Digital Equipment Corporation
Maynard, Massachusetts

digital

FORTRAN IV

PDP-10
PROGRAMMER'S REFERENCE MANUAL

DEC-10-AFC0-D

PDP-10
FORTRAN IV
PROGRAMMING MANUAL

Order No. DEC-10-AFC0-D from Program Library, Maynard, Mass. Price $2.00

Direct comments concerning this manual to Software Quality Control, Maynard, Mass.

DIGITAL EQUIPMENT CORPORATION ▫ MAYNARD, MASSACHUSETTS

1st Printing March 1967
2nd Printing (Rev) November 1967
3rd Printing (Rev) September 1968
4th Printing April 1969
5th Printing (Rev) August 1969

FOREWORD

This is a reference manual describing the specific statements and features in the FORTRAN IV language for the PDP-10. Familiarity with the basic concepts of FORTRAN programming on the part of the reader is assumed. This system conforms to the requirements of USA Standard FORTRAN.

CONTENTS

CONTENTS (continued)

CONTENTS (continued)

ILLUSTRATIONS

TABLES

CHAPTER 1

INTRODUCTION

The term FORTRAN IV (FORmula TRANslation) is used interchangeably to designate both the FORTRAN IV language and the FORTRAN IV translator or compiler. The FORTRAN IV language is composed of mathematical-form statements constructed in accordance with precisely formulated rules. FORTRAN IV programs consist of meaningful sequences of FORTRAN statements intended to direct the computer to perform the specified operations and computations.

The FORTRAN IV compiler is itself a computer program that examines FORTRAN IV statements and tells the computer how to translate the statements into machine language. The compiler runs in a minimum of 9K of core. The program written in FORTRAN IV language is called the source program. The resultant machine language program is called the object program.

FORTRAN IV includes such advanced features as logical operators, type declaration statements, double precision and complex arithmetic, named COMMON, and DATA statements.

FORTRAN IV language elements are discussed in Chapter 2 of this manual, followed by separate chapters on the five categories of FORTRAN IV statements (arithmetic, control, input/output, specification, and subprogram). The appendices contain a list of FORTRAN statements and summary descriptions of library functions and subroutines. Digital's small FORTRAN compiler, which runs in 5.5K of core, is virtually identical to the larger compiler, except for differences explained in Appendix 7.

Operating procedures and diagnostic messages for both compilers are explained in the PDP-10 Systems Users Guide (DEC-10-NGCA-D).

LINE FORMAT

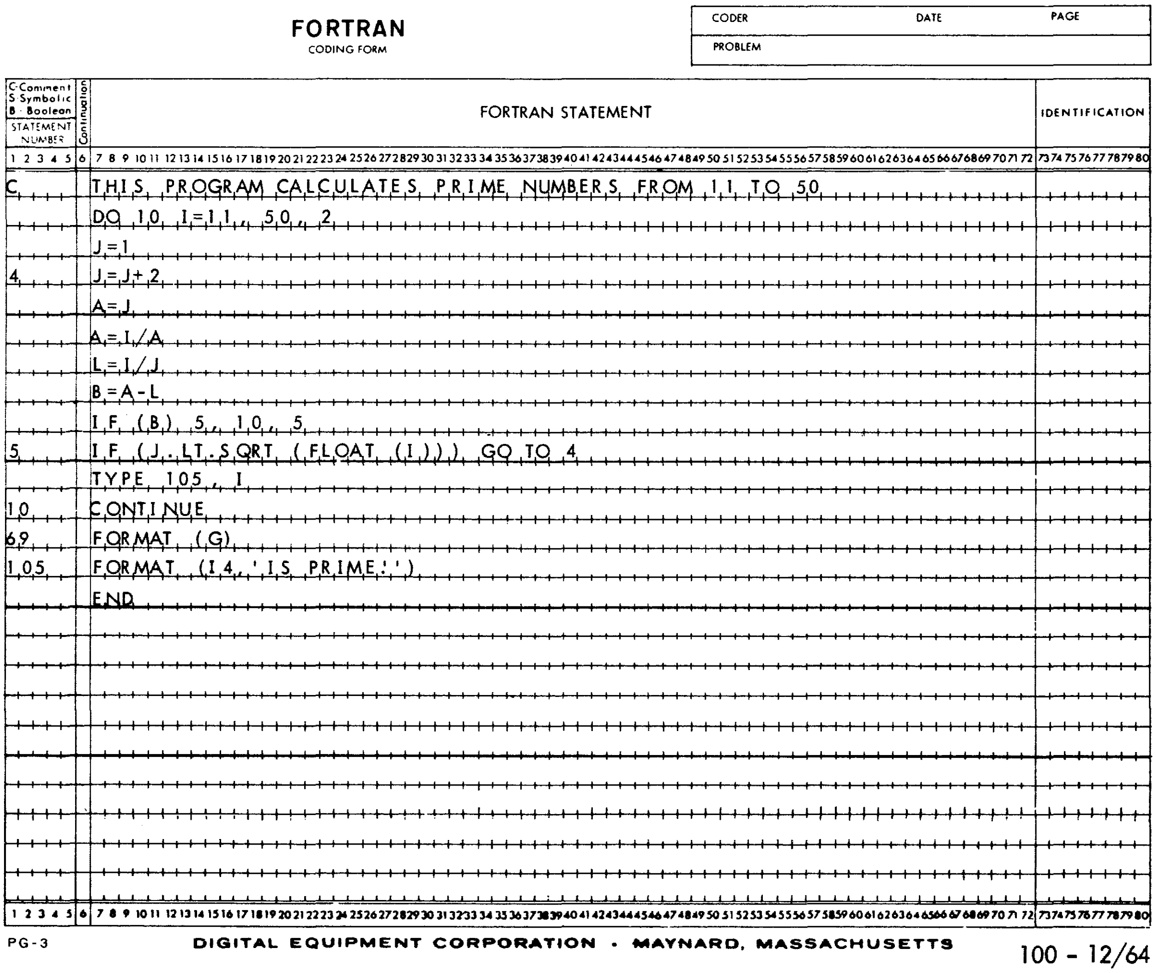
Each line of a FORTRAN program consists of three fields: statement number field, line continuation field, and statement field. A typical FORTRAN program is shown in Figure 1.

Statement Number Field

A statement number consists of from one to five digits in columns 1-5. Leading zeros and all blanks in this field are ignored. Statement numbers may be in any order and must be unique. Any statement referenced by another statement must have a statement number. For source programs prepared on a teletypewriter, a horizontal tab may be used to skip to the statement field. This is the only place a tab is legal.

Line Continuation Field

If a FORTRAN statement is so large that it cannot conveniently fit into one statement field, the statement fields of up to 19 additional lines may be used to specify the complete statement. Any line which is not continued, or the first line of sequence of continued lines, must have a blank or zero in column 6. Continuation lines must have a character other than blank or zero in column 6.




```
C     THIS PROGRAM CALCULATES PRIME NUMBERS FROM 11 TO 50
      DO 10 I=11,50,2
      J=1
4     J=J+2
      A=J
      A=I/A
      L=I/J
      B=A-L
      IF (B) 5,10,5
5     IF (J.LT.SQRT(FLOAT(I))) GO TO 4
      TYPE 105,I
10    CONTINUE
69    FORMAT (G)
105   FORMAT (I4,' IS PRIME.')
      END
```

Figure 1   Typical FORTRAN Coding Form

Statement Field

Any FORTRAN statement, as described in later sections, may appear in the statement field (columns 7-72). Except for alphanumeric data within a FORMAT statement, blanks (or spaces) are ignored and may be used freely for appearance purposes.

Comment Line

Any line which starts with the letter C in column 1 is interpreted as a line of comments. Comment lines are printed onto any listings requested but are otherwise ignored by the compiler. Columns 2-72 may be used in any format for comment purposes.

CHARACTER SET

The following characters are used in the FORTRAN IV language:

Blank	0	@	P
!	1	A	Q
"	2	B	R
#	3	C	S
$	4	D	T
%	5	E	U
&	6	F	V
'	7	G	W
(	8	H	X
)	9	I	Y
*	:	J	Z
+	;	K	↑
,	<	L	
-	=	M	
.	>	N	
/	?	O	

CHAPTER 2

CONSTANTS, VARIABLES, AND EXPRESSIONS

The rules for defining constants and variables and for forming expressions are described in this chapter.

CONSTANTS

Seven types of constants are permitted in a FORTRAN IV source program: integer or fixed point, real or single-precision floating point, double-precision floating point, octal, complex, logical, and literal.

Integer Constants

An integer constant consists of from one to eleven decimal digits written without a decimal point.

EXAMPLES:

3
-528
8085

An integer constant must fall within the range $-2^{35}+1$ to $2^{35}-1$. When used for the value of a subscript or as an index in a DO statement, the value of the integer is taken as modulo 2^{18}.

Real Constants

Real constants are written as a string of decimal digits including a decimal point. A real constant may consist of any number of digits but only the leftmost 9 digits appear in the compiled program. Real constants may be given a decimal scale factor by appending an E followed by an integer constant. The field following the letter E must not be blank, but may be zero.

EXAMPLES:

15.
.579
5.0E3(i.e., 5000.)

A real constant has precision to eight digits. The magnitude must lie approximately within the range 0.14E-38 to 1.7E38.

Double Precision Constants

A double precision constant is specified by a string of decimal digits, including a decimal point, which are followed by the letter D and the decimal scale factor. The field following the letter D must not be blank, but may be zero.

EXAMPLES:

24.671325982134D0
3.6D2 (i.e., 360.)
3.6D-2 (i.e., .036)

Double precision constants have precision to 16 digits. The magnitude of a double precision constant must lie approximately between 0.14E-38 and 1.7E38.

Octal Constants

A number preceded by a double quote represents an octal constant. An octal constant may appear in an arithmetic or logical expression or a DATA statement. Only the digits 0-7 may be used and only the first twelve digits are significant.

EXAMPLES:

"7777
"-31563

Complex Constants

FORTRAN IV provides for direct operations on complex numbers. Complex constants are written as an ordered pair of real constants separated by a comma and enclosed in parentheses.

EXAMPLES:

(.70712, -.70712)
(8.763E3,2.297)

The first constant of the pair represents the real part of the complex number, and the second constant represents the imaginary part. The real and imaginary parts may each be signed. The enclosing parentheses are part of the constant and always appear, regardless of context.

FORTRAN IV arithmetic operations on complex numbers, unlike normal arithmetic operations, must be of the form:

$$A \pm B = a_1 \pm b_1 + i(a_2 \pm b_2)$$

$$A * B = (a_1 b_1 - a_2 b_2) + i\,(a_2 b_1 + a_1 b_2)$$

$$A/B = \frac{(a_1 b_1 + a_2 b_2)}{b_1^2 + b_2^2} + i\,\frac{(a_2 b_1 - a_1 b_2)}{b_1^2 + b_2^2}$$

where $A = a_1 + ia_2$, $B = b_1 + ib_2$, and $i = \sqrt{-1}$.

Logical Constants

The two logical constants, .TRUE. and .FALSE., have the internal values −1 and 0, respectively. The enclosing periods are part of the constant and always appear.

Logical constants may be entered in DATA or input statements as signed octal integers (−1 and 0). Logical quantities may be operated on in either arithmetic or logical statements. Only the sign is tested to determine the truth value of a logical variable.

Literal Constants

A literal constant may be in either of two forms:

1. A string of characters enclosed in single quotes; two adjacent single quotes within the constant are treated as one single quote.

2. A string of the form:

$$nHx_1x_2 \ldots x_n$$

where $x_1x_2 \ldots x_n$ is the constant, and n is the number of characters following the H.

EXAMPLES:

```
'LITERAL CONSTANT'
'DON''T'
5HDON'T
```

VARIABLES

A variable is a quantity whose value may change during the execution of a program. Variables are specified by name and type. The name of a variable consists of one or more alphanumeric characters, the first one of which must be alphabetic. Only the first six characters are interpreted as defining the variable name. The type of variable (integer, real, logical, double precision, or complex) may be specified by a type declaration statement or determined by the first letter of the variable name. A first letter of I, J, K, L, M or N indicates a fixed point (integer) variable; any other first letter indicates a floating-point variable. Variables of any type may be either scalar or array variables.

SCALAR VARIABLES

A scalar variable represents a single quantity.

EXAMPLES: A

G2

POPULATION

ARRAY VARIABLES

An array variable represents a single element of an n dimensional array of quantities. The variable is denoted by the array name followed by a subscript list enclosed in parentheses. The subscript list is a sequence of integer expressions, separated by commas. The expressions may be arithmetic combinations of integer variables and integer constants. Each expression represents a subscript, and the values of the expressions determine the array element referred to. For example, the row vector A_I would be represented by the subscripted variable A(J), and the element, in the second column of the first row of the square matrix A, would be represented by A(1,2). Arrays may have any number of dimensions.

EXAMPLES: Y(1)

STATION (K)

A (3* K+2, I, J-1)

The three arrays above (Y, STATION, and A) would have to be dimensioned by a DIMENSION, COMMON, or type declaration statement prior to their first appearance in an executable statement or in a DATA or NAMELIST statement. (Array dimensioning is discussed in chapter 6.)

Arrays are stored in increasing storage locations with the first subscript varying most rapidly and the last subscript varying least rapidly. For example, the 2-dimensional array B(I,J) is stored in the following order: B (1,1), B (2,1),..., B (I,1),B (1,2),B (2,2),...,B (I,2),...,B (I,J).

EXPRESSIONS

Expressions may be either numeric or logical. To evaluate an expression, the object program performs the calculations specified by the quantities and operators within the expression.

Numeric Expressions

A numeric expression is a sequence of constants, variables, and function references separated by numeric operators and parentheses in accordance with mathematical convention and the rules given below.

The numeric operators are +, −, *, /, **, denoting, respectively, addition, subtraction, multiplication, division, and exponentiation.

In addition to the basic numeric operators, function references are also provided to facilitate the evaluation of functions such as sine, cosine, and square root. A function is a subprogram which acts upon one or more quantities, called arguments, to produce a single quantity called the function value. Function references are denoted by the identifier, which names the function (such as SIN, COS, etc.), followed by an argument list enclosed in parentheses:

```
identifier(argument, argument, ..., argument)
```

At least one argument must be present. An argument may be an expression, an array identifier, a subprogram identifier, or an alphanumeric string.

Function type is given by the type of the identifier which names the function. The type of the function is independent of the types of its arguments. (See Chapter 7.)

A numeric expression may consist of a single element (constant, variable, or function reference):

```
2.71828
Z(N)
TAN(THETA)
```

Compound numeric expressions may be formed by using numeric operators to combine basic elements:

```
X+3.
TOTAL/A
TAN(PI*M)
```

Compound numeric expressions must be constructed according to the following rules:

1. With respect to the numeric operators +, −, *, /, any type of quantity (logical, octal, integer, real, double precision, complex or literal) may be combined with any other, with one exception: a complex quantity cannot be combined with a double precision quantity.

The resultant type of the combination of any two types may be found in Table 1. The conversions between data types will occur as follows:

(a) A literal constant will be combined with any integer constant as an integer and with a real or double word as a real or double word quantity. (Double word refers to both double precision and complex.)

(b) An integer quantity (constant, variable, or function reference) combined with a real or double word quantity results in an expression of the type real or double word respectively; e.g., an integer variable plus a complex variable will result in a complex subexpression. The integer is converted to floating point and then added to the real part of the complex number. The imaginary part is unchanged.

(c) A real quantity (constant, variable, or function reference) combined with a double word quantity results in an expression that is of the same type as the double word quantity.

(d) A logical or octal quantity is combined with an integer, real, or double word quantity as if it were an integer quantity in the integer case, or a real quantity in the real or double word case (i.e., no conversion takes place).

2. Any numeric expression may be enclosed in parentheses and considered to be a basic element.

```
(X+Y)/2
(ZETA)
(COS(SIN(PI*M)+X))
```

TABLE 1 TYPES OF RESULTANT SUBEXPRESSIONS

	+,−,*,/	Type of Quantity: Real	Integer	Complex	Double Precision	Logical, Octal, or Literal
Type of Quantity	Real	Real	Real	Complex	Double Precision	Real
	Integer	Real	Integer	Complex	Double Precision	Integer
	Complex	Complex	Complex	Complex	Not Allowed	Complex
	Double Precision	Double Precision	Double Precision	Not Allowed	Double Precision	Double Precision
	Logical, Octal, or Literal	Real	Integer	Complex	Double Precision	Logical, Octal, or Literal

3. Numeric expressions which are preceded by a + or − sign are also numeric expressions:

```
+X
-(ALPHA*BETA)
-SQRT(-GAMMA)
```

4. If the precedence of numeric operations is not given explicitly by parentheses, it is understood to be the following (in order of decreasing precedence):†

Operator	
**	numeric exponentiation
*and/	numeric multiplication and division
+and-	numeric addition and subtraction

In the case of operations of equal hierarchy, the calculation is performed from left to right. This is also true for exponentiation.

† See also page 14

5. No two numeric operators may appear in sequence. For instance:

X*-Y

is improper. Use of parentheses yields the correct form:

X*(-Y)

By use of the foregoing rules, all permissible numeric expressions may be formed. As an example of a typical numeric expression using numeric operators and a function reference, the expression for the largest root of the general quadratic equation:

$$\frac{-b+\sqrt{b^2-4ac}}{2a}$$

would be coded as:

```
(-B+SQRT(B**2-4.*A*C))/(2.*A)
```

Logical Expressions

A logical expression consists of logical constants, logical variables, logical function references, and arithmetic expressions, separated by logical operators or relational operators. Logical expressions are provided in FORTRAN IV to permit the implementation of various forms of symbolic logic. Logical constants are defined by arithmetic statements, which are described in Chapter 3. Logical variables and functions are defined by the LOGICAL statement, described in Chapter 6. Binary variables may be represented by the logical constants .TRUE. and .FALSE., which must always be written with enclosing periods.

Logical Operators

The logical operators, which include the enclosing periods and their definitions, are as follows, where P and Q are logical expressions:

.NOT.P	Has the value .TRUE. only if P is .FALSE., and has the value .FALSE. only if P is .TRUE.
P.AND .Q	Has the value .TRUE. only if P and Q are both .TRUE., and has the value .FALSE. if either P or Q is .FALSE.
P.OR.Q	(Inclusive OR) Has the value .TRUE. if either P or Q is .TRUE., and has the value .FALSE. only if both P and Q are .FALSE.

P.XOR.Q	(Exclusive OR) Has the value .TRUE. if either P or Q but not both are .TRUE., and has the value .FALSE. otherwise.
P.EQV.Q	(Equivalence) Has the value .TRUE. if P and Q are both .TRUE. or both .FALSE., and has the value .FALSE. otherwise.

Relational Operators

The relational operators are as follows:

Operator	Relation
.GT.	greater than
.GE.	greater than or equal to
.LT.	less than
.LE.	less than or equal to
.EQ.	equal to
.NE.	not equal to

The enclosing periods are part of the operator and must be present.

Mixed expressions involving integer, real, and double precision types may be combined with relationals. The value (.TRUE. or .FALSE.) of such relations will be calculated by subtraction; i.e.,

$$\text{expression}_1 \text{ "relation" } \text{expression}_2$$

will be calculated as though:

$$\text{expression}_1 - \text{expression}_2 \text{ "relation" zero}$$

had been written.

The relational operators .EQ. and .NE. may also be used with COMPLEX expressions. (Double word quantities are equal if the corresponding parts are equal.)

A logical expression may consist of a single element (constant, variable, function reference, or relation):

.TRUE.

X.GE.3.14159

Single elements may be combined through use of logical operators to form compound logical expressions, such as:

```
TVAL.AND.INDEX
BOOL(M).OR.K.EQ.LIMIT
```

Any logical expression may be enclosed in parentheses and regarded as an element:

```
(T.XOR.S).AND.(R.EQV.Q)
PARITY ((2.GT.Y.OR.X.GE.Y).AND.NEVER)
```

Any logical expression may be preceded by the unary operator .NOT. as in:

```
.NOT.T
.NOT.X+7.GR.Y+Z
BOOL(K).AND..NOT.(TVAL.OR.R)
```

No two logical operators may appear in sequence, except in the case where .NOT. appears as the second of two logical operators, as in the example above.

Two decimal points may appear in sequence, as in the example above, or when one belongs to an operator and the other to a constant.

When the precedence of operators is not given explicitly by parentheses, it is understood to be as follows (in order of decreasing precedence):

```
**
*,/
+,-
.GT.,.GE.,.LT.,.LE.,.EQ.,.NE.
.NOT.
.AND.
.OR.
.EQV., .XOR.
```

For example, the logical expression

```
.NOT.ZETA**2+Y*MASS.GT.K-2.OR.PARITY.AND.X.EQ.Y
```

is interpreted as

```
(.NOT.(((ZETA**2)+(Y*MASS)).GT.(K-2))).OR.(PARITY.AND.(X.EQ.Y))
```

CHAPTER 3

THE ARITHMETIC STATEMENT

One of the key features of FORTRAN IV is the ease with which arithmetic computations can be coded. Computations to be performed by FORTRAN IV are indicated by arithmetic statements, which have the general form:

A=B

where A is a variable, B is an expression, and = is a replacement operator. The arithmetic statement causes the FORTRAN IV object program to evaluate the expression B and assign the resultant value to the variable A. Note that the = sign signifies replacement, not equality. Thus, expressions of the form:

A=A+B and
A=A*B

are quite meaningful and indicate that the value of the variable A is to be changed.

EXAMPLES:
```
Y=1*Y
P=.TRUE.
X(N)=N*ZETA (ALPHA*M/PI)+(1.,-1.)
```

Table 2 indicates which type of expression may be equated to each type of variable in an arithmetic statement. D indicates that the assignment is performed directly (no conversion of any sort is done); R indicates that only the real part of the variable is set to the value of the expression (the imaginary part is set to zero); C means that the expression is converted to the type of the variable; and H means that only the high-order portion of evaluated expression is assigned to the variable.

The expression value is made to agree in type with the assignment variable before replacement occurs. For example, in the statement:

THETA=W*(ABETA+E)

if THETA is an integer and the expression is real, the expression value is truncated to an integer before assignment to THETA.

TABLE 2 ALLOWED ASSIGNMENT STATEMENTS

Variable	Expression				
	Real	Integer	Complex	Double Precision	Logical, Octal, or Literal Constant
Real	D	C	R,D	H,D	D
Integer	C	D	R,C	H,C	D
Complex	D,R,I	C,R,I	D	H,D,R,I	D,R,I
Double Precision	D,H,L	C,H,L	R,D,H,L	D	D,H,L
Logical	D	D	R,D	H,D	D

D - Direct Replacement

C - Conversion between integer and floating point

R - Real only

I - Set imaginary part to 0

H - High order only

L - Set low order part to 0

CHAPTER 4

CONTROL STATEMENTS

FORTRAN compiled programs normally execute statements sequentially in the order in which they were presented to the compiler. However, the following control statements are available to alter the normal sequence of statement execution: GO TO, IF, DO, PAUSE, STOP, END, CALL, RETURN. CALL and RETURN are used to enter and return from subroutines.

GO TO STATEMENT

The GO TO statement has three forms: unconditional, computed, and assigned.

Unconditional GO TO Statements

Unconditional GO TO statements are of the form:

GO TO n

where n is the number of an executable statement. Control is transferred to the statement numbered n.

Computed GO TO Statements

Computed GO TO statements have the form:

GO TO $(n_1, n_2, \ldots, n_k)$, i

where $n_1, n_2, \ldots, n_k$ are statement numbers, and i is an integer expression.

This statement transfers control to the statement numbered $n_1, n_2, \ldots, n_k$ if i has the value 1, 2, ..., k, respectively. If i exceeds the size of the list, or is zero, execution will proceed to the next executable statement.

For example, in the statement:

GO TO (20,10,5), K

the variable K acts as a switch, causing a transfer to statement 20 if K=1, to statement 10 if K=2, or to statement 5 if K=3.

Assigned GO TO Statement

Assigned GO TO statements have two forms:

GO TO k

and

GO TO k, $(n_1, n_2, n_3, \ldots)$

where k is a nonsubscripted integer variable and $n_1, n_2, \ldots n_k$ are statement numbers. Both forms of the assigned GO TO have the effect of transferring control to the statement whose number is currently associated with the variable k. This association is established through the use of the ASSIGN statement, the general form of which is:

ASSIGN i TO k

If more than one ASSIGN statement refers to the same integer variable name, the value assigned by the last executed statement is the current value.

```
EXAMPLES:     ASSIGN 21 TO INT        ASSIGN 1000 TO INT
                    .                        .
                    .                        .
                    .                        .
                GO TO INT          GO TO INT, (2,21,1000,310)
```

IF STATEMENT

IF statements have two forms in FORTRAN IV: numerical and logical.

Numerical IF Statements

Numerical IF statements are of the form:

IF (expression) n_1, n_2, n_3

where n_1, n_2, n_3 are statement numbers.

This statement transfers control to the statement numbered n_1, n_2, n_3 if the value of the numeric expression is less than, equal to, or greater than zero, respectively. The expression may not be complex.

```
EXAMPLES:          IF (ETA) 4,7,12
                   IF (KAPPA-L (10)) 20,14,14
```

Logical IF Statements

Logical IF statements have the form:

IF (expressions)S

where S is a complete statement.

The expression must be logical. S may be any executable statement other than a DO statement or another logical IF statement (see page 12).

If the value of the expression is .FALSE., control passes to the next sequential statement.

If the value of the expression is .TRUE., statement S is executed. After execution of S, control passes to the next sequential statement unless S is a numerical IF statement or a GO TO statement; in these cases, control is transferred as indicated.

If the expression is .TRUE. and S is a CALL statement, control is transferred to the next sequential statement upon return from the subroutine.

Numbers are present in the logical expression:

```
IF (B)Y=X*SIN(Z)
W=Y**2
```

If the value of B is .TRUE., the statements Y=X*SIN(Z) and W=Y**2 are executed in that order. If the value of B is .FALSE., the statement Y=X*SIN(Z) is not executed.

EXAMPLES:

```
IF (T.OR.S)X=Y+1
IF (Z.GT.X(K)) CALL SWITCH (S,Y)
IF (K.EQ.INDEX)GO TO 15
```

NOTE: Care should be taken in testing floating point numbers for equality in IF statements as rounding errors may cause unexpected results.

DO STATEMENT

The DO statement simplifies the coding of iterative procedures. DO statements are of the form:

$$\text{DO } n \ i=m_1,m_2,m_3$$

where n is a statement number, i is a nonsubscripted integer variable, and m_1,m_2,m_3 are any integer expressions. If m_3 is not specified, it is understood to be 1.

The DO statement causes the statements which follow, up to and including the statement numbered n, to be executed repeatedly. This group of statements is called the range of the DO statement. The integer variable i of the DO statement is called the index. The values of m_1, m_2, and m_3 are called, respectively, the initial, limit, and increment values of the index.

A zero increment (m_3) is not allowed. The increment may be negative if $m_1 \geq m_2$. If $m_1 \leq m_2$, the increment m_3 must be positive. The parameters m_1 and m_2 may have like or unlike signs as long as m_2 is always larger than m_3, or m_3 is always larger than m_2.

EXAMPLES:	Form	Restriction
	DO 10 I=1,5,2	
	DO 10 I=5,1,-1	
	DO 10 I=J,K,5	$J \leq K$
	DO 10 I=J,K,-5	$J \geq K$
	DO 10 L=I,J,-K	$I \leq J, K<0$ or $I \geq J, K>0$
	DO 10 L=I,J,K	$I \leq J, K>0$ or $I \geq J, K>0$

Initially, the statements of the range are executed with the initial value assigned to the index. This initial execution is always performed, regardless of the values of the limit and increment. After each execution of the range, the increment value is added to the value of the index and the result is compared with the limit value. If the value of the index is not greater than the limit value, the range is executed again using the new value of the index. When the increment value is negative, another execution will be performed if the new value of the index is not less than the limit value.

After the last execution of the range, control passes to the statement immediately following the range. This exit from the range is called the normal exit. Exit may also be accomplished by a transfer from within the range.

The range of a DO statement may include other DO statements, provided that the range of each contained DO statement is entirely within the range of the containing DO statement. That is, the ranges of two DO statements must intersect completely or not at all. A transfer into the range of a DO statement from outside the range is not allowed.

Within the range of a DO statement, the index is available for use as an ordinary variable. After a transfer from within the range, the index retains its current value and is available for use as a variable. The values of the initial, limit, and increment variables for the index and the index of the DO loop, may not be altered within the range of the DO statement.

The range of a DO statement must not end with a GO TO type statement or a numerical IF statement. A logical IF statement is allowed as the last statement of the range. In this case, control is transferred as follows. The range is considered ended when, and if, control would normally pass to the statement following the entire logical IF statement.

As an example, consider the sequences:

```
DO 5 K=1,4
5 IF(X(K).GT.Y(K))Y(K)=X(K)
6 ...
```

Statement 5 is executed four times whether the statement Y(K)=X(K) is executed or not.

Statement 6 is not executed until statement 5 has been executed four times.

EXAMPLES:

```
DO 22 L=1,30
DO 45 K=2,LIMIT,-3
DO 7 X=T,MAX,L
```

CONTINUE STATEMENT

The CONTINUE statement has the form:

```
CONTINUE
```

This statement is a dummy statement, used primarily as a target for transfers, particularly as the last statement in the range of a DO statement. For example, in the sequence:

```
      DO 7 K=START,END
        :
      IF (X(K))22,13,7
        :
7     CONTINUE
```

a positive value of X (K) begins another execution of the range. The CONTINUE provides a target address for the IF statement and ends the range of the DO statement.

PAUSE STATEMENT

The PAUSE statement enables the program to incorporate operator activity into the sequence of automatic events. The PAUSE statement assumes one of three forms:

PAUSE

PAUSE n

PAUSE 'xxxxx'

where n is an unsigned string of six or less octal digits, and 'xxxxx' is a literal message.

Execution of the PAUSE statement causes the message or the octal digits, if any, to be typed on the user's teletypewriter. Program execution may be resumed (at the next executable FORTRAN statement) from the console by typing "G," followed by a carriage return. Program execution may be terminated by typing "X," followed by carriage return.

EXAMPLE: PAUSE 167

PAUSE 'NOW IS THE TIME'

STOP STATEMENT

The STOP statement has the form:

STOP

The STOP statement terminates the program and returns control to the monitor system. (Termination of a program may also be accomplished by a CALL to the EXIT or DUMP subroutines.)

END STATEMENT

The END statement has the form:

END

The END statement informs the compiler to terminate compilation and must be the physically last statement of the program.

CHAPTER 5

INPUT/OUTPUT STATEMENTS

Input/output statements are used to control the transfer of data between computer memory and peripheral devices and to specify the format of the output data. Input/output statements may be divided into three categories, as follows:

1. Nonexecutable statements that enable conversions between internal form data within core memory and external form data (FORMAT), or specify lists of arrays and variables for input/output transfer (NAMELIST).

2. Statements that specify transmission of data between computer memory and I/O devices: READ, WRITE, PRINT, PUNCH, TYPE, ACCEPT.

3. Statements that control magnetic tape unit mechanisms: REWIND, BACKSPACE, END FILE, UNLOAD, SKIP RECORD.

NONEXECUTABLE STATEMENTS

The FORMAT statement enables the user to specify the form and arrangement of data on the selected external medium. The NAMELIST statement provides for conversion and input/output transmission of data without reference to a FORMAT statement.

FORMAT Statement

FORMAT statements may be used with any appropriate input/output medium. FORMAT statements are of the form:

$$n \text{ FORMAT } (S_1, S_2, \ldots S_n/S_1^1, S_2^1, \ldots, S_n^1/\ldots)$$

where n is a statement number, and each S is a data field specification.

FORMAT statements may be placed anywhere in the source program. Unless the FORMAT statement contains only alphanumeric data for direct input/output transmission, it will be used in conjunction with the list of a data transmission statement.

Slashes are used to specify unit records, which must be one of the following:

1. A tape record with a maximum length corresponding to the printed line of the off-line printer.

2. A punched card with a maximum of 80 characters.

3. A printed line with a maximum of 72 characters for a Teletype and either 120 or 132 characters for the line printer.

During transmission of data, the object program scans the designated FORMAT statement. If a specification for a numeric field is present (see "Input/Output Lists" p 35) and the data transmission statement contains items remaining to be transmitted; transmission takes place according to the specification. This process ceases and execution of the data transmission statement is terminated as soon as all specified items have been transmitted. Thus, the FORMAT statement may contain specifications for more items than are specified by the data transmission statement. Conversely, the FORMAT statement may contain specifications for fewer items than are specified by the data transmission statement.

The following types of field specifications may appear in a FORMAT statement: numeric, numeric with scale factors, logical, alphanumeric. The FORMAT statement also provides for handling multiple record formats, formats stored as data, carriage control, skipping characters, blank insertion, and repetition. If an input list requires more characters than the input device supplies for a given unit record, blanks are supplied.

Numeric Fields

Numeric field specification codes and the corresponding internal and external forms of the numbers are listed in Table 3.

The conversions of Table 3 are specified by the forms:

1. Dw.d
2. Ew.d
3. Fw.d
4. Iw
5. Ow
6. Gw.d (for real)
   Gw (for integer or logical)
   Gw.d,Gw.d (for complex)

respectively. The letter D, E, F, I, O, or G designates the conversion type; w is an integer specifying the field width, which may be greater than required to provide for blank columns between numbers; d is an integer specifying the number of decimal places to the right of the decimal point or, for G conversion, the number of significant digits. (For D, E, F, and G input, the position of the decimal point in the external field takes precedence over the value of d in the format.)

For example,

FORMAT (I5,F10.2,D18.10)

could be used to output the line,

bbb32bbbb–17.60bbb.5962547681D+03

on the output listing.

The field width w should always be large enough to include spaces for the decimal point, sign, and exponent. In all numeric field conversions if w is not large enough to accommodate the converted number, the excess digits on the left will be lost; if the number is less than w spaces in length, the number is right-adjusted in the field.

TABLE 3 NUMERIC FIELD CODES

Conversion Code	Internal Form	External Form
D	Binary floating point double-precision	Decimal floating point with D exponent
E	Binary floating point	Decimal floating point with E exponent
F	Binary floating point	Decimal fixed point
I	Binary integer	Decimal integer
O	Binary integer	Octal Integer
G	One of the following: single precision binary floating point, binary integer, binary logical, or binary complex	Single precision decimal floating point, integer, logical (T or F), or complex (two decimal floating point numbers), depending upon the internal form

Numeric Fields with Scale Factors

Scale factors may be specified for D, E, F, and G conversions. A scale factor is written nP where P is the identifying character and n is a signed or unsigned integer that specifies the scale factor.

For F type conversions (or G type, if the external field is decimal fixed point), the scale factor specifies a power of ten so that

$$\text{external number} = (\text{internal number}) * 10^{(\text{scale factor})}$$

For D, E, and G (external field not decimal fixed point) conversions, the scale factor multiplies the number by a power of ten, but the exponent is changed accordingly leaving the number unchanged except in form. For example, if the statement:

FORMAT (F8.3,E16.5)

corresponds to the line

bb26.451bbbb–0.41321E–01

then the statement

FORMAT (–1PF8.3,2PE16.5)

might correspond to the line

bbb2.645bbb–41.32157E–03

In input operations, F type (and G type, if the external field is decimal fixed point) conversions are the only types affected by scale factors.

When no scale factor is specified, it is understood to be zero. However, once a scale factor is specified, it holds for all subsequent D, E, F, and G type conversions within the same format unless another scale factor is encountered. The scale factor is reset to zero by specifying a scale factor of zero. Scale factors have no effect on I and O type conversions.

Logical Fields

Logical data can be transmitted in a manner similar to numeric data by use of the specification:

Lw

where L is the control character and w is an integer specifying the field width. The data is transmitted as the value of a logical variable in the input/output list.

If on input, the first nonblank character in the data field is T or F, the value of the logical variable will be stored as true or false, respectively. If the entire data field is blank, a value of false will be stored.

On output, w minus 1 blanks followed by T or F will be output if the value of the logical variable is true or false, respectively.

Variable Field Width

The D, E, F, G, I, and O conversion types may appear without the specification of the field width w. In the case of input, omitting the w implies that the numeric field is delimited by any character which would otherwise be illegal in the field in addition to the characters –, +, ., E, D, and blank provided they follow the numeric field. For example, input according to the format:

```
10 FORMAT(2I,F,E,O)
```

might appear as:

```
-10,3/15.621-.0016E-10,777.
```

On output, omitting the w has the following effect:

Format	Becomes
D	D25.16
E	E15.7
F	F15.7
G	G15.7 or G25.16
I	I15
O	O15

Alphanumeric Fields

Alphanumeric data can be transmitted in a manner similar to numeric data by use of the form Aw, where A is the control character and w is the number of characters in the field. The alphanumeric characters are transmitted as the value of a variable in an input/output list. The variable may be of any type. For the sequence:

```
READ 5,V
5 FORMAT (A4)
```

causes four characters to be read and placed in memory as the value of the variable V.

Although w may have any value, the number of characters transmitted is limited by the maximum number of characters which can be stored in the space allotted for the variable. This maximum depends upon the variable type. For a double precision variable the maximum is ten characters; for all other variables, the maximum is five characters. If w exceeds the maximum, the leftmost characters are lost on input and replaced with blanks on output. If, on input, w is less than the maximum, blanks are filled in to the right of the given characters until the maximum is reached. If, on output, w is less than the maximum, the leftmost w characters are transmitted to the external medium. Since for complex variables each word requires a separate field specification, the maximum value for w is 5.

For example,	COMPLEX C	Could be used to transmit
	ACCEPT 1, C	ten alphanumeric characters
	1 FORMAT (2A5)	into complex variable C.

Alphanumeric Data Within Format Statements

Alphanumeric data may be transmitted directly into or from the format statement by two different methods: H-conversion, or the use of single quotes.

In H-conversion, the alphanumeric string is specified by the form nH. H is the control character and n is the number of characters in the string counting blanks. For example, the format in the statement below can be used to print PROGRAM COMPLETE on the output listing.

```
FORMAT (17H PROGRAM COMPLETE)
```

Referring to this format in a READ statement would cause the 17 characters to be replaced with a new string of characters.

The same effect is achieved by merely enclosing the alphanumeric data in quotes. The result is the same as in H-conversion; on input, the characters between the quotes are replaced by input characters, and, on output, the characters between the quotes (including blanks) are written as part of the output data. A quote character within the data is represented by two successive quote marks. For example, referring to:

```
FORMAT (' DON''T')
```

with an output statement would cause DON'T to be printed. The first character referenced by the FORMAT statement for output is interpreted as a carriage control character.

Mixed Fields

An alphanumeric format field may be placed among other fields of the format. For example, the statement:

```
FORMAT (I5,7H FORCE=F10.5)
```

can be used to output the line:

```
bbb22bFORCE=bb17.68901
```

The separating comma may be omitted after an alphanumeric format field, as shown above.

Complex Fields

Complex quantities are transmitted as two independent real quantities. The format specification consists of two successive real specifications or one repeated real specification. For instance, the statement:

FORMAT (2E15.4,2(F8.3,F8.5))

could be used in the transmission of three complex quantities.

Repetition of Field Specifications

Repetition of a field specification may be specified by preceding the control character D, E, F, I, O, G, L, or A by an unsigned integer giving the number of repetitions desired. For example:

FORMAT (2E12.4,3I5)

is equivalent to:

FORMAT (E12.4,E12.4,I5,I5,I5)

Repetition of Groups

A group of field specifications may be repeated by enclosing the group in parentheses and preceding the whole with the repetition number. For example:

FORMAT (2I8,2(E15.5,2F8.3))

is equivalent to:

FORMAT (2I8,E15.5,2F8.3,E15.5,2F8.3)

Multiple Record Formats

To handle a group of input/output records where different records have different field specifications, a slash is used to indicate a new record. For example, the statement:

FORMAT (3O8/I5,2F8.4)

is equivalent to:

FORMAT (3O8)

for the first record and

FORMAT (I5,2F8.4)

for the second record.

The separating comma may be omitted when a slash is used. When n slashes appear at the end or beginning of a format, n blank records may be written on output or records skipped on input. When n slashes appear in the middle of a format, n-1 blank records are written or n-1 records skipped.

Both the slash and the closing parenthesis at the end of the format indicate the termination of a record. If the list of an input/output statement dictates that transmission of data is to continue after the closing parenthesis of the format is reached, the format is repeated starting with that group repeat specification terminated by the last right parenthesis of level one or level zero if no level one group exists.

Thus, the statement:

```
           FORMAT (F7.2,(2(E15.5,E15.4),I7))
                  ↑      ↑                 ↑↑
   level 0 ───────┘      │        level 1 ─┘└─── level 0
              level 1 ───┘
```

causes the format:

F7.2,2(E15.5,E15.4),I7

to be used on the first record, and the format:

2(E15.5,E15.4),I7

to be used on succeeding records.

As a further example, consider the statement:

FORMAT (F7.2/(2(E15.5,E15.4),I7))

The first record has the format:

F7.2

and successive records have the format:

2(E15.5,E15.4),I7

Formats Stored as Data

The ASCII character string comprising a format specification may be stored as the values of an array. Input/output statements may refer to the format by giving the array name, rather than the statement number of a FORMAT statement. The stored format has the same form as a FORMAT statement excluding the word "FORMAT." The enclosing parentheses are included.

As an example, consider the sequence:

```
      DIMENSION SKELETON (2)
      READ 1, (SKELETON(I), I = 1,2)
1     FORMAT (2A4)
      READ SKELETON,K,X
```

The first READ statement enters the ASCII string into the array SKELETON. In the second READ statement, SKELETON is referred to as the format governing conversion of K and X.

Carriage Control

The first character of each ASCII record controls the spacing of the line printer or Teletype. This character is usually set by beginning a FORMAT statement for an ASCII record with 1Ha, where a is the desired control character. The line spacing actions, listed below, occur before printing:

Character	Effect
space	skip to next line
0	skip a line
1	form feed - go to top of next page
+	suppress skipping - will overprint line
-	skip 2 lines
2	skip to next 1/2 of page
3	skip to next 1/3 of page
/	skip to next 1/6 of page
*	skip to next 1/10 of page
.	skip to next 1/20 of page
,	skip to next 1/30 of page

A $ (dollar sign) as a format field specification code suppresses the carriage-return at the end of the line.

Spacing

Input and output can be made to begin at any position within a FORTRAN record by use of the format code:

Tw

where T is the control character and w is an unsigned integer constant specifying the position in a FORTRAN record where the transfer of data is to begin.

For example,

2 FORMAT(T50,'BLACK'T30, 'WHITE')

would cause the following line to be printed:

Print Position 29	Print Position 49
↓	↓
WHITE	BLACK

For input, the statements:

```
1 FORMAT(T35,'MONTH')
READ (3,1)
```

would cause the first 34 characters of the input data to be skipped, and the next 5 characters would replace the characters M, O, N, T, and H in storage.

Blank or Skip Fields

Blanks may be introduced into an output record or characters skipped on an input record by use of the specification nX. The control character is X; n is the number of blanks or characters skipped and must be greater than zero. For example, the statement:

FORMAT (5H STEPI5,10X2HY=F7.3)

may be used to output the line:

bSTEPbbb28bbbbbbbbbbY=b–3.872

NAMELIST Statement

The NAMELIST statement, when used in conjunction with special forms of the READ and WRITE statements, provides a method for transmitting and converting data without using a FORMAT statement or an I/O list. The NAMELIST statement has the form:

$$NAMELIST/X_1/A_1,A_2,\ldots,A_i/X_2/B_1,B_2,\ldots,B_i\ldots/X_m/C_1,C_2,\ldots C_n$$

where the X's are NAMELIST names, and the A's, B's, and C's are variable or array names.

Each list or variable mentioned in the NAMELIST statement is given the NAMELIST name immediately preceding the list. Thereafter, an I/O statement may refer to an entire list by mentioning its NAMELIST name. For example:

NAMELIST/FRED/A,B,C/MARTHA/D,E

states that A, B, and C belong to the NAMELIST name FRED, and D and E belong to MARTHA.

The use of NAMELIST statements must obey the following rules:

1. A NAMELIST name may not be longer than six characters; it must start with an alphabetic character; it must be enclosed in slashes; it must precede the list of entries to which it refers; and it must be unique within the program.

2. A NAMELIST name may be defined only once and must be defined by a NAMELIST statement. After a NAMELIST name has been defined, it may only appear in READ or WRITE statements. The NAMELIST name must be defined in advance of the READ or WRITE statement.

3. A variable used in a NAMELIST statement cannot be used as a dummy argument in a subroutine definition.

4. Any dimensioned variable contained in NAMELIST statement must have been defined in a DIMENSION statement preceding the NAMELIST statement.

Input Data for NAMELIST Statements

When a READ statement refers to a NAMELIST name, the first character of all input records is ignored. Records are searched until one is found with a $ or & as the second character immediately followed by the NAMELIST name specified. Data is then converted and placed in memory until the end of a data group is signaled by a $ or & either in the same record as the NAMELIST name, or in any succeeding record as long as the $ or & is the second character of the record. Data items must be separated by commas and be of the following form:

$$V=K_1, K_2, \ldots, K_n$$

where V may be a variable name or an array name, with or without subscripts. The K's are constants which may be integer, real, double precision, complex (written as (A, B) where A and B are real), or logical (written as T or .TRUE., and F or .FALSE). A series of J identical constants may be represented by J*K where J is an unsigned integer and K is the repeated constant. Logical and complex constants must be equated to logical and complex variables, respectively. The other types of constants (real, double precision, and integers) may be equated to any other type of variable (except logical or complex), and will be converted to the variable type. For example, assume A is a two-dimensional real array, B is a one-dimensional integer array, C is an integer variable, and that the input data is as follows:

```
$FRED A(7,2) =4, B =3,6*2.8, C =3.32$
↑
Column 2
```

A READ statement referring to the NAMELIST name FRED will result in the following: the integer 4 will be converted to floating point and placed in A(7,2). The integer 3 will be placed in B(1) and the floating point number 2.8 will be placed in B(2), B(3),..., B(7). The floating point number 3.32 will be converted to the integer 3 and placed in C.

Output Data for NAMELIST Statements

When a WRITE statement refers to a NAMELIST name, all variables and arrays and their values belonging to the NAMELIST name will be written out, each according to its type. The complete array is written out by columns. The output data will be written so that:

1. The fields for the data will be large enough to contain all the significant digits.
2. The output can be read by an input statement referencing the NAMELIST name.

For example, if JOE is a 2x3 array, the statements:

```
NAMELIST/NAM1/JOE,K1,ALPHA
WRITE (u,NAM1)
```

will generate the following form of output:

```
Column 2
↓
$NAM1
JOE =  -6.75,      .234E-04,        68.0,
       -17.8,           0.0,   -.197E+07,
  K1 = 73.1,       ALPHA=3,$
```

DATA TRANSMISSION STATEMENTS

The data transmission statements accomplish input/output transfer of data that may be listed in a NAMELIST statement or defined in a FORMAT statement. When a FORMAT statement is used to specify formats, the data transmission statement must contain a list of the quantities to be transmitted. The data appears on the external media in the form of records.

Input/Output Lists

The list of an input/output statement specifies the order of transmission of the variable values. During input, the new values of listed variables may be used in subscript or control expressions for variables appearing later in the list. For example:

```
READ 13,LA(L),B(L+1)
```

reads a new value of L and uses this value in the subscripts of A and B.

The transmission of array variables may be controlled by indexing similar to that used in the DO statement. The list of controlled variables, followed by the index control, is enclosed in parentheses. For example,

```
READ 7, (X(K),K=1,4),A
```

is equivalent to:

```
READ 7, X(1),X(2),X(3),X(4),A
```

As in the DO statement, the initial, limit, and increment values may be given as integer expressions:

```
READ 5, N, (GAIN(K),K=1,M/2,N)
```

The indexing may be compounded as in the following:

```
READ 11, ((MASS(K,L), K=1,4),L=1,5)
```

The above statement reads in the elements of array MASS in the following order:

```
MASS(1,1), MASS(2,1),...,MASS(4,1),MASS(1,2),...,MASS(4,5)
```

If an entire array is to be transmitted, the indexing may be omitted and only the array identifier written. The array is transmitted in order of increasing subscripts with the first subscript varying most rapidly. Thus, the example above could have been written:

```
READ 11, MASS
```

Entire arrays may also be designated for transmission by referring to a NAMELIST name (see description of NAMELIST statement).

Input/Output Records

All information appearing on external media is grouped into records. The maximum amount of information in one record and the manner of separation between records depends upon the medium. For punched cards, each card constitutes one record; on a teletypewriter a record is one line, and so forth. The amount of information contained in each ASCII record is specified by the FORMAT reference and the I/O list. For magnetic tape binary records, the amount of information is specified by the I/O list.

Each execution of an input or output statement initiates the transmission of a new data record. Thus, the statement:

```
READ 2, FIRST,SECOND,THIRD
```

is not necessarily equivalent to the statements:

```
READ 2, FIRST
READ 2, SECOND
READ 2, THIRD
```

since, in the second case, at least three separate records are required, whereas, the single statement:

```
READ 2, FIRST,SECOND,THIRD
```

may require one, two, three, or more records depending upon FORMAT2.

If an input/output statement requests less than a full record of information, the unrequested part of the record is lost and cannot be recovered by another input/output statement without repositioning the record.

If an input/output list requires more than one ASCII record of information, successive records are read.

PRINT Statement

The PRINT statement assumes one of two forms:

```
PRINT f, list
PRINT f
```

where f is a format reference.

The data is converted from internal to external form according to the designated format. If the data to be transmitted is contained in the specified FORMAT statement, the second form of the statement is used.

EXAMPLES:

```
PRINT 16,T,(B(K),K=1,M)
PRINT F106,SPEED,MISS
```

In the second example, the format is stored in array F106.

PUNCH Statement

The PUNCH statement assumes one of two forms:

PUNCH f, list
PUNCH f

where f is a format reference.

Conversion from internal to external data forms is specified by the format reference. If the data to be transmitted is contained in the designated FORMAT statement, the second form of the statement is used.

EXAMPLES:

```
PUNCH 12,A,B(A),C(B(A))
PUNCH 7
```

TYPE Statement

The TYPE statement assumes one of two forms:

TYPE f, list
TYPE f

where f is a format reference.

This statement causes the values of the variables in the list to be read from memory and listed on the user's teletypewriter. The data is converted from internal to external form according to the designated format. If the data to be transmitted is contained in the designated FORMAT statement, the second form of the statement is used:

EXAMPLES:

```
TYPE 14,K,(A(L),L=1,K)
TYPE FMT
```

WRITE Statement

The WRITE statement assumes one of the following forms:

WRITE(u,f) list
WRITE(u,f)
WRITE(u,N)
WRITE(u) list

where u is a unit designation, f is a format reference, and N is a NAMELIST name.

The first form of the WRITE statement causes the values of the variables in the list to be read from memory and written on the unit designated in ASCII form. The data is converted to external form as specified by the designated FORMAT statement.

The second form of the WRITE statement causes information to be read directly from the specified format and written on the unit designated in ASCII form.

The third form of the WRITE statement causes the names and values of all variables and arrays belonging to the NAMELIST name, N, to be read from memory and written on the unit designated. The data is converted to external form according to the type of each variable and array.

The fourth form of the WRITE statement causes the values of the variables in the list to be read from memory and written on the unit designated in binary form.

READ Statement

The READ statement assumes one of the following forms:

```
READ f, list
READ f
READ(u,f) list
READ(u,f)
READ(u,N)
READ(u) list
```

where f is a format reference, u is a unit designation, and N is a NAMELIST name.

The first form of the READ statement causes information to be read from cards† and put in memory as values of the variables in the list. The data is converted from external to internal form as specified by the referenced FORMAT statement.

EXAMPLE: READ 28,Z1,Z2,Z3

The second form of the READ statement is used if the data read from cards is to be transmitted directly into the specified format.

EXAMPLE: READ 10

† See Appendix 4

The third form of the READ statement causes ASCII information to be read from the unit designated and stored in memory as values of the variables in the list. The data is converted to internal form as specified by the referenced FORMAT statement.

EXAMPLE: READ(1,15)ETA,P1

The fourth form of the READ statement causes ASCII information to be read from the unit designated and transmitted directly into the specified format.

EXAMPLE: READ(N,105)

The fifth form of the READ statement causes data of the form described in the discussion of input data for NAMELIST statements to be read from the unit designated and stored in memory as values of the variables or arrays specified.

EXAMPLE: READ(2,FRED)

The sixth form of the READ statement causes binary information to be read from the unit designated and stored in memory as values of the variables in the list.

EXAMPLE: READ(M)GAIN,Z,AI

ACCEPT Statement

The ACCEPT statement assumes one of two forms:

ACCEPT f, list
ACCEPT f

where f is a format reference.

This statement causes information to be input from the user's teletypewriter and put in memory as values of the variables in the list. The data is converted to internal form as specified by the format. If the transmission of data is directly into the designated format, the second form of the statement is used.

EXAMPLES: ACCEPT 12,ALPHA,BETA
ACCEPT 27

DEVICE CONTROL STATEMENTS

Device control statements and their corresponding effects are listed in Table 4.

TABLE 4 DEVICE CONTROL STATEMENTS

Statement	Effect
BACKSPACE u	Backspaces designated tape one ASCII record or one logical binary record.
END FILE u	Writes an end-of-file.
REWIND u	Rewinds tape on designated unit.
SKIP RECORD u	Causes skipping of one ASCII record or one logical binary record.
UNLOAD u	Rewinds and unloads the designated tape.

CHAPTER 6

SPECIFICATION STATEMENTS

Specification statements allocate storage and furnish information about variables and constants to the compiler. Specification statements may be divided into three categories, as follows:

1. Storage specification statements: DIMENSION, COMMON, and EQUIVALENCE.
2. Data specification statements: DATA and BLOCK DATA.
3. Type declaration statements: INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, SUBSCRIPT INTEGER, and IMPLICIT.

<u>The following specification statements, if used, appear in the program prior to any executable statement:</u>

DIMENSION statement
EXTERNAL statement[†]
COMMON statement
EQUIVALENCE statement
Type declaration statements
Arithmetic function definition statements[†]
DATA statements
IMPLICIT statements

In addition, arrays must be dimensioned before being referenced in a NAMELIST or DATA statement.

STORAGE SPECIFICATION STATEMENTS

DIMENSION Statement

The DIMENSION statement is used to declare identifiers to be array identifiers and to specify the number and bounds of the array subscripts. The information supplied in a DIMENSION statement is required for the allocation of memory for arrays. Any number of arrays may be declared in a single DIMENSION statement. The DIMENSION statement has the form:

$$\text{DIMENSION } S_1, S_2, \ldots, S_n$$

where S is an array specification.

[†]EXTERNAL and arithmetic function definition statements are described in Chapter 7.

Each array variable appearing in the program must represent an element of an array declared in a DIMENSION statement, unless the dimension information is given in a COMMON or TYPE statement. Dimension information may appear only once for a given variable.

Each array specification gives the array identifier and the minimum and maximum values which each of its subscripts may assume in the following form:

identifier(min/max,min/max,...,min/max)

The minima and maxima must be integers. The minimum must not exceed the maximum.

For example, the statement:

DIMENSION EDGE(–1/1,4/8)

specifies EDGE to be a two-dimensional array whose first subscript may vary from –1 to 1 inclusive, and the second from 4 to 8 inclusive.

Minimum values of 1 may be omitted. For example,

NET(5,10)

is interpreted as:

NET(1/5,1/10)

EXAMPLES:
```
DIMENSION FORCE(-1/1,0/3,2,2,-7/3)
DIMENSION PLACE(3,3,3),JI(2,2/4),K(256)
```

Arrays may also be declared in the COMMON or type declaration statements in the same way:

```
COMMON X(10,4),Y,Z
INTEGER A(7,32),B
DOUBLE PRECISION K(-2/6,10)
```

Adjustable Dimensions

Within either a FUNCTION or SUBROUTINE subprogram, DIMENSION and TYPE statements may use integer variables in an array specification, provided that the array name and variable dimensions are dummy arguments of the subprogram. The actual array name and values for the dummy variables are given by the calling program when the subprogram is called. The variable dimensions may not be altered within the subprogram (i.e., typing the array DOUBLE PRECISION or COMPLEX after it has been dimensioned) and must be less than or equal to the explicit dimensions declared in the calling program.

```
EXAMPLE:      SUBROUTINE SBR(ARRAY,M1,M2,M3,M4)
              DIMENSION ARRAY (M1/M2,M3/M4)
                          .
                          .
                          .
              DO 27 L=M3,M4
              DO 27 K=M1,M2
                          .
                          .
                          .
         27   ARRAY(K,L)=VALUE
                          .
                          .
                          .
              END
```

The calling program for SBR might be:

```
              DIMENSION A1(10,20),A2(1000,4)
                          .
                          .
                          .
              CALL SBR(A1,5,10,10,20)
                          .
                          .
                          .
              CALL SBR(A2,100,250,2,4)
                          .
                          .
                          .
              END
```

COMMON Statement

The COMMON statement causes specified variables or arrays to be stored in an area available to other programs. By means of COMMON statements, the data of a main program and/or the data of its subprograms may share a common storage area.

The common area may be divided into separate blocks which are identified by block names. A block is specified as follows:

/block identifier/identifier, identifier, . . . , identifier

The identifier enclosed in slashes is the block name. The identifiers which follow are the names of the variables or arrays assigned to the block and are placed in the block in the order in which they appear in the block specification. A common block may have the same name as a variable in the same program or as any subroutine or function name in the same job.

The COMMON statement has the general form:

COMMON/BLOCK1/A, B, C/BLOCK2/D, E, F/. . .

where BLOCK1,BLOCK2,... are the block names, and A,B,C,... are the variables to be assigned to each block. For example, the statement:

COMMON/R/X,Y,T/C/U,V,W,Z

indicates that the elements X,Y, and T are to be placed in block R in that order, and that U,V,W, and Z are to be placed in block C.

Block entries are linked sequentially throughout the program, beginning with the first COMMON statement. For example, the statements:

COMMON/D/ALPHA/R/A,B/C/S
COMMON/C/X,Y/R/U,V,W

have the same effect as the statement:

COMMON/D/ALPHA/R/A,B,U,V,W/C/S,X,Y

One block of common storage, referred to as blank common, may be left unlabeled. Blank common is indicated by two consecutive slashes. For example,

COMMON/R/X,Y//B,C,D

indicates that B, C, and D are placed in blank common. The slashes may be omitted when blank common is the first block of the statement:

COMMON B,C,D

Storage allocation for blocks of the same name begins at the same location for all programs executed together. For example, if a program contains

COMMON A,B/R/X,Y,Z

as its first COMMON statement, and a subprogram has

COMMON/R/U,V,W//D,E,F

as its first COMMON statement, the quantities represented by X and U are stored in the same location. A similar correspondence holds for A and D in blank common.

Common blocks may be any length provided that no program attempts to enlarge a given common block declared by a previously loaded program.

Array names appearing in COMMON statements may have dimension information appended if the arrays are not declared in DIMENSION or type declaration statements. For example,

COMMON ALPHA,T(15,10,5),GAMMA

specifies the dimensions of the array T while entering T in blank common. Variable dimension array identifiers may not appear in a COMMON statement, nor may other dummy identifiers.

Each array name appearing in a COMMON statement must be dimensioned somewhere in the program containing the COMMON statement.

EQUIVALENCE Statement

The EQUIVALENCE statement causes more than one variable within a given program to share the same storage location. The EQUIVALENCE statement has the form:

$$\text{EQUIVALENCE}(V_1, V_2, \ldots), (V_k, V_{k+1}, \ldots), \ldots$$

where the V's are variable names.

The inclusion of two or more references in a parenthetical list indicates that the quantities in the list are to share the same memory location. For example,

EQUIVALENCE(RED,BLUE)

specifies that the variables RED and BLUE are stored in the same place.

The relation of equivalence is transitive; e.g., the two statements,

EQUIVALENCE(A,B),(B,C)
EQUIVALENCE(A,B,C)

have the same effect.

The subscripts of array variables must be integer constants.

EXAMPLE: EQUIVALENCE(X,A(3),Y(2,1,4)),(BETA(2,2),ALPHA)

EQUIVALENCE and COMMON

Identifiers may appear in both COMMON and EQUIVALENCE statements provided the following rules are observed.

1. No two quantities in common may be set equivalent to one another.

2. Quantities placed in a common block by means of EQUIVALENCE statements may cause the end of the common block to be extended.

For example, the statements:

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(A,Y)
```

causes the common block R to extend from X to A(4), arranged as follows:

```
X
Y A(1)      (same location)
Z A(2)      (same location)
  A(3)
  A(4)
```

3. EQUIVALENCE statements which cause extension of the start of a common block are not allowed. For example, the sequence:

```
COMMON/R/X,Y,Z
DIMENSION A(4)
EQUIVALENCE(X,A(3))
```

is not permitted, since it would require A(1) and A(2) to extend the starting location of block R.

DATA SPECIFICATION STATEMENTS

The DATA statement is used to specify initial or constant values for variables. The specified values are compiled into the object program, and become the values assumed by the variables when program execution begins.

DATA Statement

The data to be compiled into the object program is specified in a DATA statement. The DATA statement has the form:

$$\text{DATA list}/d_1, d_2, \ldots/, \text{list}/d_k, d_{k+1}, \ldots/, \ldots$$

where each list is in the same form as an input/output list, and the d's are data items for each list.

Indexing may be used in a list provided the initial, limit, and increment (if any) are given as constants. Expressions used as subscripts must have the form:

$$c_1 * i \pm c_2$$

where c_1 and c_2 are integer constants and i is the induction variable. If an entire array is to be defined, only the array identifier need be listed. Variables in common may appear on the lists only if the DATA statement occurs in a BLOCK DATA subprogram.

The data items following each list correspond one-to-one with the variables of the list. Each item of the data specifies the value given to its corresponding variable.

Data items may be numeric constants, alphanumeric strings, octal constants, or logical constants. For example,

DATA ALPHA, BETA/5, 16.E-2/

specifies the value 5 for ALPHA and the value .16 for BETA.

Alphanumeric data is packed into words according to the data word size in the manner of A conversion; however, excess characters are not permitted. The specification is written as nH followed by n characters or is imbedded in single quotes.

Octal data is specified by the letter O or the character ", followed by a signed or unsigned octal integer of one to twelve digits.

Logical constants are written as .TRUE.,.FALSE., T, or F.

EXAMPLE: DATA NOTE,K/7HRADIANS, O-7712/

Any item of the data may be preceded by an integer followed by an asterisk. The integer indicates the number of times the item is to be repeated. For example:

DATA(A(K),K=1, 20)/61E2, 19*32E1/

specifies 20 values for the array A; the value 6100 for A(1); the value 320 for A(2) through A(20).

BLOCK DATA Statement

The BLOCK DATA statement has the form:

BLOCK DATA

This statement declares the program which follows to be a data specification subprogram. Data may be entered into common only.

The first statement of the subprogram must be the BLOCK DATA statement. The subprogram may contain only the declarative statements associated with the data being defined.

EXAMPLE:
```
BLOCK DATA
COMMON/R/S,Y/C/Z,W,V
DIMENSION Y(3)
COMPLEX Z
DATA Y/1E-1,2*3E2/,X,Z/11.877D0,(-1.41421,1.41421)/
END
```

Data may be entered into more than one block of common in one subprogram.

TYPE DECLARATION STATEMENTS

The type declaration statements INTEGER, REAL, DOUBLE PRECISION, COMPLEX, LOGICAL, IMPLICIT, and SUBSCRIPT INTEGER are used to specify the type of identifiers appearing in a program. An identifier may appear in only one type statement. Type statements may be used to give dimension specifications for arrays.

The explicit type declaration statements have the general form:

type identifier,identifier,identifier...

where type is one of the following:

INTEGER,REAL,DOUBLE PRECISION,COMPLEX,LOGICAL,
SUBSCRIPT INTEGER

The listed identifiers are declared by the statement to be of the stated type. Fixed-point variables in a SUBSCRIPT INTEGER statement must fall between -2^{-27} and 2^{27}.

IMPLICIT Statement

The IMPLICIT statement has the form:

IMPLICIT $type_1(a_1,a_2,\ldots),\ldots,type_2(a_3,a_4,\ldots)$

where type represents one of the following: INTEGER, REAL, LOGICAL, COMPLEX, DOUBLE PRECISION; and $a_1a_2, \ldots$ represent single alphabetic characters, each separated by commas, or a range of characters (in alphabetic sequence) denoted by the first and last characters of the range separated by a minus sign (e.g., (A–D)).

This statement causes any program variable which is not mentioned in a type statement, and whose first character is one of those listed, to be typed according to the type appearing before the list in which the character appears. As an example, the statement:

IMPLICIT REAL(A–D,L,N–P)

causes all variables starting with the letters A through D,L, and N through P to be typed as real, unless they are explicitly declared otherwise.

The initial state of the compiler is set as if the statement

IMPLICIT REAL(A–H,O–Z), INTEGER(I–N)

were at the beginning of the program. This state is in effect unless an IMPLICIT statement changes the above interpretation; i.e., identifiers, whose types are not explicitly declared, are typed as follows:

1. Identifiers beginning with I, J, K, L, M, or N are assigned integer type.
2. Identifiers not assigned integer type are assigned real type.

If the program contains an IMPLICIT statement, this statement will override throughout the program the implicit state initially set by the compiler. No program may contain more than one IMPLICIT declaration for the same letter.

CHAPTER 7

SUBPROGRAM STATEMENTS

FORTRAN subprograms may be either internal or external. Internal subprograms are defined and may be used only within the program containing the definition. The arithmetic function definition statement is used to define internal functions.

External subprograms are defined separately from (i.e., external to) the programs that call them, and are complete programs which conform to all the rules of FORTRAN programs. They are compiled as closed subroutines; i.e., they appear only once in the object program regardless of the number of times they are used. External subprograms are defined by means of the statements FUNCTION and SUBROUTINE.

Dummy Identifiers

Subprogram definition statements contain dummy identifiers, representing the arguments of the subprogram. They are used as ordinary identifiers within the subprogram definition and indicate the sort of arguments that may appear and how the arguments are used. The dummy identifiers are replaced by the actual arguments when the subprogram is executed.

Library Subprograms

The standard FORTRAN IV library for the PDP-10 includes built-in functions, FUNCTION subprograms, and SUBROUTINE subprograms, listed and described in Appendixes 1, 2, and 3, respectively. Built-in functions are open subroutines; that is, they are incorporated into the object program each time they are referred to by the source program. FUNCTION and SUBROUTINE subprograms are closed subroutines; their names derive from the types of subprogram statements used to define them.

ARITHMETIC FUNCTION DEFINITION STATEMENT

The arithmetic function definition statement has the form:

```
identifier(identifier,identifier,...)=expression
```

This statement defines an internal subprogram. The entire definition is contained in the single statement. The first identifier is the name of the subprogram being defined.

Arithmetic function subprograms are single-valued functions with at least one argument. The type of the function is determined by the type of the function identifier.

The identifiers enclosed in parentheses represent the arguments of the function. These are dummy identifiers; they may appear only as scalar variables in the defining expression. Dummy identifiers have meaning and must be unique only within the defining statement. Dummy identifiers must agree in order, number, and type with the actual arguments given at execution time.

Identifiers, appearing in the defining expression, which do not represent arguments are treated as ordinary variables. The defining expression may include external functions or other previously defined arithmetic statement functions.

All arithmetic function definition statements must precede the first executable statement of the program.

EXAMPLES:

```
SSQR(K)=K*(K+1)*(2*K+1)/6
ACOSH(X)=(EXP(X/A)+EXP(-X/A))/2
```

In the last example above, X is a dummy identifier and A is an ordinary identifier. At execution time, the function is evaluated using the current value of the quantity represented by A.

FUNCTION SUBPROGRAMS

A FUNCTION subprogram is a single-valued function that may be called by using its name as a function name in an arithmetic expression, such as FUNC(N), where FUNC is the name of the subprogram that evaluates the corresponding function of the argument N. A FUNCTION subprogram begins with a FUNCTION statement and ends with an end statement. It returns control to the calling program by means of one or more RETURN statements.

FUNCTION Statement

The FUNCTION statement has the form:

```
FUNCTION identifer(argument,argument,...)
```

This statement declares the program which follows to be a function subprogram. The identifier is the name of the function being defined. This identifier must appear as a scalar variable and be assigned a value during execution of the subprogram which is the function value.

Arguments appearing in the list enclosed in parentheses are dummy arguments representing the function argument. The arguments must agree in number, order, and type with the actual arguments used in the calling program. Function subprogram arguments may be expressions, alphanumeric strings, array names, or subprogram names.

Dummy arguments may appear in the subprogram as scalar identifiers, array identifiers, or subprogram identifiers. A function must have at least one dummy argument. Dummy arguments representing array names must appear within the subprogram in a DIMENSION statement, or one of the type statements that provide dimension information. Dimensions given as constants must equal the dimensions of the corresponding arrays in the calling program. In a DIMENSION statement, dummy identifiers may be used to specify adjustable dimensions for array name arguments. For example, in the statement sequence:

```
FUNCTION TABLE(A,M,N,B,X,Y)
     :
DIMENSION A(M,N),B(10),C(50)
```

The dimensions of array A are specified by the dummies M and N, while the dimension of array B is given as a constant. The various values given for M and N by the calling program must be those of the actual arrays which the dummy A represents. The arrays may each be of different size but must have two dimensions. The arrays are dimensioned in the programs that use the function.

Dummy dimensions may be given only for dummy arrays. In the example above the array C must be given absolute dimensions, since C is not a dummy identifier. A dummy identifier may not appear in an EQUIVALENCE statement in the function subprogram.

A function must not modify any arguments which appear in the FORTRAN arithmetic expression calling the function. Modification of implicit arguments from the calling program, such as variables in common and DO loop indexes, is not allowed. The only FORTRAN statements not allowed in a function subprogram are SUBROUTINE, BLOCK DATA, and another FUNCTION statement.

Function Type

The type of the function is the type of identifier used to name the function. This identifier may be typed, implicitly or explicitly, in the same way as any other identifier. Alternatively, the function may be explicitly typed in the FUNCTION statement itself by replacing the word FUNCTION with one of the following:

```
INTEGER FUNCTION
REAL FUNCTION
COMPLEX FUNCTION
LOGICAL FUNCTION
DOUBLE PRECISION FUNCTION
```

For example, the statement:

COMPLEX FUNCTION HPRIME(S,N)

is equivalent to the statements:

FUNCTION HPRIME(S,N)
COMPLEX HPRIME

EXAMPLES:
FUNCTION MAY(RANGE,EP,YP,ZP)
COMPLEX FUNCTION COT(ARG)
DOUBLE PRECISION FUNCTION LIMIT(X,Y)

SUBROUTINE SUBPROGRAMS

A subroutine subprogram may be multivalued and can be referred to only by a CALL statement. A subroutine subprogram begins with a SUBROUTINE statement and returns control to the calling program by means of one or more RETURN statements.

SUBROUTINE Statement

The SUBROUTINE statement has the form:

SUBROUTINE identifier(argument,argument,...)

This statement declares the program which follows to be a subroutine subprogram. The first identifier is the subroutine name. The arguments in the list enclosed in parentheses are dummy arguments representing the arguments of the subprogram. The dummy arguments must agree in number, order, and type with the actual arguments used by the calling program.

Subroutine subprograms may have expressions, alphanumeric strings, array names, and subprogram names as arguments. The dummy arguments may appear as scalar, array, or subprogram identifiers.

Dummy identifiers which represent array names must be dimensioned within the subprogram by a DIMENSION or type declaration statement. As in the case of a function subprogram, either constants or dummy identifiers may be used to specify dimensions in a DIMENSION statement. The dummy arguments must not appear in an EQUIVALENCE or COMMON statement in the subroutine subprogram.

A subroutine subprogram may use one or more of its dummy identifiers to represent results. The subprogram name is not used for the return of results. A subroutine subprogram need not have any argument at all.

EXAMPLES: SUBROUTINE FACTOR(COEFF,N,ROOTS)
SUBROUTINE RESIDU(NUM,N,DEN,M,RES)
SUBROUTINE SERIES

The only FORTRAN statements not allowed in a function subprogram are FUNCTION, BLOCK DATA, and another SUBROUTINE statement.

CALL Statement

The CALL statement assumes one of two forms:

```
CALL identifier
CALL identifier(argument,argument,...,argument)
```

The CALL statement is used to transfer control to subroutine subprogram. The identifier is the subprogram name.

The arguments may be expressions, array identifiers, alphanumeric strings or subprogram identifiers; arguments may be of any type, but must agree in number, order, type, and array size (except for adjustable arrays, as discussed under the DIMENSION statement) with the corresponding arguments in the SUBROUTINE statement of the called subroutine. Unlike a function, a subroutine may produce more than one value and cannot be referred to as a basic element in an expression.

A subroutine may use one or more of its arguments to return results to the calling program. If no arguments at all are required, the first form is used.

EXAMPLES: CALL EXIT
CALL SWITCH(SIN,2.LE.BETA,X**4,Y)
CALL TEST(VALUE,123,275)

The identifier used to name the subroutine is not assigned a type and has no relation to the types of the arguments. Arguments which are constants or formed as expressions must not be modified by the subroutine.

RETURN Statement

The RETURN statement has the form:

```
RETURN
```

This statement returns control from a subprogram to the calling program. Normally, the last statement executed in a subprogram is a RETURN statement. Any number of RETURN statements may appear in a subprogram.

EXTERNAL Statement

Function and subroutine subprogram names may be used as the actual arguments of subprograms. Such subprogram names must be distinguished from ordinary variables by their appearance in an EXTERNAL statement. The EXTERNAL statement has the form:

EXTERNAL identifier, identifier,...,identifier

This statement declares the listed identifiers to be subprogram names. Any subprogram name given as an argument to another subprogram must appear in an external declaration in the calling program.

EXAMPLE:

```
EXTERNAL SIN,COS
         :
CALL TRIGF(SIN,1.5,ANSWER)
         :
CALL TRIGF(COS,.87,ANSWER)
         :
END

SUBROUTINE TRIGF(FUNC,ARG,ANSWER)
         :
ANSWER = FUNC(ARG)
         :
RETURN
END
```

To reference external variables from a MACRO-10 program, place the variables in named COMMON. Use the name of the variable as the name of the COMMON block:

```
COMMON /A/A /B/B (13) /C C(6,7)
```

APPENDIX 1

SUMMARY OF PDP-10 FORTRAN IV STATEMENTS

CONTROL STATEMENTS

General Form	Page References
ASSIGN i to m	18
CALL name $(a_1, a_2, \ldots)$	55
CONTINUE	21
DO i $m=m_1, m_2, m_3$	19
GO TO i	17
GO TO m	18
GO TO m, $(i_1, i_2, \ldots)$	18
GO TO $(i_1, i_2, \ldots)$, m	17
IF $(e_1) i_1, i_2, i_3$	18
IF (e_2)s	19
PAUSE	21
PAUSE j	22
PAUSE 'h'	22
RETURN	22
STOP	22
END	22

INPUT/OUTPUT STATEMENTS

General Form	Page References
ACCEPT f	39
ACCEPT f, list	39
BACKSPACE unit	40
END FILE unit	40
FORMAT (g)	23
PRINT f	36
PRINT f, list	36
PUNCH f	37

General Form	Page Reference
READ f	38
READ f, list	38
READ (unit, f)	38
READ (unit, f)list	38
READ (unit)list	38
READ (unit, name_1)	38
REWIND unit	40
SKIP RECORD unit	40
TYPE f	37
TYPE f, list	37
WRITE (unit, f)	37
WRITE (unit, f)list	37
WRITE (unit)list	37
WRITE (unit, name_1)	37
UNLOAD unit	40

SPECIFICATION STATEMENTS

General Form	Page Reference
COMMON $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	43
COMPLEX $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	48
DATA $t, u, \ldots / k_1, k_2, k_3, \ldots /$ $v, w, \ldots / k_4, k_5, k_6, \ldots / \ldots$	46
DIMENSION $a(n_1, n_2, \ldots), b(n_1, n_2, \ldots), \ldots$	41
DOUBLE PRECISION $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	48
EQUIVALENCE $(a(n_1, \ldots), b(n_2, \ldots), \ldots), \ldots$ $(c(n_3, \ldots), d(n_4, \ldots), \ldots), \ldots$	45
EXTERNAL $y, z, \ldots$	56
IMPLICIT $\text{type}_1(1_1-1_2), \text{type}_2(1_3-1_4), \ldots$	48
INTEGER $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	48
LOGICAL $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	48
NAMELIST $/\text{name}_1/a, b, \ldots /\text{name}_2/c, d, \ldots$	32
REAL $a(n_1, n_2, \ldots), b(n_3, n_4, \ldots), \ldots$	48
SUBSCRIPT INTEGER $a(n_1, n_2, \ldots), b(n_3, \ldots), \ldots$	48

ARITHMETIC STATEMENT FUNCTION DEFINITION

General Form	Page Reference
name(a,b,...)=e	51

NOTE:

$a_1, a_2, \ldots$	are expressions
a,b,c,d	are variable names
e	is an expression
e_1	is a noncomplex expression
e_2	is a logical expression
f	is a format number
g	is a format specification
'h'	is an alphanumeric string
$i, i_1, i_2, \ldots$	are statement numbers
j	is an integer constant
$k_1, k_2, \ldots$	are constants of the general form j*k where k is any constant
$l_1, l_2, \ldots$	are letters
list	is an input/output list
m	is an integer variable name
m_1, m_2, m_3	are integer expressions
$n_1, n_2, \ldots$	are dimension specifications
name	is a subroutine or function name
$name_1, name_2$	are NAMELIST names
s	is a statement (not DO or logical IF)
t,u,v,w	are variable names or input/output lists
$type_1, type_2, \ldots$	are type specifications
unit	is an integer variable or constant specifying a logical device number
y,z	are external subprogram names

APPENDIX 2

FORTRAN IV LIBRARY FUNCTIONS

This appendix contains descriptions of all standard function subprograms provided with the FORTRAN IV library for the PDP-10. These functions may be called by using the function mnemonic as a function name in an arithmetic expression.

Function	Definition	Number of Arguments	Name	Type of Argument	Type of Function
Absolute value	|Arg|	1	ABS	Real	Real
Absolute value	|Arg|	1	IABS	Integer	Integer
Absolute value	|Arg|	1	DABS	Double	Double
Truncation	Sign of Arg times largest integer ≤|Arg|	1	AINT	Real	Real
Truncation	Sign of Arg times largest integer ≤|Arg|	1	INT	Real	Integer
Truncation	Sign of Arg times largest integer ≤|Arg|	1	IDINT	Double	Integer
Remaindering	Arg_1 (mod Arg_2) †	2	AMOD	Real	Real
Remaindering	Arg_1 (mod Arg_2) †	2	MOD	Integer	Integer
Choosing largest value	Max(Arg_1,Arg_2,...)	≥2	AMAX0 AMAX1 MAX0 MAX1 DMAX1	Integer Real Integer Real Double	Real Real Integer Integer Double
Choosing smallest value	Min(Arg_1,Arg_2,...)	≥2	AMIN0 AMIN1 MIN0 MIN1 DMIN1	Integer Real Integer Real Double	Real Real Integer Integer Double

† The function MOD or AMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2]\ a_2$, where [x] is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as x.

Function	Definition	Number of Arguments	Name	Type of Argument	Type of Function
Transfer of sign	$Sgn(Arg_2) * \| Arg_1 \|$	2	SIGN	Real	Real
Transfer of sign	$Sgn(Arg_2) * \| Arg_1 \|$	2	ISIGN	Integer	Integer
Transfer of sign	$Sgn(Arg_2) * \| Arg_1 \|$	2	DSIGN	Double	Double
Positive difference	$Arg_1 - Min(Arg_1, Arg_2)$	2	DIM	Real	Real
Positive difference	$Arg_1 - Min(Arg_1, Arg_2)$	2	IDIM	Integer	Integer
Complex conjugate	For Arg=X+iY, C=X-iY	1	CONJG	Complex	Complex
Conversion from integer to real		1	FLOAT	Integer	Real
Conversion from real to integer	Result is largest integer $\leq$ a	1	IFIX	Real	Integer
Express single precision argument in double precision form, low order part = 0		1	DBLE	Real	Double
Express two real arguments in complex form	$C = Arg_1 + i * Arg_2$	2	CMPLX	Real	Complex
Obtain most significant part of double precision argument		1	SNGL	Double	Real
Obtain real part of complex argument		1	REAL	Complex	Real
Obtain imaginary part of complex argument		1	AIMAG	Complex	Real
Exponential	e^{Arg}	1	EXP	Real	Real

Function	Definition	Number of Arguments	Name	Type of Argument	Type of Function
Natural logarithm	$\log_e(\text{Arg})$	1	ALOG	Real	Real
Common logarithm	$\log_{10}(\text{Arg})$	1	ALOG10	Real	Real
Arc-sine	asin(Arg)	1	ASIN	Real	Real
Arc-cosine	acos(Arg)	1	ACOS	Real	Real
Arctangent	atan(Arg)	1	ATAN	Real	Real
Arctangent of the quotient of two arguments	$\text{atan}(\text{Arg}_1/\text{Arg}_2)$	2	ATAN2	Real	Real
Sine (radians)	sin(Arg)	1	SIN	Real	Real
Sine (degrees)	sin(Arg)	1	SIND	Real	Real
Cosine (radians)	cos(Arg)	1	COS	Real	Real
Cosine (degrees)	cos(Arg)	1	COSD	Real	Real
Hyperbolic tangent	tanh(Arg)	1	TANH	Real	Real
Hyperbolic sine	sinh(Arg)	1	SINH	Real	Real
Hyperbolic cosine	cosh(Arg)	1	COSH	Real	Real
Square root	$(\text{Arg})^{1/2}$	1	SQRT	Real	Real
Remaindering †	$\text{Arg}_1 (\text{mod } \text{Arg}_2)$	2	DMOD	Double	Double
Exponential	e^{Arg}	1	DEXP	Double	Double
Natural logarithm	$\log_e(\text{Arg})$	1	DLOG	Double	Double
Common logarithm	$\log_{10}(\text{Arg})$	1	DLOG10	Double	Double

†The function DMOD (a_1, a_2) is defined as $a_1 - [a_1/a_2]\, a_2$, where $[x]$ is the integer whose magnitude does not exceed the magnitude of x and whose sign is the same as the sign of x.

Function	Definition	Number of Arguments	Name	Type of Argument	Type of Function
Arctangent	atan(Arg)	1	DATAN	Double	Double
Arctangent of two arguments	$atan(Arg_1/Arg_2)$	2	DATAN2	Double	Double
Sine(radians)	sin(Arg)	1	DSIN	Double	Double
Cosine (radians)	cos(Arg)	1	DCOS	Double	Double
Square root	$(Arg)^{1/2}$	1	DSQRT	Double	Double
Absolute value	$C=(X^2+Y^2)^{1/2}$	1	CABS	Complex	Real
Exponential	e^{Arg}	1	CEXP	Complex	Complex
Natural logarithm	$log_e(Arg)$	1	CLOG	Complex	Complex
Complex sine	sin(Arg)	1	CSIN	Complex	Complex
Complex cosine	cos(Arg)	1	CCOS	Complex	Complex
Complex square root	$C=(X+iY)^{1/2}$	1	CSQRT	Complex	Complex

APPENDIX 3

FORTRAN IV LIBRARY SUBROUTINES

This appendix contains descriptions of all standard subroutine subprograms provided within the FORTRAN IV library for the PDP-10. These subprograms are closed subroutines and may be called with a CALL statement.

Subroutine Name	Effect
EXIT	Returns control to the monitor and, therefore, terminates the execution of the program.
DUMP	Causes particular portions of core to be dumped and is referred to in the following form: CALL DUMP $(L_1,U_1,F_1,\ldots,L_n,U_n,F_n)$ where L_i and U_i are the variable names which give the limits of core memory to be dumped. Either L_i or U_i may be upper or lower limits. F_i is a number indicating the format in which the dump is to be performed: 0=octal, 1=real, 2=integer, and 3=ASCII. If F is not 0,1,2,3, the dump is in octal. If F_n is missing, the last section is dumped in octal. If U_n and F_n are missing, an octal dump is made from L to the end of the job area. If L_n, U_n, and F_n are missing, the entire job area is dumped in octal. The dump is terminated by a call to EXIT.
PDUMP	Is referred to in the following form: CALL PDUMP$(L_1,U_1,F_1,\ldots,L_n,U_n,F_n)$ where the arguments are the same as those for DUMP. PDUMP is the same as DUMP except that control returns to the calling program after the dump has been executed.

Subroutine Name	Effect
SLITE(i)	Where i is an integer expression, turns sense lights on or off. For $1 \leq i \leq 36$ sense light i will be turned on. If i=0, all sense lights will be turned off.
SLITET(i,j)	Checks the status of sense light i and sets the variable j accordingly and turns off sense light i. If i is on, j is set to 1; and if i is off, j is set to 2.
SSWTCH(i,j)	Checks the status of data switch i($0 \leq i \leq 35$) and sets the variable j accordingly. If i is set down, j is set to 1; and, if i is up, j is set to 2.
OVERFL(j)	Checks the status of the AR OV flag and sets the variable j accordingly. If the AR OV flag is on, j is set to 1. If the flag is off, j is set to 2.

APPENDIX 4

PDP-10 FORTRAN IV OPERATING SYSTEM

SUBPROGRAM CALLING SEQUENCES

FORTRAN Subroutines

FORTRAN subroutine calling sequences appear as follows:

```
JSA  16,NAME
ARG  CODE1,A1
ARG  CODE2,A2
 .    .
 .    .
 .    .
```

where NAME is the name of the subroutine, ARG is a "pseudo-op" equivalent to a JUMP instruction (a "no-op"), $CODE_1$, $CODE_2$, etc. are 4-bit codes with the values:

0	Integer argument
1	Unused
2	Real argument
3	Logical argument
4	Octal argument
5	Literal argument
6	Double precision argument
7	Complex argument

and A_1, A_2, etc. are the argument addresses.

All accumulators are saved in subroutines except 0 for subroutines with single-word arguments and except 0 and 1 for subroutines with double-word arguments (high order or real part in 0 and low order or imaginary part in 1). All scalar arguments in a subroutine call are transferred into and restored from the subroutine by value.

FORTRAN Function Subprograms and Library Functions

The FORTRAN function calling sequence is the same as that for subroutines. The function value is returned in accumulator 0 or accumulators 0 and 1. Scalar arguments are not restored as their values may not be modified within a function.

INPUT/OUTPUT

In addition to the arithmetic functions, the PDP-10 FORTRAN IV library (LIB40) contains several subprograms which control FORTRAN IV input/output operations at run time. The input/output subprograms are compatible with the PDP-10 Monitors.

Logical-Physical Device Assignments

The logical and physical device assignments are controlled by a table called DEVTB.. The first entry in DEVTB. is the length of the table. Each entry after the first word is a sixbit ASCII device name corresponding in its position in the table to the FORTRAN logical number. For example, in Figure 2, magnetic tape 0 is the 16th entry in DEVTB.. Thus, the statement:

REWIND 16

would refer to magnetic tape 0.

The last five entries in DEVTB. correspond to the special FORTRAN statements READ, ACCEPT, PRINT, PUNCH, TYPE. Any device assignment may be changed by reassembling DEVTB..

DECtape and Disc Usage

DECtapes may be used for binary mode or ASCII (formatted) input/output.

Binary Mode

In binary mode, each block contains 127 data words, the first of which is a record control word of the form:

w	n

where w is the word count specifying the number of FORTRAN data words in the block (126 for a full block) and n is 0 in all but the last block of a logical record, in which case n is the number of blocks in the logical record. (A logical record contains all the data corresponding to one READ or WRITE statement.)

ASCII Mode

In ASCII mode, blocks are packed with as many full lines (a line is a unit record as specified by a format statement) as possible. Lines are not split across blocks.

```
                    DEVICE TABLE FOR FORTRAN IV
TITLE  DEVTB  V.017
SUBTTL  1-APR-69

ENTRY   DEVTB.,DEVND.,DEVLS.,DVTOT.
ENTRY   MTABF.,MBFBG.,TABPT.,TABP1.
ENTRY   MTACL.,DATTB.,NEG1.,NEG2.,NEG3.,NEG5.
P=17


DEVTB.: EXP       DEVND.-.        ;NO. OF ENTRIES
                                  ;LOGICAL#/FILENAME/DEVICE
        SIXBIT    .DSK.           ; 1 FOR01.DAT    DISC
CORPOS: SIXBIT    .CDR.           ; 2 FOR02.DAT    CARD READER
LPTPOS: SIXBIT    .LPT.           ; 3 FOR03.DAT    LINE PRINTER
        SIXBIT    .CTY.           ; 4 FOR04.DAT    CONSOLE TELETYPE
TTYPOS: SIXBIT    .TTY.           ; 5 FOR05.DAT    USER TELETYPE
        SIXBIT    .PTR.           ; 6 FOR06.DAT    PAPER TAPE READER
PTPPOS: SIXBIT    .PTP.           ; 7 FOR07.DAT    PAPER TAPE PUNCH
        SIXBIT    .DIS.           ; 8 FOR08.DAT    DISPLAY
        SIXBIT    .DTA1.          ; 9 FOR09.DAT    DECTAPE
        SIXBIT    .DTA2.          ; 10 FOR10.DAT
        SIXBIT    .DTA3.          ; 11 FOR11.DAT
        SIXBIT    .DTA4.          ; 12 FOR12.DAT
        SIXBIT    .DTA5.          ; 13 FOR13.DAT
        SIXBIT    .DTA6.          ; 14 FOR14.DAT
        SIXBIT    .DTA7.          ; 15 FOR15.DAT
        SIXBIT    .MTA0.          ; 16 FOR16.DAT   MAGNETIC TAPE
        SIXBIT    .MTA1.          ; 17 FOR17.DAT
        SIXBIT    .MTA2.          ; 18 FOR18.DAT
        SIXBIT    .FORTR.         ; 19 FORTR.DAT   ASSIGNABLE DEVICE, FORTR

        SIXBIT    .DSK0.          ; 20 FOR20.DAT   DISK
        SIXBIT    .DSK1.          ; 21 FOR21.DAT
        SIXBIT    .DSK2.          ; 22 FOR22.DAT
        SIXBIT    .DSK3.          ; 23 FOR23.DAT
        SIXBIT    .DSK4.          ; 24 FOR24.DAT
        SIXBIT    .DEV1.          ; 25 FOR25.DAT   ASSIGNABLE DEVICES
        SIXBIT    .DEV2.          ; 26 FOR26.DAT
        SIXBIT    .DEV3.          ; 27 FOR27.DAT
        SIXBIT    .DEV4.          ; 28 FOR28.DAT
DEVLS.: SIXBIT    .DEV5.          ; 29 FOR29.DAT    V.006
        SIXBIT    .REREAD.        ;        -6      REREAD
        SIXBIT    .CDR.           ;        -5      READ
        SIXBIT    .TTY.           ;        -4      ACCEPT
        SIXBIT    .LPT.           ;        -3      PRINT
        SIXBIT    .PTP.           ;        -2      PUNCH
DEVND.: SIXBIT    .TTY.           ;        -1      TYPE
```

Figure 2 Device Table for FORTRAN IV

File Names

File names may be declared for DECtapes or the disc through the use of the library subprograms IFILE and OFILE. In order to make an entry of the file name BOB on unit u, the following statements could be used:

BOB=3HBOB

CALL OFILE(u,BOB)

Similarly, the following statements might be used to open the file, RALPH, for reading:

```
RALPH=5HRALPH
CALL IFILE(u,RALPH)
```

After writing a file, the END FILE u statement must be given in order to close the current file and allow for reading or writing another file or for reading or rewriting the same file. If no call to IFILE or OFILE has been given before the execution of a READ or WRITE referencing DECtape or the disc the file name FORnn.DAT is assumed where nn is the FORTRAN logical number used in the I/O statement that references device nn.

The FORTRAN programmer can make logical assignments such that each device has its own unique file as intended, but each can be on the DSK. The programmer needs no additional knowledge to communicate with up to 17 I/O devices simultaneously. In order to use the devices available, the programmer can make assignments at run time and assign the DSK to those not available.

For example, the FORTRAN logical device numbers, e. g., 1=DSK, 2=CDR, 3=LPT, are used in the file name. The written file names are FOR01.DAT, FOR02.DAT, etc. The same is true for READ. For example, a WRITE (3, 1) A, B, C, in the FORTRAN program generates the file name FOR03.DAT on the DSK if the DSK has been assigned the logical name LPT prior to running the program. (Note: REREAD rereads from the file belonging to the device last referenced in a READ statement, not FOR-6.DAT, as usual.) The programmer must, of course, realize his own folly in assigning the DSK as the TTY in the case that FORSE tries to type out error messages or PAUSE messages.

More than one DSK File may be accessed, without making logical assignments at runtime, by using logical device numbers 1, and 20 through 24 in the FORTRAN program. Logical device numbers 25 through 29 refer to logical device names DEV1 through DEV5 which may be assigned to any device at runtime. Logical device number 19 refers to logical device FORTR which must be assigned at runtime and accesses file name FORTR.DAT to maintain compatibility with the past system of default file name FORTR.DAT. In all cases when the operating system fails to find a file specified, an attempt will be made to read from file FORTR.DAT as before.

The magnetic tape operation REWIND is simulated on DECtape or the disc. Thus, a program which uses READ, WRITE, END FILE, and REWIND for magnetic tape need only have the logical device number changed in order to perform the proper input/output sequences on DECtape or the disc.

Magnetic Tape Usage

The format of binary data on magnetic tape is similar to that for DECtape except that the physical record size depends on the magnetic tape buffer size assigned in the Time-Sharing Monitor. Normally, the buffer size is set at either 129 or 257 words so that either 128 or 256 word records are written (containing a control word and 127 or 255 FORTRAN data words).

ASCII Mode

The format for ASCII data is the same as that used on DECtape.

Backspacing and Skipping Records

Both the BACKSPACE u and SKIP RECORD u statements are executed on a logical basis for binary records and on a line basis for ASCII records.

A. Binary Mode

Both BACKSPACE and SKIP RECORD space magnetic tape physically over one (1) logical record; i.e., the result of one WRITE (u) statement.

B. ASCII Mode

ASCII records are packed, that is WRITE (u, f) statements do not cause physical writing on the tape until the output buffers are full or a BACKSPACE, END FILE, or REWIND command is executed by the program. BACKSPACE and SKIP RECORD on ASCII record space over one (1) line; i.e., cr --- cr.

C. BACKSPACE and SKIP RECORD following WRITE ASCII commands.

1. BACKSPACE closes the tape, writes 2 EOF's (tapemark) and backspaces over the last line.

2. SKIP RECORD cannot be used during a WRITE operation. This is an input function only.

Device Initialization, Release, and Buffering

The Operating System assigns I/O channel numbers from 1 to 17_8 to each new device in order to use. Devices are initialized only once and are released only through a CALL [SIXBIT/EXIT/] executed at the end of every FORTRAN main program.

All devices referenced by a FORTRAN program are double buffered.

REREAD

The reread feature allows a FORTRAN program to reread information from the last used input file. The format used during the reread need not correspond to the original read format, and the information may be reread as many times as desired.

A reread device of -6 has been added to the Device Table, DEVTB, (see page 69). The definition of the reread device may be altered as follows:

1. The DEVTB subprogram must be modified to contain a 6-bit device name (SIXBIT .REREAD.) at the desired position in the table.
2. The altered Device Table must be assembled.
3. The binary file must replace the present Device Table in the library file.

A. To reread from an input device, the following coding would be used:

```
N=-6
READ (16,100)A
    :
READ (N,105)A
```

where N is the number of the reread device in the DEVTB. table. The READ (N,105)A statement will cause the last input device used to be reread according to format statement 105. The original read format and a subsequent reread format need not be the same.

B. The reread feature cannot be used until an input from a file has been accomplished. If the feature is used prematurely, an error message will be generated.

C. Information may be reread as many times as desired using either the same or a new format statement each time.

D. The reread feature must be used with some forethought and care since it rereads from the <u>last</u> input file used, i.e.:

The following example will reread from the file on Device No. 10, <u>not</u> Device No. 16:

```
N=-6
READ (16,100)A
    :
READ (10,200)B
    :
READ (N,110)A
```

APPENDIX 5

BASIC DIFFERENCES BETWEEN FORTRAN II AND PDP-10 FORTRAN IV

1. Variable Type

   Variables may be declared by type through the use of the DOUBLE PRECISION, COMPLEX, INTEGER, LOGICAL, and REAL type specifications. Implicit typing may be accomplished through the use of the IMPLICIT specification statement.

2. Mixed Mode

   Mixed mode expressions are permitted except for the combination of the double precision and complex quantities.

3. Function Naming

   The initial letter of functions is used to type the values of functions. Thus, the LOG, LOG10, and FIX functions have been changed to ALOG, ALOG10, and IFIX, etc. The terminal F in function names is no longer meaningful, and function names may have from one to six characters.

4. Arithmetic Function Statement Dummy Arguments

   In FORTRAN IV if a variable appears both as a dummy argument in an arithmetic statement function and as an ordinary variable in the same program, its type is the same in both contexts.

5. Hardware Tests

   All hardware tests and settings such as IF ACCUMULATOR OVERFLOW and SENSE LIGHT i have been changed to subroutine calls such as CALL OVERFL(i) and CALL SLITE(i).

6. Input/Output

   The following input/output statements have been changed:

FORTRAN II	FORTRAN IV
READ TAPE u,list	READ (u)list
READ INPUT TAPE u,f,list	READ (u,f)list
WRITE TAPE u,list	WRITE (u,)list
WRITE OUTPUT TAPE u,f,list	WRITE (u,f)list

7. COMMON and EQUIVALENCE

In FORTRAN IV, EQUIVALENCE does not affect the ordering within common blocks. EQUIVALENCE may only have the effect of lengthening a common block. COMMON statements may contain dimension information.

8. EXTERNAL

Arguments of subprograms which are external subprograms are declared as such through the use of the EXTERNAL statement.

APPENDIX 6

FORTRAN IV COMPILER DIAGNOSTICS

	Message and Meaning	Page References
1-1	DUPLICATED DUMMY VARIABLE IN ARGUMENT STRING	
	A dummy variable (identifier) may appear only once in any one argument set representing the arguments of a subprogram.	51
1-2	ARRAY NAME ALREADY IN USE	
	Any attempt to re-dimension a variable or redefine a scalar as an array is illegal.	41-42
1-3	ATTEMPT TO REDEFINE VARIABLE TYPE	
	Once a variable has been defined as either complex, double precision, integer, logical or real it may not be defined again.	4,5,48
1-4	NOT A VARIABLE FORMAT ARRAY	
	The variable which contains the FORMAT specification read-in at object time must be a dimensioned variable, i.e., an array.	30-31
1-5	NAME ALREADY USED AS NAMELIST NAME	
	After a NAMELIST name has been defined, it may appear only in READ or WRITE statements and may not be defined again.	32-33
1-6	DUPLICATED NAMELIST NAME	32-33
1-7	A NAME APPEARS TWICE IN AN EXTERNAL STATEMENT	
	A subprogram name has been declared EXTERNAL more than once.	56
1-10	SUBROUTINE NAME ALREADY IN USE	
	A subroutine name has appeared in another statement which is not a subroutine call. A subroutine name may be referenced only by a CALL statement.	54-56
1-11	DUMMY ARGUMENT IN DATA STATEMENT	
	Dummy arguments may not appear in DATA statements.	46-47
1-12	NOT A SCALAR OR ARRAY	
	An attempt to assign a label number to an arithmetic FUNCTION name is illegal.	51-52
1-13	ILLEGAL USE OF DUMMY ARGUMENT	
	Dummy arguments may be used with functions or subprograms only.	51

	Message and Meaning (Cont)	Page References
1-14	ILLEGAL DO LOOP PARAMETER	
	The DO index must be a non-subscripted integer variable while the initial, limit and increment values of the index must be an integer expression - the index may not be zero.	19-20
1-15	I/O VARIABLES MUST BE SCALARS OR ARRAYS	
	Referencing data in an I/O statement other than scalars or arrays is illegal.	35
S-1	SYNTAX	
	Indicates an error in the format of the statement referenced.	--
S-2	ILLEGAL USE OF DO-LOOP	
	Control may not transfer into the range of a DO from any statement outside its range.	19-20
S-3	ILLEGAL FIELD SPECIFICATION	
	The field width or decimal specification in a FORMAT statement must be integer. The number of Hollerith characters in an H specification must be equal to the number specified.	24,25, 26,27
S-5	ILLEGAL TYPE SPECIFICATION	
	The type of constant specified is illegal.	5,6,7
S-6	ARGUMENT IS NOT SINGLE LETTER	
	Arguments in parentheses must be single letters.	--
S-7	NAMELIST' NOT FOLLOWED BY "/"	
	The first character following NAMELIST must be /.	32
S-10	ILLEGAL CHARACTER-LINE DELIMITER EXPECTED	
	The requirements for a complete FORTRAN statement have been satisfied; any additional characters other than a line delimiter are illegal.	1-2
S-11	A NUMBER WAS EXPECTED	
	Only arrays which are subprogram arguments can have adjustable dimensions.	42
S-12	ILLEGAL USE OF IMPLIED DO LOOP	
	Implied DO loops in I/O statements must be nested properly.	35

	Message and Meaning (Cont)	Page References
S-13	ATTEMPT TO USE AN ARRAY AS A SCALAR	
	Variables may be either scalar or array but not both. Variables appearing in a DIMENSION statement must be subscripted when used.	7-8
S-14	ARRAY NOT SUBSCRIPTED	
	See S-13	
S-15	ILLEGAL USE OF AN ARITHMETIC FUNCTION NAME	51,52, 53
	A function may not be redefined.	
S-16	ILLEGAL CHARACTER DETECTED - DELIMITER EXPECTED	32,34 43-49
	A / , or other delimiter is missing	
O-1	BLOCK DATA NOT A SEPARATE PROGRAM	
	Block Data must exist as a separate program	47-48
O-2	SUBROUTINE IS NOT A SEPARATE PROGRAM	
	A subroutine following a main program or another subroutine subprogram may have no statement between it and the preceding programs END statement and must begin with a SUBROUTINE statement. The previous program must have been terminated properly.	54
O-3	STATEMENT OUT OF PLACE	41-49, 52
	Specification statements should precede executable statements.	
A-1	MINIMUM VALUE EXCEEDS MAXIMUM VALUE	
	Minimum value of an array exceeds the maximum value specified.	42
A-2	ATTEMPT TO ENTER A VARIABLE INTO COMMON TWICE	
	A variable may appear in COMMON statement only once.	43
A-3	ATTEMPT TO EQUIVALENCE A DUMMY ARGUMENT	
	Dummy identifiers of subprograms may not appear in EQUIVALENCE statements in that subprogram.	45,51
A-4	NOT A CONSTANT OR DUMMY ARGUMENT	
	Only constant and dummy arguments may be used as arguments in dimension statements.	53

	Message and Meaning (Cont)	Page References
M-1	TOO MANY SUBSCRIPTS	
	An array variable appears with more subscripts than specified.	8, 42
M-2	NOT ENOUGH SUBSCRIPTS	
	An array variable appears with too few subscripts.	8, 42
M-3	CONSTANT OVERFLOW	
	Too many significant digits in the formation of a constant.	5,6
M-4	ILLEGAL 'IF' ARGUMENT	
	Logical IF or DO statement adjacent to a logical IF statement, or illegal expression within a logical IF statement.	18, 19
M-5	ILLEGAL CONVERSION IMPLIED	
	Attempt to mix double precision and complex data in the same expression.	22
M-6	NUMBER TOO LARGE	
	Illegal statement label.	1
M-7	UNTERMINATED HOLLERITH STRING	
	A missing single quote or fewer than n characters following an "nH" specification.	28
M-10	ILLEGAL DO LOOP CLOSE	
	Illegal statement terminating a DO loop.	21
M-11	VARIABLES AND DATA DO NOT MATCH	
	Incorrect number of constants supplied for a DATA statement.	46
M-12	NON-INTEGER PARAMETER IN 'DO' STATEMENT	
	DO statement parameters must be integers.	19, 20
M-13	NON-INTEGER SUBSCRIPT	
M-14	ILLEGAL COMPARISON OF COMPLEX VARIABLES	
	The only comparison allowed of complex variables is .NE. or .EQ.	13
M-15	TOO MANY CONTINUATION CARDS	
	More than 19 continuation cards	2

Digital Equipment Corporation
Maynard, Massachusetts

printed in U.S.A.