EY-1233E-ID-0001

ANSI-77 FORTRAN Information Document

Prepared by Educational Services
Of
Digital Equipment Corporation

1st Edition, April 1983

The information in this document is subject to change without notice. Digital Equipment Corporation assumes no responsibiliity for any errors that may appear in this manual.

The software described in this document is furnished under a license and may not be used or copied except in accordance with the terms of such license.

Digital Equipment Corporation assumes no responsibility for the use or reliability of its software on equipment that is not supplied by Digital.

The following are trademarks of Digital Equipment Corporation:

digital	DIBOL	TOPS-10
DEC	MASSBUS	TOPS-20
DECmate	PDP	UNIBUS
DECnet	P/OS	VAX
DECsystem-10	Professional	VMS
DECSYSTEM-20	Rainbow	VT
DECUS	RSTS	Work Processor
DECwriter	RSX	

5/83-15

CONTENTS

CONTENTS (Cont)

TABLES

TABLES (Cont)

PREFACE

American National Standard Programming Language FORTRAN, ANSI X3.9-1978, specifies the form and establishes the interpretation of programs expressed in the FORTRAN language. Its purpose is to promote portability of FORTRAN programs for use on a variety of data processing systems.

FORTRAN 77 is a revision of American National Standard FORTRAN, ANSI X3.9-1966. It describes two levels of the FORTRAN language, referred to as FORTRAN and Subset FORTRAN. FORTRAN 77 includes the subset, American National Standard Basic FORTRAN, ANSI X3.10-1966.

This information document provides information about ANSI-77 standard FORTRAN and Digital's FORTRAN features under TOPS-10/20, VAX, and PDP FORTRAN. The primary functions of this document are to serve as a reference point to and summary of more detailed information, and to serve as a quick review of new features and changes, such as the comparison and contrast of different versions or of different operating systems.

The target audience for this document is software specialists whose responsibilities include presale, installation, and warranty support of FORTRAN. These specialists should have the following prerequisite skills and training: programming experience; a knowledge of FORTRAN; and user-level knowledge of the TOPS-10/20, VMS, and the several operating systems that run on the PDP-11 family of computers. This document is not a tutorial in the application of FORTRAN.

CHAPTER 1
INTRODUCTION

1.1 INTRODUCTION

The FORTRAN language was originally developed for solving mathematically oriented problems. The engineering and scientific communities, as well as the business community use it extensively for analytical work. FORTRAN applications have evolved to include more general applications involving character and file manipulation. In March 1976 a new standard was drafted to ensure portability of FORTRAN programs that include the advanced file and character handling features. The final version of this standard was approved on 3 April 1978 and is referred to in this document as the FORTRAN-77 or ANSI-77 standard.

The following paragraphs summarize the appropriate sections of the FORTRAN-77 full language standard (ANSI X3.9-1978). (These sections reflect the major areas of change involved in the new versions of TOPS-10/20, VMS, and the PDP family of operating systems.) This is not meant to be a complete delineation of all the changes that were added to the previous standard (ANSI X3.9-1966). Descriptions of these changes are found in the American National Standard Programming Language FORTRAN, ANSI X3.9-1978 published by the American National Standards Institute. The following paragraphs refer to specific sections of this book.

1.2 CHARACTER CONVENTIONS

1.2.1 Character Constants and Symbolic Constants

A character datum has one character storage unit in a storage sequence for each character in the datum. A storage sequence is a sequence of storage units (either numeric or character). If a datum requires more than one storage unit in a storage sequence, these storage units are consecutive. (Section 2.13)

A character datum is a string of characters consisting of any characters capable of being represented in the processor. (Blank characters are valid and significant.) The character position of each character in the string is numbered consecutively from left to right. (Section 4.8)

A character constant is a string of printable ASCII characters enclosed by apostrophes. Blanks between delimiting apostrophes are significant, but the delimiting apostrophes are not counted

as part of the datum. Within a character constant, the apostrophe character is represented by two consecutive apostrophes with no space or other character between them.

A character constant has the form:

```
'c1c2c3.....cn'
```

where c1,c2,c3,...,cn are printable characters.

A character constant expression is a character expression in which each operand is a character constant, the symbolic name of a character constant, or a character constant expression enclosed in parentheses. (Variables, array elements, substrings, and function references are not allowed.) (Section 6.2.3)

A parameter statement allows constants to be defined symbolically during compilation. Its form is:

```
PARAMETER (p=e [,p=e] . . .)
```

where p is a symbolic name and e is a constant expression.

The constant acquires the same data type as the symbolic name. In particular, if p is a character constant expression, e must correspondingly be a character constant expression. The scope of a parameter is the program unit in which it is declared. The data type of a symbolic name is specified by a type-statement or IMPLICIT statement preceding the defining PARAMETER statement. The default length for the symbolic name is one; other lengths can be specified in type-statements or IMPLICIT statements preceding the symbolic name. (Refer to section 8.6.)

The following is an example of the parameter statement.

```
CHARACTER DELTA,EPSILON*3
PARAMETER (DELTA='DIFF',EPSILON='SUM',E=2.7)
```

1.2.2 Character Variables and Arrays

The form of a character type-statement is:

```
CHARACTER [*len [,]] nam [,nam] . . .
```

where nam is of the form:

```
v [* len]
a [(d)] [*len]
```

[v is a variable name, symbolic name of a constant, function name, or dummy procedure name; a is an array name; a(d) is an array declarator; len is the number of characters of a character

variable, character array element, or character function.] (Refer to section 8.4.2.)

The length specification can be an unsigned integer constant, a positive-valued integer constant expression in parentheses, or an asterisk in parentheses.

In the statement:

```
CHARACTER*4 ALPHA,BETA*1,GAMMA(10)*8
```

the following character declarations are made.

```
ALPHA has a length specification of 4
BETA has a length specification of 1
GAMMA has a length specification of 8
```

1.2.3 Character Relational Expressions

A character relational expression has the form a rel b where a and b are character expressions and rel is a relational operator. (Refer to section 6.3.4.) This expression is interpreted as logically true if the values of the operands satisfy the relation specified by the operator; it is interpreted as false if the relation is not satisfied. (Section 6.3.5)

The six relational operators are:

.LT.	Less than
.LE.	Less than or equal to
.EQ.	Equal to
.NE.	Not equal to
.GE.	Greater than or equal to
.GT.	Greater than

In character relational expressions, less than means "precedes in the ASCII collating sequence." For example, the expression:

```
'ABCD' .LT. 'CDEF'
```

states that 'ABCD' is less than 'CDEF'. Because this relationship does exist, the value of the expression is true. If the relationship did not exist, the value of the relationship would be false.

If the two character expressions in a relational expression are not the same length, the shorter of the two is padded on the

right with spaces until the lengths are equal. For example, in the relational expressions:

```
'ABC' .EQ.  'ABC '

'AB' .LT.  'C'
```

the first has a value of true, even though the lengths of the expressions are not equal; and the second has a value of true even though 'AB' is longer than 'C'.

1.2.4 Character Assignment Statements

A character assignment statement has the form:

```
v = e
```

where v is the name of a character variable, character array element, or character substring; and e is a character expression.

Execution of this statement causes e to be evaluated and v to be assigned the value of e. V and e may be of different lengths. (Section 10.4) In the full language, only as much of the value of e must be defined as is needed to define v. For example:

```
CHARACTER X*4,Y*5
X=Y
```

does not require that the substring Y(4:5) be defined.

1.2.5 Substring Reference and Definition

A character substring is a contiguous segment of a character variable or character array element. A character substring is identified by a substring name and can be assigned values and referenced. (Section 5.7)

A character substring reference has one of the following forms:

```
v([e1]:[e2])

a(s[,s]...) ([e1]:[e2])
```

where:

- v is a character variable name
- a is a character array name

- s is a subscript expression
- e1 is an optional numeric expression that specifies the leftmost character position of the substring
- e2 is an optional numeric expression that specifies the rightmost character position of the substring.

Character positions within a character variable or array element are numbered from left to right, starting with 1.

The values of e1 and e2 must satisfy the inequalities:

```
1 .LE. e1 .LE. e2 .LE. len
```

If e1 is omitted, its value is implied to be 1; if e2 is omitted, its value is implied to be len. The length of a character substring is e2 - e1 + 1.

If e1 or e2 is not an integer, it is converted to an integer value by truncating its fractional part. (Sections 5.7.1, 5.7.2) For example, in the statement:

```
CHARACTER TLC*8,XIN(3,4)*6
TLC='START'
XIN(2,3)='OUTPUT'
```

the substring name TLC(2:5) has a character value of "TART" and the substring name XIN(2,3)(4:6) has a character value of "PUT."

In the CHARACTER type statement, character variable TLC is declared as having a string length of 8 and each element of character array XIN has a string length of 6.

1.2.6 Character Expressions

Character expressions (including the concatenation operator) consist of character operands and character operators. A character operand can be:

1. A character constant
2. A symbolic name of a character constant
3. A character variable
4. A character array element
5. A character substring

6. A character expression, optionally enclosed in parentheses

7. A character function reference.

The character operator is the concatenation operator (//). A character expression of the form:

character operand [//character operand]...

for example x2//x1, is interpreted as x1 concatenated with x2. The resulting character string has the value of the value of x1 on the right concatenated with the value of x2 on the left. The length of this resulting character string is the sum of the lengths of x1 and x2.

Parentheses do not affect the value of a character expression.

Variables, array elements, substrings, and function references are not allowed. (Sections 6.2, 6.2.1, 6.2.2.1, 6.2.2.2, 6.2.3, 6.6.5)

For example, the value of the character expression 'XYZ'//'ABCD' is the character string 'XYZABCD'.

1.2.7 Character Variables and Constants in DATA Statements

The form of the DATA statement is:

```
DATA nlist/clist/[[,] nlist/clist/]...
```

where nlist is a list of variable names, array names, array element names, substring names, and implied-DO lists; and clist is a constant list of the form a[,a]... where each a is either a constant, a parameter, or a repetition factor of the form n* followed by a constant or parameter (denoting n successive appearances of the constant or parameter separated by commas). For example:

```
CHARACTER*4 TEXT
DATA TEXT/'YYXX'/
```

illustrates the use of a character constant in a DATA statement. (Refer to section 9.4.)

1.2.8 Character Variables, Array Elements, Arrays, and Substrings in Input Lists

These are discussed in Section 12.8.2.1.

1.2.9 Character Constants, Variables, Array Elements, Arrays, Substrings, and Expressions in Output Lists

Input/output lists are used in READ, WRITE, or PRINT statements to specify data to be transferred. (Section 12.8.2.2)

1.2.10 Character Functions

These include intrinsic functions, external functions, and statement functions that return values of type character. (Sections 15.2, 15.2.2, 15.4)

1.2.11 Dummy and Actual Arguments of Type Character

These are discussed in section 15.9.3.1.

1.2.12 Intrinsic Functions that Operate on Character Data

These functions are LEN, INDEX, CHAR, ICHAR, LGE, LGT, LLE, and LLT. (Refer to section 15.10.) LEN gives the length of the character entity. INDEX(a1,a2) gives the location of substring a2 in string a1. CHAR(i) returns the character in the i-th position of the processer collating sequence. The value is of type character of length one. ICHAR gives the position of a character in the processor collating sequence.

LGE(a1,a2) returns the value true if a1 = a2 or if a1 follows a2 in the ASCII standard collating sequence and returns the value false otherwise. LGT(a1,a2) returns the value true if a1 follows a2 in the standard collating sequence and otherwise returns the value false. LLE(a1,a2) returns the value true if a1 = a2 or if a1 precedes a2 in the standard collating sequence, and otherwise returns the value false. LLT(a1,a2) returns the value true if a1 precedes a2 in the standard collating sequence and otherwise returns the value false.

1.2.13 Character Variable Names, Array Element Names, Array Names, and Substring Names in EQUIVALENCE Statements

An EQUIVALENCE statement is used to specify the sharing of storage units by two or more entities in a program unit. (Sections 8.2, 8.2.1, 8.2.3) The form of the EQUIVALENCE statement is:

```
EQUIVALENCE (nlist)[,(nlist)]...
```

where nlist is a list of variable names, array element names, array names, and other character substring names. Character strings may be equivalenced only with other character strings. The association is made between the first storage units occupied

by entities appearing in the equivalence list. Any adjacent characters may also have the same character storage unit and thus may be associated. For example:

```
CHARACTER A*6, C(4)*4
EQUIVALENCE (A,C(2))
```

causes the following associations:

```
Storage unit: 1 2 3 4 5 6 7 8
Variable A:     - - - - - - A - - - -
Array C:        - - C(2) -/- C(3) - -
```

1.2.14 Character Variable and Arrays in COMMON Blocks

The form of a COMMON statement is:

```
COMMON [/[cb]/] nlist [,]/[cb]/ nlist]...
```

where cb is a common block name and nlist is a list of variable names, array names, and array declarators. If a character variable or array is placed in a common block, that block must contain only character data. (Section 8.3.1)

1.2.15 Character Variables and Expressions as the Values of the Specifiers in OPEN Statements

These are discussed in section 12.10.1.

1.2.16 Character Constants, Character Arrays, and Character Variables as Format Specifiers

The form of a FORMAT statement is:

```
FORMAT fs
```

where fs is a format specification. One of the options for the format identifier is a character array name, character variable, character array element, or character expression. (Section 13.1.2)

1.3 CONTROL STATEMENTS

1.3.1 BLOCK IF, ELSE IF, and END IF Statements

FORTRAN-77 has added BLOCK IF, ELSE IF, ELSE, and END IF statements. (Refer to sections 11.6, 11.7, 11.8, and 11.9.)

The IF-level of a statement s is n1 - n2 where n1 is the number of block IF statements from the beginning of the program unit up

to and including s, and n2 is the number of END IF statements in the program unit up to but not including s. This facility enables the processor to determine which IF-THEN-ELSE statements correspond to each other.

The form of a BLOCK IF statement is:

```
IF (e) THEN
```

where e is a logical expression. The processor executes a BLOCK IF statement by first evaluating e. If the value of e is true, and the IF block is empty, the program control is passed directly to the next END IF statement. If the IF block is not empty, the first statement of the IF block is executed. If the value of e is false, control is transferred to the next ELSE IF, ELSE, or END IF statement having the same IF level as the BLOCK IF statement. (An IF BLOCK consists of all the executable statements appearing between the BLOCK IF statement up to, but not including, the next ELSE IF, ELSE, or END IF statement having the same IF level as the BLOCK IF statement).

The form of an ELSE IF statement is:

```
ELSE IF (e) THEN
```

where e is a logical expression. An ELSE IF block consists of all the executable statements between the ELSE IF statement up to, but not including, the next ELSE IF, ELSE, or END IF statement having the same IF level as the ELSE IF statement.

The processor executes the ELSE IF statement by first evaluating the expression e. If the value of e is true, and the ELSE IF block is empty, control is transferred directly to the next END IF statement with the same IF level. If the value of e is true, and the ELSE IF block is not empty, the first statement of the ELSE IF block is executed. If the value of e is false, control is transferred to the next ELSE IF, ELSE, or END IF statement having the same IF level as the ELSE IF statement.

The form of the ELSE statement is:

```
ELSE
```

The form of the END IF statement is:

```
END IF
```

The END IF statement performs no executable function, but serves as a point of reference.

1.3.2 DO Loop Semantics

The form of a DO statement is:

```
DO s [,] i = e1, e2 [,e3]
```

where s is the statement label of an executable statement; i is the name of an integer, real, or double-precision variable (called the DO variable in FORTRAN-77); e1, e2, e3 are each an integer, real, or double-precision expression. E1 is the initial value of the DO variable, e2 is the limiting value of the DO variable, and e3 is the increment value for the DO variable.

Two new features of FORTRAN-77 DO loops are:

1. The possibility of zero trip loops (The FORTRAN-66 standard did not specify how many iterations a DO loop had. Most processors, including Digital and IBM, executed loops at least once.)

2. The availability of the loop index after loop exit (previously this was not available). (Sections 11.10 through 11.10.7)

1.4 INPUT/OUTPUT

1.4.1 Format Edit Descriptors

A field descriptor describes the size and format of a data item or items. An edit descriptor specifies an editing function to be performed on a data item or items. (Refer to sections 13.2, 13.5, 13.5.3, 13.5.3.1, 13.5.6, 13.5.9.2.3)

The BN edit descriptor causes the processor to ignore all the embedded and trailing blanks it encounters within a numeric input field. It has the form:

```
BN
```

The effect is that of actually removing the blanks and right-justifying the remainder of the field. A field of all blanks is treated as zero.

The BZ edit descriptor causes the processor to treat all the embedded and trailing blanks it encounters within a numeric input field as zeros. It has the form:

```
BZ
```

The SP edit descriptor causes the processor to produce a plus character in any position where this character would otherwise be optional. It has the form:

SP

The SS edit descriptor causes the processor to suppress a leading plus character from any position where this character would normally be produced as an optional character; it has the opposite effect of the SP field descriptor described below. The SS descriptor has the form:

SS

The S edit descriptor reinvokes optional plus characters (+) in numeric output fields. It has the form:

S

The S descriptor counters the action of either the SP or SS descriptor by restoring to the processor the decision-making ability to produce plus characters on an optional basis.

The T edit descriptor specifies the position, relative to the start of an external record, of the next character to be processed. It has the form:

Tn

where the term n indicates the position in the external record of the next character to be processed. The value of n must be greater than or equal to 1, but not greater than the number of characters allowed in the record.

The TL edit descriptor is a relative tabulation specifier for tabbing to the left. It has the form:

TLn

The term n specifies that the next character to be transferred from or to a record is the nth character to the left of the current character. The value of n must be greater than or equal to 1. If the value of n is greater than or equal to the current character position, the first character in the record is specified.

The TR edit descriptor is a relative tabulation specifier for tabbing to the right. It has the form:

TRn

The term n indicates that the next character to be transferred from or to a record is the nth character to the right of the current character. The value of n must be greater than or equal to 1.

The I field descriptor specifies decimal integer values. It has the form:

```
Iw[.m]
```

The corresponding I/O list element must be of integer data type.

On input, the I field descriptor specifies that w characters are to be read from an external file, interpreted as a decimal integer value, and assigned to the corresponding I/O list element.

On output, the I field descriptor specifies that the value of the corresponding I/O list element is to be transferred as a decimal value, right-justified, to an external field w characters long. If m is present, the external field consists of at least m digits; if necessary, zeros are added on the left to bring the total digits to m. If the value exceeds the field width, the entire field is filled with asterisks. If the value of the list element is negative, the field will have a minus sign as its leftmost, nonblank character, provided the term w is large enough. Plus signs are optionally suppressed (at the discretion of the processor) unless SP is specified.

The G field descriptor specifies real or double precision values, combining E- or F-type formats according to the size of the number being output. It has the form:

```
Gw.d[Ee]
```

The corresponding I/O list element must be of real or double-precision data type, or it must be either the real or the imaginary part of a complex data type.

On input, the G field descriptor does not differ from the F, E, or D descriptors.

On output, the G field descriptor specifies that the value of the corresponding I/O list element is to be transferred as a real or double-precision value in either exponential or fixed-point form depending on its magnitude, rounded to d decimal positions, and right-justified, to an external field w characters long.

1.4.2 Expressions on Output Lists

An input/output list specifies the entities whose values are transferred by a data transfer input/output statement. An

input/output list is a simple list, an implied-DO list, or two input/output lists separated by a comma. A simple input/output list item is one of the following.

1. A variable
2. An array
3. A character substring
4. An element of an array

In addition, in the full language, a simple output list may be an expression, with the exception of a character expression containing a dummy argument with a length attribute specified with an asterisk. (Sections 12.8.2, 12.8.2.1, 12.8.2.2, 12.8.2.3)

Note that a constant, an expression involving operators or function references, or an expression enclosed in parentheses may appear as an output list item but not as an input list item.

An implied-DO list is of the form:

```
(dlist, i = e1, e2 [,e3])
```

where i, e1, e2, and e3 are as specified for the DO statement and dlist is an input/output list.

1.4.3 Internal Files

Internal files provide a means of transferring and converting data from internal storage to internal storage. An internal file is a character variable, character array element, character array, or character substring. Reading and writing records is accomplished by sequential access formatted input/output statements.

1.4.4 Unit Specifier and Identifier

The form of the unit specifier is:

```
[UNIT=]u
```

where u is an external unit identifier or an internal unit identifier. An external unit identifier is an integer expression with a zero or positive value or is an asterisk, identifying a particular processor-determined external unit that is preconnected for formatted sequential access. (Section 12.3.3)

1.4.5 Format Specifier and Identifier

The form of a format specifier is:

```
[FMT=]f
```

where f is a statement label of a FORMAT statement, an integer variable that has been ASSIGNED the statement label of a FORMAT statement, a character array name, any character expression other than one involving concatenation of an operand whose length was specified by an asterisk and that is not the symbolic name of a constant, or an asterisk specifying list-directed formatting. If FMT= is omitted, the format specifier must be the second item in the control information list and the first item shall be the unit specifier without UNIT=. (Section 12.4)

1.5 ASSUMED SIZE ARRAY DECLARATORS

In an array declarator for a dummy array, the upper bound of the last dimension may be specified by an *. (Sections 5.1.2, 5.1.2.2, 5.5)

The form of an array declarator is:

```
a (d [,d]...)
```

where a is the symbolic name of the array and d is a dimension declarator.

A dummy array declarator is an array declarator in which the array name is a dummy argument.

1.6 USE OF A FORMAT STATEMENT LABEL IN AN ASSIGN STATEMENT

The form of a statement label assignment statement is:

```
ASSIGN s TO i
```

where s is a statement label of an executable statement or FORMAT statement and i is an integer variable name. (Sections 10.3, 12.4)

An integer variable assigned a statement number may be used only as a statement identifier in an assigned GOTO statement or as a format identifier in an input/output statement.

1.7 INTRINSIC AND EXTERNAL STATEMENTS

The INTRINSIC statement is used to identify a symbolic name as representing an intrinsic function. (Refer to sections 8.7 and

8.8.) It permits the name to be used as an actual argument. The form of the INTRINSIC statement is:

```
INTRINSIC fun[,fun]...
```

where fun is the symbolic name of an intrinsic function. For example:

```
INTRINSIC SIN
CALL AIDS(A,B,C,SIN)
```

The form of the EXTERNAL statement is:

```
EXTERNAL proc[,proc]...
```

where proc is the symbolic name of an external procedure, a dummy procedure name, or a block data program unit.

The EXTERNAL statement is used to identify a symbolic name as representing an external procedure or dummy procedure, and to permit such a name to be used as an actual argument.

1.8 SAVE STATEMENT

A SAVE statement is used to retain the definition of a variable, an array, or a named common block after the execution of a RETURN or END statement. (Refer to Section 8.9.) The form of a SAVE statement is:

```
SAVE [a,[,a]...]
```

where each a is a named common block name preceded and followed by a slash, a variable name, or an array name.

1.9 NULL ARGUMENT LISTS FOR FUNCTIONS

The form:

```
fun()
```

may be used to reference a function with a null argument list. (Section 15.2.1)

Statement functions with null argument lists may be declared. A null, parenthesized argument list appears in the declaration. (Section 15.4.1)

The form of a statement function statement is:

```
fun ([d,[,d...]) = e
```

where fun is the symbolic name of the statement function, d is a statement function dummy argument, and e is an expression. Parentheses may optionally be used for null argument lists in FUNCTION statements.

1.10 CONSTANT EXPRESSIONS

Constant expressions (not just constants) may be used in array declarators, PARAMETER statements, and implied DO-lists in DATA statements. (Sections 6.1.3, 6.1.3.1, 6.2.3, 6.4.4, 6.7, 5.1.1.1, 8.6, 9.3)

CHAPTER 2
TOPS-10/20 FORTRAN

2.1 INTRODUCTION

The FORTRAN language as implemented on the TOPS-10 and TOPS-20 operating systems is compatible with and encompasses the standard described in American National Standard FORTRAN, X3.9-1978 at the subset level. TOPS-10/20 FORTRAN also supports most features from the FORTRAN-77 full language standard.

FORTRAN provides many extensions and additions to the FORTRAN-77 standard that greatly enhance the usefulness of FORTRAN and increase its compatibility with FORTRAN languages implemented by other computer manufacturers.

The TOPS-10/20 FORTRAN Language Manual (AA-N383A-TK) reflects the software as of Version 7 of the FORTRAN-10/20 object time system (FOROTS), and Version 7 of the FORTRAN-10/20 debugging program (FORDDT). This manual describes the FORTRAN language as implemented for the TOPS-10 operating system (FORTRAN-10) and the TOPS-20 operating system (FORTRAN-20) and notes any differences. It supersedes AA-J127A-TK.

2.2 MAJOR FEATURES OF VERSION 7 THAT WERE NOT IN VERSION 6

2.2.1 Features Supported for Character Data

Version 7 supports character data as specified by the full language FORTRAN-77 standard. Supported features include character assignments, character relationals, substrings, concatenation, and character functions and arguments, including functions and dummy arguments of length *. Character data is supported in DATA, COMMON, and EQUIVALENCE statements; and in formatted, binary, and image mode I/O.

2.2.1.1 Character Assignment Statements - Character constants may be assigned to noncharacter variables. The standard restricts the assignment of character constants to character variables.

Version 7 extends the standard to support assignment statements in which there is overlap between the left- and right-hand sides. The results of such an assignment will be as if the expression on the right-hand side were assigned to a temporary and then the value of the temporary were assigned to the left-hand side.

2.2.1.2 Character Expressions - Version 7 extends the standard to allow concatenation of formal parameters that are length *.

2.2.1.3 Character Variables and Constants in DATA Statements - For compatibility with previous versions, Version 7 supports the use of character constants to initialize noncharacter variables.

2.2.1.4 Character Variables, Array Elements, Arrays, and Substrings in Input Lists - In addition to the A edit descriptor for input/output list items of type character, Version 7 supports the G edit descriptor. The G edit descriptor functions as the A edit descriptor for list items of type character. R edit descriptors are not supported for character data.

2.2.1.5 Dummy and Actual Arguments of Type Character - Version 7 extends the standard to provide support of character constants as actual arguments corresponding to dummy arguments that are integer, real, double-precision, complex, or logical, as well as character. This feature does not work when the name of the function called is itself a dummy argument.

If an actual argument is of type character and is not a constant, the corresponding dummy must be of type character. If a dummy argument is of type character, the corresponding actual must be of type character.

Actual arguments may be longer than corresponding dummy arguments. Length * may be used for character dummy arguments.

2.2.1.6 EQUIVALENCE Statements - It is illegal to equivalence a numeric variable to a character variable. Equivalencing a numeric variable to an unaligned character variable is fatal; equivalencing a numeric variable to a word-aligned character variable is nonfatal.

2.2.1.7 Character Variables and Arrays in COMMON Blocks - When a character variable or character array is in a COMMON block, all the entities in that COMMON block must be of type character. If both character and numeric data are specified in the same COMMON block, a nonfatal warning message is issued. Variables other than character variables begin on word boundaries; thus a COMMON block containing both character and numeric data would contain unused character positions.

2.2.1.8 Namelists - Version 7 supports substrings in namelist input, but not in namelist output.

Version 7 does not support global optimization of programs that contain character data. If the /OPTIMIZE switch is specified for such a program, the warning diagnostic:

```
Global optimization not yet supported with
character data - /OPT ignored
```

is issued.

2.2.2 IF THEN ELSE Statements

Version 7 supports the block IF, ELSE IF, ELSE, and END IF statements.

2.2.3 Expressions on Output Lists

An output list item can be a variable name, an array element name, a character substring name, an array name, or any other expression. FORTRAN-20 extends the standard to support output list expressions that include concatenation of operands of length asterisk.

2.2.4 Intrinsic and Generic Functions at the FORTRAN-77 Full Language Level

Version 7 supports all intrinsic and generic functions described in section 15.10 of the FORTRAN-77 standard. The following intrinsic functions are new in Version 7.

1. DINT - Truncation for double-precision
2. ANINT, DNINT - Nearest whole number
3. NINT, IDNINT - Nearest integer
4. DDIM - Positive difference for double-precision
5. DPROD - Double-precision product of real arguments
6. ICHAR, CHAR, LEN, INDEX, LGE, LGT, LLE, LLT - Character functions as described in the introduction

The following generic function names have been added: ACOS, AINT, ANINT, ASIN, COSH, CMPLX, DBLE, DIM, LOG, LOG10, MAX, MIN, NINT, REAL, SINH, TAN, and TANH. The second argument to CMPLX is now optional. The generic function name INT has been extended to support arguments that are COMPLEX and INTEGER (as well as REAL and DOUBLE-PRECISION).

2.2.5 Internal Files (Single-Record and Multirecord)

Version 7 conforms to the FORTRAN-77 standard.

2.2.6 FORTRAN-77 DO Loop Semantics

As an extension to the standard, Version 7 supports "extended range DO loops" (transfer into the range of a DO-loop is permitted if a previous transfer out has occurred.)

2.2.7 Assumed-Size Array Declarators

Version 7 conforms to the FORTRAN-77 standard.

2.2.8 Use of FORMAT Statement Numbers in ASSIGN Statements

Version 7 conforms to the FORTRAN-77 standard.

2.2.9 INTRINSIC Statement; EXTERNAL Statement

In Version 7, if the name of an intrinsic function appears in an EXTERNAL statement, that name is subsequently treated as the name of a user-defined function. (This is in accordance with the FORTRAN-77 standard, but incompatible with previous versions of FORTRAN 20. In Version 6, an asterisk appearing in front of an intrinsic name in an EXTERNAL statement is required to force that name to become the name of some external procedure.)

2.2.10 SAVE Statement

In Version 7, if a FORTRAN overlay contains any local variables that are SAVEd, all writable storage in that overlay is preserved. If a named COMMON block is SAVEd, that common block is preserved. Blank COMMON is always preserved.

2.2.11 Null Argument Lists for Functions

Version 7 conforms to the FORTRAN-77 standard.

2.2.12 Minor Syntax Extensions Required by the FORTRAN-77 Standard

The comma is optional in the following: DATA statements, COMMON statements, assigned GOTO, and after the statement number in DO statements.

Parentheses may optionally be used for null argument lists in SUBROUTINE and CALL statements.

Statement numbers are legal on nonexecutable statements.

Exponentiation to an integer power is allowed in the subscript expressions in DATA statements.

2.2.13 Compile Time Constant Expressions in Declarations, as Array Bounds, and String Bounds

Version 7 conforms to the FORTRAN-77 standard.

2.2.14 FORTRAN-77 PARAMETER Statements

Version 7 supports PARAMETER statements in accordance with the FORTRAN-77 standard. Compile time expressions involving multiplication, division, or exponentiation of COMPLEX data are not supported.

In Version 6 the data type of a PARAMETER was determined by the type of the constant; in Version 7 the data type is determined by its symbolic name.

In Version 6 the list of parameters is never enclosed in parentheses; in Version 7 the list of parameters must be enclosed in parentheses. If the list of parameters is not enclosed in parentheses, the compiler assumes that it is not a FORTRAN-77 PARAMETER statement and a warning message is issued. (This warning message can be suppressed by compiling with the /NOF77 (/F66) switch.)

In Version 7 (as in FORTRAN-77), PARAMETER statements may precede type declaration statements except for those statements that specify the type of parameter.

In Version 6 the parameter may only be set to simple constants; in Version 7 the parameter may be set to a constant expression.

2.2.15 DO WHILE and END DO Statements

The DO WHILE/ END DO support involves the following enhancements to the FORTRAN-77 standard.

2.2.15.1 Optional Statement Label in the Indexed (Standard) DO Statement - The syntax of the indexed DO statement is:

```
DO [s[,]] v=e1,e2[,e3]
```

where s is the label of the statement that terminates the loop. If s is omitted, the loop must be terminated by an END DO statement as discussed below.

2.2.15.2 DO WHILE Statement - The DO WHILE statement has the following syntax:

```
DO [s[,]] WHILE (e)
```

where s is the label of the statement that terminates the loop. If s is omitted, the loop must be terminated by an END DO statement.

E is a logical expression that is tested at the beginning of each execution of the loop, including the first. If the value of the expression is true, the statements in the body of the loop are executed; if the value of the expression is false, control transfers to the statement following the loop.

2.2.15.3 END DO Statement - The END DO statement has the syntax:

```
END DO
```

An END DO statement terminates the range of a DO or DO WHILE statement. The END DO statement must be used to terminate a DO block if the DO or DO WHILE statement does not contain a statement label. It may also be used as a labeled terminal statement if the DO or DO WHILE statement does contain a terminal statement label.

2.2.16 LINKtime Type-Checking of Subprogram Arguments

FORTRAN Version 7 and LINK Version 5.1 provide limited type-checking for character constants that are passed as actual arguments that correspond to numeric dummy arguments. Version 7 has modified the argument passing mechanism; the argument passing mechanism for quoted strings now involves passing the address of a descriptor for the string rather than the word address of the string (as is done in Version 6). These two methods of passing arguments may be referred to as "passing by descriptor" and "passing by address." If an actual argument is passed by descriptor and the corresponding formal is passed by address, LINK will transform the actual argument into a passed-by-address argument if the following conditions are satisfied:

1. The argument is a constant.

2. The string is in the same section as the argument block.

3. The byte pointer word in the descriptor in the user's image is word-aligned. (The object code generated by FORTRAN Version 7 now includes descriptors for character variables, primaries, and subprogram arguments).

No type-checking will be performed on calls involving old REL files since either the caller or the callee or both will not have LINK argument descriptor blocks.

Version 7 also supports a new option to the DEBUG switch of the form /DEBUG:PARAMETERS. With this option specified, FORTRAN will generate REL file blocks that specify that illegal argument type mismatches should result in nonfatal error messages at load time.

2.2.17 G-Floating Double-Precision Numbers

FORTRAN-20 Version 7 provides support for the G-floating double-precision number format. The exponent range for this number format is 2.8D-309 to 8.9D+307.

G-floating is an alternative internal format for double-precision, supported only on KL model B processors. The user specifies G-floating format by specifying the /GFLOATING command line switch to the FORTRAN compiler. /NOGFLOATING (the default) specifies the old double-precision format.

REL files that use the two different double-precision formats are not compatible. If a user attempts to LINK together programs compiled with different values of the /GFLOATING switch, a warning will be issued at LINK time.

2.2.18 Native TOPS-20 Command Interface for the Compiler

The FORTRAN-20 Version 7 compiler's command line interface now provides support of long file names, "?", and command recognition. COMPILE now works for any legal TOPS-20 file name. However, a user cannot do an EXECUTE or DEBUG of a long file name. (LINK does not yet support long REL file names). The syntax for the EXEC commands, EXECUTE, DEBUG, and COMPILE, is not affected by the new command scanner.

2.2.19 New Functionality in the ERRSET Subroutine

Version 7 provides ERRSET trapping for additional classes of errors. Also, the user can now write his own fix-up routines for arithmetic exceptions. The calling sequence for ERRSET is:

```
     CALL ERRSET (N)
     CALL ERRSET (N,I)
or
     CALL ERRSET (N,I,SUBR)
```

where N equals the maximum number of error messages to type and I equals the error to which this call applies. If I equals -1 it

will trap to any of the following errors. If I is not specified, -1 is assumed.

0	Integer overflow
1	Integer divide check
4	Floating overflow
5	Floating divide check
6	Floating underflow
8	Library routine error
9	Output field width too small
10	Input floating overflow
11	Input floating underflow
12	Input integer overflow
21	FORLIB warnings
22	Nonstandard usage warnings

SUBR is the subroutine to call on the trap. If SUBR is not specified, no routine is called on the arithmetic exception. If SUBR is specified the effect is as if:

```
CALL SUBR (I, IPC, N2, ITYPE, UNFIXED, FIXED)
```

were placed in the program just after the instruction causing the trap.

I is the error number of the trap. IPC is the program counter of the trap instruction, or if the error number equals 9, IPC equals the program counter of the FOROTS call. N2 equals the second error number (reserved for Digital). ITYPE is the data type of value. UNFIXED is the value returned by the processor, and FIXED is the value after the fix-up (can be changed by SUBR).

2.2.20 Utility Subroutine to Get a Free Unit Number

Version 7 provides an additional FORTRAN-supplied subroutine that can be used to get an unused unit number. The routine FFUNIT (first free unit) is called by:

```
CALL FFUNIT(IUNIT)
```

where IUNIT is an integer variable that is set to the first available unit number by FFUNIT.

2.3 FORTRAN-77 FEATURES THAT ARE NOT SUPPORTED

1. The INQUIRE statement (used to determine the current status of a file attribute).

2. Comment lines and blank lines may not appear between an initial line and its first continuation line, nor may they appear between two continuation lines.

3. The compile time expression in a PARAMETER statement cannot contain multiplication, division, or exponentiation of COMPLEX data.

2.4 SUMMARY OF EXTENSIONS TO THE FORTRAN-77 STANDARD

I. Programmer convenience

A. FORDDT: - Interactive debugger with FORTRAN-like commands

B. Optional array bounds checking and string bounds checking

C. LINKtime checking for subprogram arguments

D. Selective suppression of compile time and run time warnings

E. User selection of default switch values by means of SWITCH.INI

F. Run time traceback on errors; optional traceback on PAUSE

G. INCLUDE statements

H. On TOPS-20: ? and recognition in compiler commands and DIALOG mode

II. Structured programming

A. DO WHILE statement

B. END DO statement

III. Syntax

A. End-of-line comments

B. Multiple statements per line

IV. Very accurate single- and double-precision arithmetic

A. 36-bit integer (10.5 decimal digits)

B. 36-bit floating-point (8 decimal digits)

C. 72-bit floating-point (18 decimal digits)

D. 8-bit floating-point exponent: 10**(-38) to 10**(+38)

E. On 2060 systems, /GFL switch also provides for an 11-bit exponent; 10**(-308) to 10**(+308)

V. I/O

A. NAMELIST I/O

B. Logical device names provide run time device assignments

C. DIALOG mode provides run time selection of OPEN specifiers

D. IMAGE mode files

E. APPEND access to sequential files

F. DELETE, EXPUNGE, SAVE, PRINT, PUNCH dispositions in CLOSE

G. Additional I/O statements

1. REREAD reads previous record

2. TYPE outputs to the user's terminal

3. ACCEPT inputs from the user's terminal

4. ENCODE/DECODE formatted reading/writing to an array rather than a file

H. DBMS-10/20 support

I. SORT-10/20 support

J. DIL-10/20/VAX support

DIL (data interchange library) is a set of callable subroutines that enables a COBOL or FORTRAN programmer to access and use data that resides on another computer system. The DIL allows you to pass data between programs on different systems or directly access records in files on other systems. You can use the DIL to access a single record within a file and avoid having to transfer the entire file to your system. If the accessed data is of the wrong format or data type, DIL provides the necessary data conversion facilities.

To use the DIL in a multiple computer environment, the computers must be interconnected by DECnet to form a network.

Refer to the Data Interchange Library User's Guide, Version 1 (AA-M581A-TK).

CHAPTER 3
VAX-11 FORTRAN

3.1 INTRODUCTION

VAX-11 FORTRAN is an implementation of the full language ANSI FORTRAN-77 standard. It is a compatible superset of PDP-11 FORTRAN-77. (You can compile existing PDP-11 FORTRAN-77 source programs, as well as new programs that incorporate features available in VAX-11 FORTRAN.) VAX-11 FORTRAN Version 3 is also upwardly compatible from the previous versions of VAX-11 FORTRAN and VAX/VMS systems. This means that object files produced by VAX-11 FORTRAN Version 2 can be correctly linked and run on VAX/VMS Version 3. Executable images linked on previous versions of VAX/VMS systems can be run without relinking on VAX/VMS Version 3 systems. It is also possible to freely mix object files from previous versions of VAX-11 FORTRAN with Version 3 object files on Version 3 VAX/VMS systems. However, there is no backward compatibility; that is, it is not possible to link or execute files from VAX-11 FORTRAN Version 3 on previous versions of VAX/VMS systems.

This chapter describes the new features supported in Version 3 and also outlines the major features that are extensions to the standard. This information, as well as more detailed information, is found in the following documents.

1. VAX-11 FORTRAN Language Reference Manual, (April 1982), AA-D034C-TE

2. VAX-11 FORTRAN User's Guide, (April 1982), AA-D035C-TE

3. VAX-11 FORTRAN Installation Guide/Release Notes (April 1982) AA-H953B-TE

The following documents are of interest to VAX-11 FORTRAN programmers.

1. VAX/VMS Primer, AA-D030C-TE

2. VAX/VMS Command Language User's Guide, AA-D023C-TE

3. VAX-11 Symbolic Debugger Reference Manual, AA-D026D-TE

The VAX-11 FORTRAN Language Reference Manual describes the FORTRAN language elements supported by VAX-11 FORTRAN. It is

intended to be used as a reference manual in preparing FORTRAN source programs.

The VAX-11 FORTRAN User's Guide describes how to compile, link, debug, and execute programs written in the VAX-11 FORTRAN language, using the facilities of the VAX/VMS operating system. It contains other information of interest to FORTRAN programmers, such as FORTRAN input/output, error processing, programming efficiency, compatibility between VAX-11 FORTRAN and VAX-11 FORTRAN 66, and compatibility between VAX-11 FORTRAN and PDP-11 FORTRAN.

The VAX-11 FORTRAN Installation Guide/Release Notes contains detailed instructions for installing the VAX-11 FORTRAN compiler on the VAX/VMS operating system. It also contains information about the differences between VAX-11 FORTRAN Version 3 and previous versions of VAX-11 FORTRAN, including a short description of new features and a complete description of the /SHOW qualifier.

3.2 NEW FEATURES SUPPORTED BY VERSION 3

3.2.1 Bit Functions

The following additional bit-manipulation intrinsic functions have been added.

1. IBSET sets a bit.
2. BTEST tests a bit.
3. IBCLR clears a bit.
4. IBITS extracts a bit field.
5. ISHFTC performs a circular shift.
6. MVBITS moves a bit field.

IBSET, BTEST, IBCLR, IBITS, and the subroutine MVBITS operate on bit fields. A bit field is a contiguous group of bits within a binary pattern, specified by a starting bit position and a length. IBSET, IBCLR, and BTEST operate on 1-bit fields and do not require a length argument. IBITS and MVBITS operate on general bit fields. Both the starting position of a bit field and its length are arguments to these intrinsics.

The MVBITS subroutine transfers a bit field from one storage location (source) to a field in a second storage location (destination). The call to MVBITS has the form:

```
CALL MVBITS(m,i,len,n,j)
```

where m is an integer variable or array element that represents the source location (that is, the location from which a bit field is transferred); i is an integer expression that identifies the first bit position in the field transferred from m; len is an

integer expression that identifies the length of the field transferred from m; n is an integer variable or array element that represents the destination location (that is, the location to which a bit field is transferred); and j is an integer expression that identifies the bit in which the transferred bit field begins.

The MVBITS subroutine transfers len bits from positions i through i + len - 1 of the source location (m) to positions j through j + len - 1 of the destination location (n). Other bits of the destination location and all the bits of the source location remain unchanged. The values of i + len and j + len must be less than 32. ISHFTC shifts a binary pattern; a positive shift count indicates a left circular shift, while a negative shift count indicates a right circular shift.

3.2.2 Debugger Commands for Source Code Debugging

A new interface allowing FORTRAN users to access the debugger source code display facility has been provided. The following debugger commands can now be used:

1. SET STEP SOURCE
2. TYPE range
3. EXAMINE/SEARCH
4. SEARCH [range] "string"
5. SET SOURCE

Refer to the VAX-11 Symbolic Debugger Reference Manual.

3.2.3 DEFAULTFILE Keyword in the OPEN Statement

The DEFAULTFILE keyword has been added to the OPEN statement. DEFAULTFILE allows you to specify a default file name that overrides the general FORTRAN defaults.

3.2.4 IMPLICIT NONE Statement

The IMPLICIT statement now has a second format.

```
IMPLICIT NONE
```

The IMPLICIT NONE statement invalidates all implicit defaults and causes error messages to be generated for names that are not explicitly typed in a type declaration statement. You must then explicitly declare the data types of all symbolic names in the program unit. If you specify IMPLICIT NONE, no other IMPLICIT statement can be included in the program unit.

3.2.5 FORTRAN Data Manipulation Language Preprocessor

An interface to an integrated FORTRAN data manipulation language preprocessor has been provided under Version 3. The FORTRAN qualifier /DML controls the use of the interface. The preprocessor produces an intermediate file of FORTRAN source code with FORTRAN DML commands expanded into FORTRAN statements. The compiler then compiles the intermediate files and produces an optional object and listing file. The listing file includes the DML statements and, optionally, their expansions into FORTRAN source. DML error messages appear in the correct place in the listing. The preprocessing and compilation are controlled by a single qualifier on the FORTRAN command so that the user does not need to know that a preprocessor is present.

3.2.6 Faster I/O Interface

A faster I/O interface is now provided by the following changes.

1. Implied DO loops in I/O lists have been optimized.
2. Single-element, unformatted I/O lists now use a special interface.
3. A larger default record size is allowed for unformatted I/O transfers.

3.2.7 Zero-Extending Intrinsic Functions

The two new intrinsic functions IZEXT and JZEXT allow for zero-extending arguments. FORTRAN normally converts a smaller fixed-point data type to a larger fixed-point data type by sign-extending the smaller value. This means that the high-order bits of the larger data type are set to the same value as the sign bit of the smaller data type. The functions IZEXT and JZEXT zero-extend a value to either INTEGER*2 or INTEGER*4 respectively. This means that the high-order bits of the larger data type are set to zero rather than to the sign bit of the smaller data type. The generic function ZEXT selects IZEXT or JZEXT according to the setting of the /I4 command qualifier.

3.2.8 Library-Based INCLUDE Statement

The INCLUDE statement has a second format. The new format allows you to include modules from VAX/VMS text libraries.

```
INCLUDE 'file-specification (module-name) [/[NO]LIST]'
```

The INCLUDE statement specifies that the contents of a designated file are to be incorporated into a compilation directly following the statement.

3.2.9 Improved Math Routines

The accuracy of the following math routines has been improved significantly.

1. SIN
2. COS
3. LOG
4. ATAN
5. ASIN
6. ACOS

3.2.10 Namelist-Directed I/O Statements

The following statements provide namelist-directed I/O.

1. The NAMELIST statement

2. The READ, WRITE, ACCEPT, TYPE, and PRINT statements, with a namelist specifier in place of format and I/O lists

3.2.11 Optimization of Generated Code

The following improvements in compiler optimization have been implemented.

1. In-line expansion of statement functions

2. Elimination of redundant store operations

3. Elimination of redundant argument reduction for calls to SIN and COS with the same arguments

3.2.12 Optimization of I/O Routines

The speed of the following run-time library I/O processing routines has been improved.

1. Floating input conversion
2. Format interpretation

3.2.13 Checking for Extensions to the FORTRAN-77 Standard

The new FORTRAN qualifier /STANDARD provides optional checking for source code that does not conform to the FORTRAN-77 standard.

3.2.14 A Cross-Reference Listing

The new FORTRAN qualifier /CROSS-REFERENCE provides an optional cross-reference as part of the listing file.

3.2.15 Floating-Underflow Checking
The FORTRAN qualifier /CHECK=UNDERFLOW provides optional floating-underflow checking at run time.

3.2.16 Substring-Bounds Checking
The FORTRAN qualifier /CHECK=BOUNDS provides optional substring-bounds checking at run time.

3.2.17 OPTIONS Statement
The OPTIONS statement overrides or confirms the FORTRAN command qualifiers in a program unit.

3.2.18 FORTRAN Definitions for System Symbols
A text library FORSYSDEF.TLB containing the FORTRAN definitions for system symbols is now provided.

3.2.19 Trigonometric Functions in Degrees
The trigonometric functions in Table 3-1 take arguments in degrees or produce results in degrees.

Table 3-1 Trigonometric Functions

Function	Description
SIND, DSIND, QSIND	Sine functions with degree arguments
COSD, DCOSD, QCOSD	Cosine functions with degree arguments
TAND, DTAND, QTAND	Tangent functions with degree arguments
ASIND, DASIND, QASIND	Arc sine with degree result
ACOSD, DACOSD, QACOSD	Arc cosine with degree result
ATAND, DATAND, QATAND	Arc tangent with degree result
ATAN2D, DATAN2D, QATAN2D	Two-argument arc tangent with degree result

3.2.20 Run-Time I/O Error Messages
Additional diagnostic information is now provided with run time I/O error messages.

3.2.21 /SHOW Qualifier

The /SHOW qualifier controls listing options for INCLUDE files, preprocessor-generated output, and the symbol table map.

3.3 EXTENSIONS TO THE ANSI STANDARD

The following are some of the VAX-11 FORTRAN-77 extensions to the ANSI standard.

1. Language elements for keyed and sequential access to VAX-11 RMS multikey ISAM files.

2. A set of data types beyond those specified for full language FORTRAN-77

   LOGICAL*1, BYTE (synonymous)
   LOGICAL*2
   INTEGER*2
   COMPLEX*16, DOUBLE COMPLEX
   REAL*16

3. Explicit specification of storage allocation units for data types (e.g., REAL*8, INTEGER*4).

4. Data initialization in type declaration statements.

5. IMPLICIT NONE statement (described in new features section).

6. DO, DO WHILE, END DO statements - As an extension to the standard, VAX-11 FORTRAN supports "extended range DO-loops (transfer into the range of a DO-loop is permitted if a previous transfer out has occurred).

   The DO WHILE/ END DO support involves the following enhancements to the FORTRAN-77 standard.

   A. Optional statement label in the indexed (standard) DO statement. The syntax of the indexed DO statement is:

      DO [s,[,]] v=e1,e2,[e3]

      where s is the label of the statement that terminates the loop. If s is omitted, the loop must be terminated by an END DO statement.

B. The DO WHILE statement has the syntax:

```
DO [s[,]] WHILE (e)
```

where s is the label of the statement that terminates the loop. If s is omitted, the loop must be terminated by an END DO statement as discussed below. The (e) is a logical expression that is tested at the beginning of each execution of the loop, including the first. If the value of the expression is true, the statements in the body of the loop are executed; if the value of the expression is false, control transfers to the statement following the loop.

C. The END DO statement has the syntax:

```
END DO
```

An END DO statement terminates the range of a DO or DO WHILE statement. The END DO statement must be used to terminate a DO block if the DO or DO WHILE statement does not contain a statement label. It may also be used as a labeled terminal statement if the DO or DO WHILE statement does contain a terminal statement label.

7. Bit manipulation functions.

8. Hexadecimal and octal constants and Z and O format edit descriptors applicable to all data types - The O field descriptor specifies octal integer values; the Z field descriptor specifies hexadecimal (base 16) values.

9. DEFINE FILE, FIND, ENCODE, DECODE, DELETE, REWRITE, and UNLOCK statements - The DEFINE FILE statement describes direct-access sequential files that are associated with a logical unit number.

   The ENCODE and DECODE statements transfer date between variables or arrays in internal storage and translate that data from internal to character form, or from character to internal form, according to format specifiers.

   The DELETE statement deletes records in relative files and in indexed files. Specifically, it causes a record to be marked as deleted; records so marked are not accessible to subsequent READ or REWRITE statements.

The REWRITE statement transfers output data from internal storage to the current record in an indexed file.

The UNLOCK statement unlocks records in a relative or indexed file. When a record is locked, it cannot be accessed by any other program or logical unit.

10. ACCEPT, TYPE input/output statements - The ACCEPT statement transfers input data to internal storage from external records accessed under the sequential mode of access. The TYPE statement transfers output data from internal storage to external records accessed under the sequential mode of access.

11. USEROPEN subroutine invocation at file OPEN time - The USEROPEN parameter has the form:

    USEROPEN = p

    where p is an external function name. The USEROPEN keyword specifies a user-written external function that controls the opening of the file.

12. INCLUDE statement (described in new features).

13. NAMELIST - Directed I/O (described in new features).

14. 31-character identifiers that can include dollar sign ($) and underline (_).

15. Comments allowed at the end of each source line.

16. Debug statements in source.

17. Language elements that support the VAX-11 extended range and extended precision floating-point architectural features.

    A. 64- bit G-floating data type with an 11-bit exponent and 53-bit mantissa, which provides a range of 0.56*10**-308 to 0.09*10**308 and a precision of 15 decimal digits

    B. 128-bit H-floating data type with a 15-bit exponent and a 113-bit mantissa, which provides a range of 0.84*10**-4932 to 0.59*10**4932 and a precision of 33 decimal digits

    (To execute G- and H-floating data type extended range instructions directly on the VAX-11/780, both the KU780 and the KE780 hardware options must be

present. To execute these instructions directly on the VAX-11/750, the KU750 hardware option must be present. The VAX-11/730 does not require any additional options for G- and H-floating data type instruction execution.)

18. DIL support - The DIL (Data Interchange Library) is a set of callable subroutines that enables a COBOL or FORTRAN programmer to access and manipulate data on another VAX or a DECSYSTEM-20. To use the DIL in a multiple computer environment, the computers must be interconnected by DECnet to form a network.

3.4 ADDITIONAL FEATURES

1. Support for calls to VAX/VMS system service procedures
2. Generation of symbol tables for the VAX-11 symbolic debugger
3. Generation of cross-reference listings
4. Generation of shareable code
5. Up to 255 actual arguments in a CALL statement
6. Up to 250 named COMMON blocks per subprogram

3.5 VAX-11 FORTRAN COMPILER OPTIMIZATIONS

1. Constant folding
2. Optimizations of arithmetic IF, logical IF, and block IF-THEN-ELSE
3. Common subexpression elimination
4. Removal of invariant expressions from DO loops
5. Allocation of general registers across DO loops
6. In line expansion of statement functions

3.6 COMPATIBILITY: VAX-11 FORTRAN AND FORTRAN-66

The VAX-11 FORTRAN compiler selects FORTRAN-77 language interpretations by default. As a result, it contains certain

incompatibilities with FORTRAN implementations that are based on the previous standard, X3.9-1966. The areas affected are:

1. DO loop minimum iteration count
2. EXTERNAL statement
3. OPEN statement BLANK keyword default
4. OPEN statement STATUS keyword default
5. X format edit descriptor.

3.6.1 DO Loop Minimum Iteration Count

The /F77 command qualifier controls the interpretation of the DO loop minimum iteration count. In FORTRAN-77, the body of a DO loop is not executed if the end condition of the loop is already satisfied when the DO statement is executed. In most implementations of FORTRAN-66, the body of a DO loop is always executed at least once.

3.6.2 EXTERNAL Statement

The /F77 command qualifier controls the interpretation of the EXTERNAL statement. In FORTRAN-66 the EXTERNAL statement is used to specify that a symbolic name is the name of either a user-defined external procedure or a FORTRAN-supplied function. In FORTRAN-77, the INTRINSIC and EXTERNAL statements are used to accomplish this function.

3.6.3 OPEN Statement Keyword Defaults

In FORTRAN-77 the OPEN statement BLANK keyword controls the interpretation of blanks in number input fields. The FORTRAN-77 default is BLANK='NULL'; that is, blanks in numeric input fields are ignored. The FORTRAN-66 interpretation of blanks in numeric input fields is equivalent to BLANK='ZERO'.

If a logical unit is opened without an explicit OPEN statement, VAX-11 FORTRAN and FORTRAN-66 both provide a default equivalent to BLANK='ZERO'.

The BLANK keyword affects the treatment of blanks in numeric input fields read with the D, E, F, G, I, O, and Z field descriptors. If BLANK='NULL' is in effect, embedded and trailing blanks are ignored; the value is converted as if the nonblank characters were right-justified in the field. If BLANK='ZERO' is in effect, embedded and trailing blanks are treated as zeros.

3.6.4 OPEN Statement Status Keyword Default

In FORTRAN-77, the OPEN statement STATUS keyword specifies initial status of the file ('OLD', 'NEW', 'SCRATCH', or 'UNKNOWN'). The FORTRAN-77 default is STATUS='UNKNOWN'; that is, an existing file is opened or a new file is created if the

file does not exist. If you use the /F77 command qualifier and you do not specify STATUS (or TYPE) in an OPEN statement, the compiler issues an informational message to warn you that it is using a default of STATUS='UNKNOWN'. It is advisable to include an explicit STATUS (or TYPE) keyword in every OPEN statement.

3.6.5 X Format Edit Descriptor

The nX edit descriptor causes transmission of the next character to or from a record to occur at the position n characters to the right of the current position. In a FORTRAN-77 output statement, character positions that are skipped are not modified, and the length of the output record is not affected. However, in many FORTRAN-66 implementations, the X edit descriptor writes blanks and may extend the output record.

CHAPTER 4
PDP-11 FORTRAN

4.1 INTRODUCTION

PDP-11 FORTRAN is an extended implementation of the ANSI subset FORTRAN-77 standard. Version 4 contains all the features of the ANSI FORTRAN-77 subset, many of the full-set language features, and extensions that are not included in the ANSI FORTRAN-77 standard.

This chapter describes the new features supported in Version 4 and also outlines the major features that are extensions to the standards. This information, as well as more detailed information, is found in the following documents.

1. PDP-11 FORTRAN-77 Language Reference Manual, (September 1981), AA-19791-TC

2. PDP-11 FORTRAN-77 User's Guide (September 1981), AA-1884D-TC

3. PDP-11 FORTRAN-77 Installation Guide/Release Notes, AA-K503B-TC

The PDP-11 FORTRAN-77 Object Time System Reference Manual (AA-1874C-TC) is also of interest to PDP-11 FORTRAN-77 programmers.

The PDP-11 FORTRAN-77 Language Reference Manual describes the syntax and semantics of the FORTRAN-77 implementation of PDP-11 FORTRAN. It does not, however, present information specific to any operating system.

The PDP-11 FORTRAN-77 User's Guide contains the information necessary to create, link, and execute PDP-11 FORTRAN-77 programs on a PDP-11 processor. Programming information is provided for the RSX-11M/M-PLUS, IAS, and RSTS/E operating systems.

The PDP-11 FORTRAN-77 Installation Guide/Release Notes describes the procedures for installing PDP-11 FORTRAN on the RSX-11M/M-PLUS, RSTS/E, and IAS operating systems.

4.2 NEW FEATURES

Among the major features defined by the new ANSI subset language FORTRAN standard and not found in either the previous ANSI standard or previous versions of Digital PDP-11 FORTRAN are:

1. CHARACTER data type

2. Block IF construct, including IF...THEN, ELSE IF, ELSE, and END IF statements, for conditional execution of blocks of statements.

4.2.1 Features Supported for Character Data

Version 4 supports character data as specified by the subset language FORTRAN-77 standard.

4.2.1.1 Character Constants - The length of a character constant must be in the range 1 through 255.

4.2.1.2 Substring Reference and Definition - Version 4 supports character substrings as outlined in the full-language subset. (Note that substrings are not included in the subset.)

4.2.1.3 Character Expressions - A character operand can be character constant, character variable, character array element, or character substring. A character expression has the form:

character operand

and can be enclosed in parentheses. Note that the concatenation operator is not included in the subset (nor in Version 4).

4.2.1.4 Character Variables and Arrays - The length specification in the character type-statement can be an unsigned integer constant or an integer-constant expression enclosed in parentheses. [An asterisk in parentheses (*) is not allowed]. When you specify CHARACTER*len, the length specification must be in the range 1 to 255.

4.2.1.5 Character Relational Expressions - Version 4 conforms to the ANSI-77 full language standard.

4.2.1.6 Character Assignment Statements - Version 4 conforms to the ANSI-77 subset language standard. Note that you cannot assign a numeric value to a character variable, array element, or substring.

4.2.1.7 Character Variables and Constants in DATA Statements - Version 4 (and the ANSI subset language) do not support implied DO-lists in DATA statements.

4.2.1.8 Character Functions - Version 4 supports LEN, INDEX, ICHAR, LLT, LLE, LGT, and LGE. Note that CHAR is not supported.

4.2.2 IF THEN ELSE Statements
Version 4 supports the block IF, ELSE IF, ELSE, and END IF statements.

4.3 FEATURES OF THE FULL-LANGUAGE FORTRAN AS DEFINED BY THE ANSI STANDARD

4.3.1 Exponentiation Forms
(This includes double-precision and complex forms.) Table 4-1 summarizes the allowed combinations of data types of base and exponent, and the data type of the result of exponentiation. The new features are underlined.

Table 4-1 Allowed Combinations of Types of Base and Exponents

Base	Exponent Integer	Real	Double	Complex
Integer	Integer	Real	Double	Complex
Real	Real	Real	Double	Complex
Double	Double	Double	Double	No
Complex	Complex	Complex	No	Complex

4.3.2 Format Edit Descriptors
(This includes S, SP, SS, T, TL, TR, Iw.m, and Gw.dEe.) Version 4 conforms to the full FORTRAN-77 standard.

4.3.3 INTRINSIC and EXTERNAL Statements

Normally, the name of an intrinsic function refers to the FORTRAN library function with that name. However, the name can refer to a user-defined function under any of the following conditions:

1. The name is used in a function reference with arguments of a different data type from that normally used.

2. The name appears in an EXTERNAL statement.

The EXTERNAL and INTRINSIC statements enable the programmer to use subprogram names as actual arguments to other subprograms. The semantics of the EXTERNAL statement are different in FORTRAN 77 than in previous versions of PDP-11 FORTRAN. In previous versions, the appearance of an intrinsic function name in an EXTERNAL statement caused the processor to treat the name as the name of an intrinsic function. In FORTRAN 77, the appearance of an intrinsic function name in an EXTERNAL statement causes the processor to treat the name as the name of an external function. In previous versions, an intrinsic function name had to be preceded by an asterisk to be treated as an external function. The /NOF77 switch allows the programmer to select the previous semantics, rather than FORTRAN-77 semantics. The following shows the equivalent statements.

```
/F77                    /NOF77
EXTERNAL ext            EXTERNAL ext
EXTERNAL int            EXTERNAL *int
INTRINSIC int           EXTERNAL int
```

Except when they are used in an EXTERNAL statement, intrinsic function names are local to the program unit that refers to them. Thus, they can be used for other purposes in other program units. In addition, the data type of an intrinsic function does not change if you use an IMPLICIT statement or an explicit type declaration to change the implied data type rules.

You cannot have an intrinsic function and a user-defined function with the same name in the same program unit.

4.3.4 Generic Function Selection Based on Argument Data Type for FORTRAN-Defined Functions

Some intrinsic functions perform the same computation but handle different data types. These functions are references with the same generic name. A generic-function reference refers to the category of the computation to be performed, not to a specific function within the category. The selection of a specific function, that is, the actual computing procedure for a specific data type, is left to the compiler, which chooses a specific function within a category on the basis of the data type of the

relevant actual argument. For example, if D is a double-precision variable, the generic function reference SIN(D) refers to the double-precision sine function. You need not write DSIN(D).

Generic function references are independent from one another. Therefore, you could use both of the function references SIN(X) where X is a real variable, and SIN(D) where D is a double-precision variable, in the same program unit.

The intrinsic and generic functions are described in section 15.10 of the FORTRAN-77 standard.

4.3.5 PARAMETER Statements

Version 4 provides both the FORTRAN-77 and the earlier form of the PARAMETER statement. The list in the earlier form of the PARAMETER statement is not bounded with parentheses, and the form of the constant (rather than typing of the symbolic name) determines the data type of the variable.

4.3.6 Generalized DO Loop Parameters

Version 4 conforms to the full FORTRAN-77 standard. Moreover, "extended range DO loops" are supported.

4.3.7 Lower and Upper Bounds Specification in Array Declarators

Version 4 conforms to the full FORTRAN-77 standard.

4.3.8 Optional Syntax for I/O Statements (UNIT= and FMT=)

Version 4 conforms to the full FORTRAN-77 standard.

4.4 EXTENSIONS TO THE ANSI STANDARD

1. Language elements for keyed and sequential access to RMS multikey ISAM files.

2. DEFINE FILE, FIND, ENCODE, DECODE, DELETE, REWRITE, and UNLOCK statements - The DEFINE FILE statement describes direct-access sequential files that are associated with a logical unit number. The OPEN statement, which can also be used to describe direct-access sequential files, is the preferred statement.

   The FIND statement positions a direct-access file on a specified unit to a particular record. No data transfer takes place.

   The ENCODE and DECODE statements transfer data between variables or arrays in internal storage and translate

that data from internal to character form or from character to internal form, according to format specifiers. Similar results can be accomplished using internal files with formatted sequential WRITE and READ statements.

The DELETE statement deletes records in relative files and in indexed files. Specifically, it causes a record to be marked as deleted; records so marked are not accessible to subsequent READ or REWRITE statements.

The REWRITE statement repositions a sequential file currently open for sequential or append access to the beginning of the file.

The UNLOCK statement unlocks records in a relative or indexed file. When a record is locked, it cannot be accessed by any other program or logical unit.

3. TYPE and ACCEPT input/output statements - The TYPE statement transfers output data from internal storage to external records accessed under the sequential mode of access. The ACCEPT statement transfers input data to internal storage from external records accessed under the sequential mode of access.

4. Comments permitted at the end of each source line.

5. INCLUDE statement - The INCLUDE statement specifies that the contents of a designated file are to be incorporated into a compilation directly following the statement. INCLUDE has no effect on program execution.

6. BYTE data type - BYTE and LOGICAL*1 are synonymous.

7. Explicit specification of storage allocation units for data types (e.g., INTEGER*4).

8. Hexadecimal and octal constants.

9. Virtual array support for systems with memory management directives. Virtual arrays are memory-resident and require enough main memory to contain all elements of all arrays.

10. O and Z format edit descriptors - The O field descriptor specifies octal integer values; the Z field descriptor specifies hexadecimal (base 16) values.

4.5 ADDITIONAL FEATURES

The PDP-11 FORTRAN-77 compiler produces direct PDP-11 machine code optimized for execution-time efficiency on a PDP-11 with a floating-point processor. PDP-11 FORTRAN-77 compiler optimizations include:

1. Optimizations of arithmetic and logical IF statements
2. Common subexpression elimination
3. Removal of invariant expressions from DO loops
4. Allocation of processor registers across block IF constructs and DO loops.

4.6 COMPATIBILITY: PDP-11 FORTRAN AND FORTRAN-66

The PDP-11 FORTRAN compiler selects FORTRAN-77 language interpretations by default. As a result, it contains certain incompatibilities with FORTRAN implementations that are based on the previous standard, X3.9-1966. The areas affected are:

1. DO loop minimum iteration count
2. EXTERNAL statement
3. OPEN statement BLANK keyword default
4. OPEN statement STATUS keyword default
5. Blank common block PSECT
6. X format edit descriptor.

4.6.1 DO Loop Minimum Iteration Count

The /F77 command qualifier controls the interpretation of the DO loop minimum iteration count. In FORTRAN-77, the body of a DO loop is not executed if the end condition of the loop is already satisfied when the DO statement is executed. In most implementations of FORTRAN-66, the body of a DO loop is always executed at least once.

4.6.2 EXTERNAL Statement

The /F77 command qualifier controls the interpretation of the EXTERNAL statement. In FORTRAN-66 the EXTERNAL statement is used to specify that a symbolic name is the name of either a user-defined external procedure or a FORTRAN-supplied function. In FORTRAN-77 the INTRINSIC and EXTERNAL statements are used to accomplish this function.

4.6.3 OPEN Statement Keyword Defaults

In FORTRAN-77 the OPEN statement BLANK keyword controls the interpretation of blanks in the number input fields. The FORTRAN-77 default is BLANK='NULL'; that is, blanks in numeric

input fields are ignored. The FORTRAN-66 interpretation of blanks in numeric input fields is equivalent to BLANK='ZERO'.

If a logical unit is opened without an explicit OPEN statement, PDP-11 FORTRAN and FORTRAN-66 both provide a default equivalent to BLANK='ZERO'.

The BLANK keyword affects the treatment of blanks in numeric input fields read with the D, E, F, G, I, O, and Z field descriptors. If BLANK='NULL' is in effect, embedded and trailing blanks are ignored; the value is converted as if the nonblank characters were right-justified in the field. If BLANK='ZERO' is in effect, embedded and trailing blanks are treated as zeros.

4.6.4 OPEN Statement Status Keyword Default

In FORTRAN-77, the OPEN statement STATUS keyword specifies initial status of the file ('OLD', 'NEW', 'SCRATCH', or 'UNKNOWN'). The FORTRAN-77 default is STATUS='UNKNOWN'; that is, an existing file is opened or a new file is created if the file does not exist. If you use the /F77 command qualifier and you do not specifiy STATUS (or TYPE) in an OPEN statement, the compiler issues an informational message to warn you that it is using a default of STATUS='UNKNOWN'. It is advisable to include an explicit STATUS (or TYPE) keyword in every OPEN statement.

4.6.5 Blank Common Block PSECT (.$$$$.)

Under PDP-11 FORTRAN-77, the blank common block PSECT (.$$$$.) has the SAV attribute. The SAV attribute on a PSECT has the effect of pulling that PSECT into the root segment of an overlay.

4.6.6 X Format Edit Descriptor

The nX edit descriptor causes transmission of the next character to or from a record to occur at the position n characters to the right of the current position. In a FORTRAN-77 output statement, character positions that are skipped are not modified and the length of the output record is not affected. However, in many FORTRAN-66 implementation, the X edit descriptor writes blanks and may extend the output record.

CHAPTER 5
A COMPARISON OF FORTRAN LANGUAGE FEATURES

Tables 5-1 through 5-16 provide comparative FORTRAN language features. Refer also to Language Fundamentals (AA-M460A-RK).

Table 5-1 Programming Considerations

	<—AMERICAN—>		<——PDP-8——>			<—PDP-11—>		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX	RSX	VMS	TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSTS/E RT-11 IAS (FOR)	RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
Maximum number of dimensions allowed for an array	3	7	2	7	7	7	7	7	Infinity
Mixed mode expressions	-	X	-	X	X	X	X	X	X
Double exponentiation (e.g., A**B**C) permitted	-	X[2]	-	X[2]	X[2]	X[2]	X[2]	X[2]	X[2]
Statement number size (characters)	1-5	1-5	1-5	1-5	1-5	1-5	1-5	1-5	1-5
Maximum level of nesting for DO loops	-	-	20	10	10	-[3]	20	20	Infinity
Maximum number of characters allowed in a PAUSE message	-	-	-	-	-	255	255	255	Infinity
Generalized subscripts permitted	-	X	X	X	X	X	X	X	X
Adjustable dimensions permitted in subprogram	X	X	-	X	X	X	X	X	X
Specification statement can follow first executable statement	-	-	-	-	-	X	-	-	X[4]
Generic function selection	-	X	-	-	-	-	X	X	X

2 Right to left evaluation.
3 Undefined.
4 Produces warning.

Table 5-1 Programming Considerations (Cont)

	<--AMERICAN--> NATIONAL STANDARDS FORTRAN X3.9 1966	-77 (Full Language)	<-----PDP-8-----> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<---PDP-11---> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
Statement functions can follow executable statements or precede related specification statements	-	-	-	X	X	X	-	-	X[4]
Maximum number of continuation lines	19	19	Infinity	5	5	Infinity	0-99	0-99	Infinity
Embedded blanks permitted in key words	X	X	X	X	X	X	X	X	X
Key words reserved by the compiler	-	-	-	-	-	-	-	-	-
Maximum characters in a symbolic name	6	6	5	6	6	6[3]	6[3]	31[5]	6[3]
Maximum level of nesting for implied DO loops	-[1]	-[1]	2	10	10	-[2]	-[2]	-[2]	Infinity
Comment line starts with	C	C,*	C	C	C	C,D,!	C,D, *,!	C,D, *,!	C,$,*, /,D,!
Source code in EBCDIC	-[1]	-[1]	-	-	-	-	-	-	-
Source code in BCD	-[1]	-[1]	-	-	-	-	-	-	-
Source code in ASCII	-[1]	-[1]	X	X	X	X	X	X	X
End-of-line comments delimited by "!"	-	-	-	-	-	X	X	X	X
Label on any statement		X				X[6]	X	X	X
INCLUDE from source file	-	-				-	X	X	X
INCLUDE from test library	-	-				-	-	X	-

1 Not specified.
2 Undefined.
3 Warning given for >6.
4 Warning given.
5 Warning given for over 31.
6 Not on a function statement.

Table 5-2 Subprogram Statements

	<—AMERICAN—> NATIONAL STANDARDS		<——PDP-8——>			<—PDP-11—>		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
BLOCK DATA	X	X[1]	-	X	X	X[1]	X[1]	X[1]	X[1]
ENTRY entry-point [(arg[,arg]...)]	-	X	-	-	-	-	X[2]	X[2]	X[2]
FUNCTION function-name (arg[,arg]...)	X	X	X	X	X	X[2]	X[2]	X[2]	X[2]
function-name (arg[,arg]...)= arithmetic-expression	X	X	-	X	X	X	X	X	X
function-name (arg[,arg]...)= logical expression	X	X	-	X	X	X	X	X	X
function-name (arg[,arg]...)= character expression	-	X	-	-	-	-	-	X	X
[{ INTEGER / REAL / DOUBLE PRECISION / COMPLEX / LOGICAL } FUNCTION func-name [*length] (arg[,arg]...)	X[3]	X[3,7]	-	X	X[4]	X[2]	X[2]	X[2,6]	X[5,2,7]
SUBROUTINE subroutine-name [(arg[,arg]...)]	X	X	X	X	X	X	X	X	X

1 Can have name.
2 Permit () and no args.
3 *length not permitted.
4 INTEGER, REAL, LOGICAL only.
5 *length ignored in most cases; flagged with warning.
6 Also types DOUBLE COMPLEX, CHARACTER
7 Also type CHARACTER [*length].

Table 5-3 Constant and Variable Types

	<--AMERICAN--> NATIONAL STANDARDS		<-----PDP-8----->			<---PDP-11--->		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
CONSTANT FORMS									
Integer	X	X	X	X	X	X	X	X	X
Real	X	X	X	X	X	X	X	X	X
Double-precision	X	X	-	X	-	X	X	X	X
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	-
Logical	X	X	-	X	X	X	X	X	X
Literal (preceded by H)	X[3]	-	X[5]	X[5]	X[5]	X[5]	X[5]	X[5]	X[5]
Literal (enclosed in single quotes)	-	X[7]	X[5]	X[5]	X[5]	X[5]	X[7]	X[7]	X[7]
Hexadecimal	-	-	-	-	-	-	X	X	-
Octal	-	-	-	X[1]	X	X[2]	X[2]	X[2]	X[6]
Quadruple precision	-	-	-	-	-	-	-	X	-
Radix 50	-	-	-	-	-	X[1]	X[1]	X[1]	-

1 In DATA statements only.
2 ["ddd] allowed anywhere [Oddd] is allowed in DATA statements.
3 In DATA, FORMAT, and subroutine arguments only.
5 Denotes Hollerith literal.
6 "ddd allowed anywhere.
7 Denotes CHARACTER constant.

Table 5-4 Data Types

	<--AMERICAN-->		<-----PDP-8----->			<---PDP-11--->		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX RSTS/E	RSX RSTS/E	VMS	TOPS-20
	X3.9 1966	FORTRAN -77(Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RT-11 IAS (FOR)	IAS (F77)	VAX/11 FORTRAN	F-10
Floating point									
REAL	X	X	X	X	X	X	X	X	X
REAL*4	-	-	-	-	-	X	X	X	X
REAL*8	-	-	-	-	-	X	X	X	X[5]
REAL*16	-	-	-	-	-	-	-	X	X[5]
DOUBLE PRECISION	X	X	-	X[7]	-	X	X	X	X
Complex									
COMPLEX	X	X	-	X[7]	-	X	X	X	X
COMPLEX*8	-	-	-	-	-	X	X	X	X[5]
COMPLEX*16	-	-	-	-	-	-	-	X	X[5]
DOUBLE COMPLEX	-	-	-	-	-	-	-	X	-
Character									
CHARACTER*(length)	-	X	-	-	-	-	X	X	X
CHARACTER*(*)	-	X	-	-	-	-	-	X	X
Logical									
LOGICAL	X	X	-	X	X	X[2][4]	X[1][4]	X[1][4]	X[5]
LOGICAL*1	-	-	-	-	-	X[4]	X[4]	X[4]	X[5]
LOGICAL*2	-	-	-	-	-	-	X	X	X[5]
LOGICAL*4	-	-	-	-	-	X	X	X	X
Integer									
INTEGER	X	X	X	X	X	X[2]	X[1]	X[1]	X
INTEGER*1	-	-	-	-	-	-	-	-	-[5]
INTEGER*2	-	-	-	-	-	X[3]	X	X	X[5]
INTEGER*4	-	-	-	-	-	X[3]	X	X	X

1 Defaults to *2 or *4 at compile time.
2 Implemented as *4.
3 Four-byte allocation; two-byte precision.
4 BYTE is synonym for LOGICAL*1.
5 Produces warning; * length ignored.
7 Only with FPP hardware.

Table 5-5 Specification Statements

	<--AMERICAN-->		<-----PDP-8----->			<---PDP-11--->		VAX/	TOPS-10
		NATIONAL STANDARDS FORTRAN				RSX RSTS/E RT-11 IAS	RSX RSTS/E IAS	VMS	TOPS-20
	X3.9 1966	-77(Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	(FOR)	(F77)	VAX/11 FORTRAN	F-10
PROGRAM name	-	X	-	-	-	X	X	X	X
NAMELIST /namelist-name/ name [,name...] [[,] /namelist-name/ name [,name...]]...	-	-	-	-	-	-	-	X	X
PARAMETER var=constant,...[1]	-	-	-	-	-	-	X	X	X
PARAMETER (var=exp,...)[2]	-	X	-	-	-	-	X	X	X
SAVE statement	-	X	-	-	-	-	X	X	X
IMPLICIT type (characters [,type(characters)]...	-	X	-	-	-	X	X	X	X
{INTEGER / REAL / DOUBLE PRECISION / COMPLEX / LOGICAL}	X	X	-	X	-	X	X	X[4]	X[5]
{name / array-declarator / function-name}									
[{,name / ,array-declarator / ,function-name} ...]									
{INTEGER [*2] / REAL [*8] / COMPLEX [*16] / LOGICAL [*1]}	-	-	-	-	-	-	-	X[4]	X[3,5,6]
{name / array-decl. / function-name}									
[*length] [/initial-value/]									

1 Type of symbolic name determined by constant.
2 Type of symbolic name determined by first letter, IMPLICIT, and type declarations.
3 Warnings produced for INTEGER*2, COMPLEX*16, LOGICAL*1.
4 Also allows types DOUBLE COMPLEX, CHARACTER.
5 Also allows type CHARACTER.
6 Initial-value not allowed.

Table 5-5 Specification Statements (Cont)

	<--AMERICAN-->		<------PDP-8------>			<---PDP-11--->		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX RSTS/E	RSX RSTS/E	VMS	TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RT-11 IAS (FOR)	IAS (F77)	VAX/11 FORTRAN	F-10
COMMON{name / array-decl.}[{,name / ,array-decl.}]...	X	X	X	X	X	X	X	X	X
COMMON/block-name/{name / array-decl.}[{,name / ,array-decl.}] ...[/block-name/...]...	X	X	-	X	X	X	X	X	X
DIMENSION array-decl. [,array-decl.]...	X	X	X	X	X	X	X	X	X
EQUIVALENCE (name[,name]...) [,(name[,name]...)]...	X	X	X	X	X	X	X	X	X
EXTERNAL{subprogram-name / external-proc-name}[{,subprogram-name / ,external-proc-name}]...	X	X	-	X	X	X	X	X	X
EXTERNAL {[*]subprogram-name / [*]external-proc-name}...	X	-	-	-	-	-	X	X	X[1]
Array declarators allow upper and lower dimension bounds	-	X	-	-	-	-	X	X	X
INTRINSIC {subprogram-name}	-	X	-	-	-	-	X	X	X

1 & can be used in place of *.

Table 5-6 Data Initialization Statement

	<--AMERICAN--> NATIONAL STANDARDS X3.9 1966	FORTRAN -77 (Full Language)	<------PDP-8------> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<---PDP-11---> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
DATA name [,name]... /[number*]value [,[number*]value]... [,/[,name...]...]...	X	X[1]	-	X	X	X[1]	X[1]	X[1]	X[1]
Implied DO construct	-	X	X	X	X	-	-	X	X
Data values converted to type of name	-	X	-	-	-	-	X	X	X
Data initialization allowed in type declaration statements	-	-	-	-	-	-	-	X	-

1 Name may be an array name implying all elements of array.

Table 5-7 Relational Operators

	<--AMERICAN--> NATIONAL STANDARDS X3.9 1966	FORTRAN -77 (Full Language)	<------PDP-8------> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<---PDP-11---> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
.GT.	X	X	-	X	X	X	X	X	X
.GE.	X	X	-	X	X	X	X	X	X
.LT.	X	X	-	X	X	X	X	X	X
.LE.	X	X	-	X	X	X	X	X	X
.EQ.	X	X	-	X	X	X	X	X	X
.NE.	X	X	-	X	X	X	X	X	X
>	-	-	-	-	-	-	-	-	X
>=	-	-	-	-	-	-	-	-	X
<	-	-	-	-	-	-	-	-	X
<=	-	-	-	-	-	-	-	-	X
==	-	-	-	-	-	-	-	-	X
#	-	-	-	-	-	-	-	-	X

Table 5-8 Logical Operators

	<--AMERICAN-->		<------PDP-8------>			<---PDP-11--->		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX RSTS/E	RSX RSTS/E	VMS	TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RT-11 IAS (FOR)	IAS (F77)	VAX/11 FORTRAN	F-10
.NOT.	X	X	-	X	X	X	X	X	X
.AND.	X	X	-	X	X	X	X	X	X
.OR.	X	X	-	X	X	X	X	X	X
.XOR.	-	-	-	X	X	X	X	X	X
.EQV.	-	X	-	X	X	X	X	X	X
.NEQV.	-	X	-	-	-	-	X	X	X

Table 5-9 Assignment Statements

	<--AMERICAN-->		<------PDP-8------>			<---PDP-11--->		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX RSTS/E	RSX RSTS/E	VMS	TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RT-11 IAS (FOR)	IAS (F77)	VAX/11 FORTRAN	F-10
variable = arithmetic expression	X	X	X	X	X	X	X	X	X
variable = logical expression	X	X	-	X	X	X	X	X	X
ASSIGN statement-number TO variable	X	X[1]	-	X	X	X	X[1]	X[1]	X[1]
variable = character expression	-	X	-	-	-	-	X[2]	X	X

1 Statement label may be label of a format statement.
2 Character expression must be variable, substring, or constant.

Table 5-10 Control Statements

	<--AMERICAN-->		<-----PDP-8----->			<---PDP-11--->		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX RSTS/E	RSX RSTS/E	VMS	TOPS-20
	X3.9 1966	FORTRAN -77(Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RT-11 IAS (FOR)	IAS (F77)	VAX/11 FORTRAN	F-10
CALL subroutine-name [(argument [,arg[...)]	X	X	X	X	X	X[1]	X[1]	X[1]	X
CONTINUE	X	X	X	X	X	X	X	X	X
DO statement-number control-variable = initial-value, test-value [,increment]	X	X	X	X	X	X[2,4,8]	X[2,8,9,10]	X[2,8,9]	X[2,8,9]
END	X	X	X	X	X	X	X	X	X
GO TO statement-number	X	X	X	X	X	X	X	X	X
GO TO (statement-number, statement-number [,statement-number]...) [,] variable	X	X[6]	X	X	X	X[5,6]	X[5,6]	X[5,6]	X[5,6]
GO TO variable [[,] (statement number, statement-number [,statement-number]...)]	X	X	-	X	X	X	X[7]	X	X[7]
IF (arithmetic-expression) statement-number, statement number, statement-number	X	X	X	X	X	X	X	X	X
IF (logical-expression) executable-statement	X	X	-	X	X	X	X	X	X

1 Null argument permitted.
2 General expressions permitted.
3 Iteration count computer but minimum of one iteration.
4 Variable and expressions are of type INTEGER.
5 Index may be a general expression which will be converted automatically to integer if expression of other type.
6 If index out of bounds, then acts as continue.
7 If list present and assigned label not in list, then acts as continue.
8 Optional comma after statement number.
9 Compiler switch to determine minimum iteration count (0 or 1).
10 Statement number optional.

Table 5-10 Control Statements (Cont)

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	-77 (Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
IF (logical-expression) statement-number, statement number	-	-	-	-	-	-	-	-	-
IF-THEN-ELSE-ENDIF	-	X	-	-	-	-	X	X	X
DO-WHILE	-	-	-	-	-	-	-	X	X
END DO	-	-	-	-	-	-	-	X	X
PAUSE (one to six octal digits)	X	-	-	-	-	X	X	X	X[2]
PAUSE (one to five decimal digits)	-	X	X	X	X	X	X	X	X[2]
PAUSE 'message'	-	X	-	-	-	X	X	X	X
STOP (one to six octal digits)	X	-	-	-	-	X	X	X	X[2]
STOP (one to five decimal digits)	-	X	-	-	-	X	X	X	X[2]
STOP 'message'	-	X	-	-	-	X	X	X	X
Tracing after pause	-	-	-	-	-	-	-	X[4]	X
RETURN	X	X	X	X	X	X	X	X	X
RETURN (expression)	-	X	-	-	-	-	-	X	X

2 Up to 12 digits.
4 PAUSE enters command processor. User can STOP, CONTINUE, ASSIGN, SHOW, DEBUG, etc.

Table 5-11 Sequential I/O Statements

	<—AMERICAN—> NATIONAL STANDARDS		<——PDP-8——>			<—PDP-11—>		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
N = namelist-name u = unit number f = label of a format statement k = an I/O list * indicates list-directed I/O									
ACCEPT f,k	-	-	-	-	-	X	X	X	X
ACCEPT f	-	-	-	-	-	X	X	X	X
ACCEPT*,k	-	-	-	-	-	X	X	X	X
ACCEPT N	-	-	-	-	-	-	-	X	X
BACKSPACE u	X	X	-	X	X	X	X	X	X
ENDFILE u	X	X	-	X	X	X	X	X	X
PRINT f,k	X	X	-	-	-	X	X	X	X
PRINT f	X	X				X	X	X	X
PRINT*,k	-	X				X	X	X	X
PRINT N	-	-	-	-	-	-	-	X	X
PUNCH f,k	-	-	-	-	-	-	-	-	X
PUNCH f	-	-				-	-	-	X
PUNCH*,K	-	-				-	-	-	X

Table 5-11 Sequential I/O Statements (Cont)

	<—AMERICAN—> NATIONAL STANDARDS		<——PDP-8——>			<—PDP-11—>		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
N = namelist-name u = unit number f = label of a format statement k = an I/O list * indicates list-directed I/O									
READ f,k	-	X	-	-	-	X	X	X	X
READ (u,f)[k]	X	X[2]	X	X	X	X	X[2]	X[2]	X[2]
READ (u) [k]	X	X[3]	-	X	X	X	X[3]	X[3]	X[3]
READ (u[,f][,END=S1] [,ERR=S2])[k]	-	X	-	-	-	X[1,4]	X[1,2]	X[1,2]	X[1,2]
READ*,k	-	X	-	-	-	X	X	X	X
READ (u,*[,END=S]][,ERR=S2])k	-	X	-	-	-	X[1]	X[1,3]	X[1,3]	X[1,3]
READ N	-	-	-	-	-	-	-	X	X
READ (u,N[,END=S1][,ERR=S2])	-	-	-	-	-	-	-	X[3,5]	X
REREAD f,k	-	-	-	-	-	-	-	-	X
REWIND u	X	X	-	X	X	X	X	X	X
BACKFILE u	-	-	-	-	-	-	-	-	X
SKIPFILE u	-	-	-	-	-	-	-	-	X
TYPE f[,k]	-	-	-	-	-	X	X	X	X
TYPE *[,k]	-	-	-	-	-	X	X	X	X
TYPE N	-	-	-	-	-	-	-	X	X

1 Keywords in either order.
2 Also UNIT = u, FMT = f.
3 Also UNIT = u.
4 f is mandatory (formatted only).
5 Also NML=N.

Table 5-11 Sequential I/O Statements (Cont)

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	FORTRAN -77(Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
N = namelist-name u = unit number f = label of a format statement k = an I/O list * indicates list-directed I/O ary = array, array element, or variable									
WRITE (u,f)[k]	X	X[3]	X	X	X	X	X[3]	X[3]	X[3]
WRITE (u)[k]	X	X[4]	-	X	X	X	X[4]	X[4]	X[4]
WRITE (u[,f][,END=S1] [,ERR=S2])k	-	X	-	-	-	X[1,5]	X[1,3]	X[1,3]	X[1,3]
WRITE (u,*[,END=S1] [,ERR=S2])k	-	X[1,4]	-	-	-	X[1]	X[1,4]	X[1,4]	X[1,4]
WRITE (u,N[,END=S1][,ERR=S2])	-	-	-	-	-	-	-	X[6]	X
WRITE *,k	-	-				-	-	-	X
WRITE f,k	-	-				-	-	-	X
WRITE f	-	-				-	-	-	X
SKIPRECORD u	-	-	-	-	-	-	-	-	X
UNLOAD u	-	-	-	-	-	-	-	-	X
ENCODE/DECODE (cnt,fmt,ary)k	-	-	-	-	-	X	X	X	X

1 Keywords in either order.
3 Also UNIT = u, FMT = f.
4 Also UNIT = u.
5 f is mandatory (formatted only).
6 Also NML=N.

Table 5-12 File Control Statements

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	FORTRAN -77 (Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
OPEN	-	X	X[1]	-	-	X	X	X	X
CLOSE	-	X	X[1]	-	-	X	X	X	X
INQUIRE	-	X	-	-	-	-	-	X	-

1 Library subroutines.

Table 5-13 Direct Access I/O Statements

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	FORTRAN -77 (Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
u = unit number n = number of records in file rs = length in 16-bit words of record c = indicates unformatted v = associated variable									
DEFINE FILE u(n,rs,c,v) [,u_2...]...	-	-	-	X	X	X[2,3]	X[2,3]	X[2,3]	-
FIND (u'r)	-	-	-	-	-	X	X[10]	X[10]	X[10]
READ (u'r[,f]) [k]	-	X[9]	-	X	X	X[2]	X[11]	X[11]	X[11]
READ (u'r[f] [,END=S1] [,ERR=S2]) [K]	-	X[9]	-	-	-	X[2,5]	X[11]	X[11]	X[11]
WRITE (u'r[,f]) [k]	-	X[9]	-	X	X	X[2]	X[11]	X[11]	X[11]
WRITE (u'r[,f][,END=S1] [,ERR=S2]) [k]	-	X[9]	-	-	-	X[2,5]	X[11]	X[11]	X[11]
DELETE (u'r[,ERR=s2])	-	-	-	-	-	-	X[10]	X[10]	-
FORMAT statement:									
Statement-number FORMAT (format-spec)	X	X	X	X	X	X	X	X	X

1 Call DEFINE FILE with different format of arguments.
2 Unformatted only.
3 Record size measured in 16-bit words (= 1/2 storage unit).
4 Record size measured in storage units.
5 END= option not allowed.
6 Warning for #.
8 END=S1 syntax used if record number is outside of file.
9 REC=r is syntax for direct access.
10 Also UNIT = u, REC = r.
11 Also UNIT = u, REC = r, FMT = f.

Table 5-14 Indexed I/O Statements

	<--AMERICAN--> NATIONAL STANDARDS FORTRAN X3.9 1966	-77(Full Language)	<-----PDP-8-----> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<---PDP-11---> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
u = unit number f = format specifier kl = key specifier i = key identifier sl = statement label s2 = statement label k = an I/O list									
READ (u[,f],KEY=kl[,KEYID=i]) [k]	-	-	-	-	-	-	X^2	X^2	-
READ (u[,f],KEY=k2[,KEYID=i] [,END=Sl] [,ERR=S2]) [k]	-	-	-	-	-	-	X^2	X^2	-
WRITE (u[,f])[1] [k]	-	-	-	-	-	-	X^2	X^2	-
WRITE (u[,f][,END=sl] [,ERR=s2]) [k]	-	-	-	-	-	-	X^2	X^2	-
REWRITE (u[,f]) [k]	-	-	-	-	-	-	X^2	X^2	-
REWRITE (u[,f][,END=sl] [,ERR=2]) [k]	-	-	-	-	-	-	X^2	X^2	-
DELETE (u[,ERR=s2])	-	-	-	-	-	-	X^3	X^3	-
UNLOCK (u[,ERR=s2])	-	-	-	-	-	-	X^3	X^3	-

1 Syntax identical to sequential write.
2 Also UNIT = u, FMT = f.
3 Also UNIT = u.

Table 5-15 Format and Types of Conversion

	<--AMERICAN-->		<------PDP-8------>			<---PDP-11--->		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	NATIONAL STANDARDS FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
A (alphanumeric)	X	X	X	X	X	X	X	X	X
D (real D decimal exponent)	X	X	-	X	-	X	X	X	X
E (real E decimal exponent)	X	X	X	X	X	X	X	X	X
F (real, no exponent)	X	X	X	X	X	X	X	X	X
G (real)	X	X	X	X	X	X	X	X	X
H (literal)	X	X	X	X	X	X	X	X	X
'...' (literal)	-	X	X	X	X	X	X	X	X
I (integer)	X	X	X	X	X	X	X	X	X
L (logical)	X	X	-	X	X	X	X	X	X
O (octal)	-	-	-	X	X	X	X	X	X
P (scale factor)	X	X	-	X	X	X	X	X	X
Q (record length)						X	X	X	X
T (position indicator in record)	-	X	-	X	X	X	X	X	X
X (skipped data or blank)	X	X	X	X	X	X	X	X	X
Z (hexadecimal data)	-	-	-	-	-	-	X	X	X
Format specification in arrays	X	X	-	X	X	X	X	X	X

Table 5-15 Format and Types of Conversion (Cont)

	<--AMERICAN--> NATIONAL STANDARDS		<------PDP-8------>			<---PDP-11--->		VAX/ VMS	TOPS-10
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	TOPS-20 F-10
Carriage Control	X	X	X	X	X	X	X	X	X
R (right-justified A)	-	-	-	-		-	-	-	X
S (control of optional + sign)	-	X	-	-		-	X	X	X
/ (record separator)	X	X	X	X	X	X	X	X	X
: (format scan terminator)	-	X	-	-	-	X	X	X	X
$ (format separator)	-	-	-	X	X	X[3]	X[3]	X[3]	X[3]
BN (blank = null)		X					X	X	X
BZ (blank = zero)		X					X	X	X
SS (suppress optional + sign)		X					X	X	X
SP (print optional + sign)		X					X	X	X
TL (tab left)		X					X	X	X
TR (tab right)		X					X	X	X

3 Used as carriage control character also.

Table 5-16 Library Functions

	<--AMERICAN--> NATIONAL STANDARDS		<------PDP-8------>			<---PDP-11--->		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
Absolute value:									
Real	X	X	X	X	X	X	X	X	X
Integer	X	X	X	X	X	X	X	X	X
Double-precision	X	X	-	X	-	X	X	X	X
Quad precision	-	-	-	-	-	-	-	X	-
Complex to real	X	X	-	X	-	X	X	X	X
Double complex to double-precision	-	-	-	-	-	-	-	X	X[2]
Conversion:									
Integer to real	X	X	X	X	X	X	X	X	X
Integer to double	-	X	-	-	-	-[1]	X	X	X
Integer to quad	-	-	-	-	-	-	-	X	-
Real to integer	X	X	X	X	X	X	X	X	X
Double to real (obtain most significant part)	X	X	-	X	-	X	X	X	X
Double to integer									X
Quad to real	-	-	-	-	-	-	-	X	-
Real to double	X	X	-	X	-	X	X	X	X
Real to quad	-	-	-	-	-	-	-	X	-
Byte or integer*2 to integer*4 (zero-extend)								X	
G-floating to D-floating								X	X
D-floating to G-floating								X	X

1 Available as implied conversion only.
2 Argument is a two-element double-precision array.

Table 5-16 Library Functions (Cont)

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	FORTRAN -77 (Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
Complex to real (obtain real part)	X	X	-	X	-	X	X	X	X
Double complex to real (real part)	-	-	-	-	-	-	-	X	-
Complex to real (obtain imaginary part)	X	X	-	X	-	X	X	X	X
Double complex to double real (imaginary part)	-	-	-	-	-	-	-	X	-
Real to complex	X	X	-	X	-	X	X	X	X
Double real to double complex	-	-	-	-	-	-	-	X	-
Cosine:									
Real (radians)	X	X	X	X	X	X	X	X	X
Real (degrees)	-	-	-	X	X	-	-	X	X
Double (radians)	X	X	-	X	-	X	X	X[3]	X
Quad	-	-	-	-	-	-	-	X[3]	-
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	X[4]
Hyperbolic (real and double):									
Sine	X	X	-	X	X	-	X	X[2]	X
Cosine	X	X	-	X	X	-	X	X[2]	X
Tangent	X	X	-	X	X	X[1]	X	X[2]	X

1 Real only.
2 Also quad.
3 Radian and degree versions.
4 Arguments are two-element double-precision arrays.

Table 5-16 Library Functions (Cont)

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	-77 (Full Language)	<——PDP-8——> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
Arc-sine (real and double)	X	X	-	X	X	-	X	X[2,3]	X
Arc-cosine (real and double)	X	X	-	X	X	-	X	X[2,3]	X
Arc-tangent:									
Real	X	X	X	X	X	X	X	X[3]	X
Double	X	X	-	X	-	X	X	X[3]	X
Quad	-	-	-	-	-	-	-	X[3]	-
Quotient of two arguments	X	X	-	X	X	X	X	X[3]	X
Sine:									
Real (radians)	X	X	X	X	X	X	X	X	X
Real (degrees)	-	-	-	X	X	-	-	X	X
Double (radians)	X	X	-	X	-	X	X	X[3]	X
Quad	-	-	-	-	-	-	-	X[3]	-
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	X[4]

1 Real only.
2 Also quad.
3 Radian and degree versions.
4 Arguments are two-element double-precision array.

Table 5-16 Library Functions (Cont)

	<—AMERICAN—> NATIONAL STANDARDS		<——PDP-8——>			<—PDP-11—>		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77(Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
Tangent	-	X	X	X	X	-	X	X	X
Cotangent	-	-	-	-	-	-	-	-	X
Logical functions:									
IAND	-	-	-	-	-	X[2]	X[2]	X[2]	X[2]
IOR	-	-	-	-	-	X[2]	X[2]	X[2]	X[2]
IEOR	-	-	-	-	-	X[2]	X[2]	X[2]	X[2]
NOT	-	-	-	-	-	X[2]	X[2]	X[2]	X[2]
Error function	-	-	-	-	-	-	-	-	-
Gamma function	-	-	-	-	-	-	-	-	-
Log gamma	-	-	-	-	-	-	-	-	-
Switch register	-	-	X	X	-	X	X	-	-
Complex conjugate	X	X	-	X	-	X	X	X[1]	X
Positive difference $(u_1-\text{Min}(u_1,u_2))$	X	X	-	X	X	X	X	X	X

1 Also double complex.
2 Available in all expressions as .AND., .OR., .XOR., and .NOT. operators for integer values.

Table 5-16 Library Functions (Cont)

	<--AMERICAN--> NATIONAL STANDARDS		<-----PDP-8----->			<---PDP-11--->		VAX/ VMS	TOPS-10 TOPS-20
	X3.9 1966	FORTRAN -77(Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
Bit set	-	-					-	X	-
Bit test	-	-					-	X	-
Bit clear	-	-					-	X	-
Shift	-	-					X	X	-
Circular shift	-	-					-	X	-
Insert bits	-	-					-	X	-
Move bits	-	-					-	X	-
Exponential:									
Real	X	X	X	X	X	X	X	X	X
Double	X	X	-	X	-	X	X	X	X
Quad	-	-	-	-	-	-	-	X	-
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	X[1]
Logarithm:									
Real	X	X	X	X	X	X	X	X	X
Double	X	X	-	X	-	X	X	X	X
Quad	-	-	-	-	-	-	-	X	-
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	X[1]

1 Arguments are two-element double-precision arrays.

Table 5-16 Library Functions (Cont)

	<—AMERICAN—>		<——PDP-8——>			<—PDP-11—>		VAX/	TOPS-10
	NATIONAL STANDARDS					RSX	RSX	VMS	TOPS-20
	X3.9 1966	FORTRAN -77 (Full Language)	OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	RSTS/E RT-11 IAS (FOR)	RSTS/E IAS (F77)	VAX/11 FORTRAN	F-10
Square root:									
Real	X	X	X	X	X	X	X	X	X
Double	X	X	-	X	-	X	X	X	X
Quad	-	-	-	-	-	-	-	X	-
Complex	X	X	-	X	-	X	X	X	X
Double complex	-	-	-	-	-	-	-	X	X[1]
Truncation:									
Real to real	X	X	X	X	X	X	X	X	X
Real to integer	X	X	X	X	X	X	X	X	X
Double to integer	X	X	-	X	-	X	X	X	X
Quad to integer	-	-	-	-	-	-	-	X	-
Real	X	X	-	X	X	X	X	X	X
Integer	X	X	X	X	X	X	X	X	X
Double-precision	X	X	-	X	-	X	X	X	X

1 Arguments are two-element double-precision arrays.

Table 5-16 Library Functions (Cont)

	<—AMERICAN—> NATIONAL STANDARDS FORTRAN X3.9 1966	FORTRAN -77 (Full Language)	<———PDP-8———> OS/8 FORT II	OS/8 FORT IV	OS/78 FORT IV	<—PDP-11—> RSX RSTS/E RT-11 IAS (FOR)	RSX RSTS/E IAS (F77)	VAX/ VMS VAX/11 FORTRAN	TOPS-10 TOPS-20 F-10
Maximum value (Number of arguments ≥2 for all functions)	X	X	-	X	X	X	X	X	X
Minimum value (Number of arguments ≥2 for all functions)	X	X	-	X	X	X	X	X	X
Transfer of sign:									
Real	X	X	-	X	X	X	X	X	X
Integer	X	X	-	X	X	X	X	X	X
Double-precision	X	X	-	X	-	X	X	X	X
Quad precision	-	-	-	-	-	-	-	X	-
Test sense switch	-	-	-	X	-	X	X	X	X
Random number	-	-	-	-	-	X	X	X	X
Convert sign magnitude to 2s complement and vice versa	-	-	-	-	-	-	-	-	-
Remainder of time limit	-	-	-	-	-	-	-	-	X
DIVERT run time error messages to a file									X
ERRSET controls handling of error conditions									X
DATE, TIME									X
TRACE									X

Reader's Comments

Your comments and suggestions will help us in our continuous effort to improve the quality and usefulness of our publications.

What is your general reaction to this manual? In your judgment is it complete, accurate, well organized, well written, etc.? Is it easy to use? _____

_____

_____

_____

What features are most useful? _____

_____

_____

_____

What faults or errors have you found in the manual? _____

_____

_____

_____

Does this manual satisfy the need you think it was intended to satisfy? _____

Does it satisfy *your* needs? _____ Why? _____

_____

_____

_____

☐ Please send me the current copy of the *Technical Documentation Catalog*, which contains information on the remainder of DIGITAL's technical documentation.

Name _____ Street _____
Title _____ City _____
Company _____ State/Country _____
Department _____ Zip _____

Additional copies of this document are available from:

Digital Equipment Corporation
ESD&P Order Processing
12A Esquire Road
North Billerica, MA 01862

Order No. EY-1233E-ID _____ MRO

Do Not Tear — Fold Here and Staple

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 33 MAYNARD, MA.

POSTAGE WILL BE PAID BY ADDRESSEE

Digital Equipment Corporation
Educational Services/Quality Assurance
12 Crosby Drive (BUO/E08)
Bedford, MA 01730

Digital Equipment Corporation • Bedford, Ma. 01730