些文件。

c.如果全部文件已经读进来，即发出↑QC 命令，能在屏幕上看文件结尾，用↑KJ 命令删除输入文件，然后存文件。也许 Word Star 意外地出现一个不幸的错误 F29。在这种情况下，如文件类型为 $ $ $，用 REN 命令变换类型，但没有 BAK 文件。应安排较多的磁盘空间并且作一个替代的副本。

4.如果 DISK FULL 是由于块写（↑KW）命令引起的，仍可用上面的方法恢复。删除（↑KJ）写的文件，因为它写的不完全并且没关闭。如果你能通过删除其他文件或用不同的驱动器来安排另外的磁盘空间，就可以重复块写，然后进行编辑。

有时你一按 ESCAPE 键，另一个 DISK FULL 错误又显示出来，表明磁盘真满了。

所以要注意：绝不要让此类情况发生——看看你的磁盘空间！

总之，避免 DISK FULL 错误的最佳方法是以预防为主。

FILE name typ NOT FOUND

当有另外读文件命令（↑KR见第八章）或打印命令（见第十一章）时所提问的文件找不到。你实际输入的文件名也出现在信息中，其信息出现在屏幕上提问文件名的下面，光标在提问后更换了。输入正确的名字——要牢记：如果需要，请同时输入驱动器名；或者，如不想完成此命令时，则按回车。可以向前移光标（↑D）和（↑R），不需要重新输入字符，从前面回答中可重新得到字符。

INVALID FILE NAME : × × × × × ×

与前面所描述的错误类似：你所输入的字符（通过上面显示的信息× × × × × ×反应）不是一个正确的文件名，文件命名法见第三章。D命令概括屏幕上文件名的形式。

注意：WordStar 不接受非标准符号的文件名，如“*、？”等，这些字符就会出现上面的错误。

...INTERNAL ERROR I15：INVALID COPY LENGTH***press ESCAPE key***

...INTERNAL ERROR I16：INVALID ADDRESS...press ESCAPE key***

...INTERNAL ERROR I17：MEMORY FULL***press ESCAPE key***

...INTERNAL ERROR I18：MEMORY SHORTAGE...press ESCAPE key***

...INTERNAL ERROR I19：POINTER>64K FROM CURSOR...press ESCAPE key***

...INTERNAL ERROR I36：BAD ONLY****...press ESCAPE key...

上述错误是内部错误，一般情况下不应该出现。我们建议：这时立即存文件并退到操作系统状态(因为内部信息正作为一个 BAK 文件副本篡改你的文件)，然后再执行 Word Star 检查你的文件，其中错误信息是否可能再出现(即：如果你能找出一条可靠方法使之再出现)，请报告给Micro International Corporation（国际微处理公司）。

第四节　警告性信息

下面是警告性信息。同时参看第十二章第七节的打印功能警告性信息。

***WARNING WORD TOO LONG TO FIT MARGINS

当形成一行时，Word Star 发现在当前左边界和右边界之间没有字间隙（如空格、连字符等）。例如，你在屏幕上输入一行“*”（如表头部分），字包有效，且未消除边界时，一旦超过了边界就会出现上面错误。你应该缩短删除剩余的“*”，这就靠你如何安排页的形式。在重新组成段时（↑B）也显示“字”太长了。

当字太长时，Word Star 在边界后面寻找 10 个字符中的间隔，找到了，就按右对齐重排；没找到，字则在边界处分开。

CAN'T DISPLAY PAGE BREAKS IN A NON-DOCUMENT FILE

在用N命令执行非文件编辑时，给出命令↑OP，此命令将无效。

PUT AT FILE BEGINNING FOR CORRECT PAGE BREAK DISPLAY

这些信息出现在屏幕行最显著位置上，当分页显示时，在居于文件文体之前用.PL、.MT、.MB或.LH点命令就会出现上面这些信息。它提醒你不会有动态分页显示，并回答这些点命令。所以在编辑期间可显示不同的分页，打印就可以看到。命令不进入文件，打印功能可解释出来，信息只在屏幕上，不进入文件中，如果页面中断没设置，或处于非文件编辑状态（N 命令），则这些信息不出现。

?

当命令未完成或点命令明显错误时，问号在屏幕最右端显著位置出现，在需要的地方用一个不可辨认的双字母码或一个无数字变量或者一个大于 255 的数（除.PN 命令外）就会

引起？显示。此信息看上去就象等待输入命令，但此时不要管它，直到命令全部输入为止。在文件编辑时不出现此现象（N 命令）。

FILE WSMSGS.OVR NOT FOUND menus &
massages will display as @@@@ only

启动 Word Star，在非文件菜单，同时/或者当文件编辑初始化时，如果信息文本（WSMSGS.OVR）在驱动盘上找不到，在当前盘也找不到时，就会出现上述错误。见第一节。

@@@@

磁盘文本文件 WSMSGS.OVR 不存在，Word Star 想从此文件中显示一些信息或清单（包括无文件清单）。参阅本章开始部分。

注意：即使 WSMSGS.OVR 不存在，大多数共同错误信息也会显示出来，或显示出文件的一部分（如果有错误序号，也一并显示出来）后跟@@@@，但菜单并不有效，只是@@@@出现在屏幕上顶端。

＊＊＊WARNING：
WRONG VERSION OR VSRSAS.OVR--
SOME MASAGES MAY BE INCORRECT＊＊＊

如果所用的磁盘文件 WSMSGS.OVR 来自不同于 Word Star（WS.COM）的版本则产生上述信息。信息文件也许不包括所有正确的信息，也许缺少某些信息。Word Star 继续运行，想要显示的信息不在文本文件中，就要产生：IF THIS DISPLAYS YOU ARE using WRONG VERSION OF WSMSGS.OVR

例外：第一个 WSMSGS.OVR 版本不包括 WRONG version of WSMSGS.OVR 警告性信息，早先的 WSMSGS.OVR 和后本的 WS.COM 一起用就会引起 IF THIS DISPLAY you are using wrong version of WSMSGS.OVR 信息。

！'S and keeps

一旦许多！出现在屏幕上，不管光标在什么地方，通常伴随终端发出嘟嘟声，发生此现象原因是：Word Star 接受磁盘输入其速度大于它处理的速度，并耗尽了它的容量不能处理存储以后的字符。这通常发生在 Repeat 键时或按下自动 Repeat 键。许多！字符以及嘟嘟声警告你的 Word Star 已失去敲磁盘的功能，停止敲键盘或者发生此现象时释放 Repeat 键。在处理完所有不丢失字符后，Word Star 均重新显示屏幕，移去许多！就可以继续编辑了。

＊＊＊WARNING DISK FULL
DELETING OLD.BAK FILE TO MAKE SPACE (NORMALLY, THE PREVIOUS BACK UP FILE IS DELETED ONLY AFTER EDIT IS SUCCESSFULLY COMPLETED)

如果你处理的是中等大小的文件，这个信息警告你磁盘满了。在处理前必须取得大量的磁盘空间（通过删除不需要的文件或移去一些文件到新盘中）才能存进你的文件。参阅第三章第七节关于盘空间和文件大小及第十一节有关.BAK 文件部分。

当处理大的文件使得这样三个文件副本不能装入磁盘时（两个副本，如果你已指定不同的目的盘），这个信息总会在编辑过程中出现，所以不要理睬它。详见第九章。

Warning：you are editing the same file as you are printing

Word Star will not allow you to save the edited version
untill the print has completed or has been abandoned

Word Star 允许同时编辑和打印同一文件，但编辑的文件在打印过程中不能存入盘中。当你打印时要回到编辑初始状态，就显示上述信息。

第五节　情报性信息

FINISHING PRINT BEFORE EXIT

（按↑U取消 EXIT 命令）

当打印功能有效时，如果从非文件菜单给定X命令或编辑时给出↑KX 命令，就会出现上述错误。打印将继续进行；当打印结束时才退出操作系统状态。

FINISHING PRINT OF SAME BEFORE SAVING

（按↑U取消存命令）

当一边打印一边编辑时给出存命令（↑KD，↑KS，↑KX）就会出现上述错误信息。Word Star 等到打印完成，然后存文件。如果在打印和存文件完成之前要用其他命令，可用↑U中断存命令。注意，如果打印暂停，Word Star 将永远等待。在这种情况下按↑U 取消存命令，然后给出适当命令，继续或取消打印。

FINISHING PRINT OF .BAK FILE BEFORE SAVING

（按↑U取消存命令）

如果编辑.BAK 文件时又打印，如果有存命令，则出现上面信息（与前一信息类似）。

第六节　致命错误

下面错误会中止Word Star运行，并返回到操作系统状态。

You are trying to run an uninstalled Word Star
Please run INSTALL first

一旦调用由 Word Star 盘A提供的未经装配的 Word Star（文件 WS.COM）就会出现上述错误。在试图运行Word Star 之前，必须装配好由 B 驱动器提供的装置程序，并回答所有的提问完成 Word Star 的装配。参阅第十三章关于装配的详细内容。

＊＊＊FATAL ERROR F23：INVALID SCREEN HEIGHT OR WIDTH

只有在通常配置过程中用补缀（patching）法出现错误时，才有可能出现上述错误信息。所置的屏幕高度不在 16～120 行之间，或者宽度不在 64～250 列之间时，就会有上述错误信息。

＊＊＊FATAL ERROR F25：NOT ENOUGH MEMORY
OR YOUR OPERATING SYSTEM IS NOT RELOCATED TO MAKE ALL RAM AVAILABLE

在启动 Word Star 时，没有足够的内存供 Word Star 用就会出现以上错误。注意，你的操作系统（CP/M）必须重新定位，使之在内存顶端操作。用 5.25 英寸软盘的 Word Star

用户可以执行CP/M盘中的“MOVCPM5”程序完成此操作，用8英寸软盘的用户用“MOVCPM 8”程序。CP/M手册对这些程序和它们的用法有详细的解释。一旦你的内存扩展了，在内存之前必须重复进行这样的重定位。

＊＊＊FATAL ERROR F27：DISK ETTE DIRECTORY FULL

磁盘文件目录项已超过所拥有的大小范围。很少发生这样的情况，因为字节容量通常是很宽裕的，但如果你用许多小文件如一页多的字符或者只有巴掌那么大的一段作为一个文件，你也许有必要看一看文件数目。

磁盘最大的文件目录项数依赖于系统，但通常在单密度盘中有64个文件目录项。每个文件需要另外一项，供每一附加的16K或剩下的一部分也要一项。当计算文件个数时，记住Word Star能产生两个工作文件，每一个文件可能与编辑的文件一样长。

＊＊＊FATAL ERROR F28：CLOSE FAILURE
SYSTEM FAILURE, OR YOU CHANGED DISKETTES

＊＊＊FATAL ERROR F29：RENAME FAILURE
SYSTEM FAILURE OR YOU CHANGED DISKETTES

这些信息不应该出现。它表明操作系统中出现某种错误，或者在编辑时更换了磁盘，或者用↑KJ命令删除了输入文件或工作文件。

＊＊＊FATAL ERROR F46：overlay file WSOVLYI.OVR NOT FOUND

当执行Word Star时，文件WSOVLYI.OVR一定要在驱动器A的盘中(或现行盘)。从文件盘中拷贝文件（或者装入含有文件的磁盘），然后重新执行Word Star。

第七节　打印功能信息

下面信息在进行对话，初始化打印时可能出现：

FILE name typ NOT FOUND

INVALID FILE NAME：××××××

表示输入不存在文件的或非法的文件名，见第三节。

注意：你同时编辑和打印同一文件，最新存的版本将打印出来，不影响没存的文件。另外，Word Star不允许一边打印一边存正在编辑的文件。

出现这个警告，是由于使用了↑KP命令初始打印时且输入的文件名正是所编辑的文件名。此警告提醒你Word Star在同一文件打印时不准使用存命令(↑KD, ↑KS, ↑KX)，Word Star只打印磁盘中存的文件——编辑时还没存就不会打印。

END EDIT (↑KD) BEFORE STARTING PRINT
YOUR SYSTEM DOES NOT HAVE ENOUGH MEMORY TO
PERMIT SIMULTANEOUS EDITING AND PRINTING

如果你的系统没有足够的内存RAM支持目前打印和编辑，则当按↑KP命令时，即出现上面信息（也许，如果你的操作系统没有重定位使得所有RAM都有效，见上节错误F25的描述）。

当进行磁盘文件输出时，可能出现下列错误：

＊＊＊FATAL ERROR F27：DISK ETTE DIRECTORY FULL

见上节解释。

＊＊＊PRINT OUTPUT DISK FULL.PRINT PAUSED＊＊＊

表示正进行写操件的打印输出文件所在的盘满了。打印自动暂停，象往常一样“PRINT PAUSED”出现在状态行上。在此信息后，你可以先使有效的磁盘空间增大（例如，删除文件，参阅第三节ERROR E12 中的讨论），然后继续打印，即用通常命令 P（在无文件菜单时）或用↑KP（在编辑时）。如果你继续打印而增加磁盘的有效存储空间，PRINT OUTPUT DISK FULL信息在一秒钟后又会出现。

在输出磁盘满了后，如果想取消（暂停）打印状态，就连续迅速地按两个打印命令，然后对第二个打印命令提出的问题回答Y——即在非文件清单时，打 PPY；在编辑时，打↑KP↑KPY。在取消后，你可能要删除磁盘文件，因为它没完全执行并且没有关闭文件。所以要记住在删除文件前取消打印。

除了上述以外，规范打印功能没有错误信息——在其他条件下，象非法点命令，是忽略一个假想的命名处理的。参阅第十一章“打印功能”。

第八节　操作系统的一些错误信息

下面是一些联系到操作系统和 Word Star 系统时可能出现的系统错误信息。文本随系统变化而变化，这里只举出一些具有代表性的信息。

LOAD ERROR or TOO BIG

当内存大小或者操作系统没重定位，以取用较大的内存时，一旦执行 Word Star 就发现上述错误。这个信息表示内存不够，不能满足 Word Star 程序（WS.COM）的要求。

DISKd：NOT READY

表示 Word Star 存取的磁盘在驱动器中没有盘存在，或者盘存在但驱动器门没关上。Word Star 总是要求盘在驱动器 A 中或在现行盘中或者由 Word Star 命令所指的任何驱动器中。当插入盘以后，大多数系统立即开始执行命令。如果盘没插入也不等几秒钟就发出命令，就会出现上述信息。这种情况可不必理睬。

BDOS ERR R/O

如果你非法更换了磁盘或系统出现（>A）时换盘不打↑C，就会出现上述错误。出现这个信息，参阅第三章第十节关于换盘方面的内容。

第九节　其他错误信息

＊＊＊ERROR E38（～42）：BAD OVERLAY FILE.OR
WRONG VERSION OVERLAY FILE＊＊＊press ESCAPE key

＊＊＊ERROR E43（44）：WRONG VERSION OVERLAY FILE＊＊＊press ESCAPE key＊＊＊

上述错误通常由以下情况引起：

1.用的是不正确的 WSOVLYI.OVR 版本（来自不同的 Word Star 版本）。

2,当系统出毛病时，破坏了 WSOVLYI.OVR 文件或拷贝时出现错误。

解决方法：从源工作文件盘拷贝新的 WSOVLYI.OVR 副本。如果还有问题，找你的卖主求助。

* * * ERROR E45-46 : overlay file WSOVLYI.OVR NOT FOUND * * * press ESCAPE key * * *

WSOVLYI.OVR 文件（由源工作文件盘提供），一定要在 A 驱动器中或当前驱动器中。

* * * ERROR E47 : FILE MERGPRIN.OVR NOT FOUND

(The separately supplied file MERGPRIN.OVR is required for use of merge-print) * * * press ESCAPE key * * *

出现原因：在非文件清单状态键入 M，但文件 MERGPRIN.OVR 不在驱动器A盘或现行盘中。

* * * ERROR E52 : PROGRAM IS AN EMPTY FILE ! ?*** press ESCAPE key * * *

出现原因：在非文件菜单状态下用 R 的过程中，输入了非法的文件名。

* * * ERROR E53 : PROGRAM TOO BIG FOR MEMORY AVAILABLE UNDER Word Star * * * press ESCAPE key * * *

原因：在 Word Star 运行程序时，内存不够。要运行你的给定程序，必须从 Word Star 中退出。

FILE WS.COM NOT FOUND-

can' t RUN a program unless WS.COM is available

原因：在驱动器A或现行盘中找不到 WS.COM文件（或其他由装配时指定的其他文件名）。在执行其他给定的程序后，要返回到 Word Star 必须保证有 WS.COM 文件。

注意：如果 WS.COM 改了名以后也出现此错误，你要把 WS.COM 改名，就得重新装备但也不必全部都要重新装备，只要用装备选择项 B 和 C（见第十三章），输入现在存在的文件名，过一会儿输入你理想的新文件名，然后用回车键，保持其他选择项不变。

FILEd : file name.typ NOT on SAME DRIVE

用 E 命令时，新文件名与磁盘上某一文件名发生冲突。选另外一个名字或直接重新命名磁盘上存在的文件名。

FILEd : file name.typ NOT on SAME DRIVE

E 命令时，两个文件要在同一驱动器中，不能用 E 命令从一个驱动器文件传到另一个驱动器中。

TOO LITTLE MEMORY TO COPY WHILE EDITING

在编辑时要拷贝文件，系统内存不够。如果要拷贝文件，首先必须结束或退出目前的编辑状态。

FILE d : file name.typ EXISTS-OVERWRITE (Y/N) :

如果想把文件拷贝到盘中已存在的文件上去，按 Y，就删去了盘上存在的文件内容，然后再拷贝；按 N 键不妨碍存在的文件并退出现行文件，但按了 N 键后，则提问 "NAME OF FILE TO COPY TO?" : 提示你要拷贝到哪个文件上去。

第十三章 装 配

并非每个人都要进行 Word Star 装配。是否需要装配，可根据以下情况决定：

使用 Diablo1640/1650 菊花瓣字轮打印机，就不需要装配。因为 Word Star 磁盘已经装配过，可以使用该设备了。

使用 Diablo630 菊花瓣字轮打印机，必须用CP/M CONFIGUR.COM 文件，按本章第二节所述的操作来重新确立该系统的波特率。

使用其他打印机，应调用 INSTALL.COM 和CONFIGUR 两个文件，遵循整个 Word Star 装配过程进行装配。

本章介绍的装配过程，可以生成协调你的硬件系统的Word Star。Word Star磁盘上有一个名为 "INSTALL.COM" 的文件，该文件允许执行装配过程的第一部分来定制系统，以适合你的打印机。定制其他系统部件，象终端之类，INSTALL.COM 程序还可以提供一些选择方案。如果你的系统已适合这些部件了，它将告诉你无需选择。

执行 Word Star 定制过程的第二部分是利用CONFIGUR 文件进行的。大多数 CP/M 系统盘上均配有这个文件。开始装配的时候，两张 Word Star 盘和一张 CP/M 盘都要插入磁盘驱动器。

第一节 打印机装配

一、各种适用的打印机

Word Star 支持两种类型的打印机：菊花瓣字轮及其同类型打印机，和"类电传" 打印机。这里，我们将介绍这些打印机的功能和特性，以及有关打印机的装配。

菊花瓣字轮及其同类型打印机：这些打印机能够横向和纵向运动，允许使用各种行幅、字符间隔和角标以及打印增强功能。由于各台菊花瓣字轮及其同类型打印机的控制顺序不同，所以必须按照待用的打印机来装配 Word Star。下面各种菊花瓣字轮及其同类型打印机均可通过菜单（menu）的选择来装配其 Word Star。菜单选择在第二节中详述。

NEC 5500D OEM 打印机（带 Micropro"I/O Master" 接口板）

Diablo 1300串行Hy—Type IIOEM 打印机（带Micropro"I/O Master" 接口板）

Qume Sprint 3 OEM 打印机（带适配器和Micropro"I/O Master" 接口板）

Diablo 1610/1620

Diablo 1640/1650

Qume Sprint 5

NEC Spinwriter 5510/5520

当你从打印机菜单上选择了某一种打印机的时候，Word Star 就被装配成可以产生那台打印机全部功能的用途。对于上面任何一种打印机，Word Star 都能双向打印。对于类电传打印机（菜单上没有明确指出），INSTALL.COM 程序也允许为它生成系统，但是，某些功能需要修改才能实现。

"OEM"菊花瓣打印机：通过 Micropro"I/O Master"接口板和菜单选择，Word Star 可以被装配成驱动 NEC 5500D、Diablo Hy—Type II 1300 串行和 Qume Sprint 3（带电缆适配器）OEM 打印机。使用带"I/O Master" 接口板 OEM 打印机的用户，可以不考虑本节中的剩余内容。

"串行" 菊花瓣打印机：这些打印机（不属上面 "OEM" 之列）通常带有串行接口。以 Word Star 来使用串行菊花瓣打印机时，总是希望把它和计算机连成打印机能接受的最高传输速度（波特率）。

"类电传" 打印机：这些打印机均打印EPASCII码字符（代码20～7E16），并且响应回车和换行代码。使用类电传打印机，Word Star 可以下划线、删除、重复打印和隔行打印；撞击三次能打印粗体字；能在一行上、下打印上标和下标。但是，不可以改变行幅。

如果使用为类电传打印机所装配的 Word Star 来操作菊花瓣字轮或同类型打印机的话，只可能具有类电传打印机的功能，而不产生双向打印。

类电传打印机菜单选择有两种：

A．任意类电传打印机

B．能回空的类电传打印机

这两个选择项都可以装配 Word Star 来使用类电传打印机的所有共同的基本功能。B 项能较快地产生下划线、粗体字、重复和回空等打印。如果知道你的打印机有回空能力就选择B项，否则就选择A。

其他打印机：通常，可按类电传打印机选择项进行。

两种类电传打印机选择都这样设定：在送一个后面不跟换行符的回车符时，打印头只回到现行行首，而不进入下一行。如果打印机不能回车换到下一行（但可以回空），就不能隔行打印，不过 Word Star 的其他操作仍然合理。如果打印机既不能回车换到下一行又不能回空，则只好选择 A，取消下划线、删除、粗体字、重复、回空和隔行打印等功能。

二、通讯约定

"通讯约定" 是这样一种约定：打印机（串行）在打印被接受字符时，告诉 Word Star 应该什么时候停止发送字符，什么时候恢复发送。

这里讨论的通讯约定与"OEM" 打印机无关。

大多数类电传打印机不需要通讯约定。因为接口硬件能正确处理。

（串行）菊花瓣字轮或同类打印机，其接口波特率为150（每秒15个字符）或更少一些，一般不到300（每秒30个字符）的打印机也不需要通讯约定。

接口波特率为1200（每秒120个字符）的串行菊花瓣或类似打印机总是需要有通讯约定的（带特殊电缆的NEC5510/5520除外）。如果没有装配通讯约定，打印机缓冲区将会溢出，且要丢失字符。一般在发生这种情况时，打印机蜂鸣器将发声，某些打印机则停止打印。

这里介绍两种 Word Star 所支持的通讯约定："ETX/ACK约定" 和"XON/XOFF 约定"。

ETX/ACK 约定：在这个约定下，WordStar送一个规定字符最大数后跟一个"ETX" 字符所组成的信息。打印机打印所有字符，直到 ETX 时，便发送一个"ACK" 字符反馈给计算机，告诉 Word Star 可以发送另一组信息。Word Star所支持的所有菊花瓣字轮及其同类型打印机，都可以使用这个约定，尽管某些打印机需要特殊的操作来管理它。

XON/XOFF 约定：在这个约定下，当打印机发送一“XOFF” 字符（ASCII码DC 3）给计算机时，WordStar 便停止发送字符；当打印机发送一“XON”字符（ASCII码 DC1）时， WordStar 恢复发送。某些菊花瓣及其同类型打印机支持 XON/XOFF 约定作为替换约定。如另作考虑，比如有其他软件要在同一台打印机上运行，或者某台打印机只执行 XON/XOFF 而不执行 ETX/ACK 约定，建议用XON/XOFF 替代 ETX/ACK。

三、打印机驱动器

使用 NEC 5500D、Diablo Hy-Type II 和Qume Sprint 3 OEM（带 Micropro“I/O Master” 接口板）的用户可以不考虑这一部分。在装配期间，选择 OEM打印机与 I/O 主接口相连，就自动地选择了所需要的驱动器。打算驱动串行菊花瓣打印机速度达1200波特的用户，必须认真阅读该部分内容。

“清单输出” 设备不允许程序去确定打印机是否忙，或准备好接收另一个字符。确定打印机是否忙（“打印机忙测试”）不是 WordStar 操作所必须的。但是在编辑和打印同时进行期间，希望用它来改善键盘响应和提高打印速度。

为了处理通讯约定和打印机忙测试以及适应特殊情况，WordStar 提供几种主要的访问打印机的途径，每种途径都称为打印机驱动器。在装配 WordStar 的时候，可以选择以下几种主要打印机驱动器的一种，这在下面几段中将详细介绍。在装配期间可从菜单（见第二节）上选择所要求的打印机驱动器。

CP/M“清单输出” 设备（LST：）；

CP/M“TTY：” 控制台设备；

CP/M“CRT：” 控制台设备；

CP/M 清单输出设备（LST：）：它是借助于操作系统中的打印机驱动器输出字符给打印机的。如果打印机不需要通讯约定，也不需要打印机忙测试，那么LST：就是访问打印机最简单的途径。当需要通讯约定时，可以选择另一种 WordStar 的打印机驱动器。

装配过程的第一部分是“CP/M 清单输出设备的方案选择（从驱动器菜单上选择，如第二节所述）。这些可供选择的方案已被 INSTALL.COM 程序简化。对几个认可问题做出肯定回答之后，将自动退出 CP/M 操作系统。退出CP/M，就可以使用 CONFIGUR·COM 程序来执行装配过程的第二部分。

为了证实操作系统已可以驱动打印机，且 LST：的确已分配给打印机，可在系统提示符（A>）之后键入 Control-P，再键入一些其他的字符。这些在键盘上敲过的字符均应被打印机打印出来，直到再敲 Control-P为止。

控制台转换：在多种 CP/M 文本中，“控制台” 的概念是一台逻辑设备（CON：）。它可以定义为四种物理设备（通常称作“TTY：”，“CRT：”，“BAT：” 及“VL1” 的任一个。操作员使用的终端就是其一。它在调用 Word Star 时必须工作。Word Star 亦可把打印机作为另一种控制台来访问。在打印机驱动器菜单上作适当地选择，装配了 Word Star 之后，Word Star 就能把打印机当作 TTY：或 CRT：进行访问（见第二节）

在选择控制台访问打印机的情况下，允许从打印机输入字符作为通讯约定的需要，无需修改系统软件或者另加补钉。在使用波特率为1200的菊花瓣字轮打印机，需要通讯约定时，TTY：与 CRT：打印机驱动器往往是最方便的选择方案。在这种情况下，一般不提供打印机忙测试。

TTY：定义为打印机驱动器：这种驱动器以 TTY：控制台设备访问打印机。如果你的 CP/M 支持多种控制台，则可以使用这种驱动器，并且提供通讯约定。操作 Word Star的终端不是 TTY：设备。

CRT：定义为打印机驱动器：与 TTY：相类似，这种驱动器以 CRT：控制台设备访问打印机。如果你的 CP/M 支持多种控制台，可以使用这种驱动器，并且提供通讯约定。操作 Word Star 的终端也不是 CRT：设备。

四、有关打印机装配选择摘要

带 Micropro I/O Master 接口板的 OEM 打印机：在 INSTALL 程序的打印机菜单上（将在第二节中讨论）只能选择 "I/O Master/OEM" 打印机。

类电传（非菊花瓣）打印机：如果你的 CP/M 能够驱动这种打印机，就在菜单上指定"类电传"，并指定"无" 通讯约定和"CP/M 清单输出设备"打印机驱动器。

300波特或以下的串行菊花瓣字轮及其类似打印机：如果你的操作系统能够驱动这种打印机，就在菜单上选择它，并指定"无" 通讯约定和"CP/M 清单输出"设备驱动器。

1200波特的串行菊花瓣及其类似打印机：选择这种打印机需要用到通讯约定，其驱动器除了能输出字符外还要能输入字符。

五、置打印机开关

某些打印机有一些选择开关，必须把它们置位。这些开关可能在外部控制面板上，也可能在打印机盖子下面的控制板上，或者设在打印机里面的电路板上。常见的开关包括：

AUTO LF或LOCAL LF（在接收回车符时产生换行）：必须断开。

AUTO CR（打印超出一行时则换行）：建议断开。

SPEED：必须与计算机发送字符的速率相匹配。

PARITY：置成不考虑奇偶校验（某些打印机上的"M" 位置）。

Communications Protocol：如上所述，1200波特的串行菊花瓣字轮及其类似打印机必须使用通讯约定，许多打印机要置开关或在电路板上加跳线。

FORM LENGTH：使用"标准走纸" 的时候，应考虑与所用的打印纸相匹配。

SET TOP：使用"用户标准走纸" 的时候，把打印纸放到格顶之后按此开关。

第二节 装配过程

在装配 Word Star 之前，建议从 Word Star 配给盘上把 WS.COM，WSOVLY1.OVR，WSMSGS.OVR 和 INSTALL.COM 等文件，拷贝到自己的系统盘上。

一、通过装配选择硬件

开始 Word Star 装配的时候，先打命令

INSTALL

并按 RETURN键。INSTALL 经过几秒钟的存取之后，显示标志信息，并询问第一个问题。此时屏幕上呈现

A>INSTALL

COPYRIGHT(C)1981, Micropro International Corporation

INSTALL Version x.x for Micropro Word Star ralease n.n
Do you want a normal first—time INSTALLation of Word Star?
(Y = yes;N = display other options) :
响应这个问题应敲N。屏幕上出现 WordStar 装配菜单。

---

* * * * Word Star INSTALLATION OPTIONS MENU * * * *

A INSTALLation of a distributed Word Star, INSTALLing WS.COW, producing WSU.COM, and then running the INSTALLed Word Star.

B INSTALLation or re-INSTALLation of a Word Star COMfile of your choice, placing the newly INSTALLed Word Star in a file of your choice, and then exiting to the operating system.

C Same as B except run the INSTALLed Word Star.

D Modification of the INSTALLation of a Word Star COM file of your choice. The modified Word Star replaces the file. The modified Word Star is then run.

PLEASE ENTER SELECTION (A, B, C, or D) :

送"B"，选择装配选择项B。

它首先问要送入的文件名：

Filename of Word Star to be INSTALLed? :

通常，这是 WS.COM文件。也可以送"B：WS" 文件，以便从驱动器 B 那里获得 WS.COM（尽管当前联入的是驱动器A)。还可以送先前已经装配过的 Word Star文件名（即 "WS")，以便改变某些装配选择而无需重送它们。

送过 WS.COM 文件名之后，按 RETURN 键。如果这个文件不在现行联入的驱动器的盘片上，就在它驻留的驱动器名之前加上该文件名和一个冒号。

文件名的打印错误可以用 control-H 或 rubout（错误信号）靠回空来改正。删除 整个回答可用 control-U。结束回答用 RETURN。如果忘记文件名中的句点和文件代号，INSTALL 将呈现.COM形式。

如果 INSTALL 没有找到你指定的文件，INSTALL 将回答

THAT FILE DOES NOT EXIST

然后它再次问要装配的 Word Star的名字。

当在盘上找到所送的文件名时，屏幕上就出现下一问题：

Filename for saving INSTALLed Word Star?

对于这个问题，可打入你要求 INSTALL 去保存的 Word Star的文件名，这是为你的终端和（或者）打印机而定制的文件。对此问题一般用"WS" 来回答；也可以送"B：WS" 来保存别的驱动器盘片上的Word Star；还可以送另一个文件名而 不是"WS" 。然 后 按 RETURN。

接收了上述两个文件名之后，选择项B之下的对话继续进行，其过程下面再谈。经最后认可对问题的回答之后，INSTALL 就按照指定的名字将装配好的 Word Star保存（记录）

起来，然后退回操作系统。至此，选择项B尚未运行该已装配的 Word Star。

若想修改上述选择项 B 之下装配的WordS tar，可以按 U 键；如不需要改变对菜单的选择就按空格键。不改变，INSTALL 则显示“前面选择保留不变”以及所选择的终端、打印机、驱动器和通讯约定等作为响应，并且允许你按照常规的方式回答 OK（Y/N）：来确认。

终端选择“菜单”如下：

Micropro Word Star release n.n serial 000000

•••••Word Star TERMINAL MENU#1•••••

A	Lear-Siegler ADM-3A	C	Lear-Siegler ADM-31
D	Hazeltine 1500	E	Microterm ACT-IV
F	Beehive 150/Cromenco 3100	G	IMSAT VIO
H	Hewlett-packard 2621 A/P	I	Infoton I-100
J	Processor Tech SOL/VDM	K	Soroc IQ-120
L	Perkin-Elmer 550(Bantam)	2	Terminal Menu#2
3	Terminal Menu#3	Z	None of the above
U	No Change		

PLEASE ENTER SECTION(1 LETTER)：

由于Word Star 系统是按照使用H/Z-89终端定制的，故按U键选择“NO Change”。

送入终端类型字母之后，INSTALL 显示一段简短的信息用以确认终端类型，并且提问

OK（Y/N）：

假如产生了错误或者想改变选择，就按N键再次提问终端类型问题。再问时，就再按一下U键，选择菜单上的“No Change”。当你对 OK（Y/N）：回答Y时，INSTALL 就进入打印机选择。对这个问题或下述任何一个问题，可以送一个B，即能回到终端选择菜单。

1.有关打印机的选择：

对打印机装配Word Star 包括选择特定的打印类型、通讯约定和打印机驱动器，如第一节所述。

INSTALL 显示下面的菜单，允许选择所装配的Word Star 将要驱动的打印机类型：

•••••PRINTER MENU•••••

(More specific info is displayed after choice is entered)

A Any“Teletype-like”printer(ie almost any printer)

C “Teletype-like”printer that can BACKSPACE

D DIABLO 1610/1650 daisy wheel printer

E DIABLO 1440/1650 daisy wheel printer

F QUME Sprint 3 daisy wheel printer

G NEC Spinwriter 551/05520 thimble printer
I "Half-Line-Feed" Printers
M I/O MASTER/O·E·M· printer combination
U no change
Z none of the above
PLEASE ENTER SELECTION(1 LETTER):

如果打印机类型已列在表中，就送入适当的字母。如果不了解打印机，就送入A（如果知道你的打印机能够回空就送C）。有关打印机的选择见第一节。接着，INSTALL将显示一段说明有关这个打印机选择的具体考虑的信息，然后询问

OK (Y/N):

假如想改变选择，就敲N，打印机菜单问题则将重新显示。对重现的菜单，敲Y。同时，也可以送一个B，返回到这段对话的开始。如不改变选择就按空格键，使先前对问题的回答生效。

选择上面菜单中的M项，适用于与 Micropro "I/O Master" 接口板相连的所有"OEM"菊花瓣字轮及其同类打印机（NEC5500D、Qume Sprint5、Diablo HyType II）。有关这些与"I/O Master"接口相组合的打印机的讨论见第一节。为使用具有"I/O Master"接口的打印机而装配 Word Star 的时候，由于既不需要通讯约定又不需要选择驱动器，所以选择了M项就跳过下面两个菜单。

如果选择M，就按本节前面所述内容进行。

下面展示的菜单让你选择通讯约定，象本节前面所讨论的那样。

·····COMMUNICATIONS PROTOCOL MENU·····

A "Communication protocol" is necessary with some printers to prevent printer buffer overflow and character loss.
E ETX/ACK prntocol
X "X-ON/X-OFF" pretocol
N NONE required(or handled outside of WordStar)
U no change
PLEASE ENTER SELECTION(E,X,N,B,or U):

按N键，选择"NONE required（即超出 WordStar 处理的范围）"。如果你想回到前面的菜单而不需要现在的选择约定的菜单，就送 B，引出从终端菜单开始的先前那些菜单。除B以外，选择任一项之后，INSTALL 程序将显示一段确认该选择的信息，然后问"OK (Y/N):"，对上述的那些菜单的选择，如表示满意就回答"Y"。

最后是有关打印机驱动器菜单的选择：

---

* * * * * DRIVER MENU * * * * *

or, how should Word Star send characters to your printer?
L CP/M "List" device
P Direct I/O to 8-bit port
S User-installed driver subroutines

U no change
PLEASE ENTER SELECTION(L, P, S, B, OR U):

送L作为这个菜单的合适选择。INSTALL程序再次用"OK (Y/N) :" 信号来询问你的选择是否认定，除非你选择"B" 引出前面的那些菜单。如果对你的选择表示满意，就敲Y键加以认可。

2.旁路补钉程序

在送完驱动器菜单选择之后，将出现下面的显示：

ARE THE MODIFICATIONS TO WORD STAR NOW COMPLETE?
IF THEY ARE ANSWER YES TO THE NEXT QUESTION
IF YOU WISH TO MAKE ADDITIONAL PATCHES TO THE
USER AREAS IN WORD STAR ANSWER NO TO THE NEXT QUESTI-
ONOK(Y/N):

正常的装配，回答Y。如果你送N，补钉程序将要起作用。补钉程序允许对定制Word Star作另外的修改。第一次装配Word Star时一般不需要补钉。

3.确认或者修改你的选择

至此，所有装配选择均已产生。下面，INSTALL将显示你所选择的终端类型、打印机类型、通讯约定和打印机驱动器，然后再一次询问OK (Y/N) :。如果你打算做更改，就送N或B，INSTALL便回到终端菜单，使你能够反复进行选择。如果不打算改变这些选择就敲一下空格键，INSTALL便保持先前的选择。

如果终端、打印机、通讯约定和驱动器等装配选择均是理想的，就敲Y键。这样，就在装配过程的第一部分期间最后确认了你所作的各项选择。确认之后，自动回到CP/M操作系统。然后可以执行装配过程的第二部分。

二、用CONFIGUR来确定打印机特性

在你通过INSTALL.COM程序确定了各种选择之后，屏幕上出现CP/M操作系统提示符。响应这个提示，应该送"CONFIGUR"文件名，并按RETURN键。CONFIGUR是驻留在许多CP/M配给盘中的一个程序文件，它用于定制系统。CONFIGUR的详尽论述见CP/M手册。

引入CONFIGUR时，它将作如下显示：

Heath/zenith Configuration Program
Version 2·2·02
Serial Numbei: nnn-nnnnn
This program configures the CP/M operating system to a particular hardware environment.
Please wait during hardware verification···
H/Z89 with 64k of random access memory(RAM)
03 minifloppy drive(s)
CRT bard rate is 9600
02 additional serial ports found

Standard system(Y or N)?<Y>:

对此标准系统文本的响应应送一个N，这将产生以下主菜单：

A Set Terminal and Printer Characteristies
B Set Disk Parameters
C Change the Default I/O Cofiguration
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:

按A键，选择"置终端和打印机特性"（可任选其一），将出现下面的子菜单：

A CRT: Baud rate: 9600 Port: 0E8H = 350Q
B TTY: Baud rate: 300 Port: 0D0H = 320Q
C LST: Baud rate: 1200 Port: 0E0H = 340Q
D UR1: Baud rate: 300 Port: 0D8H = 330Q
E UP1: Baud rate: 300 Port: 0D8H = 330Q
F Force output to upper case on CRT: FALSE
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: FALSE
Y Finished, make changes and return to main menu
Z Quit, make no changes and return to main menu

Selection:

从这个菜单上选择C项，使系统适合你的打印机波特率。CONFIGUR 将显示：

LST Baud rate:

对此，用你的打印机波特率除了最后两位数的其他数字作为响应。CONFIGUR 将在同一行上显示：

00 Port:

对此，应送字符"0E0"

然后，CONFIGUR 将显示"置终端和打印机特性"菜单和你改变的信息。如果送入的波特率和接口号有错，可敲C键重新送入正确数。这些值均被正确记录之后就可送 Y，回到主 CONFIGUR 菜单：

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.

这时，按C键便引出“改变非法 I/O 结构” 子菜单，它在屏幕上显示如下：

A CON: =CRT: Available TTY:CRT:BAT:UC1:
B RDR: =UR1: Available TTY:PTR:UR1:UR2:
C PUN: =UP1: Available TTY:PTP:UP1:UP2:
D LST: =UL1: Available TTY:CRT:LPT:UL1:
Y Finished, make changes and return to the main menu
Z Quit, make no changes and return to main menu
Selection:

从这个子菜单里，选择可以替换的D。CONFIGUR 将显示：

LST: =

如果你的打印机是 Diablo 1640/1650或者 Diablo630，就对此显示送“UL1”。如果是其他类型的，就送“LPT”。

CONFIGUR 将再次显示“改变非法的I/O结构” 子菜单。按Y键返回主 CONFIGUR 菜单。

再按 Y 键，从 CONFIGUR 程序彻底退出。这样便回到CP/M 操作系统，就象驱动器字母提示符所指示的那样。出现这段提示符的时候，就可以送入你的 Word Star 文件名，并能开始使用它。

第三节 装配错误信息

如果显示出一段信息而你没能发现这段已自身说明问题的信息，参阅这一部分内容。

NOT ENOUGH MEMORY TO RUN INSTALL,
OR YOUR OPERATING SYSTEM IS NOT
RELOCATED TO MAKE ALL RAM AVAILABLE.

如果计算系统没有足够大的内存，引用 INSTALL 时将产生这段信息。甚至于尽管你的计算机有足够的内存，但是操作系统（CP/M，或者相当的）已经不可能置换 RAM 的时候，也会发生这段信息。置换内存是通过系统命令 MOVCPM5 或者 MOVCPM8 来完成的，详见系统文献。

NOT ENOUGH MEMORY TO RUN Word Star
OR YOUR OPERATING SYSTEM IS NOT
RELOCAED TO MAYE ALL RAM AVAILABLE.

当存储能力足够运行 INSTALL 但不足以运行 Word Star的时候，将产生这段信息。如得到这种信息，仍然可以引用 INSTALL 并指定装配选择项 B；也可靠磁盘文件来装配 Word Star，以便在系统具有更多的内存之后再使用，或在不同的系统上使用。这种错误信息只可能在正常的装配期间以及装配完成后选择运行Word Star 方案的时候产生。

WS.COM NOT FOUND ON CURRENT DISK

这个信息发生在正常装配期间，或者在装配选择项A的情况下，在联机驱动器上没有找到待装配的 Word Star(WS.COM文件)的时候。例如，Word Star不在任何驱动器中的盘上；或者，现行驱动器是A：但WS.COM文件却在驱动器B中盘上。对此，可以把 WS.COM 拷贝到所用的盘上，或者使用装配选择项 B 或 C，从另一个驱动器的盘上来读它。

THAT FILE DOES NOT EXIST

在正常装配期间不会发生这个信息。若在装配选择项B、C、D之下产生这个信息，说明没有找到送入的要装配的Word Star 文件名。通常，这个信息说明你按的名字有错，或者遗漏了驱动器名，或者需要的磁盘不在那个驱动器中。出现这个信息之后，INSTALL 再次询问文件名，让你送入改正过的名字，或者插入正确磁盘之后重新送文件名。

INCORRECTLY FORMED FILENAME

在装配选择项 B、C 和 D 的情况下，这个信息说明你送的文件名不是正确的格式化 CP/M 文件名。出现这个信息之后，INSTALL 将反复提出文件名问题。应送入改正过的文件名。

UNEXPECTED END OF FILE DURING READ

从未经装配的Word Star 那里读到的文件比期待的文件短，表示你可能没有拷贝好 WSU.COM。应力求从配给盘上得到新的拷贝。如果是在装配选择项 B、C、D 之后送的文件名，这个信息也可能说明你命名的文件不是Word Star。

DISK FULL

DISKETTE DIRRECTORY FULL

磁盘（或它的文件目录）已满，你还试图记录所装配的Word Star。这个信息出现之后，WS.COM 是无效的，或者在装配选择项B、C、D之下所写的其他文件也是无效的。对此，可把某些文件拷贝到别的盘上，以配制更多的磁盘空间，再重新装配。

ERROR CLOSING FILE

这个信息不应该出现。它表示系统产生故障，或者是在 INSTALL 正在记录所装配的 Word Star 时换盘而造成。

This version of INSTALL is incompatible with the version of Word Star you are trying to install

如果将某一种版本的 INSTALL 与不同版本的 WSU.COM 一道使用，或者与以前装配的不同版本的Word Star 一道使用，就会发生这段信息。如果某个正在处理尚未装配的 Word Star的文件(WSU.COM，或者操作者在装配选择项 B、C、D 之下送入的文件名)实际上并未包含Word Star，也会发生这种信息。

INSTALL 和Word Star 均在自己信息上加注数字，表示版本号。这些版本号码在屏幕上错误信息下面也都显示出来。

第四节　检查你所装配的 Word Star

不论是在装配结束自动引出Word Star 时，还是在你敲 WS(即保存的你装配 Word Star 的无论什么文件名) 调用Word Star时，Word Star都应清除屏幕，并且显示它的版

本号和一串数字，以及你在装配时选择的终端、打印机、约定和打印机驱动器等等。

几秒钟之后，非文件菜单和你磁盘上的各文件名显示如下：

```
        editing no file
D = create or edit a Document file          H = set Help level
N = create or edit a Non-document file      X = exit to system
M = Merge-print a file                      P = Print a file
F = File directory off(ON)                  Y = delete a file
L = change Logged disk drive                O = copy a file
R = Run a program                           E = rEname a file
DIRECTORT of disk A :
EXAMPLE.TXT INSTALL.COM PIP.COM          STAT.COM
WS.COM          WSU.COM      WSOVLY1.OVR WSMSGS.OVR
```

上图中，符号表示光标位置。如果配备有强光功能，那么被显示的字最亮；如果采用反相的视频光线，那么将形成反相的视频框围绕着菜单和屏幕上部的那条短语。

假如屏幕显示出如下内容：

```
="D = create or edit a Document file H = set Help level = •
N = creat or edit a Non-document file X = exit to system = $  M =
merge-print a file              P = Print a file  = %  F = File
e directory off(ON)          Y = delete a file = &  L = change
Logged disk drive          O = copy a file  =' R = Run a pro
gram                     E = rEname a file = editing no fil
e = + EXAMPLE.TXT INSTALL.COM PIP.COM STAT.COM =, WS.C
OM    WSU.COM WSOVLY1.OVR WSMSGS.OVR = •DIRECTORY of
disk A :
```

并且各行一起滚动，行间还可能出现随机标点，则光标定位控制代码不在工作。出现这种情况应检查你的终端记录和选择开关，重新装配 WordStar 或者按要求改正你的终端。

如果是这样：

```
              editing no file
@@@@
File WSMSGS.OVR not found. Menus &
messages will display as @@@@ only
```

说明配给盘上提供的 WSMSGS.OVR 文件还没有被拷贝到使用盘上（这张盘既不在驱动器 A 中，也不在联机驱动器中）。使用Word Star 时，这个文件总是应该在驱动器 A 中的盘片上，或者是在联机驱动器中的盘片上（或者两者）。敲 X从Word Star 退出，返回操作系统，把 WSMSGS.OVR 拷贝到驱动器A中（或者联机驱动器中）的磁盘上（PIP 命令可用于拷贝大多数 CP/M 操作系统中的文件），再调用你所装配的Word Star。

非文件菜单仍应在屏幕上。现在，敲D。应该出现：

```
D     editing no file
  Use this command to create a new document file,
  or to initiate alteration of an existing document file
    A file name is 1 to 8 letters/digits, a period,
    and an optional 1-3 character type.
    File name may be preceded by disk drive letter A-D
    and colon, otherwise current logged disk is used
  ↑S = delete  character  ↑Y = delete entry  ↑F = file directory
  ↑D = restore character  ↑R = restore entry  ↑U = cancel command
    NAME OF FILE TO EDIT?
DIRECTORY of disk A:
  EXAMPLE.TXT INSTALL.COM PIP.COM          STAT.COM
  WS.COM          WSU.COM       WSOVLY1.OVR WSMSGS.OVR
```

如果非文件菜单部分（屏幕上最后的那些）出现一些空行，则要装配（并非运行）一个擦除结束行代码。

如果出现：

…FATAL ERROR F 46：Overlay File WSOVL Y1.OVR Not Found 说明配给盘上提供的 WSOVLY1·OVR文件没有拷贝到工作盘上(即磁盘既不在驱动器A中，又不在联机驱动器中)。应把 WSOVLY1 拷贝到驱动器A(或者联机驱动器)中磁盘上，再调用你所装配的 WordStar。

如果以上屏幕显示都是正确的，就按 RETURN 键。非文件菜单和目录应再次显示，屏幕的其余部分清除。

送D，再次获得最后的屏幕说明。回答问题“NAME Of FILE TO EDIT？”敲

TESE.DOC

（或者别的不在你磁盘中使用的文件名），按 RETURN 键。短语

NEW FILE

将加到屏幕显示上。几秒钟之后，屏幕显示如下：

```
    A:TEST.DOC PAGE 1 LINE 1 COL 1 INSERT ON
CURSOR:  ↑A = left word   ↑S = left char    ↑D = right char
         ↑F = right word  ↑E = up line      ↑X = down line
SCROLL:  ↑Z = up line     ↑W = down line    ↑C = up screen
         ↑R = down screen
DELETE:  DEL = char left  ↑G = char right    ↑T = word right
         ↑Y = entive line
OTHER:   ↑V = insert off/on  ↑I = tab        RETURN = end para
         ↑U = stop
```

```
                ↑N = insert   RETURN  ↑B = reform to end para
                ↑L = find/replace again
   HELP :       ↑J displays menu of information commands
   PREFIX KEYS ↑Q ↑J ↑K ↑O ↑P display menus of additional com-
    mands
 L----!----!----!----!----!----!----!----!----!----!---------R
```

敲几个字。这几个字应出现在"标尺"(行间夹有虚线和惊叹号)下面的屏幕行上，但不是最亮的。你敲字时光标应向右移动。如果光标不移动，那是光标控制代码的列向没起作用。

按↑J(↑J在屏幕菜单上和在这本手册中意味着敲control-J，就是把键盘上的CTRL键按下不放再敲J)。一、两秒之后，出现不同的菜单。在屏幕的顶行，"标尺"和你敲的那几个字保留不变。如果先前菜单部分保留到新菜单（较窄）的右边，则要装配（并非运行）一个擦除结束行代码。按空格键返回先前的菜单（主菜单）。

现在，敲几个正文文件。此时随便送些无用字符是可以的。假如你敲的字符超出标尺末尾的R还没有敲RETURN的话，Word Star将把你敲的这一行信息复盖掉。这是正常的字复盖操作。

光标可以在送入的正文范围内运动：↑E向上，↑D向右，↑S向左，↑X向下。可试试看。记住↑意味着保持CTRL键压下再敲后面的字母。

敲control-Y，含有光标的正文行随之消失，下面各行上移。这样，就可测试是否装配了"行删除"控制代码。敲control-N，应出现一空行，光标后面的其余各行下移。这样，就可测试是否装配了"行插入"控制代码。

附加强光测试：敲control-K，出现不同的菜单。敲B（或者b，或者↑B），重现最初的菜单，而且在光标位置出现"〈B〉"。如果你装配有强光功能，〈B〉显得最亮。光标右边那一行的字符均右移三个字符位置，让给〈B〉。把光标右移几个字符（↑D），或者下移一、两行（↑X），再敲两次↑K，菜单可能不改变，这取决于你第二次敲↑K有多快。如果没有装配强光功能，〈K〉则出现在光标位置；如果装配了强光功能，从〈B〉的位置到敲↑KK的位置之间的那些字符应当最亮——反相视频显示应不太亮，只比屏幕的其余部分稍亮些。〈B〉应该消失。(在为块移动或块拷贝命令做准备的时候，刚才做的这些叫做定标正文块。)如果〈B〉消失了，但是字符并不最亮，说明装配的强光控制代码无效。如有必要，检查这些代码，重新加以装配；如果仍不能工作，就取消这些代码，以便显示〈B〉和〈K〉。另一种可能，就是采用明/暗光，仅仅按照终端的亮度作对比。这种情况下，字符明暗区别不明显。

现在敲↑K，显示一个新的菜单。敲D（或者d，或者↑D），信息

SAVING FILE A : TEST.DOC

将出现。这样就把你送入的正文存在磁盘上，以致于能够打印它。无论是送正文，还是用Word Star修改正文，都需要存盘，以便永久保留或者打印。存盘之后（几秒时间），非文件菜单再次呈现，就象上面所说的那样。

为了测试打印机的装配，对非文件菜单敲一个P，Word Star将问你打印文件名。送TEST.DOC，或者你命名过的无论什么文件名。按RETLRN键。之后，还将提问其他

一些问题。当打印机准备好时，而且打印机的确转动了，按RETURN键，便出现连续不断地打印（无论使用什么打印功能），然后按任意键。你送入的正文应被打印出来。当一页纸走到头时，打一个“1”字，表示第1页。

现在，可以敲Y键来删除送入的测试文件。回答“NAME OF FILE TO DO DELETE?”（删除文件名?）问题，送TEST.DOC，或者你命名过的无论什么文件名。再按RETURN键。

为了返回操作系统，对非文件菜单敲X，系统提示符A>就出现在屏幕的顶头。

编译校对：蒋庆培、王建新、郭芝传央郭桂银、沈明玉、汪明霓

责任编辑：谢雪峰、王宏成、暴雪琴、吴雪莹、王晓东