微型计算机软件资料汇编

第 二 册

机械工业部计算中心
合肥工业大学微型机应用研究所 编译

GRAPHICS

muLISP

rDBMS

CP/M

机械工业部仪表局情报室
《仪表工业》编辑部

微型计算机软件资料汇编（第二册）

机械工业部计算中心
合肥工业大学微型机应用研究所 编译

*

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

*

1984年12月 北京
代号：8401

内 容 提 要

《微型计算机软件资料汇编》第二册包括两篇文章：

“COBOL—80用户手册”描述了以ANSI 3.23—1974为基础的COBOL—80文本以及在CP/M操作系统下COBOL程序的编译、装入和执行过程，并叙述了实用程序软件MACRO—80宏汇编程序、LINK—80连接装入程序、LIB—80库管理程序和CREF—80相互对照功能的使用。

“FORTRAN—80参考手册”是以ANSI X3.9—1966FORTRAN语言为基础的CP/M版本，对标准文本增加了某些扩充和限制。在附录中，叙述了在CP/M操作系统下，FORTRAN—80的安装和使用。

本资料可作为应用程序设计人员及微型计算机用户的参考手册，也可供高等学校有关专业师生用作学习程序设计语言的参考资料。

微型计算机软件资料汇编

第 二 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

COBOL-80 用户手册

FORTRAN-80 参考手册

COBOL-80 用户手册

第一章　COBOL-80 用户指南

第一节　概　　述

1.1　引言

本章目的在于使读者对于如何在计算机设备上建立和运行 COBOL-80 程序有一个实际的了解。使用 COBOL-80 所有必要步骤——编译、装入、执行等将在下面详细讨论。

本手册中给出的实例和文件都基于COBOL-80的CP/M版本。如果你使用的是另一种操作系统，则其命令和文件名的格式将会稍有差异。如何在你的操作系统上使用 COBOL，请看附录 D中的描述。

1.2　提供的软件磁盘

Microsoft 磁盘包含如下文件：

COBOL 编译程序

COBOL.COM
COBOL1.OVR
COBOL2.OVR
COBOL3.OVR
COBOL4.OVR

运行时系统

COBLIB.REL——运行时库
CRT 驱动程序——名字以CD打头的文件
源—CD——.MAC
目标—CD——.REL，CRTDRV.REL

实用程序软件

L80.COM——连接装入程序
LIB.COM——库管理程序
M80.COM——宏汇编程序
CREF80.COM——汇编相互对照程序

其他文件

SQUARO.COB
CRTEST.COB
SEQCVT.COM
COPCOB.SUB

1.2.1　COBOL 编译程序

编译程序由一个主程序和四个覆盖程序组成，这五部分对应于编译的五个“阶段”。主程序常驻内存并控制每个阶段到下一阶段的转换。主程序的覆盖部分编译标识部（IDEN-

TIFICATION DIVISION) 和环境部 (ENVIRONMENT DIVISION)。覆盖1用于编译数据部 (DATA DIVISION)。覆盖2用于编译过程部 (PROCEDURE DIVISION)。这三部分构成了编译的第一次扫描。它们的功能是建立程序的中间版本，以STEXT. INT为文件名存于当前磁盘中。覆盖3读中间文件并建立目标码。最后，覆盖4分配文件控制块并检测某些错误条件，然后删除中间文件。

1.2.2 运行时系统

运行时库是由一组子例程组成，这组子例程负责解释应用程序经编译生成的目标码。在执行装入步骤时，这些子例程将与你的目标程序一同包括进来（见本章第三节）。并非所有程序都需要全部库例程。装入程序将查找库并自动包括所需要部分，排除不需要部分。提供CRT驱动程序使用户能依照其所具有的CRT终端类型去配置其系统。用户需选择适当的驱动程序（见本章附录A）。选定之后，该驱动程序将与用连接装入程序装入的每一个程序一起自动包括进来。驱动程序提供光标定位及其他功能，用以支持交互的ACCEPT和DISPLAY语句。

1.2.3 实用程序软件

连接装入程序用于连接COBOL目标程序与运行时系统（见本章第三节）。其他实用程序软件也都是为用户使用方便提供的。所有这些程序在“实用程序软件”一章中都有说明。

1.2.4 其他文件

SQUARO.COB是一个计算由用户提供数的平方根的COBOL源程序。它用于检验用户是否具有编译程序及运行时系统的工作版本。

CRTEST.COB是一调试交互CRT驱动程序功能的COBOL源程序（见本章附录A）。

SEQCVT.COM是一特殊的实用程序，它将COBOL文件从行顺序 (LINE SEQUENTIAL) 格式转换成顺序 (SEQUENTIAL) 格式。在3.0版本发表时改变了COBOL-80顺序文件的格式。由早期版本建立的顺序组织的文件格式现在称之为行顺序格式。

COPCOB.SUB是一个命令文件，它将提供的软件磁盘上的文件拷贝到另一磁盘上去，是为了方便用户而提供的。

1.3 启动

首先，用提供的原版磁盘建立工作拷贝，并将原始磁盘保留作为备用盘。这可通过使用COPCOB命令或其他磁盘拷贝功能来完成。

然后，通过编译、装入和执行测试程序SQUARO. COB，检验你的编译程序和运行时系统拷贝。具体步骤可参阅下面1.4中的实例。

最后，如果你打算在COBOL程序中使用交互的ACCEPT和DISPLAY功能，则必须选择CRT驱动程序，并将它配置到运行时系统中。此过程只需做一次，此后，所选择的驱动程序将自动包含到每个目标程序之中。详见本章的附录A。

1.4 程序开发步骤

COBOL程序的准备由以下三个基本步骤组成：

1. 用文本编辑程序建立源文件。
2. 用COBOL编译程序进行编译。
3. 用连接装入程序进行装入。

源程序是由以回车换行为结束的若干行ASCII正文组成的一个文件。你可用EDIT-80或

任何使用7位ASCII字符编码的其他编辑程序去建立源文件。行号可写在每行的1～6列中，可以是8位的ASCII码，编译程序忽略第7列之前除TAB和回车以外的其他字符。编译程序假设的TAB停止是在第7，17，25，33，41，49，57，65和73列。超过第73列的所有字符都被忽略，直至遇到CR为止。如果使用EDIT-80，则自动地在每个插入行的第7列开始打入。

在建立了源程序文件之后，下一步骤是对它进行编译。打入命令行执行COBOL编译程序，并提供源文件名，即可开始编译。在CP/M之下，包含COBOL编译程序的磁盘必须是当前联机磁盘，因为编译程序覆盖总是从当前磁盘上读出。下例是对测试程序SQUARO进行编译的命令，假定驱动器A包含该磁盘的一个拷贝（测试程序包括在提供的磁盘中）。

```
A>COBOL  SQUARO.REL, TTY:=SQUARO.COB
```

该命令将编译SQUARO.COB，将其浮动目标码放入名为SQUARO.REL的文件中，并在终端上打出清单。下列命令行利用了编译假定的缺省文件扩展名，其命令形式较短，作用与上述命令相同：

```
A>COBOL  SQUARO,TTY:=SQUARO
```

所有命令行中最短的形式是利用一个编译开关，强制生成缺省文件名为SQUARO.REL的目标：

```
A>COBOL.TTY:=SQUARO
```

上述三个命令实例都产生完全相同的结果。在第二节中，将详细描述命令行的语法规则。

源程序经过编译之后，在执行前的最后一步就是用连接装入程序L80装入该程序。该步骤将用户的浮动目标程序转换成一绝对地址版本，并将其与COBOL-80运行时系统组合在一起。绝对地址版本建立在内存中，然后可将其保存到磁盘上，也可直接执行。下述命令装入SQUARO执行，并且不保留其绝对地址版本：

```
A>L80  SQUARO/G
```

L80假定被装入的SQUARO文件的扩展名为.REL。一旦SQUARO执行完成，用户就不能在未执行装入命令之前，再次执行该程序，因为它的绝对地址版本未被保留。若将绝对版本保存在一个磁盘文件之中而不直接执行它，可打入：

```
A>L80  SQUARO/N,SQUARO/E
```

然后要执行该程序，只须打入：

```
A>SQUARO
```

由于保留了绝对地址版本，所以可在任何时候执行该程序，而不用再执行装入步骤。将前面二例组合起来，可在保留绝对版本的同时直接执行，打入：

```
A>L80  SQUARO/N,SQUARO/G
```

关于L80命令的详细描述，请参阅本章第三节和第三章“实用程序软件”。

第二节　编译 COBOL 程序

2.1 COBOL-80 命令行语法

COBOL-80编译程序读用户的COBOL源程序文件作为输入，并产生该程序的列表文件和浮动目标版本。命令行调用COBOL编译程序，并告诉它要用的三个文件的名字。该行的语

法要求打入COBOL后跟一空格，再跟一命令串，如下所述。从磁盘上读出COBOL-80，然后检验命令串，如果命令串正确，则开始编译；否则，将显示出信息“? Command Error”后跟一个星号提示符，然后等待打入另一命令串。当编译完成时，COBOL-80 总 是 退出返回到操作系统。

一个 COBOL-80 命令串的格式为：目标文件，列表文件 = 源文件

分隔符为逗号和等号，不允许有空格。格式中使用的术语为：

目标文件

写入目标程序的文件名

列表文件

写入程序清单的文件名

源文件

COBOL程序的源文件名

每个文件都可以是磁盘文件名或系统设备名。文件名的全部描述决定于所用的操作系统。对于 CP/M，文件描述形式为：

设备：文件名.扩展名

此处的分隔符是冒号和句号，术语意为：

设备

系统设备名，可以是磁盘驱动器、终端、行式打印机或操作系统支持的其他设备。若设备是磁盘，则还必须给出文件名，否则，设备名本身就是完整的文件描述。COBOL-80 采用如下符号作为设备名：

TTY：用于控制台终端

LST：用于系统打印机

RDR：用于高速阅读机

文件名

磁盘文件名。若指定文件名而未指定设备，则假定为当前磁盘。

扩展名

给定文件名的扩展名。若未指定，则假定的缺省如下：

.COB　用于源程序文件

.PRN　用于列表文件

.REL　用于目标程序文件

在命令串中，目标文件、列表文件都可被省略。如果列表文件和目标文件都不需要，则 COBOL-80 将检查错误，并在控制台上打出全部错误。若 在 等号左边未打入任何字符，则目标文件将以与源文件相同的文件名和缺省的扩展名被写到同一设备上。

举例：

命令串	效　果
,=PAYROLL	编译 PAYROLL.COB 中的源程序，仅对错误进 行 计数，并在控制台上显示此错误计数。
=PAYROLL	编译 PAYROLL.COB 并将目标 置 于 PAYROLL.REL 中。不生成清单。

,TTY:=PAYROLL	编译PAYROLL.COB中的源程序并在终端上列出程序清单。不生成目标程序。
PAYOBJ,LST:=PAYROLL	编译PAYROLL.COB中的源程序，在打印机上打印程序清单并置目标码于PAYOBJ.REL之中。
PAYOBJ=B:PAYROLL	编译磁盘B中的PAYROLL.COB并置目标码于PAYOBJ.REL之中。不生成清单。
PAYROLL, PAYROLL=PAYROLL	编译PAYROLL.COB，置清单于PAYROLL.PRN之中，并置目标码于PAYROLL.REL之中。

2.2 编译程序开关

可在命令串中附加一个或多个开关对其进行修改，这些开关影响编译过程。要增加开关，可打入一斜杠后跟一字符的开关名。

开关	动　作
R	强迫编译程序生成一目标文件。这个速记符号导致编译程序将目标文件以与源文件相同的文件名和目标文件用的缺省扩展名写到同一磁盘上。
L	强迫编译程序生成一列表文件。同/R一样，它导致编译程序将列表文件以与源文件相同的文件名和列表文件用的缺省扩展名写到同一磁盘上。
P	每个/P分配一个额外的100字节的栈空间为编译程序使用。若在编译期间发生栈溢出错误，则使用/P（见本章2.3节）。否则不需要/P。

使用开关的命令串举例：

命令	等价形式
,=PAYROLL/R	PAYROLL=PAYROLL或=PAYROLL
,=B:PAYROLL/L	,B:PAYROLL=B:PAYROLL
,=B:PAYROLL/R/L	B:PAYROLL, B:PAYROLL=B:PAYROLL
=PAYROLL/L/P	PAYROLL, PAYROLL=PAYROLL/P

2.3 输出清单和错误信息

由COBOL-80输出的列表文件是关于源文件的逐行说明，带有页标题及错误信息。页标题行从页首开始打印，后跟两空行。列出的每一源文件行前面有一连续的4位十进制数字，用于在最后的错误信息中指出错误发生在哪一行，也用于运行时系统发生运行错误后指出是哪一条语句引起的。

在编译过程中可产生两类诊断信息。

低级错误标志是当发生一般语法违例时，直接显示在清单上的源语句行之下的。假设在各个情况下采取如下补救措施并且继续编译。若发生了一个低级错误，则在列表文件的结尾处将产生一高级诊断信息，提供出与该低级错误相关的行号。因此，在结尾给出的错误计数包括这两类错误。

标志	出现标志的原因	编译程序补救措施
"QLIT"?	引用文字的错误 1.零长度	忽略并继续

	2.不适当的延续	假定可接受
	3.过早到达文件结束（在结束限制符之前）	假定为程序结束
LENGTH?	引用文字长度超过120字符，或数值文字超过18位，或‘字’（标识符或名）超过30个字符	超过的字符被忽略
CHRCTR?	非法字符	忽略并继续
PUNCT?	不适当的标点（例如逗点后未跟空格）	假定可接受
BADWARD	当前字不完整，诸如最后为一个连字符，或在一数值文字中有多个十进制小数点	忽略并继续
SEQ?	不适当的顺序号（包括顺序号混乱的情况）	接受并继续
NAME?	不是以字母A-Z打头的名	接受并继续
PIC=X	不适当的PICTURE子句	假定为PIC X
COL·7?	在只允许出现*，—，/，D的源行第七列中出现了不适当的字符	假定第七列为空格
AREA A?	在继续行的A区（8-12列）不是空格	忽略A区中的字符（假定空格）

高级诊断信息包括二或三部分：

1．相关的源行号——四位数字，后面跟一冒号（：）。

2．由编译程序查出的错误的英文注释。如果这些注释正文以/W/开头，则仅仅是一个警告；否则，说明是一个相当严重的错误，将会禁止目标程序的连接和执行。

3．（任选的）列出错误点引用的程序成份，高级诊断信息正文设计成不需要列出‘Messages and error codes’，信息是自我解释的。

不论有无列表设备，或物理上是什么样的列表设备，在编译结束时，总是有一个表示错误或警告总数的信息显示在控制台上。这就允许用户对COBOL程序做简单的修改或重新编译它，在无列表文件的情况下，仍可知道编译程序是否在该程序中遇到了有问题的语句。

必须注意，有两条不常出现的错误信息也在控制台上显示。一个是“? Memory Full”，当编译程序从源程序中取出的全部符号和其他信息无足够存储空间存放时，出现这个信息。它说明被编译的程序太大，必须减小长度或分成若干个独立进行编译的模块。在编译过程中，数据名和过程名的符号表通常占的空间最大。所有的名都需要有与该名的字符数相同的字节数，并且还需要每个数据名约10个字节，每个过程名约2个字节的开销。平均起来，编译时，数据部的每一行需要大约14个字节的内存，过程部的每一行需要大约$3\frac{1}{4}$字节。

另一错误信息，“? Compiler Error”，出现在编译程序发生混乱时，它通常由一或二个问题引起：或是栈溢出，这时可用/P开关解决；或是磁盘上的编译程序或覆盖文件之一被破坏，这时应该试用拷贝副本。如果这些解决办法都不行，则编译时可采取先编译几行源程序，再逐渐增加编译行数，直至错误重新发生为止的办法来确定错误的原因。上述两种错误情况导致编译过程的立即终结。

2.4 COBOL-80使用的文件

COBOL—80使用的文件除源文件、列表文件和目标文件以外，还有如下文件：

首先是STEXT．INT文件，编译程序总是将该文件放在当前磁盘上。它用于收容编译第一次扫描和第二次扫描之间的中间符号文本。该文件被建立、写入，然后关闭、读出，并在退出

编译程序之前被删除。当然，除非编译失败，否则你是不会碰到它的。

另一个文件是由于COBOL程序中 COPY 动词引起的需要被拷贝的文件（请看COBOL-80参考手册中关于COPY的讨论）。记住，被拷贝的文件中不能有COPY语句，并且该行的COPY语句之后的其余部分被忽略。

最后，COBOL程序使用分段特性，使装入程序为程序的每个独立段建立一个文件。该文件名本身是在标识部中定义的PROGRAM-ID。扩展名是.Vnn，其中nn是一个2位十进制数，它等于段号减49。

第三节　装入COBOL程序

Microsoft连接装入程序用于将已编译的用户程序的浮动目标版本转换成可执行的绝对版本。它自动地组合目标程序和COBOL运行时系统的所需部分，装入程序也用于将一个或多个子程序与主程序连接在一起。这些子程序可以分别指定或从库中提取出来，也可以用COBOL，FORTRAN-80或MACRO-80汇编语句编写。

3.1 LINK-80命令行语法

用于LINK-80的完整语法规则在第三章“实用程序软件”第四节中给出。不过，在装入COBOL程序时，有些功能是没用的。本节概述 COBOL 程序所用的装入程序的使用。

可以用两种方法调用LINK-80：其一，打入 L80 后跟一回车，并在显示出星号提示符后，打入命令串；其二，打入L80后跟一空格，再在同一行上跟一命令串。

命令串是由逗号分隔的一个文件名表。装入程序将每个指定的文件名装入内存，并置于下一个可用的存储地址上。命令串中使用开关指定装入程序要执行的功能。命令串也可分解成许多小串并分成不同行输入。装入程序将用星号作为提示符，并等待再输入命令串，直到处理完一个带G或E开关的命令串为止。同时，装入程序退出返回操作系统。文件名的指定与编译程序中的方式相同。但由装入程序读入的文件缺省扩展名总是 .REL。这些文件都应该是浮动目标格式的。所以它们必须是经过编译（或汇编）的。

在装入COBOL程序时使用最多的开关为：

开关	作　用
文件名/N	在完成装入处理时，指导装入程序将可执行程序以名〈文件名〉保留到磁盘上。
/E	指导装入程序完成装入过程并退出至操作系统。装入程序查找 COBLIB.REL 和 CRTDRV.REL，以解决未定义的全局符号。最后一步是将可执行程序保存在磁盘上，只要开关/N已经指定。
/G	指导装入程序完成装入处理并开始程序的执行。同有/E 开关时一样，查找COBOL运行时库，并且，若指定了/N，则保存可执行程序。

在装入COBOL程序时，偶尔可能用到的开关还有：

开关	作　用
/R	立即将装入程序复位为初始状态。其效果如同装入程序异常终止并重新从磁盘上装入一样。
文件名/S	指导装入程序查找〈文件名〉以解决未定义的全局符号。此命令用于有选择地

从用户建立的库中装入被调用的子程序。

/M　　打印全局符号和它们的值的映象，未定义的全局符号出现时，在名后面带一个星号。

/U　　打印所有未定义的全局符号表。

实例：

命令串

MYPROG, SVPROG/N/E

装入MYPROG.REL，保留其绝对版本于SVPROG.COM中，并退出至操作系统。

MYPROG/G

装入MYPROG.REL，并开始执行，不保留绝对版本。

MYPROG, SUBPR1, B:SUBPR2, MYPROG/N/E

装入 MYPROG . REL , SUBPR1 . REL 和 B : SUBPR2. REL, 将绝对版本保留于MYPROG.COM中，并退出到操作系统。

MYPROG/N, MYPROG, MYLIB/S/E

装入MYPROG.REL，查找MYLIB.REL，找出被“CALL”语句引用的子程序，保留绝对版本于MYPROG.COM中，并退出到操作系统。

3.2 子程序

如果用户已将程序组成为一个主模块和一个或多个子程序模块，则装入程序可将它们组合成一个可执行的程序。在装入前，编译（或汇编）所有的模块，以便获得每一个模块的浮动目标版本。然后执行装入程序，并在命令串中指定要装入的每个模块名。可以按任意顺序指定模块。例如，若有一个已编译的主程序文件MAINPG.REL和两个子程序文件SUBPR1.REL和SUBPR2.REL，则可用下面的任一个装入命令装入可执行程序并保存之。

```
1. L80  MAINPG/N, MAINPG, SUBPR1, SUBPR2/E
2. L80
* MAINPG/N, MAINPG, SUBPR1, SUBPR2
* /E
3. L80  SUBPR1, SUBPR2
* MAINPG/N
* MAINPG/E
```

3.3 功能库

Microsoft 库管理程序LIB-80（仅对CP/M版本）允许将任意数量的子程序集合成一个文件（一个库），它可被装入程序检索。例如，如果有六个由不同主程序使用的子程序，名字为SUBPR1.REL到SUBPR6.REL，则可用下面命令将其放在一个库中：

```
LIB
* USRLIB = SUBPR1, SUBPR 2 , SUBPR3,
          SUBRR 4 , SUBPR 5 , SUBPR 6 /E
```

此命令将建立一个名为USRLIB.REL的库文件（关于 LIB-80 的详细描述，请看“实用程序软件”一章的第三节）。然后，若有一主程序MAINPG调用SUBPR 2 和SUBPR 6，则装入命令：

L80 MAINPG/N, MAINPG, USRLIB/S/E

将装入MAINPG并在USRLIB中查找SUBPR 2和SUBPR 6。

在建立库时，要保证全部子程序的标识是唯一的。由于COBLIB中的所有COBOL运行时例程名开头有美元符（$），所以在子程序命名时应避免使用美元符。

第四节 执行COBOL程序

可用下列三种方式执行一个COBOL程序。第一种方式是在装入程序命令串中使用G开关，如本章3.1节所述。第二种方式是简单地打入可执行程序文件名，即在装入程序命令串中使用开关N保存的名。最后一种方式，可使用CALL或CHAIN语句从另一个COBOL程序中直接执行一个程序。（请参阅第二章“COBOL-80参考手册”第五节中关于对程序CALL和CHAIN的解释）。

4.1 运行时系统

由编译程序生成的用户程序的浮动目标版本不是8080或Z80机器码，而是专为COBOL指令设计的特定目标语句的格式。COBOL-80运行时系统通过检测每个目标语言指令和执行所需要的功能来执行用户程序，包括处理CRT、打印机和磁盘文件输入、输出所需要的功能。

运行时系统包括被收集到COBLIB.REL库中的机器语言子程序和名为CRTDRV.REL的CRT驱动程序。在装入COBOL程序时，装入程序自动检索COBLIB，找到并装入执行用户源程序指令所需要的例程。需要的例程数量取决于用户在主程序和子程序中所用的COBOL语句特性的数量。若源程序中包括DLSPLAY或ACCEPT语句，则装入程序自动地检索文件CRTDRV.REL，以包括终端有关功能。

在运行时，COBOL程序需要的内存数量等于数据部中定义的数据项所需的存储数量，加上每个文件约500字节，对过程部中每一行加上约12个字节，加上运行时系统所用的最多为24K字节。

4.2 打印文件处理

应该将打印文件简单地看成是一个送到打印机上的字符流。记录应简单地定义为出现在打印机上的字段。记录中不需要额外的字符用于走纸（托架）控制。行与行间的回车、换行和格式馈给在需要时被送到打印机。但要注意，在打印行结束处的空字符(空格)应被截断，以加快打印速度。

对于FD中的一个打印机文件，不应给出“VALUE OF”子句，但须指定“LABEL RECORD IS OMITTED”。对于打印机文件不能使用BLOCK子句。

4.3 磁盘文件处理

磁盘文件必须已说明为“LABEL RECORD IS STANDARD”，并有包含文件标识(ID)的“VALUE OF”子句。文件ID的格式在“实用程序软件”一章中描述。BLOCK子句用于语法检查，对任何类型的文件都没有作用。

通常，顺序（SEQUENTIAL）组织文件的格式是有2个字节的记录长度计数，后跟实际的记录，用于文件中现存的全部记录。行顺序（LINE SEQUENTIAL）组织中的记录后跟一个回车/换行限制符，用于存在于该文件中的所有记录。对于两种组织，最后一个物理

块的剩余空间都用Control-Z字符填充，标志文件结束。为最大限度地使用磁盘空间，需将记录压缩到一起，使记录之间没有不必要的字节。

相对（RELATIVE）文件的格式总是具有定长记录，且记录长度为文件定义的最大记录长度，不需要有限制符。删除的记录用十六进制值'00'填充。另外，在文件的开始要保留6个字节，以包含系统程序加工信息。

索引（INDEXED）文件的格式是很复杂的。它是关键字、数据、行指针、删除指针和不规划功能指针的复杂混合体。COBOL程序员不一定要了解这些信息，在这里就不作说明了。

4.4 CRT处理

4.4.1 终端输出

所有到终端的输出都是由DISPLAY语句执行的。字符由DISPLAY运行时模块或由CRT驱动程序送出。若对任何显示项未指定光标位置，则在最后一个显示项之后送出一个回车和换行。否则，DISPLAY模块认为没有走纸（托架）控制。

4.4.2 键盘输入

所有的键盘输入都是由ACCEPT语句执行的。可使用下述两种输入方法中的一种，具体取决于所执行的ACCEPT语句的类型。

对于格式2的ACCEPT语句，使用操作系统功能作字符返回和输入编辑，打入整行输入，以一回车结束。对于这种类型，定义在CRT驱动程序中的字符码无效。

对于格式3或格式4的ACCEPT语句，打入的每个字符都由运行时的ACCEPT模块通过使用一个操作系统调用直接读入。ACCEPT模块执行所有必要的字符返回以及输入编辑功能。编辑控制字符、功能键和终止符键都定义在CRT驱动程序之中（见本章附录A）。

4.5 运行时错误

有些程序设计的错误不能由编译程序发现，但在执行时却引起程序提前终止。每个这样的错误都产生四行提示，显示在控制台上。

```
** RUN-TIME ERR:
原因（如下表所列）
行号
程序标识
```

可能引起终止的原因及其附加解释如下：

REDUNDANT OPEN

企图打开一个已经打开的文件。

DATA UNAVAILABLE

企图引用一个未打开的文件或已到达文件结束条件的文件记录中的数据。

SUBSCRIPT FAULT

下标值非法（通常小于1）。这适用于索引引用，如I+2，其值决不能小于1。

INPUT/OUTPUT

不可恢复的I/O错误，并且在用户的COBOL程序中未使用AT END子句、INVALID KEY子句、FILE STATUS项和DECLARATIVE过程等，给出此时应采取的动作。

NON-NUMERIC DATA

无论何时，只要一个数字项的内容与给定的PICTURE不匹配，就可能产生这种错误。如果是由于使用NUMERIC测试产生的错误，则应该检查输入数据。

PERFORM OVERLAP

PERFORM语句使用顺序非法。例如，在执行A段且尚未退出之前，开始另一个PERFORM A。

CALL PARAMETERS

调用程序和被调用的子程序参数数目不一致。

ILLEGAL READ

企图读一个不是以输入或I-O方式打开的文件。

ILLEGAL WRITE

企图写到一个未以输出方式打开的顺序存取文件，或未以输出或I-O方式打开的随机或动态存取文件。

ILLEGAL REWRITE

企图将一记录重写到一个未以I-O方式打开的文件之中。

REWRITE, NO READ

在最后一个操作不是成功的READ操作时，企图重写一个顺序存取文件的记录。

REDUNDANT CLOSE

企图关闭一个未打开的文件。

OBJ.CODE ERROR

遇到了一个未定义的目标程序指令。仅当内存中或磁盘文件中程序的绝对版本已被破坏时，才会发生。

FEATURE UNIMPL.

遇到了一个调用未实现特性的目标程序指令。仅当目标程序已被破坏时，才会发生。

GO TO. (NOT SET)

企图执行一个仅包含空GO语句的未赋初值的可变段。

FILE LOCKED

企图在前面执行过CLOSE WITH LOCK之后打开文件。

READ BEYOND EOF

在已遇到文件结束之后，企图读下一个记录。

DELETE, NO READ

在最后一个操作不是一个成功的READ时，企图删除顺序存取文件的一个记录。

ILLEGAL DELETE

相关文件未以I-O方式打开。

ILLEGAL START

文件未以输入或I-O方式打开。

NO CRT DRIVER

正在执行一个用光标定位的ACCEPT或DISPLAY语句，但未选择CRT驱动程序（见本章附录A）

SEG nn LOAD ERR

在试图装入一个分段程序的一段时，产生了不可恢复的读错误。两个数字位nn是十六进制数，为段号减49，并与磁盘上的文件扩展名（.Vnn）相匹配。

在程序链的情况下，错误信息可能由CHAIN处理模块产生。这时，错误格式为“**CHAIN：问题”，同时也导致程序的终结。请看本章附录B中的CHAIN错误信息表。

附录A 配置CRT

A.1 一般指令

为使用交互的ACCEPT和DISPLAY功能，COBOL-80需要一个终端驱动程序模块以提供基本的终端相关功能。当程序由L80连接时，系统可望以CRTDRV.REL的名字找到这一模块。

版本4的每个COBOL磁盘都提供有模块CRTDRV。这是一个缺省的伪驱动程序，它将使程序能成功地连接，并提供对ANSI标准ACCEPT和DISPLAY语句的支持。那些在ACCEPT或DISPLAY语句中使用光标定位，并用缺省驱动程序连接的程序将不能成功运行；它们将显示“NO CRT DRIVER”运行时错误信息，并异常终止。

在连接由版本4编译的任何COBOL程序之前，CRTDRV模块应替换成为与所用终端类型相适应的驱动程序。这只需将适当的驱动程序拷贝到CRTDRV.REL上。Microsoft已为各种普遍使用的终端提供了驱动程序，若这些驱动程序中无合适的，用户可以自己编写；请看本节A.3“编写CRT驱动程序”。

Microsoft提供的CRT驱动程序模块是以字母CD（CRT Driver）开头的浮动目标文件。它包含用于每个驱动程序的MACRO-80源码。任何驱动程序都支持一种类型以上的终端，只要该终端有兼容的控制序列。若你的终端没列在下面，请查找本节A.2，将你的终端功能码与提供的终端驱动程序代码相比较。如果你的终端代码与提供的某驱动程序相匹配，则可使用它。

终端及相应的驱动程序有：

1. ANSI 标准终端	CDANSI
2. Lear-Siegler ADM3-A	CDADM3
3. Beehive 100, 150	CDBEE
4. Microbee 2	CDBEE
5. Cromemco 3130, 3102	CDBEE
6. SOROC IQ	CDSROC
7. Hazeltine 1500	CDHZ15
8. Heath WH19	CDWH19
9. DEC VT52	CDWH19
10. ADDS Regent Terminals*	CDADDS
11. Perkin-Elmer	CDPERK
12. Zentec Zephr	CDZEPH
13. Intertec Superbrain	CDISB

14. IMSAI VIO　　CDADM3

• 支持 ADDS Regent40，60，100和200终端。Regent20 和25终端不支持高亮度视频码，但如果去掉该代码，则可使用CDADDS驱动程序。

A.2 终端图表

下面描述软件磁盘中提供的驱动程序所支持的终端特性，对于支持的每个终端都有一页描述。

每页的第一部分定义 COBOL-80 承认的键（用于执行ACCEPT 功能）。列在标题“换码”之下的值是一整数，若该键导致格式 3 或格式 4 ACCEPT 语句的终结，则此值可用于使用格式 1 ACCEPT… FROM ESCAPE KEY；列在标题“输入码”之下的值是在打入该键时由终端生成的十六进制码；“键标号”之下的项给出了键名（与键盘上所示相同）。

每页的第二部分显示出从 COBOL-80 送给终端的代码序列（用于执行 DISPLAY 和 ACCEPT功能）。表中的代码用空格分隔开，但空格不作为代码序列的一部分送给终端。每两位数字代表一个绝对的十六进制值。所有其余代码都描述标准ASCII字符编码，但一些缩写码有如下解释：

R 1　十进制行号加十进制31。

R 2　行号转换成 2 个ASCII数字，首先送高位。

C 1　二进制列号加十进制31。

C 2　列号转换成 2 个ASCII数字，首先送高位。

C 3　若列号小于32，则列号加十进制95，否则用列号减 1 。

N/A　此终端不能使用的功能。

E 1　若光标是在起始位置，则使用一个清除屏幕码（十六进制 1 A）；否则，送足够的空格以清空屏幕的剩余部分，并将光标移至其原始位置。

E 2　送足够的空格清空屏幕剩余部分，并将光标移至原始位置。

N 1　10个空（二进制零）字符。

A.3 编写CRT驱动程序

CRT驱动程序应该用汇编程序写成，并用 Microsoft 汇编程序M80进行汇编。它包含14个入口点，这些入口点须用M80 ENTRY语句说明作为全局标号。各驱动程序的源码都在提供的软件磁盘上（文件名为 CD— · MAC），可作为实例供用户参考。下面作一些解释：

在写好CRT 驱动程序后，就可使用提供在软件磁盘上的程序CRTEST测试全部功能和键码。首先对其进行编译，然后用L80与你的CRT 驱动程序连接并按照它提供的指令执行。

有五个入口点包含了描述终端和键盘的数据。$CRLEN 是一个字节，它包含终端上的行号，$CRWID 包含列号。$CLIST、$TLIST和$FLIST 是定义调用ACCEPT功能的键盘码的字节序列。注意，这些代码并不送到终端，它们仅说明那些应该被 ACCEPT模块承认的键。所有这些码都应该是唯一的。

$CLIST定义编辑键，它们必须按如下顺序指定：

1. 行删除（域删除）
2. 字符删除
3. 进格（光标前移）

CDADDS

ADDS Regent终端
24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		06	CONTROL-F
4．退格		08	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	CONTROL-I
b．回车		0D	NEW LINE
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC Y R_1 C_1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删除到屏幕结束	ESC k
F．删除到行结束	ESC k
G．振铃	07
H．置高亮度方式	ESC 0 P
I．重置高亮度方式	ESC 0 @

CDADM3

Lear-Siegler ADM-3A

24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		0C	CONTROL-L
4．退格		0B	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	CONTROL-I
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC = R1 C1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删除到屏幕结束	E1
F．删除到行结束	E2
G．振铃	07
H．置高亮度方式	N/A
I．重置高亮度方式	N/A

CDANSI

ANSI标准终端
24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL, RUB
3．进格		06	CONTROL-F
4．退格		0B	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB,CONTROL-I
b．回车		0D	RETURN,ENTER
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC[R2;C2f
B．光标回退	08
C．光标开	ESC[>5l
D．光标关	ESC[>5h
E．删除到屏幕结束	ESC[0J
F．删除到行结束	ESC[0K
G．振铃	07
H．置高亮度方式	ESC[7m
I．重置高亮度方式	ESC[0m

CDBEE

Beehive终端

24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		06	CONTROL-F
4．退格		08	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB, CONTROL-I
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC F R_1 C_1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删除到屏幕结束	ESC J N_1
F．删除到行结束	ESC K
G．振铃	07
H．置高亮度方式	ESC 1
I．重置高亮度方式	ESC m

Hazeltine 1500系列终端
24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		5D	]
4．退格		08	BACK SPACE
5．加号		2B	+
6．减号		2D	-

B．终结符键	换码	输入码	键标号
1．返回标记	99	5C	\
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB
b．回车		0D	RETURN
c．换行		0A	LINE FEED

C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	~DC1 C3 R1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删除到屏幕结束	~CAN
F．删除到行结束	~SI
G．振铃	07
H．置高亮度方式	~US
I．重置高亮度方式	~EM

CDISB

Intertec Superbrain

24行80列

I．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		18	CONTROL-X
2．字符删除		7F	DEL
3．进格		06	CONTROL-F
4．退格		08	BACK SPACE
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	04	CONTROL-D

Ⅱ．输出功能	代码序列
	ESC Y R1 C1
A．置光标位置	
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删除到屏幕结束	ESC～k
F．删除到行结束	ESC～k
G．振铃	07
H．置高亮度方式	N/A
I．重置高亮度方式	N/A

CDPERK

Perkin-Elmer终端
24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		06	CONTROL-F
4．退格		08	BACK SPACE
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC X R_1 ESC Y C_1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删至屏幕结束	ESC J
F．删至行结束	ESC I
G．振铃	07
H．置高亮度方式	N/A
I．重置高亮度方式	N/A

CDSROC

SOROC IQ终端
24行80列

Ⅰ．键盘输入	换码	输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		0C	CONTROL-L
5．退格		08	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	CONTROL-I
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC = R1 C1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
F．删至屏幕结束	ESC Y
F．删至行结束	ESC T
G．振铃	07
H．置高亮度方式	N/A
I．重置高亮度方式	N/A

CDWH19

Heath WH19/DEC VT52

24行80列

Ⅰ. 键盘输入		输入码	键标号
A. 编辑键			
1. 行删除/域删除		15	CONTROL-U
2. 字符删除		7F	DELETE
3. 进格		06	CONTROL-F
4. 退格		08	BACK SPACE
5. 加号		2B	+
6. 减号		2D	-
B. 终结符键	换码	输入码	键标号
1. 返回标记	99	02	CONTROL-B
2. 换码	01	1B	ESC
3. 域终结符	00		
a. 标记		09	TAB,CONTROL-I
b. 回车		0D	RETURN
c. 换行		0A	LINE FEED
C. 功能键	换码	输入码	键标号
1.	02	01	CONTROL-A
2.	03	03	CONTROL-C
3.	04	18	CONTROL-X

Ⅱ. 输出功能	代码序列
A. 置光标位置	ESC Y R1 C1
B. 光标回退	08
C. 光标开	ESC Y 5
D. 光标关	ESC X 5
E. 删至屏幕结束	ESC J
F. 删至行结束	ESC K
G. 振铃	07
H. 置高亮度方式	ESC P
I. 重置高亮度方式	ESC Q

CDZEPH

Zentec Zephr

24行80列

Ⅰ．键盘输入		输入码	键标号
A．编辑键			
1．行删除/域删除		15	CONTROL-U
2．字符删除		7F	DEL
3．进格		06	CONTROL-F
4．退格		08	CONTROL-H
5．加号		2B	+
6．减号		2D	-
B．终结符键	换码	输入码	键标号
1．返回标记	99	02	CONTROL-B
2．换码	01	1B	ESC
3．域终结符	00		
a．标记		09	TAB,CONTROL-I
b．回车		0D	RETURN
c．换行		0A	LINE FEED
C．功能键	换码	输入码	键标号
1．	02	01	CONTROL-A
2．	03	03	CONTROL-C
3．	04	18	CONTROL-X

Ⅱ．输出功能	代码序列
A．置光标位置	ESC 0 R1 C1
B．光标回退	08
C．光标开	N/A
D．光标关	N/A
E．删至屏幕结束	ESC Y N1
F．删至行结束	ESC T
G．振铃	07
H．置高亮度方式	ESC G 4
I．重置高亮度方式	ESC G 0

4．退格（光标后移）

5．加号

6．减号

此表以一含零字节终结。

$FLIST定义结束格式3或格式4 ACCEPT语句的功能键。$FLIST中代码放置的顺序决定可用于ACCEPT…FROM ESCAPE KEY语句的换码键(ESCAPE KEY)值。第一个键生成值02，第2个键生成值03，如此下去，直到最大值为39。此表以一含零字节结束。

$TLIST定义几个键，所有这些键结束格式3 ACCEPT语句。表中第一个键必须是返回标记键。若用在格式4 ACCEPT中，则此键导致当前域的终结，且将光标移至前一个输入域。若用在格式3 ACCEPT中，则返回标记键结束该ACCEPT语句，并置换码值为99。表中下一个键是换码键。此键结束格式3或格式4 ACCEPT语句，并置换码值为01。它还导致程序去执行格式4 ACCEPT语句的一个ON ESCAPE子句。最后，有一个以零字节为终结的正常域终结符键表。此表中的任何一个键均可结束当前输入域，并置换码值为00。该域的终结结束格式3 ACCEPT，并将光标移到格式4 ACCEPT的下一个域。若光标是在最后一个输入域中，则整个ACCEPT语句结束。

剩下的九个入口点是通过送代码到终端执行终端功能的子例程。每个码都通过用A寄存器的值调用外部例程$OUTCH送出。$OUTCH保护寄存器HL和DE中的值。

$SETCR将光标移到屏幕上的一个指定位置上。在入口上，寄存器H包含指定的行号，L包含列号。注意：COBOL认为屏幕最高一行为第一行，最左一列为第一列。

$CURBK将光标向左移一位，而不破坏该处已显示的字符。在入口上，寄存器HL包含顺序格式中当前光标位置。大多数终端通过ASCII退格码去执行此功能。

$ALARM使终端音响器或铃发声。大多数终端都通过ASCII铃码去执行此功能。

$CUROF和$CURON指示终端消去或建立光标的显示。许多终端未提供此功能，但对于这些终端可在驱动程序中加一条简单的RET指令。

$ERASE消除屏幕上从当前光标位置到屏幕结束的所有部分。光标必须在其起始位的左边。在入口点，寄存器HL包含顺序格式中的当前光标位置。某些终端如ADM-3A，不提供换码序列去执行此功能。在驱动程序实例CDADM3中提供一个例程，送足够空格去清除屏幕，并将光标返回到其起始位置。该例程可用于本身没有提供删除功能的终端。

$EOL消除屏幕上从光标当前位置到该行结束之前的部分。在入口点，寄存器H包含当前行号，L包含当前列号。

$HILIT使终端成反视频方式（或者当反视频方式不可用时，为某种高亮度方式）。

$LOLIT使终端回到正常方式（取消$HILIT的影响）。

附录B　程序间通信

本节描述通过CALL USING语句在主程序和子程序之间，或通过CHAIN USING语句在两个主程序之间传递的参数格式。若两程序都是用COBOL编写的，则参数连接完全由COBOL-80运行时系统处理。若被调用或被链接的程序是用汇编语言或FORTRAN语

言写的，参见本节B.1和B.2。

B.1 子程序调用过程

一个COBOL程序可以调用 COBOL 子程序或者调用 FORTRAN 及汇编语言子例程。但目前还不能用一个 FORTRAN 或汇编语言程序去调用 COBOL 子例程。下面描述的调用序列与Microsoft FORTRAN-80调用FORTRAN 或汇编语言子例程时相同。

COBOL 运行时系统借助于机器语言 CALL 指令使执行转向子例程。该子例程应该通过正常的汇编语言或 FORTRAN 语言的返回指令返回。

参数通过引用，亦即通过传送参数的地址来传递。传送这些地址的方法依赖于参数的个数。若参数个数小于等于 3，则它们被传送到寄存器中：

参数 1 在 HL 中

参数 2 在 DE 中

参数 3 在 BC 中

若参数个数大于 3，则 1 和 2 仍传送到HL和DE中，但 BC 指向内存中保存参数地址表的连续数据块。

子例程可使用的参数数量只限于被传递的参数，调用程序负责这些参数的传递。编译程序和运行时系统都不检查参数数目是否正确。调用程序传送的变量长度和类型对于被调用子例程是否可接受，也可完全由用户决定。注意：字母数字型数据(在 COBOL 和 FORTRAN 中）是唯一以相同格式存放的数据类型。任何数字型数据都是不可交换的。

COBOL 程序用的栈空间包含在程序范围之内，因此使用栈的汇编语言程序决不能出现栈上溢出或栈下溢出。保证安全的最常用的方法是在进入例程时保留COBOL栈指针，并置该栈指针于另一栈区。然后在返回主程序前，汇编语言例程须恢复保留的COBOL 栈指针。

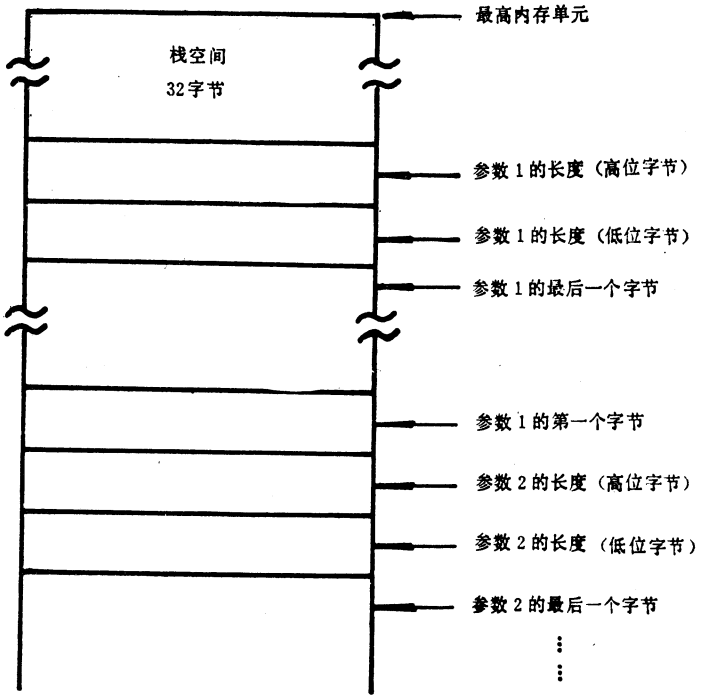


图 B·1

调用子程序，可在COBOL的CALL语句中使用该子程序名。若该子程序是汇编语言或FORTRAN语言程序，则子程序名由ENTRY、SUBROUTINE或FUNCTION语句定义。COBOL子程序名与PROGRAM-ID段中所给定的相同。然后，使用LINK-80将子程序连接至主程序上，如本章3.2节所述。

B.2 CHAIN参数

使用CHAIN USING语句在程序之间传递的参数存放于可用的最高内存地址中，内存布局如下：从可用的最高地址开始一直存放至零单元。首先，保留32个字节作为栈空间，然后为USING表中第一个参数，前面是参数的长度（以字节为单位），参数长度存于两个字节中，首先是高位字节。参数本身作为一字节串，以它们存于数据部中的相同顺序存放，开始地址为其长度地址减长度本身。USING表中每个参数依次排列在后面，每个参数的前面都有该参数的长度。被链接的程序所要接收的参数数量和格式必须与被传递的参数相同，因为编译程序和运行时系统不作任何检查。

B.3 CHAIN错误信息

在CHAIN处理过程中，用于报告运行时错误的正常机构可能已被新程序所覆盖。因此，CHAIN处理程序生成自己的错误信息，格式为“**CHAIN：问题”。下面是可能的“问题”表和引起问题的原因：

Bad file name

被装入的文件名语法无效

File not found

指定的文件在磁盘上未找到

Out of memory

没有足够内存空间可用于装入新程序。对于较大的链接和被链接程序，加上全部CHAIN参数，加上程序装入程序用的256个字节，必须有足够的内存空间才行。

附录C 用户化

本节旨在为善于使用调试程序和（或）汇编语言并希望改变COBOL-80某些内部参数的用户提供参考。

C.1 源程序标记停

若在COBOL源程序中使用标记符（十六进制09），则编译程序将它们转换为空格以按照其内部TAB表中的定义，到达下一个标记停。系统提供的TAB表在下述各列位置上定义了九个停止点（从第一列算起）：

7，17，25，33，41，49，57，65和73

用户可以修改该表改变标记位，该表地址在从COBOL·COM开始的七个字节处。对于每个标记停，在表中都有一个字节。用户可以提供所需的任何值，只要提供的数按顺序并仍有九个定义的停止点。

C.2 编译程序清单页长度

在编译程序中有一个字节，定义清单页长度为55（十六进制37）行。它的位置在从COBOL·COM开始后的第六个字节，并可改为1到255中的任何值。

C.3 运行时日期，时间，行号

对于所有没有提供日期或时间调用的操作系统，COBOL-80使用编译程序为格式1 ACCEPT语句发布日期。对于单用户系统，COBOL总用'00'作为行号。若你有一多用户系统或选用系统时钟（或用一些其他的固定日期和时间），则可替换掉执行该功能的运行时系统。为此，需根据下面给定的指令写一个汇编语言模块，用MACRO-80进行汇编，并用库管理程序把它放到COBLIB.REL中。假设将该模块命名为ACPDAT.MAC，把它放到库中的LIB-80命令为：

```
LIB
*NEWLIB=COBLIB<..ACPDAT—1>,ACPDAT
*COBLIB<ACPDAT+1..>/E
```

首先建立NEWLIB.REL，然后可保存COBLIB·REL并将NEWLIB.REL改名为：COBLIB.REL。

ACPDAT模块

入口点：$ACPDT

外部引用：$EVAL，$GETOP，$FLAGS，$ESKEY，$MOVE

此模块处理对COBOL格式1 ACCEPT源语句的运行时支持：

	DAY
	DATE
ACCEPT 标识符 FROM	TIME
	ESCAPE KEY
	LINE NUMBER

要改变ACPDT模块，可在下面给出的模块结构中修改ACLINE例程和增加ACTIME，ACDAY和ACDATE。每一个这样的例程都与HL寄存器中目标存储区的地址一同进入。每一个例程都必须通过执行一条JMP $GETOP指令退出。各例程要求如下：

1. ACTIME——送一个表示时间（形式为HHMMSSFF）的ASCII字符串到目标区域中。

2. ACDAY——送一个表示儒略历日期（形式为YYJJJ）的ASCII字符串到目标区域中。

3. ACDATE——送一个表示日期(形式为YYMMDD)的ASCII字符串到目标区域。

4. ACLINE——送二个表示行（CRT）号的ASCII数字到目标区域中。

一个外部的传送例程可用于从一个地址传送一个数据串到另一地址。其使用如下：

```
EXT  $MOVE
     HL=源字符串地址
     DE=目标域地址
     BC=字符串的长度（以字节为单位）
CALL  $MOVE
     HL=源以外的第一字节地址
     DE=目标以外的第一字节地址
     BC=0
```

ACPDAT 模块结构

```
   TITLE ACPDAT-ACCEPT DAY/DATE/TIME/ESC KEY/LINE NUM
   ENTRY $ACPDT
   EXT $EVAL, $GETOP, $FLAGS, $ESKEY
$ACPDT: POD   H
        INX   H
        MOV   A,M
        INX   H
        ANI   7
        STA   $FLAGS      ;SAVE ACCEPT OPTION
        CALL  $EVAL       ;GET TARGET ADDRESS
        LDA   $FLAGS
        CPI   2           ;WHICH OPTION?
        JM    ACDATE      ;DATE
        JZ    ACDAY       ;DAY
        CPI   4
        JC    ACTIME      ;TIME
        JZ    ACLINE      ;LINE NUMBER
ACESC:                    ;ESCAPT KEY CODE FROM ACCEPT
        XCHG
        LHLD  $ESKEY
        XCHG
ACESC1: MOV   M,D
        INX   H
        MOV   M,E
        JMP   $GETOP
ACLINE:                   ;LINE (CRT) NUMBER-ALWAYS'00
        LXI   D,3030H
        JMP   ACESC1
ACTIME:                   ;TIME: HHMMSSFF
        .
        .
        .
        JMP   $GETOP
ACDAY:                    ;DAY: YYJJJ
        .
        .
        .
        JMP   $GETOP
ACDATE:                   ;DATE: YYMMDD
        .
        .
        .
        JMP   $GETOP
        END
```

附录D　非CP/M操作系统下使用COBOL-80

本资料中其他部分列举的许多实例和指令都涉及到CP/M操作系统特定的过程和文件名。命令串的语法对于所有操作系统都是相同的；不过，文件指定和开关分隔符可以不同。如果你使用其他操作系统，则应该注意下面各节所述的区别。

D.1 TRSDOS ModelⅡ

D.1.1 文件名描述

COBOL-80和LINK-80的文件描述形式与TRS-80用户手册中描述的完全相同，即：

文件名/扩展名.口令：磁盘名

分隔符为斜线、句号和冒号。

D.1.2 提供的软件磁盘

提供的软件磁盘上的文件名不同于CP/M名，并遵循TRSDOS文件命名约定。该磁盘包含与CP/M磁盘相同的文件，但有如下例外：

* 包含用于ModelⅡ终端的CRT驱动程序。

* 在ModelⅡ上不可用的一些实用程序不包括在内，它们是LIB-80和SEQCVT

D.1.3 命令行语法

用于COBOL-80和LINK-80的命令串在TRSDOS下和在CP/M下形式相同。不过，用连字符作为分隔符，而不用斜线（因为斜线可用于TRSDOS文件名中），并且控制台和打印机的设备符号名分别为：TT和：LP。下面的例子给出如何使用TRSDOS法语编译、装入和执行测试程序SQUARO/COB。

```
TRSDOS  READY
COBOL, :TT=SQUARO-R
L80  SQUARO-N, SQUARO-E
SQUARO
```

由COBOL-80假定的缺省文件扩展名为：

/COB　用于源程序文件

/LST　用于列表文件

/REL　用于目标程序文件

D.1.4 DATE和TIME

COBOL-80使用由TRSDOS提供的日期和时间去标志编译程序清单题头，并返回ACCEPT TIME和DATE语句所要求的值。

D.1.5 CRT处理

由于ModelⅡ有一内部键盘和显示监控程序，所以提供的COBOL-80对你的硬件已经过配置。你可忽略附录A中关于配置CRT的描述。图D-1展示了如何使用键盘去录入格式3或格式4 ACCEPT语句用的数据和用于DISPLAY功能的管理程序调用。COBOL-80使用访管号8为到屏幕的全部输出，除光标位功能外，后者使用访管号10。

TRS-80 ModelⅡ终端

Ⅰ．键盘输入	输入码	键标号

A. 编辑键

1. 行删除/域删除	15	CONTROL-U
2. 字符删除	08	BACK SPACE
3. 进格	1D	→
4. 退格	1C	←
5. 加号	2B	+
6. 减号	2D	-

B. 终结符键	换码	输入码	键标号
1. 返回标记	99	1E	↑
2. 换码	01	1B	ESC
3. 域终结符	00		
a. 标记		09	TAB
b. 回车		0D	ENTER
c. 换行		1F	↓

C. 功能键	换码	输入码	键标号
1.	02	01	F1
2.	03	02	F2

Ⅱ. 输出功能　　管理程序调用

A. 置光标位置	SVC 10 B=行-1 C=列-1 DE=00
B. 光标回退	SVC 8 B=1C
C. 光标开	SVC 8 B=01
D. 光标关	SVC 8 B=02
E. 删至屏幕结束	SVC 8 B=18
F. 删至行结束	SVC 8 B=17
G. 振铃	SVC 8 B=07
H. 置高亮度方式	SVC 8 B=1A
I. 重置高亮度方式	SVC 8 B=19

D.2 ISIS-Ⅱ

D.2.1 文件名描述

用于ISIS-Ⅱ的文件说明形式为：设备:文件名.扩展名

用于COBOL-80命令行中的缺省扩展名为：

- COB　用于源文件
- LST　用于列表文件
- REL　用于目标文件

在提供的磁盘中可执行文件和由L80建立的文件没有扩展名。用于控制台和打印机的符号名分别为TTY:和LST:。

D.2.2 提供的软件磁盘

ISIS-Ⅱ COBOL-80提供的盘包含与CP/M磁盘相同的文件，但不包括实用程序LIB和SEQCVT

第二章　COBOL-80参考手册

引　言

Microsoft COBOL 是以美国国家标准文本X3.23-1974为基础的。整个 COBOL 语言由十二个不同的功能模块组成。

每个标准的 COBOL 模块有两个级别（一级和二级），一级是二级模块所包含的全部功能和特性的一个子集。

任何一个 COBOL 系统，至少要由核心、表处理和顺序存取的一级模块组成。

以下概述与标准有关的 Microsoft COBOL 的内容。

模块	COBOL-80特性
核心	一级模块的全部功能，加上二级模块的下列功能： 条件（CONDITIONS）： · 具有数值序列或值域范围的88层条件 · 条件中可使用逻辑 AND/OR/NOT · 可使用代数关系符号<，>，= · 在关系条件中有蕴含的主语或既蕴含主语也蕴含关系 · 符号测试 · 嵌套的 IF 语句，条件中可用括号 动词（VERBS）： · 对于格式屏幕处理的接收（ACCEPT）和显示（DISPLAY）的功能扩充 · 接收从 DATE/DAY/TIME 来的数据 · STRING 和 UNSTRING 语句 · 带有多个接收域的运算（COMPUTE） · PERFORM VARYING…UNTIL 标识符（IDENTIFIERS）： · 用于 ACCEPT 或 DISPLAY 设备的助记名 · 过程名只由数字组成 · 名的受限（仅用在过程部的语句中）
顺序相对和索引 I/O	· 一级的全部功能加上二级的以下功能： · RESERVE 子句 · 在 OPEN 和 CLOSE 中可出现多个操作数，其中每个文件有各自的选择项 · VALUE OF FILE-ID IS 数据名

顺序 I/O	· OPEN 的 EXTEND 模式 · WRITE ADVANCING 数据名行 · LINKAGE 短语和 AT END-OF-PAGE 子句
相对和索引 I/O	· 动态（DYNAMIC）存取模式（用 READ NEXT） · 启动（START），用 EQUAL，GREATER 或者 NOT-LESS 键关系符）
库	· 一级功能
程序间通讯	· 一级功能
表处理	· 一级功能 · SEARCH 语句的全部二级格式
调试	· ANSI-74标准的专门扩充，提供了方便的追踪方式调试功能 · 条件编译：在第七列中带有 D 的行在编译时不作处理，除非在 SOURCE-COMPUTER 段中给出“WITH DEBUGGING MODE”
分段	· 一级功能

第一节 COBOL的基本概念

1.1 字符集

COBOL 源程序字符集由如下字符组成：

字母 A～Z

空格

数字 0 ～ 9

特殊字符：

+	加号
-	减号
*	星号
=	等号
>	关系符号（大于）
<	关系符号（小于）
$	美元符号
,	逗号
;	分号
.	句号或小数点
“	引号
(	左圆括号
)	右圆括号
’	单引号，撇号

/ 斜杠

下列字符用于组成 COBOL 字：

0～9

A～Z

-（连字符）

下列字符作为分隔符号：

(左圆括号

) 右圆括号

, 逗号

. 句号

; 分号

下列关系符号用于简单条件中：

>

<

=

在非数值文字（引号内）注释体和注释行的情况下，COBOL字符集可扩充到计算机所包含的全部字符集。

1.2 标点符号

在书写源程序时，请注意以下通用的标点规则：

1．句号、分号和逗号作标点符号时前面不应出现空格，但其后必须跟一个空格。

2．两个相继的字或文字之间必须至少出现一个空格。除数值文字的情况外，多个连续的空格被视为一个空格处理。

3．关系符和算术运算符的前后应各出现一个空格。

4．逗号作为语句中相继操作数之间的分隔符号，或者作为两个下标之间的分隔符号。

5．分号和逗号用来分隔一系列的语句和子句。

1.3 字的格式

用户定义的字和系统保留字由不超过30个的字符组成，可从以下37个字符中选择：

0～9（数字）

A～Z （字母）

- （连字符）

除了过程名可以全部由数字组成以外，所有的字至少要含有一个字母或者连字符。连字符不可以出现在字的开头和结尾。每个字用一个空格或者适当的标点符号结束。一个字可以嵌入若干个连字符，相继的连字符也是允许的。

所有的字或者是预定义的保留字，或者是程序员定义的名。如果程序员给出的名不是唯一的，那么，必须有一种唯一的引用方法，这可以通过使用名限定语实现。例如，TAX-RATE IN STATE-TABLE。

非保留字主要用于识标一个数据项或数据段，称其为数据名。其他非保留字可以是文件名、条件名、助记名和过程名。

1.4 格式表示法

为了指导程序员在自己的程序中书写各式各样的语句，本书中规定了“通用格式”。用一套统一的规则表示，说明如下：

1．完全用大写字母印出的字是保留字，这些字已预先给出了确定的意义，在任何格式中，用大写字母写出的字表示实际的字实体。

2．所有画底线的保留字都是必写的，除非包含这些保留字的部分本身是任选的。这些字是“关键字”。任何关键字的拼写错误或漏写，在程序中均被认为是错误。程序员可以决定忽略还是保留那些不带底线的保留字，这些保留字是任选的。它们仅被用来增加程序的可读性。

3．字符<，>，=虽然未画底线，但在语句格式中出现时不能忽略。

4．所有的标点符号和其他专用字符表示这些字符的实际实体。凡是出现标点的地方一定都要写出。按照本章1.2节中规定的标点规则，还可以插入一些其他的标点符号。所有逗号、分号、句号作为分隔符时，其后面必须出现一个空格。

5．格式中以小写字母表示的字是类属项（如数据名），用户必须在源程序中用具体的项代替它们（注：小写字母在本手册中已译成汉字形式）。

6．方括号（[]）中的语句或数据描述体部分都是任选的，花括号（{}）内各部分之间是表示互斥任选。

7．一些项的格式由大写字母和其后跟的子句（Clause）或语句（Statement）组成，这些子句和语句在其他章节中描述。

8．为了在解释正文中方便地引用小写字母组成的字，在这些字后加一个连字符；其后再跟一个数字或字母，这并不改变该字的语法含义。

9．在互斥项之间加一竖杠说明可替换的任选项，例如：

AREA | AREAS 等价于 $\left\{\begin{matrix}\text{AREA}\\\text{AREAS}\end{matrix}\right\}$

10．省略号（…）表示紧接在它前面的部分可以出现一次或重复出现多次。“单元”是指一个小写的字或一组小写字以及方括号或花括号内的若干保留字。当指定花括号或方括号内的项重复时，它们作为一个整体被重复。

11．对于不妨碍理解的选择表示，可以用圆括号代替方括号指定任选成分。

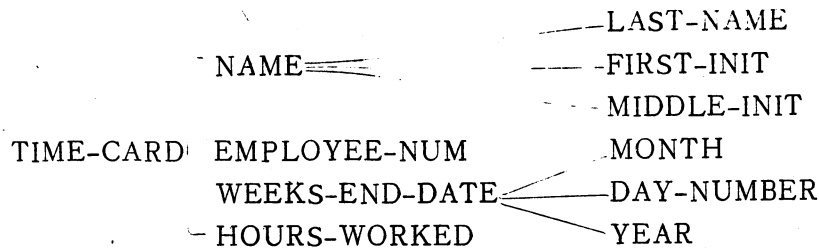
12．本书有关的各章节中叙述了各种格式的意义和使用方法，并给出了有关的注释、限制和使用说明。

1.5 层号和数据名

为了处理数据，把文件的内容划分成逻辑记录，逻辑记录的描述从层号01开始。构成一个逻辑记录的从属项组成层次，用层号02到49标识。层号不必是连续出现的。另外，层号77在工作存储节和连接节中标识一个“独立项”，即它不象01层一样有从属的基本项。88层用来定义条件名和有关的条件。小于10的层号可用一位数字来表示。

层次是为了引用数据对记录进行划分而设的。指定记录的子部分，可进一步划分记录，更精细地分层引用数据。

下面表明了一个“每周工时卡”记录，其中包括四个主项：姓名（NAME）、职工号（EMPLOYEE-NUM）、周末日期（WEEKS-END-DATE）和工时（HOURS-WORKED），而在姓名和周末日期项下有更详细的描述信息。



记录中本身不再进行划分的子部分称为基本项。而含有子部分的数据项叫做组项。当过程部语句中引用一个组项时，它实际上是引用组项的整个保留区域的内容。所有的基本项必须用 PICTURE 或者 USAGE IS INDEX 子句来描述。从属于同一个文件的相继的逻辑记录表示隐含地对相同的数据域的再定义，而在工作存储节中，每个01层的记录具有自己的存储区定义。

层号小的组项包含层号大的组项。组内项的层号不必是连续的。层号为K 的组项包含其下所有层号大于K 的组项和基本项。

在源程序中应分层书写组项，单独地列出实体。为了说明层号和组项，以前用到的实例“每周工时记录卡”用数据部描述体说明。层号、数据名和 PICTURE 定义如下：

```
01  TIME-CARD.
    02 NAME.
       03 LAST-NAME          PICTURE X(18).
       03 FIRST-INIT         PICTURE X.
       03 MIDDLE-INIT        PICTURE X.
    02 EMPLOYEE-NUM          PICTURE 99999.
    02 WEEKS-END-DATE.
          05 MONTH              PIC 99.
          05 DAY-NUMBER         PIC 99.
          05 YEAR               PIC 99.
    02 HOURS-WORKED          PICTURE 99V9.
```

数据名是由用户指定，为标识程序中所用数据项而设的字。数据名是用来引用数据的，它涉及一个区域，而不是指一个具体的数值。常常假设程序中的数据项在不同时期有不同的值。

每个数据名必须用字母开头，如下面的通用格式所示。在每个记录描述体中，层号后的第一个字必须是数据名，或者是关键字 FILLER：

〈层号〉 {〈数据名〉 / FILLER}

数据名是这个描述体的定义名，用它来引用相应的数据区（包含数据项值的区域）。如果在某些程序的处理步骤中，记录中的某些字符不用，那么，这些字符的数据描述就不必含有数据名。这时，只把 FILLER 写到层号后数据名所出现的地方。

1.6 文件名

文件是若干个数据记录的集合，例如列表打印输出或软磁盘上的一个区，包含相同种类或应用的若干不同记录。在数据部的文件节中，用文件描述体 FD 定义文件名。FD 是一个

保留字，其后必须出现一个由程序员提供的唯一的文件名。文件名的组成规则与数据名的组成规则一样（见本章1.3节）。通常在过程部语句OPEN，CLOSE和READ以及环境部中引用文件名。注意：不要把文件名与本章3.14.2节中描述的文件标识符ID混淆。

1.7 条件名

条件名可在数据部的层号88下定义。在数据项所设的值域内，可以给条件名赋予特定的值，或是一组值，或是一个值域。名的组成规则见1.3节中的规定。我们将在数据部和过程部的有关章节中进一步解释条件名说明以及引用它的过程语句。

1.8 助记名

助记名在环境部指定，以便在ACCEPT和DISPLAY语句中引用。它把用户定义的字分配给一个实际设备名，例如PRINTER。助记名的组成规则仍按1.3节中的规定。

1.9 文字

文字是不用数据名标识的常数。文字或者是数值的，或者是非数值的。

非数值文字

非数值文字必须要用配对的双引号或撇号括住。引号内的所有空格都是文字的一部分。非数值文字不允许超过120个字符。

下面是非数值文字的例子：

```
''ILLEGAL CONTROL CARD''
'CHARACTER-STRING'
''DO'S & DON'T'S''
```

文字中的每个符号都可以是除定界符外的任何字符，即如果文字以双引号定界，那么文字内可出现单引号，反之也成立。非数值文字的最小长度（不包括定界符）是1。

一个文字内相邻的两个定界符被看作是一个单独的定界符来处理。非数值文字可以从一行继续到另一行上。当非数值文字太长，程序纸一行不够时，可利用下面规则：

1. 在继续行的第七列上放一个续行符。
2. 把定界符放在B区继续行的前面。
3. 前一行结束时的所有空格，继续行定界符后以及最后的定界符前的所有空格，都被认为是文字的一部分。
4. 任何继续行上，A区应为空。

数值文字

数值文字至少要含1位数字，但不能超过18位数字。每位数字可以是0到9以及小数点（前面可带符号）。但数值文字中只能包含一个小数点和一个符号。不带符号的数值文字被看作是正数。

小数点可出现在数值文字的任何位置，但不能是最右位置。不包含小数点的数值文字，则认为是整型数。

下面是数值文字的一些例子：

```
72      +1011      3.14159      -6      -3.33      -0.5
```

如果在环境部使用说明DECIMAL-POINT IS COMMA，则句号的功能与逗号的功能互换，使用欧洲表示法。例如，此时数值文字π则写成3.14159。

1.10 象征常数

象征常数是文字的一种特殊形式。它代表一个赋给标准数据名的值，象征常数不用双引号定界。

在程序的许多地方可以使用ZERO作为数值文字。其他的象征常数提供非数值数据。表示各种不同字符的保留字如下：

SPACE	空格字符，以八进制40表示
LOW-VALUE	以八进制数00表示的字符
HIGH-VALUE	以八进制数177表示的字符
QUOTE	引号，以八进制42表示
ALL 文字	一个或多个包含单字符的非数值文字或象征常数，若为象征常数时，ALL是冗余的，则保留以增加易读性。

编译程序可以接受象征常数的复数形式，其作用是等价的。象征常数可表示语句上下文中所要求数量的字符。

象征常数可用在通用格式中要求文字的任何地方，除非文字被限定为数值型，此时只允许象征常数ZERO。

1.11 程序的结构

每个COBOL源程序都可以划分成四个部分。每一部分必须要按照一定的次序放置。每个部分必须以部分头开始。

以下是按顺序列出的四个部分及其功能：

标识部，对程序命名。

环境部，指定程序所使用的计算机设备及其特性。

数据部，定义要处理的数据名及其属性。

过程部，在执行时指导数据处理，控制语句执行的过程。

如果漏写了部分头或有拼写错误，COBOL将无法编译源程序，此时，可能发生无法预测的事件。

下面列出程序规定的各部分的结构及其出现的次序：

```
IDENTIFICATION DIVISION.
PROGRAM-ID. 程序名.
[AUTHOR. 注释体…]
[INSTALLATION. 注释体…]
[DATE-WRITTEN. 注释体…]
[DATE-COMPILED. 注释体…]
[SECURITY. 注释体…]
ENVIRONMENT DIVISION.
[CONFIGURATION SECTION.
[SOURCE-COMPUTER. 描述体]
[OBJECT-COMPUTER. 描述体]
[SPECIAL-NAMES. 描述体]]
[INPUT-OUTPUT SECTION.
FILE-CONTROL. 描述体…
```

〔I-O-CONTROL. 描述体…〕〕
DATA DIVISION.
〔FILE SECTION.
〔文件描述体 记录描述体…〕…〕
〔WORKING-STORAGE SECTION.
〔数据项描述体…〕…〕
〔LINKAGE SECTION.
〔数据项描述体…〕…〕
〔SCREEN SECTION.
〔屏幕描述体…〕…〕
PROCEDURE DIVISION〔USING〔标识符1〕…〕.
〔DECLARATIVES.
〔节名 SECTION. USE 句子.
〔段名.〔句子〕…〕…〕…
END DECLARATIVES.〕
〔〔节名 SECTION.〔分段号〕〕
〔段名.〔句子〕…〕…〕…

1.12 源程序书写规则

既然 Microsoft COBOL 是 ANSI COBOL 的一个子集，因此，程序可按照规定编写在标准的 COBOL 纸上，并适用如下规则：

1．每行的1-6列中应有一个六位数字的顺序号，序号必须是递增的。在1-6列中也允许出现空格。

2．部、节、段头所设保留字必须从A区写起(8-11列)，过程名也必须从A区写起。层号也可以在A区出现，层号77，01和层指示字FD必须从A区开始。

3．所有其他的程序元素应在12-72列上书写，它受语句的标点法规则约束。

4．编译程序忽略73-80列，程序员可用其标记程序用。

5．说明性的注释可以插到源程序的任一行上，只要在该行第7列上写一个星号（*）即可。这样的行在源程序列表时照样列出，没有别的作用。如果在某行的第7列位置上出现一个斜杠（/），则相应的行就作为注释行处理，在编译程序列表输出时，打印在每页的顶端。

6．任何一个程序的元素都可以在源程序下一行中继续。非数值文字的续行规则已在1.9节中解释。而任何其他的字或文字，或者程序元素，都通过在续行的第7列上放入一个连字符“-”来实现续行。其效果是忽略前一行最后一个字符后的空格和续行开始第一个字符前的空格，而把相继的两个部分连接起来。在续行上A区必须为空。

7．一行中的任何标记字符都可被扩展，就好象每第8列有一个标记停一样；但第一个标记停在第7列上，只经过6个顺序的列，其后则按照一般规则使标记停在17，25，33列等等。

1.13 名的受限

当数据名、条件名或者段名不唯一时，使用限定语可使名的引用唯一。例如，若YEAR

是若干个项的命名，则用限定引用YEAR OF HIRE-DATE可区分 HIRE-DATE和TERMINATION-DATE中的YEAR字段。名的受限用OF或IN来实现，后继的数据名和条件名须指定为较低层号的组项，这些组项含有复合引用中所有前面的名，例如，HIRE-DATE必须是包含YEAR项的一个组项。段名可通过节名来受限，限定语的最大数是5个。文件名和助记名必须是唯一的。受限名只可以书写在屏幕节或过程部中。当从同一节内引用段名时，对多重定义的段名无法受限。

1.14 COPY语句

COPY语句用来在逻辑上把磁盘文件正文（而不是源程序文件）插到输入给COBOL-80编译程序的源代码中。COPY语句的格式是：

COPY 正文名.

其中正文名是按现有操作系统格式要求的磁盘文件名。例如，假设BDEF.COB是包含了如下源代码的正文文件：

```
05 B.
  10 B1 PIC X.
  10 B2 PIC X.
```

那么，源文件

```
05 A.
  10 A1 PIC 9.
COPY BDEF.COB.
05 C.
  10 C1 PIC Z.
```

的编译如同处理如下源程序：

```
05 A.
  10 A1 PIC 9.
05 B.
  10 B1 PIC X.
  10 B2 PIC X.
05 C.
  10 C1 PIC Z.
```

包含COPY语句的一行源程序从正文名结束到该行结尾只能是空格。

第二节 标识部和环境部

2.1 标识部

每个COBOL程序必须以标题IDENTIFICATION DIVISION开始。标识部分为几段，段名已规定如下：

```
PROGRAM-ID.        程序名
AUTHOR.            注释
INSTALLATION.      注释
```

DATE-WRITTEN. 注释
DATE-COMPILED. 注释
SECURITY. 注释

PROGRAM-ID段是必需的，而且它必须作为第一段出现。程序名是以字母开头的字母数字串。编译程序只保留程序名的前六个字符。程序名也标识目标程序，而且在打印编译清单时，它出现在清单标题中。

其他段的顺序没有规定，只作为文件标注。

2.2 环境部

环境部用一种标准的方法表示COBOL程序中与特定的计算机物理特性有关的方面。每个COBOL程序都需要有环境部。

环境部的通用格式是：

```
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER. 计算机名〔WITH DEBUGGING MODE〕.
OBJECT-COMPUTER. 计算机名
    〔MEMORY SIZE 整数 WORDS | CHARACTERS | MODULES〕
    〔PROGRAM COLLATING SEQUENCE IS ASCII〕.
SPECIAL-NAMES. 〔PRINTER IS 助记名〕 ASCII IS { STANDARD-1 | NATIVE }
    〔CURRENCY SIGN IS 文字〕
    〔DECIMAL-POINT IS COMMA〕.
INPUT-OUTPUT SECTION.
FILE-CONTROL. {文件控制体}…
I-O-CONTROL.
    〔SAME { RECORD | SORT | SORT-MERGE } AREA FOR 文件名 …〕…
```

2.2.1 配置节

配置节可有三个任选的段，它们是源计算机段（SOURCE-COMPUTER）、目的计算机段（OBJECT-COMPUTER）和专用名段（SPECIAL-NAMES）。除子句WITH DEBUGGING MODE外（见4.22节），前两段只作为注释说明用。第三段，SPECIAL-NAMES使得设备名与用户定义的名联系起来，并改变缺省编辑字符。用户可用 PRINTER IS短语定义一个在带有UPON的DISPLAY语句中使用的名。

如果不希望货币符号是美元符，用户可以在CURRENCY SIGN子句中指定一个单字符非数值文字。但所指定的字符不能是引号，不能是数字0～9，也不能是PIC描述中定义的任何字符。

欧洲的习惯是用逗号代替小数点分隔整数和小数，这可用 DECIMAL-POINT IS COMMA 来指定。

注意：在货币符定义和小数点约定说明中，子句中使用的保留字IS不能省略。

描述体ASCII IS NATIVE/STANDARD-1指定数据用美国标准信息交换码（ASCII码）来表示。当ASCII描述体缺省时，该约定也成立。本编译中，NATIVE和STANDARD-1是同等的，参考附录IV中指定的字符集。

2.2.2 输入输出节

环境部的第二节是必需的，除非程序没有数据文件；它以如下标题开始：

INPUT-OUTPUT SECTION.

这节有两个段：文件控制段（FILE-CONTROL）和I/O控制段（I-O-CONTROL）。在这里，程序员定义文件赋值参数和规定缓冲区。

2.2.2.1 文件控制描述体

在数据部的文件节中有记录描述的每个文件，在文件控制段要求一个选择句描述体，描述以SELECT开始。一个顺序文件的选择句描述体格式是：

```
SELECT 文件名 ASSIGN TO DISK | PRINTER
〔RESERVE 整数 AREAS | AREA〕
〔FILE STATUS IS 数据名1〕
〔ACCESS MODE IS SEQUENTIAL〕〔ORGANIZATION IS〔LINE〕SEQUENTIAL〕.
```

选择描述体必须从源程序行的A区右边开始。在“SELECT 文件名”之后的所有短语的次序是任意的，对顺序输入/输出处理，可选用ACCESS和ORGANIZATION子句。对索引和相对文件，该节也有通用格式可供选用，将在有关章节中解释。

顺序磁盘文件有两种格式。一种是ORGANIZATION IS SEQUENTIAL要求的规则格式，另一种是由ORGANIZATION IS LINE SEQUENTIAL所要求的格式。两种格式都假设文件中的记录是可变长的。规则的顺序组织是每个实际的记录前有两个字节的记录长度计数。行顺序组织是每个记录后有一个回车换行作为分隔符号。不应该把COMP或者COMP-3信息写入行顺序文件，因为这些数据项可能含有与用作回车换行符相同的二进制代码。否则，在以后读文件时就会出现问题。两种文件都将一些多余的空格和Control-Z字符放在物理块的最后，用以表示文件结束。所有记录都连续存放在文件中，跨越物理块边界。

虽然COBOL-80中的RESERVE子句不起作用，但要对它进行语法扫描。通常总是把一块物理缓冲区分配给一个指定的逻辑记录区，这就允许逻辑记录跨越物理块的边界。对于分配给PRINTER的文件，逻辑记录区也用作物理缓冲区。

在FILE STATUS描述体中，数据名1必须引用两个字符的工作存储或连接节字母数字项，在运行时，数据管理功能在I-O语句后把状态信息放入该字母数字项中。数据名1的左字节设有如下值：

‘0’ 顺利完成

‘1’ 文件结束

‘2’ 无效关键字（仅用于索引和相对文件）

‘3’ 不可恢复的I-O错误

如果对于以前的I-O操作不出现进一步的状态信息，则数据名1的右字节被置成‘0’。可能出现如下的组合值：

文件状态左字节	文件状态右字节	意义
'0'	'0'	O.K.
'1'	'0'	EOF
'3'	'0'	永久错误
'3'	'4'	盘空间满
'9'	'1'	文件损坏

在OPEN INPUT或OPEN I-O语句中，文件状态'30'意味着文件没有找到。

如果文件状态的左字节有一个值'2'时，状态右字节的情况见索引文件和相对文件的有关章节。

2.2.2.2 输入输出控制段

SAME AREA子句是任选的。在COBOL-80中只有SAME RECORD AREA形式有效，而其他形式只作语法检查，不分享任何物理缓冲区。

SAME RECORD AREA形式使所有命名的文件共享同一个逻辑记录区，从而节省了内存空间。

SAME AREA描述体格式如下：

SAME、[RECORD / SORT / SORT-MERGE] AREA FOR 文件名…

在SAME AREA子句中列出的文件不必有同样的组织形式和存取方式，但同一个文件不能在多个SAME AREA子句中出现。

仅在支持SORT功能的COBOL-80版本中允许分类合并（SORT-MERGE）任选。

第三节 数据部

数据部是程序中必须出现的部分，它由四个节组成：文件节、工作存储节、连接节和屏幕节，将分别在3.13～3.16节中讨论。下面首先讨论每节都要用到的数据说明。

3.1 数据项

COBOL程序可用几种不同的数据项，介绍如下：

3.1.1 组项

含有子部分的数据项称为组项。组项包含一个或多个基本项。此外，一个组项也可以包含其他的组项。一个项，当且仅当它的层号小于紧接项的层号时为组项。不是组项的项就是基本项。通常，任何数据项的最大长度是4095个字节。为了允许有些应用中的表超出这种限制，对于层号为01的组项不检查和限制其长度。编译程序不允许过程部语句操作数中有大于4095字节长度的项，例如在MOVE、INSPECT等语句中。

3.1.2 基本项

基本项是不包含从属项的数据项。

字母数字项：字母数字项包含字符的任意组合，这种字符组合形成一个“字符串”数据字段。如果相应的PIC语句含有“编辑”字符，那么它就是字母数字编辑项。

报表（编辑）项：报表项是含有数字和（或）专门的编辑字符的编辑“数值”项。它的长度不超过30个字符。报表项只能用来作为接收数值数据的接收域。它的目的是为接收数值项，但它自身不能作为数值项使用。

3.1.3 数值项

数值项是只能含有数值数据的基本项。

外部十进制项：外部十进制项的每个字节用来表示一位数字。其长度允许最大18位数字。通过在PIC描述中给出具体的“9”的个数来确定数字的位数。例如“PIC 999”确定了一个三位数字的项，也就是说，此项所能表示的最大值是999。

如果PICTURE描述体以字母“S”打头，那么这一项还可以含有一个运算符号。运算符号不另占用字节，除非在项描述中使用了带SEPARATE格式的SIGN子句。不管运算符的表示形式如何，其目的是提供一种常规的代数符号功能。

USAGE子句在外部十进制项中用DISPLAY格式（见3.4节USAGE子句）。

内部十进制项：一个内部十进制项用压缩十进制格式存放。它可用 USAGE IS COMPUTATIONAL-3子句实现。在PICTURE 子句中由几个9定义的压缩十进制项，占用内存（n+2）/2个字节（下舍入）。除了最右边的一个字节外，每个字节都含有2位数字，而每位数字用与有效数字值0～9等价的二进制数表示。项的最低位数字和项的符号放在组项的最右端字节中。因此，即使原来的PICTURE描述中没有S字符，编译程序也认为该组项有一个算术符号。

二进制项：二进制项使用二进制数表示从－32768到＋32767范围内的一个整数。该数占用一个16位字。字的最左边一位作为符号位。二进制项由 USAGE IS COMPUTATIONAL 指定。

索引数据项：索引数据项不用 PICTURE 描述，它用 USAGE IS INDEX 子句来定义（参考第六节“索引法表处理”）。

3.2 数据描述体

数据描述体规定一个数据记录中每一项(字段)的特性。必须根据项在记录中出现的先后次序，用单独的实体逐个描述每一项。每一个数据描述体由一个层号，一个数据名和一系列子句，后跟一个句号组成。数据描述体的通用格式如下：

层号 {数据名 / FILLER} (REDEFINES-子句) (JUSTIFIED-子句)
(PICTURE-子句) (USAGE-子句) (SYNCHRONIZED-子句)
(OCCURS-子句) (BLANK-子句) (VALUE-子句) (SIGN-子句).

当用这个格式描述具体的数据项时，要受所描述数据性质的限制。不同的数据类型所允许的描述格式在下面给出。对于各种数据类型所禁止的子句在下面的相应格式中不再出现。在某些数据项的描述体中必须要出现的子句不再有圆括号。除了REDEFINES子句外（如果用到，它必须出现在开始），其他子句可以按任何次序排列：

组项格式：

层号 {数据名 / FILLER} (REDEFINES-子句) (USAGE-子句)
(OCCURS 子句) (VALUE 子句) (SIGN-子句).

例：

```
01  GROUP-NAME.
        02  FIELD-B  PICTURE  X.
        02  FIELD-C  PICTURE  X.
```

注意：为了避免USAGE子句在从属项中重复书写，可把它写在组项层中。

3.3 基本项格式

字母数字项（亦称字符串项）：

层号 {数据名 / FILLER} (REDEFINES-子句) (OCCURS-子句).
PICTURE IS 字符形式 (USAGE IS DISPLAY) (JUSTIFIED-子句)
(VALUE IS 非数值文字) (SYNCHONIZED-子句).

例：

```
02  MISC-1 PIC X(53).
02  MISC-2 PICTURE BXXXBXXB.
```

报表项（也叫数值编辑项）：

层号 {数据名 / FILLER} (REDEFINES-子句) (OCCURS-子句).
PICTURE IS 报表形式 (BLANK WHEN ZERO) (USAGE IS DISPLAY)
(VALUE IS 非数值文字) (SYNCHRONIZED-子句).

例：

```
02  XTOTAL PICTURE $999,999.99-.
```

十进制项：

层号 {数据名 / FILLER} (REDEFINES-子句) (OCCURS-子句)
PICTURE IS 数值形式 (SIGN-子句).
(USAGE-子句) (VALUE IS 数值文字) (SYNCHRONIZED-子句).

例：

```
02  HOURS-WORKED PICTURE 99V9,USAGE IS DISPLAY.
02  HOURS-SCHEDULED PIC S99V9,SIGN IS TRAILING.
-- -- -- -- -- -- -- -- -- --
11  TAX-RATE PIC S99V999 VALUE  1.375,COMPUTATIONAL-3.
```

二进制项：

层号 {数据名 / FILLER} (REDEFINES-子句) (OCCURS-子句)
PICTURE IS 数值形式
USAGE IS COMPUTATIONAL | COMP | INDEX
(VALUE IS 数值文字) (SYNCHRONIZED-子句).

注意：PICTURE或VALUE不能描述索引数据项；

例：

02 SUBSCRIPT PICTURE 999 COMP,VALUE ZERO.

02 YEAR-TO-DATE PIC S9(5) COMPUTATIONAL.

3.4 USAGE 子句

USAGE子句用来指定表示数据的形式。

USAGE子句可以出现在数据的任一层。如果没有指定 USAGE 子句，那么系统就假设该项数据是“DISPLAY”格式。USAGE子句的一般格式是：

USAGE IS { COMPUTATIONAL / INDEX / DISPLAY / COMPUTATIONAL-3 }

INDEX的用法在第六节“表处理”中解释。COMPUTATIONAL可缩写成COMP，定义一个整数的二进制字段。COMPUTATIONAL-3可缩写成COMP-3，定义一个压缩（内部十进制）字段。

如果在组项层给出了USAGE子句，那么，它的描述适用于组项内每一个基本项的描述。因而，任何基本项的USAGE子句不能与其隶属的组项的USAGE描述相矛盾。

3.5 PICTURE 子句

PICTURE子句用来详细地描述基本层的数据项，它还包括规定专门的报表编辑。可以把保留字PICTURE缩写成PIC。PICTURE子句的通用格式是：

PICTURE IS { 字符形式 / 数值形式 / 报表形式 }

有三种类形的数据形式：字符形式，数值形式和报表形式。

字符形式任选：此选择形式适用于字母数字项(字符串项)。字母数字项用到的PICTURE子句是由数据描述字符 X、A或者 9 以及编辑字符B、 0 或者/的组合而形成。而字符X表示数据形式中X的位置上可以是计算机ASCII字符集中的任何一个字符。

一个PICTURE子句至少包含下列组合的一种（顺序任意）：

a) A和 9

b) X和 9

c) X和A

系统把每个字符 9 、A或者X看成X。可用字符B、 0 和/表示在数据项的数据中插入空格、零或斜杠。插入了这些字符的数据项叫做字母数字编辑项。

如果描述字符串只由A和B组成，则所描述的对象就是字母数据项。而只由 9 组成的字符串所描述的对象是数值数据项。

数值形式任选：数值项的PICTURE子句的描述字符串可由如下字符组成：

9 ——字符 9 指出必须包含数值字符的位置。PICTURE中最多可以用18个 9 来描述。

V——任选字符V设定一个小数点的位置。既然数值项不能包含一个实际的小数点，那么设定的小数点用来为编译程序对计算项进行比例对齐。字符V并不占用存储区。在每个PICTURE中只允许一个V。

S——任选字符S表示数据项有一个运算符号。它必须是PICTURE的第一个字符。见3.12

节中SIGN子句。

P——字符P表示一个设定的十进制小数点位置，当数据项中的数据没有小数点时，用它来指定假设十进制小数点的位置。定位字符P不算在数据项的长度内，即这些字符位置不占内存。然而，在算术语句中，对于作为操作数出现的项，或在数值编辑项中，当确定最大数字位置数（18个）时，则包括定位字符P。在PICTURE描述中，定位字符P只能连续地出现在描述字符串中其他字符的左边或者最右边。只有符号S和设定的十进制小数点V可出现在最左P字符串的左边。既然定位字符P隐含着一个假设的小数点(如果P是最左边的PICTURE字符，隐含小数点在P字符串左边，如果P是最右边的PICTURE字符，则隐含的小数点在P字符串的右边），因而符号V在有P字符串出现的PICTURE子句中，无论是放在最左边，还是放在最右边都是多余的。

报表形式任选：这种形式描述了适合于接收具体的数值字段的编辑数据项。可以用来组合形成报表项的编辑字符总结如下：

9 V. Z CR DB, $ + * B O - P /

字符9，P和V的用法和意义与数值项中这些字符的意义相同。现在就其他的编辑字符的意义叙述如下：

. ——小数点规定了在指定的位置上插入一个实际的小数点，因而也使源数据项校准定位。在PICTURE子句中，实际的小数点右边的数值字符位置上只能由同类型的字符组成。小数点字符不能是PICTURE字符串中的最后一个字符。如果使用了描述字符“.”，那么可以不用字符P。

Z——字符Z和*叫做替换字符。每个字符表示一位数字位置。在执行期间，报表数据项中用字符Z或*定义的位置上的前零被取消，而换成空格或者*。取消前零的过程中当遇到小数点（.或者V），或者遇到一个非零的数字时就停止。所有要替换的数字位置必须用相同字符（或者都是Z，或者都是*）并且从左到右连续进行。仅当所有的数字位置相同时，Z或*才可以出现在实际小数点的右边。

CR DB——这两个符号分别叫做贷、借符号。它们只可以出现在PICTURE子句的右端。每个符号占用两个字节，表示：如果源项的值是负的，规定的贷、借符号要出现在指示的位置上；如果源数据项的值为正数或零，则在指示的位置上填入空格。PICTURE中，符号CR、DB、+和-是互相排斥的。

, ——该字符表示在数字之间插入一个逗号。每个插入的逗号都算在数据项长度之中，但它并不代表一个数据的数字位置。逗号用来联接一个浮动串。在PICTURE字符串中，逗号不能作为最后的字符出现。

一个浮动串由$、+、-中之一的字符组成，它可以选择地插入若干个小数点或者逗号。例如：

```
$$, $$$, $$$
+ + + +
- -, - - -, - -
+ (8) . - +
$$, $$$. $$
```

含有N+1个$或者+或者-的浮动串定义了N位数字位置。当把一个数值传送给报表项时，有关的适当字符从左向右浮动，使得到的报表项在最高位的非零数字左边恰好有一个实际的符号（或者是$、或者是+、或者是-）在PICTURE中$、+或-指定的一个位置上，而把空格放在浮动定位后的单个字符（$、+、-）左边的空余位置上。如果有效数字的最高位置出现在浮动串定义位置的右边，那么在浮动串的最右边位置上出现单个字符（$、+、-），而在其后可以跟着无效零字符。在浮动串中，存在一个实际的或者隐含的小数点时，我们可把它看成是小数点右边的所有位置都是用PICTURE子句中的9描述的。下例中，'b'表示结果项中的空格。

PICTURE	数值	结果项
$ $ $999	14	bb$014
$ $ $ $ $ $	14	bbb$14
- -，- - -，999	-456	bbbbb-456

如上例所示，浮动串不必构成报表项的完整的PICTURE子句。对浮动串后字符的限制在以后说明。

当在浮动串的右边出现一个逗号时，浮动字符将穿过逗号而浮动，以便它尽可能地接近有效数字的高位。

+ -——在PICTURE子句中，字符+或者-可以单个地出现，或在一个浮动串中出现。当+或者-作为一个固定的控制符号出现时，它必须是PICTURE子句中的最后一个字符。加号表示：根据放在报表字段中的数值的代数符号，通过那个字符位置上的加号或减号指出项的符号。减号表示：根据报表项中的值的代数符号是正的还是负的，分别决定把空格还是减号放在那个字符位置上。

B——在结果编辑值中，每个B在PICTURE子句中代表一个空格。

/——PICTURE子句中的每个斜杠代表最后的结果编辑值的一个斜杠。

0——在PICTURE子句中每个0在结果编辑项值数字0出现的地方代表一位零。

报表编辑项的其他应用规则是：

1．如果出现了一类浮动串，那么则禁止其他类型的浮动串再出现。

2．至少要有一位数字位置字符。

3．如果出现了一个浮动符号串，或出现了固定的插入字符加号或者减号，则不再允许出现任何其他的符号控制插入字符，即字符+、-、CR、DB。

4．从小数点右边开始一直到PICTURE描述结尾的字符，除了可能存在的固定插入字符+、-、CR、DB外，应该遵循以下规则：

a.在字符位置上只可以出现一类字符。即：字符类Z、*、9与浮动串数字位置字符$、+、-是互相排斥的。

b.如果小数点右边的一个数值字符位置是由+、-、$或者Z表示的，则PICTURE子句中的所有数值字符位置必须要用同样的字符来表示。

5．PICTURE字符9不允许出现在浮动串的左边和替换字符的左边。

此外，PICTURE子句还应注意以下几点：

1．PICTURE子句必须用在基本项层。

2．跟在X、9、$、Z、P、*、B、-和+字符后面的圆括号中的整数表示该PICTURE描

述字符连续出现的次数。

3．字符V和P不计在数据项的内存空间分配内，而CR和DB分别占用两个字符位置。

4．PICTURE后的描述字符串长度最大允许30个字符位置。例如，PICTURE X (89) 由5个描述字符组成。

5．PICTURE的描述字符至少要包含A、Z、＊、X和9字符中的一个，或者至少要含有二个连续出现的＋、－、$字符之一种。

6．PICTURE子句中，字符·、S、V、CR、DB只能出现一次。

7．当指定DECIMAL-POINT IS COMMA时，以上对于逗号和句号的解释分别适用于句号和逗号。

下面描述用PICTURE子句编辑数据的例子。每例中，蕴含着数据的传送（其中数据值显示了存储器的内容，PICTURE中给出了源数据域的比例因子）。

源域		接收域	
PICTURE	数据值	PICTURE	编辑数据
9(5)	12345	$$$,$$9.99	$12,345.00
9(5)	00123	$$$,$$9.99	$123.00
9(5)	00000	$$$,$$9.99	$0.00
9(4)V9	12345	$$$,$$9.99	$1,234.50
V9(5)	12345	$$$,$$9.99	$0.12
S9(5)	00123	------.99	123.00
S9(5)	-00001	------.99	-1.00
S9(5)	00123	++++++.99	+123.00
S9(5)	00001	------.99	1.00
9(5)	00123	++++++.99	+123.00
9(5)	00123	------.99	123.00
S9(5)	12345	******.99CR	**12345.00
S999V99	02345	ZZZVZZ	2345
S999V99	00004	ZZZVZZ	04

3.6 VALUE 子句

VALUE子句指定了工作存储项的初值。它的通用格式如下：

VALUE IS 文字

VALUE子句不能写在包含OCCURS或者REDEFINES子句的数据描述体中，也不可出现在这样的项描述体中，即：它从属于包含OCCURS和REDEFINES子句的描述体。此外，除了88层中的条件描述外，它也不能在文件节或连接节中使用。VALUE子句中给出的文字长度一定要小于或等于PICTURE子句中描述的项的长度。在数据域中文字的定位与指定MOVE把文字传送到数据域的定位方法大体相同，不同的是PICTURE中的编辑字符不影响初始化。BLANK WHEN ZERO或JUSTIFIED子句对初值也不影响。要根据数据项的类型书写文字类型，如文章前面的数据项格式中指出的。编辑项的值必须被指定为非数值文字，且必须以编辑形式出现。象征常数可以作为文字。

当未指定初值时，则无法假设工作存储区中项的初始内容。

VALUE子句可在组项层指定，其文字可以是规定范围内的非数值文字形式，或者是象征常数形式。这种情况下，VALUE子句不能再用到该组项的从属项中。但是VALUE子句不能用于这样的组项，即：它的从属项描述中含有JUSTIFIED，SYNCHRONIZED或者USAGE（USAGE IS DISPLAY除外）子句（用于88层项的子句格式将在3.16节中说明）。

3.7 REDEFINES 子句

REDEFINES子句是指要用同一个数据区存放不同的数据项，或者为同一个数据区提供另一种分组或描述。其格式如下：

REDEFINES 数据名2

使用这个子句时，它必须是数据描述体中数据名后的第一个子句。数据名2用的数据描述体既不能含有REDEFINES子句，也不能含有OCCURS子句。

当对某个数据区重定义时，该数据区的全部描述仍然起作用。这样，由于重定义，如果使两个独立的项B和C共享同一个存储区，在程序的任一点则可能执行过程语句MOVE X TO B或MOVE Y TO C。第一种情况，B将采用X的值而取B描述所指定的形式；而在第二种情况下，同一物理存储区将按照C的描述接收Y值。

为了便于讨论重定义子句，我们用术语“源”来称数据名1，而用“目标”称数据名2。源和目标所在的层分别用s和t来表示。为了正确地使用重定义子句，必须遵循以下规则：

1. s必须等于t，但不能等于88。
2. 目标必须被包含在同一记录（01层的组项）中，除非s = t = 01。
3. 在源定义之前和目标定义之后不能含有小于s的层号。

数据名1的长度乘以数据名1的出现次数不可以超过数据名2的长度，除非数据名1的层号是01（这仅在文件节以外允许）。除了88层外，数据名1及其任何从属项的描述体均不能含有赋初值子句。在文件节中，对从属于同一个FD的多个01层描述体，实际上都隐含地重定义了相同的存储区。

3.8 OCCURS 子句

OCCURS子句用来定义一些相关的重复数据集，例如：表、清单和数组。该子句规定一个数据项以同一格式重现的次数（在这里最多为1023次）。若某项含有OCCURS子句描述，则与该项有关的数据描述子句适用于该项的每个重复。当用到OCCURS子句，数据描述体中规定的数据名在过程部出现时，必须要用下标或索引进行引用。如果该数据名是一个组项名，那么使用所有从属于该组项的数据名时，必须要用下标或索引来引用。

在01层和77层的数据描述体中不能使用OCCURS子句。OCCURS子句的格式如下：

OCCURS 整数 TIMES[INDEXED BY 索引名…]

OCCURS子句在一个记录内只能用在从属层描述中，表的大小限制规则与组项的限制规则一样。见3.1.1节中有关组项的说明。

下标：下标为引用“表”中未指定单独数据名的数据项提供了手段。下标由数据描述中的OCCURS子句决定。如果某项有一个OCCURS子句，或者它从属于某个有OCCURS子句的组项，那么在用到该数据项时必须用下标或索引来引用。见“表处理”一节中有关索引及其用法的说明（例外：在SEARCH语句中，表名不可用下标引用）。

下标是一个非零的正整数，它的值确定了表中引用的元素。下标可以用一个文字或者有一个整数值的数据名来表示。无论用数据名或文字表示，下标均要用圆括号括住，并放在元素名末位之后。下标必须是十进制项或二进制项（为提高效率，常用后者作下标）。

对于任一数据项，最多可用三个 OCCURS 子句描述。这样就可能用到一，个或三用逗个下标。当需要一个以上的下标时，则下标用递减数据组织维数的次序来书写。下标之间号分隔，例如 ITEM(I, J)。

例：

```
01  ARRAY.
        03  ELEMENT,OCCURS 3,PICTURE 9(4).
```

上例按如下方式分配存储单元：

元　素(1)	ARRAY 共包含
元　素(2)	12个字符，每项有
元　素(3)	4位数字。

如果一个数据名被用作：

1. 一个下标
2. 数据描述体的定义名
3. REDEFINES 子句中的数据名 2
4. 限定符

则该数据名不可有下标。

3.9 SYNCHRONIZED 子句

为了用有效的方法分配存储空间(对计算机内存而言)，专门设计了 SYNCHRONIZED 子句。但是在本编译系统中，SYNCHRONIZED 只起到注释的作用。

该子句格式如下：

SYNC | SYNCHRONIZED[LEFT | RIGHT]

3.10 BLANK WHEN ZERO 子句

BLANK WHEN ZERO 子句表示：如果送入一报表编辑项的值是零，则该报表编辑域除含有空格外，不包含任何其他字符。当该子句用于数值的 PICTURE 描述时，该项作为报表项来考虑。

3.11 JUSTIFIED 子句

JUSTIFIED RIGHT 子句仅适用于非编辑的字母数字项（字符串项）。它表示该值是按从右到左的方式存放的，因而，当把一字段传送至一个较长的 JUSTIFIED 说明的接收域中时，左面不足部分补之以空格；反之，当把一个较长的字段传送至一个 JUSTIFIED说明的较短的接收域中时，左面多余部分被截去。仅当有关的域在 MOVE 语句中作为接收域时，JUSTIFIED 子句才是有效的。

JUST 为 JUSTIFIED 的缩写。

3.12 SIGN 子句

可以使用四种方法来表示外部十进制项的运算符号，究竟选用何种方式，由SIGN子句的格式决定：

〔SIGN IS〕 { TRAILING / LEADING } 〔SEPARATE CHARACTER〕

下表概括了子句四种可能形式的效果。

SIGN 子句	符号表示
TRAILING	嵌入最右边字节中
LEADING	嵌入最左边字节中
TRAILING SEPARATE	存入最右边一单独字节中
LEADING SEPARATE	存入最左边一单独字节中

当用上述几种形式书写程序时，PICTURE子句必须以S开始。如果没有S，就不能为该项标记符号（只能存放其绝对值），因而也禁止使用SIGN子句；而当PICTURE之前有S开头，但在数据项描述中未用到SIGN子句时，缺省的情况是SIGN IS TRAILING。

SIGN子句也可以出现在组项层。这时，子句所指定的符号格式适用于所有从属的外部十进制项。SEPARATE CHARACTER短语使数据项长度增加一个字符。适用于SIGN子句的描述体必须隐含或者明确地被描述成USAGE IS DISPLAY。

注意：当用CODE-SET子句描述一个文件时，该文件中所有带符号的数值数据都必须用SIGN IS SEPARATE子句描述。

3.13 88层条件名

88层条件名描述体为基本项设定一个值，一组值或者给出一个值域范围。符合条件时，命名的逻辑为真，否则为假。88层项的赋初值子句格式是：

VALUE IS 文字1〔文字2…〕

VALUES ARE 文字1 THRU 文字2

一个88层描述体后或者是另一个88层描述体(这时，几个相继的条件名属于一个基本项)或者是一个基本项（可以是FILLER）。

索引数据项后不应该跟着88层项。属于一个基本项的每个条件名是这样的：该条件名可由基本项名和该基本项的限定符限定。条件名可以用在过程部中代替简单的关系条件。条件名也可以属于一个带下标的基本项（一个条件变量）。在这种情况下，当在过程部中使用它们时，必须按照对基本项使用下标的有关要求进行。条件名描述体中用到的文字类型必须与条件变量的数据类型一致。下例中，PAYROLL-PERIOD是条件变量。与其有关的字形描述使88层条件名数值限制为一位数字。

```
02  PAYROLL-PERIOD PICTURE IS 9.
    88  WEEKLY VALUE IS 1.
    88  SEMI-MONTHLY VALUE IS 2.
    88  MONTHLY VALUE IS 3.
```

用以上描述，可以写出如下的过程条件名测试：

```
IF MONTHLY GO TO DO-MONTHLY .
```

它等价于：

IF PAYROLL-PERIOD = 3 GO TO DO-MONTHLY.

注意：对编辑基本项，条件名描述体的值，必须以非数值文字的形式表示。

VALUE 子句既不包括文字序列，也不包括文字值域。

3.14 文件节，FD 描述体（对顺序I-O）

在数据部的文件节中，对于每一个选定的文件都必须对应一个文件描述体。这个描述体放在文件记录结构描述的前面。

文件描述体 FD 的一般格式是：

FD 文件名 LABEL 子句〔VALUE-OF 子句〕
〔DATA-RECORD(S)子句〕〔BLOCK 子句〕〔RECORD 子句〕
〔CODE-SET 子句〕〔LINAGE 子句〕.

在“FD 文件名”之后，各子句次序无关。

3.14.1 LABEL 子句

FD 描述体要求的标号子句格式是：

$$\underline{\text{LABEL}}\ \left\{\begin{matrix}\underline{\text{RECORD}}\\ \underline{\text{RECORDS}}\end{matrix}\right\}\ \left[\begin{matrix}\text{IS}\\ \text{ARE}\end{matrix}\right]\ \left\{\begin{matrix}\underline{\text{OMITTED}}\\ \underline{\text{STANDARD}}\end{matrix}\right\}$$

OMITTED任选指出：该文件不存在标号。对指定给PRINTER 的文件，必须给出该任选。

STANDARD 任选指出：该文件有标号，且它与系统规定一致，对于磁盘文件必须指定该任选。

3.14.2 VALUE OF 子句

对于文件描述体中所有分配给磁盘的文件都要用到VALUE OF子句。该子句含有一个用最多16个字符非数值文字表示的文件名，或者用一个数据名表示的文件名。这个文件名按照操作系统中文件名的组成规则组成。文件名中不能含有空格字符。如果指定一个数据名，那么数据名含有的文件名可以含有任意多的字符，但它必须要用一个空格字符结束。其一般格式是：

$$\underline{\text{VALUE}}\ \underline{\text{OF}}\ \underline{\text{FILE-ID}}\ \text{IS}\ \left\{\begin{matrix}\text{数据名}\\ \text{文　字}\end{matrix}\right\}$$

例：

```
VALUE OF FILE-ID  "A:MASTER.ASM"      (CP/M)
VALUE OF FILE-ID  "EMPLOY/DAT:2"      (TRSDOS Model II)
VALUE OF FILE-ID  ":F1:INVNT.LST"     (ISIS-II)
```

注意：如果把一个文件指定给 PRINTER，则该文件不能用标号，而且在有关的文件描述体FD中也不能含有 VALUE 子句。如果把一个文件指定给磁盘，那么在有关的文件描述体FD中，必须用到LABEL RECORDS STANDARD 和 VALUE 子句。有关文件名的格式见第一章 “COBOL-80用户指南”。

3.14.3 DATA RECORD(S) 子句

DATA RECORDS 子句用名来标识文件中的若干记录。但在这个 COBOL 系统中，该子句仅用于文本中，其一般格式是：

DATA { RECORD IS / RECORDS ARE } 数据名1〔数据名2…〕

出现一个以上数据名表示：该文件包括一种以上的数据记录类型。即：两个或两个以上的记录描述可用于同一个存储区。格式中数据名所列出的次序无关紧要。

这里数据名1，数据名2等是数据记录的名字。在记录描述体中每个名字之前必须出现层号01，其后是文件节中的合适的文件描述（FD）。

3.14.4 BLOCK 子句

BLOCK CONTAIN 子句用来指定与逻辑记录有关的物理记录的特性。它的一般格式是：

BLOCK CONTAINS 整数2 { CHARACTERS / RECORDS }

分配给PRINTER的文件在有关的文件描述体中一定不能用BLOCK 子句描述，并且在本COBOL 系统中，BLOCK 子句对于磁盘文件无效。但是，要对它进行语法检查。一般它也适用于磁带文件，但本 COBOL 系统不支持此功能。

当用到BLOCK子句时，通常用RECORDS 来说明物理块的大小；但如果记录的长度是可变长的，或者超出物理块的长度，这时，物理块的大小应该用 CHARACTERS表示。

当忽略BLOCK CONTAIN子句时，则假设记录是不分块的。如既不选择 CHARACTERS，也不选择 RECORDS，则假定缺省任选为 CHARACTERS。当使用 RECORDS 任选时，编译系统认为一个物理块可以存放整数 2 所指出的那么多个记录，并且为任何需要的控制字符提供了附加的存储空间。

3.14.5 RECORD 子句

因为每个数据记录的大小完全由组成01层记录说明的一系列数据描述体所确定，因此该子句总是可选择的，并且作为文件说明。

句子的一般格式是：

RECORD CONTAINS 〔整数1 TO〕整数2 CHARACTERS

整数2是文件说明中最大记录的长度。如果记录是可变长的，那么必须指定整数1，而且它指出的是最小记录的长度。这里记录的大小是要求存放逻辑记录的字符位数。

3.14.6 CODE-SET 子句

子句的格式如下：

CODE-SET IS ASCII

这个子句只用于非海量存储的文件，不过在本编译系统中，它仅作为文件的说明，指出不论是内部数据还是外部数据都是用ASCII码来表示的。然而，在这种文件的记录中，任何带正负号的数值数据描述体都应当含有 SIGN IS SEPARETE 子句，并且 该文件中的所有数据都应该采用 USAGE IS DISPLAY。

3.14.7 LINAGE 子句

LINAGE 子句可以为分配 给 PRINTER 的文件提供一种手段——即指定一页内的打印区，通称为“页体”。可指定页体内行数，也可选择地指出上下边缘的大小以及页脚区开始于页体内的行数。其一般格式是：

LINAGE IS { 数据名1 / 整数1 } LINES,〔WITH FOOTING AT { 数据名2 / 整数2 }〕

〔LINES AT TOP { 数据名3 / 整数3 }〕〔LINES AT BOTTOM { 数据名4 / 整数4 }〕

其中，所有的数据名必须引用无符号的整型数字数据项。整数1必须大于零，且整数2不能大于整数1。

总的页大小是除FOOTING之外的每段数值之和。如果不指定边缘值TOP和BOTTOM，则缺省值为零。页脚区包括FOOTING值所指出的行和页体最后一行之间的部分。

在打开文件时（通过执行一个OPEN OUTPUT语句），以上每个短语的值相应地指出了第一个逻辑页中每段的行数。每当执行一个带有ADVANCING PAGE短语的WRITE语句，或者发生“页溢出”条件时（见WRITE语句），则使用以上每个短语的值指定下一个逻辑页中每一段的行数。

通过使用LINAGE子句可建立一个LINAGE-COUNTER计数器。LINAGE-COUNTER计数器的值表示打印机定位于当前页体内的行号。可以引用这个计数器的值，但不能由过程部的语句修改。它只能按照如下规划，在WRITE语句执行时自动地改变。

1．当WRITE语句中指定“ADVANCING PAGE”短语，或者“页溢出”条件发生时，LINAGE-COUNTER计数器被置成1。

2．当指定“ADVANCING标识符或整数”短语时，则计数器增加ADVANCING的值。

3．若没有指定ADVANCING短语，则LINAGE-COUNTER增加值1。

有关LINAGE说明的作用见WRITE语句的描述。

3.15 工作存储节

数据部的第二节要用标题

WORKING-STORAGE SECTION.

开头。

这节所描述的数据和记录不是外部数据文件的一部分，而是在数据文件内部形成和处理的。

工作存储节中的数据描述体可以应用层号01-49以及层号77。在文件节中禁止使用的赋初值语句（88层除外），在工作存储节是允许的。

3.16 连接节

数据部的第三节以如下标题开始：

LINKAGE SECTION.

本节中，用户用名和属性来描述数据，但并不分配其存储空间。用过程部头的USING表机构，这些“空”描述适用于这样一些数据，这些数据所在单元的地址在一个单独编译的程序调用子程序时被传送给该子程序。除88层条件名描述体外，在连接节禁止使用VALUE子句。详细说明参考第五节“程序间通讯”。

3.17 屏幕节

数据部的第四节用来定义CRT屏幕格式。这一节由屏幕数据描述体组成。

如同在文件节和工作存储节中一样，通过分配合适的层号可将描述分类。于是，有两类屏幕描述项。基本屏幕项在屏幕轮廓内定义了每个显示域和一些数据录入域。组合屏幕项用来命名一组基本屏幕项，以便可以用单个的过程部语句接收(ACCEPT)和显示 (DISPLAY)它们。组合屏幕描述体的格式是：

层号　屏幕号 〔AUTO〕〔SECURE〕

其中，层号必须是0到49之间的一个整数。屏幕名的使用规则与1.3节中给出的名字形成规则一致。组合屏幕描述体之后必须跟以若干个从属屏幕项，这通过增大层号来指出。如果组合屏幕项下写了AUTO和SECURE，其效果等价于为每一个从属于它的基本屏幕项写出AUTO或SECURE。

基本屏幕项的格式如下：

层号〔屏幕名〕
〔BLANK SCREEN〕
〔LINE NUMBER IS 〔PLUS〕 整数1〕
〔COLUMN NUMBER IS 〔PLUS〕 整数2
〔BLANK LINE〕
〔BELL〕
[{ HIGHLIGHT / BLINK }]
〔〔VALUE〕 IS 文字1〕

[{ PICTURE / PIC } IS 形象符号串 { 〔FROM {文字2 / 标识符1}〕〔TO 标识符2〕 / 〔USING 标识符3〕 }]

〔BLANK WHEN ZERO〕
{ JUSTIFIED / JUST } RIGHT〕
〔AUTO〕
〔SECURE〕

其中，层号和屏幕名的使用规则与组项屏幕数据描述的规则一致。基本屏幕数据描述体中的子句次序是无关紧要的，但是，如果用到屏幕名，则它必须直接跟在层号之后。如果用到PICTURE，那么或者要用到USING，或者要用到FROM或TO。屏幕项既可以有FROM，也可以有TO子句。仅当指定PICTURE时，才可以使用AUTO、SECURE、BLANK WHEN ZERO和JUSIFIED子句。基本屏幕项的最大长度是80个字符。

当运行到ACCEPT和DISPLAY语句时，基本屏幕的数据描述子句可能影响数据输入和数据显示操作。所指定的子句效果如下：

1. BLANK SCREEN 清除整个屏幕，并使光标置于初始位置（第1行、第1列）。

2. LINE 和 COLUMN 影响与基本屏幕项有关的屏幕位置。当编译处理到 SCREEN SECTION 时，则保持当前的光标位置，以便可以用具体的屏幕区标识每个基本屏幕项。当

遇到01层的屏幕项时，将光标位置重置在第1行第1列。然后，每当处理一个基本屏幕数据描述体，则根据定义调节它们的位置。因此，连续定义的字段作为缺省值从一端到另一端出现在CRT屏幕的连续区域上。利用LINE和COLUMN规定，可以改变任何基本屏幕数据描述体开始时指定的当前位置。如果既不写LINE，也不写COLUMN，则不改变当前的屏幕位置。如果只写了COLUMN不写LINE，则不调整当前屏幕上行的位置。如果只写了LINE而不写COLUMN，则假定为第一列。不带PLUS的LINE整数和COLUMN整数子句使得指定的整数取为当前屏幕项开始的行、列位置。LINE PLUS整数或COLUMN PLUS整数子句使得指定整数加到当前屏幕项的行、列位置上，结果作为当前屏幕开始的行、列位置。如果给出LINE或COLUMN，而未给出整数1或整数2，则假定它为LINE PLUS 1或COLUMN PLUS 1。

3. BLANK LINE子句清除从当前光标位置到当前行结束的屏幕区域，而不改变光标位置。

注意：下列功能总是按如下次序执行，而不考虑它们在程序中出现的次序：

1）BLANK SCREEN

2）LINE/COLUMN定位

3）BLANK LINE

4）显示（DISPLAY）或接收（ACCEPT）操作

4. BELL使终端音响报警器在系统接收域准备好接收时发出声音。BELL对输出域没有作用。

5. HIGHLIGHT和BLINK是同义的。它们使DISPLAY屏幕项通过闪灼、高亮度或反视频显示出现在CRT屏幕上。也可以用特殊类型的终端硬件提供的其他醒目显示方法。但它对输入域毫无作用。

6. 当用DISPLAY语句引用定义的屏幕项时，VALUE IS文字1明确地指定了要在屏幕上显示的字符串。文字1必须用引号括住，并且它不能是象征常数。所有的ACCEPT语句都忽略指定初值的那个屏幕项。

7. PICTURE子句指定数据出现在屏幕上的格式。它按照3.2节工作存储节中描述的PICTURE子句的规则书写。在DISPLAY语句执行期间，FROM或USING域的内容在屏幕上显示前，根据指定的PICTURE被传入一个隐含的临时项中。而在ACCEPT语句执行时，将录入域显示的内容加上标点符号，以便使它与PICTURE格式一致。

8. FROM、TO和USING描述了文件节、工作存储节以及连接节中屏幕项、文字和字段之间的关系。依照屏幕项的显示，使得FROM或USING文字或字段被传送到屏幕项的PICTURE定义的临时项中。然后使临时项的结果内容展示在屏幕上。而在接收该屏幕项时，运行时系统隐含地把接收的数据传送到对该项所指定的任何TO或USING域中。

9. 如果屏幕项值为零，则BLANK WHEN ZERO使用空格显示屏幕项。

10. JUSTIFIED和JUST子句指出：在屏幕上显示操作员键入的数据，或来自FROM、USING域的数据或文字时，以屏幕项的右界对齐。

11. AUTO指出：当用户的输入充满一个域时，光标自动地跳到下一个输入域，而并不等待打入最后的字符。当无剩余的输入域时，ACCEPT过程结束。

12. SECURE抑制输入字符的回送。而对于每个接收了的数据字符显示一个星号。

3.18 数据部的限制

在数据部的工作存储节、连接节和文件节中对数据项的个数有一个限制。本COBOL-80系统有通讯级I的功能，通讯数据CD的个数也是有限的，其和为：

$$\frac{W+4095}{4096}+F+L+C$$

这个和必须要小于等于14。其中W是工作存储节的字节个数，F是文件节中所描述的文件个数，L是连接节中01层或77层描述体个数，C是通讯节中通讯数据CD的个数。在同一个运行单位（与任意多个子程序连接在一起的主程序）中最多可以打开14个文件。

第四节 过程部

本节讨论过程部的基本概念。一些更深入的内容，如表索引、索引文件存取、程序间通讯及说明，在后面章节叙述。

4.1 语句、句子、过程名

源程序的过程部分说明那些执行特定数据处理功能所需的过程。这些步骤（计算、逻辑判定等等）以类似于英语的语句来表达：用动词表示动作，用语句和句子描述过程。过程部分必须以下列标题开始：

PROCEDURE DIVISION

一个语句包含动词，后跟适当的操作数（数据名或文字）以及构成完整的语句所必须的其他字。

有两种类型的语句：命令语句和条件语句。

命令语句 (Imperative Statements)

命令语句指示目标程序执行无条件动作。一个命令语句由动词及其操作数组成，不包括IF和SEARCH条件语句以及包含了INVALID KEY, AT END, SIZE ERROR,OVERFLOW或ON ESCAPE子句的任何语句。

条件语句 (Conditional Statements)

条件语句规定一个测试条件，确定选择程序流程中哪一条路径。IF和SEARCH语句提供了这种能力。包含INVALID KEY或AT END子句的输入输出语句也被认为是条件语句。当一个算术语句带有SIZE ERROR后缀时，也被认为是条件语句，而不是命令语句。包含OVERFLOW子句的STRING或UNSTRING语句，包含ON ESCAPE子句的ACCEPT语句，也是条件语句。

句子 (Sentences)

句子是以句号后跟一个空格结束的一条语句或一系列语句，如果需要，句子的各语句之间可用分号或逗号分隔开。

段 (Paragraphs)

段是由零个、一个或多个句子组成的逻辑实体，每一段必须以段名开始。

节 (Sections)

节由一个或多个连续的段组成，必须以节头开始。节头由节名后跟SECTION和一个任选的分段号及句号组成，节名与过程名组成规则相同。节头必须自成一行，每个节名都必须

是唯一的。

4.2 过程部组织

程序的过程部分可以三种方式划分：

1．过程部只包含段。

2．过程部包含零个或多个段，后面跟以若干节（每一节又划分为一个或多个段）。

3．过程部包含说明部分和一系列的节（每一节又划分为一个或多个段）。

过程部的说明部分是任选的，它提供了I/O错误事件中指定被调用过程的手段。如果使用说明部分，则只能使用上述第三种可能性。说明部分将在第九节详细讨论。

4.3 MOVE语句

MOVE语句用于从主存一个域传送数据至另一个域，并且对被传送数据进行转换和(或)编辑。MOVE语句格式如下：

MOVE { 数据名1 / 文字 } TO 数据名2[数据名3 …]

由数据名1表示的数据或指定的文字被传送到由数据名2给定的域中。还可以指定另外的接收域（数据名3等）。当接收域为组项时，传送字符不考虑该组项包含的层结构，且不进行编辑。

与数据名2相关的下标和索引，在数据被传送到接收域之前求出其值。对其他接收域(如数据名3等）也是如此。但对于源域和与数据名1相关的下标和索引，仅在数据被传送之前求值一次。

为进一步说明，请看语句：

```
MOVE  A(B)  TO  B, C(B)
```

该语句等价于：

```
MOVE  A(B)  TO  temp
MOVE  temp  TO  B
MOVE  temp  TO  C (B)
```

其中，temp是由编译程序自动指定的中间结果域。

下列考虑适用于传送项：

1．数字（外部或内部十进制、二进制、数值文字或零）或字母数字数据传送到数字或报表数据项：

a．传送项按十进制小数点对齐，必要时两端补零或截断。如果源项为字母数字型，则被看作无符号整数，长度不应超过31个字符。

b．当源字段和接收字段类型不一样时，转换为接收字段类型。字母数字源项被看作无符号整数（USAGE DISPLAY）。

c．可对这些项进行特殊编辑：消去无用零，插入美元符，以及十进制小数点对齐，如接收域所指定等。

d．不能将PICTURE描述为字母型或字母数字型的项传送到数字项或报表项中。也不能将任何一种数字项传送到字母项中，虽然整型数字项或报表型数字项可以经过或不经编辑被传送到字母数字项中，但是，在这种情况下，即使指定了“SIGN IS SEPARATE”，运算符号也并没有传送。

2．非数字型源和目的地：

a．除非指定JUSTIFIED RIGHT，否则字符从左至右放入接收域。

b．如果被传送数据没有完全填满接收域，则其余位置以空格填充。

c．如果源域长于接收域，则一旦接收域被填满，即结束传送。

3．如果包含重迭域，则结果不能预料。

4．附录Ⅱ以表格形式说明源字段和接收字段类型所有可能的组合。

5．一个USAGE IS INDEX的项，不能作为MOVE语句的操作数。

数据传送实例（b代表空格）：

源字段		接收字段		
PICTURE	值	PICTURE	MOVE 前的值	MOVE 后的值
99V99	1234	S99V99	9876-	1234+
99V99	1234	99V9	987	123
S9V9	12-	99V999	98765	01200+
×××	A2B	×××××	Y9×8W	A2Bbb
9V99	123	99.99	87.65	01.23

4.4 INSPECT语句

INSPECT语句帮助程序员检查字符串项，可以选择下列动作的各种不同组合：

1．指定字符串出现计数。

2．用指定字符串替换另一字符串。

3．要求出现另一个特定字符串来限定上述动作。

INSPECT语句格式如下：

INSPECT 数据名1 [TALLYING 子句] [REPLACING 子句]

其中TALLYING子句格式为：

TALLYING 数据名2 FOR { CHARACTERS / ALL | LEADING 操作数3 }

[BEFORE | AFTER INITIAL 操作数4

REPLACING 子句格式为：

REPLACING { CHARACTERS / ALL | LEADING | FIRST 操作数5 }

BY 操作数6 [BEFORE | AFTER INITIAL 操作数7]

INSPECT把数据名1看作字符串，因此，数据名1必须描述为USAGE IS INDEX，COMP或COMP-3，数据名2必须为数字数据项。

在上述格式中，操作数n可以是长度为1的、括在引号内的文字，或者有效位为单个字符的象征常数，或者长度为1的项的数据名。

TALLYING子句和REPLACING子句不能同时都略去，如果两者都出现，则TALLYING子句必须在前面。

TALLYING子句对数据名1从左至右逐个字符进行比较，每找到一个匹配字符，则对数据名2增值1。如果有“AFTER INITIAL操作数4”分子句，则仅当检测到数据名1中字符与操作数4匹配时，才开始计数过程。如果指定“BEFORE INITIAL操作数4”，则当遇到数据名1中字符与操作数4匹配时，结束计数过程。

REPLACING子句在该子句规定的条件下，从左至右进行字符替换。如果指定“BEFORE INITIAL操作数7”，则当检测到数据名1中字符与操作数7匹配之后，不再继续进行替换。如果指定“AFTER INITLAL操作数7”，则直至检测到数据名1中字符与操作数7匹配时，才开始进行替换。

对数据名1的替换，取决于TALLYING和REPLACING对下列指定字符的处理：

1. “CHARACTERS”意指对数据名1界限内的每一个字符均进行计数或替换。
2. “ALL操作数”意指数据名1界限内与“操作数”匹配的字符均参与计数或替换。
3. “LEADING操作数”意指仅数据名1界限内最左位置起与“操作数”匹配的连续字符（如若干个前零）参与计数或替换。
4. “FIRST操作数”意指仅首次遇到的与“操作数”匹配的字符才参与替换（该任选不能用于TALLYING中）。

当TALLYING和REPLACING两个子句均出现时，这两个子句的作用仿佛是写两条INSPECT语句，一条仅包含TALLYING子句，另一条仅包含REPLACING子句。

不断求出TALLYING值，最后数据名2的结果等于累计数加上数据名2初值。在下面第一个例子中，假定在该程序的其他地方已将COUNTX项初始值设置为零。

```
INSPECT ITEM TALLYING COUNTX FOR ALL “L”
REPLACING LEADING “A” BY “E” AFTER INITIAL “L”
原来 (ITEM)：SALAMI ALABAMA
结果 (ITEM)：SALAMI ALEBAMA
最后 (COUNTX)：1        1
INSPECT WORK-AREA REPLACING ALL DELIMITER
BY TRANSFORMATION
原来 (WORK-AREA)：          NEW YORK N Y (长度16)
原来 (DELIMITER)：          (空格)
原来 (TRANSFORMATION)：     . (句点)
结果 (WORK-AREA)：          NEW.YORK..N.Y...
```

注：如果数据名1或操作数n被描述为有符号数字，则在处理时把它当作无符号数字。

4.5 算术语句

算术语句有五种：ADD，SUBTRACT，MULTIPLY，DIVIDE和COMPUTE。任何一种算术语句既可以是命令语句，也可以是条件语句。当一个算术语句包含ON SIZE ERROR说明时，整个语句就称为条件句，因为SIZE ERROR条件和数据相关。

下面是条件算术语句实例：

```
ADD 1 TO RECORD-COUNT
ON SIZE ERROR MOVE ZERO TO RECORD-COUNT
DISPLAY “LIMIT 99 EXCEEDED”.
```

如果发生容量错（在本例中，很显然，RECORD-COUNT 描述为 PICTURE 99，不能容纳值100），则执行 MOVE 和DISPLAY 两条语句。

可以出现在算术语句中的三个语句成分（GIVING 任选、ROUNDED 任选和 SIZE ERROR 任选）将在后面详细讨论。

算术语句基本规则为：

1．算术语句中使用的所有数据名必须是在程序的数据部中定义的基本数字数据项。GIVING任选的操作数例外，可以是报表（数字编辑型）项。在这些算术语句中，不允许索引名和索引数据项。

2．在计算过程中，自动对齐十进制小数点。

3．算术表达式求值产生的中间结果字段能保证结果字段的精度，在须高位截断时例外。

4.5.1 SIZE ERROR 任选

在十进制小数点对齐，低位舍入之后，如果计算结果的绝对值超出接收域所能容纳的最大值，则产生容量溢出。

任选的 SIZE ERROR 子句紧接着写在任一算术语句后面，作为该语句的扩充。SIZE ERROR 任选的格式为：

ON SIZE ERROR 命令语句…

如果有 SIZE ERROR 任选，且发生容量溢出情况，则结果数据名的值不改变，并执行对该条件所指定的命令语句序列。

如果未指定 SIZE ERROR 任选，且发生容量溢出情况，则对最后结果不作任何假定。

一个算术语句如果带有 SIZE ERROR 任选，则不是命令语句，而是一个条件语句，禁止在只允许出现命令语句的地方使用。

4.5.2 ROUNDED 任选

在十进制小数点对齐之后，如果结果的小数位数大于存放计算结果的数据项小数部分的位数，则发生截断，除非指定了 ROUNDED 任选。如果指定了 ROUNDED 任选，则当剩余位最高位有效数字大于或等于 5 时，将结果数据名最低位有效数字的值加 1 。

计算结果为负时，按计算结果的绝对值进行舍入处理，然后将最终结果变为负。

下表描述了计算结果与接收项的值之间的关系，列出带舍入和不带舍入两种情况。

当结果标识符中低位整数位在 PICTURE 中用字符 P 表示时，舍入和截断相对于存储器容许的最左整数位进行。

	接收计算结果的项		
计算结果	PICTURE	舍入后的值	截断后的值
-12.36	S99V9	-12.4	-12.3
8.432	SV9	8.4	8.4
35.6	99V9	35.6	35.6
65.6	S99V	66	65
.0055	SV999	.006	.005

4.5.3 GIVING 任选

如果在算术语句中写了GIVING任选，则跟在GIVING后面的数据名值等于该算术运算的计算结果。跟在GIVING后的数据名并不参与计算，因此可是报表（数字编辑）项。

4.5.4 ADD 语句

ADD语句将两个或多个数字值相加，存放结果和,其语句格式如下：

```
ADD  {数值文字}  ...  {TO    }  数据名n
     {数据名1}       {GIVING}
〔ROUNDED〕〔SIZE ERROR 子句〕
```

当使用TO任选时，语句中所有数据名（包括数据名n）和文字的值均被相加，结果和取代数据名n的值。当使用GIVING任选时，ADD和GIVING之间至少必须有两个数据名和（或）数值文字。将这些数据名和文字的值相加（不包括数据名n），结果和取代数据名n的值。

下面是合法的ADD语句实例：

```
ADD INTEREST, DEPOSIT TO BALANCE ROUNDED
ADD REGULAR-TIME OVERTIME GIVING GROSS-PAY
```

第一个语句将INTEREST、DEPOSIT和BALANCE相加，结果放入BALANCE中；而第二个语句将REGULAR-TIME和OVERTIME相加，结果放入GROSS-PAY中。

4.5.5 SUBTRACT 语句

SUBTRACT语句从指定的数据项中减去一个或多个数字数据项，存放结果差。其语句格式如下：

```
SUBTRACT  {数据名1}...FROM
          {数值文字}

          {数据名m〔GIVING 数据名n〕}
          {数值文字m GIVING数据名n }
          〔ROUNDED〕〔SIZE-ERROR 子句〕
```

SUBTRACT语句的效果是将FROM之前的所有操作数值求和,然后从FROM后面数据项的值中减去这个和。

如果指定GIVING任选，则结果（差）存放在数据名n中,否则，结果存入数据名m。

4.5.6 MULTIPLY 语句

MULTIPLY语句将两个数字数据项相乘，存放结果积。其语句格式如下：

```
MULTIPLY {数据名1  }
         {数值文字1}
      BY {数据名2〔GIVING 数据名3〕}
         {数值文字2 GIVING 数据名3 }
   〔RUUNDED〕〔SIZE-ERROR子句〕
```

当略去GIVING任选时，第二个操作数必须为数据名；结果积代换数据名2的值。例如，下列语句：

```
MULTIPLY 1.03 BY BALANCE
```

计算出新的BALANCE值。

4.5.7 DIVIDE 语句

DIVIDE 语句将两个数字值相除，存放结果商。其语句格式如下：

DIVIDE {数据名1 / 数值文字1} {BY / INTO} {数据名2 / 数值文字2}

〔GIVING 数据名3〕〔ROUNDED〕〔SIZE-ERROR 子句〕

BY 形式表明第一个操作数（数据名1或数值文字1）是被除数（分子），第二个操作数（数据名2或数值文字2）是除数（分母）。在这种情况下，如果不写 GIVING，则第一个操作数必须为数据名，且商存放于其中。

INTO 形式表明第一个操作数是除数，第二个操作数是被除数。在这种情况下，如果不写 GIVING，则第二个操作数必须为数据名，且商存放于其中。

被零除总是产生容量溢出情况。

4.5.8 COMPUTE 语句

COMPUTE 语句计算一个算术表达式，将结果存放于指定数字项或报表（数字编辑）项中。其语句格式为：

COMPUTE 数据名1〔ROUNDED〕… =

{数据名2 / 数值文字 / 算术表达式} 〔SIZE-ERROR 子句〕

下面是 COMPUTE 语句的一个实例：

```
COMPUTE GROSS-PAY ROUNDED = BASE-SALARY*
    (1 + 1.5* (HOURS - 40)/40).
```

算术表达式是数值文字、数据名、算术运算符和括号的适当组合。通常，在一个算术表达式中，数据名必须指定数字数据。连续的数据名（或文字）之间必须用一个算术运算符分隔开，且算术运算符两边必须有一个或多个空格。运算符有：

+ 加
- 减
* 乘
/ 除
* * 整数幂

当对给定变量或项执行多次运算时，其优先顺序是：

1. 一元（只含一个变量）加、减
2. 幂
3. 乘、除
4. 加、减

可用括号改变运算顺序，首先对括号内的表达式求值，括号可以嵌套任意多层。如下述表达式：

A + B/(C - D * E)

其运算顺序如下：

1. 计算 D 与 E 的乘积，中间结果 R_1。

2．计算C减R_1，中间结果R_2。

3．将B除以R_2，中间结果R_3。

4．将A与R_3相加，得最后结果。

如果没有括号，表达式变为：

A+B/C-D*E

其运算过程是：

R_1=B/C

R_2=A+R_1

R_3=D*E

最后结果=R_2-R_3

当使用括号时，要遵循下列标点规则：

1．左括号前要有一个或多个空格。

2．右括号后要跟一个或多个空格。

表达式A-B-C等价于(A-B)-C，可允许一元运算符，例如：

```
COMPUTE A= +C+ -4.6
COMPUTE X= -Y
COMPUTE A, B(I)= -C-D(3)
```

4.6 GO TO语句

GO TO语句从程序的一个部分传送控制到另一部分。该语句的一般格式为：

GO TO〔过程名1〕〔过程名2〕…DEPENDING ON 数据名〕〕

其简单形式GO TO过程名1可改变控制通路，使控制转向指定的段或节。如果GO语句不带过程名，则该GO语句必须是一段中唯一的，且在执行前必须用ALTER语句修改（见4.12节）。

该语句的一般形式指定N个过程名，作为N条选择通路（过程名的值从1～N）。否则，没有控制转移，按正常顺序执行。数据名必须为数字型基本项，且不应有位于小数点右边的数位。

如果GO语句（无DEPENDING）出现在命令语句序列中，则它必须在该序列最后。

4.7 STOP语句

STOP语句用于终止或延迟目标程序的执行。该语句格式为：

STOP{RUN / 文字}

“STOP RUN”终止程序的执行，关闭所有文件，控制返回操作系统。如果用在命令语句序列中，则它必须为该序列的最后一个语句。

“STOP 文字”这种格式将指定的文字显示在控制台上，并暂停执行。仅当操作员干预后，才恢复程序执行。可以预计，操作员在按回车键恢复程序执行之前，要执行该文字内容提示的功能。

4.8 ACCEPT 语句

ACCEPT语句用于处理程序运行时获得小量的输入。可有四种格式：

格式1：

ACCEPT 标识符1 FROM { DATE | DAY | TIME | LINE NUMBER | ESCAPE KEY }

格式2：

ACCEPT 标识符2

格式3：

ACCEPT 位置说明标识符3 〔WITH { SPACE-FILL | ZERO-FILL | LEFT-JUSTIFY | RIGHT-JUSTIFY | TRAILING-SIGN | PROMPT | UPDATE | LENGTH-CHECK | AUTO-SKIP | BEEP } ...〕

格式4：

ACCEPT 屏幕名〔ON ESCAPE 命令语句〕

上述ACCEPT 语句的每一种格式的功能是从程序的外部源获得数据，并将它放在一个指定的接收域或一组接收域中。这些格式的主要区别是指定与之接口的数据源不同。格式1从系统定义的数据项获得信息，其他三种格式都是接收操作员从系统控制台键入的数据。对格式2，控制台设备假定为电传打字机或滚动方式的CRT 终端；对格式3，假定输入设备是视频终端，且不带滚动。格式4在终端操作完成数据项录入后，以整个数据描述体形式接收（如在 SCREEN SECTION 中定义）。注意，对于格式2、格式3或格式4 ACCEPT 语句，用通常的CRT 终端作为输入设备是适宜的，尽管对于屏幕格式的影响可能不同。格式3 ACCEPT 语句的各种WITH 任选短语作用在4.8.3.3节中列出。

4.8.1 格式1 ACCEPT语句

使用格式1 ACCEPT 语句，可在执行时获得系统定义的几种数据项。

系统定义的数据项格式为：

DATE——六位数字值，形式为：YYMMDD（年、月、日），例如1983年3月21日是830321。

DAY——五位数字儒略日期，形式为 YYNNN。其中 YY 是年份的两位低位数字，NNN 是一年中的日期号，1～366。

TIME——八位数字值，形式为 HHMMSSFF。其中 HH 为小时，00～23；MM为分，00～59；SS 为秒，00～59；FF表示百分之一秒，00～99。

LINE NUMBER——二位数字值，表示程序当前运行在哪一行。在 COBOL-80 系统中，LINE NUMBER 值总为00。

ESCAPE KEY——二位数字码，由结束最近执行的格式3或格式4 ACCEPT 语句的键产生。标识符1，可用以询问确定打入了哪个键。打入下列任一个键，可终止输入，且使得ESCAPE KEY 值被设置如下：

标记返回（只终止格式3 ACCEPT）	99
换码	01
域终结符（用格式4 ACCEPT时的最后一个域）	00
功能键	02～nn

对所用终端的所有键码均在 CRT 驱动程序中定义（参看第一章附录 A)。在大多数终端上，标记退回键可以是 CONTROL-B 或∧键，换码键可以是 ESCAPE 或 ALT 键，域终结符可以作为回车、换行、TAB、ENTER、NEW LINE 或CONTROL-I键入，功能键通常是 CONTROL-A、CONTROL-C 和 CONTROL-X,分别产生 ESCAPE KEY 值02、03、04。如输入结束是由于使用了 AUTO-SKIP 的结果，即未打终止符键，则 ESCAPE KEY 值置为00。

标识符1应为无符号数字整数，其长度与系统定义的数据项内容相符。否则，按MOVE的标准规则在接收域（标识符1）中存放源值。

4.8.2 格式2 ACCEPT 语句

格式2 ACCEPT 语句用于从滚动设备（如电传打字机或滚动方式的 CRT终端）接收输入字符串。当执行 ACCEPT 语句时，控制台设备读入输入字符，直至遇到回车为止，然后将一对回车换行符送回控制台。输入数据串包含打回车前（不包括回车）键入的所有字符。

对接收域为字母数字型的格式2 ACCEPT 语句，输入数据串被传送到接收域，与从长度等于该数据串字符数的字母数字域被 MOVE 到接收域中完全一样(亦即，缺省为左对齐，填充空格，右截断；如果接收域描述为 JUSTIFIED RIGHT，则右对齐，左截断)。如果接收域是字母数字编辑型，则被看成等长的字母数字域(如同每个字符的 PICTURE 是'X')，因此无插入编辑。

对接收域为数字型或数字编辑型的格式2 ACCEPT 语句，输入数据串要进行有效性检查，这与接收域的 PICTURE 有关（如果接收域描述为 COMP，则其 PICTURE 视为“S9(5)”）。数字0～9在输入数据串的任何地方均被认为是有效的。

下列情况中，十进制小数点（句号或逗号）是有效的：

1. 在输入数据串中仅出现一次。
2. 接收域的 PICTURE 包含小数位，亦即在假定的小数点（“V”）或实际的小数点（“.”）右边出现“9”、“Z”、“*”或浮动插入符。

运算符号“+”和“-”仅当其作为输入串的第一个或最后一个字符，且接收域的PICTURE包含一个符号指示符“S”、“+”、“-”、“CR”或“DB”时才是有效的。

所有其他字符均认为是无效字符。如果输入数据串无效，则送“INVALID NUMERIC INPUT——PLEASE RETYPE”信息到控制台，并读入另一输入数据串。

当接收到有效输入数据串时，数据被传送到接收域，如同执行 MOVE 语句将具有下列特征的假想发送域传送到接收域：

1. PICTURE 形式S9…9V9…9

2. USAGE DISPLAY

3. 总长度等于输入数据串中数字的数目。

4. 假定的小数点右边位数与输入数据串中小数点右边实际数位相同（如果输入数据串中无小数点，则为零）。

5. 当前内容等于插入到输入数据串中的数字串。

6. 如输入数据串中含字符“－”，则为当前负状态并带符号；否则为当前正状态。

4.8.3 格式3 ACCEPT 语句

格式3 ACCEPT 语句用于从非滚动视频终端接收数据，使用格式3 ACCEPT 语句，必须遵循下列规则：

1. 标识符3引用的数据项长度必须小于或等于1920个字符。

2. 在同一个 ACCEPT语句中不可同时指定SPACE-FILL和 ZERO-FILL 任选。

3. 在同一个ACCEPT语句中不可同时指定 LEFT-JUSTIFY 和RIGHT-JUSTIFY 任选。

4. 如果标识符3描述为数字编辑项，则不能指定 UPDATE 任选。

5. 仅当标识符3描述为基本数字数据项时，才可指定 TRAILING-SIGN 任选。如果标识符3描述为无符号数据项，则 TRAILING-SIGN 任选忽略。

6. 当标识符3为字母数字型或字母数字编辑型时，如果不指定 ZERO-FILL 任选，则假定为 SPACE-FILL 任选；如果不指定 RIGHT-JUSTIFY 任选，则假定为 LEFT-JUSTIFY 任选。

7. 对数字型或数字编辑型标识符3，如不指定 SPACE-FILL 任选，则假定为 ZERO FILL 任选。

4.8.3.1 数据输入域

格式3 ACCEPT 语句位置说明和接收域（标识符3）用于定义控制台视频终端屏幕数据输入域的位置和特征。

1. 数据输入域位置

位置说明格式为：

$$\left(\begin{array}{l}\text{LIN}\left[\left\{\begin{matrix}+\\-\end{matrix}\right\}\text{整数 1}\right]\\ \text{整数 2}\end{array}\right),\left(\begin{array}{l}\text{COL}\left[\left\{\begin{matrix}+\\-\end{matrix}\right\}\text{整数 3}\right]\\ \text{整数 4}\end{array}\right)$$

两组主要的方括号要用开、闭括号括起来，中间用逗号及空格分开。定位参数指定了控制台CRT 屏幕上数据输入域开始位置。LIN 和 COL 是 COBOL专用寄存器。每个定位参数作用如同 USAGE COMP 的数字数据项，但它们可以为每个 COBOL程序引用，而无须在数据部中说明。

如指定LIN，则数据输入域开始于屏幕上行号与LIN 专用寄存器值相等的行，且如果指定“＋整数1”（或“－整数1”），则开始行为行号加（或减）整数1；如指定整数2，则数据输入域开始于行号为整数2的行。如LIN 和整数2均不指定，则数据输入域开始于屏幕上光标当前所在的行。

如指定COL，则数据输入域开始于屏幕上列号与COL专用寄存器值相等的列，且如果指定“+整数3”（或“-整数3”），则开始列为列号加(或减)整数3；如指定整数4，则数据输入域开始于列号为整数4的列。如COL和整数4均不指定，则数据输入域开始于屏幕上光标所在的列。

2．数据输入域特征

数据输入域的特征（除位置外）由接收域的PICTURE描述决定（对于USAGE为COMPUTATIONAL的项，其PICTURE被视为S9(5)）。对字母数字或字母数字编辑型标识符3，数据输入域就是屏幕上由定位参数指定位置开始的数据输入字符串。数据输入域长度（字符位置）等于内存中接收域长度。

对数字或数字编辑型标识符3，数据输入域可包含：整数数位、小数数位、符号位、十进制小数点位。对标识符3 PICTURE中每个“9”、“Z”、“*”、“P”或非起始浮动插入符（浮动插入符是“+”、“-”或“$”）且在PICTURE字符串中非最后一个符号均有一个数位。在一个数据输入域中，如果相应的PICTURE字符是在标识符3的PICTURE中假定小数点（“V”）或实际小数点（“.”）右边，则该数据输入域中每个数字位都是小数字位，否则就是整数字位。如果标识符3描述为有符号的，则有一个符号位，否则没有符号位。如果至少有一个小数字位，则有一个十进制小数点位，否则没有小数点位。

所定义的数据输入位占据了CRT屏幕上连续字符位置，其开始位置由定位参数指定。如在ACCEPT语句中指定了TRAILING-SIGN，则数据输入位顺序为：整数字位、小数点位、小数字位。如未指定TRAILING-SIGN，则数据输入位顺序为：符号位、整数字位、小数点位，小数字位。

4.8.3.2 *数据输入和数据传送*

由终端操作员送入数据输入域的字符，可以看成是编辑字符、终止符键或数据字符。当打入终止符键时，ACCEPT语句终止，ESCAPE KEY键置为4.8.1节所述的值。使用格式1 ACCEPT语句FROM ESCAPE KEY可以询问该值。

编辑字符是行删除、进格、退格和字符删除，在大多数终端上，可分别用control-U、control-F、control-H和DEL（或RUB）打入这些字符。关于编辑和终止符的定义，见第一章“COBOL-80用户指南”。

1．字母数字接收域

首先考虑接收域为字母数字或字母数字编辑型的格式3 ACCEPT语句的执行情况。字母数字编辑型接收域可看成等长的字母数字域，不发生插入编辑。数据输入域的初始外部特性取决于ACCEPT语句中的WITH短语说明。如指定UPDATE，则标识符3当前内容显示在输入域中。在此情况下，所有数据输入位均被看成由终端操作员键入。如未指定UPDATE，但指定了PROMPT，则在每个输入数据位置显示一个句号(“.”)。如果UPDATE和PROMPT均未指定，则数据输入域不变；光标停在第一个数据输入位置，接收的字符与操作员从终端键入的字符一样，直至遇到终止符（通常是回车）。如果在ACCEPT语句中指定AUTO-SKIP，则当操作员在最后一个数据输入位置（最右位置）键入字符时，ACCEPT语句亦终止。

当每个输入字符被接收时，将接收字符回送至CRT屏幕，对非显示字符，则回送“?”。如数据输入域的所有位置均已填满，则终止符或上述编辑字符前的其他输入字符被忽略。如

ACCEPT 语句中指定了 RIGHT-JUSTIFY，则当 ACCEPT 终止时，操作员键入的字符被移到数据输入域的最右位置。在终止时，所有未键入字符的位置均用填充字符填满。填充字符或为空格（如指定 SPACE-FILL）或为零（如指定 ZERO-FILL）。

接收域中的内容与输入域中出现的一组字符相同，但操作员键入字符的对齐受接收字段数据描述中 JUSTIFIED 的控制，而不受 ACCEPT 语句 RIGHT-或 LEFT-JUSTIFY 的控制。接收域的剩余位置用空格还是用零填充，根据 ACCEPT语句中是 SPACE-FILL 还是ZERO-FILL 来指定。

2. 数字接收域

其次考虑接收域为数字或数字编辑型的格式 3 ACCEPT 语句的执行情况。如上所述，在控制台 CRT 屏幕上，数据输入域可包含整数字位、小数字位或同时包含整数位和小数位。先假定为后一种情况。

对于字母数字型 ACCEPT，数据输入域可按 WITH 任选确定的方式进行初始化。若指定 UPDATE 任选（对数字编辑型接收域不允许），该数据输入域的整数和小数部分被置为该接收域初值之十进制表示法的整数和小数部分，必要时可带有前导零和尾随零，以填满所有数字位。除前导零外，其他初始字符均被视为操作员键入的数据。如未指定 UPDATE，但指定了 PROMPT，则在每个输入数字位均显示零。在上述两种情况下 (UPDATE 和 PROMPT)，在十进制小数点位置均显示一个小数点。

如 UPDATE 和 PROMPT 均未指定，则屏幕上输入域除符号位外均不初始化。符号位总是初置为正，除指定 UPDATE 外。当指定 UPDATE 时，符号位根据接收域当前内容的符号置初值。在大多数系统中，正符号位示为空格，负符号位示为减号。

光标开始位于整数字位的最右边，每当操作员键入一个字符时，则接收一个字符。如进入字符是数字，则以前键入的数字在输入域中左移一位，新数字显示在最右整数位上。如果整数字位尚未填满，则光标留在最右数字位上，并接受另一个字符输入。如输入域的整数部分已被填满，且指定了 AUTO-SKIP，则整数部分结束，光标移至最左小数字位。如整数部分已被填满且未指定 AUTO-SKIP，则光标移至十进制小数点位置，且再键入的数字均被忽略，直至整数部分以十进制小数点结束。

如果送入的字符是“+”或“-”，则符号位相应变为正或负状态，光标位置不受影响。

如果送入的字符是十进制小数点，则整数部分结束，且光标移至最左小数字位。

如果送入的字符是域终止符（通常是回车），则 ACCEPT 终止，且光标关闭，再打入任何其他字符均被忽略。

当整数部分结束后，光标移至最左小数字位，操作员键入字符又可被接受。数字回送至控制台，符号“+”或“-”如同在整数部分数字送入时作同样处理。域终止符终止 ACCEPT 语句（如指定 AUTO-SKIP，则整个小数部分被填满时也终止ACCEPT），其他字符被忽略。在小数部分全部数位均被填满后，再送入的数字也被忽略。

如果没有小数位，则小数点作为输入字符被忽略，且可用终止整个 ACCEPT 语句的方法结束整数部分数字的录入；如果没有整数位，则光标初始置于最左小数位置，小数部分数字的录入按上述方法进行。

当结束数字项或数字编辑项的格式 3 ACCEPT 语句时，数据被传送到接收域。在执行 ACCEPT 语句后接收域数据的精确格式，如格式 2 ACCEPT 语句最后一段所述，在该段

中述及的输入数据串的作用由该数据输入域中显示的字符串承担。如果指定了SPACE-FILL，则终结后该数据输入域整数部分中的前零由空格取代，如果前面有运算符，则被移至最右空格位置。

3．编辑字符

编辑字符（行删除、进格、退格和rubout）可用于改变已键入（或由COBOL运行时系统指定WITH UPDATE所产生的）数据。打入行删除字符将导致ACCEPT重新开始，操作员键入的数据或开始出现在接收域的数据全部丢失。如果指定了PROMPT，则控制台屏幕数据输入域重新初始化。否则，数据输入域按SPACE-FILL或ZERO-FILL指定，填充空格或零。

打入进格或退格字符，在字母数字或字母数字编辑型接收域的情况下，将光标向前或向后移动一个数据输入位置；在数字或数字编辑型接收域的情况下，移动一个数位。无论哪一种情况，进格或退格字符都不会在下列位置范围之外移动光标：（1）包含已由操作员键入的位置（或指定WITH UPDATE时COBOL运行时系统填入的位置）；（2）在执行本条ACCEPT语句期间光标已占据的最右数据输入位置。如果光标移动至该范围内除最右位之外的位置上，且送入合法数据字符，则该字符显示于当前光标位置上，且光标向前移动一个数据位（对字母数字或字母数字编辑型）或一个数字位（对数字或数字编辑型）。

打入rubout字符的结果是删去最后一个录入字符。光标往回移一个数据位（或数字位），且在光标位置显示填充字符（空格或零）。当光标在十进制小数点左边时（对数字型ACCEPT），不显示填充字符，且光标不移动，但在光标位置上的数字被删去，其左边的所有数字均向右移动一位。只有当光标在接收新数据字符的位置时，rubout字符才有效。换句话说，如果已经用退格字符将光标回移至已键入数据的位置，则rubout字符无效。

4.8.3.3 WITH短语

下面列出具有字母数字或字母数字编辑型接收域的格式3 ACCEPT语句使用的WITH短语的效果：

1．SPACE-FILL使ACCEPT在终止时，数据输入域和接收域中未键入字符位置填充空格。

2．ZERO-FILL使ACCEPT在终止时，数据输入域和接收域中未键入字符位的置被置为ASCII零。

3．LEFT-JUSTIFY对本编译系统作为注释处理。

4．RIGHT-JUSTIFY使ACCEPT在终止后，操作员键入字符占据屏幕数据输入域的最右位置。注意，传送至接收域中数据的对齐是由接收域数据描述的缺省值或JUSTIFIED说明来控制的，而不是使用WITH RIGHT-JUSTIFY短语控制。

5．PROMPT使屏幕数据输入域在接收输入字符前全部置为句点“．”。

6．UPDATE使数据输入域以接收域的初始内容为初值，且该初始数据被视为操作员键入的数据。

7．LENGTH-CHECK使域终止符被忽略，除非每个数据输入位均已填满。

8．AUTO-SKIP迫使ACCEPT语句在填满所有数据输入位置时终结。通常，键入终止符具有同样的效果。

9．BEEP使ACCEPT开始，当系统准备好接收操作员输入时，音响报警器发声。

下面列出具有数字或数字编辑型接收域的格式3 ACCEPT 语句中WITH短语的效果：

1. SPACE-FILL 使数据输入域中小数点左边的未键入数字位在 ACCEPT 语句终结时填充空格，且前面的运算符显示在最右空格位置。

2. ZERO-FILL 使数据输入域中所有未键入数字位在 ACCEPT 终结时置为零。

3. LEFT-JUSTIFY 和 RIGHT-JUSTIFY 对数字型和数字编辑型接收域无效。

4. TRAILING-SIGN 使运算符出现于数据输入域的最右位置，通常该符号是在输入域最左位置。

5. PROMPT 使数据输入域在接收输入字符之前初置如下：数字位置零；小数点位置十进制小数点字符；符号位置空格。

6. UPDATE 使数据输入域初置为接收域当前内容，且该初始数据被视为操作员键入的数据。

7. LENGTH-CHECK 使接收的小数点字符被忽略，除非已键入所有整数位；且域终止符被忽略，除非已键入所有数字位。

8. AUTO-SKIP 使当已键入所有整数数字位时，ACCEPT 的整数部分终止；当已键入所有数字位时，整个 ACCEPT 语句终止。

9. BEEP 使 ACCEPT 开始，当系统准备好接收操作员输入时，音响报警器发声。

4.8.4 使用格式3 ACCEPT 语句实例

例1：

接收域 05 RS-DISCOUNT PIC X(8). 初始内容： ABCDEFGH ACCEPT 语句： ACCEPT(1, 1)RS-DISCOUNT WITH PROMPT	执行前的设置
ACCEPT 开始时： ..... 操作员打入 N N..... 操作员打入ONE NONE..... 操作员打入回车 NONEbbbb	执行 ACCEPT 语句
接收域最后内容 NONEbbbb	结 果

例 2：

接收域： 10 VEND-NAME PIC X(12). 初始内容： ACME b WIDGETS ACCEPT 语句： ACCEPT(1, 1)VEND-NAME WITH PROMPT UPDATE.	执行前的设置
ACCEPT 开始时： ACMEbWIDGETS （如在这里打入回车，则接收域不变） 操作员打入行删除： ..... 操作员打入 XYZ： XYZ..... 操作员打入回车： XYZbbbbbbbbb	执行ACCEPT语句
接收域最后内容： XYZbbbbbbbbb	结 果

例 3：

接收域： 05 CREDIT PIC S9(4)V99 初始内容： + 111111 ACCEPT 语句： ACCEPT (LIN+4, OCL-3) CREDIT WITH PROMPT TRAILING-SIGN.	执行前的设置
ACCEPT 开始时： 0000.00b 操作员打入 8： 0008.00b 操作员打入 7： 0087.00b	执行ACCEPT语句

操作员打入－： 0087.00- 操作员打入6： 0876.00- 操作员打入N： 0876.00- 操作员打入．： 0876.00- 操作员打入5： 0876.50- 操作员打入回车： 0876.50-	
接收域最后结果： 0876 50	结 果

4.8.5 格式4 ACCEPT 语句

格式4 ACCEPT 语句将来自操作员控制台的信息传送到 SCREEN SECTNIO 幕屏名定义中指定的 TO和（或）USING 字段（或从属于屏幕名的屏幕项）。仅有 VALUE 文字或 FROM 子句的屏幕项对 ACCEPT 语句的操作无效。每个传送隐含对一个域的格式3 ACCEPT 语句，该域由相应屏幕项的 PICTRUE 定义，后面跟以隐含 MOVE 至相关的 TO 或 USING 字段。当 ACCEPT 终结时，ESCAPE KEY 值的设置如下所述（并请参见4.8.1节）。该值可用格式1 ACCEPT 语句 FROM ESCAPE KEY 来询问。字段的接收按照 SCREEN SECTION 中屏幕名下定义的顺序。该顺序可用返回标记键改变，但接收域在屏幕上的位置并不影响该顺序。

如果在数据输入期间打入换码键，则整个 ACCEPT 语句终结，当前字段不传送至相关的 TO 或 USING 项，ESCAPE KEY 值置为01，且执行 ON ESCAPE 语句。如果打入功能键，则设置相应的 ESCAPE KEY 值，整个 ACCEPT 语句终结。如果打入域终止符键（如回车、标记等），则 ESCAPE KEY 值置为00，且光标移至屏幕名下定义的下一个输入域。如果当前域是最后一个域，则整个 ACCEPT 语句结束，如果打入返回标记键，则当前域结束，且光标移至屏幕名下定义的前一个输入域。如果当前域是第一个域，则光标不移动。当一个域由功能键、域终止符键或返回标记键结束时，则当前域内容传送至相关的 TO 或 USING项，除非该域中未进入数据字符和编辑字符。这就使得操作员可以在整个输入域向前或向后移动标记，而不影响接收项的内容。

在4.8.3.2节中对格式3 ACCEPT 语句的所有编辑和有效性检查特性，均适用于格式4 ACCEPT 语句。在3.17节中所列的若干屏幕节说明，对应于格式3 ACCEPT 语句任选：AUTO 对应于 AUTO-SKIP；BELL 对应于 BEEP；JUSTIFIED 对应于 RIGHT-JUSTIFY。并且，如果输入域指定了 USING 子句，或者同时指定了 FROM 和 TO 子句，则ACCEPT 执行带 UPDATE 任选。当执行每条格式3 ACCEPT 语句时，格式4 ACCEPT 语句

总是使用 PROMPT 和 TRAILING-SIGN 任选。

如果屏幕项的 PICTURE 指定数字编辑或字母数字编辑输入域，则 ACCEPT 执行如同该输入域为数字或字母数字型一样。当该输入域结束时，数据按PICTURE 描述进行编辑，且重新显示于指定的屏幕位置。在这种情况下 JUSTIFIED 子句无效。

从屏幕域至接收项的传送，依照标准 COBOL-80 MOVE 语句的规则，所不同的是允许从数字编辑域传送。这时，数据输入如同该域是数字型一样，传送只使用符号、小数点和数值字符。

格式 4 ACCEPT 不产生正文显示或提示标号信息。见下节 “DISPLAY” 语句。

4.9 DISPLAY 语句

DISPLAY 语句提供在运行时输出小量数据的能力而不用进行文件定义。其格式如下：

```
DISPLAY { { [位置说明] { 标识符 / 文字 / ERASE } } ...[UPON 助记名]
        { [屏幕名]                                  }
```

DISPLAY 语句必须按下列规则书写：

1. 标识符引用的数据项长度必须小于等于1920字符。
2. 助记名必须在配置节 SPECIAL-NAMES 段的 PRINTER IS 子句中定义。
3. 屏幕名必须在数据部屏幕节定义。

如果没有指定 UPON 助记名，则 DISPLAY 语句将输出送至系统控制台设备，如果指定了助记名，则输出送至打印机，对每个显示项(即每个标识符、文字、ERASE 或屏幕名)将进行处理，如下所述。如果既无位置说明，也无屏幕名指定，则将回车—换行符送至接收设备。

4.9.1 位置说明

对每个显示项，如果指定了位置说明，则在该项数据传送之前对光标进行定位。位置说明的格式如下：

```
( LIN [ {+/-} 整数 1 ]  ,  COL [ {+/-} 整数 3 ] )
      [       整数 2 ]        [       整数 4 ]
```

对两组主要的方括号，要用左、右圆括号括起来且中间必须用逗号和空格分隔开。位置说明指示光标停留在控制台 CRT 屏幕上的位置。LIN 和 COL 是 COBOL 专用寄存器，每个寄存器的作用类似于带 USAGE COMP 的数字数据项，但它们可由每个 COBOL 程序引用，而不必在数据部中说明。

如果指定 LIN，则光标定位于屏幕上行号等于 LIN 寄存器值的行上，如指定“+整数 1”（或“-整数 1”），则行号加（或减）整数 1。如果指定整数 2，则光标定位于行号等于整数 2 的行上。如果 LIN 和整数 2 均未指定，则光标定位于包含光标当前位置的屏幕行上。

如果指定 COL，则光标定位于屏幕上列号等于 COL 寄存器值的列上。如指定“+整数

3”（或“－整数3”），则列号加（或减）整数3。如果指定整数4，则光标定位于列号等于整数4的列上。如果COL和整数4均未指定，则光标定位于包含光标当前位置的屏幕列上。

4.9.2 标识符、文字和ERASE

如果对于给定的显示项指定了标识符或文字，则标识符内容或文字值被送至接收设备。既然数据传送不发生转换和重格式化，因此建议将数字数据传送到数字编辑域，以便用于显示。

如果对于本显示项或前一个显示项指定了ERASE，且给出了位置说明，则控制台屏幕从当前光标位置至屏幕结束将被清除。对下一个显示项的初始光标位置将是ERASE显示项中位置说明所指定的位置，或者是前一个显示项留下的光标位置。如果指定了ERASE，但到DISPLAY语句的该点上尚未遇到位置说明，则不采取任何动作。

4.9.3 屏幕名

“DISPLAY 屏幕名”语句使信息从屏幕名（或从属于屏幕名的每个基本屏幕项）传送到控制台CRT屏幕。对每个具有VALUE、FROM或USING说明的屏幕项，指定的文字或字段是被显示数据的源。对只有TO子句的字段，其效果如同指定了FROM ALL“.”(句号）一样。源数据隐含地被传送到相应屏幕项PICTURE（或VALUE文字的数据长度）定义的临时项中。然后，按相应屏幕项定义中定位和控制子句的修改，对构成的临时项执行标识符型的DISPLAY。见3.17节。

4.10 PERFORM 语句

PERFORM语句允许执行程序步骤中的每个独立部分。该语句有两种格式：

任选1

PERFORM 范围 [{ 整数 / 数据名 } TIMES]

任选2：

PERFORM 范围 [VARYING { 索引名 / 数据名 } FROM
参量1 BY 参量2 UNTIL 条件.

在上述语法表达式中，假设有如下定义：

1. 范围是段名、节名或结构“过程名1 THRU 过程名2”（THRU与THROUGH同义）。如仅指定段名，则在执行该段最后一个语句后返回；如仅指定节名，则在执行该节最后一段的最后一个语句后返回；如果指定范围，则在执行相应段或节的最后一个句子后控制返回。这些返回点仅当执行PERFORM建立之后才有效。在其他情况下，控制将跳过返回点。

2. 类属操作数参量1和参量2可以是数值文字、索引名或数据名。事实上，这些参量常常是整数或包含整数的数据名，且以指定的数据名用作该范围内的下标。

在任选1中，设定的范围执行固定次数，由整数或整数据项值决定。如果未给出TIMES短语，则该范围只执行一次。当PERFORM完成后，执行PERFORM语句后面的一个语句。

在任选2中，范围执行可变次数，由参量1、参量2和条件决定，从数据名初值等于参

量1开始，以参量2为增量变化，直至满足 UNTIL 后面指定的条件时，执行 PERFORM 语句后面的一个语句。

任选2 PERFORM 中的条件，在每次试图执行该范围之前先求值判断，如果条件不能满足，则不执行该范围。类似地，在任选1中，如果数据名≤0，则该范围根本不执行。

在运行时，如果几个并发执行的 PERFORM 语句的规定范围有同一个终点，则认为是不合法的。

4.11 EXIT 语句

EXIT 语句在有必要提供一个过程结束点处使用。该语句格式是：

EXIT

EXIT 语句必须作为只含一个字的段出现在源程序中，前面为段名，后面跟一个句号。EXIT 段提供一个出口，其前面的语句可跳过一节中的某些部分，控制转移至出口点。

4.12 ALTER 语句

ALTER 语句用于修改过程部其他地方的简单 GO TO 语句，以改变程序语句的执行顺序。该语句的一般格式为：

ALTER 段 TO [PROCEED TO] 过程名

段（第一个操作数）必须是仅包含一简单 GO TO 语句的 COBOL 段；执行 ALTER 语句将该 GO TO 语句以前的操作数代换为过程名。请看下面程序段中 ALTER 语句的前后关系：

```
GATE.       GO TO MF-OPEN.
MF-OPEN.    OPEN INPUT MASTER-FILE.
            ALTER GATE TO PROCEED TO NORMAL.
NORMAL.     READ MASTER-FILE, AT END GO TO EOF-MASTER.
```

从上述程序看出，这是一种“关门”技巧，提供一次起动的程序步，即第一次从GATE转向 MF-OPEN，以后就直接转向 NORMAL。

4.13 IF 语句

IF 语句允许程序员指定当所述条件为真时执行一系列过程语句。如果所述条件为假，可执行另外一系列语句。IF 语句的一般格式为：

IF 条件 {语句序列1 / NEXT SENTENCE} [ELSE {语句序列2 / NEXT SENTENCE}]

语句后面必须立即跟一个句号。

IF 语句实例：

```
1. IF BALANCE = 0 GO TO NOT-FOUND.
2. IF T LESS THAN 5 NEXT SENTENCE ELSE GO TO T-1-4.
3. IF ACCOUNT-FIELD = SPACES OR NAME = SPACES ADD 1 TO
   SKIP-COUNT ELSE GO TO BYPASS.
```

语句序列1仅当条件为真才执行，语句序列2中 ELSE 部分仅当条件为假才执行。不管条件为真还是为假，下一个句子仅在执行相应的语句序列之后才执行。除非在被执行的命令语句中包含 GO TO，或者由于活动的 PERFORM 语句改变了标定的程序流程。

4.13.1 条件

条件可以是简单条件或复合条件。简单条件有四种，这就是：关系条件、分类条件、条件名和符号条件测试。简单关系条件结构如下：

操作数1　　关系　　操作数2

其中，操作数是数据名、文字或象征常数。

复合条件可由任意种类的两个简单条件，用逻辑运算符AND或OR连接而成，例如，A<B OR C=D。参看附录I中其他可允许的形式，包括带括号的条件，NOT和缩写条件。

最简单的“简单关系”有三种基本形式，由关系符等于、小于或大于（即=或<或>）表示。

简单关系的另一种形式可将保留字NOT放在上述三种关系符前。这样就形成了六种简单关系条件，列出如下：

关系	含义
=	等于
<	小于
>	大于
NOT=	不等于
NOT<	大于或等于
NOT>	小于或等于

用保留字AND或OR可指定一系列关系测试：

1. 用AND连接的各个关系说明一个复合条件，仅当所有单个关系均满足，该复合条件才满足（为真）。

2. 用OR连接的各个关系说明一个复合条件，当各关系中任一个满足，则该复合条件满足（为真）。

下面是包含AND和OR连接符的复合关系条件实例：

IF X=Y AND FLAG='Z' OR SWITGH=0 GO TO PROCESSING.

在上例中对不同数据值执行如下：

数据值				是否执行 PROCESSING?
X	Y	FLAG	SWITCH	
10	10	'Z'	1	是
10	11	'Z'	1	否
10	11	'Z'	0	是
10	10	'P'	1	否
6	8	'P'	0	是
6	6	'P'	1	否

使用保留字形成的短语EQUAL TO, LESS THAN, GREATER THAN分别与= < >等价。任何关系形式都可以在前面写上“IS”。

在讨论分类测试、符号测试和条件名条件之前，先讨论一下执行“比较”的方法。

数值比较：数据操作数在对齐十进制位置后进行比较。结果如数学上定义的，任何负值均小于零，零小于任何正值。索引名或索引数据项可在比较中出现。可允许对任意两个数值操作数进行比较，不管其在各自的 USAGE 子句中指定的格式，也不管其长度。

字符比较：允许进行非等长比较，必要时可以补充空格，延长较短项的长度。关系以 ASCII 码定义，从字母A～Z为升序，且数字小于字母。组项在比较时视为字符串。参看附录 IV 中的所有 ASCII 字符表示法。

再来看简单条件，除关系形式外，还有三种简单条件形式：分类测试、条件名测试和符号测试。

分类测试条件语法格式如下：

数据名 IS [NOT] { NUMERIC / ALPHABETIC }

该条件指明对数据项内容进行测试，以确定是否所有字符表示都是正确的。当测试NUMERIC时，不考虑运算符或所有字符是否都是数字表示；当测试ALPHABETIC时，是否只有字母和空格字符。NUMERIC 测试只对组项、十进制项或字符项（不具有字母型 PICTURE）有效；ALPHABETIC 测试只对组项或字符项（字母数字型 PICTURE 格式）有效。

符号测试语法格式如下：

数据名 IS [NOT] NEGATIVE | ZERO | POSITIVE

这种测试等价于将数据名与零比较，以确定所述条件的真实性（是正数、零，还是负数？）。

在条件名测试中，测试条件变量，以确定其值是否等于与该条件名相关的值中的一个。条件名测试的语法格式如下：

条件名

其中，条件名由88层数据部描述体定义。

4.14 OPEN 语句（顺序 I-O）

在进行文件处理之前，必须执行 OPEN 语句。其一般格式为：

OPEN { INPUT / I-O / OUTPUT / EXTEND } 文件名… …

对顺序输入文件，打开操作开始将该文件第一个记录读入内存，以便后继的读语句无须等待就可执行。

对输出文件，打开操作建立一个记录域，用于存放一个记录，在执行写语句时，将该记录传送到指定的输出设备。一个同名文件将被由 OPEN OUTPUT 建立的文件取代。

OPEN I-O 语句仅对磁盘文件有效，它允许使用 REWRITE 语句修改已由 READ 语句存取的记录。WRITE 语句不能以 I-O 方式用在顺序组织文件中。在打开时，文件必须已存在于磁盘中，不能用 OPEN I-O 建立。

当指定EXTEND短语时，OPEN语句将该文件定位于紧接在该文件最后一个逻辑记录之后。访问该文件的后继WRITE语句将记录添加在该文件末尾。因此，进行处理时可认为：该文件已经由OUTPUT短语打开，并定位于其末尾。EXTEND只能用于顺序或行顺序文件。

如果执行OPEN语句进行文件读、写处理失败，则产生执行时错误，将导致程序运行的异常终止。如果一个文件已用“WITH LOCK”关闭(见4.17节“CLOSE语句”)，则该”文件不能打开。

对INPUT或I-O存取打开的顺序文件，必须已按适当格式写入，见第一章“用户指南”中关于这类文件的描述。

4.15 READ语句（顺序I-O）

READ语句将来自指定设备的某文件的下一个逻辑数据记录读入内存，并更新FILE STATUS数据项的值（如果指定的话）。该语句的一般格式为：

READ 文件名 RECORD〔INTO 数据名〕
〔AT END 命令语句序列〕.

因为有时会碰到文件结束，所以应包含AT END子句。保留字END后可跟任意多个命令语句，所有这些语句仅在产生文件结束条件时才执行。AT END语句序列的最后一个语句后面必须有句号，表示句子结束。

如果READ语句中无AT END子句，则碰到文件结束时执行说明过程。如果既无AT END子句，又无说明过程，且对该文件未指定FILE STATUS，则发生运行时错误，程序异常终止。

当被读记录存在，READ语句执行成功，则紧接着执行下一个句子。

如果多个01层项从属于一个文件描述，则这些记录共享同一个存储域。因此，用户必须能够分辨可能的记录类型，以便确定当前哪一种记录可用。这可通过数据比较来实现，用IF语句测试对于每种类型记录有唯一值的字段，以确定记录类型。

INTO任选允许用户指定将数据记录的拷贝除放入文件记录区外，也放入指定的数据域中。INTO后的数据名不能在文件节中定义。

还有，当文件具有各种不同长度的记录时，不应使用INTO短语。数据名的下标和索引，在数据读入之后，但在传送到数据名中之前求出。此后，该数据就可以用于文件记录和数据名了。

对分块输入文件（如磁盘文件），并非每条READ语句都实际执行从外存设备传送的物理数据，而只是从输入缓冲区中读取下一个逻辑记录。

如果实际记录短于文件记录域，则在该文件记录域右边填以空格。

4.16 WRITE语句（顺序I-O）

WRITE语句的一般格式如下：

WRITE 记录名 〔FROM 数据名1〕

〔{AFTER / BEFORE} ADVANCING {操作数 / PAGE} {LINE(S)}〕

[AT { END-OF-PAGE / EOP } 命令语句]

我们先不考虑 ADVANCING 任选，而解释 WRITE 语句的主要功能。

在 COBOL 中，文件输出是由执行 WRITE语句实现的。根据指定的设备，“写”输出可取打印或软盘介质记录的格式。在执行 WRITE 语句时，有关文件必须以 OUTPUT 或 I-O 方式打开。

记录名必须是为输出文件定义的一个01层记录，并且可由该文件名限定。执行 WRITE 语句将逻辑记录交付给该文件，并更新其 FILE STATUS 项。

如果输出数据已在工作存储域或另一个区域（例如输入文件记录域）形成，用户可用 FROM 后缀规定将数据名 1 中的数据拷贝到记录名指定的区域，然后从那里输出。记录名和数据名 1 必须引用不同的存储域。

当试图执行超越顺序文件外部定义边界的写操作时，将执行说明过程，且 FILE STATUS 将指示越界。如果既无说明过程，也无 FILE STATUS 指定，则发生严重的运行时错误。

ADVANCING 任选仅限于行打印机输出文件，允许程序员控制打印纸上的行间距，操作数可是无符号整数文字或数据名，允许取值 0 至120：

整数	托架控制动作
0	无间距
1	正常单间距
2	倍间距
3	三倍间距
⋮	⋮

如果在 WRITE 语句中无 BEFORE 或 AFTER 任选，则假定为单间距（即“前进一行之后”打印）。

使用关键字 AFTER 隐含托架控制动作先于打印一行的动作；而使用 BEFORE 隐含打印先于托架动作。如指定 PAGE，则在打印机重新定位于下一物理页之前 (BEFORE) 或之后(AFTER)打印数据。但是，如果LINAGE 子句与该文件有关，则重新定位于下一逻辑页的首行，按 LINAGE 子句指定。

如果指定 END-OF-PAGE 短语，则在相关文件的文件描述体中必须指定 LINAGE子句，EOP 等价于 END-OF-PAGE。

当带有 END-OF-PAGE 短语的 WRITE 语句出现页结束条件时，则在页体的底部区域内产生打印或间距。这种情况发生在 WRITE 语句使得 LINAGE-COUNTER 等于或超过 FOOTING 指定的值时。在这种情况下，执行 WRITE 语句后，就执行 END-OF-PAGE 短语中的命令语句。

当 WRITE 语句不能完全适合于当前页体内部时，发生页溢出。这种情况发生在WRITE 语句使得 LINAGE-COUNTER 超过 LINAGE 子句中指定为页体尺寸的值时。在这种情况下，该记录在打印机重新定位于下一逻辑页首行之前或之后打印。如果在 END-OF-PAGE 子句中指定了命令语句，则在打印完该记录、打印机重新定位之后执行。

很明显，如果在 LINAGE 子句中未指定 FOOTING 值，或者页结束和页溢出条件同时发生，则仅溢出条件起作用。

4.17 CLOSE 语句（顺序 I-O）

在文件处理完成时，必须执行 CLOSE 语句，使系统对该文件作适当的整理。文件关闭后或文件未打开前，READ、REWRITE 或 WRITE 语句都不能正确执行，都将发生运行时错误，中止程序运行。

CLOSE 语句的一般格式如下：

CLOSE {文件名〔WITH LOCK〕} …

如果使用 LOCK 短语，则运行时系统将使得在当前作业运行期间，对该文件的后继 OPEN 操作失败。如文件名后未指定 LOCK，则如果程序有逻辑规定，在该程序中晚些时候还可以重新打开这个文件。

试图对当前未打开的文件执行 CLOSE 操作将产生运行时错误，并使执行中止。

CLOSE 语句实例：

```
CLOSE MASTER-FILE-IN WITH LOCK,WORK-FILE;
CLOSE PRINT-FILE,TAX-RATE-FILE, JOB-PARAMETERS WITH LOCK
```

4.18 REWRITE 语句（顺序 I-O）

REWRITE 语句可以替换顺序磁盘文件上的逻辑记录，其一般格式为：

REWRITE 记录名〔FROM 数据名〕

“记录名”是数据部文件节中的逻辑记录名，可以受限。记录名和数据名必须引用不同的存储域。

在该语句执行时，记录名所从属的文件须以 I-O 方式打开(见4.14节 “OPEN 语句”)。

如果该语句包含 FROM 短语，其效果相当于在 REWRITE 语句前执行 “MOVE 数据名 TO 记录名”。

执行 REWRITE 语句可替换最近的 READ 语句读入的记录。因此，REWRITE 前的读语句必须已经完成。如果被重写入文件中的记录长于该文件的记录，则只有可容纳的字节被实际重写。另一方面，如果被重写入文件中的记录短于该文件的记录，则在该记录之后写入不可知的信息，直至该文件下一个记录开始。

4.19 I/O 错误处理的一般注释

如果发生 I/O 错误，则文件的 FILE STATUS 项设置相应的两字符代码，否则认为该值为 “00”。

如果发生 I/O 错误，且其类型与 AT END 或 INVALID KEY 子句有关，则执行子句后面的命令语句序列。但是，如果语句中没有相应的子句（例如，OPEN 或 CLOSE 语句中是不出现这些子句的，对其他 I-O 语句，这些子句是任选的），则程序流程如下：

1. 如果有一个说明 ERROR 过程（见第九节），则自动执行。用户写的程序逻辑必须能够决定产生该错误时应采取的动作。从出错处理过程返回时，允许执行正常程序流程中的下一个句子（I/O 语句后面的句子）。

2. 如果没有说明 ERROR 过程，但有一个相关的 FILE STATUS 项，则用户可以根据对该状态项的测试情况而决定所要采取的动作。程序可以按照正常流程，执行下一个句

子。

仅当上述各项（INVALID KEY/AT END 子句，说明 ERROR 过程，可测试的 FILE STATUS 项）均不存在时，运行时错误处理程序才获得控制，使出错单元（源程序行号）在控制台上显示，运行异常终止。

上述附注适用于任何文件的处理，不管是顺序组织、行顺序组织、索引组织还是相对组织的文件。

4.20 STRING 语句

STRING 语句可将多个传送数据项值连接起来，送到单个接收项中。该语句的一般格式为：

STRING { 操作数 1 … DELIMITED BY {操作数 2 / SIZE} }…
INTO 标识符 1 [WITH POINTER 标识符 2]
[ON OVERFLOW 命令语句]

在这个格式中，操作数系指非数值文字、单字符的象征常数或数据名。标识符 1 是接收数据项名，必须是不带编辑符号或 JUSTIFIED 子句的字母数字。标识符 2 是一个计数器，必须为整型基本数字数据项，该数据项值（加 1 ）指向标识符 1 中的字符位置。

如果没有 POINTER 短语，则逻辑指针缺省值是 1，逻辑指针值指出接收域的开始位置，数据从这里开始存放。在向接收域传送时，每一个源的终止是由 DELIMITED BY 短语控制的：

DELIMITED BY SIZE：传送整个源字段（除非接收域满）

DELIMITED BY 操作数 2：以由操作数 2 指定的字符串作为检索规范，如发现它与连续的发送字符序列相匹配，就停止对当前正在发送的操作数的传送（并自动转换到传送下一个操作数）。在发送域中相匹配字符不传送到标识符 1 。

如果在任一点上，逻辑指针（每向标识符 1 中存入一个字符，则自动加 1 ）小于 1，或大于标识符 1 的大小，则不再发生数据传送，而去执行 OVERFLOW 短语中给出的命令语句。如无 OVERFLOW 短语，则控制转向下一条可执行语句。

对标识符 1 的任何位置，不自动填入空格。即在 STRING 语句完成时，接收项中未被存取的位置保持不变。

在 STRING 语句中，如果有 POINTER 短语，则在该语句完成时，标识符 2 的结果值等于其原来值加上 STRING 语句执行期间所传送的字符数。

4.21 UNSTRING 语句

UNSTRING 语句将单个发送字段分成若干子字段，放到多个接收域中。该语句的一般格式为：

UNSTRING 标识符 1
[DELIMITED BY [ALL] 操作数 1 [OR [ALL] 操作数 2]…]
INTO {标识符 2 [DELIMITER IN 标识符 3] [COUNT IN 标识符 4]}…
[WITH POINTER 标识符 5]
[TALLYING 标识符 6]
[ON OVERFLOW 命令语句]

子字段划分的规范是由 DELIMITED BY 短语给出的。每当有一系列连续字符与操作数 i 所命名的一个数据项值、非数值文字或一个字符的象征常数相匹配，则停止对发送字符的收集，并将其传送到 INTO 子句指定的下一个接收域中。当指定 ALL 短语，标识符 1 中连续多次出现操作数 i 时，也按出现一次处理。限定字符串不传送到当前接收域。

如果有两个或多个定界符，则相当于存在 OR 条件。每个定界符按 UNSTRING 语句规定的顺序与发送字段进行比较。

标识符 1 必须是组项或字符串（字母数字型）项。如果用一个数据项作操作数 i，则该操作数也必须是一个组项或字符串项。

接收域（标识符 2）可以是下述任一种类型：

1. 非编辑字母项
2. 字符串（字母数字）项
3. 组项
4. 外部十进制项（数字型 USAGE DISPLAY）其 PICTURE 描述不含 P 字符。

当测试时碰到两个相邻的定界符，则当前接收域或填以空格，或填以零，依其类型而定。如果在 UNSTRING 语句中有“DELIMITED BY”短语，则 INTO 子句所涉及的任一接收项（标识符 2）后面可跟有“DELIMITER IN”短语。在这种情况下，传入标识符 2 数据的定界字符本身被存入标识符 3，它应该是字母数字项。如果出现 “COUNT IN”短语，则传入标识符 2 的字符数送至标识符 4，它必须是整型基本数字项。

如果有“POINTER”短语，则标识符 5 必须为整型数字项，其初值变为逻辑指针初值（否则假设逻辑指针值为 1）。对源字符的测试从标识符 1 中由逻辑指针设定的位置开始。在 UNSTRING 语句完成时，逻辑指针终值拷贝回标识符 5 中。

如果逻辑指针值小于 1 或超过标识符 1 的长度，则认为发生溢出，控制转到“ON OVERFLOW”子句中给出的命令语句序列。

在源字段全部传送完之前，如果接收域已满，则也发生溢出。

在源字段扫描期间（查找匹配的定界符序列）组成一个变长字符串，当识别到一个定界符，或所得到的字符数已达到当前接收域可容纳的长度，则按标准 MOVE 方式将该字符串传送到当前接收域中。

如果有“TALLYING IN”短语，标识符 6 必须为整型数字项，在完成 UNSTRING 语句后，接收项个数加上标识符的初值，放入标识符 6 中。

任何与标识符 1、5 或 6 有关的下标或索引，仅在 UNSTRING 语句开始时求值一次。任何与操作数 i 或标识符 2、3、4 有关的下标，在访问该数据项之前立即求值。

4.22 动态调试语句

可以动态设置或重置执行跟踪方式。当设置时，过程名按其执行顺序打印在用户控制台上。

执行 READY TRACE 语句设置跟踪方式，可打印每次进入的每一个节名和段名。按执行顺序打印过程名清单，对于检查程序故障是很有价值的，它可以辅助确定在哪一点上程序的实际流程偏离了预期的流程。

为显示过程中特别指定的某些点上的临界数据值，还需要另一个调试特性，EXHIBIT 提供了这种功能。其语句格式为：

EXHIBIT NAMED 〔位置说明〕 { 标识符 / 文 字 / ERASE } …〔UPON 助记名〕

该语句产生指示文字值或数据项(以数据名=值的格式)的打印输出，位置说明和UPON短语的作用与DISPLAY语句中的一样。

EXHIBIT、READY、TRACE和RESET TRACE语句是对ANS-74标准 COBOL的扩充，这些语句设计用于辅助程序调试。

程序设计时需注意：以上调试语句应在源程序行第7列中包含D，这样这些语句在编译时可被忽略，除非在SOURCE-COMPUTER段指定WITH DEBUGGING MODE。

第五节 程序间通讯

分别编译的各COBOL程序模块可以组成一个可执行程序。利用数据部的连接节（跟在工作存储节之后）和CALL语句中附加在一个子模块过程部头的USING表可以实现程序间的通讯。连接节描述了存储器中另一程序的可用数据。连接节中的记录描述体提供了可由其他程序引用并保留在内存中的数据名。连接节中的项不保留在内存区中，因为我们认为在调用程序中该数据已在内存其他地方出现。

任何记录描述子句均可用于连接节中描述数据项，但除88层项外，不能指定VALUE子句。

程序链接功能允许COBOL程序传送控制至其他可执行程序，并将数据项作为参数传递给被链接程序。

5.1 CALL语句

CALL语句的格式是：

CALL 文字 〔USING 数据名…〕

其中文字是一个子程序名，它是分别编译的各程序的程序标识，是非数字的。通过将地址传递给被调用子程序，可使USING表中的各数据名用于被调用子程序。这些地址将分配给在该子程序USING表中说明过的连接节各项。因此CALL语句中指定的数据名个数必须与过程部USING表中的数据名数目相同。机器语言级的信息传递约定在第一章“COBOL-80用户指南”中描述。

注意，调用程序与被调用程序表之间的对应是按位置不是按名。

5.2 EXIT PROGRAM语句

EXIT PROGRAM语句出现在被调用的子程序中，使得控制返回到调用程序CALL语句后的下一条可执行语句，该语句必须自成一段。

5.3 CHAIN语句

CHAIN语句的格式为：

CHAIN { 文 字 / 标识符 1 } 〔USING 标识符 2 …〕

其中文字和标识符1必须是字母数字，且标识符1必须包含一个结束空格。标识符2的各参数（项）必须在工作存储节或连接节中定义，或在执行CHAIN语句时打开文件的记录域中定义。

当CHAIN语句执行时，文字或标识符1的值、直至（但不包括）遇到的第一个空格（或文字结尾），被解释为所采用相应操作系统格式的一个可执行程序文件名。把这个命名的程序装到存储域中，并且执行。除使用USING子句将参数传送给被链接程序外，所有链接程序及其数据结构均要遭到破坏。

被链接的程序不一定是COBOL程序，如果是，则它必须是主程序。

5.4 带CALL和CHAIN的过程部头

主程序的过程部头写作：

PROCEDURE DIVISION 〔CHAINING 数据名1…〕

子程序的过程部头写作：

PROCEDURE DIVISION USING 〔数据名2…〕

过程部头的各种形式描述了程序链接和参数初置要求。主程序或者由它本身链接，或者与若干子程序链接。然后，它可以独立运行，也可以在另一程序中执行CHAIN语句来调用。一个子程序必须与一个主程序链接，或者任选地和若干其他子程序链接。它只能由CALL语句调用执行。关于链接处理，见第一章“用户指南”。

当且仅当调用CHAIN或CALL语句有USING表时，被链接或被调用的程序应该有CHAINING表或非空的USING表。表中的项目数应相等，在位置上应对应于该两个表中的项，且引用大小相同，并有相同USAGE的数据项。不遵循这些规则将在运行时导致不可预测的结果。

在过程部头中命名的数据项值，在程序初始化时，使用CALL或CHAIN语句调用时命名的对应位置的数据项内容来建立。在CALL的情况下，由传递指针来标识。因此，如果在PROCEDURE DIVISION USING子句中命名的数据项值在程序执行期间改变了，则在控制从子程序返回后调用程序相应数据项中将反映出这个变化。

CALL和CHAIN语句参数传递格式请参阅第一章。

第六节 索引法表处理

6.1 索引名和索引项

一个索引名的说明不是用通常的层号、名和数据描述子句来说明，而是由附加在OCCURS子句中的“INDEXED BY索引名”来隐含说明。索引名必须是唯一的。

索引数据项是一个由USING IS INDEX短语定义的项，它不能有PICTURE说明。索引名或索引数据项只能由SET语句或SEARCH语句引用，或由CALL语句的USING表或过程头USING表引用；同时它还可以用在关系条件中，或作为PERFORM VARYING语句中的变化项使用，或作为下标出现，其处理方式等价于用二进制整数下标进行处理。

索引名在SET、SEARCH或PERFORM语句使用前必须赋初值。

6.2 SET语句

为进行表处理可用SET语句对索引名、索引项或二进制下标作代换。该语句有两种格

式：

格式1：

$$\underline{\text{SET}}\left\{\begin{array}{l}\text{索引名1}\\\text{索引项1}\\\text{数据名1}\end{array}\right\}\cdots\text{TO}\left\{\begin{array}{l}\text{索引名2}\\\text{索引项2}\\\text{数据名2}\\\text{整　数2}\end{array}\right\}$$

格式2：

$$\underline{\text{SET}}\ \text{索引名3}\cdots\left\{\begin{array}{ll}\underline{\text{UP}} & \underline{\text{BY}}\\\underline{\text{DOWN}} & \underline{\text{BY}}\end{array}\right\}\left\{\begin{array}{l}\text{数据名4}\\\text{整　数4}\end{array}\right\}$$

其中格式1等价于传送“TO”值（如整数2）到SET后的多个接收域。

格式2等价于增加（UP）或减少（DOWN）SET后跟的名字的值，其增加或减少的量由BY后跟的名或数值决定。对任何SET语句，数据名限制为整数项。

6.3 相对索引

用户在一个OCCURS子句控制的表中查找某一项，可用一个适当的数作为下标，下标用逗号分开，放在一对括号中。例如：

TAX-RATE (BRACKET, DEPENDENTS)

XCODE (I, 2)

这里下标是通常的十进制整型数据名，或整常数，或二进制整数项（COMPUTATION或INDEX），或是一个索引名。下标可以受限，但不能再带下标。下标可带符号，但必须是正号。下标的最低值为1，指向表的第一个元素。最高值是OCCURS子句中指定的最大重现次数。也可使用相对索引，在这种情况下，一个下标表示为：

名　±　整常数

这里加号或减号的两侧要留有一个空格，“名”可以是任意适当的索引名。例如：

XCODE (I + 3, J - 1)

6.4 SEARCH语句（格式1）

对一个表进行线性检索可以使用SEARCH语句，一般格式为：

$\underline{\text{SEARCH}}$ 表 〔$\underline{\text{VARYING}}$ 标识符 | 索引名〕

〔AT $\underline{\text{END}}$ 命令语句1〕

$$\left\{\underline{\text{WHEN}}\ \text{条件1}\left\{\begin{array}{l}\underline{\text{NEXT}}\ \underline{\text{SENTENCE}}\\\text{命令语句2}\end{array}\right\}\right\}\cdots$$

其中：表是一个带OCCURS子句的数据项名，这个OCCURS包含INDEX BY列表，表必须写成不带下标或索引，因为SEARCH语句具有自动将索引名与相关表连接起来的特性。

VARYING有四种可能的情况：

1．没有VARYING短语——该表的第一个索引名是变化的。

2．VARYING后面是另一个表中的索引名——表定义中的第一个索引名是隐含为可变的，同时，在VARYING短语中所列索引名也以同样的方式变化。

3．VARYING后面是表中定义的索引名——该索引名是唯一可变的。

4．VARYING后跟整型数据项名——数据项和表中列出的第一个索引名同时都是可变的。

项变化有如下解释：

1．假设初值已由前面出现的语句（如SET语句）建立。

2．若初始值超过了OCCURS子句说明中给出的最大值，则SEARCH操作立即停止；如果 AT END 短语存在，则执行相应的命令语句1。

3．若索引值在有效的索引表范围内（1、2、……直至并包括最大重现值），则计算每一次的WHEN条件直至找到一个“真”值，或全部值为“假”。如果某值为“真”，则执行相应的命令语句，且 SEARCH 操作停止，如果没有真值则索引值增加1，重复作第三步。注意，索引增值适用于项还是索引表，根据规则1～4选择。

如果一个表是另一个表的附属表，则必须借助各OCCURS子句中的INDEXED BY短语把索引名与整个表的各维数联系起来。只有这个 SEARCH 表的索引名是可变的。要检索一个二维或三维表，必须执行多次SEARCH语句，且每次可能要用PERFORM或 VARYING 语句，适当设置其他索引名。

6.5 SEARCH语句（格式2）

格式2 SEARCH语句使用定序数据表。如SEARCH ALL 语句的一般格式为：

SEARCH ALL 表〔AT END 命令语句1…〕

WHEN 条件 {命令语句2… / NEXT SENTENCE}

这里只允许使用一个WHEN子句。对于该“条件”的使用有如下规则：

1．只能使用简单的关系条件或条件名，而且按与表相联的第一个索引名（如果有 OCCURS子句，则还有其他索引）对该条件名加索引。进一步说，条件中出现的与条件名有关的数据名必须在该表的KEY子句中说明。KEY子句是OCCURS子句的附属语句，格式如下：

ASCENDING | DESCENDING KEY IS 数据名…

这里数据名是数据描述体中定义的名或是一个从属数据名。如果给出多个数据名，则它们都必须是从属于这一组项的项名。KEY 短语指明被重复数据按所列数据名升序或降序排列。数据名按有效值的降序列出。

可以指定多个 KEY 短语。

2．在一个简单关系条件中，只允许使用“等于”条件（=或IS EQUAL TO）来测试。

3．任何条件名变量（88层项）必须定义为只含有单值。

4．条件可以通过逻辑连接符AND连接，不得用 OR。

5．在一个简单关系条件中，目标（等号右边的内容）可以是文字或标识符；这个标识符不得在该表 KEY 子句中引用，即不能用和该表相连的第一个索引名进行检索（标识符可以是一个含有限定语和（或）下标或索引的数据名）。

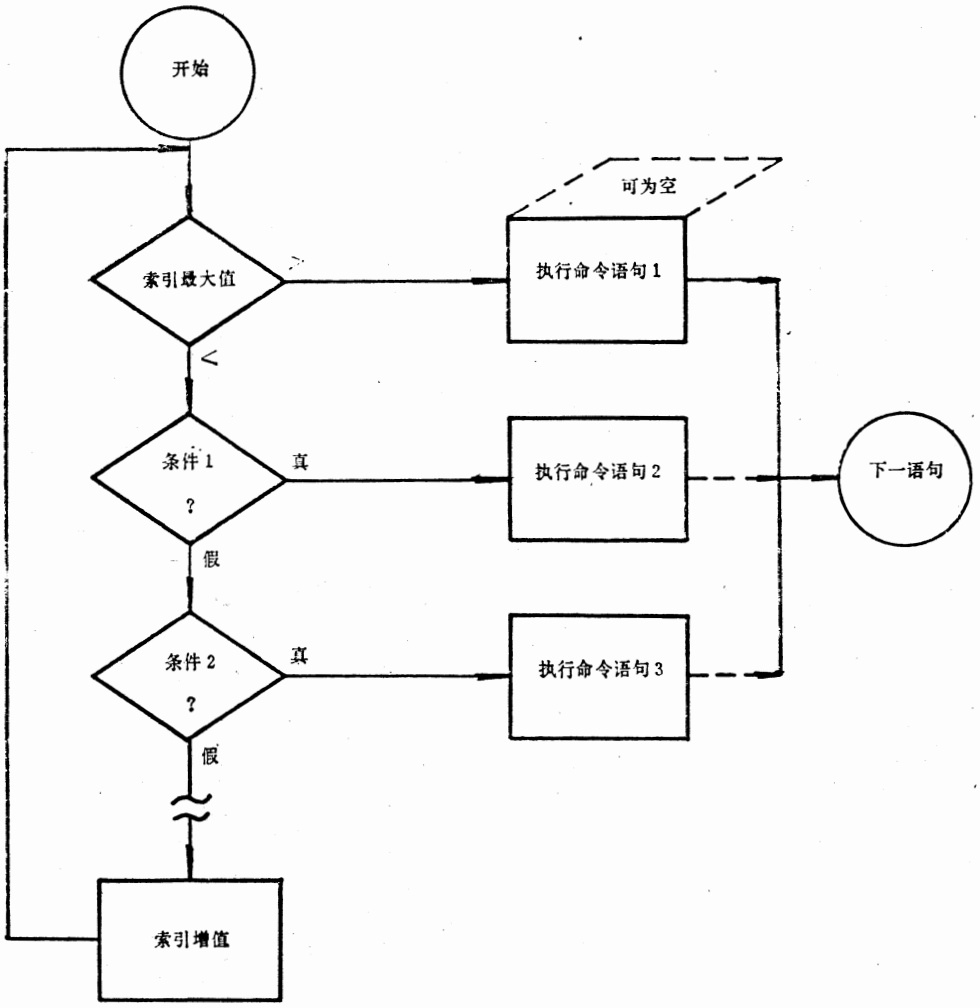
如果这些限制使用不统一，将导致无法预料的结果。如果表数据排列顺序与 KEY 子句中说明的顺序不一致，或在 WHEN 条件中所引用的 KEY 不足以标识唯一的表元素，也将导致无法预料的结果。

使用格式2的 SEARCH 语句也可以对说明过的顺序数据进行非顺序检索操作，该表

索引名的初始值被忽略，在检索过程中它总是在最大值范围内自动变化。如果条件(WHEN)对于任何有效的索引值都不能满足，则当 AT END 子句存在时，控制将转给命令语句1；如果不存在 AT END 子句，则控制转向下一个可执行语句。

如果一个 WHEN 条件中的所有简单条件都满足，则结果索引值指明满足所有条件的具体值，并且控制转给命令语句2，否则最后一次的设置是无法预料的。

SEARCH 语句格式1的逻辑图如下：



第七节　索引文件

7.1　索引文件组织的定义

索引文件组织通过一个指针目录（称为控制索引）对数据文件的记录进行登记或存取。指针指出了有唯一键值的记录的位置。索引文件必须在 SELECT 语句中指明分配到磁盘上。

索引文件组织既可以顺序存取，也可以动态存取或随机存取。顺序存取按 RECORD KEY 值升序存取数据记录。随机存取时，存取记录的顺序受程序员控制，要求被存取记录

的关键字值预先放置在存取语句的KEY数据项中。采用动态存取方式时，程序员可以以顺序存取改为随机存取，反之亦然。

7.2 语法考虑

在环境部中，SELECT项必须规定ORGANIZATION IS INDEXED，且ACCESS子句的格式是：

ACCESS MODE IS SEQUANTIAL | RANDOM | DYNAMIC

INDEX文件在FD描述体中时，必须出现LABEL RECORDS STANDARD和VALUE OF FILE-ID子句。可以使用3.14中的格式，但是仅与磁盘相关的形式才可以使用。

7.2.1 RECORD KEY子句

这个子句的一般格式为：

RECORD KEY IS 数据名1

此处的数据名1是在有关文件的记录描述中定义过的项，它可以是一个组项或是一个字母数字型的基本项。最大关键字长是60字节，而且不能全为空格。

如果规定随机存取方式，则数据名1的值要指明是由下一次DELETE、READ、REWRITE或WRITE语句来存取的记录。每个记录必须有一个唯一的记录键值。

7.2.2 文件状态报告

对索引组织文件，如果环境部出现FILE STATUS子句，则在I/O语句之后设置两个字符数据项。

下面总结了可能出现的设置情况：

状态数据项左字符	状态数据项右字符				
	无进一步说明(0)	结构错(1)	重复键(2)	没找到记录(3)	磁盘空间满(4)
成功完成(0)	×				
结　束(1)	×				
无效键(2)		×	×	×	×
永久错(3)	×				×
特殊情况(9)		×			

对于索引文件，如果ACCESS MODE是SEQUENTIAL，且WRITE语句没有按递增顺序出现或键值在REWRITE以前被改变，则都将产生文件状态'21'。在一个OPEN INPUT或OPEN I-O语句中，文件状态'30'表示'文件没找到'。对于结构已破坏的索引文件或相对文件的OPEN INPUT或OPEN I-O语句，则出现文件状态"91"。当这个状态是在OPEN INPUT返回时，该文件被认为已经打开，且可执行读语句。当在OPEN I-O返回时，该文件没有打开，所有的I/O操作失败。其他设置则是自我解释性的。

注意，发生"磁盘空间满"与索引文件和相对文件处理使用无效键(2)有关，对顺序文

件来说与“永久错误(3)”有关。

如果在执行时出错且没有给出 AT END 或 INVALID KEY 语句，没有提供适当的说明 ERROR 节，也没有规定 FILE STATUS，那么，错误将显示在控制台上，同时程序终止。见4.19节。

7.3 索引文件的过程部语句

顺序文件 OPEN 语句的语法（见4.14节）也可以应用于索引组织文件，只是 EXTEND 不能用。

下表列出了有效的语句类型和它们所允许的 ACCESS 方式和有效的 OPEN 选择项。X 出现的地方表示相应语句可以使用，否则在有关 ACCESS 方式和 OPEN 选择项下相应语句不能使用。

存取方式	过程语句	有效的 OPEN 选择项		
		输 入	输 出	I-O
SEQUENTIAL	READ	×		×
	WRITE		×	
	REWRITE			×
	START	×		×
	DELETE			×
RANDOM	READ	×		×
	WRITE		×	×
	REWRITE			×
	START			
	DELETE			×
DYNAMIC	READ	×		×
	WRITE		×	×
	REWRITE			×
	START	×		×
	DELETE			×

除了上面的语句以外，在各种条件下 CLOSE 是允许的。其格式与4.17节中所叙述的相同。

7.4 READ 语句

格式1（顺序存取）：

READ 文件名 〔NEXT〕 RECORD 〔INTO 数据名1〕

〔AT END 命令语句…〕

格式2（随机或动态存取）：

READ 文件名 RECORD 〔INTO 数据名1〕〔KEY IS 数据名2〕

〔INVALID KEY 命令语句…〕

不带 NEXT 选择项的格式1读语句用于顺序存取方式的文件。带有 NEXT 选择项的格式1读语句用于动态存取方式的文件。当产生逻辑文件结束条件时，执行 AT END 子句，如果此语句没有写在源程序中，则文件结束时控制转向说明 ERROR 节。

随机存取方式的文件或动态存取方式的文件在进行随机检索时，都可以使用格式2。

在格式2中，如果文件中不存在要存取的键值，则执行INVALID KEY子句规定的动作。如果没有给出该子句，则控制交给说明ERROR节。

任选的KEY IS子句必须指明在文件的SELECT项中说明过的记录键项。该子句只用作说明，在执行一个随机存取的READ语句前，用户必须给所有指定的键一个有效键值。

顺序文件有关INTO短语的规则在这里同样适用。

7.5 WRITE 语句

WRITE语句为输出或输入/输出文件释放一个逻辑记录。

其一般格式为：

WRITE 记录名 〔FROM 数据名1〕

〔INVALID KEY命令语句…〕

在执行WRITE语句前，在记录名（或有FROM出现时的数据名1）中必须有一个唯一有效值。这个记录名充当RECORD KEY。

当出现非法键值时，语句中若有INVALID KEY子句，则执行指定的命令语句，否则调用说明ERROR节。

在下列情况可产生INVALID KEY条件：

1. 对顺序存取来说，从一个WRITE到下一个WRITE，键值不是递增的；
2. 键值不唯一；
3. 超过所分配的磁盘空间。

7.6 REWRITE 语句

REWRITE语句逻辑地代换一个已存在的记录，其格式为：

REWRITE 记录名 〔FROM 数据名〕〔INVALID KEY 命令语句…〕

对于顺序存取文件，为使一个REWRITE语句有效，必须完成最后一个READ语句。如果记录名中的键值不等于其前READ语句给出的键值，则产生无效键条件错，如有命令语句，则执行之；否则执行说明ERROR节。

对于随机方式或动态存取方式文件，要代换的记录必须由记录键指明，不需要事先执行READ语句。当文件中不存在要找的记录键值时，则产生INVALID KEY条件。

7.7 DELETE 语句

DELETE语句从索引文件中逻辑地删除一个记录。其一般格式为：

DELETE 文件名 RECORD 〔INVALID KEY 命令语句…〕

对于顺序存取文件，其文件名执行的最后一个输入/输出语句必须是一个成功的READ语句，DELETE语句将读入的记录删除。因此对于顺序存取文件不必规定INVALIDKEY短语。

对于随机存取或动态存取文件，删除的记录是与给出的记录键值有关的记录；如果没有相匹配的记录，则产生'无效键'条件，控制转向INVALID KEY子句中的命令语句，如果没有INVALID KEY子句，则控制转向说明ERROR节。

7.8 START 语句

START语句根据规定的键值为索引组织文件确定了读的位置。该语句对于顺序或动态

存取方式打开的文件都适用，其格式为：

START文件名[KEY IS { GREATER THAN / NOT LESS THAN / EQUAL TO } 数据名]

INVALID KEY 命令语句…]

其中数据名必须是一个说明过的记录键，且在记录中与之相匹配的值必须预先存入该数据名。当执行这个语句时，有关文件必须以 INPUT 或 I-O 方式打开。

如果 KEY 短语不存在，则寻找该文件中和所给记录键值相等的记录。如果键值规定为 GREATER 或NOT LESS，则该文件定位在键值大于等于指定值的第一个记录作为下一存取位置。如没有匹配记录，则执行 INVALID KEY 子句中的命令语句或说明 ERROR 节。

第八节 相对文件

8.1 相对文件组织定义

相对文件组织限于磁盘文件。记录根据相对记录号来区分，记录号的范围从1到32,767。对一个较小的文件，最大值也可取得小些。相对文件与索引文件不同，索引文件的键字段作为数据记录的一部分，而相对文件的记录号是概念性的，并不插入到数据之中。

相对组织文件可以顺序存取，动态存取或随机存取。在顺序存取方式中，按照记录号递增顺序存取记录。

在随机存取方式中，将记录号放在相对键项中，由程序控制存取顺序。在动态存取方式中，程序可以在需要时从随机存取改为顺序存取。

8.2 语法考虑

在环境部，SELECT 项必须规定 ORGANIZATION IS RELATIVE，同时ACCESS 子句的格式为：

ACCESS MODE IS SEQUENTIAL | RANDOM | DYNAMIC.

分配、保存和文件状态子句格式与顺序组织、索引组织文件中使用的格式相同。当 STATUS KEY1 等于'2'时，STATUS KEY2 的值为：

'2' 试图写一个重复键

'3' 找不到记录

'4' 磁盘空间满

在有关的 FD 描述体中，必须说明 STANDARD 标号，并包含 VALUE OF FILE-ID 子句。与相对文件有关的记录域的第一个字节不能使用任何记录描述把它描述成为 COMP 或Comp-3项的一部分。

8.2.1 RELATIVE KEY 子句

在 SELECT 项中，除一般子句外，对随机或动态存取方式还需要有下列形式的子句：

RELATIVE KEY IS 数据名1

如果该文件存在 START 语句，则顺序存取方式也需要这个子句。

数据名1必须是无符号的二进制整数项，它不包含在文件本身的任何记录描述中，它的

值必为正数和非零数字。

8.3 相对文件的过程部语句

与索引组织文件相同，在过程部中OPEN、CLOSE、READ、WRITE、REWRITE、DELETE和START 语句都是可用的(因此在7.2.2和7.3节中的图表也适用于RELATIVE文件)。除“EXTEND”短语外，顺序文件的OPEN 和CLOSE 语句格式也可用于相对文件。

8.4 READ语句

格式1：

READ 文件名〔NEXT〕RECORD〔INTO 数据名〕
〔AT END 命令语句…〕

格式2：

READ 文件名RECORD〔INTO 数据名〕
〔INVALID KEY 命令语句…〕

所有顺序存取文件必须使用格式1。如果文件说明为动态存取方式，则必须使用NEXT短语实现顺序存取。当产生逻辑的“文件结束”条件时，便执行AT END子句，如果没有AT END子句，控制交给说明ERROR节。当文件说明为动态或随机存取方式时，使用格式2读语句完成随机存取。

如果在文件的SELECT 描述体中定义了相对键，则在执行了格式1 READ 语句后，便可更新RELATIVE KEY 项（“数据名1”）的内容，使之包含被检索记录的记录号。

对于格式2 READ 语句，被检索记录的记录号预先放在RELATIVE KEY 项中，如果此记录不存在，则产生INVALID KEY 条件，或者执行READ 语句中INVALID KEY后的命令语句，或者执行一个有关的说明节。

顺序文件有关INTO 短语的规则在此也适用。

8.5 WRITE 语句

相对文件WRITE 语句的格式与索引文件相同：

WRITE 记录名〔FROM 数据名〕
〔INVALID 命令语句…〕

如果是顺序方式存取，则WRITE 语句的完成使刚输出记录的相对记录号放入RELATIVE KEY 项中。如果存取方式是随机的或动态的，为给记录赋予一个相对号，则用户必须预先设置RELATIVE KEY 项的值。如果已存在带有指定记录号的记录，或磁盘空间满，则产生INVALID KEY 条件。

8.6 REWRITE 语句

相对文件REWRITE 语句格式与索引文件REWRITE 语句格式相同：

REWRITE 记录名〔FROM 数据名〕
〔INVALID KEY 命令语句…〕

对于顺序存取文件，在重写之前必须完成读语句再执行REWRITE 语句使该读入记录被取代。如果前面的READ 语句不成功，则发生运行时错误，并终止该程序。因此顺序存取文件可以没有INVALID KEY 子句。

对于说明为动态或随机存取方式的文件，通过REWRITE 语句替换的记录，其记录号

预先放在RELATIVE KEY项中。如果该记录不存在，则产生INVALID KEY条件。

8.7 DELETE语句

相对文件的DELETE语句与索引文件的DELETE语句格式相同。

DELETE 文件名 RECORD〔INVALID KEY 命令语句…〕

对于顺序存取文件，在此语句之前必须是一个成功的READ语句；DELETE语句执行后，前面读入的记录被逻辑地从相应文件中删除。如果前面的读语句未成功，则发生运行时错误，并终止该程序的执行。因此对顺序文件可以不指定INVALID KEY短语。

对于说明为动态或随机存取方式的文件，其被删除记录的记录号预先在RELATIVE KEY项中指定。如果此记录号不存在，则产生INVALID KEY条件。

8.8 START语句

此语句格式同索引文件中START语句格式一样：

$$\text{START 文件名}\left(\text{KEY IS}\left\{\begin{matrix}\text{GREATER THAN}\\ \text{NOT LESS THAN}\\ \text{EQUAL TO}\end{matrix}\right\}\text{数据名1}\right)$$

〔INVALID KEY 命令语句…〕

执行此语句为读操作规定了起始位置。该语句只可用于顺序或动态存取文件。其中数据名只限于预先说明过的RELATIVE KEY项，在START语句执行前相对记录号必须存于其中。当执行START语句时，有关文件必须以INPUT或I-O方式打开。

如没有KEY短语，则寻找文件中与该记录键值相等的记录；如规定了键值GREATER或NOT LESS，则文件定位在大于（或大于等于）指定键值的第一个记录，作为下一个存取位置。

如果没有找到这个相对记录，则执行INVALID KEY子句中的命令语句或执行相应说明ERROR节。

第九节 DECLARATIVES和USE句子

说明区提供了一个包含过程的方法，这些过程不作为程序员顺序编写的程序的一部分来执行，而是在通常程序员不能预测到的条件发生时执行。在输入/输出错时，系统可以自动地检查和产生标准标号，并执行错误恢复子程序，同时COBOL程序员还可以指定附加的过程。

因为这些过程仅是在发生读/写错误时才执行，所以它们不能在正常过程语句序列中出现。这些过程必须写在过程部开始的DECLARATIVES子区中。指明这些过程功能的USE句子写在有关过程前面。一个说明节在带有USE句子的另一个节名出现时结束，或用关键字END DECLARATIVES结束。关键字DECLARATIVES和END DECLARATIVES都必须在A区开始并用句号结束。例如：

```
PROCEDURE DIVISION.
DECLARATIVES.
```

{ 节名 SECTION. USE 句子.

{ 段名. { 句子 } … } … } …

END DECLARATIVES.

USE 句子定义了所编程序中有关节的应用范围。

一个 USE 句子必须紧跟在过程部说明部分的节头后，且后面必须跟一个句号和空格。该节的剩余部分必须由零个、一个或多个用于定义过程的过程段组成。USE句子本身是不执行的，它只定义 USE过程的执行条件。USE 句子的一般格式为：

USE AFTER STANDARD EXCEPTION | ERROR PROCEDURE

ON {文件名… | INPUT | OUTPUT | I-O | EXTEND}.

其中 EXCEPTION 和 ERROR 可以互换使用。当文件的标准 I-O恢复过程指定后或在一个没有 INVALID KEY 或 AT END 子句的语句执行后，都将执行有关说明节。一个给定的文件名只能和一个说明节相关连。

在说明节中不能引用没有说明的过程。相反，在没有说明的部分不能引用说明节中出现的过程名，只有 PERFORM语句可以引用 USE 语句及其过程；但在某一范围说明中，如果有一个过程名在一个说明节中，则这个范围中的其他过程名也必须在同一个说明节中。

说明节的出口由编译程序插入在该说明节的最后一个语句之后，在该说明节中的所有逻辑程序路径必须通向出口点。

第十节　程序分段

程序分段功能使超过实际内存的 COBOL-80程序能够执行。当使用程序分段时(即程序中任何一个节头包含一个分段号）必须在各节写上完整的 PROCEDURE DIVISION (过程部）。各节在节头指定一个分段号，其格式如下：

节名 SECTION 〔分段号〕.

分段号必须为 0 ～99的整数。如果分段号省略，则假设为 0 。说明节的分段号必须小于50。具有相同分段号的所有节组成一个程序段，并且必须在源程序中一起出现。分段号小于50的所有程序段必须在过程的开始处一起出现。

分段号为 0 ～49的程序段称为固定分段，并且在执行期间常驻内存中。分段号超过49的段称为独立段，各独立段作为程序复盖来处理。在程序执行期间，当控制第一次转给一个独立段时，它处于初始状态，同样当控制从另一个具有不同分段号的段转给该节时，独立段也处于初始状态。当固定段或不同独立段结束而转到此独立段时，它也处于初始状态。

在使用 ALTER 和 PERFORM 语句时程序分段有如下限制：

1. 在任何其他分段中的 ALTER 语句不得引用独立分段中的 GO TO 语句。
2. 在固定分段中的 PERFORM 语句只能在它的范围内具有：
   a. 整个包含在固定分段内的节和（或）段；
   b. 整个包含在一个独立分段内的节和（或）段；
3. 在独立分段中 PERFORM 语句只能在它自己的范围内具有：

a. 整个包含在固定分段内的节和（或）段；

b. 整个包含在与 PERFORM 语句在同一独立段内的节和（或）段中。

附录 I　条件的高级形式

组合条件的求值规则

1. 首先做各简单条件的求值（关系、分类、条件名和符号测试）。

2. 然后，对由 AND 连接的简单条件进行求值，作为一个单独结果。

3. 最后计算 OR 和其相邻条件（或前面的计算结果）。

实例：

1. A<B OR C=D OR E NOT>F，其值等价于(A<B)OR(C=D) OR (E≤F)，并且当三个分别括起来的简单条件在任一个为真时，全式为真。

2. WEEKLY AND HOURS NOT = 0，在扩展88层条件名 WEEKLY 之后，此式等价于：

(PAY-CODE) = 'W') AND(HOURS≒0)，并且仅当两个简单条件都为真时，此式才为真。

3. A = 1 AND B = 2 AND G > - 3 OR
P NOT EQUAL TO "SPAIN" 等价于
[(A = 1)AND(B = 2)AND(G > - 3)]OR(P≒ "SPAIN")。若P = "SPAIN"，则当下面三个条件都为真时，此组合条件为真：

(C.1) A = 1

(C.2) B = 2

(C.3) G > - 3

但是，若 P不等于 "SPAIN"，则不管 A、B和G为何值，此组合条件都为真。

带括号的条件

在组合条件中可以使用括号以指定求值的优先顺序。

实例：

```
IF A = B AND (A = 5 OR A = 1)
  PERFORM PROCEDURE-44.
```

在这种情况下，在 A = B的同时，若A = 5 或 A = 1，则执行 PROCEDURE-44。采用这种方法，通过使用括号，可以在组合条件中包含其他组合条件，而不仅仅是简单条件。

缩写条件

为了简化起见，如果 "主语" 对于几个后继的条件测试是共同的，则可以省略。例如，条件 A = 5 OR A = 1 可写成 A = 5 OR = 1，也可写成 A = 5 OR 1，此处主语和蕴含的关系都相同。

又例如：

```
IF  A = B OR <C OR Y
```

是下式的缩写

```
IF  A = B OR A<C OR A<Y
```

在缩写条件中使用字“NOT”的解释为：

1．若紧跟在‘NOT’之后的项是一关系运算符，则‘NOT’作为关系运算符的一部分。

2．否则，若一个新的完全独立的条件开始，则必须跟在‘NOT’之后，而不作为缩写条件的一部分。

注意：只有在关系测试中，才允许蕴含主语和关系的缩写。符号测试或分类测试的主语不可省略。

NOT，*逻辑否定操作符*

NOT除作为关系的一部分（如IF A IS NOT = B）之外，还可以放在一个条件之前。例如，当(A = B OR A = C)为假时，条件 NOT(A = B OR C) 为真。NOT 也可放在88层条件名之前。

附录Ⅱ　允许的 MOVE 操作数表

源操作数	MOVE 中的接收操作数					
	数字型整数	数字型非整数	数字编辑型	字母数字编辑型	字母数字	组项
数字型整数	OK	OK	OK	OK(A)	OK(A)	OK(B)
数字型非整数	OK	OK	OK			OK(B)
数字编辑型				OK	OK	OK(B)
字母数字编辑型				OK	OK	OK(B)
字母数字	OK(C)	OK(C)	OK(C)	OK	OK	OK(B)
组　项	OK(B)	OK(B)	OK(B)	OK(B)	OK(B)	OK(B)

图例：（A）　任何源符号被忽略。

（B）　若源操作数或接收操作数是一组项，则传送被当作为一组项传送，见第4.3节中组项传送的描述。

（C）　源操作数当作无符号整数，其长度不可超过31。

注：在编译程序中对字母型和字母数字型不作区别，不能将数字型项传送到字母型项之中，反之亦然。

附录Ⅲ　IF 语句的嵌套

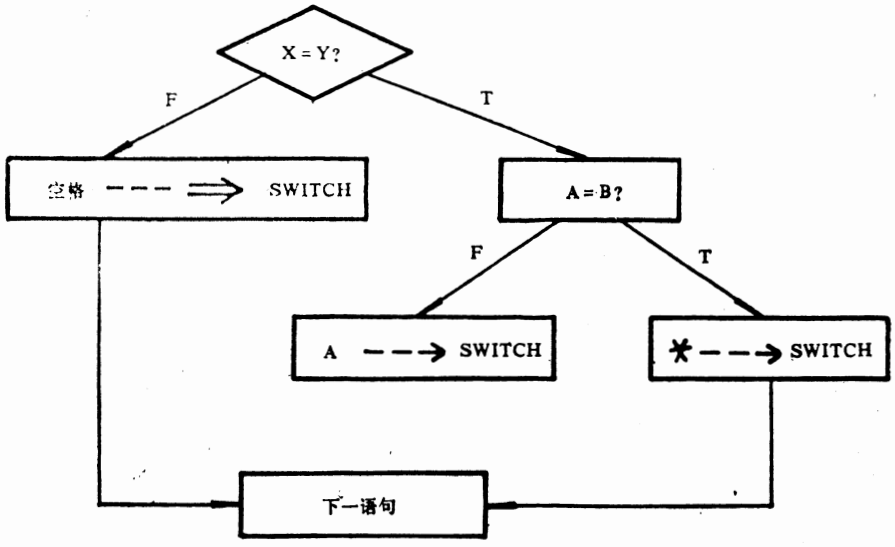
当在一个句子中出现多个 IF动词时，称存在“嵌套的 IF”。

例：

```
IF  X = Y
  IF  A = B
```

```
        MOVE "*" TO SWITCH
    ELSE
        MOVE "A" TO SWITCH
    ELSE
        MOVE SPACE TO SWITCH
```

上述句子的流程可用下面的树结构表示：



观察嵌套IF结构的另一有效方法是根据IF和ELSE的编号表明其优先顺序。

IF1 X=Y

	IF2 A=B
为真	为真则执行动作：MOVE "*" TO SWITCH
动作 1	ELSE2
	为假则执行动作：MOVE "A" TO SWITCH

ELSE 1

为假则执行动作：MOVE SPACE TO SWITCH.

以上图示清楚地表明IF2完全被嵌套在IF1的真动作中。

一个句子中的ELSE数目不一定要与IF的数目相同，ELSE分枝可以少些。

例如：

```
IF  M=1
    IF  K=0
        GO  TO  M1-K0
    ELSE
        GO TO M1-KNOT0.
```

```
IF  AMOUNT IS NUMERIC
    IF  AMOUNT IS ZERO
        GO TO CLOSE-OUT.
```

在后一种情况下，IF2可写作 AND，二者完全等价。

附录Ⅳ　ANS-74 COBOL ASCII 字符集

字符	八进制值	字符	八进制值
A	101	1	61
B	102	2	62
C	103	3	63
D	104	4	64
E	105	5	65
F	106	6	66
G	107	7	67
H	110	8	70
I	111	9	71
J	112	（空格）	40
K	113	"	42
L	114	$	44
M	115	'（非—ANSI）	47
N	116	(	50
O	117	)	51
P	120	*	52
Q	121	+	53
R	122	,	54
S	123	—	55
T	124	.	56
U	125	/	57
V	126	;	73
W	127	<	74
X	130	=	75
Y	131	>	76
Z	132	正零（带正号的零）；	173
0	60	负零（带负号的零）；	175

附录Ⅴ　保留字表

+表示 COBOL-80对交互屏幕、调试扩展和压缩十进制格式要求的附加字。

ACCEPT	ACCESS	ADD
ADVANCING	AFTER	ALL
ALPHABETIC	ALSO	ALTER
ALTERNATE	AND	ARE
AREA(S)	ASCENDING	ASCII+
ASSIGN	AT	AUTHOR
AUTO	AUTO-SKIP+	BEEP+
BEFORE	BELL	BLANK
BLANK	BLINK	BLOCK
BOTTOM	BY	CALL
CANCEL	CD	CF
CH	CHAIN	CHAINING
CHARACTER(S)	CLOCK-UNITS	CLOSE
COBOL	CODE	CODE-SET
COL+	COLLATING	COLUMN
COLUMN	COMMA	COMMUNICATION
COMP	COMP-3+	COMPUTATIONAL
COMPUTATIONAL-3+	COMPUTE	CONFIGURATION
CONTAINS	CONTROL(S)	COPY
CORR(ESPONDING)	COUNT	CURRENCY
DATA	DATE	DATE-COMPILED
DATE-WRITTEN	DAY	DE(TAIL)
DEBUG-CONTENTS	DEBUG-ITEM	DEBUG-LINE
DEBUG-NAME	DEBUG-SUB-1	DEBUG-SUB-2
DEBUG-SUB-3	DEBUGGING	DECIMAL-POINT
DECLARATIVES	DELETE	DELIMITED
DELIMITER	DEPENDING	DESCENDING
DESTINATION	DISABLE	DISK+
DISPLAY	DIVIDE	DIVISION
DOWN	DUPLICATES	DYNAMIC
EGI	ELSE	EMI
ENABLE	END	END-OF-PAGE
ENTER	ENVIRONMENT	EOP
EQUAL	ERASE+	ERROR
ESCAPE	ESI	EVERY
EXCEPTION	EXHIBIT+	EXIT
EXTEND	FD	FILE
FILE CONTROL	FILE-ID+	FILLER
FINAL	FIRST	FOOTING

FOR
FROM
FROM
GENERATE
GIVING
GO
GREATER
GROUP
HEADING
HIGH-VALUE(S)
HIGHLIGHT
I-O
I-O-CONTROL
IDENTIFICATION
IF
IN
INDEX
INDEXED
INITIAL
INITIATE
INPUT
INPUT-OUTPUT
INSPECT
INSTALLATION
INTO
INVALID
IS
JUST(IFIED)
KEY
LABEL
LAST
LEADING
LEFT
LEFT-JUSTIFY+
LENGTH
LENGTH-CHECK+
LESS
LIMIT(S)
LIN+
LINAGE
LINAGE-COUNTER
LINE
LINE(S)
LINE-COUNTER
LINKAGE
LOCK
LOW-VALUE(S)
MEMORY
MERGE
MESSAGE
MODE
MODULES
MOVE
MULTIPLE
MULTIPLY
NAMES+
NATIVE
NEGATIVE
NEXT
NOT
NUMBER
NUMBER
NUMERIC
OBJECT-COMPUTER
OCCURS
OF
OFF
OMITTED
ON
ON
OPTIONAL
OR
ORGANIZATION
OUTPUT
OVERFLOW
PAGE
PAGE-COUNTER
PEN
PERFORM
PF
PH
PIC(TURE)
PLUS
PLUS
POINTER
POSITION
POSITIVE
PRINTER+
PRINTING
PROCEDURE(S)
PROCEED
PROGRAM
PROGRAM-ID
PROMPT+
QUEUE
QUOTE(S)
RANDOM
RD
READ
READY+
RECEIVE
RECORD(S)
REDEFINES
REEL
REFERENCES
RELATIVES
RELEASE
REMAINDER
REMOVAL
RENAMES
REPLACING
REPORT(S)
REPORTING
RERUN
RESERVE
RESET
RETURN

REVERSED	REWIND	REWRITE
RF	RH	RIGHT
RIGHT-JUSTIFY+	ROUND	RUN
SAME	SCREEN	SD
SEARCH	SECTION	SECURE
SECURITY	SEGMENT	SEGMENT-LIMIT
SELECT	SEND	SENTENCE
SEPARATE	SEQUENCE	SEQUENTIAL
SET	SIGN	SIZE
SORT	SORT-MERGE	SOURCE
SOURCE-COMPUTER	SPACE(S)	SPACE-FILL+
SPECIAL-NAMES	STANDARD	STANDARD-1
START	STATUS	STOP
STRING	SUB-QUEUE-1,2,3	SUBTRACT
SUM	SUPPRESS	SYMBOLIC
SYNC(HRONIZED)	TABLE	TALLYING
TAPE	TERMINAL	TERMINATE
TEXT	THAN	THROUGH
THRU	TIME	TIMES
TO	TOP	TRACE+
TRAILING	TRAILING-SIGN+	TYPE
UNIT	UNSTRING	UNTIL
UP	UPDATE+	UPON
USAGE	USE	USING
VALUE(S)	VARYING	WHEN
WITH	WORDS	WORKING-STORAGE
WRITE	ZERO((E)S)	ZERO-FILL+
*	* *	+
-	/	<
=	>	

附录Ⅵ　带VARYING和AFTER子句的PERFORM语句

PERFORM 范围

　　VARYING　标识符1　FROM　参量1　BY　参量2

　　UNTIL　条件1

```
                AFTER  标识符2  FROM  参量3  BY
                  参量4  UNTIL  条件2
               [AFTER  标识符3  FROM  参量5  BY]
                  参量6  UNTIL  条件3
```

此处的标识符为数据名或索引名。参量1、3、5可以是数据名、索引名或文字。参量2、4、6只可以是数据名或文字。

这一复合PERFORM语句的操作等价于下面的COBOL[illegible]变化三项的例子):

```
START-PERFORM.
        MOVE  参量1   TO  标识符1
        MOVE  参量2   TO  标识符2
        MOVE  参量3   TO  标识符3
TEST-CONDITION-1.
 IF  条件1  GO  TO  END-PROGRAM.
TEST-CONDITION-2.
 IF  条件2
        MOVE  参量3   TO  标识符2
        ADD   参量2   TO  标识符1
        GO  TO  TEST-CONDITION-1.
TEST-CONDITION-3.
 IF  条件3
        MOVE  参量5   TO  标识符3
        ADD   参量4   TO  标识符2
        GO  TO  TEST-CONDITION-2.
 PERFORM  范围
        ADD   参量6   TO  标识符3
        GO  TO  TEST-CONDITION-3.
END-PROGRAM. 下一语句.
```

注：若上述的任一标识符为索引名时，则相应的MOVE代换为SET（TO形式），并且相应的ADD代换为SET（UP形式）。

第三章　实用程序软件

第一节　引　　言

MACRO-80是一个用于8080和Z80微机系统的浮动宏汇编程序。它可以在运行 CP/M、ISIS-Ⅱ、TRSDOS 或 TEKDOS 操作系统的任何8080或Z80开发系统中，对8080或Z80代码进行汇编。MACRO-80 软件包包括 MACRO-80 汇编程序、LINK-80 连接装入程序和 CREF-80 相互对照功能，CP/M 版本还包括 LIB-80 库管理程序。MACRO-80约占14K内存空间，其汇编的速率为每分钟1000行以上。

MACRO-80 在不牺牲速度和存储空间的同时，几乎包括“大计算机”汇编程序的全部特性。它支持完整的 Intel标准宏功能，包括 IRP、IRPC、REPEAT、局部变量以及EXITM。宏的嵌套只受内存的限制。代码被汇编成为浮动模块，这些模块由功能灵活的连接装入程序处理。一组扩充的条件伪码操作加强了条件汇编的能力，包括对汇编扫描次数、符号定义以及宏参数的测试。条件可嵌套255层。

MACRO-80 连接装入程序提供一系列通用的装入程序功能，它可通过简单的命令行和开关实现。任意多的程序可用一个命令装入，浮动模块可以装入到用户指定的单元，并且各模块间的外部引用也由装入程序自动解决。装入程序还执行库检索功能，查找系统子程序并生成一个内存装入映象，以显示主程序及子程序的位置。包括在该软件包中的相互对照功能提供一个使用方便的全部程序变量名的字母表，以及这些变量名被引用和定义的行号。

本章作为 MACRO-80 软件包的参考指南，定义、解释 MACRO-80 中的全部特性，并给出一些实例，从而使其对于每个熟悉汇编语言程序设计的人都很容易理解；但并不作为教材，因此要求读者对汇编语言程序设计有基本的了解。要想学习关于汇编程序设计的基本知识，请参阅有关教材。

第二节　MACRO-80汇编程序

2.1　运行 MACRO-80

运行 MACRO-80的命令是：

M80

MACRO-80返回提示符“*”表示已准备好接受命令。

注：若使用 TEKDOS操作系统，可参看本章附录 A中有关的命令格式。

2.2　命令格式

MACRO-80命令由文件名和任选的开关组成。所有的文件名均应遵循操作系统对文件名和扩展名的命名约定。Microsoft 提供的软件隐含扩展名如下：

文件	CP/M	ISIS—Ⅱ
浮动目标文件	REL	REL

列表文件	PRN	LST
MACRO-80源文件	MAC	MAC
FORTRAN源文件	FOR	FOR
COBOL源文件	COB	COB
BASIC源文件	BAS	BAS
绝对文件	COM	

MACRO-80 命令传送被汇编的源文件名和要建立的文件名，并且指明所需的汇编选 择项。命令格式为：

目标文件，列表文件 = 源文件

使用缺省的文件名和扩展名 REL 建立浮动目标文件，只需要有等号和源文件字段即可。否则，如果只填写了目标字段，则只建立目标文件；如果只填写了列表字段，则只建立列表文件。

为汇编源文件而不生成目标文件或列表文件，可在等号左边放一个逗号。在汇编成目标模块之前，用这个过程检查语法错误是非常方便的。

实例：

* = TEST

汇编源文件 TEST.MAC，将目标文件放在 TEST.REL中，不产生列表文件。

*, = TEST

汇编源文件 TEST.MAC，不建立目标或列表文件。可用于错误检查。

TEST, TEST = TEST

汇编源文件 TEST.MAC，将目标文件放在 TEST.REL中，列表文件放在TEST.PRN中（若操作系统为 ISIS-Ⅱ，则列表文件为 TEST.LST)。

* OBJECT = TEST

汇编源文件 TEST.MAC，并置目标文件于 OBJECT.REL中。

OBJECT, LIST = TEST

汇编源文件 TEST.MAC，置目标文件于 OBJECT.REL，列表文件于LIST.PRN（对 ISIS-Ⅱ，为 LIST.LST)中。

MACRO-80也支持命令行，即调用和命令可打在同一行上。例如：

M80, = TEST

2.2.1 设备

可在 MACRO-80 命令串的任何字段指定一个设备名。对于 CP/M 操作系统，缺省设备名为当前联机的磁盘名。对于 ISIS-Ⅱ操作系统，缺省设备名为磁盘驱动器。命令格式为：

设备：目标文件，设备：列表文件 = 设备：源文件

设备名如下：

设备	CP/M	ISIS-Ⅱ
磁盘驱动器	A:, B:, C:, …	:FO:, :F1:, :F2:, …
行式打印机	LST:	LST:
电传打字机或 CRT	TTY:	TTY:

高速阅读机　　HSR:

例如：

*，TTY:=TEST

汇编源文件TEST.MAC并在控制台上列出该程序，不生成目标码。可用于错误检查。

*SMALL，TTY:=B:TEST

汇编TEST.MAC（在磁盘驱动器B上），置目标文件于SMALL.REL中，在控制台上列出该程序。

2.2.2　开关

开关是加在命令串中的一个字母，其前有一斜线。它指出汇编时要执行的选择任务；可用多个开关，但每个开关前必须有一斜线（对于TEKDOS操作系统，开关前须有一逗号或空格。参看附录A）。所有开关都是任选的。可利用的开关有：

开关	动　作
O	以八进制形式列表
H	以十六进制形式列表
R	强制生成一目标文件
L	强制生成一列表文件
C	强制生成一相互对照文件

例如：

*=TEST/L

汇编TEST.MAC，置目标文件于TEST.REL中，列表文件于TEST.PRN中(对ISIS-Ⅱ，在TEST.LST中)。

*=TEST/L/O

同上，但列表文件地址是八进制的。

*LAST=TEST/C

汇编TEST.MAC，置目标文件于LAST.REL中，相互对照文件于TEST.CRF中（参看第三节）。

另外，在3.4版本中补充的还有如下开关：

开关	动　作
M	初始化数据区。 若程序员需要把DS（定义空间）伪操作定义的区域初置为零，则应在命令行中使用/M开关。否则，不能保证该空间含零。亦即，DS不自动地将该空间初置为零。
X	设置初始的当前方式和缺省初值（用于列出或不列出假条件组）。/X置当前方式和缺省初值为不列出假条件组。无/X置当前方式和缺省初值为列出假条件组。当前方式决定假条件是否列出。缺省初值与.TFCOND伪操作一同使用。因此，.TFCOND独立于.SFCOND和.LFCOND。若某程序含.SFCOND或.LFCOND，则在该文件中遇到.SFCOND或.LFCOND之后/X无效，直至遇到一个.TFCOND。因此，/X只在用于一个包括无条件列表伪操作的文件中或

与.TFCOND同用时有效。

下表列出了在/X和无/X情况下遇到上述三种伪操作时的效果。对此三种条件列表伪操作的详细描述，请看2.6.27节。

伪操作	无/X	/X
（无）	ON	OFF
⋮	⋮	⋮
.SFCOND	OFF	OFF
⋮	⋮	⋮
.LFCOND	ON	ON
⋮	⋮	⋮
.TFCOND	OFF	ON
⋮	⋮	⋮
.TFCOND	ON	OFF
⋮	⋮	⋮
.SFCOND	OFF	OFF
⋮	⋮	⋮
.TFCOND	OFF	ON
.TFCOND	ON	OFF
⋮	⋮	⋮
.TFCOND	OFF	ON

P　每个/P分配附加的256字节的栈空间供汇编时使用。若在汇编时发生栈溢出错，则用/P；否则不需要用/P。

2.3 MACRO-80源文件的格式

长度不大于132个字符的输入源文件行都是可接收的。

MACRO-80保留用小写字母提供的串和注释。所有用小写打入的符号、操作码和伪操作码将被转换成大写。

若源文件包含从一编辑程序得来的行号，则该行的每个字节高位（bit）必须置位。来自Microsoft EDIT-80编辑程序的行号是可接收的。

2.3.1 语句

输入到MACRO-80的源文件由下列形式的语句组成：

〔标号：〔：〕〕〔操作符〕〔变元〕〔；注释〕

对于ISIS汇编程序$控制（见2.11节）是个例外，它不要求语句从第一列开始。可使用多空格或标记使其更易读。

若有标号，则其必须是该语句的第一项，并且后面紧跟着一个冒号。若其后跟两个冒号，则它被说明为PUBLIC（见2.6.10节“ENTRY/PUBLIC”）。例如：

```
FOO::RET
```

等价于

```
FOO:PUBLIC FOO
    RET
```

标号后的下一项，即无标号行的第一项，是一个操作符。操作符可以是一个8080或Z80的助记符、伪操作码、宏调用或表达式。执行顺序为：

1. 宏调用
2. 助记符/伪码操作
3. 表达式

汇编程序不将一个表达式标为错误，而把它看作为一个 DB 语句（见2.6.4节）。跟在操作符后面的变量的形式随操作符而变化。

注释总是以一个分号开始，以回车换行结束。注释可以自成一行或附属于包含语句的语句行。补充注释可使用 .COMMENT 伪操作输入（见2.6.20节）。

2.3.2 符号

MACRO-80 符号可为任意长，不过，仅最前面的六位是有效的。如下字符是用于符号中的合法字符：

A～Z　0～9　$　.　?　@

对于 Microsoft 8080/Z80/8086 汇编程序，在一个符号中，底线字符也是合法的。符号不能以数字开始。在读一符号时，小写字母被转变成大写。若一符号引用后面跟着##，则它被说明为外部引用（见2.6.12节“EXT/EXTRN 伪操作”）。

2.3.3 数字常数

对于数字型常数，缺省的基数为十进制。可以通过伪操作 .RADIX （见 2.6.22 节）加以改变。可选择从二进制到十六进制的任何基数。当基数大于10时，跟在 9 之后的数字是 A～F。若某数的第一位不是数字型的，则该数的前面必须有一个零。

数字是16个字位的无符号量。一个数字总是按当前的基数进行计算，除非指定了如下的特殊符号：

nnnnB	二进制
nnnnD	十进制
nnnnO	八进制
nnnnQ	八进制
nnnnH	十六进制
X'nnnn'	十六进制

一个数字超过二个字节的部分被忽略，且结果为低位的16位。

一个字符常数是由括在引号中的零个、一个或两个 ASCII 字符组成，并用于非简单表达式之中。例如，在语句

```
DB 'A' + 1
```

中，'A'是一个字符常数，而在语句

```
DB 'A'
```

中，'A'是作为一个字符串，因为它是在一简单表达式中。字符常数的定义符规则与字符串定义规则相同。

由一个字符组成的字符常数取值为该字符的 ASCII 值。即，该常数的高位字节为零，而低位字节为该字符的 ASCII 值。例如，常数'A'的值是41 H。

由两个字符组成的字符常数以第一字符的 ASCII 值为高位字节值，第二字符的 ASCII

值为低位字节值。例如，字符常数'AB'的值是41H * 256 + 42H。

2.3.4 字符串

字符串由括在引号中的零个或多个字符组成。单引号或双引号都可用作为字符串的定界符。如果定界符引号对每个预期的字符值出现两次，则它也可被用作为字符。例如，语句

DB "I am" "great" "today"

存储的字符串为

I am "great" today

若定界符之间没有字符，则该字符串是一个空字符串。

2.4 表达式的计算

2.4.1 算术和逻辑操作符

下列操作符允许出现在表达式中。各操作符按优先权先后顺序列出：

NUL

LOW, HIGH

*, 1, MOD, SHR, SHL

一元减

+, -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

使用括号可改变操作符执行的先后顺序。在计算一个表达式期间，每出现一个新的操作符，都与前一个操作符进行比较。当这个新出现操作符的优先数小于或者等于前一个操作符时，就执行到这个操作符为止的全部操作。即，包含较高优先数操作符的子表达式先被计算。

除+、-、×、/外的全部操作符必须用起码一个空格与其操作数分开。

字节分离操作符 (HIGH, LOW) 将一个绝对的十六位值分成高八位和低八位。若操作数为浮动值，则 HIGH 和 LOW 将视其为相对于零单元来处理。

2.4.2 方式

在表达式中用作为操作数的全部符号都是下列方式之一：绝对的，数据相对的，程序(代码)相对的或公用的（见2.6节中，对 ASEG、CSEG、DSEG 和 COMMON 伪操作的描述）。在 ASEG、CSEG（缺省）或 DSEG 伪操作下汇编的符号，分别为绝对的、代码相对的或数据相对的方式。一个程序中 COMMON 方式数决定于已经由 COMMON 伪操作命名的 COMMON 块数。不在同一 COMMON 块中的两个 COMMON 符号不能同方式。

在除加法或减法外的任何操作中，两个操作数的方式都必须是绝对的。

若为加法操作，则适用如下规则：

1. 至少有一个操作数是绝对的。
2. 绝对的 + 〈方式〉= 〈方式〉

若为减法操作，则适用如下规则：

1. 〈方式〉- 绝对的 = 〈方式〉

2. 〈方式〉-〈方式〉= 绝对的（其中两方式相同）

表达式计算中的每一个中间步骤必须遵循上述有关方式的规则，否则会产生错误。例如，若FOO、BAZ 和 ZAZ 是三个程序相对符号，则表达式

FOO + BAZ - ZAZ

因为第一步（FOO + BAZ）将两个浮动值相加（两值之一必须是绝对的）而产生错误。这个问题可通过插入括号解决。在表达式

FOO + (BAZ - ZAZ)

中，由于第一步（BAZ - ZAZ）生成一绝对的值，然后将此值与前面的程序相对值 FOO 相加，所以是合法的。

2.4.3 外部符

除运用方式进行分类以外，一个符号还分为外部的或非外部的（见2.6.12节 "EXT/EXTRN"）。一个外部值必须被汇编到两字节的字段中（不支持单字节外部符）。在表达式中应用外部符的规则如下：

1. 外部符只在加、减法中是合法的。
2. 若外部符用于一表达式中，则该表达式的结果也总是外部的。
3. 在加法运算时，任意一个（但不是两个）操作数可以是外部的。
4. 在减法运算时，只有第一个操作数可以是外部的。

2.5 操作码和操作数

8080操作码是合法的单字节操作数。注意只有第一个字节是合法操作数。例如：

```
MVI     A, (JMP)
ADI     (CPI)
MVI     B, (RNZ)
CPI     (INX H)
ACI     (LXI B)
MVI     C, MOV A, B
```

若操作数中多于一个字节——例如(CPI 5)，LIX B，LABEL 1)或 (JMP LABEL 2)，则将产生错误。

用作为单字节操作数的操作码不必括在括号中。

注：在 Z80 方式中，操作码不是合法的操作数。

2.6 伪码操作

2.6.1 ASEG

ASEG 置指令计数器到内存的一个绝对段中。这个绝对计数器的单元将是最后一个 ASEG 的单元（缺省为 0），除非在 ASEG 之后，又做了 ORG 而改变了该单元。ASEG 的效果在 LINK-80 中也使用码段（CSEG）伪操作和/P 开关获得。见2.6.28节。

2.6.2 COMMON

COMMON/〈块名〉/

COMMON 置指令计数器为存储区中选定的公用块的地址。该地址单元总是该区域的起始地址，以便保持与 FORTRAN 的 COMMON 语句兼容。若〈块名〉被省略或由空格组成，则它被认为是空的公用块。见2.6.28节。

2.6.3 CSEG

CSEG

CSEG 置指令计数器为内存中的代码相对段的地址。该地址单元将是最后一个 CSEG（缺省为零）的地址，除非在 CSEG 之后又做了 ORG 而改变了该单元。CSEG 是汇编程序的缺省条件（INTEL 汇编程序缺省为 ASEG）。见2.6.28节。

2.6.4 DB——定义字节

DB〈表达式〉[，〈表达式〉…]

DB〈串〉[〈串〉…]

DB 的变元量可以是表达式，也可以是字符串。DB 将这些表达式的值或串中字符存于以指令计数器的当前值为起始地址的连续存储区中。

表达式计算结果必须在一个字节中（若结果的高位字节为 0 或 255 则不出错，否则有一个 A 错误）。

大于等于三个字符数的串不可用在表达式中（即，它们后面必须跟一个逗号或行结束）。一个串的字符按出现的先后顺序存储，每个字符作为一个字节值，且高位补零。

例：

```
0000' 41  42      DB      'AB'
0002' 42          DB      'AB'AND 0FFH
0003' 41  42  43  DB      'ABC'
```

2.6.5 DC——定义字符

DC〈串〉

DC 将〈串〉中的字符存入以指令计数器当前值为开始地址的连续存储区中。同DB 一样，字符以其出现先后顺序进行存储，每个都作为单字节的值，且高位补零。不过，DC 存储串中最后一个字符时将高位置位为 1，若 DC 的自变量是一空字符串，将出现错误。

2.6.6 DS——定义空间

DS 〈表达式〉

DS 保留一个存储区。〈表达式〉的值给出分配的字节数。〈表达式〉中使用的所有名都必须是预先定义过的（即，在第一遍扫描时，全部名字都已知道）。否则，在第一遍扫描时会产生 V 错误，而在第二遍扫描时会产生 U 错误。若第二遍扫描时，未产生 U 错误，则由于 DS 在第一遍扫描时未产生代码，可能在第二遍扫描时产生一个 P 错误。

2.6.7 DSEG

DSEG

DSEG 置指令计数器为内存中数据相对段的地址。此数据相对计数器的地址单元是最后一个 DSEG 单元（缺省为 0），除非在 DSEG 之后又做了 ORG 而改变了该地址单元。见2.6.28节。

2.6.8 DW——定义字

DW 〈表达式〉[，〈表达式〉…]

DW 将表达式的值存入以指令计数器当前值为开始地址的连续存储区中。表达式作为双字节（一个字）值进行计算。

2.6.9 END

END 〔〈表达式〉〕

END 语句指出程序的结束。若出现〈表达式〉，则它是程序的开始地址。若不出现〈表达式〉，则对该程序没有开始地址传递给 LINK-80。

注：若主程序是一汇编语言程序，则必须指定开始地址（标号）。否则，LINK-80将发出“无开始地址”错。若程序是FORTRAN子程序，则由于 FORTRAN 已经提供了开始地址，故不需再指定。

2.6.10 ENTRY/PUBLIC

ENTRY 〈名〉〔，〈名〉…〕

或 PUBLIC 〈名〉〔，〈名〉…〕

ENTRY 或 PUBLIC 说明所列出的每个名为内部名，并因此可利用来给本程序和装入的其他程序共同使用。列出的所有名都必须在当前程序中定义，否则会出现 U 错误。若为外部名或公共用名，则会产生一个 M错误。

2.6.11 EQU

〈名〉EQU 〈表达式〉

EQU 赋〈表达式〉的值给〈名〉。若〈表达式〉是外部的，则会产生错误。若〈名〉已有了一个不等于〈表达式〉的值，则产生一个M错误。

2.6.12 EXT/EXTRN

EXT 〈名〉〔，〈名〉…〕

或 EXTRN 〈名〉〔，〈名〉…〕

EXT，EXTRN 说明所列出的〈名〉为外部名（即定义在另一个程序中）。若表中的任何一项企图引用在当前程序中被定义的名，则产生 M 错误。引用一个其后紧跟两个英镑符号（例如，NAME＃＃）的名，也说明该名为外部名。

2.6.13 INCLUDE

INCLUDE 〈文件名〉

INCLUDE 伪操作汇编另外一个源文件中的源语句到当前源文件中。用 INCLUDE 语句可以使经常用到的语句序列不必在当前的源文件中重复出现。伪操作 INCLUDE、$INCLUDE 和 MACLIB 同义。

〈文件名〉是由操作系统决定的任意合法说明。缺省的文件名扩展及设备名都与MACRO-80命令行中的相同。

INCLUDE 文件被打开并被汇编到当前源文件之中，紧跟在 INCLUDE 语句之后。在到达文件结束时，从 INCLUDE 之后的语句继续汇编。

在 MACRO-80 列表文件中，在从 INCLUDE 文件汇编出的每一行中，汇编码和源文件行之间必须打印一个“+”号（见2.12节）。

INCLUDE 不允许嵌套。若出现嵌套，将产生一个不能采用的语法错误‘O’。

操作数段中指定的文件必须存在。若该文件未找到，则给出错误‘V’（值错），并且该INCLUDE也被忽略。

2.6.14 NAME

NAME（‘模块名’）

NAME 为模块定义名字。模块名中只有前六个字符是有效的。模块名也可用 TITLE

伪操作进行定义。在NAME和TITLE伪操作都不出现时，模块名就可以从源文件名中得到。

2.6.15 ORG——定义起始地址

ORG 〈表达式〉

指令计数器置为〈表达式〉的值，同时汇编程序以该值赋予生成代码。用于〈表达式〉中的所有名在第一遍扫描中必须是已知的，并且其值必须是绝对的或是在与指令计数器相同的域中。

2.6.16 PAGE

PAGE〔〈表达式〉〕

PAGE使汇编程序开始一新的输出页。如果包含〈表达式〉，则表达式的值为新页的长度，并且必须是10到255之间（按每页的行数计）的数。缺省的页长度为每页50行。汇编程序将列表文件中的走纸符置于一页的结尾处。

2.6.17 SET

〈名〉SET〈表达式〉

SET与EQU相同，不同的是如果〈名〉已经定义，则不会产生错误。

2.6.18 SUBTTL

SUBTTL〈正文〉

SUBTTL指出在每页标题行总标题之后列出小标题(见2.6.19节“TITLE”)。〈正文〉的60个字符之后的部分将被截断。一个程序中可给定任意多的小标题。

2.6.19 TITLE

TITLE 〈正文〉

TITLE指定要打印在每页第一行上的标题。若给出了多个TITLE，则产生Q错误。如果不使用NAME伪操作，则标题的前六个字符被作为模块名。若NAME和TITLE伪操作都未使用，则把源文件名取为模块名。

2.6.20 .COMMENT

.COMMENT〈定界符〉〈正文〉〈定界符〉

在.COMMENT之后遇到的第一个非空字符是定界符。后面的〈正文〉组成一个注释块，它一直延续到下一个〈定界符〉出现为止。例如，使用一个星号作为定界符，注释块的格式将为：

```
.COMMENT *
输入的任何文字
此处作为注释块
        ⋮ *
; 返回到正常方式
```

2.6.21 .PRINTX

.PRINTX 〈定界符〉〈正文〉〈定界符〉

.PRINTX之后的第一个非空字符为定界符。其后的正文在汇编期间显示在终端上，直到遇见另一个定界符为止。. PRINTX对于显示一长汇编过程以及显示汇编的条件开关值等都是有用处的。例如：

```
IF  CPM
.PRINTX/CPM 版本/
ENDIF
```

注：

.PRINTX 将在两次扫描中都输出。若只希望一次打印输出，则使用IF 1 或IF 2 伪操作。例如：

```
IF 2
IF  CPM
.PRINTX/CPM 版本/
ENDIF
ENDIF
```

仅当 CPM 是真且 M80 是在第二次扫描中时打印。

2.6.22 .RADIX

.RADIX〈表达式〉

对于所有的常数，缺省的基数（即数系）为十进制。.RADIX 语句允许将该缺省基数改变为 2 到16中的任何基数。例如：

```
MVI     B, OFFH
.RADIX  16
MVI     B, OFF
```

例子中的两个 MVI 相同。在.RADIX 语句中的〈表达式〉总是以十进制为基数，而不管当前的基数。

2.6.23 .Z80

.Z80 使汇编程序能接收 Z80 操作码。当汇编程序在 Z80 操作系统下运行时，这是一缺省条件。Z80 方式也可通过在 MACRO-80 命令串上附加Z开关来设置，见2.2.2节。

2.6.24 .8080

.8080 使汇编程序能够接收8080操作码。当汇编程序在8080操作系统下运行时，这是一个缺省条件。8080方式也可通过在 MACRO-80 命令串上附加 I 开关来设置，见2.2.2节。

2.6.25 .REQUEST

.REQUEST〈文件名〉[,〈文件名〉…]

.REQUEST 送一个请求给 LINK-80 装入程序查找列出的文件名，以找出未定义的全局符。表中文件名应是合法的符号形式。应不包含文件扩展名和磁盘说明。LINK-80支持缺省扩展名并假定其为缺省磁盘驱动器。

2.6.26 条件伪操作

条件伪操作有：

IF/IFT〈表达式〉	若〈表达式〉≠0，为真
IFE/IFF〈表达式〉	若〈表达式〉= 0，为真
IF 1	第一次扫描为真
IF 2	第二次扫描为真
IF DEF〈符号〉	若〈符号〉已被定义或已经说明为外部符时，为真

IFNDEF〈符号〉　　若〈符号〉未定义或未被说明为外部符时，为真

IFB〈变元〉　　若〈变元〉是空，为真。变元两边要用尖括号〈 〉

IFNB〈变元〉　　若〈变元〉非空，为真。在提供伪参数时使用，用于测试在变元两边要有尖括号〈 〉

所有条件都使用下列格式：

```
IFxx 〈变元〉
    ⋮
〔ELSE
    ⋮〕
END IF
```

条件可以嵌套任意层。条件中的任何变元在第一次扫描中都必须是已知的，以避免V错误和不正确的计算，对于IF、IFI、IFF和IFE来说，表达式必须包含前面已定义的值，并且表达式必须是绝对的。若名是在IFDEF或IFNDEF之后定义的，则第一次扫描认为该名是未定义的，但它将在第二遍扫描时被定义。

2.6.26.1　ELSE

每个条件伪操作均可任选地使用ELSE伪操作，它允许在对立条件成立时，生成替换代码。一个给定的IF只允许有一个ELSE，并且一个ELSE也总与最近的开IF相对应。一个条件有多个ELSE或有一个ELSE而没有条件，都将导致一个C错误。

2.6.26.2 ENDIF

每个IF都有一个对应的ENDIF来结束该条件。否则，在每次扫描的结尾都将产生一个'未结束条件'信息。一个ENDIF如果没有与之匹配的IF，将导致C错误。

在3.4版本中还补充了两个条件伪操作：

IFIDN〈变元1〉,〈变元2〉

当串〈变元1〉等同于〈变元2〉时为真，要求有尖括号。

IFDIF〈变元1〉,〈变元2〉

当串〈变元1〉与串〈变元2〉不相同时为真。要求有尖括号。

2.6.27 列表控制伪操作

列表文件的输出可通过两种伪操作进行控制：

.LIST 和.XLIST

若不是正在生成列表文件，则这些伪操作无效。.LIST为缺省条件。在遇到.XLIST时，将不列出源码和目标码，直至遇到一个.LIST为止。

相互对照信息的输出受.CREF和.XCREF的控制。若未调用相互对照功能，则.CREF和.XCREF无效。缺省条件为.CREF。在遇到一个.XCREF时，不输出相互对照信息，直至遇到.CREF为止。

MACRO/REPT/IRP/IRPC扩展的输出受三种伪操作.LALL，.SALL和.XALL的控制。.LALL列出全部扩展的完整的宏文本。.SALL只列出由宏指令生成的目标码而不列出宏文本。.XALL为缺省条件，它类似于.SALL，但只有在其生成目标码时才列出源文件。

在3.4版本中，还补充了如下三种列表的操作：

.SFCOND, .LFCOND, .TFCOND

这三种伪操作控制等于假的条件伪操作组的列表。它们在四种情况下给出程序员控制。

1. 正常列出假条件

对于这种情况，程序可以缺省方式去控制列表。缺省方式是列出假条件。若程序员决定抑制假条件，则可在命令行中发出/X 开关以代替编辑源文件。

2. 正常抑制假条件

对这种情况，程序员在程序文件中发出.TFCOND 伪操作。.TFCOND 改变缺省值，抑制假条件。如果程序员决定列出假条件，则可以在命令行中发出 /X 开关以代替编辑源文件。

3. 总是抑制或列出假条件

对于这些情况，程序员或发出.SFCOND 伪操作去抑制假条件，或发出.LFCOND 伪操作去列出全部假条件。

4. 抑制或列出一些假条件

对这种情况，程序员已决定了大多数假条件是列出还是抑制，但对少数假条件，则尚未决定。对已决定的假条件，使用.SFCOND 或.LFCOND。对尚未决定的假条件，使用.TFCOND。.TFCOND 把当前方式和缺省值设置为与原缺省值相反。开始时，缺省值通过在命令行中给出/X 或无/X来设置。存在两种情况：

1. 程序员不想列出一些假条件，除非给出/X 。程序员用.SFCOND 和.LFCOND伪操作控制哪些区域总是抑制或列出假条件。程序员可在条件组的开始和结尾处都发出.TFCOND 来有选择地抑制一些假条件。若条件组为假，则这些行不列出。在这种情况下，在命令行中发出/X 开关导致条件组受.TFCOND 影响，即使为假也列出。

2. 程序员在不给出/X 时，想要列出一些假条件，两个连续的 .TFCOND 将条件列表设置为初始态（由/X 开关的有无决定）。然后，选定的条件组对/X 开关作出响应：若在命令行中发出一个/X 开关，则当假时，条件组被抑制；若命令行中无/X 开关，则即使是假，也要列出条件组。

然后程序员必须重发.SFCOND 或.LFCOND 条件列表伪操作以恢复抑制或列出方式。仅发出另一个 .TFCOND 将不能恢复前面的方式，但将转换缺省设置。由于在此种情况中，下一个代码域总是被假定为列出或抑制假条件，所以程序员必须发出.SFCOND 或.LFCOND。

上述三种条件列表伪操作概述如下：

伪操作	定义
.SFCOND	抑制等于假的条件组的列表
.LFCOND	恢复等于假的条件组的列表
.TFCOND	将控制列出假条件的当前值转变为相反值。.TFCOND 将当前和缺省值置为非缺省值。对于含有.TFCOND 的文件，若在 MACRO-80运行命令行中给出一/X 开关，则/X 使.TFCOND 的作用相反。

2.6.28 重定位伪操作

可以建立浮动模块是Microsoft汇编程序的主要特性之一。浮动模块提供容易编码，可迅速检测、调试和修改的优点。另外，可以指定汇编码段，在以后装入到RAM(数据相关段)

和ROM/PROM（代码相关段）中，选择可重定位域的伪操作是CSEG和DSEG。ASE G 伪操作用于生成非浮动（绝对）码。COMMON 伪操作为在程序中命名的每个公用块建立一个公用数据区。

汇编程序的缺省方式为代码相关。即，汇编开始时，自动执行一个CSEG伪操作，同时代码相关方式的指令计数器指向内存代码相关段中的 0 单元。所有后继指令都将被汇编到内存中的代码相关段中，直至执行了ASEG或DSEG 或COMMON伪操作为止。例如，第一个遇到的DSEG伪操作置指令计数器的值为内存中数据相关段中的 0 单元。其后的代码以数据相关方式进行汇编，即，它被赋给内存中数据相关段。若遇到后继的CSEG伪操作，指令计数器将返回到代码相关段中下一个自由单元。依此类推。

ASEG, DSEG, CSEG伪操作没有操作数。若需要改变指令计数器当前值，可用ORG伪操作。

2.6.28.1 ORG 伪操作

任何时候，指令计数器的值都可通过 ORG 伪操作进行改变。ORG 语句的格式为：

ORG〈表达式〉

此处〈表达式〉的值将是当前方式下指令计数器的新值。用于〈表达式〉中的所有名必须在第一次扫描中为已知，且〈表达式〉的值必须是绝对的或与指令计数器的当前方式相同。例如：

```
DSEG
ORG  50
```

语句置数据相关指令计数器为50，相对于内存中数据相关段的起始地址。

2.6.28.2 LINK-80

LINK-80 连接装入程序（见本章第四节）在装入程序的时候，组合各段并在存储器中建立各自的浮动模块。浮动段的起始地址在程序被装入并由 LINK-80分配起始地址之前是不固定的。LINK-80命令可以通过使用/P（用于代码相关）和/D（用于数据和公用段）开关，由用户指定起始地址。

例如，开始语句为

```
ASEG
ORG  800H
```

同时又完全以绝对方式汇编的一个程序将总是装在开始地址为 800 的存储区中，除非在源文件中改变了该 ORG 语句。不过，同样的程序以代码相关方式汇编并且无 ORG 语句，则可以通过在 LINK-80 命令串上附加/P：〈地址〉开关装入到指定的任何地址处。

2.6.29 装入前的重定位

两个伪操作.PHASE 和.DEPHASE 允许将代码装在同一域中，但只可在不同的指定域中执行。例如：

```
0000′                          .PHASE    100H
0100    E8  0003    FOO:       CALL      BAZ
0103    E9  FF01               JMP       ZOO
0106    C3          BAZ:       RET
                               .DEPHASE
0007′   E9  FFFB    ZOO:       JMP       5
```

一个.PHASE 组中的所有标号都被定义为以该程序段域起始地址算起的绝对值。代码则装入到当前域中（本程序中，即从'0' 开始）。该组中的代码可在以后送到100H 处并执行之。

2.7 宏和块伪操作

由 MACRO-80提供的宏功能包括三个重复的伪操作：重复 (REPT)，不定重复(IRP)和不定重复符 (IRPC) 。还提供宏定义操作 (MACRO) 。这四种宏操作中的每一个都以 ENDM 伪操作终止。

2.7.1 项

在讨论宏和组操作时，将要使用如下项：

1. 〈dummy〉用来代表一空参数。所有出现在宏扩展体中的空参数都是合法的符号。

2. 〈dummylist〉是用逗号分隔的一系列空参数。

3. 〈arglist〉是由逗号分隔的一系列变元。〈变元表〉必须用尖括号括起来。两个尖括号中无字符(〈 〉)或两个逗号之间无字符表示输入一个空变元。而一个变元是由一个〈逗号或〉结束的一个字符或一系列字符。如果尖括号嵌套在 〈变元表〉中，则每次括号中变元用于〈变元表〉时，去掉一层括号 (见 2.7.5 节实例)。可以接受带引号的串作为变元。变元中的前导及尾随空格被删去（除非在封闭的括号中或在带引号串内）。

4. (Paramlist) 用于表示一系列由逗号分隔的实际参数。参数表由行结尾或注释结尾而终止，无需定界符。进入空参数和嵌套括号的规则如 (arglist) 中所述。

2.7.2 REPT-ENDM

```
REPT  〈表达式〉
        ⋮
ENDM
```

在 REPT 和 ENDM 之间的语句组被重复由〈表达式〉值确定的次数。〈表达式〉是作为一个十六位无符号数参加运算的。若〈表达式〉包括任何外部符或未定义项，则产生错误。例如：

```
SET     0
REDT    10      ；生成 DB1-DB10
SET     X+1
DB      X
ENDM
```

2.7.3 IRP-ENDM

```
IRP 〈空参数〉〈变元表〉
  ⋮
ENDM
```

〈变元表〉必须包含在尖括号中。〈变元表〉中变元的个数决定语句组被重复执行的次数。每次重复用〈变元表〉中下一项替换语句组中〈空参数〉的每个具体值。若〈变元表〉为空(即〈 〉)，则该语句组处理一次，将〈空参数〉每个具体值取消。例如：

```
IRP     X,〈1，2，3，4，5，6，7，8，9，10〉
DB      X
ENDM
```

产生与 RFPT 例子中相同的字节。

2.7.4 IRPC-ENDM

IRPC〈空参数〉，串（或〈串〉）

⋮

ENDM

IRPC 类似于 IRP，但是变元表可由一个文字串或外加尖括号的文字串替换。组中的语句对串中的每个字符重复执行一次。每次重复用串中的下一个字符取代语句组中〈空参数〉的每个具体值。例如：

```
IRPC    X, 0 1 2 3 4 5 6 7 8 9
DB      X+1
ENDM
```

产生与前面两例相同的代码。

2.7.5 MACRO

通常从程序的不同地方生成一个给定的语句序列是很方便的，尽管在每次使用该序列时可能要求不同的参数。MACRO 语句提供了这种可能性，其格式为：

〈名〉 MACRO 〈空参数表〉

⋮

ENDM

其中〈名〉遵守符号形成规则。〈名〉是调用宏功能所用的名。〈空参数表〉中的〈空参数〉是在每次调用 MACRO 时将被替换的参数。在 ENDM 前的语句组成宏体。在汇编期间，宏功能在每次调用时被扩展，不象 REPT/IRP/IRPC，遇到宏功能时并不扩展。

宏调用的格式为：

〈名〉 〈参数表〉

其中，〈名〉是在宏定义中提供的名，并且〈参数表〉中的参数将一对一地代换 MACRO〈空参数表〉中的〈空参数〉。在〈空参数表〉和〈参数表〉中项的数目仅受到行长度的限制。在进行宏调用时所用的参数数目不必与〈空参数表〉中的〈空参数〉数目相同。若参数比〈空参数〉多，则多余的参数被忽略。若参数比〈空参数〉少，则多余的〈空参数〉被置为空。在每次宏调用之后，汇编代码将包含宏扩展码。

注：一个 MACRO/REPT/IRP/IRPC 中的空参数总被唯一地视为一空参数。寄存器名如 A 和 B 等，若被用作为空参数，则在扩展时改变。

下面是 MACRO 定义的例子，它定义了一个称为 FOO 的宏：

```
FOO   MACRO   X
Y     SET     0
      REPT    X
Y     SET     Y + 1
      DB      Y
      ENDM
      ENDM
```

这个宏在执行

```
FOO   10
```

调用时，生成与前面三个例子相同的代码。

另一个例子，也生成同样的码，该例指出当一个变元用作变元表时，则去掉一层括号。

```
FOO     MACRO  X
        IRP    Y〈X〉
        DB     Y,
        ENDM
        ENDM
```

在调用

```
FOO〈1,2,3,4,5,6,7,8,9,10〉
```

被执行时，宏展开的形式为：

```
IRP  Y, 〈1,2,3,4,5,6,7,8,9,10〉
DB   Y
ENDM
```

2.7.6 ENDM

所有 REPT, IRP, IRPC 和 MACRO 伪操作都必须终止于 ENDM 伪操作。否则在每次扫描结束时，将会产生‘未结束的 REPT/IRP/IRPC/MACRO’信息。一个不匹配的 ENDM 将导致 O 错误。

2.7.7 EXITM

EXITM 伪操作用于终止一个 REPT/IRP/IRPC 或 MACRO 调用。在执行 EXITM 时，立即退出扩展，且剩余的扩展或重复不再生成。若一个包含 EXITM 的语句组嵌套在另一语句组中，则外层的语句组继续被扩展。

2.7.8 LOCAL

LOCAL 〈空参数表〉

LOCAL 伪操作只允许用于 MACRO 定义内部。在执行 LOCAL 伪操作时，汇编程序为〈空参数表〉中的每个〈空参数〉建立一个唯一的符号，并将该符号代换宏扩展的每个具体值。这些唯一的符号常常用于定义宏功能中的标号。避免在连续的宏扩展中使用多重定义的标号。由汇编程序建立的符号范围从‥0001 到‥FFFF。因此，用户应想办法避免以‥nnnn 作为自己的符号形式。若使用 LOCAL 语句，则必须作为宏定义中的第一个语句。

2.7.9 特殊宏操作符和格式

& 用于宏扩展中以连接文本或符号。在宏扩展中，一个括号内的字符串空参数仅当前面有 &符号时才能被代换。为了从文本和空参数形成一个符号，则要在它们之间放一个&。例如：

```
ERRGEN      MACRO    X
ERROR&X:    PUSH     B
            MVI      B,'&X'
            JMP      ERROR
            ENDM
```

在此例中，调用 ERRGNA 将产生：

```
ERRORA:     PUSH     B
```

```
        MVI         B,'A'
        JMP         ERROR
```

;;　在成组操作中，前面有两个分号的注释不作为扩展的一部分保留（就是说，即使在.LALL 情况下，它也不会出现在列表中）。但是，前面有一个分号的注释将被保留并出现在扩展中。

!　在一个变元中使用叹号时，下一个输入的字符作为文字(即，！；和<；>相等)。

NUL　是一个操作符，当其变元（一个参数）为空时，它返回真值。一行中 NUL 后面的剩余部分被认为是 NUL 的变元。若在扩展时，变元的第一字符不是一个分号或回车，则条件

```
IF  NUL 变元
```

为假。使用 IFB 和 IFNB 条件做空参数的测试是比较合适的。

%　百分号仅用于宏变量中。%将其后的表达式（通常是一个符号）转换成当前数制中的一个数。在宏扩展过程中，从表达式转换得到的数用于代换空参数。使用%这一特殊操作符允许按值进行宏调用。在%之后的表达式必须与 DS 伪操作的规则相同。要求有一个返回非浮动常数的合法表达式。

举例：

通常，MAKLAB 的变元 LB，用于替换 MACRO 的变元 Y 作为一个字符串。%使 LB 转换为一非浮动常数，然后取代。Y没有%特殊操作符，汇编结果将是'Error LB'，而不是'Error 1'，等等。

```
MAKLAB  MACRO  Y
ERR&Y:  DB     'Error &Y',0
        ENDM
MAKERR  MACRO  X
LB      SET    0
        REPT   X
LB      SET    LB+1
        MAKLAB %LB
        ENDM
        ENDM
```

当通过 MAKERR 3 调用时，汇编程序将产生：

```
ERR1:   DB     'Error 1',0
ERR2:   DB     'Error 2',0
ERR3:   DB     'Error 3',0
```

TYPE　返回一个字节以描述其变元的特性：1）方式，2）是否为外部的。TYPE的参数可以是任何表达式（字符串，数字式，逻辑式）。若表达式无效，则TYPE返回零。返回的字节构成如下：

两个低位是方式，如果两个低位是：

0　方式为绝对的

1　方式为程序相关的

2　　方式为数据相关的

3　　方式为公用区相关的

则两个高位（80H）是外部位。若高位为1，则表达式包含一外部符。若高位为0，则表达式为局部的（非外部）。

定义位是20H。若表达式是局部定义的，则该字位为1；若表达式是未定义的或外部的，则该位为0；若两字位均为0，则表达式非法。TYPE通常用于宏的内部，其中需要对一个变量类型进行测试以决定程序流程。例如：

```
FOO     MACRO  X
        LOCAL  Z
Z       SET  TYPE  X
IF      Z ...
```

2.8 使用Z80伪操作

在使用MACRO-80汇编程序时，下面的Z80伪操作是合法的。每个伪操作的功能等价于它们的对应部分。

Z80伪操作	等价的伪操作
COND	IFT
ENDC	ENDIF
*EJECT	PAGE
DEFB	DB
DEFS	DS
DEFW	DW
DEFM	DB
DEFL	SET
GLOBAL	PUBLIC
EXTERNAL	EXTRN

出现不同的格式时与其前一个的格式一致。即，DFFB和DEFW允许一个变元表（同DB和DW），DEFW允许一个串或数值变元（同DB）。

2.9 汇编举例

```
A>M80
*EXMPL1,TTY:=EXMPL1
   MAC80  3.2 PAGE   1
                     00100    ; CSL3(P1,P2)
                     00200    ; SHIFT P1 LEFT  CIRCULARLY 3
              BITS
                     00300    ; RETURN RESULT IN P2
                     00400    ENTRY CSL3
                     00450    ; GET VALUE OF FIRST
PARAMETER
                     00500    CSL3:
```

```
0000'    7E          00600          MOV A,M
0001'    23          00700          INX H
0002'    66          00800          MOV H,M
0003'    6F          00900          MOV L,A
                     01000    ; SHIFT COUNT
0004'    06 03       01100          MVI B,3
0006'    AF          01200    LOOP: XRA A
                     01300    ; SHIFT LEFT
0007'    29          01400          DAD H
                     01500    ; ROTATE IN CY BIT
0008'    17          01600          RAL
0009'    85          01700          ADD L
000A'    6F          01800          MOV L, A
                     01900    ; DECREMENT COUNT
000B'    05          02000          DCR B
                     02100    ; ONE MORE TIME
000C'    C2 0006'    02200          JNZ LOOP
000F'    EB          02300          XCHG
                     02400    ; SAVE RESULT IN SECOND
PARAMETER
0010'    73          02500          MOV M, E
0011'    23          02600          INX H
0012'    72          02700          MOV M,D
0013'    C9          02800          RET
                     02900 END
   MACRO 3.2 PAGE   S
CSL3 00001' LOOP 0006'
No Fatel Error(S)
```

（无严重错误）

2.10 MACRO-80错误

MACRO-80错误由列表文件第一列上的一个字符标识指出。若列表文件没有在终端上显示，则在终端上打印或显示每个错误信息行。下面是MACRO-80的错误码表。

A　变元错

伪操作中的变元格式不正确或越界（.PAGE 1;.RADIX 1;PUBLIC 1;JMPS TOOFAR）

C　条件嵌套错

有ELSE无IF，有ENDIF无IF，两个ELSE对应一个IF。

D　双重定义符号

引用一个重复定义符号。

E　外部错

使用了上下文中非法的外部符(例如:FOO SET NAME##; LIX B, 2 - NAM##E)。

M　多重定义符号

对一个符号的定义是重复的。

N　数字错

出现在数字中的错，通常是一个错误数字（例如8Q）。

O　错操作码或不适当的语法

ENDM,LOCAL出现在语句组外,SET,EQU或MACRO中没有名;一个操作码中的语法错;一个表达式中的语法错(不匹配的括号、引号、连续操作符等等)。

P　标号值错

一个标号的值或EQU名在第二次扫描时不同。

Q　疑问

通常指一行没有正常结束。这是一个警告性错误（例如MOV AX, BX,）

R　浮动

在表达式中非法使用再定位，例如abs-rel。数据，代码和公用区是浮动的。

U　符号未定义

在表达式中引用的符号未定义。（对某些伪操作，在第一次扫描显示V 错误，在第二次扫描显示U 错误。）

V　值错

一个在第一次扫描中应必须有已知值的伪操作（例如，.RADIX, .PAGE, DS, IF, IFE, 等等），在第一次扫描中有一个未定义的值，若该符号在该程序的后面定义，则在第二遍扫描的列表中将不出现U错误。

错误信息:

'No end statement encountered on input file'

无END语句：或是漏掉了，或是由于处在一假条件中未终结的IRP/IRPC/REPT 语句组中或已终结的宏功能中，因而未进行语法分析。

'Unterminated conditional'

在文件的结尾，至少有一个条件未结束。

'Unterminated REPT/IRP/IRPC/MACRO'

至少有一语句组未终结。

〔xx〕〔NO〕Fatal error (s) 〔, xx warnings〕

严重错误和警告性错误数目。信息被列在CRT和列表文件中。

2.11 与其他汇编程序的兼容性

MACRO-80提供的$EJECT和$TITLE控制与INTEL ISIS汇编程序相兼容。但仅当美元符与控制字被空格或标记分开时，美元符号才必须出现在第一列。控制

$EJECT

与MACRO-80的PAGE伪操作相同。控制

$TITLE ('正文')

与MACRO-80的 SUBTTL〈正文〉伪操作相同。

INTEL操作数PAGE和INPAGE在与MACRO-80的 CSEG或 DSEG 伪操作一起使用时产生 Q 错误。这些错误是警告性的，汇编程序忽略这些操作数。

在进入 MACRO-80 时，起始地址缺省为代码相关 0 。而对于 INTEL 的 ISIS 汇编程序，缺省为绝对 0 。

对于 MACRO-80，美元符（$）是给定常数，它指定语句开始时指令计数器的值。其他汇编程序可用十进制小数点或星号。由 MACRO-80定义的其他常数有如下值：

B = 0　　D = 2　　H = 4　　M = 6　　PSW = 6

C = 1　　E = 3　　L = 5　　SP = 6　　A = 7。

2.12 列表格式

在 MACRO-80列表文件的每一页上，最开始两行的格式为：

〔TITLE　正文〕M80　3.3　PAGE　X〔-Y〕

〔SUBTTL正文〕

其中：

1. 若在源程序中给定了 TITLE 伪操作，则 TITLE 正文是提供给 TITLE 伪操作的正文。

2. X 是正页号，它只在源文件中遇到一个格式馈给时，才被增值（在使用 Microsoft 的 EDIT-80文本编辑程序时，在每个页标志出现时插入一格式馈给）。正在打印符号表时，X = S。

3. Y 是次页号，不论何时，在源文件遇到 .PAGE 伪操作或当页的长度已被填满时，Y 就被增值。

4. 若在源程序中给出了 SUBTTL 伪操作，则 SUBTTL 正文是提供给 SUBTTL 伪操作的正文。

其次，在打印输出第一行前打印一空行。

在 MACRO-80列表中的输出行格式为：

〔crf#〕〔error〕loc#m/xx |xxxx| …source

若正在输出相互对照信息，则该行第一项是相互对照号，后面跟一标记。

若一行包含有错误，则在行下一位置将出现一个字母错误码跟一个空格；若无错误，则打印一个空格。若无相互对照号，则错误码所在行即该列表的第一列。

指令计数器的值出现在该行的下一位置处。它是一个四位数字的十六进制数或六位数字的八进制数，这决定于在 MACRO-80命令串中是否给了/O 或/H 开关。

在指令计数器值结尾处的字符是方式指示符。为下列符号之一：

′　　代码相对

″　　数据相对

!　　公用区相对

〈空格〉 绝对的

*　　外部的

然后，空三个格，打印汇编码。单字节值后跟一空格。双字节值后跟一方式指示符。双字节值打印顺序与其存储顺序相反，即高位字节先被打印。外部符或是偏移值，或是指向链中下

一个外部符的指针值。

如果 MACRO-80列表上的一行输出是来自 INCLUDE 文件，则在该行的汇编码之后，打印一个字符'C'。如果一行输出是一个正文扩展（MACRO, REPT, IRP, IRPC）的一部分，则在该行的汇编码之后，打印一个加号'+'。

该行的剩余部分包含源码行，与其输入时一样。例如：

0C49 3 A A91Z' C+ LDA LCOUNT

'C+' 指出这行是来自INCLUDE 文件，且是宏展开的一部分。

2.12.1 列出符号表

在符号表清单中，按字母顺序列出该程序中全部宏功能名，后面按字母顺序列出该程序中的全部符号。在每个符号之后打一个标记，其后为该符号的值。若符号为 PUBLIC，则在其值后立即打印一个 I。下一个要打印的字符如下：

U 未定义的符号

C 公用块名（公用块的"值"是它的十六或八进制长度 字节数）。

* 外部符号

〈空格〉绝对值

′ 程序相对值

″ 数据相对值

! 公用区相对值

第三节 CREF-80相互对照功能

注：如果你使用 TEKDOS 操作系统，可参看附录 A 相应的命令格式

为生成相互对照清单，汇编程序必须输出一特殊的带有控制符的列表文件。MACRO-80 命令串告诉汇编程序输出这一特殊列表文件。/C 是相互对照开关。在一个 MACRO-80 命令串中遇到/C 开关时，汇编程序打开.CRF文件而不打开 .LST 文件(见2.6.27节中."CREF 和 .XCREF 伪操作")。例如：

* = TEST/C

汇编 TEST.MAC 文件并建立目标文件 TEST.REL 和相互对照文件 TEST.CRF。

* T.U = TEST/C

汇编 TEST.MAC 文件并建立目标文件 T.REL 和相互对照文件 U.CRF。

在汇编程序结束之后，可通过打入 CREF-80运行相互对照功能。CREF-80 用一个星号提示用户。CREF-80从汇编时建立的.CRF 文件中生成一相互对照表。CREF-80 命令格式为：

* 列表文件 = 源文件

源文件的缺省扩展名为.CRF。在 CREF80 命令中没有开关。CREF-80 命令串举例：

* = TEST

检查 TEST.CRF 文件并生成相互对照列表文件 TEST.LST。

* T = TEST

检查 TEST.CRF 文件并生成相互对照列表文件 T.LST。

相互对照列表文件不同于一般列表文件。在相互对照列表文件中：

1．每个源语句用相互对照号编号。

2．在清单的结尾，按字母顺序出现的变量名旁列出它们被引用或定义处的行号。定义该符号的那些行号用'#'标识。

第四节　LINK-80连接装入程序

注：若使用TEKDOS操作系统，可参看附录A中相应的命令格式。

4.1　运行LINK-80

运行 LINK-80的命令为

L80

LINK-80返回一提示符"*"，表示已准备接受命令。

4.2　命令格式

发给 LINK-80 的每条命令由一串目标文件名组成，文件名之间用逗号分开。这些文件由 LINK-80 装入。命令格式为：

目标文件 1，目标文件 2，…，目标文件n

对于所有文件名，其缺省的扩展名为 REL。可以命令行形式打入命令，即，调用和命令可在同一行上打入。例如：

L80 MYPROG, YRPROG

LINK80命令串中的任何文件名也可指定为设备名。对于 CP/M 操作系统，缺省的设备名是当前登录的磁盘。对于ISIS-Ⅱ操作系统，缺省设备为磁盘驱动器 0 。格式为：

设备 1：目标文件 1，设备 2：目标文件 2，…，设备n:目标文件 n

设备名同2.2.1节中所列。例如：

L80 MYPROG, A:YRPROG

在每行打入之后，LINK-80 将装入该指定文件。在 LINK 处理结束之后，它将列出全部未定义的符号，后面加一个星号：例如：

```
*MAIN
DATA                0100      0200
SUBR1  *            (SUBR1 未定义)
*SUBR1 (用户装入 SUBR1)
DATA                0100      0300
*  (现在所有符号都已定义)
```

通常，为执行 MACRO-80 程序和子例程，用户可以打入一列系文件名，后面跟一个/G（开始执行）。为判定外部的未定义的符号，可首先查找库例程，这可通过在装入程序命令串后附加文件名，再跟一个/S 来执行（见第五节在 CP/M 下用 LIB-80建立用户库）。

```
*MYLIB /S 查找 MYLIB.REL 中没有判定的全局符号
*/G 开始执行
```

4.2.1　LINK-80 开关

在 LINK-80 命令串中可设置一些开关参数指明影响程序装入或执行的动作。每个开关

前必须有一斜线（/）（对 TEKDOS 操作系统，开关之前放一短划。见附录A）。

开关可放在命令串中可使用开关的任何地方：

1．在命令级。一个开关可以是完整的 LINK-80 命令，或出现在命令串的最开始。例如：

*/G　告诉 LINK-80 开始执行已装入的程序

*/M　列出已装入程序中的所有全局说明符

*/P：200，FOO　装入 FOO，其程序区开始于地址200

2．紧跟在文件名后。在命令串中，一个 S 或 N 开关可以只涉及一个文件名。因此，如果需要S或 N 开关，可以紧跟在该文件名后面，而不管该文件名出现在命令串中的什么位置。例如：

*MYLIB/S，MYPROG 查找MYLIB.REL 并装入必要的库模块，然后装入 MYPROG.REL。

*MYPROG，MYPROG/N/E 装入 MYPROG.REL，将 MYPROG.COM 保存在磁盘上并退出LINK-80。

3．在命令串的结束处。影响整个装入过程的开关可附加在命令串的末尾。例如：

*MYPROG/N，MYPROG/M/E打开名为MYPROG.COM的 CP/M COM 文件，装入MYPROG.REL并列出全部全局说明符。保留COM文件并退出LINK-80。

MYLIB/S，MYSUB，MYPROG/N，MYPROG/M/G 查找 MYLIB.REL，装入并连接MYSUB.REL，打开名为MYPROG.COM的CP/M COM 文件，装入并连接 MYPROG.REL，列出全部全局说明符，保存COM文件，执行MYPROG。

可使用的开关有：

开关	动作
R	复位。置装入程序为初始状态。若装入了错误的文件并且想重新开始，可使用开关/R。在一命令串中遇到/R时立即起作用。
E 或 E：名	退出 LINK-80 并返回操作系统环境。在当前磁盘上查找系统库以满足现存的任何未定义的全局符号。在退出 LINK-80 之前，打印三个数字：开始地址、下一可用字节地址以及所用的页数(以 256 字节为一页)。其任选的格式 E：名（名是以前定义在一个模块中的全局符号）使用“名”作为该程序的起始地址。使用开关/E以装入程序并退回到监控程序。
G或G：名	一旦当前命令行被解释之后，立即开始执行该程序。系统库将在当前磁盘上被查找，以满足存在的任何未定义全局符。在实际开始执行之前，LINK-80 打印三个数字以及开始执行信息。三个数为开始地址、下一可用字节地址以及使用的以 256 字节为一页的页数。供选择的格式 G：名(名是一个前面定义在一模块中的全局符)使用名作为程序起始地址。
N	若指定〈文件名〉/N，则在做完/E或/G 的时候，程序将以所选定的文件名保存在磁盘上（对于CP/M，文件名的扩展名是 · COM）。若需要的话，可插入一条转移到程序开始的指令以使程序能顺利执行（对CP/M为100H）。
P和D	/P和/D允许为下一个装入的程序建立起始地址。在遇到 /P 和/D 时（不延迟）立即起作用。/P 和/D 对已装入的程序不产生影响。格式为

/P：〈地址〉或/D：〈地址〉，其中〈地址〉是所期望的起始地址按当前输出的数制形式表示。缺省数制为十六进制，/O设置为八进制，/H为十六进制）。LINK-80执行一个缺省/P：〈连接起始地址〉+3（对CP/M为103H，对ISIS为4003H）以留下空间用于转移到开始地址。

注：不要利用/P或/D将程序或数据装入到装入程序转移至的开始地址单元（对CP/M为100H到102H），除非将该程序的开始装入到这些单元。若程序或数据被装入到这些单元，则不产生转移。

若未给出/D，则对每个模块，在程序区之前，装入一个数据区。若指定/D，则全部数据和公用区从数据初始地址开始装入，而程序区从程序初始地址开始装入。例如：

```
*/P:200, FOO
数据          200  300
*/R
*/P:200/D:400, FOO
数据          400  480
程序          200  280
```

U　在当前命令行被解释之后，立即列出程序和数据区的起始和结束地址以及全部未定义的全局符。仅当执行完/D后，才打印程序信息。否则，该程序被存放在数据区中。

M　列出程序和数据区的起始和结束地址，所有已定义的全局符和它们的值以及所有未定义全局符，后边跟一星号。仅当执行完/D后，才打印程序信息。否则，程序被存放在数据区中。

S　查找命令串中紧接在/S前面的文件名，以满足任何未定义的全局符。

4.2.2 CP/M LINK-80开关

下面的开关仅适用于CP/M版本。

X　若指定了文件名/N，/X将使该文件以Intel ASCII十六进制格式并带一个HEX扩展名保存起来。例如：FOO/N/X/E将建立一个Intel ASCII HEX格式化的装入模块，名为FOO.HEX。

Y　若指定了文件名/N，则当打入/E时，/Y将建立一个名为.SYM的文件。该文件包含用于数字研究公司的符号指令调试程序SID和ZSID的全部全局符的名和地址。例如：FOO/N/Y/E建立FOO.COM和FOO.SYM。MYPROG/N/X/Y/E建立MYPROG.HEX和MYPROG.SYM。

4.2.3 连接举例

连接并运行

```
A>L80
*EXAMPL, EXMPL1/G
DATA 3000 30AC
〔304F 30AC 49〕
〔开始执行〕
```

```
  1792      14336
 14336     -16383
-16383         14
    14        112
   112        896
A>
```

连接并保存

```
A>L80
*EXAMPL, EXMPL1, EXAM/N/E
DATA 3000 30AC
〔304F 30AC 49〕
A>
```

装入并连接EXAMPL.REL, EXMPL1.REL并建立EXAM.COM。

4.3 LINK兼容目标文件的格式

注：本节作为希望了解LINK-80浮动目标模块装入格式的用户的参考材料。不包括程序包的操作说明。

LINK兼容目标文件由一个比特流组成。在比特流中各字段不对齐字节界，下面所注者除外。对于浮动目标文件，使用比特流可使目标文件的长度为最小，从而可减少磁盘读/写的次数。

有两类基本装入项：绝对的和浮动的，由每项的第一位指示。若第一位为0，则其后被装入的八位作为绝对字节；若第一位为1，则其后的两位用于指示下面四类浮动项中的一种：

00 特殊的连接项（见后面）

01 程序相对。在加当前程序基地址后，装入后面的十六位

10 数据相对。在加当前数据基地址后，装入后面的十六位

11 公用区相对。在加当前公用区基地址后，装入后面的十六位

特殊连接项包含比特流100，后跟：

一个四位的控制字段

一个任选的A字段包含两位地址类型，它与前面的两位字段相同，不同的是，00指示绝对地址

一个任选的B字段，包含三位用于给出符号长度，后跟符号名字符（可多达八位）。

特殊连接项的一般表示为：

1 00 xxxx yy nn（A字段） zzz+符号名字符（B字段）

xxxx 四位控制字段（0～15以下）

yy 两位地址型字段

nn 十六位值

zzz 三位符号长度字段

下面的特殊类型只有一个B字段：

0 入口符号（用于查找的名）

1　选择公用块

2　程序名

3　要求库查找

4　扩展连接项（见后面）

下面的特殊连接项只有一个 A 字段和一个B字段：

5　定义公用区长度

6　链接外部符（A是地址链的头，B是外部符的名）

7　定义入口点（A是地址，B是名）

下面的特殊连接项只有一个 A 字段：

8　外部符－偏移。用于 JMP 和外部符调用

9　外部符＋偏移。A 值加到执行前开始于当前指令计数器的二个字节上

10　定义数据区的长度（A 是长度）

11　置装入指令计数器为A

12　链地址。A 是链头，用当前指令计数器代换链中所有登记项。链中最后一项有一绝对零地址字段

13　定义程序长度（A 是长度）

14　程序结束（强制为字节界）

下面的特殊连接项无A字段或 B 字段：

15　文件结束

扩展连接项遵循只有 B 字段的特殊连接项的一般格式，但 B 字段内容不是符号名。而是一个符号域，包含一个指定连接项扩展名的字符，后面跟着 1 到 7 个附加信息字符。

这样，每个扩展连接项格式均为

1　00　0100 zzz i jjjjjjj

其中

zzz　可以是三位的任意整数（000代表 8）

i　是八位的扩展连接项类型标识符

jjjjjjj　是zzz－1 八位信息字符，其意义决定于i

目前，仅有一个扩展连接项：

i　= X'35'COBOL复盖段标记

zzz　= 010_2

j　= COBOL段号－49_{10}。

当连接程序遇到覆盖段标记时，当前覆盖段号被置值为 j＋49。若前面存在的段号非零且/N开关有效，则数据区被写到磁盘上，其文件名为当前程序名且扩展名为Vnn，其中 nn 是二个十六进制数字，表示j＋49_{10}。

4.4 LINK-80错误信息

LINK-80有如下错误信息：

? No Start Address

已发出一个/G开关，但主程序尚未装入。

? Loading Error

输入的最后一个文件不是相应格式化的LINK-80目标文件。

? Out of Memory

无足够存储区用于装入程序。

? Command Error

不可识别的LINK-80命令。

? <file> Not Found

命令串中给出的<文件>不存在。

%2nd Command Larger /XXXXXX/

公用块的第一次定义/XXXXXX/不是最大的定义。重新规定模块装入顺序或改变公用块定义。

%Mult. Def. Global YYYYYY

在装入过程中，遇到对全局符(内部符)YYYYYY的多次定义。

% Overlaying Program Area ,Start = xxxx
,Public = <symbol namc> (xxxx)
,External = <symbol name> (xxxx)

或% Overlaying Data Area ,Start = xxxx
,Public = <symbol name> (xxxx)
,External = <symbol name> (xxxx)

一个/D或/P将导致已装入的数据被破坏。

? Intersecting Program Area 或 Intersecting Data Area

程序和数据区交叉并且一个地址或外部链入口是在此交叉部分。因此其最后值不能转换成当前值。

? Start Symbol - <name> - Undefined

在给定/E：或/G：之后，指定的符号未定义。

Origin Above Loader Memory, Move Anyway (Y or N) ?

或 Origin BelowLoader Memory, Move Anyway (Y or N) ?

在给出 /E或/G 之后，数据或程序区有一初始值或最高值位于装入程序存储区之外(即，装入程序开始地址到最后地址之间)。若回答Y<CR>，LINK-80 将传送该区并继续执行。若回答Y以外的其他字符，LINK-80将退出。不管哪种情况，若给出/N，则该映象均已被保存。

? Nothing Loaded

给出了<文件名>/S，或/E 或/G，但没有装入目标文件。即，在尚未装入任何程序的情况下，企图查找一个库，退出连接程序，或执行一个程序，例如：

TEST/N/E

产生 '? Nothing Loaded' 信息，因为TEST/N 命名 TEST.COM，而不装入 TEST. REL。

? Can't Save Object File

正在保存文件时，发生磁盘错误。

4.5 程序中断信息

如果被装入程序已定义名为$MEMRY的全局符号，则LINK-80将第一个自由存储区的地址存于该符号中。$MEMRY置为该数据区最高地址加1。

注：若给出/D，且数据区起始地址小于程序区起始地址，则用户须确保有足够空间去保存该程序以免遭破坏。对于使用$MEMRY分配磁盘缓冲区和FCB的FORTRAN-80磁盘驱动程序尤其如此。

第五节 LIB-80库管理程序（仅对CP/M版本）

LIB-80是在目标执行时，CP/M 版下FORTRAN-80、COBOL-80和BASIC编译程序的库管理程序。将来的LIB-80版本将与其他操作系统相连接。

特别要注意：请详细阅读此说明，并在使用LIB之前，先复制一个库文件的备用副本。因为若使用不当，很容易破坏库文件。

5.1 LIB-80命令

要运行LIB-80，可打入LIB后跟一个回车。LIB-80将回答提示符“*”，表示准备接收命令。LIB-80中每个命令或列出有关库的信息，或将新模块加到库中。

发至LIB-80的命令包括一个任选的目的文件名用作被建立的库名，后面跟一等号，再后面是模块名。模块名之间用逗号分开。缺省的目标文件名为FORLIB.REL。例如：

```
*NEWLIB=FILE1 <MOD2>,FILE3, TEST
*SIN, COS, TAN, ATAN
```

任何指定一组模块的命令将所选择的模块连接到给定的最后一目的文件的末尾。因此，

```
*FILE1, FILE2<BIGSUB>,TEST
```

等价于

```
*FILE1
*FILE2 <BIGSUB>
*TEST
```

5.1.1 模块

模块通常是一个标准的FORTRAN或COBOL子程序、主程序或包含ENTRY语句的MACRO-80汇编程序。

LIB-80的主要功能是连接.REL文件中的模块以形成一个新库。为从前面的库或.REL文件中提取出模块，已设计了一个强有力的语法检查程序，以指定一个.REL文件中模块的范围。

要指定一个文件中的模块，最简便的方法是利用模块名。例如：

```
SIN
```

但是也可以使用相对数量加或减255。例如：

```
SIN+1
```

指定SIN之后的模块，同样

```
SIN-1
```

指定SIN的前一个模块。

还可通过使用两个点来指定模块的范围：

..SIN　意为直到 SIN 为止的所有模块（包括SIN）。

SIN..　意为从 SIN到该文件结束的全部模块。

SIN..COS　意为 SIN 和COS 以及二者之间的全部模块。

模块和相对偏移的范围也可组合使用：

SIN+1..COS-1

为从一文件中选择给定的模块，可使用该文件名，后面跟指定的模块名，模块名用尖括号括起来，之间用逗号隔开：

FORLIB〈SIN..COS〉或

MYLIB.REL〈TEST〉或

BIGLIB.REL〈FIRST, MIDDLE, LAST〉等等。若没有指定从一个文件中选择模块，则隐含选择该文件中的所有模块：

TESTLIB.REL

5.2 LIB-80 开关

LIB-80操作可通过一些开关来控制。这些开关前都带有一斜线：

/O　八进制——为/L命令设置八进制输出方式。

/H　十六进制——为/L命令设置十六进制输出方式（缺省）。

/U　列出在对指定文件进行查找时未定义的符号。

/L　列出指定文件中的模块和它们包含的符号定义。

/C　（建立）抛弃结构中的库，重新开始。

/E　退出 CP/M。将结构中的库（.LIB）改为.REL，且删除以前的任何副本。

注：若没有在结构中的新库，则/E 将破坏当前的库。若不修改库，可用 Control-C 退出 LIB-80。

/R　重命名——同/E，但在结束时不退出至CP/M。

5.3 LIB-80列表文件

为以相互对照的格式列出文件的内容，可用/L：

*FORLIB/L

在建库时，非常重要的一点是安排好模块的顺序，使模块之间的引用都是“正向”的。即，包括全局引用的模块在物理上应出现在包括入口点的模块之前，否则，LINK-80 在对该库进行单次扫描中可能满足不了所有的全局引用。

使用 /U 列出那些在对一个库进行单次扫描中未定义的符号。若库中一模块对另一模块中的符号进行反向引用，则 /U 将列出该符号。例如：

*SYSLIB/U

注：由于在标准 FORTRAN 和 COBOL 系统中的某些模块总是强行装入的，所以它们将被/U 作为未定义的符号列出，但在装入 FORTRAN 或 COBOL 程序时不会引起错误。

清单一般总是送到终端，也可使用Control-P 送到打印机输出。

5.4 LIB 使用举例

建一个库：

A>LIB

*TRANLIB=SIN,COS,TAN,ATAN,ACOG

```
*EXP
*/E
A>
列库清单:
A>LIB
*TRANLIB.LIB/U
*TRANLIB.LIB/L
    ⋮
(TRANLIB.LIB中符号表)
    ⋮
*Control-C
A>
```

5.5 开关与语法概要

/O 八进制——设置清单基数

/H 十六进制——设置清单基数

/U 列出未定义符号

/L 列出相互对照表

/C 建立——重新开始 LIB

/E 退出——将.LIB改名为.REL并退出

/R 改名——将.LIB改名为.REL

模块::=模块名{+或-数}

模块序列::=模块|..模块|模块..|模块1..模块2

文件说明::=文件名{<模块序列>{,<模块序列>}}

命令::={库文件名=}{文件说明表}{开关表}

附录A TEKDOS操作系统

在TEKDOS操作系统下，用于MACRO-80、LINK-80和CREF-80的命令格式稍有不同。

A.1 TEKDOS命令文件

文件F80、M80、L80和C80实际上是分别对于编译程序、汇编程序、装入程序和相互对照程序的命令文件。这些命令设置仿真方式为0，并选择Z80汇编处理程序（见TEKDOS说明），然后执行适当的程序文件。这些命令文件都以驱动器*1建立以执行Microsoft程序。至3.36版本为止，LINK-80还是在驱动器*1上查找库（FOR LIB）。若希望执行驱动器*0上的软件，则必须编辑命令文件且LINK-80应给出显式的库查找命令“FORLIB-S”（见4.2.1节）。

A.2 MACRO-80

M80汇编程序仅接受命令行，而不显示提示符，并且不接受交互命令。其命令格式与TEKDOS汇编程序命令相同。即，三个文件名或设备名参数加上任选的开关：

M80〔目标文件〕〔列表文件〕源文件〔开关1〕〔开关2…〕

目标和列表文件参数是任选的。如果省略这些参数，则不建立这些文件，但全部错误信息仍将显示在控制台上。可利用的开关如本章第二节所述。不同的是，开关之间用逗号或空格分开，而不是用斜线。

A.3 CREF-80

对CREF-80的命令格式为：

C80 列表文件 源文件

两个文件名参数都是必须的。源文件参数总是在汇编时用开关C建立的CREF80文件的名。例如：

用MACRO-80建立一个CREF80文件：

M80,,TSTCRF TSTMAC C

建立一个CREF80文件的相互对照列表：

C80 TSTLST TSTCRF

A.4 LINK-80

对于TEKDOS，LINK-80装入程序仅接受交互命令，不支持命令行。

在调用LINK-80并等待输入时，将显示一星号提示符。命令是一列文件名或设备名，其间用逗号或空格分开，并可任选地插入开关参数。LINK-80的输入必须是Microsoft浮动目标码（不同于TEKDOS装入程序的格式）。

LIK-80的开关在TEKDOS下是由连字符作定界符，而不是用斜线。所有LINK-80开关（象第四节所述）都可使用，但“G”和“N”目前还未实现。例如：

1．汇编一个名为XTEST的MACRO-80程序，建立一个名为XREL的目标文件和一个名为XLSL的列表文件：

```
>M80 XREL XLST XTEST
```

2．装入XTEST并保存已装入的模块：

```
> L80
* XREL-E
〔04AD 22B8〕
* DOS * ERROR 46
L80 TERMINATED
>M XMOD 400 22B8 04AD
```

注意，由于执行HLT指令“-E”经过一个错误信息退出。内存映象不受影响。当然，也可用“Module”命令去保存它。当一程序以模块格式被保存时，它不可不通过LINK-80而再次直接执行。

在退出之前，由LINK-80打印的括号内的数字分别是入口点地址和最高装入地址。装入程序缺省是在400H处开始装入。但装入程序也可以放一个转移至0单元起始地址的指令，因此，允许从0单元开始执行。在0003和0400H单元之间的内存空间保留作为运行时SRB和I/O缓冲区。

本手册编译校对：王建科、刘运基、郑远新、赵槟槟、阚庆利

FORTRAN-80 参考手册

第一章 绪 论

第一节 综 述

FORTRAN 语言是一种通用的、面向问题的程序设计语言。它可以简化计算机程序的预处理与程序检查工作。该语言的名字 FORTRAN 是公式翻译 (FORmula TRANslator) 的缩写。

使用这种语言的语法规则是严格的，并且要求程序员用一系列正确的语句去定义某个问题的整个特征。FORTRAN 编译程序把那些称为源程序的语句翻译成一个可重定位的模块，而后该模块又由称为链接装配程序的程序将其翻译成为计算机的机器语言，于是程序在机器上可以执行。

本手册为 Heath H8及 Heath/Zenith H/Z89计算机定义了 Microsoft FORTRAN-80 源语言。

本语言包含了于 1966 年 3 月 7 日通过的 ANSI X3.9—1966 资料中所描述的美国标准 FORTRAN 语言的大多数内容，并对该语言增加了某些扩充与限制。

整个手册所包含的例子用来说明该语言的结构与用途。为了掌握语言功能的特点，程序员应该熟悉该语言的整个内容。

下面列出该语言对 ANSI 标准 FORTRAN (X3.9—1966) 语言所做的扩展：

1. 当在 "STOP C" 或 "PAUSE C"语句中使用 C 时，C 不可以超过 6 个 ASCII 字符。

2. "Error" 与 "End of File" 逻辑分支可在 READ (读) 与 WRITE (写) 语句中用 ERR = 与 END = 选择来指出。

3. 标准子程序 PEEK、POKE、INP 与 OUT 已增加到 FORTRAN 库文件中。

4. 语句函数可把下标变量用作参数。

5. 可以使用十六进制常量，在正常情况下使用整型常量。

6. 允许在标准 nH 格式中使用文字型数据 (单引号之间的字符串) 的文字格式。

7. 对继续行的行数没有限制。

8. 允许混合方式的表达式与赋值，其转换是自动完成的。

9. 逻辑变量可作为在 - 127～ + 127之间的整型量。

10. 在整型数据上可执行逻辑运算 (.AND.、.OR.、.NOT.、XOR.可用作 8 位或16 位逻辑运算)。

11. ENCODE/DECODE 编码与译码可用来编辑或转换数据。

12. 整个语言功能是由随机存取文件提供的。

FORTRAN 程序员必须注意到上述扩充的特征并充分发挥其优点。

FORTRAN-80根据 ANSI 标准 FORTRAN 作下述限制：

1. 不能实现 COMPLEX (复数) 数据类型。

2. 程序体中的语句必须以下述次序出现：

（1）PROGRAM（程序），SUBROUTINE（子程序），FUNCTION（函数），BLOCK DATA（数据块）

（2）TYPE（类型），EXTERNAL（外部），DIMENSION（维数）

（3）COMMON（公用区）

（4）EQUIVALENCE（等价）

（5）DATA（数据）

（6）语句函数

（7）可执行语句。

3. 对于每种数据类型（整型、实型、双精度型与逻辑型），计算机的存储分配是各不相同的。

4. 赋值语句的等号（=）及循环（DO）语句的第一个逗号（，）必须出现在最初的语句行中。

5. 非格式顺序 I/O 语句总是要提供一个变量表。

FORTRAN 程序员必须注意到上述几个限制，并在编写 FORTRAN 源程序时遵守这些约定。

第二节 FORTRAN程序格式

FORTRAN 源程序由一个称作主程序的程序体及任意多个称作子程序的程序体所组成。主程序及程序体由一系列语句组成，它们正确地描述了解题所需要的过程，并且定义了 FORTRAN 编译程序在执行目的程序期间所要用到的信息。每个语句必须用 FORTRAN 语言字符集来编写，并且必须遵循所描述的行格式。

2.1 FORTRAN 字符集

为了简化引用及说明，FORTRAN 字符集划分为四个子集，每个子集均给予一个名字。

2.1.1 字母

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, $

（$ 可以看作是一个字母）

大写与小写之间没有差别，但为了清晰与可读，建议只使用大写。

2.1.2 数字

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

表示数字常量的数字串在正常情况下按十进制数字解释。但在某些语句中，可按十六进制数字系统解释。这时，A、B、C、D、E、F 也就用作十六进制数字。

2.1.3 字母数字符

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, $

0, 1, 2, 3, 4, 5, 6, 7, 8, 9

该子集部分包含了整个字母与数字。

2.1.4 专用字符

专用字符有：

= 等号

+ 加号

- 减号

(左括号

) 右括号

. 十进制小数点

下述专用字符按算术运算符分类，在算术表达式中有明显的意义：

+ 加法或正值

- 减法或负值

* 乘法

** 取幂

/ 除法

其他专用字符在 FORTRAN 语言的语法表达式及 FORTRAN 语句结构中有专门的 应用。

任何可打印的字符可以出现在文字型或字符字段中。

2.2. FORTRAN 行格式

FORTRAN 源程序的行由80个字符位置或列组成，序号从 1 到80，可以划分为四个字段：

1. 语句标号（或数）字段——第 1 ～ 5 列（参阅语句标号的定义
2. 继续字符字段——第 6 列
3. 语句字段——第 7 ～72列
4. 标识符字段——第73～79列

标识符字段适用于 FORTRAN程序员所需要的任何目的，而FORTRAN程序忽略处理。

注意：最后一列（第80列）决不包含任何其他字符，该列保留作回车换行字符。

2.2.1 行类型

根据行类型的要求，一个 FORTRAN 语句的行应放在其格式的第 1 ～72列。有四种行类型，它们的定义及其列的格式是：

注解行——为便于程序员对源程序作注释而采用，列格式为：

1. 第 1 列含有字符 C。
2. 第 2 ～72列可用任何形式去描述注释，或者为空。
3. 一个注解行后只可以后继初始行、结束行或另一个注解行。
4. 除了在源程序列表中显示外，注解行对目的程序毫无影响，编译程序不处理它。

例如：

```
C       COMMENT LINES ARE INDICATED BY THE
C       CHARACTER C IN COLUMN 1
C       THESE ARE COMMENT LINES
```

结束行——程序体的最后一行。其列格式为：

1．第 1～5 列可含有一个语句标号。

2．第 6 列必须是零或空格。

3．第 7～72列之间必须按字母 E、N、D 的次序各出现一次，它们之间可由空格字符隔开。

4．每个 FORTRAN 程序体必须有一个作为程序体最后一行的结束行，它将告知 FORTRAN 编译程序，这是程序体物理的结束位置。

5．结束行后面可以跟有其他任何类型的行。

例如：

```
END
```

初始行——每个语句的第一行。其列格式：

1．第 1～5 列可以含有标识该语句的语句标号。

2．第 6 列必须是零或空格。

3．第 7～72列可以是该语句的全部或部分。

4．初始行可在该语句段中的任何地方开始写。

例如：

```
C       THE STATEMENT BELOW CONSISTS
C       OF AN INITIAL LINE
        A = .5*SQRT(3 - 2.*C)
```

继续行——程序员为了完成由初始行开始的语句而需要增加一些程序行时，可以使用继续行。其列格式为：

1．不考虑第 1～5 列，而第 1 列不允许是字符 C。

2．如果第 1 列含有字符 C，则其为注解行。

3．第 6 列必须是零或空格字符以外的其他字符。

4．第 7～72列是该语句的继续部分。

5．如果需要，该语句可以有很多个继续行。

例如：

```
C       THE STATEMENTS BELOW ARE AN INITIAL LINE
C       AND 2 CONTINUATION LINES
              BETA(1, 2) =
     1  A6BAR**7 - CBETA(2, 2) - A5BAR*50
     2   + SQRT(BETA(2, 1))
```

2.2.2 语句标号

语句标号可以放在 FORTRAN 语句初始行的第 1～5 列，其他语句中可引用它们。关于语句标号的规则是：

1．标号可以是1～99999之间的任何整数。

2．在计算标号数值时，前面的零与空格无意义。

3．同一程序体中的标号应各不相同。

4．FORTRAN 处理程序并不处理在继续行中的标号。例如：

```
200  R = SQRT(A/PI)
```

第三节 语　句

单个语句与程序体中描写的具体过程有关，它们被分为可执行语句和非执行语句两种类型。

3.1 可执行语句

可执行语句指明动作并可使 FORTRAN 编译程序产生其目的程序指令。有下述三种类型的可执行语句：

1. 赋值语句
2. 控制语句
3. 输入与输出语句

3.2 非执行语句

非执行语句向编译程序描述数据的特性及排列，提供关于输入与输出的信息以及在程序装入与执行期间把初始数据提供给目的程序。有下述五种类型的非执行语句：

1. 说明语句
2. 数据初值语句
3. 格式语句
4. 函数定义语句
5. 子程序语句

第二章　FORTRAN源程序的编译

第一节　综　述

当已经建立好 FORTRAN 源程序之后，就对它进行编译。为了通知 FORTRAN编译程序要编译什么和有什么选择，需要输入一个“命令串”，它是由 FORTRAN-80命令扫描程序来读入处理的。

为了编译源程序，该命令串中必须包含 FORTRAN 编译程序所需要的信息。

当源程序已被编译好并无错误时，在它被执行之前，必须对它进行链接装配。这一操作在本参考手册“LINK-80”这一节来描述。

第二节　命令格式

为了运行 FORTRAN-80，可以打入命令“F80”，其后接一个回车键 RETURN。FORTRAN-80将以提示信息“*”作回答，表示已准备好接收命令。此时，打入命令串，其后接一个回车键。FORTRAN 命令的一般格式是：

目的程序设备：文件名·扩展名，清单设备：文件名·扩展名=源程序设备：文件名·扩展名

其中：

目的程序设备：即是目的程序将要写入的设备。如果省略其设备名，则缺省值表示其为当前盘。

清单设备：即是程序清单所需要的设备。它可以是终端、硬拷贝设备或某个磁盘文件。对于磁盘文件的缺省设备是当前盘。

源程序设备：即是将源程序输入到 FORTRAN-80的设备。如果省略其设备名，缺省表示当前盘。

文件名·扩展名：这些是目的文件、清单文件、源文件的文件名与扩展名。文件的扩展名可以省略。

缺省的文件扩展名是：

源文件　·FOR

目的文件　·REL

清单文件　·PRN

目的文件、清单文件或两者同时都可以省略。

如果既不需要清单文件，也不需要目的文件，则只需在等号（=）左边放上一个逗号，这对于检查刚刚建立好的源程序的语法是有益的。

如果省略目的文件名与列表文件名，则就将源文件及其缺省的扩展名作为其补缺的文件名与扩展名。

例如：

*=TEST　　编译当前盘上名为TEST·FOR的源程序，其目的文件名将是 TEST·REL。

*B:TEST=B:TEST　　编译 B 盘上名为 TEST·FOR 的源程序，其目的文件在B盘上为 TEST·REL，清单文件在 B 盘上为 TEST·PRN。

*,=TEST·FOR　　编译当前盘上名为 TEST·FOR的源程序，但不产生目的文件与清单文件。这在检查语法错误时是有益的。

*B:SAMPLE,LST:=SAMPLE　　编译当前盘上名为 SAMPLE·FOR的源程序，将清单文件输出到清单设备 LST:上，并将目的文件放在 B 盘的 SAMPLE·REL 文件中。

当该程序编译好以后，系统再次在终端设备上显示提示符"*"。为了使控制返回到CP/M系统，可打入 CTRL-C 命令。

也可以把编译命令组合到命令串中。如这样做，可以打入"F80 〈命令串〉"（该空格是必需的）。使用这种方法时，在该程序被编译通过以后，编译程序则自动返回到 CP/M 系统状态。

例如：

F80 TEST=TEST　　调用编译程序，编译当前盘上名为 TEST·FOR 的源程序，并将目的文件写到 TEST·REL 文件中，把清单文件写到 TEST·PRN 文件中（注：此句原稿遗忘，但按上述第二例应有——译者）。

编译完成后就返回到 CP/M 系统状态。使用这种组合的方法，编译程序不再显示提示符"*"。

第三节　FORTRAN 编译开关

在命令串中可以给出若干个不同的编译开关，这将会影响编译的处理。

每个编译开关之前必须先加上斜杠（/）：

开关	功能
O	按八进制数打印所有的清单地址
H	按十六进制数（可补缺）打印所有的清单地址
N	不必列出编译程序所生成的操作码
A	列出编译程序生成的操作码（缺省值）
R	强制建立目的文件
L	强制建立清单文件

P　每按一次/P，就额外分配给编译期间使用的 100 字节的堆栈空间。在编译期间如果产生堆栈溢出错误，可使用/P 开关，否则就不必按/P 开关

M　指定编译程序所生成的代码应处于可以装入到 ROM 中的形式。当指定/M 时，所生成的代码将与以下常规方法不同：

a. 将 FORMAT 放在程序区，用"JMP"绕过它们

b. 在运行时就初始化参数块（对于三个参数以上的子程序调用），而不是由装入程序初始化。

例如：

*=TEST/L　　编译当前盘上名为 TEST·FOR 的文件，将清单写到TEST.PRN文件中，并生成 TEST.REL 目的文件。

*=BIGGONE/P/P　　编译当前盘上 BIGGONE.FOR 源文件并生成 BIGGONE.REL 目的文件。编译程序分配200个字节的堆栈空间。

*PROG, PROG=PROG/O　编译当前盘上名为 PROG.FOR 源文件，将清单写到 PROG.PRN 文件中并生成 PROG.REL 目的文件。清单文件中的地址是八进制的。

*=SAMPLE/L/A　　编译当前盘上 SAMPLE.FOR 源文件，将清单写到 SAMPLE.PRN 文件中并生成 SAMPLE.REL 目的文件。在清单文件中打印编译程序所生成的操作码。

*FIRM,FIRM=FIRM/M　　编译 FIRM.FOR 源文件，将清单写到 FIRM.PRN 文件中并生成 FIRM.REL 目的文件。生成适合于 ROM 的代码。

如果 FORTRAN 程序想要用 ROM，则程序员要注意下述细节：

1. 将不再使用 DATA（数据）语句去初始化 RAM。其初始化由装入程序完成，因此执行时就不再初始化。在执行期间，变量与数组可以由赋值语句或读入语句去初始化。

2. 在执行期间不再读入FORMAT 语句。

3. 不能打开逻辑设备号为非 6，7，8，9，10的磁盘文件。

编译示例

```
B:SAMPLE,LST:=B:SAMPLE/N

FORTRAN-80 Ver. 3.4 Copyright 1978,79,80(C) By Microsoft-Bytes:9320
created: 26-NOV-80

1       C   SAMPLE PROGRAM AVERAGE
2       C   WILL   COMPUTE AVERAGE OF THREE NUMBERS
3                  PROGRAM SAMPLE
4                  REAL NUMBER(3)
5                  DATA NUMBER/100.,200.,300./
6                  DATA SUM,AVER/0.,0./
7       C   LOOP   TO CALCULATE SUMMATION
8                  DO 600 I=1,3
9                  SUM=SUM+NUMBER(I)
10      600        CONTINUE
11                 AVER=SUM/3
12                 WRITE(1,700) AVER
13      700        FORMAT(' ',F6.2)
14                 STOP SAMPLE
15                 END
```

```
PROGRAM UNIT LENGTH = 0067 (103) BYTES
DATA AREA LENGTH = 0021 (33) BYTES

SUBROUTINES REFERENCED:

$I1            $INIT          $L1
$AB            $T1            $DA
$W2            $ND            $ST

VARIABLES:

NUMBER  0001″          SUM  000D″          AVER  0011″

I  0015″

LABELS:

$$L     0006′          600L  0024′          700L  0017″
```

注：单引号（′）表示某个程序的相对值而双引号（″）表示某个数据的相对值。

第四节　FORTRAN 编译程序错误信息

FORTRAN 编译程序可检查两类错误：警告性错误与严重错误。

当出现警告性错误时，编译程序将继续处理该行的后继项，而当找到严重错误时编译就不再处理该逻辑行，包括其任何继续行的余留部分。

警告性错误信息之前有百分比符号（%）而严重错误信息之前有问号（？）。

接着，由编译程序打印出一个行号，该行号是编译程序发现其错误地方的行号。需着重说明的是，该行号可能与该错误的实际的物理行号不相符。该行号后面跟有错误码或错误信息。在检查到错误之前所扫描到的20个字符也会被打印出来。

例如：

```
? Line25: Mismatched Parenthesis     括号不匹配
%Line16: Missing Integer Variable    缺少整型变量
```

当出现警告性或严重错误时，就要修改程序，以使编译无错。否则不能保证编译有错误的程序会合理地执行。

后面几页给出了编译程序可以检测到的各种不同的错误清单，同时也指出了运行时的错误信息。

4.1　严重错误

错误号	信　　息
100	Illegal Statement Number 非法语句号
101	Statement Unrecognigable or Misspelled 无法识别或错拼的语句
102	Illegal Statement Completion 非法语句结束
103	Illegal Do Nesting 非法DO嵌套
104	Illegal Data Constant 非法数据常量
105	Missing Name 缺少名字
106	Illegal Procedure Name 非法过程名
107	Invalid DATA Constant or Repeat Factor 无效的数据常量或重复因子
108	Incorrect Number of DATA Constants 错误的数据常量值
109	Incorrect Integer Constant 错误的整型常量
110	Invalid Statement Number 无效的语句号
111	Not a Variable Name 非变量名
112	Illegal Logical Form Operator 非法逻辑操作
113	Data Pool Overflow 数据池溢出
114	Literal String Too Large 文字串太大
115	Invalid Data List Element in I/O　I/O操作中的非法数据列表元素
116	Unbalanced DO Nest　DO嵌套不对称
117	Identifier Too Long 标识符太长
118	Illegal Operator 非法操作符
119	Mismatched Parenthesis 括号不匹配
120	Consecutive Operators 操作符相连
121	Improper Subscript Syntax 不合适的下标语法
122	Illegal Integer Quantity 非法整型量
123	Illegal Hollerith Construction 非法字符结构
124	Backwards DO reference 逆DO引用
125	Illegal Statement Function Name 非法语句函数名
126	Illegal Character for Syntax 非法语法字符
127	Statement Out of Sequence 语句次序不对
128	Missing Integer Quantity 缺少整型量
129	Invalid Logical Operator 无效的逻辑操作符
130	Illegal Item in Type Declaration 类型宣称中非法项
131	Premature End of File on Input Device 输入设备中过早的文件结束
132	Illegal Mixed Mode Operation 非法混合方式操作
133	Function Call with No Parameters 函数调用不带参数
134	Stack Overflow 堆栈溢出

135	Illegal Statement Following Logical IF 逻辑 IF 后非法语句

4.2 警告性错误

错误号	信　　息
0	Duplicate Statement Label 语句标号重复
1	Illegal DO Termination 非法 DO 结束
2	Block Name = Procedure Name 块名 = 过程名
3	Array Name Misuse 数组名误用
4	COMMON Name Usage 使用公共名
5	Wrong Number of Subscripts 错误的下标号
6	Array Multiply EQUIVALENCED Within a Group 一组中多等价矩阵
7	Multiple EQUIVALENCE of COMMON 多个数据公用块等价
8	COMMON Base Lowered 公用块基址之下
9	Non-COMMON Variable in BLOCK DATA 块数据中非公共变量
10	Empty List for Unformatled WRITE 非格式 WRITE (写) 语句中空列表
11	Non-Integer Expression 非整型表达式
12	Operand Mode Not Compatible With Operator 与操作符矛盾的操作数方式
13	Mixing of Operand Modes Not Allowed 不允许混合的操作数方式
14	Missing Integer Variable 缺少整型变量
15	Missing Statement Number on FORMAT FORMAT 语句中缺少语句标号
16	Zero Repeat Factor 重复因子零
18	Format Nest Too Deep 格式嵌套太深
19	Statement Number Not FORMAT Associated 与FORMAT不相连的语句号
20	Invalid Statement Number Usage 使用无效的语句号
21	No Path to this Statement 该语句无引用路径
22	Missing DO Termination 缺少 DO 语句结束
23	Code Output in BLOCK DATA 块数据中代码输出
24	Undefined Labels Have Occurred 使用没定义的标号
25	RETURN in a Main Program 主程序中出现返回语句
27	Invalid Operand Usage 使用无效的操作数
28	Function with No Parameter 函数没带参数
29	Hex Constant Overflow 十六进制数溢出
30	Division by Zero 除数为零

32	Array Name Expected 希望有数组名
33	Illegal Argument to ENCODE/DECODE ENCODE 与DECODE中的非法自变量

第五节 FORTRAN运行时的错误信息

当运行过程中出现错误时，运行时的错误信息就会显示在终端上，前后各有两个星号围着它。例如：

FW

严重错误会中止执行（使控制返回到操作系统）。在未超过20个警告性错误之前，是不理睬该警告性错误的（虽然数据是错误的），否则就中止执行。

警告性错误：

错误码	含 义
FW	Field Width Too Small 字段宽度太小
EX	Illegal Exponentiation 非法取幂
IB	Input Buffer Limit Exceeded 超出输入缓冲区限制
TL	Too Many Left Parentheses on FORMAT FORMAT 中左括号太多
OB	Output Buffer Limit Exceeded 超出输出缓冲区限制
DE	Decimal Exponent Overflow(exponent>99) 十进制指数溢出（指数>99）
IS	Integer Size Too Large 整型域太大
BE	Binary Exponent Ovevflow 二进制指数溢出
IN	Input Record Too Long 输入记录太长
OV	Arithmetic Overflow 算术溢出
CN	Conversion Overflow (REAL to INTEGER Conversion) 转换溢出（实型到整型转换）
GL	Computed GOTO Number Too Large 计算 GOTO 语句号太大
GS	Computed GOTO Number Too Small 计算 GOTO 语句号太小
SN	Argument to SIN Too Large SIN 函数的自变量太大
A2	Both Arguments of ATAN2 Function Are 0 ATAN2 反三角函数的两个自变量都是零
BI	Buffer Size Exceeded During Binary I/O 二进制 I/O 操作时超出缓冲区容量
RC	Negative Repeat Count in FORMAT FORMAT 中负的重复计数

严重错误：

错误码	含 义
ID	Illegal FORMAT Descriptor 非法 FORMAT 描述

Fϕ	FORMAT Field Width is Zero　FORMAT中字段宽度为零
MP	Missing Period in Format 格式中缺少句点
IR	Attempting Real in Integer Field 整型字段中企图实型
IT	I/O Transmission Error　I/O操作传递错
DO	Illegal Increment or Limit in DO Loop　DO 循环中非法增量或限制
ML	Missing Left Parenthesis in FORMAT　FORMAT 中缺少左括号
DZ	Division by Zero,REAL or INTEGER 实型或整型中除数为零
LG	Illegal Argument to LOG Function(Negative or Zero)　LOG对数函数中非法自变量（负数或零）
SQ	Illegal Argument to SQRT Function (Negative)　SQRT开平方函数中非法自变量（负数）
IO	Illegal I/O Operation 非法 I/O 操作
DT	Data Type Does Not Agree With Format Specification 与指定格式不相符的数据类型
EF	EOF Encountered on READ　READ（读）语句中遇到 EOF（文件结束）
FN	File Not Found on OPEN　OPEN（打开）语句中文件没找到
DF	Disk Full Encountered on WRITE　WRITE(写）语句中遇到盘满
UN	Logical Unit Number (LUN) Too Large　逻辑设备号太大
OM	Out of Memory 内存溢出

第三章　数据表示法与存储格式

第一节　综　　述

FORTRAN 语言规定，其数据类型分成几类。其分类是：整型、扩展整型、实型、双精度型、逻辑型和文字型。

在各种不同的数据类型中，按照其成分的功能又存在另一种分类。这种按成分功能的分类是：常量、变量、数组与数组元素。

常量表示一个固定值，因此在程序执行期间就不能对其修改。然而，数据变量就要经常被修改。数组是与单个符号名相连的一组存储单元，该符号名叫数组名。数组元素是数组中的一个成分。

第二节　数据类型

FORTRAN 语言可以识别一些特殊的数据类型：整型、实型、双精度型、扩展整型、逻辑型以及文字型。在源程序中可以使用数据类型语句去选择某种数据类型（数据类型语句将在第九章“说明语句”中讨论）。

数据类型可以由下述预先规定好的缺省的习惯来建立。这些习惯是：以字母 I、J、K、L、M和N 开头的整个符号名与整型数据相连，而所有其他的符号名与实型数据相连。

通常由特定程序的需要来决定数据类型。例如，由于整型量执行起来总要比双精度型快得多，因此整型变量总是反复地赋给 DO 循环作为其循环变量的计数器。

2.1　整型

整型是整数（正，负或零）的精确表示形式。它可以精确到 5 位数字，其取值范围为 $-32768 \sim +32767$（即 $-2**15 \sim 2**15-1$）。

整型量按16位二进制位存储，低位 0 ～14表示其二进制数值。当该值为正或负时，就由其高位（第15位）来表示。负整型量按正整型量的二进制补码形式来存储。

规则：

1 ～ 5 个十进制数字按十进制数来解释

例如：－763

1

＋00672

数前的正号（＋）或负号（－）可以任选

例如：－32768

＋32767

不允许数中出现小数点（.）或逗号（,）

2.2 四字节整型

扩展整型是整数（正、负或零）的精确表示形式。它精确到10位数字，其取值范围为－2147483648～＋2147483647（即－2＊＊31～2＊＊31－1）。

扩展整型按32位二进制位存储，低位0～30表示其二进制数值。当该值为正或负时，就由其高位（第31位）来表示。负整型量按正整型量的二进制补码形式来存储。

规则：

1～10个十进制数字按十进制数解释

例如：　－596

1

1314653487

数前的正号（＋）或负号（－）可以任选

例如：－2147483648

＋2147483647

不允许数中出现小数点（.）或逗号（，），但源程序中的空格字符是允许的

例如：　31　150

－2　846　766

2.3 实型

在计算机存储器中实数（正、负或零）是以四个字节，即浮点形式来表示的。实数的存储精确到7位有效数字。其取值范围约为10＊＊－38～10＊＊38（即2＊＊－127～2＊＊127）。

实型数与双精度型数均是以浮点形式来存储的。一个实型数的存储单位是32位长。第0～23位用来表示该数的尾数，而第24～31位留作其首数。

尾数以二进制补码的表示形式来存储。尾数是二进制小数，因此小数点总是假定在小数的最左边。尾数是规格化的，其最高位总是1，因此，不需要去存储它，除非幂是0。仅当幂是0的情况下，其最高位才假定为0。

规则：

带有7位精度的十进制数可用下述形式之一来表示：

a. ＋/－.f　　或　＋/－i.f

b. ＋/－i.E＋　　或　＋/－i.－e

c. ＋/－.fE＋　　或　＋/－.f－e

d. ＋/－i.fE＋　　或　＋/－i.f－e

其中：i、f与E（－e）分别表示为整数、小数及指数。

正号（＋）与负号（－）字符可以任选。

十进制小数点不可以选择。所有实数必须带有一个十进制小数点。指数E（－e）的值按该实数＊10＊＊e来解释。其中指数的取值范围在＋38与－38之间（即：－38＜＝e＜＝＋38）。

如果E＋或－e之前的常量含有比实数所许可的精度更多的有效数字，就会出现截断。因为只能表示在实数范围内许可的最多有效数字。

2.4 双精度型

实型的双精度型数（正、负或零）的近似值在计算机存储器中是以8个字节，即浮点形式来表示的。双精度型实数精确到16位有效数字，其取值范围与实型数相同。

实型与双精度型均是以浮点形式来存储的，然而双精度型的存储单位是64位长。

规则：

具有16位精度的十进制数可以用下述形式之一来表示：

a. +/-.f　　或+/-i.f

b. +/-i.D+　或+/-i.-d

c. +/-.fD+　或+/-.f-d

d. +/-i.fD+　或+/-i.f-d

其中：i、f与D（-d）分别表示整数、小数及指数。注意：一个实型常量，如果其指数部分不以“D”来表示，则就假定其是单精度的。

正号（+）与负号(-)可以任选。十进制小数点不可以选择。所有的实型数都必须带有一个十进制小数点。在上述b形式中（应为b，c，d三种形式——译者），如果r表示为D+或-d前的任何形式（即rD+或-d），则该常量的值按r*10**d来解释，其中：-38<=d<=38。

如果D+或-d前的常量含有比双精度所允许的精度更多的有效数字，就会出现截断。因为只能表示在双精度范围内所许可的最多有效数字。

2.5 逻辑型

逻辑型数据在计算机存储器中仅仅表示为取逻辑真值与假值。单个逻辑值的存储单位是7位。第8位经常用来表示一个带符号的整数。逻辑真值（true）常量赋值为-1。任何一个非零值也按逻辑真值来处理。

逻辑表达式只能取两个值，即真（true）或假（false)。假值的内部表示形式是零。一个非零值总是被表示成并在机内被当作真值来存储。

当FORTRAN语句中出现逻辑表达式时，它总是按照下述所给出的规则来计算。逻辑型也可以用-128～+127间单字节带符号整数。

规则：

任何非零值均赋予逻辑真值.TRUE.。

当该字节中所有的位全置为零时才赋予其假值.FALSE.。

逻辑值可按单字整数来使用。把逻辑值用作整数的规则保持相同。注意：把逻辑值用作整数时，其范围仅从-128～+127。

2.6 文字型

来自于计算机字符集的任意个数的字符均可以有效地输入到文字串中。包括空格字符在内，所有的字符都是有意义的。文字型数据要求串中每个字符的存储形式都是一个字节。十六进制数据也可以（经过DATA语句）与任何类型的数据相连。其存储分配和与其相连的数据相同。

文字型或字符型数据可以使用DATA初始化语句（参阅第九章“说明语句”）与任何的数据类型相连。

与双精度型相连时最多不超过八个文字型字符。实型最多允许四个字符，整型不超过二个字符，而逻辑型只允许一个字符。

第三节　数据存储

存储每种数据类型所需要的存储量，是根据与TYPE语句一起说明的变量而变化的（参阅第九章的“TYPE语句”一节）。

数据类型及所需的存储量列表如下：

1．字节、单字节整型、单字节逻辑型及逻辑，每种数据类型均需要一个字节（8位）的存储单元。

2．双字节整型、双字节逻辑型及整型，每种数据类型需要二个字节（即16位）的存储单元。

3．实型、四字节整型及四字节实型，每种数据类型需要四个字节（即32位）的存储单元。

4．双精度型与八字节实型，每种数据类型需要八个字节（即64位）的存储单元。

第四节　函数成分

可由FORTRAN编译程序识别的数据名可以分成三种不同类型：常量、变量与数组。在源程序中，常常使用这些数据名去识别并赋值给某个特定的项。

4.1　常量

FORTRAN中的常量由其实际的值来标识。一个常量可以是正值，也可以是负值。标明为负值的符号（－）总是放在该负常量的前面；而标明为正值的符号（＋）可以放在该常量的前面，也可以不标。

4.2　变量

FORTRAN中的变量由其符号名来标识。FORTRAN中的符号名由1～6个字母或数字字符组成的各不相同的串来表示。符号名的第一个字符必须是字母。

注：系统变量名及运行时的子程序名可以由在其他变量名前加上美元符号（$）来识别。因此，为了避免冲突，推荐FORTRAN源程序中的符号名用非“$”的一些字符来开头。

4.3　数组

FORTRAN中的数组是按某种意义排列起来的一组有序数据的集合。该数据的特点在于维数的性能。数组可以是一维、二维或三维的，并由某个符号名来识别。数组中每个元素可由下标来单独寻址（关于数组的更详细的资料，可以参阅第九章“说明语句”）。

数组元素

数组元素是数组所建立起来的数组集合中的一个成分。FORTRAN中对于数组元素的引用由数组名加一个下标来完成。下标用来识别该数组中某个确定的元素。

下面给出了下标使用的规则：

1．括号中可以含有一、二或三个下标表达式。

2．如果括号中有二或三个下标表达式，则它们之间用逗号（，）来分隔。

3．下标表达式的个数必须与数组所指定的维数相同（对于该规则的例外是EQUIVA

LENCEC（等价）语句。对于该例外的整个讨论可参阅第九章“说明语句”）。

4．下标表达式可以用下述形式之一来书写：

K	C * V	V - K
V	C * V + K	C * V - K
V + K		

其中：C与K是整型常量，而V是整型变量名。

5．下标本身不允许再是下标

合法下标的例子是：

X(2 * J - 3, 7)

A(I, J, K)

I(20)

C(L - 2)

Y(I)

表3-1　　数据类型的存储分配

类型	分　配
INTEGER	其存储形式需要二个字节。 负数用正数表示法的二进制补码形式表示。
LOGICAL	其存储形式需要一个字节。 零为假值，非零为真值。 一个非零值的字节表示为真值（逻辑常量真值.TRUE.由十六进制数FF表示），而一个值为零的字节表示为假值。将其用作算术数值时，可把逻辑型数据当作 - 128～ + 127间的整型数处理。
REAL	其存储形式需要四个字节。 第一个字节以超过（八进制数）200的记数法来表示其幂。即八进制数200表示为2的零次幂，小于200的数为负次幂，大于200的数为正次幂。根据定义，如果其幂为零则该数就为零。后三个字节构成其尾数，而尾数是被规格化了的，即其最高位是1。实际上不必去存储它。故最高位被用来指明该数的符号，如果是1则表示该数为负、是零则表示为正。尾数被假定为是二进制小数，其小数点总是位于尾数的最左边。
EXTENDED INTEGER	其存储形式需要四个字节。 负数采用正数表示法的二进制补码形式。
DOUBLE PRECISION	其存储形式需要八个字节。 除尾数多用了四个字节外，双精度型数据的内部形式与实型数据相同。

第四章　FORTRAN表达式

第一节　综　　述

FORTRAN表达式由单个操作数或以操作符相连的一组操作数所组成。

FORTRAN提供了算术表达式与逻辑表达式两种类型。在以下章节中描述了操作数、操作符及两类表达式的使用规则。

第二节　算术表达式

算术表达式由算术操作数及算术操作符组成。算术表达式的计算可得到单个数值。算术操作数可以是：

- 常量
- 变量名
- 数组元素
- 函数引用

算术操作符及其功能是：

操作符	功　能
* *	取幂
*	乘法
/	除法
+	加法或一元正号
-	减法或一元负号

下述规则定义了所有允许的算术表达式形式：

1. 单个常量、变量名、数组元素引用或函数引用可以是一个表达式。

例如：S(I)　JOBNO　217　17.26　SQRT(A+B)

2. 如果E是第一个字符为非操作数的表达式，则+E与-E就称作带符号的表达式。

例如：-S　+JOBNO　-217　+17.26　-SQRT(A+B)

3. 如果E是一表达式，则(E)就表示计算表达式E后所得到的量。

例如：(-A)　(JOBNO)　-(X+1)　(A-SQRT(A+B))

4. 如果E是一无符号表达式，而F是一任意的表达式，则F+E、F-E、F*E、F/E及F**E都是表达式。

例如：-(B(I, J)+SQRT(A+B(K, L)))

1.7E-2**(X+5.0)

-(B(I+3, 3*J+5)+A)

5．要计算的表达式可以是整型、实型、双精度型或逻辑型，结果类型由表达式元素的数据类型来决定。

如果表达式的元素类型不完全相同，则表达式的类型可由具有最高级类型的元素来决定。类型的级别（由高到低）如下：双精度型、实型、四字节整型、整型和逻辑型。

6．表达式可以含有如下所示的嵌套的括号元素。

例如：A＊(Z－((Y＋X)/T))＊＊J

其中：Y＋X是最内层的元素，(Y＋X)/T 是次内层元素，Z－((Y＋X)/T) 是外层元素。在这样的表达式中，要注意所用左括号的个数必须与右括号的个数相等。

表达式的计算

算术表达式可以按照下述规则来计算：

1．先计算括在括号中的表达式元素。如果括号中的表达式是嵌套，则先计算最内层，然后是次内层，直到整个表达式计算完。

2．在同层括号内或无括号时按优先级次序执行运算，操作符优先级按大小排列如下：

a．函数运算

b．取幂运算

c．乘法与除法

d．加法与减法

表达式A＊(Z－((Y＋R)/T))＊＊J＋VAL是按下列顺序进行计算的：

Y＋R＝e1

(e1)/T＝e2

Z－e2＝e3

e3＊＊J＝e4

A＊e4＝e5

e5＋VAL＝e6

3．表达式X＊＊Y＊＊Z是不允许的，但它可以写成如下的形式：

(X＊＊Y)＊＊Z　或　X＊＊(Y＊＊Z)

4．引用数组元素要计算其下标。下标表达式可按照一般表达式的相同规则来计算。

第三节　逻辑表达式

逻辑表达式可以是下述任一类型：

1．单个逻辑常量（即.TRUE.、.FALSE.），逻辑变量，逻辑数组元素或逻辑函数引用（参阅第十章“函数与子程序”）

2．由关系运算符分隔的两个算术表达式（即关系表达式）

3．逻辑操作符可以对逻辑常量、逻辑变量、逻辑数组元素、逻辑函数、关系表达式或其他逻辑表达式进行操作。逻辑表达式的值不是取真值.TRUE.，就是取假值.FALSE.。

3.1　关系表达式

关系表达式的一般形式如下：

e1 r e2

其中 e1 与 e2 是算术表达式，而 r 是关系运算符。下面列出六个关系运算符：

.LT.　小于

.LE.　小于或等于

.EQ.　等于

.NE.　不等于

.GT.　大于

.GE.　大于或等于

如果该关系运算所定义的条件具备，该关系表达式就取真值.TRUE.，否则就取假值.FALSE.。例如：

A.EQ.B

(A**J).GT.(ZAP*(RHO*TAU-ALPH))

3.2 逻辑运算符

表4-1列出了逻辑运算符，U与V均表示为逻辑表达式。

表4-1　　逻辑运算符

.NOT.U	该表达式的值是U的逻辑补码（即所有为1的位变为0，所有为0的位变为1）
U.AND.V	该表达式的值是U与V的逻辑积（即仅当U与V中相应的位均是1时，结果值中相应的位才为1）
U.OR.V	该表达式的值是U与V的逻辑和（即如果U与V中有一项相应位是1，或者两者之相应位均是1，则结果中相应位也是1）
U.XOR.V	该表达式的值是U与V的异或（即如果U与V中相应位分别是1和0或0和1，则结果值中相应位是1） 例如：假如U=01101100　而V=11001001 则：.NOT.U=10010011 U.AND.V=01001000 U.OR.V=11101101 U.XOR.V=10100101

下面列出了构造逻辑表达式所需的附加条件：

1．任何逻辑表达式均可以括在括号中。然而当操作符为逻辑非.NOT.时的逻辑表达式含有两个或多个元素时，就必须将其括在括号中。

2．在运算的优先级中，可以使用括号去规定表达式执行的先后次序。使用括号或者没有括号时，可按下述规定去理解其运算的先后次序：

a．函数引用

b．取幂（**）

c．乘法与除法（*与/）

d．加法与减法（+与-）

e．小于.LF.、小于等于.LE.、等于.EQ.、不等于.NE.、大于.GT.、大于等于.GE.

f. 逻辑非.NOT.

g. 逻辑乘.AND.

h. 逻辑加.OR.、异或.XOR.

例如：

表达式X.AND.Y.OR.B(3, 2) .GT.Z可按下述次序执行：

e1 = B(3, 2).GT.Z

e2 = X.AND.Y

e3 = e2.OR.e1

而表达式X.AND.(Y.OR.B(3, 2).GT.Z) 可按下述次序执行：

e1 = B(3, 2).GT.Z

e2 = Y.OR.e1

e3 = X.AND.e2

3. 除第二个操作符是逻辑非.NOT.外，必须避免两个逻辑操作符相连。例如：

A.AND..NOT.B是允许的

A.AND..OR.B是不允许的

第四节 文字型、字符型及十六进制常量

在表达式中允许用文字型、字符型及十六进制常量代替整型常量。这些专用常量计算得到一个整型值。因此必须限制到2个字节的长度。对于这点的例外仅仅是：

1. 较长的文字型或字符型常量可以用作子程序的参数。

2. 当与实型变量或四字节整型变量相连时，文字型、字符型及十六进制常量在DATA语句中可以不超过四个字节；而当与双精度型变量相连时，可以不超过八个字节。

文字型、字符型常量可以由括在一对单引号中的整个字符串来构成。两个连续的单引号可以用来表示字符串中的单引号字符。例如：

'THIS IS A LITERAL'

十六进制常量可以由字母Z 或X 后面跟有括在一对单引号中的、不超过四个的十六进制数字（0～9）与（A～F）来指定。例如：

X'FFFF'

Z'AB'

第五章　赋值语句

第一节　综　　述

赋值语句的作用是使一个表达式的值与一个符号名联系起来。通常符号名指的是变量。

赋值语句有三种形式：

1．算术赋值语句

2．逻辑赋值语句

3．ASSIGN语句

算术赋值语句是将一算术表达式的值赋给一算术变量或数组元素。逻辑赋值语句是将一逻辑表达式的值赋给一逻辑变量。ASSIGN语句是将一语句标号赋给一整型变量。

第二节　算术赋值语句

算术赋值语句的一般格式为：V = e

这里的 V 是任意一个变量或数组元素，e 是一个表达式。在 FORTRAN 语义中定义〔=〕等号是“被××替代”，而不是“等于××”。

执行算术赋值语句时，将求出等号右边表达式的值并将其放入等号左边变量或数组新分配的存储空间位置中。

对算术赋值语句规定了下列条件：

1．变量 V 和等号必须在同一行出现。当语句是逻辑 IF 语句的一部分时，也是如此（逻辑 IF 语句的详细讨论见第六章“FORTRAN 控制语句”）。

2．包含有“V = ”的项必须是语句的起始行，除非句子是逻辑 IF 语句的一部分。在该种情况下，“V = ”必须出现在 IF 之后第一行结束之前。

3．如果变量 V 和表达式 e 的数据类型不同，那么表达式所确定的值将被转换。如果可能，应使该值与变量数据类型一致。例如：如果表达式的值是 32800，若要把这个值赋给一整型变量，则将产生不可预料的结果。

表5-1说明了哪种类型的表达式可以等于某种变量类型。y 表示一个有效的取代，y 的下标表示转换条件。

表5-1　　表达式类型（e）

变量类型 (V)	整数型 Integer	实数型 Real	逻辑型 Logical	双精度型 Double	扩展整数型 Ext Int
整数型	y	y_a	y_b	y_a	y_b
实数型	y_c	y	y_c	y_a	y_c
逻辑型	y_d	y_a	y	y_a	y_d

双精度型	y_a	y	y_a	y	y_a
扩展实数型	y_f	y_a	y_a	y_a	y
			$y_{b,g}$		

下标说明：

a. 转换实数表达式的值为整型数

b. 符号由第二字节扩展

c. 把表达式整型值以实数型表示法赋给变量

d. 赋给变量以整型表达式的截断值（用低字节，不管符号如何）

e. 赋给变量以实数型表达式的四舍五入值

f. 符号由第三和第四字节扩展

g. 赋给变量以扩展整型表达式的截断值

第三节　逻辑赋值语句

逻辑赋值语句的一般格式是：V = e

这里 V 是一逻辑变量或一逻辑数组元素，而 e 是一逻辑表达式。这个表达式 e 在赋值语句执行时要进行求值的计算。这个计算的结果可能是零（假）或是非零（真）。表达式 e 计算的结果将被放入逻辑变量新分配的存储器单元。

变量或数组元素 V 必须明确定义为逻辑变量类型。例如：

```
FLAG = .TRUE.
TEST = .FLASE.
COMP = (L.LT.10)
```

第四节　ASSIGN（标号赋值）语句

ASSIGN 语句的作用是将一语句标号赋值给一整型变量。此时，整型变量可以在随后的赋值 GOTO 语句中作为这种控制转移的目的行号（对于赋值 GOTO 语句的详细讨论参见第六章“FORTRAN 控制语句”）。

语句的一般格式是：ASSIGN j to i

这里 j 是一个执行语句的语句标号，而 i 是一个整型变量。

虽然整型变量可以通过重新定义算术整型值来正确使用它，但是对已经赋以语句标号的整形变量来说则不可以用来作算术变量。例如：

语句 ASSIGN 400 TO LABEL

将会把变量 LABEL 同标号为 400 的语句联系起来。含有变量 LABEL 的算术运算将是无效的。

语句 LABEL = 100

将不会把变量 LABEL 同标号为400的语句联系起来。这个变量 LABEL 现在就可被用作算术操作，但不可作为语句标号使用。

第六章　FORTRAN控制语句

第一节　综　　述

FORTRAN 控制语句是可执行语句，它影响并指引 FORTRAN 程序的逻辑流程。这类 FORTRAN 控制语句有如下一些语句：

1. GO TO 转移语句：
   1）无条件转移的 GO TO 语句
   2）按计算值转移的 GO TO 语句
   3）按所赋值转移的 GO TO 语句
2. IF 条件语句：
   1）算术 IF 语句
   2）逻辑 IF 语句
3. DO 循环语句
4. CONTINUE 继续运行语句
5. STOP 停止语句
6. PAUSE 暂停语句
7. CALL 调用语句
8. RETURN 返回语句
9. END 结束语句

第二节　GO TO（转移）语句

2.1　无条件转移语句

无条件 GO TO 语句是在程序单元内部将控制转移到其他一些语句去执行。这种语句有下面的格式：

```
GO TO K
```

这里 K 是在同一程序体中可执行语句的语句标号。

这个语句无条件地使控制转移到同指定标号一致的语句去执行。每次执行 GO TO 语句时，则把控制转移到同一语句。例：

```
    GO TO 376
     ⋮
376 A = 100
```

这个语句把控制转移到标号为376的语句。

2.2　计算转移语句

按计算值转移的 GO TO 语句的作用是根据在语句中给定的一个整型变量值，控制转移

到一个语句去执行。

按计算值转移的 GO TO 语句有下述形式：

GO TO(K_1,K_2,…, Kn),j

这里 Ki 是语句标号，而 j 是一个整型变量，且 $1 \leqslant j \leqslant n$。这个语句使控制转移到标号为 Ki 的语句。如果 $j < 1$ 或 $j > n$，控制转移到 GO TO 语句之后的下一条语句。例：

GO TO(7, 70, 700, 7000, 70000),J

当 J = 3 时，按计算使控制转移到标号为700的语句。若 J = 0 或 J = 6 则控制转移到GO TO 语句之后的下一条语句。

2.3 赋值转移语句

这种语句使控制转移到整型变量所代表的标号语句。

按所赋值转移的 GO TO 语句的格式如下：

GO TO j,(K_1,K_2,…Kn)

GO TO j

这里 j 是一个整型变量名，而Ki（如果在语句中出现）是可执行语句的语句标号。

这种语句使控制转移到标号等于整型变量 j 所赋值的语句去执行。

j 的值必须经过一个 ASSIGN 赋值语句来确定（ASSIGN 语句的详细资料参见第五章“赋值语句”）。

限定 (QUALIFICATIONS)

1. ASSIGN 语句在逻辑上必须优先于赋值 GO TO 语句。

2. ASSIGN 语句必须赋一个值到 j，j 是一个语句标号。如果我们指定了 K'S 的列表，则 j 就包括在这个表中。例：

```
ASSIGN 80 TO LABEL
  ⋮
GO TO LABEL, (80, 90, 100)
```

只有语句标号为80、90、100才可以赋值给 LABEL。根据 GO TO 语句的执行，控制转移到变量所赋的标号处。在上面的程序段中，控制转移到标号为80的语句处。

GO TO TRANS

赋给 TRANS 的值必须是程序体内可执行语句的标号。当 GO TO 语句执行时，控制转移到赋值给整型变量 TRANS 的标号处。

第三节 IF（条件）语句

3.1 逻辑 IF 语句

一个逻辑 IF 语句引起一个条件语句执行。逻辑 IF 语句的格式如下：

IF (u) S

这里 u 是一个逻辑表达式，而 S 是除 DO 语句或其他逻辑条件 IF 语句之外的任意一个可执行语句。逻辑表达式 u 的值为真（.TRUE.）或假（.FALSE.）。第四章“FORTRAN 表达式”中有逻辑表达式的讨论。

控制条件

如果 u 是假（.FALSE.），则语句被忽略，且控制转移到逻辑 IF 语句后面的下一条语句。

如果 u 是真（.TRUE.），则控制转移到语句 S 之后的程序，且控制按正常的条件进行。

如果 S 是一个替换语句（V = e），则变量及等号必须在同一行上。运行可能直接地跟在 IF(u) 之后或在一个分隔的连续行之后。

例：

```
IF(I.GT.20)GO TO 115
IF(Q.AND.R)ASSIGN 10 TO J
IF(Z)CALL DECL(A,B,C)
IF(A.OR.B.LE.PI/2)I=J
```

3.2 算术 IF 语句

算术 IF 语句是使控制转移到由算术表达式的值所决定的一连串语句之一处去执行。

算术 IF 语句的格式为：

$$IF(e)m_1,\ m_2,\ m_3$$

这里 e 是一个算术表达式，而 m_1、m_2、m_3 是语句标号。表达式 e 所求的值确定三种可能转移之一。

如果 e 是：	转移到：
<0	m_1
$=0$	m_2
>0	m_3

例：

语句	表达式的值	转移到
IF(A)3，4，5	15	5
IF(N-1)50，73，9	0	73
IF(B-100)6，7，8	-15	6

第四节 DO（循环）语句

DO 语句是为反复执行一串语句提供的一种方法。

DO 语句可选择下面两种格式之一：

$$DO\ K\quad j=m_1,\ m_2,\ m_3$$

或 $$DO\ K\quad j=m_1,\ m_2$$

K 必须是一个语句标号，j 必须是一个整型数或逻辑变量。m_1、m_2、m_3必须是整型数常量或者整型数或逻辑变量。j、m_1、m_2和 m_3必须为正数。

变量定义如下：

K　终止语句标号

j　控制变量

m_1　初始变量

m_2 终止变量

m_3 递增变量

如果 m_3 是 1，则可以省略。

标号为 K 的语句称为终止语句，它必须是一个可执行语句。终止语句在物理上必须跟在相关的 DO 语句之后。紧跟在 DO 语句后的可执行语句，一直到包括终止语句在内构成了 DO 语句的区域。终止语句不可以是算术 IF 语句、GO TO 语句、RETURN 语句、STOP 语句、PAUSE 语句或其他任何 DO 语句。

如果终止语句是一个逻辑 IF 语句，而且表达式的值是假（.FALSE.），则这个语句在 DO 语句区域内反复执行。如果表达式的值是真（.TRUE.），则先执行逻辑 IF 语句，然后在 DO 区域内反复执行。逻辑 IF 语句不可以是 GO TO 语句、算术 IF 语句、RETURN 语句、STOP 语句或 PAUSE 语句。

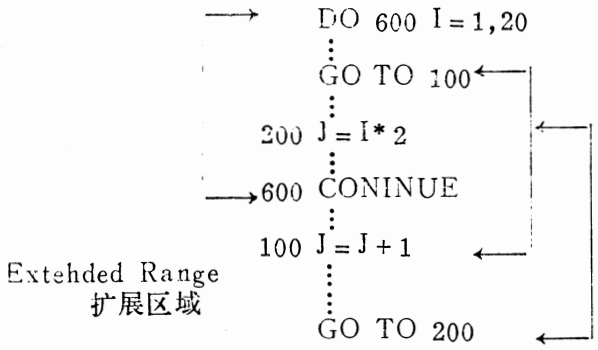
控制整型变量 j 被叫做 DO 语句范围的索引（index），这个索引必须为正，并且不会被这个语句范围中的任何一个语句所修改。如果 m_1、m_2、m_3 是 INTEGER*1 型变量或常量，这时 DO 循环较快，但重复也是最短，其范围被限定在127次重复。

在 DO 区域内语句第一次执行 $j=m_1$，第二次执行 $j=m_1+m_3$，第三次执行 $j=m_1+2*m_3$ 等等，直到 j 小于或等于 m_2 的值，DO 语句就结束了。在 DO 语句区域内的句子，总是要执行一次的，即使是 $m_1>m_2$。

当 DO 语句已经满足，控制转到语句后面的终止语句，亦即，控制转回到 DO 语句后面的第一个执行语句。

一个 DO 语句的区域可以扩展到包括所有的语句，这些语句在逻辑上是可以在 DO 和它的结束语句之间被执行的。这样，DO 语句区域的部分可能在逻辑上处于 DO 语句和它的结束语句之间，而不是在物理上处于 DO 语句和它的结束语句之间，这被算为扩展区域。

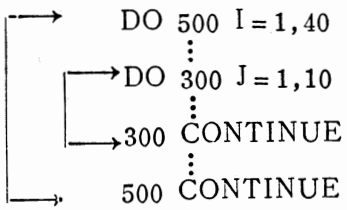
扩展区域的例子：



在上面的程序段中，DO 循环的区域已扩展到包括语句标号 100，这使控制返回到 DO 循环的语句。尤其重要的是控制变量既不能在 DO 循环中，也不能在扩展区域内被改变。还应该注意的是在结束语句之前的一个语句必须使控制返回到 DO 循环。

在 DO 语句的区域内，可以有别的DO 语句。这种情况称之为 DO 语句的嵌套。也就是说，如果一个 DO 的区域包括另外的 DO 语句，则内部的 DO 的范围必须整个地包含在外部 DO 的范围内，内部 DO 的结束语句也可以是外部 DO 的结束语句。

嵌套DO循环的例子：



在上述程序段中，应注意内部循环的区域是整个地包含在外循环的范围内。

第五节 CONTINUE（继续）语句

CONTINUE继续语句的作用是控制转移到下一个可执行语句。

CONTINUE语句的格式如下：

CONTINUE

当结束语句不允许采用通常的各种语句时，CONTINUE在DO语句中则常常被用作结束语句。

第六节 STOP（停止）语句

STOP语句常被用来结束程序的执行。

STOP语句可用下列格式之一：

STOP

或 STOP C

这里C是任何一个1～6个字符的字符串。在目的程序执行时碰到STOP，如果字符C存在，则在操作控制台上显示字符C，同时程序停止执行。因此STOP语句是程序的逻辑结束。

第七节 PAUSE（暂停）语句

PAUSE语句是暂时挂起程序的执行。

PAUSE语句可用下面的格式之一：

PAUSE

或 PAUSE C

这里C是直到6个字符的任意一个字符串。在目的程序执行时，碰到PAUSE字符C（如果C存在的话），在操作控制台上显示出来，同时程序中止进行。在控制台上打一个“T”和回车，程序就结束执行；若再打入任意一个其他的字符和回车时，程序则继续执行。

第八节 CALL（调用）语句

CALL 语句使控制转入 SUBROUTINE 子程序，同时提供作为子程序使用的参数。

CALL调用语句可以有以下格式之一：

CALL S(a_1,a_2…an)

或 CALL S

这里 S 是 SUBROUTINE 子程序名，而 ai是子程序所用的实际变量。CALL 语句将在第十章“功能与子程序”中详细讨论。

第九节 RETURN（返回）语句

一个子程序的逻辑结束是 RETURN 返回语句。

RETURN 语句的一般格式是：

RETURN

在第十章“功能与子程序”中，有对 RETURN 格式、语句使用和解释的详细讨论。

第十节 END（结束）语句

END 语句在物理上必须是任何一个 FORTRAN 程序的最后的语句。

它的格式是：

END

END 语句是一个可执行语句，并且可以有一个语句标号。当一个主程序体结束时，碰到 END，就会使控制转移到系统出口程序 $EX，而使控制返回到操作系统。

第七章　输入与输出语句

第一节　综　　述

FORTRAN 的 I/O 是由 READ 和 WRITE 语句来实现的。READ 语句被用作输入，而 WRITE 语句被用作输出。非格式化的 I/O 传送二进制信息，格式化的 I/O 用来连接格式化的要求以便控制数据的编辑和转换。

READ 和 WRITE 语句是通过一个逻辑单元号来指定 CP/M 下 I/O 设备的，这些逻辑单元号事先就分配给了一些指定的设备。

READ 和 WRITE 语句分类如下：

1．非格式化顺序 I/O

规定二进制数据的传送是利用一个顺序 I/O 设备。

2．格式化顺序 I/O

规定字符数据的传送是利用一个顺序 I/O 设备。为了数据的编辑和传送，必须指定FORMAT 语句。

3．非格式化的随机 I/O

规定二进制数据的传送是利用一个随机存取 I/O 设备。为了便于控制数据的编辑和传送，必须指定 FORMAT 语句。

第二节　逻辑单元号

FORTRAN 同 CP/M I/O 设备的通讯是通过逻辑单元号（LUN）来实现的。逻辑单元号必须是一个无符号的整型数或者是 1～10 范围内的整型变量。如果是用整型变量，则其必须在 I/O 语句执行之前把整数赋值给变量。

逻辑单元号 1～10已经事先赋值给指定的 CP/M 下的 I/O 设备。逻辑单元号 1、3、4 和 5 是引用控制台终端；单元 2 已分配给硬拷贝设备；单元 6～10分配给磁盘文件。下表总结了这些分配，其中有 d 的是指当前缺省盘。

LUN	CP/M 下的设备名
1	TTY　(终端)
2	LST　(打印机)
3，4，5	TTY　(终端)
6	d:FORT06.DAT
7	d:FORT07.DAT
8	d:FORT08.DAT
9	d:FORT09.DAT
10	d:FORT10.DAT

改变逻辑单元缺省值用OPEN子程序来完成。下面FORTRAN语句说明了这点。

CALL OPEN C3, 'DATA DAT', O

这就把LUN3和当前所选驱动器上的文件DATA.DAT联系起来了。以后对这个文件进行I/O操作，可由READ或WRITE语句中指定的单元来代替这个文件，从而实现对文件的I/O操作。

注意，在这个例子中，文件名必须用空格填满11个位置，而且文件扩展名必须在最后三个位置上出现。

FORMAT的详细说明

所有格式化I/O语句都要指定FORMAT语句，以便在数据传送时定义所完成的变换和编辑的要求。FORMAT语句的引用可以是语句标号或者是包括格式化说明的一个数组名。FORMAT语句将在第八章中讨论。

第三节 输入与输出列表

READ/WRITE语句的通常格式可以包含一个数据名的有序列表，它用于识别传送的数据。列表中，项的顺序则为被传送的顺序。列表格式如下：

$m_1, m_2, \cdots m_n$

这里mi是列表项，用逗号分开。

3.1 输入与输出列表的长度

I/O列表的长度指定了存放列表元素所要求的字节数。按照I/O的几种形式，知道I/O列表的长度是很重要的。I/O列表所需要的字节数是每个独立的I/O单元的数目和数据类型的函数。下表说明这个关系。

数据类型	元素要求的字节数
LOGICAL 逻辑型	1
INTEGEG 整数型	2
EXTENDED INTEGER 扩展整数型	4
REAL 实型	4
DOUBLE PRECISION 双精度型	8

3.2 简单列表

一个简单的I/O列表是由一个变量、一个数组或数组名组成。可以用弧括把这些项括起来，而不会改变它们内部的含义。例：

A

C (1, 2)

若没有下标的数组名出现在表中，则它等同于数组依次按每个元素排列的列表。例：

如B是一个二维矩阵，列表项B等于：

B (1, 1), B (2, 1), B (3, 1), …B (1, 2), B (2, 2) …B (j, k).

这里j和k是B下标的极限值。

3.3 隐含的DO列表

隐含的DO列表在I/O列表内作为重复之用。隐含DO列表的格式是：

(I/O list, i = m_1, m_2, m_3)

或 (I/O list, i = m_1, m_2)

元素 m_1、m_2、m_3 具有在 DO 语句中的同样含义。这种隐含的 DO 列表功能如同 I/O语句是在 DO 循环区域内（有关 DO 语句更多的细则参见第六章）。DO 隐含适用于括弧内具有隐含意义的所有列表项。例：

隐含的 DO 列表	等同的列表
(X(I),I = 1, 4)	X(1),X(2),X(3),X(4)
(Q(J),R(J),J = 1,2)	Q(1),R(1),Q(2),R(2)
(G(K),K = 1,7,3)	G(1),G(3),G(7)*1
(A(I,J),I = 3,5),J = 1,9,4	A(3,1),A(4,1),A(5,1)
	A(3,4),A(4,4),A(5,4)*2
	A(3,9),A(4,9),A(5,9)
(R(M),M = 1,2),I,ZAP(3)	R(1),R(2),I,ZAP(3)
(R(3),T(I),I = 1,3)	R(3),T(1)
	R(3),T(2)
	R(3),T(3)

如例中所述，在存储器中，一个按顺序存放的数组单元，可按不同的顺序进行传送。数组 A(3,3)按顺序占用的存储器是：

A(1,1),A(2,1),A(3,1),
A(1,2),A(2,2),A(3,2),*3
A(1,3),A(2,3),A(3,3)。

根据指定的 DO 隐含列表项的数组的传送如：

((A(I,J),J = 1,3),I = 1,3)

传送的顺序是：

A(1,1),A(1,2),A(1,3),
A(2,1),A(2,2),A(2,3),
A(3,1),A(3,2),A(3,3)。

* 1 原文是 G(1)、G(4)、G(7)，其中的 G(4)是笔误，应改为 G(3)

* 2 原文的隐含列表 J = 1, 9, 4，而数组表（等同列表）中，中间一行是 A(3,5),A(4,5),A(5,5),应改为 A(3,4),A(4,4),A(5,4)

* 3 原文的矩阵第 2 行少一个“，”，应加上——译者注

第四节 顺序输入与输出

4.1 非格式顺序输入与输出

非格式化的顺序 I/O 用来传送二进制数而不做任何数据的编辑或者格式化。传送数据的总量是 I/O 列表中元素的数量和数据类型的函数。

I/O 列表必须包括在 I/O 语句中，一个 I/O 列表的错误归结为是一种编译的错误。

非格式化顺序 I/O 语句的两种格式是：

READ(U,ERR = L_1,NED = L_2) K

WRITE(U,ERR = L_1,NED = L_2) K

这里U是指定的一个逻辑单元号

L_1是指定的一个I/O错误分支（可选）

L_2是指定的EOF分支（可选）

K是一个I/O列表

非格式化顺序记录的长度(如果需要的话)可大于128字节。如果记录长度大于128字节，一些特殊考虑必须加以注明。

特殊说明

非格式化顺序I/O处理器假定是128字节的记录，任何非格式化的一个顺序I/O文件的I/O把记录指针定位在128字节物理记录的结束处，如果所读的字节数不同于原来的字节数，就可能产生不希望的结果。

例如：假定几个180字节的记录写入一个非格式化顺序文件。并假定有100个字节是从文件中第180个字节的记录中读，那么，这100个字节的记录被输入并赋值给I/O列表中的元素。记录指针则定位在下一个128字节物理记录的开始。注意，这将导致在随后的读语句中开始读时，读了额外的数据，而不会跳过额外的数据。

避免这种情况发生的最好方法是保持输入记录和输出记录长度一样。如果遵守简单的预防措施，则利用非格式化顺序I/O就可以传输大于128字节的记录。

4.1.1 非格式化顺序输入（READ)

非格式化顺序输入（READ）是读一个逻辑记录，这个逻辑记录可以扩展并通过一个以上的物理记录。如果逻辑记录的尺寸大于128字节，则必须注以特殊说明（见前面的“特殊说明”部分）。

传送数据的总数对应I/O列表K中的数量和数据类型。

如果I/O列表中的元素与输入记录字段相同，则整个记录被读。如果在I/O列表中的元素比输入记录的字段少，则在记录中未读的项可以跳过。如果在I/O列表中的元素大于输入记录字段，则为了填满I/O列表所需的多个记录就会被读入。例：

READ(6)A，B，C

它从外部存储器读一个记录。读记录与逻辑单元6相关。赋输入数据到变量A、B、C。

4.1.2 非格式化顺序输出（WRITE)

输出语句(WRITE)将在I/O列表中所指定的数据输出到由逻辑单元号U所指定的文件。每次WRITE语句执行时，总有一个逻辑记录被输出(如果这个逻辑记录大于256个字节，则必须注明某些特殊的考虑。见“特殊说明”中的注释）。例：

WRITE(6)I,J,K,L

赋值给整型变量I、J、K、L的数据是输出到与逻辑单元6相关的那一文件。这个语句输出128个字节，但只有前8个字节包含数据（4个整型变量乘2字节除以每个整型元素），其余的128个字节包含ASCII NUL字符（0）。

4.2 格式化顺序输入与输出

格式化的顺序I/O用来传送字符数据。数据的传送是按顺序的方法进行。为了控制数据的编译和格式化，FORMAT语句必须被引用（见第8章“FORMAT语句”)。

这种语句的两种格式是有效的：

READ(U,f,ERR = L_1,END = L_2)K

WRITE(U,f,ERR = L_1,END = L_2)K

这里 U 指定一个逻辑单元号

f 指定 FORMAT 语句标号

L_1指定 I/O 错误分支（可选）

L_2指定 EOF 分支（可选）

K 是 I/O 列表（可选）

每一个I/O记录长度不能大于128字节。企图使写出的记录大于127字节，都会使所写记录截断超过127字节之后的部分。如果记录相当长，以致超出内部缓冲区的长度，则会产生一个运行时的错误。

若要读一个比127字节还要大的记录，则会导致在内部 I/O 缓冲区出现溢出，从而产生一个运行时的 I/O 错误。

TORTRAN 的程序员有责任校验记录的长度是小于还是等于127字节。记录长度是 I/O 列表中元素的数量和数据类型的一个函数。I/O 列表的内容包含在第七章的“简单列表”和“隐含 DO 列表”中。

END = 和 ERR = 语句的使用是可选择的。ERR = 分支仅仅在有关硬件 I/O 错误发生时使用。END = 分支仅在当文件出现条件结束时使用。如果省略这些选择，那么硬件的错误和文件的条件结束都将引起致命性的运行错误。这样程序将被终止执行。

4.2.1 READ（格式化顺序输入）

当输入语句使用变量表时，应当输入与列表 K 中一些项相对应的元素。这些数据将被编辑，同时应根据标号为 f 的格式语句的规定进行转换。

每次 READ 语句开始执行时，应从输入文件中读入一个新的记录。由一个单独的 READ 语句输入的记录数是由列表 K 和格式说明所确定的。

每一个记录的长度不得超过 127 字节。

如果 I/O 列表的元素和输入记录字段数一样多，整个记录将被读入，如果 I/O 列表的元素少于输入记录中的字段数，则未读入的字段将被跳过。

如果 I/O 列表中元素多于输入记录中的字段数，则需要填满 I/O 列表所需数量的记录应被读入。例：

```
      READ (6,100) A, B, C, D
        ⋮
  100 FORMAT (4 F 6.2)
```

上面的程序段将从与逻辑单元号 6 相关的外部存储设备输入数据。输入的数据赋给变量 A、B、C、D，而输入数据将根据标号为100的 FORMAT 语句指定的格式将数据格式化。

当输入语句 READ 不采用变量列表时，它将把文字数据读到已有的文字字段中。在 READ 语句中所指定的 EORMAT 语句，包含这个新的文字字段。例：

```
      READ (6,100)
        ⋮
  100  FORMAT (10H1234567890)
```

这个程序段将会读入与逻辑单元号6相关的文件中的最后10个字符。输入数据将替换FORMAT语句中的字符1 2 3 4 5 6 7 8 9 0。

4.2.2 WRITE（格式化顺序输出）

当输出语句采用变量表时，列表K所指定的数据将输出到逻辑单元号u所指定的文件。FORMAT语句规定了数据的外部表示法。由单个的WRITE语句可以输出所需的多个记录。记录输出的数量是由列表和FORMAT说明来确定的。连续的数据将一直输出，直到表中所指定的数据送完为止。程序员的责任是检查记录长度不要超过127字节。

FORMAT语句的第一个字符假定是回车控制字符，当写一个磁盘文件时，好的编程办法是采用一个加号〔+〕作为回车控制字符。如果不用，则回车控制字符就存放在每个记录的第一个字段处（见第8章“FORMAT语句”）。例：

```
      WRITE (3,10) A, B, C, D
        ⋮
10    FORMAT('+',4A4)
```

将赋值给变量A、B、C、D的数据输出到逻辑单元号指定的设备并按照标号为10的FORMAT语句进行格式化处理。应注意加号〔+〕这种回车控制字符的使用。

如果输出语句（WRITE）不采用变量列表，而且所写字符已经在FORMAT语句内指定，就可以用来写字母数字信息。

例如在单元1上要写字符'A CONVERSION'时，可以用下面的程序段：

```
  WRITE (1,26)
    ⋮
26FORMAT ('1',12HA CONVERSION)
```

第五节 随机输入与输出

5.1 非格式随机输入与输出

对非格式的随机磁盘存取来说，在READ或WRITE语句中是用REC=n选择项来指定所要求的记录号的，这种I/O方式允许对磁盘文件的第n个记录直接存取。

记录号n可以是整型变量或整型常数，如采用的是整型变量，则整型值必须在I/O语句执行之前赋给这一变量。

数据传送无须任何编辑或格式处理，实际上是传送一个128字节的物理记录。

两种非格式化随机I/O的方式是：

READ (u,REC=n, ERR=L_1, END=L_2) K（输入）

WRITE (u,REC=n, ERR=L_1, END=L_2) K（输出）

这里：

u	逻辑单元号
n	存取的记录号
L_1	I/O的错误分支（可选）
L_2	EOF分支（可选）
K	一个I/O列表

如果指定记录号的整型值是负或者是0，则会产生运行时的I/O错误。

逻辑单元号u必须对应一个磁盘文件。试图随机存取一个没有随机存取能力的设备都会产生运行时I/O错误。

ERR=和END=语句的使用是可选择的，ERR=分支仅仅在有与硬件相关的错误时使用，END=分支仅仅是在文件条件结束时使用。

如果这些选择都省略，与硬件相关的错误和文件的条件结束都将产生致命性错误，这样会终止程序的执行。

5.1.1 READ（非格式化随机输入）

输入语句（READ）将按整型值n输入记录，并赋数据到I/O列表K中的单元。

如果I/O表包含的元素和输入记录中的字段数一样，则整个记录被读入，如果I/O列表包含的元素少于输入记录中的字段数，未读入的项将被跳过。

如果I/O表中包含的元素大于输入记录中的字段数，同一记录将会读入多次以便填满I/O列表。例：

READ（6，REC=3）I，J，K

从逻辑单元号6相关的文件中读记录号3的记录，将输入数据赋给整型变量I、J、K（注：将读入6个字节——3个整型变量乘以每个整型变量2个字节）。

5.1.2 WRITE（非格式化随机输出）

输出语句（WRITE）是将I/O列表K所指定的值输出到由整型变量n所对应的记录中。如果上述的记录号n存在，它将完成写输出。如果上述记录号n不存在，文件将扩展以建立一个记录号。

数据输出的总数取决于I/O列表K中元素的类型和数量。如果数据输出的总数小于128字节，记录将用ASCII NUL空字符〔0〕填满，使之正好等于128字节。

如果它是大于128字节的输出，原来的128字节将被写完，直到I/O列表中所有的元素都输出为止，这样将产生一部分或所有数据被破坏。

程序员的责任是保证输出不得超过128字节，数据输出的总数是I/O列表中元素的数量和类型的一个函数。例：

WRITE（6，REC=4）J，K，A

写记录号4的记录到逻辑单元号6所相关的文件中。注意128个字节将被写入，但是只有前8个字节包含数据（2个整型变量乘2字节/每个整型变量加1个实型变量乘4字节/每个实型变量）。剩下的120个字节将是ASCII NUL字符〔0〕。

5.2 格式化随机输入与输出

格式化随机I/O用作在随机存取设备中直接存取字符数据。在READ或WRITE语句中允许存取文件的第n个记录，这是通过REC=n的选择项来实现的。实际上，这是传送一个物理记录（对格式化文件来说，一个物理记录为127字节）。

一个格式化语句在传送期间被指定作为控制数据的编辑和数据格式处理。

我们建议用'+'这个字符来代替回车控制字符。如果用了'+'字符，就无须在记录中存储回车字符。有关回车控制字符的详细讨论参见第8章。

格式化随机I/O有两种形式：

READ（u，f，REC=n，ERR=L_1，END=L_2）K（输入）

WRITE (u, f, REC=n, ERR=L_1, END=L_2) K (输出)

这里：

u　　逻辑单元号

f　　FORMAT 语句标号

n　　存取记录号

L_1　　I/O 错误分支（可选）

L_2　　EOF 分支（可选）

K　　一个 I/O 列表

如果指定记录号的整型变量是负或者是 0，将引起运行时 I/O 错误。

逻辑单元号必须指定一个磁盘文件，若对一个没有随机存取能力的设备进行随机存取，将引起运行时 I/O 错误。

ERR= 和 END= 语句的使用是可选择的。ERR= 分支标号仅仅是在与硬件相关的错误发生时才采用，END= 分支标号仅仅是在文件条件结束时才采用。

如这些选择省略，则有与硬件相关的错误和文件条件结束时都会产生致命性 I/O 错误。

5.2.1　READ（格式化随机输入）

输入语句（READ）将输入的整型变量 n 所对应的记录输入，将数据赋给 I/O 列表 K 中的元素，数据按标号为 f 所指定的 FORMAT 语句的要求进行编辑。

如果 I/O 列表中的元素与输入记录中的字段数一致，则整个记录读入，如果 I/O 列表中的元素少于输入记录字段数，则未读入的项就被跳过。

如果 I/O 列表中的元素大于输入记录中的字段数，则输入记录会读入多次，以填满 I/O 列表。

特别要注意的是，格式化随机读语句输入可以小于127字节。虽然 CP/M 扇区包括128字节，但由于格式化随机 I/O 只有127字节是有效的，因此跟在数据后面的字节由 FORTRAN I/O 处理器来处理，通常安排一个换行字符（ASCII 0A）。

在格式化随机读语句执行时，FORTRAN I/O 处理器必须寻找这个换行字符。如果处理器没有找到这个字符，则会引起运行时 I/O错误。

由于这个原因，只有由格式化随机读语句可以读出的文件，才能用一个格式化随机写语句来建立。试图读一个没有用格式化随机写来建立的文件，通常也会引起运行时的 I/O 错误。例：

```
      READ (6, 100, REC= 2) A, B, C
        ⋮
100 FORMAT (3A4)
```

它是从逻辑单元 6 读 2 号记录并赋输入值给变量 A、B、C。

5.2.2　WRITE（格式化随机输出）

输出语句（WRITE）是将 I/O 列表 K 所对应的值输出给整型变量 n 所对应的记录。如果上述记录号 n 存在，它将完成写；如果不存在，文件将被扩展并建立一个记录号。

传送数据的总数，取决于 I/O 列表 K 中元素的数量和类型。如果这个总数小于 127 字节，记录将用 ASCII NUL 字符〔0〕补满，使之正好是127字节，最后的数据字节是一个回车字符。FORTRAN 处理器在随后读文件时产生这个字符。

注意：程序员的责任就是保持只能是127个字节输出。这127个字节的记录也包括回车控制字符，如果一个'+'号被用作回车控制，用户就可以在记录中存取全部的127字节。

如果回车控制字符省略，I/O 处理器将产生一个 ASCII 换行字符（HEX-OA），输出文件的第一个字节就包含这个字符，这样可用字符只剩下 126 个字节。

如果要写的字节数大于127字节，FORTRAN I/O 处理器则不能产生标志记录结束的换行字符，在写语句执行时，虽然没有显示出错，但是随后对这个文件的 READ 操作时，会产生运行时的 I/O 错误。例：

```
      DIMENSION J (120)
        ⋮
       I= 6
      WRITE (6, 100,REC=I) J
  100 FORMAT ('+', 60A 2)
```

这个语句将60个元素的数组 J 的内容写入与逻辑单元号 6 相关的文件中的第 6 个记录中。记录中没有存回车控制字符，用这个语句将会写120个字节，下一个字节存放 FORTRAN I/O 处理器产生的换行字符，余下的 7 个字符是 ASCII NUL〔0〕字符。

第六节　辅助输入与输出语句

FORTRAN 提供了几种辅助的 I/O 语句，它们用来完成各种各样的文件管理功能。

6.1 OPEN（打开）子程序

一个文件可以用 OPEN 子程序打开，LUN_s1-5也可用 OPEN 语句分配给磁盘文件。

OPEN 子程序允许程序指定一个文件名和设备，而这些设备和 LUN 有关。

打开一个不存在的文件，将建立一个适当名字的空文件。跟在某顺序输出后的一个存在文件的 OPEN 操作将删除这个存在的文件，跟在某输入后的一个存在文件的 OPEN 操作可允许对当前文件内容进行存取。OPEN 子程序的格式为：

```
CALL OPEN (u, filename,driver)
```

这里 u 是与文件相对应的逻辑单元号，u 必须是一个无符号的整型常数或变量。它们的范围在 1～10 之间。如果使用整型变量，其值必须在调用 OPEN 子程序之前就要赋给变量。

"filename" 是一个 ASCII 名，CP/M 将它与文件联系起来。文件名应当是一个何勒内斯（Hollerith）常数或字母常数，或者是一个变量，或者是一个 ASCII 文件名的数组名。文件名必须是 CP/M 确认的文件名，而且必须实际填满11个空格位置。如果有扩展名，则其必须占用最后的三个位置。如果文件名的内容少于11个字符，则需用空格字符填满剩余的位置。

'driver' 是文件所在的驱动器号，这个号必须是整型常数或变量，且它们是在操作系统允许的范围之内。如果指定的驱动器是 0，那么可假定当前要选择的驱动器：1 是驱动器A，2 是驱动器 B 等等。例：

```
      CALL OPEN (7,'DATA       DAT', 0)
      CALL OPEN (3,'DATAFILE        ',2)
```

6.2 ENDFILE（文件结束）语句

ENDFILE 作为文件结束标志，然后关闭与 LUNu 相关的文件。ENDFILE 语句的格式是：

```
ENDFILE  u
```

6.3 REWIND（反绕）语句

REWIND 语句是关闭与 LUN u 相关的文件，然后再打开它。REWIND 语句的格式是：

```
REWIND  u
```

6.4 ENCODE/DECODE（编码与译码）语句

ENCODE 和 DECODE 语句是从存储器的一个部分传送数据到另一个部分。其格式按照格式语句规定。DECODE 是将内部格式数据变换到指定的数据格式，ENCODE 是将指定的数据格式变换为内部数据格式。两种语句的格式是：

```
ENCODE (a, f) K
DECODE (a, f) K
```

其中：

a　　一个数组名

f　　FORMAT 语句标号

K　　一个 I/O 列表

DECODE 模拟一个 READ 语句。它根据格式语句规定在数组 a 中产生字符数据的变换，赋给 I/O 列表 K 的元素，因此格式化的处理是由语句标号 f 来引用的。

ENCODE 模拟一个 WRITE 语句。它将 I/O 列表 K 中的元素转换成字符的格式，并存储于数组 a 中，因此由 f 指定的 FORMAT 语句将控制这个转换过程。

可以由 ENCODE 或 DECODE 语句处理的字符总量是由数组 a 的数据类型所确定的，下表列出了这个关系：

数据类型	每个矩阵元素的字符数
LOGICAL	1
INTEGER	2
EXTENDED INTEGER	4
REAL	4
DOUBLE PRECISION	8

应当注意的是：数组 A 应是足够大的，以致它可以包括对全部数据的处理。由于溢出没有检查，当 ENCODE 操作时，如超出数组容量，则可能擦掉跟在数组后面的重要数据；而当DECODE 操作时，如超出数组容量，则将数组后面的数据进行处理。

第八章　FORMAT（格式）语句

第一节　综　　述

FORMAT 语句是一不可执行语句，它用来同格式化 I/O 语句和 ENCODE及DECODE 语句一起使用。它们确定数据变换的方式和数据编辑时所需的信息。FORMAT 语句要求有指定的标号。通常FORMAT 语句的格式如下：

m　FORMAT (S_1, S_2, …Sn)

其中：

m 是语句标号，而每一个 Si 都是一字段标识符，且语句中必须有 'FORMAT' 和括号。

第二节　字段描述符

字段描述符描述数据字段的大小和指定各种数据传送时实际的变换方式。FORMAT字段描述符可以为下述的任何一种：

描述符	分类
rEW.d	数字转换
rFW.d	
rGW.d	
DW.d	
rIW	
rAW	字串转换
nHh_1, h_2, …h_n	
I_1, I_2, …I_n	
rLW	逻辑转换
nX	空格说明
mP	比例因子

其中：

W　在外部数据表达式中定义字段宽度是一个正的整型常数（包括数字、十进制小数点、代数符号）

d　在外部数据表达式中是一个整数，指定小数的位数

F、G、E、D、I、A 和 L　表示用于 I/O 列表中项的类型

r　是一种选择，它为非零的整数，表示标识符重复的次数

hi 和 Ii　是 FORTRAN 字符集中的字符

m　整型常数（正、负或 0），表示比例因子

　整型常数，定义插入记录中的空格数

第三节　数字转换

任何一种数字转换的输入操作，其数据允许按自由格式来表示。即在外部表达式中，可用逗号来分隔字段。

3.1　F 方式转换

实数或双精度型数据采用这种转换来处理。处理 W 字符时，将字符 D 看作为小数部分来处理。

格式：FW.d

3.1.1　F-输入

在 F 转换下，处理的数据值格式相对来说是不够严谨的。其格式如下：

1．打头的空格字符（忽略）
2．'+'或'-'符号（对无符号的输入假设为正）
3．一个字串
4．一个十进制小数点
5．第二个数字串
6．字母 E（幂指示符）
7．'+'或'-'符号
8．整数的幂

其格式必须遵守下面的条件：

如果有整数幂存在，则必须包含幂指示符和符号（'+'或'-'）。如果符号省略，则认为是正。

所有非打头的空格字符都认为是 0 。

输入数字可以是任意长，其大小以后讨论。但对第三章中有关实数的"数据表示法/存储格式"中的扩充规定来说还应保持其实数的精度。

F 输入举例：

格式	输入值	内部值
F 8.5	234562341	234.56234
F 6.2	12.123	12.123
F 9.2	89.56 E + 3	89560.0
F 5.2	1234567.89	123.45
F 8.5	- 12345678	- 12.34567

注意：在上例中，如果在输入字符之间没有十进制小数点，那么在FORMAT 引用时，D前会加上小数点；如果在输入字符中已经包括了十进制小数点，则 d 的引用将不起作用。

3.1.2　F-输出

将值进行转换，同时输出。如果数为负，则一个减号跟着的是数的整数部分、十进制小数点和数的小数部分 d。

如值不满规定的长度，则进行右调整，在字段前面插入足够的空格以填满字段。

如值比 W 允许的字段宽度还要宽，则会引起运行时的错误。

F-输出的格式：

格式	内部值	输出（b=空格）
F 10.4	368.42	bb 368.4200
F 7.1	-4786.361	-4786.4
F 8.4	8.7 E-2	bbb.0870
F 6.4	4739.76	**FW**

3.2 E 方式转换

格式： EW.d

实数或双精度类型的数据，用这种转换进行处理。处理 W 字符时，D 被看作小数进行处理，数据是按指数形式。

3.2.1 E-输入

进行 E 转换处理数据值时，其编辑类同于 F 转换的方法。

3.2.2 E-输出

值被转换，对小数 d 进行四舍五入，其输出格式为：

1. 减号（为负时）
2. 十进制小数点
3. 十进制数字
4. 字母 E（幂指示符）
5. 幂符号（负或正）
6. 两位指数数字

如上所述，如有必要，可进行右调整，在 W 字段前加空格字符以填满这个字段长度。字段宽度 W 应满足下面关系：

$$W >= d + 7$$

否则会引起运行时的错误。

E-输出举例：

格式	内部值	输出（b=空格）
E 12.5	76.573	bb.76573 E+02
E 14.7	-32672.354	bb-32672354 E+05
E 7.3	56.93	**FW**
E 13.4	-0.0012321	bbb-1232 E-02
E 9.2	76321.73	bb.76 E+05

3.3 D 方式转换

格式： DW.d

除转换输入数据和赋给一个双精度数据类型外，D-输入方式与 E-输入方式是一样的。

3.3.1 D-输入举例：

格式	输入值	内部值
D 10.2	23456bbbbb	23456000.0D0
D 10.2	bb234.56bb	234.56 D0
D 15.3	123.5678901 D+04	1.235678901 D+06

3.3.2 D-输出

同样，D-输出除 E 的幂指示符处换以 D 的指示符外，其方式与 E-输出一样。

D-输出举例：

格式	输入值	输出值（b=空格）
D 12.5	76.573	bb.76573D+02
D 14.7	-32672.354	-.32672354 D+05
D 13.2	-0.0012321	bb-.12321 D-02
D 8.2	76321.73	b.76D+05

3.4 G 方式转换

格式： GW.d

实数或双精度类型的数据用这种转换处理。处理 W 字符时，将d 看作小数处理。

3.4.1 G-输入

如 F 标识符一样，G 标识符以同样方式编辑输入数据。

3.4.2 G-输出

输出转换方法取决于输出数的大小，当事先不知道数的大小时，这种方式是有用的。

下表列出了数的输出方式，其中 n 是数的大小：

大 小	等值转换（b=空格）
$n<0.1$	EW.d
$1<=n<1$	F(W-4).d bbbb
$1<=n<10$	F(W-4).(d-1)bbbb
⋮	⋮
$10^{d-2}<=n<10^{d-1}$	F(W-4).1 bbbb
$10^{d-1}<=n<10^{d}$	F(W-4).0 bbbb
$n>10^{d}$	EW.d

G-输出举例：

格 式	内部值	输出(b=空格)
G13.6	0.01234567	bb.1234567E-01
G13.6	123.45678901	bb123.457 bbbb
G13.6	123456.7890	bb123457.bbbb
G13.6	-1234567.89012345	b-.123457E+07

作为比较，下面列出了在 F-输出时同一值的输出：

格 式	内部值	输出(b=空格)
F13.6	0.01234567	bbbb.0123456
F13.6	123.45678901	bbb123.456789
F13.6	123456.7890	123456.789000
F13.6	-1234567.89012345	**FW**

注意：在最后一个例子中，F 格式的标识字段相对数据来说太小了。在这种情况下，将引起运行的错误。

3.5 I 方式转换

格式：　　IW

这个标识符被指定为整数数据的传送。

3.5.1　I-输入

输入一个宽度为W的字段并转换为内部整型格式，减号可位于整型数前面。如果没有符号，则该值认为是正。整型值范围取为：－32768～32768。

I-输入举例：

格　式	输入(b＝空格)	内部值
14	b124	124
14	－124	－124
17	bbb732b	7320

3.5.2　I-输出

转换值为整型常数，为负值时可在数前加减号。如果值未填满字段，则应进行右调整，插入空格字符在值的前面以填满整个字段。如果值超过字段长度，则引起运行时的错误。

I-输出举例：

格　式	输入值	输出(b是空格)
16	＋281	bbb281
16	－23261	－23261
13	126	126
14	－226	－226
13	1234	**FW**

第四节　文字型转换

4.1　A方式转换

A转换格式如下：

格式：　AW

A描述符的功能是从一个指定的列表项中读入未修改的字串或把它写到该列表项中。

利用AW，实际可以传递的字符最大数等于存放相应列表项需要的字节数（即逻辑项是1个字符，整型项是2个字符，实数项是4个字符，而双精度项是8个字符)。有关各种数据类型存放的要求，参考第三章“数据表示法/存储格式”一节。

4.1.1　A-输入

如果W大于n（这里n是对应列表项存储时所需字节数)，那么最右边的字符n是从外部输入的字段中取得的。

如果W小于n，则W 字符出现在左边，在内部表示形式中要经过调整，在 W 之后要填加W-n个空格字符。

A-输入举例：

格式	输入	类型	内部值(b是空格)
A_1	A	整型	Ab
A_4	ABCD	整型	AB

A_1	A	实型	Abbbb
A_7	ABCDEFG	实型	DEFG

4.1.2 A-输出

如果W大于 n，则外部输出字段是由W-n个空格字符加在从内部表达式输出的几个字符的后面组成；如果W小于 n，则由从内部表达式来的最左边的字符组成。

A-输出举例：

格 式	内 部 值	类 型	输 出
A_1	AZ	整型	A
A_2	AB	整型	AB
A_3	ABCD	实型	ABC

4.2 H方式转换

H 转换的格式如下：

$nHh_1,h_2,\cdots\cdots h_n$

$'h_1,h_2,\cdots\cdots h_n'$

这些标识符是在标识符与外部字段之间处理 Hollerith 文字字符串。这里，各个 h 表示ASCII字符集合中的任何一个字符。

在第二种格式中，如果采用〔'〕，就需要做特殊考虑。在字串中的一个〔'〕字符是由两个连续的撇号〔'〕表示，见下面的例子。

4.2.1 H-输入

字串hi的 n 个字符将会被输入记录中最后的 n 个字符所取代。在字段标识符中产生一个新的字符串。详见第七章的举例。

H-输入举例：

格 式	输入(b是空格)	结果(b是空格)
4H1234	ABCD	4HABCD
7HbbFALSE	TRUEbbb	7HTRUEbbb

4.2.2 H-输出

n 个字符的hi字串放在外部的字段中，在$nHh_1h_2\cdots h_n$的形式中，串中的字符个数实质上必须由 n 来指定；否则，从其他标识符来的字符将成为这个字符串的一部分。

在两种格式中，空格字符被当作字符一样计算。

H-输出举例：

格式(b是空格)	输出(b是空格)
1HA或'A'	A
8HbSTRINGb或'bSTRINGb'	bSTRINGb
11Hx(2,3)=12.0或'x(2,3)=12.0'	x(2.3)=12.0
12HIbSHOULDN'T或'IbSHOULDN''T'	IbSHOULDN'T

第五节 逻辑型转换

逻辑转换的格式如下：

格式： LW

5.1 L型输入

外部表达式占用W的宽度，它是由可选的空格字符，后跟一个“T”或“F”组成，紧接着是可选择的字符。

5.2 L型输出

如果对应这个标识符的输出表中的一个项值为0，则输出“F”,否则输出“T”。如果W大于1，那么W－1个打头的空格字符就加在字母的前面。

L-输出举例：

格 式	内部值	输出(b是空格)
L_1	<>0	T
L_5	<>	bbbbT
L_7	=0	bbbbbbF

第六节 X描述符

x 描述符的格式如下：

格式：nx

这个描述符不产生转换，也不与I/O表中的项相对应。当输出使用它时，将引起几个空格字符插入到输出记录中。在输入情况下，这个描述符将跳过输入记录中的最后几个字符。

x-输入举例：

格 式	输 入	内 部 值
10 FORMAT(F4.1,3x,F3.0)	12.5ABC120	12.5,120

x-输出举例：

格 式	输 出
3 FORMAT(1HA,4x,2HBC)	A bbbb BC

第七节 比例因子

P 标识符用来指定实数转换时的比例因子。

格式： np

其中，n 是整型常数（正、负或零）。

比例因子的指定必须放在使用它的字段标识符之前。一旦比例因子确定，它应适用于在FORMAT 语句中碰到的所有实数变换。比例因子一直保持不变，除非碰到 P 标识符或I/O终止。我们可利用指定一个 0 比例因子来使它不起作用。

7.1 比例因子在输入时的作用

在输入时，比例因子产生下面的作用：

赋给I/O表元素的值 = 外部值/10**n

如果在外部数据段有一指数出现，那么比例因子将不起作用。例：

格 式	输 入	内 部 值

2PF9.5	bbb45.123	.45123
-2PF9.5	bbb45.123	4512.3

7.2 比例因子在输出时的作用

E-输出，D-输出

系数相对于十进制小数左移n位，指数则降低n倍（保持其值不变）。

F-输出

外部值等于10**n 乘以内部值。

G-输出

如果采用F-转换，则内部值在输出范围时，比例因子不起作用。例：

格 式	内 部 值	输出(b是空格)
1PE12.5	34.567	b3.45670E+01
2PF9.3	12.345	b1234.500
3PG9.3	10.00	b10.0bb

第八节 格式语句中的其他控制特性

8.1 重复说明

E、F、D、G、I、L、X和A 字符段标识符可以重复计数，r 表示重复标识符。其格式如下：

rEW.d, rFW.d, rGW.d, rIW, rLW, rX和rAW

例：

语句：600 FORMAT(A2, A_2, I_2, I_2, I_2, F6.2, F6.2)

等于：600 FORMAT(2A2, 3I_2, 2F6.2)

一组字段标识符的重复，相当于在括弧中这个组的标识符，这组标识符的前面总是重复计数值。如果没有重复计数符，则表示一次计数。

包括 FORMAT 语句所要求的括弧在内，两级以上的括弧是允许的。例：

句子：600 FORMAT(2(4x, I_2, F5.2), A_4)

等于：600 FORMAT(4x, I_2, F5.2, 4x, I_2, F5.2, A_4)

当在 FORMAT 语句中所有标识符都已经使用，但在I/O列表中的项还没有处理完时，FORMAT 标识符就得初始化。

这种情况发生时，在FORMAT 语句中，FORMAT 标识符在第一次打开括弧时开始重新使用。

FORMAT 标识符这种重复使用的方式用来终止当前记录处理，并且在每次重新开始使用时，开始新的记录处理。

8.2 字段分隔符

在FORMAT语句中两个相邻的标识符必须用一个逗号或一个或多个斜线分开。例：

2A4, F6.3或2A4/F6.3

在FORMAT 语句中用逗号简单地分隔字段，斜线不仅分隔字段标识符，还指定格式记录的分界。

每一个斜线结束一个记录，并且开始下一个要处理的记录。如输入时碰到斜线，输入记

录的剩余部分就省略；如输出时碰到斜线，则输出记录的剩余部分用 ASCII NUL 字符填满，并且开始下一个输出记录的处理。

连续的斜线将导致在输入时连续的记录被省略掉；而在输出时，则写出连续的空记录。

8.3 格式回车控制

每一个格式化输出记录的第一个字符用来送回车信息到输出设备上。在向 CRT 终端或一个硬拷贝设备输出时，这是一个很有用的特性。

回车控制字符确定了在这行打印之前会作什么样的动作。回车控制字符在 FORMAT语句中应是第一个文字段，并用作为格式化的输出。

如果没有回车控制字符，则 FORTRAN 语句的第一个字符将作为回车控制。其选择如下：

字 符	作 用	ASCII代码(十六进制)
0	跳过 2 行	OA, OA
1	插入格式馈送	OC
+	不起作用	(无)
其他	跳过 1 行	OA

如果格式化输出是完成一个磁盘文件的输出，那么对回车控制字符应作特殊考虑。磁盘文件的第一个字段将由FORMAT 语句中的回车控制字符来确定。如果字符“+”号被采用作为回车控制字符，回车控制字符就不存储在磁盘文件中。例：

```
FORMAT('+',12A2)
```

在FORMAT 语句中，当输出数据到一个磁盘文件时，回车控制字符是非常有用的，它将去掉任何一个正在存储的记录中的回车控制字符。

```
FORMAT ('1'6I1, 2x, I2)
```

在这个FORMAT语句中，回车控制字符将在这行打印前发出一个格式馈送。

8.4 数组中的格式说明

在格式化 READ 或 WRITE 语句中，FORMAT说明 f 可以用一个数组名来代替语句标号，这就提供了在执行目的程序时，更换一个FORMAT说明的方便条件。

如有这样的指定，则在数组中所含顺序排列的信息就必须构成有效的 FORMAT说明。

插入在数组中的 FORMAT 说明，具有 FORMAT 语句定义同样的性质(即它用一个左括弧开始，而用一个右括弧作结束)。

FORMAT 说明可以利用一个 DATA 初始化语句插入在数组中，也可以利用一个格式 READ 语句连同一个AW FORMAT 操作来插入到数组中。例如假定 FORMAT 说明为：

```
(3F10.3, 6I6)
```

或者说是存放到数组中的一个类似于12个字符说明，因而数组必须允许最少存放 12 字节。FORMAT 可以被存放在一个数组中，格式如下：

```
DATA A/'(3F1', '0.3','4I6)'/
```

下面语句将利用数组中的这种格式来读。

```
READ(6,A)B, C, D, I1, I2, I3, I4, I5, I6
```

注意：数组名 A 指定来代替一个语句标号。

第九章　说明语句

第一节　综　　述

说明语句是不可执行语句，它提供给 FORTRAN 编译以确定的信息，这个信息用来定义变量和数组的数据类型、指定数组的维数和大小以及分配数据存放的单元。说明语句有：

1. PROGRAM 语句
2. TYPE语句
3. EXTERNAL 语句
4. DIMENSION 语句
5. COMMON 语句
6. EQUIVALENCE 语句
7. DATA 初始化语句

所有的说明语句在一个程序单元的开始分成组，且必须按上面的次序排列。PROGRAM 语句必须是主程序体的第一条语句，并且在其他任何程序体中都不出现。

其他的说明语句可以紧靠在 FUNCTION、SUBROUTINE或BLOCK DATA 语句之前，所有的说明语句必须在语句函数和第一条执行语句的前面。

第二节　数组说明

三种说明语句可以指定数组的说明，这些语句是：

DIMENSION 语句

TYPE语句

COMMON 语句

在这些语句中， DIMENSION 语句仅仅具有数组宣称的功能，其他两种语句提供了双重的用途：

数组说明符用来确定数组的名字、维数的大小，一个数组在一程序体中仅被说明一次。数组说明有下列格式：

Ui(K)

Ui(K_1, K_2)

Ui(K_1, K_2, K_3)

Ui是数组名，也称为说明符名，而 Ki 是整型常数。数组存储单元的分配是在数组说明符出现时建立的，数组存储单元分配是由 FORTRAN 编译程序线性地分配，这里优先顺序是由下标决定的。第一个下标最早安排，最后面的下标则最后安排。

举例：如数组说明符AMAT(3,2,2)出现，则12个存储单元按下述顺序进行安排：

AMAT (1,1,1)

```
AMAT (2,1,1)
AMAT (3,1,1)
AMAT (1,2,1)
AMAT (2,2,1)
AMAT (3,2,1)
AMAT (1,1,2)
AMAT (2,1,2)
AMAT (3,1,2)
AMAT (1,2,2)
AMAT (2,2,2)
AMAT (3,2,2)
```

第三节 语 句

3.1 PROGRAM 语句

PROGRAM 语句为主程序体提供了一个确定名字的方法，它的格式为：

```
PROGRAM    n
```

其中，n 为程序名。

如果有程序语句，则程序语句在主程序体中，必须在其他任何语句之前。这个名字是由1到6个字母数字组成，且第一个字符必须是字母。如果在主程序中，没有 PROGRAM 语句，编译程序将赋一个 0MAIN 名给这个程序。在编译过程中，程序名将显示在控制台设备上。

3.2 TYPE(类型)语句

变量、数组以及函数名将由预先定义的类型自动地赋予整型或实型，除非它们被类型语句所改变。

例如，如果一个项的第一个字母是 I、J、K、L、M，或 N，则类型是一个整型数，否则是实型数。TYPE语句用作指定一个项的类型是有效或是无效，此外这些语句还可以用来说明数组。

TYPE语句通常有以下的格式：

$$t\ V_1,\ V_2,\ \cdots\cdots V_n$$

这里 t 表示以下项中的一个：

BYTE

INTEGER, INTEGER*1, INTEGER*2, INTEGER*4

REAL, REAL*4, REAL*8

LOGICAL, LOGICAL*1, LOGICAL*2

DOUBLE PRECISION

每一个 V 是一个数组的说明符或一个变量、数组或FUNCTION名。

下面说明这些关系：

1. BYTE, INTEGER*1, LOGICAL*1和LOGICAL　　字节数相同

2. INTEGER*2, LOGICAL*2, 和INTEGER　　字节数相同

3. REAL, INTEGER　　字节数相同

4. 双精度和REAL　　字节数相同

举例：

```
BYTE BUFF (256)
REAL IN, IOUT
DOUBLE PRECISION DPARG
```

3.3 EXTERNAL(外部)语句

这个语句允许外部过程名作为其他子程序的参变量来使用。外部程序名可能是 SUBROUTINE、BLOCK、DATA 或 FUNCTION 名。这种语句的一般格式为：

EXTERNAL U_1, U_2, ……, U_n

每一个 Ui 是SUBROUTINE、BLOCK、DATA或FUNCTION的名字。

EXTERNAL 语句允许表 Ui 中的任何一个名在调用子程序时作为一个参变量。

当将一 BLOCK DATA 子程序看作为其他子程序的参变量时，它的名字在主程序体内必须出现在一个 EXTERNAL 语句中。

举例：如 SUM 和 AFUNC 是子程序名，并被用作为子程序 SUBR 中的参变量，那么下面的语句在调用的程序体中将会出现：

```
      ⋮
EXTERAL SUM, AFUNC
      ⋮
CALL SUBR (SUM, AFUNC, X, Y)
```

3.4 DIMENSION(维数)语句

定维语句是一个不可执行语句。它是为数组的存储服务的，此外它还定义数组中的维数和元素的数目。数组元素由数组名后跟一个下标来说明。语句的一般格式为：

DIMENSION U_1, U_2, U_3

每一个 Ui 都是数组的说明符。例：

```
DIMENSION RAT (5,5),BAR (20)
```

这个语句说明两个数组——25个元素的 RAT 数组和20个元素的 BAR 数组（数组和数组存储分配见数组说明符的讨论）。

3.5 COMMON (公用区)语句

COMMON 语句是一个不可执行语句，也是存储分配的语句。它赋变量和数组到一个存储空间，这个空间称为COMMON 存储区，并为各种各样的程序体共享同一个存储空间提供方便。它们的一般格式为：

COMMON/cb/nlist/cb/nlist/…/cb/nlist

这里每一个 cb 是 COMMON 块存储的名字，而每一个 nlist 是一系列用逗号分隔的变量名、数组名或常数数组说明符。在 nlist 中的元素构成了 COMMON 块存储空间范围，这个范围由 cb 来指定。

COMMON块名是1～6个字母数字组成，第一个字符必须是字母。COMMON 块的名字可能在同一个COMMON语句中多次出现，或在多个COMMON语句中出现。 COMMON块

名不同于其他任何一个由程序调用的子程序名。

如果任意一个 nlist 被省略，而剩下两个连续的斜线字符（//），这样表示的存储块称为 blank COMMON。如果第一块被省略掉，则头两个斜线也可以省掉。

COMMON 空间长度为包含变量和数组所需的存储单元数，而这些变量和数组是在COMMON语句中说明的。

同名所需的COMMON块的长度，在所有包含这个名字的程序体中都是不相同的。如果长度不一样，指定最大长度的程序体必须最先装入(见 Link-80手册)。

例：COMMON/AREA/A, B, C/BDATA/X, Y, Z, FL, ZAP(30)

在这个例子中，两个COMMON 存储块被分配——AREA 为三个变量的空间，BDATA 为四个变量的空间和有30个元素的数组 ZAP 。例：

COMMON//A_1, B_1/CDATA/ZOT (3, 3) //T_2, T_3

在这个例子中，A_1、B_1、T_2、T_3被赋值给空的COMMON块，因此ZOT (3,3) 是数组说明符，ZOT不必事先说明。

3.6 EQUIVALENCE (等价)语句

EQUIVALENCE 语句允许两个或更多的实体共享同样的存储单元。

按顺序排列的每一个元素由编译程序赋给同一个存储单元。这样同一个存储单元可由不同的变量来确定。

元素是否按顺序出现是不重要的。这个语句的一般格式为：

EQUIVALENCE (nlist), (nlist),… (nlist)

这里每个 nlist 都表示两个或两个以上用逗号隔开的变量或数组元素的一个序列。例：

EQUIVALENCE (A, B, C)

在执行目的程序时，变量A、B、C共享相同的存储单元。

如果一个 EQUIVALENCE 语句中使用一个数组元素，那么下标数必须与数组说明建立的维数相同。它也可以是 1 。这里下标说明指数组元素是数组中的第一个元素。

如果一个数组的维数 Z 已经说明为 Z (3,3) ，则在EQUIVALENCE 语句中 Z(6)和 Z (3,2) 具有同样的含义。数组元素的下标必须是整型常数。

3.6.1 构成数组的等值语句

如果一个数组的元素要与另一个数组的元素等值，则EQUIVALENCE 语句可建立对应数组元素间的等值关系。如果两个相同维数数组的第一个元素是等值的，那么两个数组将共享同一存储空间。

同一数组两个或两个以上元素的等价是无意义的。两个或两个以上的元素属于同一个或不同的COMMON块的等值性也是无效的。例：

```
DIMENSION A (7),B (3)
EQUIVALENCE (A(5),B(3))
```

这个EQUIVALENCE语句建立的等值语句如下：

矩阵 A		矩阵 B
A (1)		
A (2)		
A (3)	=	B (1)

A (4) = B (2)
A (5) = B (3)
A (6)
A (7)

3.6.2 等值语句与公用语句的相互影响

变量可以经过一个EQUIVALENCE 语句赋给 COMMON 块，EQUIVALENCE 可以附加更多的元素到块的尾部从而增加 COMMON 块的尺寸。COMMON 块的尺寸可以用 COMMON 语句建立，而元素的增加是从块中最后一个元素处向前增加，而不可从它的第一个元素处反方向增加。例：

```
DIMENSION Z (4),X (6)
COMMON Z
EQUIVALENCE (Z(2),X(1))
```

Z(1)	Z(2)	Z(3)	Z(4)			
	X(1)	X(2)	X(3)	X(4)	X(5)	X(6)

现有的 COMMON　　扩展的部分

3.7 DATA（数据初始化）语句

数据初始化语句是一个不可执行语句，它提供了初始化变量和数组元素的一种方法。语句的格式如下：

DATA list/u1, u2, …un/list…/uk, uk+1 ……uk+n/

这里 list 是表示变量、数组或数组元素名的列表，ui 是对应于表中元素的量和类型的常数。

这里除了常数与列表一一对应之外，数组名（无下标）可在列表中出现；为了填满数组所需的常数，还可在斜线间对应的位置出现。为了说明连续的同一常数和 ui，可以用K*ui来代替。这里 K 必须是一个正整数。例：

```
DIMENSION (7)
DATA A, B, C (1), C (3)/14, 73, -8.1, 2*7.5/
```

这意味着：

A=14.73, B=-8.1, C (1)=7.5, C (3)=7.5

每一个常数 ui 的类型必须与列表中对应项的类型一致，但Hollerith或Literal（字串或文字）常数例外，它可以与任何类型的项配对。

当使用字串型或文字型常数时，其字串中的字符数必须不大于对应项所要求的字节数。即逻辑变量为一个字节，整型变量为 2 个字节，实型变量为 4 个字节，双精度为 8 个字节。

如果指定的字串或文字符较少，则其尾部应填加空格字符以填满存储区的剩余部分。

以类似的方式存储十六进制数。如果16进制数的字符较少，则用足够多的连续 0 来填满存储单元的剩余部分。

3.8 IMPLICIT（隐含）语句

IMPLICIT 语句用来重定义缺省变量的类型。语句格式如下：

IMPLICIT type (range), type (range),…

这里 type 是下列形式之一：INTEGER, REAL, LOGICAL, DOUBLE PRECISION, BYTE, INTEGER*1, INTEGER*2, INTEGER*4, REAL*4, REAL*8, LOGICAL *1, LOGICAL*2。而范围是字母字符构成的列表并用逗号或连字符（—）分开。例：

IMPLICIT INTEGER (A, W-Z), REAL (B-V)

用字母 A、W、X、Y、Z 开始的所有变量（没其他说明的）都是 INTEGER 类型。用字母 B 到 V 开始的所有变量都是实数型。

IMPLICIT INTEGER(I-N), REAL(A-H, O-Z)

这是缺省变量定义。

任何一个 IMPLICIT 语句必须在 TYPE 和 DIMENSION程序的分组语句中出现。IMPLICIT 语句必须在其他说明语句之前出现。如 IMPLICIT 语句在 TYPE和DIMENSION 语句之后，已经说明过的变量类型将不起作用。

3.9 INCLUDE（包含）语句

INCLUDE 语句使编译程序把外部 FORTRAN 的原代码引入当前程序。包含的代码可以是通常使用的子程序或像 COMMON语句一样的说明语句。语句的格式是：

INCLUDE filename

INCLUDE 可以改变在当前原文件中经常反复用到的语句序列的使用。

第十章　函数与子程序

第一节　综　　述

FORTRAN 语言提供了一种定义和使用过程的方法。这就是 所需的过程语句或语句组只在一个程序中出现一次。这些过程可以被指定，并常常将过程引入到程序的逻辑执行序列中。这些过程有：

1. 语句函数
2. 库函数
3. FUNCTION 函数子程序
4. SUBROUTINE 子程序

每个过程都有它自己的独特要求。这一章将解释过程的构成和说明过程的各种要求。

在下面这些过程的介绍中，术语“Calling Program”的意思是调用其他过程的程序体或者过程，而术语“Called Program”的意思则是这个过程被别的过程所调用。

第二节　语句函数

语句函数是由一个简单的算术或逻辑赋值语句所定义的，并且只与函数所在的程序体有关。语句函数的一般格式为：

f (a1, a2, …an) = e

这里 f 是函数名，ai 是哑元变量， e 是算术或逻辑表达式。

如果它们在程序体中存在，那么语句函数必须在程序体及随后的所有说明语句、执行语句之前定义。

ai 是性质不同的变量名或数组元素，然而开头的哑元变量可以在程序体 中 的 其他地方具有相同的变量名。

表达式 e 按照第四章“FORTRAN 表达式”的规则构成，它可能只包含 引 用的哑元变量和非文字常数，或者引用的变量和数组元素，或者引用的实用程序和数字函数以及预先引用的已定义过的语句函数。

任何一个语句函数或参变量的类型若与原来规定的方式不同，则必须由一个类型语句来定义（见第九章“说明语句”中关于类型说明语句的讨论）。

一个语句函数的调用可用它的名字其后面紧跟一个括弧，括弧内是参变量的列表组成。调用中的表达式以引用的参变量来求值，其结果赋给所引用的变量。

在每个参变量列表中的第 i 个参数必须与语句函数中第 i 个哑元参变量在类型上一致。

下面的例子列出了一个语句函数及一个语句函数的调用：

```
C       STATMENT FUNCTION DEFINITION
C       FUNC 1 (A, B, C, D) = ( (A + B)**C)/D
```

```
      ⋮
C     STATMENT FUNCTION CALL
C     A12=A1-FUNC1 (X,Y,Z7, C7)
```

第三节　库函数

库函数分为实用程序和数字函数，它们是基于 FORTRAN 系统之上的。在表 10-1 和表10-2中列出了这些函数。在表中，如果需要多于一个以上的参变量，则参变量写作 a1，a2，…an；如果只需 1 个参变量，则写作 a。

一个库函数，当它的名字已用在一个算术表达式中而被调用时，采用下面的格式：

f (a1, a2, …an)

这里，f 是函数名，ai 是有效参变量。参变量必须与表10-1和表10-2中引用的类型和次序一致。

除了在表 10-1 和表 10-2 中所列的函数外，还有 4 种附加的库子程序，使之能对 8080（或 Z80）硬件进行存取。即：

PEEK, POKE, INP, OUT

PEEK 和 INP 是逻辑函数，POKE 和 OUT 是子程序。PEEK 和 POKE 允许对任何一个存储体进行存取，PEEK (a) 返回由 a 所指定的存储单元内容。

CALL POKE (a1, a2) 将会使由 a 1 所指定的存储单元的内容被 a 2 所指定的存储单元内容所替换。INP 和 OUT 允许对 I/O 口进行存取。INP (a) 完成从 a 口的输入，而返回一个 8 位的值。CALL OUT (a1, a2) 则将 a 2 的值输出到由 a 1 所确定的口去。例：

```
A1 = FLOAT (I7)
```

转换整数 I 7 为实数，并将结果赋给实型变量 A 1。

```
POS = ABS (R1)
```

赋实型变量 R 1 的绝对值给实型变量 POS。

```
S3 = SIN (T12)
```

计算 T12的 SiN 值，将结果赋给实型变量 S 3。

表10-1　　内在函数

函数名	定　义	参变量	函　数
ABS	求绝对值	实型	实型
IABS		整型	整型
DABS		双精度	双精度
AINT	a 乘以小于等于 |a| 的最大	实型	实型
INT	整数的符号	实型	整型

IDINT		双精度	整型
AMOD	返回值为第一自变量除以第二自变量	实型	实型
MOD	的余数	整型	整型
AMAXO	从列表元素中送回最大值	整型	实型
AMAXI		实型	实型
MAXO		整型	整型
MAXI		实型	整型
DMAXI		双精度	双精度
AMWO	返回参变量列表中最小值	整型	实型
AMINI		实型	实型
MINO		整型	整型
MINI		实型	整型
DMINI		双精度	双精度
FLOAT	整数型转换为实数型	整型	实型
IFIX	实型数转换为整型数	实型	整型
SIGN		实型	实型
ISIGN		整型	整型
DSIGN		双精度	双精度
DIM	求 a 1 - MIN (a1, a2)	实型	实型
IDIM		整型	整型
SNGL	双精度型转换为实数型	双精度	实型
DBLE	实数型转换为双精度型	实型	双精度

表10-2　　基本的外部函数

函数名	自变量数	定　义	自变量类型	函数类型
EXP	1	e**a	实型	实型
DEXP	1		双精度	双精度
ALOG	1	In (a)	实型	实型
DLOG	1		双精度	双精度
ALOG10	1	Log_{10} (a)	实型	实型
DLOG10	1		双精度	双精度
SIN	1	Sin (a)	实型	实型
DSIN	1		双精度	双精度
COS	1	cos (a)	实型	实型
DCOS	1		双精度	双精度
TANH	1	tanH (a)	实型	实型
SQRT	1	(a)**1/2	实型	实型
DSQRT	1		双精度	双精度
ATAN	1	arctan (a)	实型	实型
DATAN	1		双精度	双精度
ATAN2	2	arctan (a_1/a_2)	实型	实型
DATAN2	2		双精度	双精度
DMOD	2	a 1 (mod a_2)	双精度	双精度

第四节　函数子程序

以 FUNCTION 语句开始的程序单元称为函数子程序，它有如下格式的一种：

t　FUNCTION f (a1, a2, …an)

　FUNCTION f (a1, a2, …an)

这里 t 表示函数类型为下面类型的一种：INTEGER, REAL, DOUBLF PRECISION 或LOGICAL。如果没有 t 则像第二种格式那样，函数为双精度型。

f 是 FUNCTION 子程序的名字。

ai 是哑元变量，在这里至少要有一个，它用来表示变量名、数组名、子程序或其他功能的子程序的哑名。

FUNCTION 名的数据类型可用类型语句建立，在 FUNCTION 语句中显然是以类型语句开始的，或者是采用预先定义的数据类型。

在任何情况下，FUNCTION 语句中所定义的数据类型都必须与调用程序中使用的数据类型一致。

FUNCTION 语句必须是程序体的第一条语句。它不必包括语句标号。

FUNCTION 语句返回一个单值给调用程序，这个值的数据类型将由 FUNCTION 名的数据类型来决定。

4.1　构造 FUNCTION（函数）子程序

在 FUNCTION 子程序内，FUNCTION 名必须在赋值语句等号左边至少出现一次。FUNCTION 名也可以是输入语句 I/O 表中的一个元素，这样就定义了 FUNCTION 的值以便可以返回到调用程序。

此外可以通过给哑元变量赋值返回到调用程序。

哑元参变量列表中的名字，在 FUNCTION 子程序中不会在 EQUIVALENCE、COMMON 或 DATA 语句中出现。

如果哑元参变量是一个数组名，那么一个数组说明符必须出现在有定维语句的子程序中，而定维信息连同说明符一起又在调用子程序中。

FUNCTION 子程序可以包括任何已经定义过的 FORTRAN 语句，但不包括 BLOCK DATA 语句、SUBROUTINE 语句或者其他 FUNCTION 语句。FUNCTION子程序也不包括正在定义的 FUNCTION 的任何语句或正在定义的FUNCTION子程序中的任何语句。

FUNCTION 子程序的逻辑终止语句是 RETURN 语句，而且必须至少有一个。FONCTION 子程序在物理上必须用 END 语句结束。例：

```
FUNCTION SUM (ARRAY,  I)
DIMENSION  ARRAY (10)
SUM = 0.0
DO  100  K = 1 , I
100  SUM = SUM + ARRAY (K)
RETURN
END
```

以上程序段被用来构成 FUNCTION 子程序。这个子程序称为 SUM，数据类型没有说明，因此它遵守预先的约定。这个子程序是计算数组 ARRAY 的子程序。

4.2 引用 FUNCTION（函数）子程序

每当 FUNCTION 名字连同一个参变量列表一起被用作为操作数时，就要调用 FUNCTION 子程序。这种引用有下面的格式：

```
f (a1, a2, …an)
```

这里 f 是 FUNCTION 名，ai 是实参，圆弧号的格式如上所示。

参变量 ai 必须同被调用子程序中 FUNCTION 语句所引用的哑元变量的类型、顺序和数量一致。它们可以是下面的任何一种：

1. 变量名
2. 数组元素名
3. 数组名
4. 表达式
5. SUBROUTINE 或 FUNCTION 子程序名
6. Hollerith 或 Literal 字串或文字常数

如果一个参变量是一个子程序的名字，那么这个名字必须与事先在外部语句中出现过的原始变量名不同。此外，被调用的 FUNCTION 子程序中，对应的哑元变量必须在该子程序中引用。

如果参变量是一个字串或文字常数，则对应的哑元变量应包含足够的存储单元，它同常数所需的存储单元总数严格对应。

当 FUNCTION 子程序被调用时，程序控制就转到 FUNCTION 语句后第一条可执行语句去执行。

下面的例子说明了一个 FUNCTION 子程序的引用：

```
DIMENSION  ARRAY (10)
    ⋮
RESULT = SUM (ARRAY, 10)
```

这个程序引用前一节例举的子程序，且单值返回到调用程序，并赋给变量 RESULT 。

第五节 SUBROUTINE 子程序

用 SUBROUTINE 语句开始的程序体叫做 SUBROUTINE 子程序。SUBROUTINE 语句的格式如下：

```
SUBROUTINE  S (a1, a2, …an)
SUBROUTINE  S
```

这里 SUBROUTINE 是子程序名，ai 表示一个哑元变量，它代表一个变量或数组名以及其他任何 SUBROUTINE 或 FUNCTION 名。

SUBROUTINE 语句必须是子程序的第一条语句。SUBROUTINE 子程序名不可以在任何语句中出现，它只能在子程序开始时的 SUBROUTINE 语句中出现。哑元变量名不允许在子程序中的 EQUIVALENCE、COMMON 或 DATA 语句中出现。

如果哑元变量是一个数组名，则数组说明符必须在有定维语句的子程序中，并都在调用程序体中。

如果任一哑元变量代表了SUBROUTINE子程序确定的值，并返回调用程序，则这些哑元变量必须在赋值语句等号的左边，或者在一个输出语句列表中出现，或者在子程序中引用来作为一个内部的参数。

一个SUBROUTINE子程序可包括除下列语句之外的任何FORTRAN语句。

· BLOCK DATA 语句

· FUNCTION 语句

· 任何其他的SUBROUTINE语句

· PROGRAM 语句

SUBROUTINE语句可以包括任意数量的RETURN语句，且必须至少有一个。RETURN语句是子程序的逻辑终止点。SUBROUTINE子程序的物理终止是END语句。

如果由调用程序传递给SUBROUTINE子程序一个实参是SUBROUTINE或FUNCTION子程序的名字，那么对应的哑元变量必须用于SUBROUTINE子程序中。

SUBROUTINE子程序举例：

```
C  SUBROUTINE TO COUNT POSITIVF ELEMENTS
C  IN AN ARRAY
SUBROUTINE COUNTP (ARRAY, I, I  COUNT)
DIMENSION ARRAY (10)
I  COUNT = 0
DO  9  J= 1 , I
IF (ARRAY (J)) 9 , 5 , 5
9   CONTINUE
RETURN
5   I  COUNT = I  COUNT + 1
GO TO  9
END
```

5.1 引用一个SUBROUTINE（例常）子程序

SUBROUTINE子程序可以用一个CALL语句调用。CALL语句可用下列格式之一：

```
CALL  S (a1, a2, …an)
CALL  S
```

这里S是SUBROUTINE子程序名，ai是子程序使用的实参。

ai在子程序定义的SUBROUTINE语句中，应与相应的哑元变量类型、顺序和数量一致。参变量在CALL语句中必须遵守下面的规则：

· 在参变量表中出现的FUNCTION和SUBROUTINE名必须在EXTERNAL语句之前已经出现过。

· 如果被调用的SUBROUTINE子程序包括一个可变的数组说明符，那么CALL语句必须包括作为参变量使用的数组的实际名字和实际确定的维数。

· 如果在SUBROUTINE子程序哑元变量表中的项是一个数组，那么在CALL语句参

变量表中的相应项也必须是一个数组。

·当 SUBROUTINE 子程序被调用时，程序控制转到 SUBROUTINE语句后第一条执行语句去执行。

例：

```
    DIMENSION DATA (10)
        ⋮
C   THE STATMENT BELOW CALLS THE
C   SUBROUTINE CONSTRUCTION IN THE PREVIOUS
    SECTION CALL COUNTP (DATA, 10, CPOS)
```

第六节　从函数与例常子程序返回

FUNCTION 或 SUBROUTINE 子程序的逻辑终止是RETURN 语句，它传送控制块到调用程序。RETURN 语句的一般格式为：

RETURN

RETURN 语句遵循下面的规则：

·从一个 FUNCTION 子程序返回是返回到 FUNCTION 之后的调用程序指令序列。

·从一个 SUBROUTINE 子程序返回是返回到调用程序中的下一条可执行语句。此调用程序在逻辑上是在 CALL 语句之后。

·根据从 FUNCTION 子程序的返回，子程序的单值结果就是 FUNCTION 子程序中表达式所求之值。

·根据 SUBROUTINE 子程序返回，在 SUBROUTINE 中赋给参变量的值，就是调用程序所需要的值。例：

```
Calling  PROGRAM Unit
  ⋮
CALL  SUBR (Z9, B7, R1)
  ⋮
Called  PROGRAM Unit
SUBROUTINE SUBR (A, B, C)
READ (3, 7) B
A = B**C
RETURN
7  FORMAT (F9.2)
END
```

在这个例子中，Z 9 和B 7 在返回主程序时就是调用程序有用的结果数据。

第七节　子程序中数组的处理

如果一个调用程序传递一个数组名给子程序，则子程序必须包括与数组有关的维数信息。

如果子程序的任何一个哑元变量表示数组或数组元素，子程序必须包括数组说明符。例：

```
Calling  Program Unit
DIMENSION Z1 (50), Z2 (25)
    ⋮
  AI = AVG (Z1, 50)
Called Program Unit
FUNCTION AVG (ARG, I)
DIMENSION ARG (50)
SUM = 0.0
DO 20 J = 1, I
20 SUM = SUM + ARG (J)
AVG = SUM/FLOAT (I)
RETURN
END
```

注意，用 FUNCTION 子程序处理实际数组时，是在调用程序中定义维数的。数组名和它们的实际维数根据 FUNCTION 子程序的引用传送给 FUNCTION 子程序。FUNCTION 子程序本身包含一个哑元数组并指定了一个数组说明符。

维数信息也可以传递给参数表中的子程序。例：

```
Calling Program Unit
DIMENSION A( 3 , 4 , 5 )
CALL SUBR(A, 3 , 4 , 5 )
END
Called Program Unit
SUBROUTINE SUBR(X,I,J,K)
DIMENSION X(I,J,K)
RETURN
END
```

只有当数组名和所有维数变量是哑元变量时，使用可变维数才是正确的。可变维数必须是整型数，在调用程序内修改可变维数的值是无效的。

第八节　数据块子程序

一个 BLOCK DATA 块数据子程序在一个 FORTRAN 目的程序装载时，只有初始化 COMMON 块数据起作用。BLOCK DATA 子程序是以一个 BLOCK DATA 语句开始的，其格式如下：

```
BLOCK DATA (子程序名)
```

并以一个 END 语句结束。

子程序名是可选择的，它是与 BLOCK DATA 子程序相关的符号名。

这样的子程序只包含 TYPE、EQUIVALENCE、DATA 等语句，并受下面条件的约束：

· 如果在一个 COMMON块中任何一个元素已经初始化，那么这个块的所有元素必须列入 COMMON 语句的列表中，不必考虑全部都要初始化。

· 在一个以上的 COMMON 通用块中，数据的初始化可以在一个BLOCK DATA 子程序中完成。

· 在任何给定的时间内，允许一个以上的 BLOCK DATA 子程序被装载。

· 任何一个专用的 COMMON 块中的项只能由一个程序来初始化。

第九节　程序链接

链接处理给用户提供了顺序执行一系列独立程序的可能性。多个程序由 FORTRAN 程序通过 CALL FCHAIN 来完成装载和执行。通常其语句格式为：

```
CALL FCHAIN ('filename')
```

程序链受下述条件的限制：

· 文件名必须符合 CP/M 的规定。

· FCHAIN 程序必须是一个“MAIN”程序，就是说要有一个 ENTRY入口点。FORTRAN、COBLE 和汇编语言子程序不包含“MAIN”主程序的入口点。

· 参数不可以传递给 FCHAINed 程序。

· 非法文件名、非法驱动器引用、文件找不到以及磁盘读错误等都会引起致命性的 I/O 错误。

第十一章　FORTRAN语句总结

第一节　综　　述

本章是该版本FORTRAN语言中所实现语句的汇总。它包括了每个语句的扼要描述。每个语句的结构遵照下列记号的约定：1．符号变量由斜体的小写字母序列表示；2．符号常量或关键字由大写字母序列表示；3．一对括号(〔　〕)指明一种可选项；4．连续三个句点或缩写符号表示可以重复零次或重复多次的项。

第二节　语句总结

ASSIGN j TO i　　(标号赋值语句)

其中，j是一个语句标号，i是一个整型变量。

该语句将与赋值GO TO语句一起使用。当执行赋值GO TO语句时，控制会传递给标号为j的语句。如果在赋值GO TO语句内指定一个列表，则j必须包含在该列表内。

BLOCK DATA〔subprogram-name〕　　(数据块语句)

其中，〔"subprogram-name"〕是任意合法的符号名。

该语句用来指定数据块子程序的名字。数据块子程序用来初始化公共区中的变量。数据块子程序以BLOCK DATA语句开始，以END语句结束。数据块子程序只能含有TYPE、EQUIVALENCE、DATA、COMMON以及DIMENSION等语句。

CALL s〔(〔a〕〔a〕…)〕　　(调用语句)

其中，s是该子程序名，a是该子程序使用的实际变量。

CALL（调用）语句用来将程序的控制传递给某个子程序。当调用某个SUBROUTINE（子程序）时，程序控制就转移到子程序的第一个可执行语句。传递给子程序的变量必须在类型、顺序及个数上与SUBROUTINE中相应的形式变量相一致。

CALL FCHAIN(`filename')

其中，(`filename')是任何合法的、机器可执行的文件。

该语句用来从原始程序内部装入并执行附加程序。`filename'（文件名）是对于机器可执行文件的一个合法的文件说明，它依赖于操作系统。

COMMON〔/〔cb〕/〕nlist〔〔, 〕/〔cb〕/list〕…　　(公用区语句)

其中，cb是公用区名，list是变量列表。

COMMON（公用区）语句是存储分配语句，它把变量与数组分配给称作COMMON的

存储区。这就允许不同的程序体使用相同的存储区域。变量列表可以是一系列变量名、数组名或常量数组说明符。这些名字必须以逗号分隔。COMMON(公用区)块的名字可以省略，称之为空白共用区（COMMON）。

CONTINUE　　（继续语句）

在DO循环体中，如果通常情况下的终止语句恰好为某种不允许使用的语句，则采用CONTINUE语句作为终止语句。

DATA list/clist/[[,]list/clist/]…　　（数据语句）

其中，list是以逗号隔开的变量列表，clist是赋给该组变量的常量值。

DATA(数据)语句用来把常量值编译到目的程序中，并把这些值赋给变量与数组元素。

DECODE (a, f) k　　（译码语句）

其中，a是数组名，f是FORMAT语句标号，k是I/O列表。

该语句的作用是把数组a中的元素从字符格式翻译成FORMAT语句f中所规定的内部格式。该转换结果放在I/O元素k的列表上。除了数据传递是从一内存区到另一内存区以外，该语句类似于READ（读）语句。

DIMENSION s (d) [, s(d)]…　　（维数语句）

其中，s是数组名，d是数组维数说明。

该语句将为数组的每个元素保留存储单元，这样数组元素可以通过该数组名后加一个下标变量来引用。

DO k i = m1, m2[, m3]　　（循环语句）

其中，k是终止语句的语句标号，i是循环控制变量，m1是初始值，m2是终止值，m3是增量（如果省略，则缺省值为1）。

DO（循环）语句为重复执行一系列语句提供了一种方法。标号k所指的语句必须是可执行语句。循环控制变量必须是正值，并且不允许由DO循环体内的任何语句修改。执行一个DO循环时将发生下述几步：1. 设置i = m1；2. 经过k执行语句列；3. i = m1 + m3；4. 已经到达终值（i = m2）？是——转移到k之后的语句；否——重复2～4。

ENCODE (a, f)　k　　（编码语句）

其中，a是数组名，f是FORMAT语句标号，k是I/O列表。

该语句是把I/O列表k中的元素转换成字符格式。该数据按照FORMAT语句f中的规定来转换。转换好的数据放到数组a中。除了这种转换是从一内存区到另一内存区之外，这个语句类似于WRITE（写）语句。

END　　（结束语句）

END（结束）语句实际上是任何FORTRAN程序的最后一个语句。

ENDFILE u　　（文件结束语句）

其中，u是一个整型变量或常量。

该语句将关闭与逻辑设备号u相连的文件。这个语句仅适用于磁盘文件。

EQUIVALENCE (list)〔, (list)〕…　　（等价语句）

其中，list是两个或多个变量或者数组元素的序列，它们彼此之间用逗号分隔。

使用EQUIVALENCE（等价）语句允许有两个或多个实体来共享相同的存储单元。编译程序将把列表中每一元素分配到相同的存储单元，元素在列表中出现的先后无关紧要。

EXTERNAL v〔, v〕…　　（外部语句）

其中，v是子程序名。

该语句允许把外部过程名用作为其他子程序的变量。外部过程名可以是某个SUBROUTINE名、BLOCK DATA名或FUNCTION名。当调用某个子程序时，EXTERNAL语句允许把任何名为v的变量用作子程序的变量。

FORMAT (s〔, s〕…)　　（格式语句）

其中，s是字段描述符。

FORMAT（格式）语句将与带格式READ（读）语句与格式WRITE（写）语句一起使用。它们规定了在计算机主存与外部存储设备之间的转换方式以及作为传递数据执行的编辑信息。FORMAT语句要求带有语句标号以满足READ与WRITE语句中引用的需要。

t FUNCTION f〔([a〔, a〕…])〕　　（函数语句）

其中，t是数据类型（可选择），f是子程序名，a是哑元变量名。

该语句表明FUNCTION（函数）子程序的开始，名字f是作为该FUNCTION(函数)的引用名。t可以是整型、实型、双精度型或逻辑型。FUNCTION（函数）语句必须是该程序体的第一个语句，该程序体必须以RETURN（返回）语句作结束。

GO TO k (Unconditional GO TO)　　（无条件转向语句）

其中，k是可执行语句的标号。

该程序的执行控制被转移到语句k。

GO TO (k1, k2, …kn) , j (Computed GO TO)　　（计算转向语句）

其中，ki是可执行语句的标号，j是整型变量。

该语句将把控制转移到标号为kj的语句（如果j= 1，则控制转移到标号为k1的语句；如果j= 2，则控制转移到标号为k2的语句；依次类推。如果j< 1或j>n，则控制将转移到该计算转向语句的下一个语句）。

GO TO j, 〔(k1, k2, …, kn)〕(Assigned GO TO)　　（赋值转向语句）

其中，j是整型变量，ki是可执行语句的标号。

该语句把控制转移到其标号等于j的当前值的语句。其中j由ASSIGN（标号赋值）语句赋给一个值。如果语句标号ki存在，则j就已获得了一个包含在该列表中的值。ASSIGN语句从逻辑上讲必须放在该赋值转向语句之前。

IF(e)m1, m2, m3 (ARITHMETIC IF)　　(算术IF语句)

其中，e是算术表达式，mi是语句标号。

根据算术表达式（e）的计算结果来传递控制。如果e的计算结果为负（<0），则控制转移到标号为m1的语句；如果e的计算结果为0（=0），则控制转移到标号为m2的语句；如果e的计算结果为正（>0），则控制转移到标号为m3的语句。

IF (u) s (Logical IF)　　(逻辑IF语句)

其中，u是逻辑表达式，s是除了DO语句之外的任意可执行语句。

逻辑表达式u按真值(TRUE)或假值(FALSE)来计算。如果u取假值（FALSE），则不处理该语句而将控制转移到该逻辑IF语句的下一语句；如果u取真值（TRUE），则控制转移到语句s。以后的程序控制遵循正常的条件。如果s是替换语句（v=e），则变量v与等号（=）必须在同一行上。

PAUSE〔c〕　　(暂停语句)

其中，c是长度不大于6的字串。

当在执行目的程序期间遇到PAUSE语句，字符串c（如果有的话）就会显示在终端设备上并且程序执行中断。此时，如在终端设备上送入一字符“T”与回车键，则程序执行结束；而送入任何其他字符与回车键，可使执行继续下去。

PROGRAM name　　(程序语句)

其中，name规定了该主程序的名字。

PROGRAM（程序）语句为某个主程序体提供了一种命名的方法。该名字含有1～6个字母数字字符，而第一个字符必须是字母。如果某个主程序中没有PROGRAM语句，则编译程序就会自行把一个名为$MAIN的程序名赋予该程序。

READ (u,f〔, ERR=L1〕〔, END=L2〕)k　　(格式顺序读语句)

其中，u是逻辑设备号，f是FORMAT语句标号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达EOF的话），k是变量名列表。

该语句将按照列表中的变量名来输入一组数据。这种输入来自逻辑设备号为u的文件。该输入按照FORMAT语句f来转换。

READ (u〔ERR=L1〕, 〔END=L2〕)k　　(非格式顺序读语句)

其中，u是逻辑设备号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达EOF的话），k是变量名列表。

除了执行一个不带数据转换或编辑信息的数据内存映象传递以外，该语句等同于格式

READ 读句。所以传递的数据量应符合列表 k 中变量的个数与类型。

READ (u,f,REC=i[, ERR=L1])k　　(格式随机存取语句)

其中，u 是逻辑设备号，f 是 FORMAT 语句标号，i 是要读入的记录号，L1是传递标号（如果遇到某个错误的话），k 是I/O 列表。

该语句基本上类似于格式 READ 语句，只是文件中的任意记录可由包含的 REC=i 条件来读入。

READ (u,REC=i[,ERR=L1])k　　(非格式随机存取语句)

其中，u 是逻辑设备号，i 是要读入的记录号，L1是传递标号（如果遇到某个错误的话），k 是 I/O 列表。

该语句读入指定文件的第 i 个记录。输入的数据将赋给列表 k 中的变量。该数据在传递过程中将不带任何编辑信息或格式。

READ (u,f[ERR=L1][,END=L2])　　(H 类型转换)

其中，u 是逻辑设备号，f 是 FORMAT 语句标号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达 EOF 的话），不需要变量列表。

该语句将与 FORMAT 语句一起使用，把 H类型的字母数字数据读到已经存在的 H类型字段中。

RETURN　　(返回语句)

将控制返回到调用的程序。

FUNCTION（函数）或 SUBPROGRAM（子程序）的逻辑终止是 RETURN 语句。该语句将把控制返回到原调用程序。

REWIND u　　(反绕语句)

其中，u 是整型变量或常量。

该语句先关上，然后再打开与逻辑设备 u 相连的文件。它对非磁盘文件无效。

STOP[c]　　(停止语句)

其中，c 是长度不大于 6 的字串。

在执行目的程序时，如果遇到 STOP 语句，字符串 c（如果有的话）就会显示在终端设备上，然后程序执行中止。STOP 语句可以作为该程序的逻辑结束。

SUBROUTINE s[([a][,a]…)]　　(子程序语句)

其中，s 是子程序名，a 是哑元变量。

以 SUBROUTINE（子程序）语句开始的程序体可称为 SUBROUTINE 子程序。子程序名不必出现在该子程序内部的任何其他语句中。该子程序可以被主程序的 CALL 语句引用。

TYPE v[, v]…　　(类型语句)

其中，TYPE是数据类型说明符，v是变量名、数组名或函数名。

TYPE（类型）语句为某个变量名提供相应的某种数据类型。这可用来核实或说明预定义的约定。另外，这些语句可用来说明数组。数组类型规定符可以是下述任何一种：

整型	实型	逻辑型	双精度型
单字节整型	四字节实型	单字节逻辑型	字节型
双字节整型	八字节实型	双字节逻辑型	
四字节整型			

WRITE(u,f[,ERR=L1][,END=L2])k　　（非格式顺序写语句）

其中，u是逻辑设备号，f是FORMAT语句标号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达EOF的话），k是变量列表。

该语句用来把列表k中所指定的数据输出到逻辑设备为u的输出设备上。输出时，数据将按照FORMAT语句f来转换并编辑。

WRITE(u,ERR=L1,END=L2)k　　（非格式顺序写语句）

其中，u是逻辑设备号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达EOF的话），k是变量列表。

除了把数据的存储映象传递到输出设备外，并不进行数据转换和编辑。该语句类似于格式WRITE语句。所传递的数据量应与列表k中的变量数和变量类型相符合。

WRITE (u,f,REC=i[,ERR=L1])k　　（格式随机存取语句）

其中，u是逻辑设备号，f是FORMAT语句标号，i是要写的记录号，L1是传递标号（如果遇到某个错误的话），k是变量列表。

除了文件中要写的任何记录号可以包含在REC=i条款以外，该语句与格式WRITE语句基本相同。

WRITE (u,REC=i[,ERR=L1])k　　（非格式随机存取语句）

其中，u是逻辑设备号，i是要写的记录号，L1是传递标号（如果遇到某个错误的话），k是I/O列表。

除了引用非格式语句之外，该语句类似于格式随机存取语句。因此在数据传递时没有格式化或编辑。

WRITE (u,f[,ERR=L1][,END=L2])　　（无变量列表写语句）

其中，u是逻辑设备号，f是FORMAT语句标号，L1是传递标号（如果遇到某个错误的话），L2是传递标号（如果已到达EOF的话）。

当要打印FORMAT语句中所规定的字符时，该语句用来输出字母数字信息，在此情况下则不需要变量列表。

附　录

Microsoft FORTRAN-80 CP/M 版本安装指南

盘片内容

你收到的盘片上将有下列文件：

Microsoft FORTRAN-80配给盘 I：

F80.COM

M80.COM

L80.COM

FORLIB.REL

F80.COM 是 FORTRAN-80的编译程序。在 Microsoft FORTRAN-80参考手册中定义了其命令及操作。MACRO-80汇编程序 (M80.COM) 与连接装入程序 (L80.COM) 在Microsoft "实用程序软件" 中描述。FORLIB.RFL 是 FORTRAN-80 编译系统的库文件，该文件的改进型是通过使用库文件管理程序来完成。

Microsoft FORTRAN-80配给盘 II：

CREF.COM

LIB.COM

DSKDRV.MAC

FCHAIN.MAC

INIT.MAC

IOINIT.MAC

LPTDRV.MAC

DTBF.MAC

LUNTB.MAC

PI.FOR

交叉引用工具(CREF.COM)与库文件管理程序 (LIB.COM) 在 Microsoft "实用程序软件" 中描述。PI.FOR 是一个计算 π 的值的样品程序，可以利用它来了解编译、链接及执行某个程序所必需的过程。在利用这个文件达到上述目的之后，就可以从盘片上把该文件删除掉。

在 FORTRAN-80 配给盘 II 上已经包含了一些扩展名为 ".MAC" 的文件。这些文件

包含了FORTRAN-80运行库文件中关于一些库功能的源程序。下表列出了这些源文件及其功能。

文件名	功能
DSKDRV.MAC	CP/M 运行磁盘驱动器程序
FCHAIN.MAC	FORTRAN 调用 FCHAIN 语句
INIT.MAC	运行时初始化
IOINIT.MAC	I/O 标识及变量初始化
LPTDRV.MAC	行打机设备驱动程序
DTBF.MAC	运行时数据缓冲区
LUNTB.MAC-	逻辑设备号调度表

这些源文件是为熟练的系统程序员而提供的，为他们自己应用的需要而希望去修改程序时采用。这些子程序利用了 MACRO-80 汇编程序的条件汇编这一特征，并且已经为它们形成了只能在 CP/M 系统下可运行的程序。在 FORTRAN-80 用户手册 “程序链” 一节中描述了 FCHAIN 的用途。关于使用 “·MAC” 的附加情况可以参阅 FORTRAN-80 参考手册附录 B。

当你已经汇编好你所习惯的子程序后，就可以建立一个包含这些子程序的新的库文件。库文件管理实用程序 LIB-80可使用这些功能。在建立一个新的库文件之前，要保证你已经建立好一个标准库文件的复制品。

在把这些子程序包含到你的库文件中之前，你应该测试每个新的子程序，这一点是非常重要的。我们不能对已作修改的或用户所习惯的库文件的引用负责。

根据所收到的配给盘介质类型的不同，上述所说的那些文件，可以记录在一张或多张盘片上。

磁盘使用

磁盘装载

打开磁盘驱动器门，把贴有磁盘标签这一面，对准所打开的驱动器门，插入磁盘，然后小心地关上驱动器门。

磁盘处理

磁盘很容易损坏，在处理磁盘时需注意下述保护措施：

1. 当不使用磁盘时，应把其放在磁盘封套中
2. 应使盘片远离磁场，包括磁性纸夹、带磁的剪刀或螺丝刀及强电场。磁场会改变记录在磁盘上的数据
3. 替换已损坏或已磨损的盘套
4. 只能在磁盘标签上写，且只能用毡芯软笔写。不要用铅笔或圆珠笔，这样做会损坏所记录的盘面
5. 使磁盘远离热的或污染的物质
6. 不要把磁盘暴露在阳光下或放置在液体与气体中
7. 不要接触磁盘表面，磨擦会改变所存储的信息

写保护

磁盘可以写保护，以使数据不能再写到磁盘上（所有的配给盘均已被贴上写保护）。写保护这种办法依赖于磁盘的容量。

5·25英寸软磁盘上有一个可写保护的缺口。当该缺口用薄纸或不透明的纸贴起来盖上时，数据就不能再写到该磁盘上。

8英寸磁盘上也有一个写保护缺口。如果该写保护的缺口开着，则数据就不能写到该磁盘上。为了把数据写到8英寸磁盘上，可用薄纸或不透明的纸把该缺口复盖起来。注意，当写保护纸去掉以后，8英寸磁盘正好与5.25英寸磁盘相反，是写保护的。

准备工作盘

利用CP/M手册中简明的过程，开机并引导CP/M系统。

如果用户具有两个或两个以上容量相同的驱动器，就可以利用DUP.COM文件来复制FORTRAN-80配给盘。如没有两个或两个以上容量相同的驱动器，则需按下述两步来做：

1. 运用FORMAT.COM文件来格式化你所要复制的空格盘片。

2. 利用PIP.COM文件来复制FORTRAN-80配给盘。注：所有的配给盘是写保护的，以保证你任何时候总能有准确的软件复制品。然后复制配给盘并把其存放在安全的地方。每天所使用的应该是这些程序的复制品。

本手册编译校对：周琪琪、张文秋、康兴鹤

责任编辑：谢雪峰、王宏成、暴雪琴、吴雪莹、王晓东