FORTRAN-10/20 and VAX FORTRAN Compatibility Manual

Order Number: AA–Y868C–TK

February 1987

This document describes the similarities and differences between FORTRAN–10/20 and VAX FORTRAN.

This manual supersedes the *FORTRAN–10/20 and VAX FORTRAN Compatibility Manual*, order number AA–Y868B–TK.

Operating System and Version:	TOPS–10 V7.03
	TOPS–20 V4.1, V6.1
	VAX/VMS V4.5
Software Version:	FORTRAN–10 V11
	FORTRAN–20 V11
	VAX FORTRAN V4.5

digital equipment corporation
maynard, massachusetts

First Printing, August 1983
Revised, May 1985
Revised, February 1987

CONTENTS

CHAPTER 1

INTRODUCTION

This manual describes the similarities and differences between FORTRAN-1Ø/2Ø Version 11 and VAX FORTRAN Version 4.5. This manual is intended for users who may want to transport FORTRAN-1Ø/2Ø programs to a VAX computer, or transport VAX FORTRAN programs to a DECsystem-1Ø/2Ø computer. This manual describes any incompatibilities of the implementation of language features between these FORTRAN versions. It also notes any features provided by one FORTRAN version and not by the other.

Both FORTRAN-1Ø/2Ø and VAX FORTRAN are based on American National Standard FORTRAN (ANSI X3.9-1978), referred to as FORTRAN-77. Both VAX FORTRAN and FORTRAN-1Ø/2Ø are compatible with FORTRAN-77 at the full-language level. Both compilers provide many extensions and additions to the FORTRAN-77 standard. This manual describes the extensions to FORTRAN-77 for both compilers.

Both compilers support a compatibility flagger, invoked by a compiler switch, which flags features that are extensions to the FORTRAN-77 standard. The FORTRAN-1Ø/2Ø flagger also flags features that are incompatible with VAX FORTRAN.

In addition, this manual discusses the differences in software limits imposed by the two compilers, and describes how the FORTRAN data types are represented by the hardware of the DECSYSTEM-1Ø/2Ø and VAX computers.

NOTE

When 'word' is used in this manual, it refers to either a 32-bit word on VAX systems, or to a 36-bit word on FORTRAN-1Ø/2Ø systems.

See the following documents for more information on FORTRAN-1Ø/2Ø:

- _TOPS-1Ø/TOPS-2Ø FORTRAN Language Manual_ (AA-N383B-TK, AD-N383B-T1)
- _TOPS-1Ø/TOPS-2Ø FORTRAN Pocket Guide_ (AA-P529C-TK)

See the following documents for more information on VAX FORTRAN:

- _Programming in VAX FORTRAN_ (AA-DØ34D-TE)
- _VAX FORTRAN User's Guide_ (AA-DØ35D-TE)
- _VAX FORTRAN Language Summary_ (AV-M763B-TE)
- _VAX FORTRAN Installation Guide/Release Notes_

CHAPTER 2

EXTENSIONS PROVIDED BY FORTRAN-10/20 AND VAX FORTRAN

This chapter describes the extensions to the FORTRAN-77 standard provided by FORTRAN-10/20 and VAX FORTRAN.

2.1 CHARACTERS AND LINES

- Comment lines

  Both FORTRAN-10/20 and VAX FORTRAN allow comment lines to begin with characters other than C or * (asterisk) in column one. VAX FORTRAN also allows comment lines to begin with ! (exclamation point) in column one, and FORTRAN-10/20 allows comment lines to begin with !, $ (dollar sign), and / (slash) in column one.

- Comment on statements

  Both FORTRAN-10/20 and VAX FORTRAN allow any line of a program to be commented. If a ! (exclamation point) appears in the statement field of any line (except in a character or Hollerith constant), then the rest of the line is considered a comment.

EXTENSIONS PROVIDED BY FORTRAN-1Ø/2Ø AND VAX FORTRAN

- DEBUG lines

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. As an aid to debugging, a D in column one of any line causes that line to be interpreted as a comment line. However, the /INCLUDE switch to FORTRAN-1Ø/2Ø and the /D_LINES qualifier to VAX FORTRAN causes the compiler to treat the D in column one as a space, so that the remainder of the line is compiled as an ordinary (noncomment) line. This allows the program to contain statements only useful for debugging, without sacrificing object program size or execution time when not debugging.

- Long identifiers

  Supported by VAX FORTRAN and FORTRAN-1Ø/2Ø. Both allow identifiers to be up to 31 characters long.

- Multi-Statement lines

  Supported by FORTRAN-1Ø/2Ø. Not supported by VAX FORTRAN. FORTRAN-1Ø/2Ø allows users to place more than one statement per line. Multiple statements on a line are separated by semicolons. For example:

  ```
  DIST=RATE*TIME; TIME=TIME+Ø.5; CALL PRIME
  ```

- Special characters in identifiers

  Supported by VAX FORTRAN and FORTRAN-1Ø/2Ø. Identifiers may contain the special characters $ (dollar sign) and _ (underscore).

2.2 COMPILATION CONTROL STATEMENTS

- INCLUDE statement

  Supported by FORTRAN-10/20 and VAX FORTRAN. The INCLUDE statement directs the compiler to insert the contents of the designated file immediately following the INCLUDE statement. This allows programs to share common code or declarations by placing them in an INCLUDE file. (See Section 3.11 for the incompatibilities between the FORTRAN-10/20 and VAX FORTRAN INCLUDE statements.)

- OPTIONS statement

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. The OPTIONS statement allows the programmer to give the compiler directives controlling the compilation of the program.

2.3 DATA TYPES

- Alternate number format for DOUBLE PRECISION

  Both FORTRAN-10/20 and VAX FORTRAN allow the user to specify, when compiling a program unit, which floating-point number format (D-floating or G-floating) is used to store DOUBLE-PRECISION quantities. You specify which format with the /DFLOATING and /GFLOATING switches for FORTRAN-10/20, and with the /G_FLOATING and /NOG_FLOATING qualifiers for VAX FORTRAN.

D-floating has less range, but more precision, than G-floating (see Section 4.2.4). FORTRAN-10/20 supports G-floating only on KL10 model B processors. VAX FORTRAN supports G-floating on all VAX processors. However, processors without the extended floating-point feature emulate G-floating instructions in the software, and thus run slower.

- BYTE data type (BYTE is a synonym for LOGICAL*1)

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20.

- COMPLEX*16 data type (synonym for DOUBLE COMPLEX)

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. FORTRAN-10/20 acts differently if you declare COMPLEX*16 data with the DOUBLE COMPLEX statement, instead of with a COMPLEX statement with a length attribute of 16. If you use the DOUBLE COMPLEX statement, a fatal error message is issued; if you use the COMPLEX*16 statement, a warning is given and the COMPLEX*8 data is used. FORTRAN-10/20 does provide subroutines that take arguments of two-element DOUBLE PRECISION arrays, and that calculate DOUBLE COMPLEX transcendental functions.

- Hexadecimal constants (typeless)

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20.

EXTENSIONS PROVIDED BY FORTRAN-10/20 AND VAX FORTRAN

- Hollerith constants of the form 'ccc'

  Supported by FORTRAN-10/20 and VAX FORTRAN. Both FORTRAN-10/20 and VAX FORTRAN use the context of the expression to determine if a constant of the form 'ccc' is a Hollerith constant or a FORTRAN-77 string constant. If the constant is the actual argument of a subprogram, both FORTRAN-10/20 and VAX FORTRAN provide the linker with enough information to determine the type of the constant at load-time.

- Hollerith constants of the form nHccc

  Supported by FORTRAN-10/20 and VAX FORTRAN. Incompatibility: FORTRAN-10/20 right-pads all Hollerith constants with blanks until they reach a word boundary. VAX FORTRAN does not pad Hollerith constants used as actual arguments to subprograms.

- INTEGER*2 data type

  Supported by VAX FORTRAN. FORTRAN-10/20 gives a warning and uses INTEGER*4.

- LOGICAL*1 data type

  Supported by VAX FORTRAN. FORTRAN-10/20 gives a warning message and uses LOGICAL*4.

- LOGICAL*2 data type

  Supported by VAX FORTRAN. FORTRAN-10/20 gives a warning message and uses LOGICAL*4.

- MIL-Standard-1753 octal and hexadecimal constants

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. These constants are used only in DATA statements.

EXTENSIONS PROVIDED BY FORTRAN-1Ø/2Ø AND VAX FORTRAN

- Mixing CHARACTER and non-character data in COMMON or EQUIVALENCE

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. If /FLAG:ANSI is in effect, FORTRAN-1Ø/2Ø produces a warning message about this nonstandard use of COMMON or EQUIVALENCE. VAX FORTRAN produces the warning if /STANDARD is in effect.

- Octal constants (typeless)

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN, although with different syntax. Also, when assigning a single-word octal constant to a double-word variable, FORTRAN-1Ø/2Ø sets the high-order word to the constant, and sets the low-order word to zero; VAX FORTRAN sets the the high-order word to zero, and sets the low-order word to the constant.

  Conversely, if a double-word octal constant is assigned to a single-word variable, FORTRAN-1Ø/2Ø truncates the constant on the right, and VAX FORTRAN truncates the number on the left. Also, VAX FORTRAN does not allow double-word octal constants as actual arguments to subprograms.

- Octal constants of type INTEGER

  Supported by VAX FORTRAN. FORTRAN-1Ø/2Ø typeless octal constants have the same syntax and will work in most contexts as the VAX integer octal constants. The most notable exception is mixed-mode expressions, such as:

  ```
  R = "1 + 1.Ø
  ```

  VAX FORTRAN will assign R the value 2.Ø. FORTRAN-1Ø/2Ø does not convert the octal constant, and so adds a very small, unnormalized floating-point number to 1.Ø.

- RADIX-50 constants in DATA statements

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20.

- REAL*16 data type

  Supported by VAX FORTRAN. FORTRAN-10/20 gives a warning and uses REAL*8. REAL*16 is the VAX H-floating number format (see Section 4.2.4), which is available on all VAX processors. However, processors without the extended floating-point feature emulate H-floating instructions in the software, and thus run slower.

- RECORD data type

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. RECORD is a data item that can be composed of multiple fields containing aggregate and scalar data items with any mix of the VAX FORTRAN data types.

- Symbolic constant used as real or imaginary part of COMPLEX constant

  Supported by FORTRAN-10/20 and VAX FORTRAN.

2.4 SPECIFICATION AND DATA STATEMENTS

- Alternate form of the PARAMETER statement

  Both FORTRAN-10/20 and VAX FORTRAN allow PARAMETER statements of the form:

  ```
  PARAMETER p=c[,p=c...]
  ```

  where p is a symbolic name, and c is a compile-time constant expression. When this form of the PARAMETER statement is used, the type of the symbolic name is determined by the type of constant, rather than the implicit or explicit type of the name. This feature is for compatibility with earlier versions of FORTRAN-10/20 and VAX FORTRAN.

- Compatibility mode for EXTERNAL statements

  Both FORTRAN-10/20 and VAX FORTRAN allow the user to specify at compile-time that the EXTERNAL statements in a program should be processed as FORTRAN IV EXTERNAL statements. FORTRAN IV EXTERNAL statements can be used to declare a symbolic name as being the name of FORTRAN library routine, as well as the name of a user-supplied subprogram.

  Both FORTRAN-10/20 and VAX FORTRAN extend the syntax of FORTRAN IV EXTERNAL statements. They allow a user-supplied subprogram that has the same name as a FORTRAN library routine to be declared if its name is preceded by an asterisk in the EXTERNAL statement.

- DICTIONARY statement

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. This statement incorporates VAX Common Data Dictionary data definitions into the current FORTRAN source file during compilation.

EXTENSIONS PROVIDED BY FORTRAN-10/20 AND VAX FORTRAN

- Easy EQUIVALENCE of multidimensional arrays

  Supported by FORTRAN-10/20 and VAX FORTRAN. Multidimensional arrays may be equivalenced by specifying only one subscript. For example:

  ```
  DIMENSION TABLE(2,2), TRIPLE(2,2,2)
  EQUIVALENCE (TABLE(4),TRIPLE(7))
  ```

  may be used instead of:

  ```
  DIMENSION TABLE(2,2), TRIPLE(2,2,2)
  EQUIVALENCE (TABLE(2,2),TRIPLE(1,2,2))
  ```

- IMPLICIT NONE statement

  Supported by VAX FORTRAN and FORTRAN-10/20. The IMPLICIT NONE statement is used to remove all default data types, and thus requires that all identifiers be explicitly declared. Note that the program listing produced by FORTRAN-10/20 indicates which identifiers have not been explicitly declared.

- Implied DO in DATA statements of substring bounds

  Supported by FORTRAN-10/20 and VAX FORTRAN. For example:

  ```
  DATA (C(I:I), I=6,10) /5*'*'/
  ```

- Length specifiers in FUNCTION statements

  Supported by FORTRAN-10/20 and VAX FORTRAN. The length specifier can be either after the data type in the FUNCTION statement, or after the function name in the statement. For example:

  ```
  REAL*8 FUNCTION SIMPSN(A, B, F)
  REAL FUNCTION SIMPSN*8(A, B, F)
  ```

EXTENSIONS PROVIDED BY FORTRAN-1Ø/2Ø AND VAX FORTRAN

- Numeric data type length specifiers

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. You can specify the length of numeric data in type declaration statements. For example:

  ```
  REAL*8 X, Y*4, Z(1Ø)*4, A
  ```

- Numeric initialization of CHARACTER variables

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø. CHARACTER variables with a length of one may be initialized using numeric constants in DATA or type declaration statements. The character is initialized to the character whose ASCII code is specified by the numeric constant.

  Both compilers support an alternate way to initialize CHARACTER variables based on a given ASCII code. FORTRAN-1Ø/2Ø and VAX FORTRAN support the function CHAR in compile-time constant expressions. Therefore, CHAR can be used in a PARAMETER statement to form a symbolic constant, and the symbolic constant could be used in a DATA statement to initialize a CHARACTER variable. For example:

  ```
  CHARACTER*1 LINEFD, CHAR1
  PARAMETER (LINEFD=CHAR(1Ø))
  DATA CHAR1/LINEFD/
  ```

- RECORD statement

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø. The RECORD statement creates a record of the form specified in a previously declared structure.

- STRUCTURE and END STRUCTURE statements

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. These statements define the structure or form of a record. The STRUCTURE statement indicates the beginning of a structure declaration, and the END STRUCTURE statement indicates the end of a structure declaration.

- Value initialization in type declaration statements

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. Type declarations and DATA statements may be combined. For example:

  ```
  REAL X/-1./, A(10)/5*0.,5*1./
  ```

- VIRTUAL statement (synonym for the DIMENSION statement)

  Supported by VAX FORTRAN for compatibility with PDP-11 FORTRAN. Not supported by FORTRAN-10/20.

- VOLATILE statement

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. The VOLATILE statement prevents all optimization from being performed on the variables, arrays, or common blocks that it identifies.

2.5 EXPRESSIONS

- Alternate syntax for relational operators

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. FORTRAN-10/20 allows the relational operators to be specified by the following special characters:

>	for	.GT.
>=	for	.GE.
<	for	.LT.
<=	for	.LE.
==	for	.EQ.
#	for	.NE.

- Concatenation of length * operands

  Supported by FORTRAN-10/20 and VAX FORTRAN. Except in character assignment statements, the FORTRAN-77 standard forbids any character expression that involves the concatenation of an operand whose length specification is an asterisk in parentheses, unless the operand is the symbolic name of a constant. Neither FORTRAN-10/20 nor VAX FORTRAN has this restriction.

- Consecutive arithmetic operators

  Supported by FORTRAN-10/20 and VAX FORTRAN, although with different precedence rules. An example of an expression with two consecutive operators is:

  ```
  X = A ** -B * C
  ```

  This expression is not permitted by the FORTRAN-77 standard because the exponentiation operator is adjacent to the negation operator. FORTRAN-10/20 interprets the above statement as:

  ```
  X = (A ** (-B)) * C
  ```

VAX FORTRAN interprets the statement as:

```
X = A ** (- (B*C))
```

- Extended compile-time constant expressions

  VAX FORTRAN allows the following functions to be used to form compile-time constant expressions: ABS, CHAR, CMPLX, CONJG, DIM, DPROD, IAND, ICHAR, IEOR, IMAG, IOR, ISHFT, LGE, LGT, LLE, LLT, MAX, MIN, MOD, NINT, and NOT.

  FORTRAN-10/20 allows the following functions to be used to form compile-time constant expressions: CHAR, ICHAR, IAND, and IOR.

- General numeric expressions in contexts where INTEGER required

  Both FORTRAN-10/20 and VAX FORTRAN allow any type of numeric expression in the following contexts where the FORTRAN-77 expressions are standard requires an INTEGER expression:

  - The values of I/O statement specifiers
  - Subscript expressions
  - Substring bounds expressions
  - Index of computed GOTO statements
  - Index of a RETURN statement

  In addition, VAX FORTRAN allows any type of numeric expressions in the following contexts where FORTRAN-10/20 requires an INTEGER expression:

  - Implied DO loop bounds in DATA statements
  - The upper and lower limits of array declarations

Any noninteger numeric expression in these contexts is converted to INTEGER before use (any fractional part is truncated).

- LOGICAL expressions in numeric contexts

  Supported by FORTRAN-10/20 and VAX FORTRAN. However, VAX FORTRAN always treats the LOGICAL expression as an INTEGER, and FORTRAN-10/20 treats the LOGICAL expression as a typeless quantity (a bit string).

- Numeric expressions in LOGICAL contexts

  Supported by FORTRAN-10/20. Supported to some extent by VAX FORTRAN. FORTRAN-10/20 treats any numeric expression in a LOGICAL context as if it was a LOGICAL value (bit string).

  VAX allows any INTEGER expression to appear in a LOGICAL context and treats the INTEGER expression as a LOGICAL value. VAX FORTRAN only allows other types of numeric expressions to appear in one LOGICAL context: that of a numeric expression assigned to a LOGICAL variable. The numeric expression is converted to INTEGER and assigned to the LOGICAL variable as if it was a LOGICAL value.

- Numeric scalars or arrays in contexts where CHARACTER required

  For compatibility with older programs, both FORTRAN-10/20 and VAX FORTRAN allow numeric scalars and arrays that contain Hollerith data expressions are to be used in some contexts where CHARACTER expressions are required. Examples of such contexts are OPEN and CLOSE statement keywords and arguments to FORTRAN-supplied subroutines such as DATE.

- Overlap on CHARACTER assignments

  Supported by FORTRAN-10/20 and VAX FORTRAN. The FORTRAN-77 standard forbids the right hand side of a CHARACTER assignment statement from referencing any of the character positions of the CHARACTER variable or substring on the left hand side of the assignment. For example, such an overlapping assignment statement follows:

  ```
       C(2:) = C
  ```

  FORTRAN-10/20 and VAX FORTRAN allow overlap on character assignments. They always treat character assignments as if the right side of the assignment was assigned to a temporary variable, and then assigned to the variable or substring on the left of the assignment statement.

- .XOR. logical operator (Synonym for .NEQV.)

  Supported by FORTRAN-10/20 and VAX FORTRAN.

2.6 FUNCTIONS AND SUBROUTINES

- Access to addresses of storage elements

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. The %LOC built-in function returns the address of its argument. This is most useful in creating argument blocks for VAX/VMS system service routines.

- Syntax for alternate return labels

  FORTRAN-10/20 allows the characters $ (dollar sign), & (ampersand), and * (asterisk) to be used to indicate alternate return labels. VAX FORTRAN allows the characters & (ampersand) and * (asterisk).

- User controlled argument passing mechanisms

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø. By using the functions %VAL, %REF, or %DESCR, a user can control whether an argument is passed by value, by reference, or by descriptor. This allows VAX/VMS system service routines to be called directly from FORTRAN.

2.7 CONTROL STATEMENTS

- Compatibility mode for one trip DO loops

  Both FORTRAN-1Ø/2Ø and VAX FORTRAN allow the user to specify at compile-time that the DO loops in a program are one trip (FORTRAN IV style) DO loops instead of zero trip (FORTRAN-77 style) DO loops.

- DO WHILE statement

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN.

- END DO statement

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. This statement terminates a DO loop or DO WHILE loop without the need for a statement label.

- Extended range DO loops

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. Extended range DO loops make it possible to transfer out of a loop, perform a series of statements elsewhere in the program, and then transfer back into the DO loop.

- Two-branch LOGICAL IF

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. FORTRAN-10/20 allows LOGICAL IF statements to have both a true and false branch. For example:

  ```
  IF (X .GT. Y) 10,20
  ```

  will branch to 10 if X is greater than Y, otherwise it will branch to 20.

2.8 I/O STATEMENTS

- ACCEPT statement

  Supported by FORTRAN-10/20 and VAX FORTRAN.

- Apostrophe form of record specifier
  Supported by FORTRAN-10/20 and VAX FORTRAN. An apostrophe (') may be used as an alternative to the REC= specifier in I/O statement control lists.

- DECODE statement

  Supported by FORTRAN-10/20 and VAX FORTRAN. DECODE is similar to an internal file READ.

  In FORTRAN-10/20, the character count specified in DECODE statements is interpreted as an internal record size in characters; and a / (slash) format descriptor or indefinite repeat causes the record pointer to advance to the next internal record.

  In VAX FORTRAN, the character count is interpreted as a total string count; and a / (slash) format descriptor or indefinite repeat, which for normal files causes a new record to be read or written, is illegal.

  DECODE is considered obsolete.

- DEFINE FILE statement

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. The DEFINE FILE statement is similar to the OPEN statement. DEFINE FILE is considered obsolete.

- Default unit in WRITE statements

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. FORTRAN-10/20 allows WRITE statements of the form:

  ```
      WRITE f[,iolst]
  ```

  where f is an asterisk, format specifier, or namelist specifier. This provides symmetry with the FORTRAN-77 form of the READ statement with a default unit.

- DELETE statement

  Supported by VAX FORTRAN and FORTRAN-10/20. The DELETE statement removes a record from a relative (direct-access) or indexed file.

- ENCODE statement

  Supported by FORTRAN-10/20 and VAX FORTRAN. ENCODE is similar to an internal file WRITE.

  In FORTRAN-10/20, the character count in ENCODE statements is interpreted as an internal record size in characters; a / (slash) format descriptor or indefinite repeat causes the record pointer to advance to the next internal record.

  In VAX FORTRAN, the character count is interpreted as a total string count. A / (slash) format descriptor or indefinite repeat, which for normal files causes a new record to be read or written, is illegal.

  ENCODE is considered obsolete.

- FIND statement

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. The FIND statement can be used to pre-position random access relative files to allow greater overlap of computation and I/O.

- Indexed I/O

  Supported by VAX FORTRAN and FORTRAN-1Ø/2Ø. Indexed I/O allows records in files to be selected for I/O using keys contained in the records.

- I/O in I/O

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø. VAX FORTRAN allows a user function to do I/O even if the function was called from an I/O statement. For example, the statement:

  ```
  WRITE(5,'(1X,F1Ø.4)') F(X)
  ```

  calls the function F. VAX FORTRAN allows the function F to do I/O to any unit except the unit referenced by the I/O statement that called the function. FORTRAN-1Ø/2Ø will not allow a function called by an I/O statement to do I/O to any unit. The FORTRAN-77 standard forbids I/O statements from referencing any function if such a reference would cause another I/O statement to be executed.

- List-directed internal file I/O

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø.

- NAMELIST statement

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. However, FORTRAN-1Ø/2Ø uses either FMT= or NML= as the optional specifier used to specify the NAMELIST name in I/O statements. VAX FORTRAN uses only NML= as the optional specifier.

- Prompting during NAMELIST input

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. VAX FORTRAN allows interactive display of the NAMELIST group and values by outputting the current group name or the name of the variables in response to a question mark (?) or an equal sign and question mark (=?).

- PUNCH statement

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN.

- REREAD statement

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. The REREAD statement allows the last record read by a formatted READ or ACCEPT statement to be accessed again and reprocessed. This statement can be used to recover from certain types of errors. For example, the user may be able to recover from an "illegal character in data" error by rereading the same record using a different format.

- REWRITE statement

  Supported by VAX FORTRAN and FORTRAN-10/20. The REWRITE statement is used to update the record last read in an indexed file.

- TYPE statement

  Supported by FORTRAN-10/20 and VAX FORTRAN.

- UNLOCK statement

  Supported by VAX FORTRAN and FORTRAN-10/20. The UNLOCK statement unlocks a record in a relative or indexed file locked by a previous READ statement.

2.9 FORMATTING

- $ format edit descriptor

  Supported by FORTRAN-10/20 and VAX FORTRAN. The $ (dollar sign) edit descriptor suppresses the output of a carriage return to end the current line during output operations.

- Comma as an external field separator during formatted input

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. VAX FORTRAN allows a comma to be used to separate numeric data during formatted data input. For example, if the following statements:

  ```
            READ(5,10) I, J, A, B
        10  FORMAT(2I6,2F10.2)
  ```

  read the record:

  ```
        1,-2,1.0,35
  ```

  then I=1, J=-2, A=1.0, B=0.35

- Default widths for format edit descriptors

  Supported by FORTRAN-10/20 and VAX FORTRAN. For FORTRAN-10/20 output and VAX FORTRAN input or output, if the format edit descriptors I, O, Z, L, F, E, D, G, A, or R (FORTRAN-10/20 only) are used without specifying field width values, default values are supplied based on the data type of the I/O list element. FORTRAN-10/20 and VAX differ on the default field widths (see Section 3.10.4).

  For FORTRAN-10/20 input, the data is scanned until a blank, comma, or character illegal for the specified edit descriptor is encountered (except for A format).

EXTENSIONS PROVIDED BY FORTRAN-1Ø/2Ø AND VAX FORTRAN

- CHARACTER I/O using the G format edit descriptor

  Supported by FORTRAN-1Ø/2Ø. Not supported by VAX FORTRAN.

- Numeric array elements to store formats

  Supported by VAX FORTRAN. Not supported by FORTRAN-1Ø/2Ø.

- Numeric arrays to store formats

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. Numeric arrays can contain Hollerith constants, which can be used as formats.

- O (octal) format edit descriptor

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN.

- Optional commas in formats

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. Commas are not needed to separate format field descriptors if there is no ambiguity in how the format should be interpreted. For example, the following format is legal:

  ```
  1Ø    FORMAT(1X'I is 'I5)
  ```

- Q format edit descriptor

  Supported by FORTRAN-1Ø/2Ø and VAX FORTRAN. The Q edit descriptor returns the number of characters left in the current record (valid only during input operations).

- R format edit descriptor

  Supported by FORTRAN-10/20. Not supported by VAX FORTRAN. The R edit descriptor is used to edit Hollerith data, and thus is similar to the A field descriptor. The difference between the R and A field descriptors is that the A descriptor works with left-justified Hollerith data, and the R descriptor works with right-justified Hollerith data with leading nulls. The R descriptor allows Hollerith characters to be stored one per word, and thus manipulated easily.

- Variable format expressions

  Supported by VAX FORTRAN. Not supported by FORTRAN-10/20. VAX FORTRAN allows formats to contain an expression inclosed in angle brackets anywhere an integer constant is needed (except as the length of a Hollerith constant). The expression in angle brackets is evaluated at runtime and used in the format. For example, if J is 3, the format:

  ```
  10    FORMAT(I<J+1>)
  ```

  will act as if I4 was specified.

- Z (hexadecimal) format edit descriptor

  Supported by FORTRAN-10/20 and VAX FORTRAN.

CHAPTER 3

FORTRAN-10/20 AND VAX FORTRAN INCOMPATIBILITIES

3.1 PROGRAM FORMAT

FORTRAN-10/20 and VAX FORTRAN have several minor differences in the format of programs.

The tab character is treated differently by the two compilers when it appears in the statement field of a line. For the purpose of determining where the remark field begins, FORTRAN-10/20 counts the tab as being the number of spaces it appears to be in the listing. VAX FORTRAN counts the tab as being one character. Thus, part of a statement in the statement field as recognized by VAX FORTRAN may be in the remark field as recognized by FORTRAN-10/20.

VAX FORTRAN has a /EXTEND_SOURCE qualifier, which specifies that the compiler is to extend the range of FORTRAN source text from columns 1 through 72 to columns 1 through 132. FORTRAN-10/20 does not have a similar qualifier, and always limits statements to columns 1 through 72.

FORTRAN-1Ø/2Ø and VAX FORTRAN treat the characters formfeed (control/L) and vertical tab (control/K) differently when they appear in source programs. FORTRAN-1Ø/2Ø uses these characters (along with linefeed) to determine where the end of line is. VAX FORTRAN does not use these characters for this purpose (as it uses RMS to determine where the end of line is). VAX FORTRAN treats formfeeds as spaces when they appear outside of CHARACTER and Hollerith constants, and it gives an error message when it encounters vertical tabs outside one of those constants. Thus, a source program that is formatted with formfeeds should only have the formfeeds following lines that have already been terminated by the normal method of the user typing a return.

Both FORTRAN-1Ø/2Ø and VAX FORTRAN allow nonprintable characters to be included in the source program (this is only useful in CHARACTER or Hollerith constants). FORTRAN-1Ø/2Ø does not allow the characters null, linefeed, vertical tab, formfeed, or carriage return to be entered directly into CHARACTER constants (TOPS-1Ø and TOPS-2Ø have system-wide conventions about the treatment of these characters).

3.2 ARGUMENT PASSING

FORTRAN-1Ø/2Ø and VAX FORTRAN use different mechanisms to pass arguments to subprograms. FORTRAN-1Ø/2Ø passes scalar numeric arguments by value-result, and VAX FORTRAN passes scalar numeric arguments by reference.

When arguments are passed by value-result, the subprogram is passed the values of its actual arguments. These values are then stored in local copies of the arguments, which the subprogram then processes. Before the subprogram returns, the values of the local copies of the arguments are copied back into the actual arguments in the calling program.

When arguments are passed by reference, the subprogram is passed the addresses of its actual arguments. Whenever the subprogram processes one of the arguments, it uses the address of the argument to process the argument directly. Since the arguments are acted upon directly, local copies of the arguments do not exist.

These different mechanisms for passing arguments usually do not concern the FORTRAN programmer, since there are only three situations where call by value-result differs from call by reference. The three situations are the following:

- When multiple copies of the same variable are passed to a subprogram
- When a variable is passed to a subprogram and the subprogram also references the variable by COMMON
- When one entry point of a subprogram with multiple entry points refers to an argument passed to a previous entry point, and that argument was modified between the two calls.

All three of these cases are prohibited by the FORTRAN-77 standard.

The following program illustrates the different results that can be obtained when a subprogram accesses a variable both as an argument and by COMMON:

```
      COMMON I
      I = 1
1Ø    WRITE(5,1ØØØ) 1Ø, I
      CALL SUB(I)
2Ø    WRITE(5,1ØØØ) 2Ø, I
1ØØØ  FORMAT(' Line', I3, ' of Main Program, I=', I1)
      END
      SUBROUTINE SUB(I)
      COMMON J
1Ø    WRITE(5,1ØØØ) 1Ø, I, J
      I = 2
2Ø    WRITE(5,1ØØØ) 2Ø, I, J
      J = 3
3Ø    WRITE(5,1ØØØ) 3Ø, I, J
1ØØØ  FORMAT(' Line', I3, ' of SUB, I=', I1, ' and J=', I1)
      END
```

If this program is compiled by FORTRAN-10/20, it prints:

```
     Line 10 of Main Program, I=1
     Line 10 of SUB, I=1 and J=1
     Line 20 of SUB, I=2 and J=1
     Line 30 of SUB, I=2 and J=3
     Line 20 of Main Program, I=2
```

If this program is compiled by VAX FORTRAN, it prints:

```
     Line 10 of Main Program, I=1
     Line 10 of SUB, I=1 and J=1
     Line 20 of SUB, I=2 and J=2
     Line 30 of SUB, I=3 and J=3
     Line 20 of Main Program, I=3
```

The difference in the program output is because the FORTRAN-10/20 program modifies a local copy of its argument, while the VAX FORTRAN program modifies its argument directly.

3.3 LOGICAL TESTS

The logical constants .TRUE. and .FALSE. are defined, respectively, as all ones and all zeros by both FORTRAN-10/20 and VAX FORTRAN. However, the test for true and false differs.

FORTRAN-10/20 tests the left-most bit (the sign bit) of a LOGICAL value to determine if the value is true or false.

VAX FORTRAN tests the right-most bit (the low-order bit) of a LOGICAL value to determine if the value is true or false.

In most cases, this difference is not apparent to the FORTRAN programmer. It is only significant in the following cases:

- A program EQUIVALENCEs a LOGICAL variable and a numeric variable.

- A program performs numeric operations on LOGICAL values.
- A program assigns octal or numeric constants to a LOGICAL variable, or uses octal or numeric values as LOGICAL values.

The following program illustrates this incompatibility:

```
LOGICAL L
L = 1
IF (L) THEN
     WRITE(5,'(1X,A)') 'L is true.'
ELSE
     WRITE(5,'(1X,A)') 'L is false.'
END IF
END
```

If this program is compiled by FORTRAN-1Ø/2Ø, it prints:

```
L is false.
```

If this program is compiled by VAX FORTRAN, it prints:

```
L is true.
```

3.4 THE ASSIGNED GOTO STATEMENT

VAX FORTRAN ignores the optional list of labels in an assigned GOTO statement. FORTRAN-1Ø/2Ø checks at runtime to make sure that the assigned variable matches one of the labels in the list, and proceeds to the next statement if there is no match.

3.5 I/O INCOMPATIBILITIES

There are several incompatibilities between FORTRAN-1Ø/2Ø and VAX FORTRAN for input/output capabilities. The following sections describe these differences.

3.5.1 Default Filenames Associated with Units

The unit number assignments differ between FORTRAN-10/20 and VAX FORTRAN. The default unit number assignments are important because programs that do not open their files explicitly using the OPEN statement get a file with a default name on the default device associated with that unit number.

FORTRAN-10/20 associates a different device with each unit number. For example, unit 1 is disk, unit 2 is the card reader, unit 3 is line printer, unit 5 is the terminal, and so on. The filename that FORTRAN-10/20 associates with each unit is of the form "FORxx.DAT", where xx is the two digit unit number with leading zeros.

VAX FORTRAN associates disk with all units except 5 and 6. VAX FORTRAN associates unit 5 with the terminal for input, and unit 6 with the terminal for output. The default filename that VAX FORTRAN associates with each unit is "FORxxx.DAT", where xxx is the three digit unit number with leading zeros. The default unit device assignments affect how the OPEN statement works as well. When a NAME= (or FILE=) specifier does not specify a device name, the default device associated with the unit is used.

3.5.2 Unit 5 for Both Input and Output

On the VAX, unit 5 may be used for both input and output to the terminal for interactive jobs; however, under batch unit 5 must be used for input and unit 6 for output. On the DECSYSTEM-10/20, unit 5 may always be used for both input and output.

3.5.3 STATUS='NEW' and File Generation (Version) Numbers

If a VAX FORTRAN or FORTRAN-20 program opens a file with STATUS='NEW' and no "version" (VAX/VMS) or "generation" (TOPS-20) number is specified, a new generation of the file will be created. If a FORTRAN-10 program attempts to open a file with STATUS='NEW' and that file exists, an error message is given. Note that the generation number does not apply to TOPS-10.

3.5.4 STATUS='UNKNOWN' and File Generation (Version) Numbers

If a file is opened for output with STATUS='UNKNOWN' and no "version" (VAX/VMS) number is specified by a VAX FORTRAN program, the highest existing version number is written. If a FORTRAN-20 program attempts to open a file for output with STATUS='UNKNOWN', and does not specify a "generation" (TOPS-20) number, then a new generation of the file is always written.

3.5.5 The CARRIAGECONTROL= Specifier

The meanings of CARRIAGECONTROL='FORTRAN' and CARRIAGECONTROL='LIST' on a VAX and a DECSYSTEM-10/20 are similar, but there are differences which the user should be aware of.

There is a slight difference between DECSYSTEM-10/20 and VAX with regard to CARRIAGECONTROL= 'FORTRAN'. On the VAX, a file whose attribute is CARRIAGECONTROL= 'FORTRAN' will have the first character of each record translated to one or more vertical motion characters when the file is either printed or typed. On the DECSYSTEM-10/20, this action occurs only when the file is printed.

In addition, FORTRAN-10/20 supports CARRIAGECONTROL= 'TRANSLATED', which forces translation of the first character of each record as it is output to the specified file.

There is an interaction of the PRINT command under the TOPS-10 and TOPS-20 operating systems and files with the extension of "DAT" that is of interest to users of FORTRAN. When one of these two operating systems prints a file with the extension of "DAT", it strips off the first character of every line and uses it to control the vertical spacing of the printer using FORTRAN carriage-control rules.

3.5.6 The RECL= Specifier and Formatted Sequential Files

The RECL= specifier has a meaning for formatted sequential access files on the VAX that it does not have on the DECSYSTEM-10/20.

On the VAX, formatted sequential access files have by default variable-length records (RECORDTYPE='VARIABLE'). For this type of file organization, the RECL= specifier gives the maximum length of any record. If the RECL= specifier is not specified for a new formatted sequential access file, a default of RECL=133 is assumed. If an attempt is made to write a record greater than the RECL value, an error occurs. Thus, if a program writes a 200 character record to a new, formatted sequential access file without giving a RECL= specifier in a OPEN statement, it will get a run-time error on a VAX processor. The program will not have any problems on a DECSYSTEM-10/20 processor.

On the DECSYSTEM-10/20, RECL= specifies that the file has fixed-length records. If no RECL= specifier is given, the records are assumed to be variable length, and the maximum record size is 655360 characters.

3.5.7 ASSOCIATEVARIABLE= Specifier

The execution of an OPEN statement with an ASSOCIATEVARIABLE= specifier in FORTRAN-1Ø/2Ø has the effect of setting the associate variable to 1. The execution of such an OPEN statement in VAX FORTRAN does not affect the value of the associate variable at all.

3.5.8 ACCESS='APPEND' and REWIND

VAX FORTRAN allows a program to REWIND a file that has been opened ACCESS='APPEND'. FORTRAN-1Ø/2Ø allows this feature on TOPS-1Ø, but not on TOPS-2Ø.

3.6 INQUIRE STATEMENT

FORTRAN-1Ø/2Ø and VAX FORTRAN differ on the action taken when an error occurs in an INQUIRE. FORTRAN-1Ø/2Ø takes an error branch if file the being inquired about is an illegal file specification, or if the logical unit number being inquired about is out of range. VAX FORTRAN reports that the file or unit does not exist.

The following INQUIRE statement specifiers are supported by VAX FORTRAN only:

1. RECORDTYPE='SEGMENTED','STREAM_CR','STREAM_LF'

The BYTESIZE specifier is supported by FORTRAN-1Ø/2Ø only.

3.7 THE OPEN STATEMENT

This section lists the OPEN statement specifiers in several categories.

- Section 3.7.1 lists the specifiers allowed by the FORTRAN-77 standard.

- Section 3.7.2 lists the specifiers supported by both FORTRAN-10/20 and VAX FORTRAN.
- Section 3.7.3 lists the specifiers supported only by FORTRAN-10/20.
- Section 3.7.4 lists the specifiers supported only by VAX FORTRAN.
- Section 3.7.5 lists suggestions on converting OPEN statements so that they can be used with either FORTRAN-10/20 or VAX FORTRAN.

3.7.1 Standard OPEN Specifiers

The following OPEN statement specifiers are allowed by the FORTRAN-77 standard:

1. ACCESS='DIRECT' and ACCESS='SEQUENTIAL'
2. BLANK='ZERO' and BLANK='NULL'
3. ERR=
4. FILE=
5. FORM='FORMATTED' and FORM='UNFORMATTED'
6. IOSTAT=
7. RECL=
8. STATUS='NEW', STATUS='OLD', STATUS='SCRATCH', STATUS='UNKNOWN'
9. UNIT=

3.7.2 OPEN Specifiers Common to FORTRAN-10/20 and VAX FORTRAN

The following OPEN statement specifiers are supported by both FORTRAN-10/20 and VAX FORTRAN in addition to the specifiers in Section 3.7.1:

1. ACCESS='APPEND' and ACCESS='KEYED'

2. ASSOCIATEVARIABLE=

3. BLOCKSIZE=

4. BUFFERCOUNT=

5. CARRIAGECONTROL='FORTRAN', CARRIAGECONTROL='LIST', and CARRIAGECONTROL='NONE'

6. DEFAULTFILE=

7. DISPOSE='DELETE', DISPOSE='KEEP', DISPOSE='PRINT', and DISPOSE='SAVE'

8. KEY=

9. MAXREC=

10. NAME=

11. NOSPANBLOCKS

12. ORGANIZATION=

13. READONLY

14. RECORDSIZE=

15. SHARED

16. TYPE='NEW', TYPE='OLD', TYPE='SCRATCH', TYPE='UNKNOWN'

17. RECORDTYPE='FIXED', RECORDTYPE='STREAM', and RECORDTYPE='VARIABLE'

18. USEROPEN=

3.7.3 OPEN Specifiers Only in FORTRAN-10/20

The following OPEN statement specifiers are supported only by FORTRAN-10/20:

1. ACCESS='SEQIN', ACCESS='SEQOUT', ACCESS='SEQINOUT', ACCESS='RANDOM' and ACCESS='RANDIN'
2. BYTESIZE=
3. CARRIAGECONTROL='DEVICE', CARRIAGECONTROL='TRANSLATED'
4. DENSITY=
5. DEVICE=
6. DIALOG and DIALOG=
7. DIRECTORY=
8. DISPOSE='EXPUNGE', DISPOSE='LIST', DISPOSE='PUNCH'
9. FILESIZE=
10. INITIALIZE=
11. MODE=
12. ORGANIZATION='UNKNOWN'
13. PADCHAR=
14. PARITY=
15. PROTECTION=
16. RECORDTYPE='UNKNOWN'

17. STATUS='DELETE', STATUS='EXPUNGE', STATUS='KEEP'

18. TAPEFORMAT=

19. TYPE='DELETE', TYPE='EXPUNGE', TYPE='KEEP'

20. VERSION=

FORTRAN-10/20 checks only the first six characters of OPEN specifiers for correct spelling. Therefore, it accepts a six letter abbreviation of any of its specifiers or any misspelling of its keywords if the first six characters are correct. Future versions of FORTRAN-10/20 may not allow this, so it should not be relied upon.

3.7.4 OPEN Specifiers Only in VAX FORTRAN

The following OPEN statement specifiers are supported only by VAX FORTRAN:

1. DISP=

2. DISPOSE='PRINT/DELETE', DISPOSE='SUBMIT', and DISPOSE='SUBMIT/DELETE'

3. EXTENDSIZE=

4. INITIALSIZE=

5. RECORDTYPE='SEGMENTED', RECORDTYPE='STREAM_CR', RECORDTYPE='STREAM_LF'

3.7.5 Conversion of OPEN Statements

Many OPEN statement specifiers that are unique to FORTRAN-10/20 or VAX FORTRAN have synonyms that are acceptable to either compiler. This section gives suggestions on converting OPEN statements so that they can be used with either FORTRAN-10/20 or VAX FORTRAN.

This section also lists which specifiers unique to the two compilers are synonyms. However, these specifiers cannot be put in a form that is acceptable to both compilers.

The VAX FORTRAN only specifier DISP= can be made acceptable to either compiler (provided its value is supported by both compilers) if it is converted to DISPOSE=.

Several FORTRAN-10/20 values of the ACCESS= specifier can be rewritten as a series of other specifiers, which are acceptable to either compiler:

```
ACCESS='SEQIN' can be written as
      ACCESS='SEQUENTIAL',STATUS='OLD',READONLY

ACCESS='RANDIN' can be written as
      ACCESS='DIRECT',STATUS='OLD',READONLY

ACCESS='SEQOUT' can be written as
      ACCESS='SEQUENTIAL',STATUS='UNKNOWN'

ACCESS='SEQINOUT' can be written as
      ACCESS='SEQUENTIAL',STATUS='UNKNOWN'

ACCESS='RANDOM' can be written as
      ACCESS='DIRECT',STATUS='UNKNOWN'
```

NOTE

> The OPEN specifier STATUS='UNKNOWN' has different effects on the generation or version numbers for FORTRAN-20 and VAX FORTRAN (see Section 3.5.4).

The FORTRAN-10/20 specifier DISPOSE='LIST' is a synonym to the VAX FORTRAN DISPOSE='PRINT/DELETE'. However, these specifiers cannot be put in a form that is acceptable to both compilers.

The FORTRAN-1Ø/2Ø INITIALIZE= specifier is a synonym for the VAX FORTRAN INITIALSIZE= specifier. FORTRAN-1Ø/2Ø will recognize the INITIALSIZE specifier because FORTRAN-1Ø/2Ø checks only the first six characters of OPEN specifiers for correct spelling.

3.8 THE CLOSE STATEMENT

This section lists the CLOSE statement specifiers in several categories.

- Section 3.8.1 lists the specifiers allowed by the FORTRAN-77 standard.
- Section 3.8.2 lists the specifiers supported by both FORTRAN-1Ø/2Ø and VAX FORTRAN.
- Section 3.8.3 lists the specifiers supported only by FORTRAN-1Ø/2Ø.
- Section 3.8.4 lists the specifiers supported only by VAX FORTRAN.
- Section 3.8.5 lists suggestions on converting CLOSE statements so that they can be used with either FORTRAN-1Ø/2Ø or VAX FORTRAN.

3.8.1 Standard CLOSE Specifiers

The following CLOSE statement specifiers are allowed by the FORTRAN-77 standard:

1. ERR=
2. IOSTAT=
3. STATUS='KEEP' and STATUS='DELETE'
4. UNIT=

3.8.2 CLOSE Specifiers Common to FORTRAN-1Ø/2Ø and VAX FORTRAN

The following CLOSE statement specifier is supported by both FORTRAN-1Ø/2Ø and VAX FORTRAN in addition to the specifiers in Section 3.8.1:

1. DISPOSE='PRINT' and DISPOSE='SAVE'

3.8.3 CLOSE Specifiers Only in FORTRAN-1Ø/2Ø

The following CLOSE statement specifiers are supported only by FORTRAN-1Ø/2Ø:

1. DEVICE=
2. DIALOG and DIALOG=
3. DIRECTORY=
4. DISPOSE='EXPUNGE', DISPOSE='LIST', DISPOSE='PUNCH', and DISPOSE='RENAME'
5. FILE=
6. NAME=
7. PROTECTION=
8. STATUS='EXPUNGE'
9. TYPE=

FORTRAN-1Ø/2Ø only checks the first six characters of CLOSE specifiers for correct spelling. Therefore, it accepts any six letter abbreviation of any of its specifiers or any misspelling of its keywords if the first six characters are correct. Future versions of FORTRAN-1Ø/2Ø may not allow this, so it should not be relied upon.

3.8.4 CLOSE Specifiers Only in VAX FORTRAN

The following CLOSE statement specifiers are supported only by VAX FORTRAN:

1. DISP=

2. DISPOSE='PRINT/DELETE', DISPOSE='SUBMIT', and DISPOSE='SUBMIT/DELETE'

3. STATUS='PRINT', STATUS='PRINT/DELETE', STATUS='SAVE', STATUS='SUBMIT', and STATUS='SUBMIT/DELETE'

3.8.5 Conversion of CLOSE Statements

As is the case with OPEN statement specifiers, many CLOSE statement specifiers that are unique to FORTRAN-10/20 and VAX FORTRAN have synonyms, which are acceptable to either compiler. This section gives suggestions on converting CLOSE specifiers into a form that is acceptable to either FORTRAN-10/20 or VAX FORTRAN.

This section also lists which specifiers unique to the two compilers are synonyms. However, these specifiers cannot be put in a form that is acceptable to both compilers.

The VAX FORTRAN only specifier DISP= can be made acceptable to either compiler (provided its value is supported by both compilers) if it is converted to DISPOSE=.

The VAX FORTRAN only specifier STATUS='PRINT' is acceptable to both compilers if it is converted to DISPOSE='PRINT'.

The VAX FORTRAN only specifier STATUS='SAVE' is acceptable to both compilers if it is converted to STATUS='KEEP'.

FORTRAN-1Ø/2Ø AND VAX FORTRAN INCOMPATIBILITIES

The FORTRAN-1Ø/2Ø specifier DISPOSE='LIST' is a synonym for the VAX FORTRAN specifier STATUS='PRINT/DELETE'. However, the specifiers cannot be put in a form that is acceptable to both compilers.

3.9 CARRIAGE CONTROL

FORTRAN-1Ø/2Ø and VAX FORTRAN have the following carriage-control characters in common:

'+'	Overprinting
' '	Single Spacing
'Ø'	Double Spacing
'1'	Skip to next top of page
'$'	Single space, leaving cursor at end of line

FORTRAN-1Ø/2Ø has the following carriage-control characters not supported by VAX FORTRAN:

'2'	Skip to next half of page
'3'	Skip to next third of page
'-'	Triple spacing
'*'	Single space; suppress automatic formfeed
'.'	Triple space with automatic formfeed after every 2Ø lines
','	Double space with automatic formfeed after every 3Ø lines
'/'	Space to next sixth of page

VAX FORTRAN has the following carriage-control character not supported by FORTRAN-1Ø/2Ø:

null	Overprint, leaving cursor at end of line

3.1Ø FORMAT EDIT DESCRIPTORS

3.1Ø.1 Interaction of the X and $ Edit Descriptors

Normally, the X format edit descriptor cannot be used to write blanks at the end of a line. Thus, the following program:

```
           WRITE(5,1Ø)
     1Ø    FORMAT(' Hello', 1ØX)
           END
```

will not write ten blanks after the word "Hello".

FORTRAN-1Ø/2Ø has a single exception to this rule. If the line was written with $ (dollar sign) carriage control, or the $ edit descriptor was used to end the line, then the X edit descriptor can be used to write blanks at the end of the line. VAX FORTRAN does not have this exception.

Both FORTRAN-1Ø/2Ø and VAX FORTRAN provide a way to write blanks at the end of a line, regardless of the exception for $ carriage control or the $ edit descriptor. If blanks are to be written at the end of a line, a literal containing blanks should be used. For example, the above format could be rewritten as:

```
     1Ø    FORMAT(' Hello', 1Ø(' '))
```

3.1Ø.2 Interaction of the $ Edit Descriptor and Carriage-Control Characters

Both FORTRAN-1Ø/2Ø and VAX FORTRAN allow the use of the $ (dollar sign) edit descriptor on output to modify the carriage control specified by the first character of the record.

For FORTRAN-1Ø/2Ø, the $ edit descriptor suppresses all carriage control at the end of the current record (for CARRIAGECONTROL='LIST') or at the beginning of the next record (for CARRIAGECONTROL='FORTRAN' or 'TRANSLATED').

For VAX FORTRAN, the $ edit descriptor is ignored if the first charcter of the record is "Ø" or "1". If the first character is a plus sign (+), the $ descriptor causes the output to begin at the end of the previous line and leaves the print position at the end of the line.

3.1Ø.3 F, E, D, and G Edit Descriptors

FORTRAN-1Ø/2Ø allows an alternate form of the E, F, D, and G edit descriptors:

Fw	for	Fw.Ø
Ew	for	Ew.Ø
Dw	for	Dw.Ø
Gw	for	Gw.Ø

VAX FORTRAN does not support this alternate form of the descriptors.

FORTRAN-1Ø/2Ø allows the use of the G edit descriptor with character data. VAX FORTRAN does not support this use.

3.1Ø.4 O and Z Edit Descriptors

The forms of the O and Z edit descriptors are:

```
O[w[.m]]
Z[w[.m]]
```

Where:

w	is the length of the field.
m	is the number of digits guaranteed to be output to the field.

If m is not specified for the O and Z edit descriptors, FORTRAN-10/20 assumes a default value equal to w, while VAX FORTRAN assumes a default value of 1. Thus, FORTRAN-10/20 prints leading zeros for the O and Z edit descriptors, and VAX prints leading spaces.

FORTRAN-10/20 allows octal and hexadecimal numbers to have an optional sign when input. A plus sign has no effect, and a minus sign causes the negative (two's complement) of the number to be input. VAX FORTRAN does not allow octal and hexadecimal numbers to have a sign.

3.10.5 Default Field Widths

Both FORTRAN-10/20 and VAX FORTRAN allow format edit descriptors to be used without specifying field width values. If an edit descriptor is used without specifying a field width, then a default width is supplied based on the data type of the I/O list element (see Table 3-1).

Note that for the purposes of assigning the default field widths, the FORTRAN data types fall into four distinct classes:

Half-Word Data Types: LOGICAL*2 (VAX FORTRAN), INTEGER*2 (VAX FORTRAN)

Single-Word Data Types: COMPLEX*8, INTEGER*4, LOGICAL*4, REAL*4

Double-Word Data Types: COMPLEX*16 (VAX FORTRAN), REAL*8

Quad-Word Data Type: REAL*16 (VAX FORTRAN)

The CHARACTER Data type

Table 3-1: Default Field Widths for Edit Descriptors

Descriptor	Data Type	10/20	VAX
I	Half-Word	I15	I7 (*)
I	Single-Word	I15	I12
F	Single-Word	F15.7	F15.7 (&)
F	Double-Word	F25.18	F25.16 (&)
F	Quad-Word		F42.33 (#)
E	Single-Word	E15.7E2	E15.7E2
E	Double-Word	E25.18E2	E25.16E2
E	Quad-Word		E42.33E3 (#)
D	Single-Word	E15.7E2	E15.7E2
D	Double-Word	E25.18E2	E25.16E2
D	Quad-Word		E42.33E3 (#)
G	Single-Word	G15.7E2	G15.7E2 (%)
G	Double-Word	G25.18E2	G25.16E2
G	Quad-Word		G42.33E3 (#)
G	Character*n	Gn	(%)
O	Single-Word	O15	O12
O	Double-Word	O25	O23
L	Single-Word	L15	L2
Z	Single-Word	Z15	Z12
Z	Double-Word	Z25	Z23
A	Single-Word	A5	A4
A	Double-Word	A10	A8
A	Character*n	An	An

Notes:

* FORTRAN-10/20 treats INTEGER*2 and LOGICAL*1 variables as if they were INTEGER*4 and LOGICAL*4 variables, respectively.

& FORTRAN-10/20 expands the default field width for the F edit descriptor if the default field width is too small for the data.

% VAX FORTRAN does not allow INTEGER or CHARACTER I/O with the G edit descriptor.

FORTRAN-10/20 does not support REAL*16.

3.11 INCLUDE STATEMENT

The FORTRAN-1Ø/2Ø and VAX FORTRAN INCLUDE statements have the following incompatibilities:

- The VAX FORTRAN INCLUDE statement does not have a /NOCREF switch.

- FORTRAN-1Ø/2Ø lists the text from the INCLUDEd file by default. VAX FORTRAN requires that the /LIST switch be specified to list the included file, or that the /SHOW=INCLUDE qualifier be given when the program unit is compiled.

- FORTRAN-1Ø/2Ø does not support the inclusion of text from a text library. VAX FORTRAN allows a name in parentheses to follow the name of the file in the INCLUDE statement. The name in parentheses designates a module (section) of a text library file that is to be included.

3.12 PROGRAM STATEMENT

Both FORTRAN-1Ø/2Ø and VAX FORTRAN prohibit the name of the main program, as given by the PROGRAM statement, from being the same as the name of any subprogram, entry point, or COMMON block in the executable program. VAX FORTRAN further restricts the program name from being the same as the name of any variable, PARAMETER, or NAMELIST in the main program.

3.13 FORTRAN-SUPPLIED SUBPROGRAMS

This section describes the differences in the FORTRAN-supplied subprograms between FORTRAN-1Ø/2Ø and VAX FORTRAN.

3.13.1 Functions That Are Generic Only in FORTRAN-1Ø/2Ø

The following functions are generic in FORTRAN-1Ø/2Ø, but do not have the generic property in VAX FORTRAN:

1. ALOG - synonym for the standard generic function LOG

2. ALOG1Ø - synonym for the standard generic function LOG1Ø

3. AMAX1 - synonym for the standard generic function MAX

4. AMIN1 - synonym for the standard generic function MIN

3.13.2 Functions That Are Generic Only in VAX FORTRAN

The following functions are generic in VAX FORTRAN, but do not have the generic property in FORTRAN-1Ø/2Ø:

1. ACOSD - arc cosine with angle measured in degrees

2. ASIND - arc sine with angle measured in degrees

3. ATAN2D - arc tangent of quotient with angle measured in degrees

4. ATAND - arc tangent with angle measured in degrees

5. COSD - cosine of angle in degrees

6. SIND - sine of angle in degrees

7. TAND - tangent of angle in degrees

3.13.3 Generic Functions Only Available in VAX FORTRAN

VAX FORTRAN has the following generic routines not available in FORTRAN-10/20:

1. DCMPLX - conversion to DOUBLE COMPLEX
2. QEXT - conversion to REAL*16

Note that most of the VAX FORTRAN generic functions can result in calls to intrinsic functions to process INTEGER*2, DOUBLE COMPLEX, or REAL*16 data. The corresponding FORTRAN-10/20 generic functions can never do this because FORTRAN-10/20 lacks those data types.

3.13.4 Intrinsic Functions Only Available in FORTRAN-10/20

FORTRAN-10/20 has the following intrinsic functions not available in VAX FORTRAN:

1. COTAN - REAL cotangent of an angle
2. DCOTAN - DOUBLE PRECISION cotangent of an angle

3.13.5 Intrinsic Functions Only Available in VAX FORTRAN

VAX FORTRAN has the following intrinsic functions not available in FORTRAN-10/20:

1. ACOSD, DACOSD, QACOSD - The REAL*4, REAL*8, REAL*16 arc cosine with angle measured in degrees
2. ASIND, DASIND, QASIND - The REAL*4, REAL*8, REAL*16 arc sine with angle measured in degrees
3. ATAN2D, DATAN2D, QATAN2D - The REAL*4, REAL*8, REAL*16 arc tangent of quotient with angle measured in degrees

4. ATAND, DATAND, QATAND - The REAL*4, REAL*8, REAL*16 arc tangent with angle measured in degrees

5. DBLEQ - Convert REAL*16 to REAL*8

6. QACOS - REAL*16 arc cosine

7. QASIN - REAL*16 arc sine

8. QATAN - REAL*16 arc tangent

9. QATAN2 - REAL*16 arc tangent of a quotient

10. QCOS - REAL*16 cosine

11. QCOSD - REAL*16 cosine with angle measured in degrees

12. QCOSH - REAL*16 hyperbolic cosine

13. QDIM - Positive difference of two REAL*16 numbers

14. QEXP - REAL*16 exponential function

15. QEXT - Generic conversion to REAL*16

16. QEXTD - Convert REAL*8 to REAL*16

17. QLOG - REAL*16 natural logarithm

18. QLOG10 - REAL*16 common logarithm

19. QMAX1 - Finds the maximum of its REAL*16 arguments

20. QMIN1 - Finds the minimum of its REAL*16 arguments

21. QMOD - Remainder of two REAL*16 arguments

22. QSIGN - REAL*16 transfer of sign

23. QSIN - REAL*16 sine

24. QSIND - REAL*16 sine with angle measured in degrees

25. QSINH - REAL*16 hyperbolic sine

26. QSQRT - REAL*16 square root

27. QTAN - REAL*16 tangent

28. QTANH - REAL*16 hyperbolic tangent

29. SNGLQ - Convert REAL*16 to REAL*4

30. TAND, DTAND, QTAND - REAL*4, REAL*8, REAL*16 tangent of an angle measured in degrees

31. ZEXT, IZEXT, JZEXT - Zero extend a LOGICAL or INTEGER value giving an INTEGER

3.13.6 INTEGER Function Differences

The presence of both the INTEGER*2 and INTEGER*4 data types in VAX FORTRAN has caused VAX FORTRAN to treat the standard names of the intrinsic functions that take INTEGERs as arguments as generic names. These generic names select between the intrinsic functions that take the appropriate type of INTEGER argument. Table 3-2 shows which VAX INTEGER functions are selected by these names.

Since FORTRAN-10/20 has only one INTEGER data type, programs that are to be transported between the two machines generally should not refer to INTEGER functions using VAX INTRINSIC names, such as IIABS or JIINT. Instead, the INTEGER functions should be referred to by using the FORTRAN-77 standard name for the INTEGER functions, such as IABS or INT. However, the FORTRAN-10/20 runtime library does define the VAX intrinsic integer function names as synonyms for FORTRAN-10/20 integer function names. Note that these functions are not considered to be intrinsic by the FORTRAN-10/20 compiler.

Table 3-2: VAX-11 FORTRAN INTEGER Function Names

VAX FORTRAN Generic Name	Function of an INTEGER*2 Argument	Function of an INTEGER*4 Argument
IABS	IIABS	JIABS
INT	IINT	JIINT
IDINT	IIDINT	JIDINT
NINT	ININT	JNINT
IDNINT	IIDNNT	JIDNNT
FLOAT	FLOATI	FLOATJ
IFIX	IIFIX	JIFIX
DFLOAT	DFLOTI	DFLOTJ
MAXØ	IMAXØ	JMAXØ
MAX1	IMAX1	JMAX1
AMAXØ	AIMAXØ	AJMAXØ
MINØ	IMINØ	JMINØ
MIN1	IMIN1	JMIN1
AMINØ	AIMINØ	AJMINØ
IDIM	IIDIM	JIDIM
MOD	IMOD	JMOD
ISIGN	IISIGN	JISING

3.13.7 DOUBLE-PRECISION COMPLEX Functions

FORTRAN-1Ø/2Ø does not support the DOUBLE COMPLEX data type. However, it does supply subprograms to calculate many of the same functions as the VAX FORTRAN DOUBLE COMPLEX functions. The FORTRAN-1Ø/2Ø subprograms to do these calculations are subroutines that take arguments of two-element DOUBLE PRECISION arrays, which are manipulated as if they were DOUBLE COMPLEX variables. The remainder of this section describes the correspondence between the VAX FORTRAN functions of DOUBLE COMPLEX numbers and the appropriate FORTRAN-1Ø/2Ø subprograms.

The following are DOUBLE COMPLEX functions of DOUBLE COMPLEX arguments in VAX FORTRAN, but in FORTRAN-1Ø/2Ø they are subroutines of two DOUBLE PRECISION two-element arrays (the second argument is the result): CDSQRT, CDLOG, CDEXP, CDSIN.

The VAX FORTRAN function DCMPLX has no corresponding subprogram in FORTRAN-1Ø/2Ø. However, a similar effect could be obtained by assigning values to the elements of the two-element DOUBLE PRECISION array that is being used as a DOUBLE COMPLEX variable.

The VAX FORTRAN functions DREAL and DIMAG have no corresponding subprograms in FORTRAN-1Ø/2Ø. However, their effects could be obtained by referring to the first element (for the real part) or the second element (for the imaginary part) of the DOUBLE PRECISION array being used as a DOUBLE COMPLEX variable.

The VAX FORTRAN function DCONJG has no corresponding subprogram in FORTRAN-1Ø/2Ø. However, in FORTRAN-1Ø/2Ø, the complex conjugate can be formed by negating the second element of the DOUBLE PRECISION array being used as the DOUBLE COMPLEX variable.

The DOUBLE PRECISION function CDABS is a function in both VAX FORTRAN and FORTRAN-1Ø/2Ø. However, in VAX FORTRAN its argument is a DOUBLE COMPLEX variable, and in FORTRAN-1Ø/2Ø it is a two-element DOUBLE PRECISION array, which is treated as if it is a DOUBLE COMPLEX variable. Note that the generic function ABS in VAX FORTRAN may result in a call to the function CDABS. This can never happen in FORTRAN-1Ø/2Ø.

3.13.8 Similar Subroutines in FORTRAN-10/20 and VAX FORTRAN

The following subroutines exist in both FORTRAN-10/20 and VAX FORTRAN. However, they have various differences that are described below:

1. DATE - On both compilers, this subroutine returns the date in the form 'dd-mmm-yy'. However, when a numeric variable is used as the argument, VAX FORTRAN returns only those 9 characters, while FORTRAN-10/20 ends the date with a trailing blank, and thus returns 10 characters.

2. ERRSET - Allows the user to control the actions taken when an error occurs. This subroutine has different arguments in FORTRAN-10/20 than in VAX FORTRAN, and VAX FORTRAN ERRSET provides somewhat less capabilities than FORTRAN-10/20 ERRSET. The VAX Condition Handling Facility provides a more general method of defining the actions that are to be taken on error than VAX FORTRAN ERRSET, and is recommended over ERRSET.

3. ERRSNS - Determines the error number of the last error that occurred. The FORTRAN-10/20 ERRSNS has a different number of arguments than the VAX FORTRAN ERRSNS, and the error numbers on the two runtime systems are different.

4. EXIT - Terminates execution of the program. In VAX FORTRAN, EXIT optionally accepts one argument. The FORTRAN-10/20 EXIT does not take any arguments.

5. RAN - Returns a random number. This subroutine is similar to the RAN function in both FORTRAN-10/20 and VAX FORTRAN. Note that the FORTRAN-10/20 RAN and the VAX RAN use different algorithms, and thus return different sequences of random numbers.

6. MVBITS - Transfers a bit field from one storage location (source) to a field in a second storage location (destination).

7. TIME - Returns the time of day as a character string. In VAX FORTRAN, this subroutine takes one argument and returns the time of day in the form "hh:mm:ss". The FORTRAN-10/20 TIME subroutine takes either one or two arguments and returns the time in the form "hh:mm" in the first argument and " ss.t" in the second argument. (Where "hh" is hours, "mm" is minutes, "ss" is seconds, and "t" is tenths of seconds.)

8. SECNDS - Returns the number of seconds since midnight, minus the argument to this subroutine.

3.13.9 Subroutines Only Available in FORTRAN-10/20

The following subroutines are available only in FORTRAN-10/20:

1. ALCCHR - Allocates space on the character stack for dynamic character operations.

2. CHKDIV - Returns the number of the unit to which error messages are written.

3. CLRFMT - Discards a FORMAT statement saved by the execution of the SAVFMT routine.

4. DIVERT - Enables you to redirect error messages from the current device to an open file on a specified device.

5. DTOGA - Converts elements of DOUBLE-PRECISION arrays from D-floating format to G-floating format. (Available as Runtime Library Routine on VAX/VMS.)

6. DUMP - Causes specified portions of memory to be dumped to the line printer.

7. FFUNIT - Returns the unit number of the first available FORTRAN logical unit.

8. GTODA - Converts elements of DOUBLE-PRECISION arrays from G-floating format to D-floating format. (Available as Runtime Library Routine on VAX/VMS.)

9. ILL - Sets the ILLEG flag. The ILLEG flag, if set, allows illegal characters to appear in floating-point input. If such an illegal character is encountered, no error message is produced and the corresponding value is set to zero.

1Ø. LEGAL - Clears the ILLEG flag, and thus causes illegal characters in floating-point input to produce an error.

11. OVERFL - Returns information about the last overflow, underflow, or divide check.

12. PDUMP - Like the DUMP subroutine except that control returns to the calling program after the dump has been executed.

13. QUIETX - Suppresses all summary type-out when the program exits.

14. SAVFMT - Causes a format specification contained in the specified array or character variable to be encoded into an internal form that allows more efficient processing. (In VAX FORTRAN, the use of Variable Format Expressions (VFEs) provide most of the capability of SAVFMT.)

15. SAVRAN - Saves the last internal seed value generated by the RAN function.

16. SETRAN - Sets the internal seed value for the RAN function.

17. SORT - Sorts one or more files using the SORT program. (FORTRAN-10/20 SORT and VAX SORT use different arguments.)

18. SRTINI - Directs FOROTS to start allocating memory from top downward to account for large overlay programs and preallocates pages 600:677 for SORT.

19. TOPMEM - Directs FOROTS to start allocating memory from top downward to account for large overlay programs.

20. TRACE - Generates a list of the active subprograms on the terminal.

3.13.10 Subroutines Only Available in VAX FORTRAN

The following subroutines are only available in VAX FORTRAN:

1. ASSIGN - Associates a filename with a logical unit number. ASSIGN provides a subset of the capabilities of the OPEN statement.

2. CLOSE - Closes a file. The CLOSE subroutine provides a subset of the capabilities of the CLOSE statement.

3. ERRTST - Checks if a particular error occurred.

4. FDBSET - Specifies I/O options. FDBSET provides a subset of the capabilities of the OPEN statement.

5. IDATE - Returns the date as three integers (the number of the month, day, year).

6. IRAD50 - Converts Hollerith data to Radix-50 and counts the number of characters.

7. R50ASC - Converts Radix-50 data to Hollerith.

8. RAD5Ø - Converts Hollerith to Radix-5Ø.

9. RANDU - Returns a random number with a uniform distribution.

1Ø. USEREX - Sets the name of a subroutine to call as part of the program termination process.

CHAPTER 4

SOFTWARE AND HARDWARE LIMITS

4.1 SOFTWARE LIMITS

FORTRAN-10/20 and VAX FORTRAN apply different limits on the size and complexity of programs. This section describes different limits imposed by the two compilers and the associated run-time systems.

4.1.1 Number of Continuation Lines

Both FORTRAN-10/20 and VAX FORTRAN accept at least 19 continuation lines, but the differences in the two compilers may cause a statement that contains more than 19 continuations to be acceptable to one compiler and not the other.

VAX FORTRAN allows the continuation limit to be set from 0 to 99 lines using the /CONTINUATIONS qualifier (the default value is 19). Although the /CONTINUATIONS qualifier takes a value measured in lines, the number of continuations allowed by VAX FORTRAN is actually based on the number of characters in the statement field of the lines. Thus, VAX FORTRAN allows more continuation lines than the value of /CONTINUATIONS implies if the continuation lines are short. (The /EXTEND_SOURCE qualifier increases the length of lines; therefore, it increases the total number of characters to be processed for the /CONTINUATIONS value.)

The continuation line limit of FORTRAN-1Ø/2Ø (like VAX FORTRAN) also is actually based on characters instead of lines. But unlike VAX FORTRAN, FORTRAN-1Ø/2Ø does not require that all the lines that make up the statement fit into the character buffer. FORTRAN-1Ø/2Ø only requires that the type of the statement be able to be determined from the number of characters that fit into the buffer. Thus, if FORTRAN-1Ø/2Ø can determine from the first 19 lines of a statement that the statement is an assignment, then it places no actual limit on the number of lines that make up the statement.

4.1.2 Other Compiler Limits

Table 4-1 lists other limits that FORTRAN-1Ø/2Ø and VAX FORTRAN place on the complexity of programs. Where "Memory" is entered in the FORTRAN-1Ø/2Ø column, it means that FORTRAN-1Ø/2Ø imposes no real limit on the complexity of the user subprogram being compiled other than that all the compiler's tables and work areas must be able to fit into memory. FORTRAN-1Ø/2Ø has about 1ØØK words available for this purpose.

Currently, FORTRAN-2Ø does produce code that can run in the extended memory sections available under TOPS-2Ø. However, FORTRAN-1Ø programs plus the run-time system on the DECsystem-1Ø must fit into 256K words. This is a much more severe restriction on the size of user programs than is made on a VAX.

Table 4-1: Compiler Imposed Limitations

Item	VAX FORTRAN	FORTRAN-1Ø/2Ø
# of DO loops that can be nested	2Ø	79
Length of character constants	2ØØØ characters	Memory
Length of character variables	65535 characters	Memory
Number of array subscripts	7	128
Number of arguments to subprograms	255	127
Number of named COMMON blocks	25Ø	Memory
Number of NAMELIST elements	25Ø	Memory
Length of PAUSE messages	255 characters	Memory
# of INCLUDE statements that can be nested	1Ø	12

4.1.3 I/O Limits

Table 4-2 lists the limits in the maximum record length imposed by the run-time systems of FORTRAN-1Ø/2Ø and VAX FORTRAN. Where "Memory" is entered in the FORTRAN-1Ø/2Ø column, it means that FORTRAN-1Ø/2Ø imposes no real limit on the maximum record length.

Table 4-2: Maximum Record Lengths

	VAX		10/20	
	Formatted (Bytes)	Unformatted (Longwords)	Formatted (Bytes)	Unformatted (Words)
Sequential	32766	8191	655360	Memory
Sequential, Variable records of ASCII tape	9999	2499	9999	8192
Indexed	16380	4095	17985	3579
Relative	16380	4095	262143	52428

4.2 HARDWARE LIMITS

This section describes how the FORTRAN data types are represented by the hardware of the DECSYSTEM-10/20 and VAX processors, and the implications that this has for the FORTRAN programmer.

4.2.1 The INTEGER Data Type

Both the DECSYSTEM-10/20 and the VAX processors store integers using two's complement representation with the left-most bit being the sign bit. VAX FORTRAN has two integer data types: INTEGER*2 and INTEGER*4. INTEGER variables that do not have a specified length are treated as INTEGER*4 variables by default. The programmer can cause these variables to be treated as INTEGER*2 by specifying the /NOI4 qualifier.

FORTRAN-10/20 has only one integer data type: INTEGER*4. If an attempt is made to declare a variable as INTEGER*2, a warning is issued and INTEGER*4 is used.

Table 4-3 compares the different INTEGER data types.

Table 4-3: Integer Number Format

Data Type	Bits	Range
10/20 INTEGER*4	36	-34359738368 to 34359738367
VAX INTEGER*2	16	-32768 to 32767
VAX INTEGER*4	32	-2147483648 to 2147483647

4.2.2 The LOGICAL Data Type

FORTRAN-10/20 has only one LOGICAL data type: LOGICAL*4. It is stored as a word with the high-order bit (the sign bit) determining whether the word is true or false. If an attempt is made to declare LOGICAL*1 variables, FORTRAN-10/20 issues a warning and uses LOGICAL*4. If an attempt is made to declare LOGICAL*2 variables, an error is given.

VAX FORTRAN has three LOGICAL data types: LOGICAL*1, LOGICAL*2, and LOGICAL*4. They are implemented as a single byte, two bytes, or a long word, respectively. VAX FORTRAN tests the low-order bit to determine whether a LOGICAL variable is true or false.

Note that FORTRAN-10/20 and VAX FORTRAN test different bits to determine whether the value contained in a logical variable is true or false. (Section 3.4 explains the impact that this can have on programs.)

Table 4-4 summarizes the LOGICAL data types.

Table 4-4: Logical Data Types

Data Type	Bits	Range (if used to store an INTEGER)
1Ø/2Ø LOGICAL	36	34359738368 to 34359738367
VAX LOGICAL*1	8	-128 to 127
VAX LOGICAL*2	16	-32768 to 32767
VAX LOGICAL*4	32	-2147483648 to 2147483647

4.2.3 The CHARACTER and Hollerith Data Types

Both FORTRAN-1Ø/2Ø and VAX FORTRAN represent CHARACTER and Hollerith data as ASCII characters packed into contiguous locations. However, two important differences make the maximum number of characters that can be packed into the FORTRAN data types different on the two machines:

- Characters on a DECSYSTEM-1Ø/2Ø are represented by 7 bits, and characters on a VAX are represented by 8 bits.

- The number of bits used to store the FORTRAN data types on a DECSYSTEM-1Ø/2Ø differs from the number of bits used to store the FORTRAN data types on a VAX.

These differences do not affect the FORTRAN programmer who uses the CHARACTER data type, because all the operations on CHARACTER data are carried out transportably by the compiler and run-time system. However, these differences greatly affect programs that use Hollerith data. Single-word variables and constants can hold up to five characters on a DECSYSTEM-1Ø/2Ø, but only four characters on a VAX.

Table 4-5 lists the maximum number of characters that can be packed into the various FORTRAN data types.

Table 4-5: Characters per FORTRAN Data Type

Data Type	Number of Characters on a VAX	Number of Characters on a DECSYSTEM-10/20
BYTE	1	Not Available
COMPLEX	8	10
COMPLEX*16	16	Not Available
INTEGER*2	2	Not Available
INTEGER*4	4	5
LOGICAL*1	1	Not Available
LOGICAL*2	2	Not Available
LOGICAL*4	4	5
REAL*4	4	5
REAL*8	8	10
REAL*16	16	Not Available

Another incompatibility is that VAX programs can manipulate individual characters in Hollerith data by storing the Hollerith string in a LOGICAL*1 or BYTE array. Since FORTRAN-10/20 does not support these data types, this cannot be done on a DECSYSTEM-10/20 processor.

The best way to avoid the system dependent aspects of Hollerith data is not to use it; the CHARACTER data type is superior and is transportable.

4.2.4 The Floating-Point Data Types

Both the DECSYSTEM-10/20 and VAX processors have several floating-point formats that allow the programmer a great deal of flexibility in solving numerical problems. These formats are the following:

- F-floating format minimizes storage space and execution time at the expense of both range and precision.
- D-floating format requires twice as much space and has the same range as F-floating, but provides more than twice the precision.
- G-floating format requires the same space as D-floating, but trades a little precision for greatly increased range.
- H-floating format (only available on the VAX) requires much more storage space and execution time, but greatly increases the precision and range of numbers that can be represented.

The corresponding number formats on the two different machines have many similarities including:

- Floating-point numbers are represented as a signed fraction multiplied by two raised to the power of some exponent.
- The exponent field is stored in excess 1024 notation for G-floating, and in excess 128 notation for F-floating and D-floating.
- The fraction is normalized, and its binary point is to the left of the most significant bit.

However, there are several important differences that must be taken into consideration when moving an application from one processor to the other.

Unlike the DECSYSTEM-1Ø/2Ø, which stores the most significant bit of the mantissa, the VAX floating-point number formats use hidden bit normalization. Hidden bit normalization means that the high-order bit of the mantissa, which is always one because numbers are normalized, is not stored.

This has two effects: First, since there is no way to have a zero mantissa (since there is always that hidden bit, which is one), the zero exponent field is reserved to mean that the number represented is zero. Thus, VAX floating-point numbers do not have as great a range as floating-point numbers of the corresponding DECSYSTEM-1Ø/2Ø floating-point number format. Second, since the most significant mantissa bit is not stored, VAX makes more effective use of its mantissa bits. This causes VAX floating-point number formats to be slightly more precise than the word size of the VAX would imply.

The two machine architectures use different representations for negative floating-point numbers. The VAX processor simply has a sign bit; if the bit is zero, then the number is positive. If the bit is one, then the number is negative. The sign bit for VAX floating-point numbers is not in the same bit position in a long word as the sign bit for VAX integer numbers. Thus, if an INTEGER and a REAL variable are equivalenced, the same bit pattern could test negative as the REAL variable and positive as the INTEGER variable. The DECSYSTEM-1Ø/2Ø processor represents negative floating-point numbers as the two's complement of the positive floating-point numbers. This representation of negative numbers means that a DOUBLE-PRECISION variable cannot be equivalenced to a REAL variable in order to truncate the double-precision number to single precision; that operation can result in unnormalized numbers.

The F-floating number format is used to store the REAL*4 and COMPLEX*8 data types. Table 4-6 compares the F-floating number format on the DECSYSTEM-1Ø/2Ø and VAX processors.

The D-floating number format is used to store the REAL*8 and COMPLEX*16 (VAX FORTRAN only) data types. Table 4-6 compares the D-floating number format on the DECSYSTEM-1Ø/2Ø and VAX processors.

The G-floating number format is used to store the REAL*8 and COMPLEX*16 (VAX FORTRAN only) data types if the /GFLOATING switch is given to FORTRAN-1Ø/2Ø, or the /G_FLOATING qualifier is given to VAX FORTRAN. Not all DECSYSTEM-1Ø/2Ø or VAX processors support G-floating. On VAX processors without G-floating, the runtime library may be used to provide software emulation of G-floating. Table 4-6 compares the G-floating number formats on the two machines.

The H-floating number format is available only on the VAX. The REAL*16 data type in VAX FORTRAN uses the H-floating format. Table 4-6 compares H-floating to the other floating-point number formats. Not all VAX processors support H-floating. On VAX processors without H-floating, the runtime library may be used to provide software emulation of H-floating.

Table 4-6: Floating-Point Number Formats

	Bits of Exponent	Bits of Mantissa	Range	Digits of Precision
			F-FLOATING NUMBER FORMAT	
10/20	8	27	1.47 x 10**-39 to 1.70 x 10**+38	8.1
VAX	8	24	2.94 x 10**-39 to 1.70 x 10**+38	7.2
			D-FLOATING NUMBER FORMAT	
10/20	8	62	1.47 x 10**-39 to 1.70 x 10**+38	18.7
VAX	8	56	2.94 x 10**-39 to 1.70 x 10**+38	16.9
			G-FLOATING NUMBER FORMAT	
10/20	11	59	27.78 x 10**-309 to 8.99 x 10**+307	17.8
VAX	11	53	5.56 x 10**-309 to 8.99 x 10**+307	16.0
			H-FLOATING NUMBER FORMAT	
VAX	15	113	8.4 x 10**-4933 to 5.9 x 10**+4931	34.0

NOTE

There is a symmetric (mirror-image) negative range, as well as a unique representation for zero.

INDEX

FORTRAN-10/20
and VAX FORTRAN
Compatibility Manual
AA-Y868C-TK

READER'S COMMENTS

NOTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

Did you find this manual understandable, usable, and well-organized? Please make suggestions for improvement.

Did you find errors in this manual? If so, specify the error and the page number.

Please indicate the type of reader that you most nearly represent.

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Other (please specify) ______

Name ______ Date ______

Organization ______ Telephone ______

Street ______

City ______ State ______ Zip Code or Country ______

---Do Not Tear - Fold Here and Tape---

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO. 33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SOFTWARE PUBLICATIONS
200 FOREST STREET MRO1-2/L12
MARLBOROUGH, MA 01752

-- Do Not Tear - Fold Here and Tape---

Cut Along Dotted Line