FORTRAN

Compatibility Guide

digital

FORTRAN Compatibility Guide

Order Number: EJ–31491–41

MAY 1988

This manual documents the compatibility issues that exist between the DIGITAL FORTRAN IV, FORTRAN–77, and VAX FORTRAN compilers for PRO, PDP–11, and VAX systems.

Revision/Update Information: This is a new manual.

Software Version: FORTRAN IV Version 2.6
FORTRAN–77 Version 5.2
VAX FORTRAN Version 4.7

digital equipment corporation
maynard, massachusetts

MAY 1988

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist in preparing future documentation.

The following are trademarks of Digital Equipment Corporation:

DEC	DIBOL	UNIBUS
DEC/CMS	EduSystem	VAX
DEC/MMS	IAS	VAXcluster
DECnet	MASSBUS	VMS
DECsystem-10	PDP	VT
DECSYSTEM-20	PDT	and the DIGITAL logo
DECUS	RSTS	
DECwriter	RSX	

ZK4701

Production Note

This book was produced with the VAX DOCUMENT electronic publishing system, a software tool developed and sold by DIGITAL. In this system, writers use an ASCII text editor to create source files containing text and English-like code; this code labels the structural elements of the document, such as chapters, paragraphs, and tables. The VAX DOCUMENT software, which runs on the VMS operating system, interprets the code to format the text, generate a table of contents and index, and paginate the entire document. Writers can print the document on the terminal or line printer, or they can use DIGITAL-supported devices, such as the LN03 laser printer and PostScript™ printers (PrintServer 40 or LN03R ScriptPrinter), to produce a typeset-quality copy containing integrated graphics.

™ PostScript is a trademark of Adobe Systems, Inc.

Contents

Preface

This manual provides reference information for transporting FORTRAN programs from FORTRAN IV to FORTRAN–77 and VAX FORTRAN, and from FORTRAN–77 to VAX FORTRAN. This manual is intended to augment, not replace, the information contained in the various FORTRAN compiler documentation sets.

Intended Audience

This manual is intended for programmers who are transporting their programs from FORTRAN IV to FORTRAN–77 and VAX FORTRAN, and from FORTRAN–77 to VAX FORTRAN.

Document Structure

This manual has two chapters. They are as follows:

Chapter 1, Introduction, describes the relationship of the various FORTRAN implementations to the different FORTRAN standards. It also describes the process of developing FORTRAN programs on VMS, RSX, RSTS/E, and P/OS systems.

Chapter 2, Language Compatibility Issues, describes in detail all the FORTRAN language features that cause transportability problems across VMS, RSX, RSTS/E, and P/OS systems.

Associated Documents

For more information on topics discussed in this manual, refer to the appropriate manuals in the FORTRAN documentation sets.

Conventions

Convention	Meaning
RET	In examples, a key name (usually abbreviated) shown within a box indicates that you press a key on the keyboard; in text, a key name is not enclosed in a box. In this example, the key is the RETURN key. (Note that the RETURN key is not usually shown in syntax statements or in all examples; however, assume that you must press the RETURN key after entering a command or responding to a prompt.)
CTRL/C	A key combination, shown in uppercase with a slash separating two key names, indicates that you hold down the first key while you press the second key. For example, the key combination CTRL/C indicates that you hold down the key labeled CTRL while you press the key labeled C. In examples, a key combination is enclosed in a box.
$ SHOW TIME 05-JUN-1988 11:55:22	In examples, system output (what the system displays) is shown in black. User input (what you enter) is shown in red.
$ TYPE MYFILE.DAT . . .	In examples, a vertical series of periods, or ellipsis, means either that not all the data that the system will display in response to a command is shown or that not all the data you enter is shown.
input-file, . . .	In examples, a horizontal ellipsis indicates that additional parameters, values, or other information can be entered, that preceding items can be repeated one or more times, or that optional arguments in a statement have been omitted.
[logical-name]	Brackets indicate that the enclosed item is optional. (Brackets are not, however, optional in the syntax of a directory name in a file specification, or in the syntax of a substring specification in an assignment statement.)
quotation marks apostrophes	The term quotation marks is used to refer to double quotation marks (″). The term apostrophe (′) is used to refer to a single quotation mark.

Chapter 1

Introduction

This chapter discusses the relationships of the various FORTRAN language implementations to each other and to any existing standards, for example the American National Standards. The DIGITAL FORTRAN implementations are designed to comply with the current FORTRAN ANSI standard; however, each implementation also includes extensions. This chapter highlights these extensions.

This chapter also discusses the tools and methods of program development on the following systems: PRO, PDP-11, and VAX.

1.1 Language Implementation Comparisons

Both VAX FORTRAN and FORTRAN-77 are based on American National Standard FORTRAN-77 (ANSI X3.9-1978). This standard corresponds to the ISO standard for FORTRAN-77 (ISO 1539-1980 (E)) and the FIPS 69-1 standard. FORTRAN IV conforms to the previous FORTRAN standard (ANSI X3.9-1966); however, there is upward compatibility with the more recent standard.

VAX FORTRAN, FORTRAN-77, and FORTRAN IV all support some common extensions to the standards on which they are based. For example, all three compilers support tab-format lines. That is, instead of using a fixed format, you can specify the statement label field, the continuation indicator field, and the statement field using tab formatting. All three compilers support Hollerith constants and other additional data types.

VAX FORTRAN and FORTRAN-77 support the following additional extensions to the ANSI X3.9-1978 standard:

- Indexed file organizations
- Relative file organization
- Exclamation point (!) as a comment indicator for comments at the end of a line of source code
- Letter D in column 1 to indicate a debugging statement (for use with the /D_LINES qualifier)
- Arithmetic expressions as control parameters in computed GO TO statements
- INCLUDE statements to incorporate statements from a separate file into a FORTRAN program during compilation

VAX FORTRAN also supports the following extensions to the ANSI X3.9-1978 standard that are not supported by FORTRAN-77:

- Conformance with the VAX procedure-calling standard

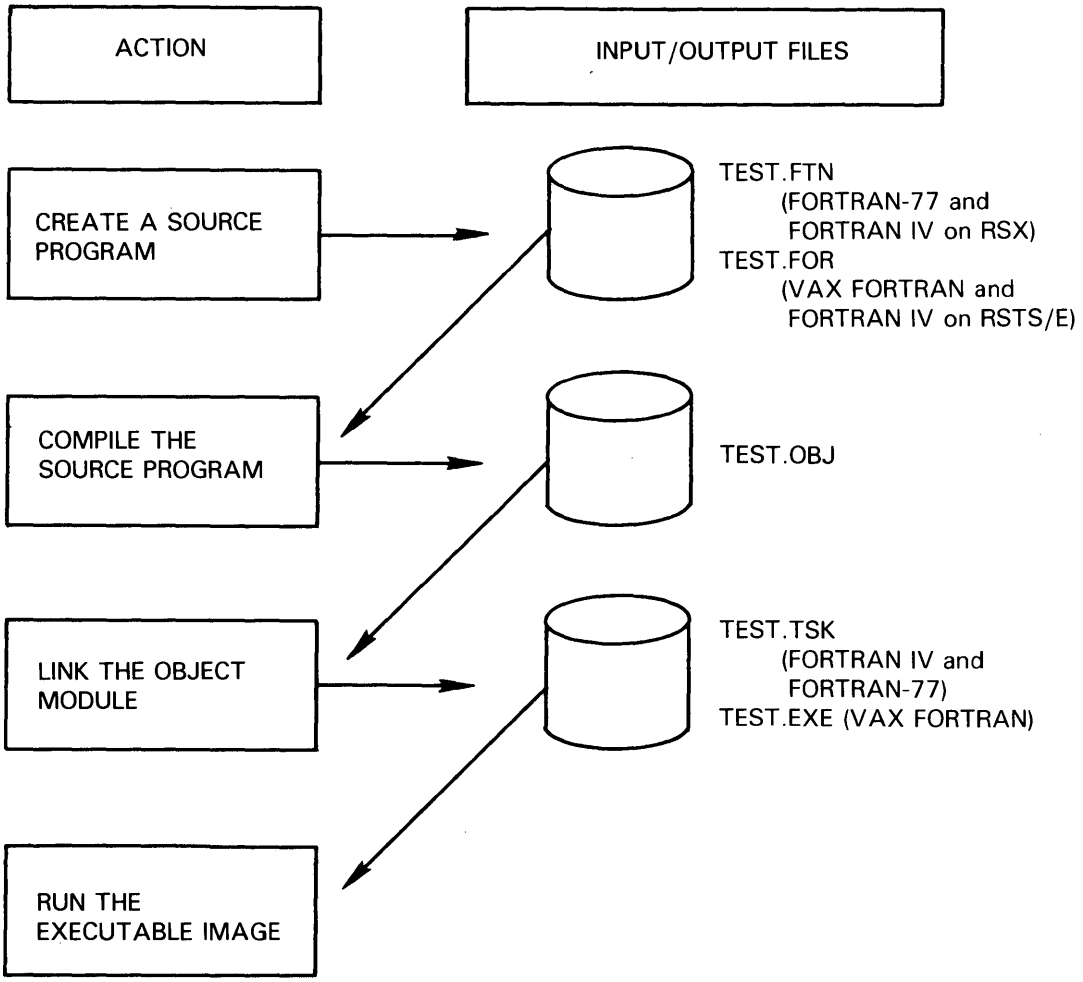
- Records and structures
- DO WHILE statement
- Namelist-directed input and output
- IMPLICIT NONE statement
- Support for double-precision complex (D_floating and G_floating), REAL*16 (H_floating), and G_floating double-precision data types
- Intrinsic functions to manipulate the bits in binary patterns that represent integers

Both VAX FORTRAN and FORTRAN-77 provide a /STANDARD qualifier to the FORTRAN compile command. This standard causes the compiler to flag any extensions to the ANSI X3.9-1978 standard. For more information, see Chapter 2.

1.2 Developing FORTRAN Programs

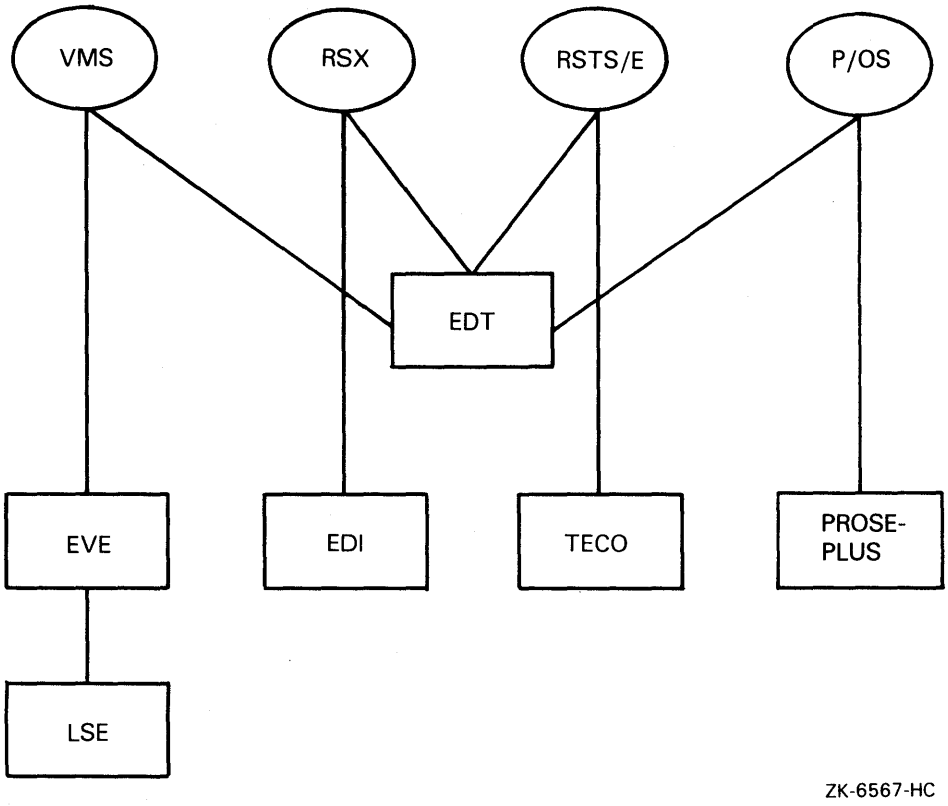
To develop FORTRAN programs on VMS, RSX-11M-PLUS, RSTS/E, and P/OS systems, you must invoke a text editor to create the source program and then compile, link, and run it. Figure 1-1 shows this process.

Figure 1–1: Commands for Program Development



Different operating systems allow you to use different text editors. For compatibility, it is recommended that you use EDT, since EDT is available on most operating systems. Figure 1–2 illustrates all of the text editors that are available on PRO, PDP–11, and VAX systems. The following sections briefly describe these editors. For more information, see the appropriate operating system compatibility guide.

Figure 1–2: Text Editors



1.2.1 Using EDT

EDT is an interactive, general-purpose text editor that offers three editing modes: keypad, nokeypad, and line. With keypad mode, you issue commands by using the numeric keypad that appears to the right of your main keyboard. With nokeypad mode, you issue commands on a command line, which EDT processes when you press the RETURN key. Line mode focuses on the line as the unit of text. With line mode, you issue commands at the line mode asterisk prompt (*).

Keypad mode and nokeypad mode continually display the contents of the file on your screen. When you begin your editing session, editing in line mode is the default. Unlike keypad and nokeypad mode, line mode is not suitable for editing text arranged in paragraphs.

The following command line invokes the EDT editor and creates the file PROG1.FTN:

```
$ EDIT/EDT PROG1.FTN
```

To change from line mode to keypad mode, type the CHANGE command, or simply C, at the asterisk prompt. To return to line mode from keypad mode, press CTRL/Z. To change from line mode to nokeypad mode, type the SET NOKEYPAD command and then type the CHANGE command at the asterisk prompt.

When you invoke EDT to create a file, a journal file is automatically created. You can use this journal file to recover your edits if the system fails during an editing session. To recover your edits, type the EDIT/RECOVER command specifying the name of the file you were editing originally, not the name of the journal (.JOU) file.

EDT provides an online HELP facility that you can access during an editing session. In line mode, type the HELP command. EDT displays general information on EDT as well as detailed information on both line mode editing and nokeypad mode editing. In keypad mode, press the HELP key or the PF2 key. EDT displays a keypad diagram on your screen and a list of keypad editing keys. For help on a specific keypad function, press the key you want help on.

For more detailed information on how to use EDT, see the EDT documentation set for the appropriate operating system or the *VAX EDT Reference Manual.*

1.2.2 Using VAXTPU

The VAX Text Processing Utility (VAXTPU) is a high-performance, programmable utility that is only available on VMS systems. VAXTPU provides the Extensible VAX Editor (EVE). You can also create your own interfaces.

EVE is an interactive text editor that allows you to execute editing functions using either the EVE keypad or the EVE command line. The following command line invokes the EVE editor and creates the file PROG_1.FOR:

```
$ EDIT/TPU PROG_1.FOR
```

You can define a global symbol for the EDIT/TPU command by placing a symbol definition in your LOGIN.COM file. For example:

```
$ EVE == "EDIT/TPU"
```

Once this command line is executed, you can type EVE at the DCL prompt followed by the name of the file you want to modify or create.

For more information on using the advanced features of EVE, see the *Guide to VMS Text Processing.*

Like EDT, VAXTPU provides you with an online HELP facility that you can access during your editing session. For help on the keypad, press the PF2 key. For general help on VAXTPU commands, press the DO key and enter HELP. For help on a specific command, press the DO key and enter the name of the command you want help on.

Like EDT, when you invoke VAXTPU to create a file, a journal file is automatically created. You can use this journal file to recover your edits if the system fails during an editing session. To recover your edits, include the /RECOVER qualifier on the command line that invokes the editor.

Unlike EDT, however, VAXTPU supports multiple windows and buffers. This feature allows you to view two files on your screen at the same time, or remote parts of the same file.

1.2.3 Using VAXLSE

The VAX Language-Sensitive Editor (VAXLSE) is available only with VMS and can be used as a multiwindow, screen-oriented text editor. Users familiar with EDT or EVE can use the corresponding keypads with VAXLSE.

One of the major features of VAXLSE is that it provides language templates that supply the correct format and syntax for whatever programming language you are currently using. For example, by typing the following line at the DCL command prompt, you initiate a VAXLSE editing session that includes the appropriate template for VAX FORTRAN:

```
$ LSEDIT PROG_2.FOR
```

Within this editing session, you can code a syntactically correct VAX FORTRAN program without knowing anything about FORTRAN syntax. VAXLSE allows you to compile your source programs and review and correct any errors, all without ever leaving the editing session. In addition, VAXLSE also allows you to create templates for languages that you define.

1.2.4 Using EDI

The EDI text editor lets you work with text one line at a time. The editor has two control modes: edit mode and input mode. In edit mode, you issue commands to change text already created. In input mode, you type text that goes into the file.

If you use EDI on RSX, and want to move a line editor on VMS, your best choice is to move to EDT and use the line editing mode there. Some of the commands are the same, but not all. For example, both EDI and EDT have INSERT and DELETE commands that work the same. However, the LOCATE command in EDI is the same as the FIND command in EDT. In general, EDT's line editing mode offers more capabilities than EDI.

For more information on the EDI editor, see the *RSX–11M–PLUS Utilities Manual.*

1.2.5 Using TECO

TECO is a powerful text editing program that is supported on RSTS/E systems. You may use TECO to edit any form of ASCII text such as source programs. Since TECO is a character-oriented editor rather than a line editor, text edited with TECO does not have line numbers associated with it, and it is not necessary to replace an entire line of text to change one character. Because TECO is not supported on other systems, it is recommended that you not use it if compatibility is an issue.

1.2.6 Using PROSE-PLUS

PROSE-PLUS is a compound document available only on P/OS systems. This editor enables you to combine both text and graphics into a single document. PROSE-PLUS immediately displays the effects of all text editing functions such as bold, underline, and right-margin justification. It also provides the ability to display the combined text and graphics document prior to printing.

To create drawings, you can either use the picture mode provided by PROSE-PLUS, or you can insert GIDIS format graphic files that were created by other applications.

PROSE-PLUS also features an interactive spelling checker with a base dictionary of over 60,000 words, along with a user dictionary for proper names and acronyms.

Chapter 2

Language Compatibility Issues

This chapter discusses the compatibility issues that exist between VAX FORTRAN, PDP-11 FORTRAN IV, PDP-11 FORTRAN-77, and PRO/Tool Kit FORTRAN-77. Compatibility issues generally deal with features that are available only on a particular system. However, they can also deal with inconsistencies in the behavior of certain FORTRAN language elements, depending on which FORTRAN language implementation you are using.

2.1 Principles of Writing Transportable Code

Writing FORTRAN code that runs on PRO, PDP-11, and VAX systems is an effort well spent. The techniques for writing transportable FORTRAN code coincide with many of the techniques of good programming practice, so that your transportable code will be understandable and easily maintained. Additionally, the effort involved in writing transportable code is small compared to the effort required to convert a program designed without considering transportability.

The following sections describe some programming practices to use when designing transportable FORTRAN programs. These practices may help you avoid some of the common pitfalls encountered when running FORTRAN programs on PRO, PDP-11, and VAX systems.

2.1.1 General Principles of Transportability

There are two important principles to remember when writing transportable code:

1. Modularize the code
2. Isolate system dependencies

Modularizing the code is important because it gives you logical, functional program units that are easy to create and maintain. Additionally, if you break your program into manageable units, you can isolate system dependencies in a single module, so that only that module is affected when you need to run your program on a different system. Although this makes maintenance and transportability of system-specific code easier, it is still a good idea to avoid using any system-specific features when writing transportable programs unless absolutely necessary.

2.1.2 Speed of Program Execution

Programs that use system-specific functions or capabilities generally execute faster than those that do not, because system-specific capabilities are designed to make the best possible use of the system hardware and software or both. However, in gaining execution speed, your program must sacrifice transportability. You must weigh both of these factors when determining whether or not to use system-specific capabilities in your applications.

2.2 Incompatibilities in FORTRAN Language Elements

PDP–11 FORTRAN IV is a compatible subset of PRO/Tool Kit and PDP–11 FORTRAN–77, which is a compatible subset of VAX FORTRAN.

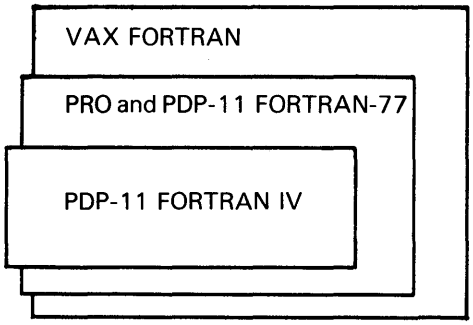
NOTE

The implementation of FORTRAN–77 for the PDP–11 and the PRO are almost identical, so throughout this chapter both of these compilers are referred to as FORTRAN–77.

Because of this layering of the various compilers, any FORTRAN–77 program not using superset (extension) features will generally run correctly under FORTRAN IV, and any FORTRAN–77 or FORTRAN IV program will run correctly under VAX FORTRAN, because VAX FORTRAN is an extension of PRO and PDP–11 FORTRAN. Similarly, any VAX FORTRAN program not using any language extensions can run under either FORTRAN IV or FORTRAN–77. Remember that FORTRAN IV is based on a previous standard, so any incompatibilities in the standards will cause corresponding incompatibilities between FORTRAN IV, FORTRAN–77, and VAX FORTRAN.

Figure 2–1 illustrates the layering of the various FORTRAN compilers.

Figure 2–1: Layering of the Various FORTRAN Compilers



ZK-6569-HC

To run FORTRAN IV or FORTRAN-77 on VMS systems, you must install VAX-11 RSX. Under VAX-11 RSX, both FORTRAN IV and FORTRAN-77 support the transporting of object code to RSX systems.

The following restrictions apply when you are running either PDP-11 FORTRAN compiler on a VMS system under VAX-11 RSX:

- FORTRAN IV only supports the linking and running of programs that do not use virtual arrays.
- FORTRAN-77 does not support the running of tasks.

Aside from the FORTRAN language extensions, the compatibility of your PRO, PDP-11, and VAX FORTRAN programs may also be affected by the following:

- Differences between the hardware architectures of PRO, PDP-11, and VAX systems
- Differences between the P/OS, RSX-11, RSTS/E, and VMS operating systems
- Differences between the standards on which the various language implementations are based

2.2.1 Invoking the Compilers

The method used to invoke the various FORTRAN compilers varies based on the following factors:

- The FORTRAN compiler itself
- The operating system
- The command line interface

Table 2-1 illustrates the different methods used to invoke the various FORTRAN compilers.

Table 2-1: Invoking a FORTRAN Compiler

Compiler	System	CLI	Command Line
FORTRAN IV	RSTS/E	DCL	$ RUN $FORTRAN[1] FOR> [obj] [, list][/sw] = file1[/sw]
			$ FORTRAN/FOR[/qual] file[/qual][2]
	RSX	DCL	$ FORTRAN/FOR[/qual] file[/qual]
		MCR	> FOR FOR> [obj] [, list][/sw] = file1[/sw]
FORTRAN-77	RSX, P/OS, and RSTS/E	DCL	$ FORTRAN/F77[/qual] file[/qual]
	RSX, RSTS/E, and VAX-11 RSX	MCR	> F77 F77> [obj] [, list][/sw] = file1[/sw]
VAX FORTRAN	VMS	DCL	$ FORTRAN[/qual] file

[1]This command invokes the RSTS/E FORTRAN IV compiler as if under RT-11.

[2]This command invokes the RSTS/E FORTRAN IV compiler as if under RSX.

NOTE

Any notation used for FORTRAN-77 and FORTRAN IV on RSX systems can be used under VAX-11 RSX.

2.2.2 Compiler Qualifiers and Switches

FORTRAN command qualifiers influence the way in which the compiler processes your file. In many cases, the simplest form of the FORTRAN command is sufficient. However, you can select appropriate optional qualifiers if you need special processing.

FORTRAN IV, FORTRAN-77, and VAX FORTRAN support many similar FORTRAN DCL command qualifiers. FORTRAN IV and FORTRAN-77 also support switches, many of which correspond to a particular DCL command qualifier. If you have VAX-11 RSX installed on a VMS system, you can use these switches when compiling a FORTRAN IV or FORTRAN-77 program.

Table 2-2 lists the DCL qualifiers and the corresponding switches, and indicates their availability for the various FORTRAN implementations.

NOTE

The switches listed in Table 2-2 and in the subsequent sections consist of MCR switches. These switches are available when you invoke FORTRAN IV on RSX systems or FORTRAN-77 using the MCR command line interface. RSTS/E systems do not support the MCR command line interface; however, you can implement the specified switches for interactive compilations by using the FORTRAN IV compiler on RSTS/E systems. That is, once you have invoked the FORTRAN IV compiler on RSTS/E systems, you can use the specified switches as if you were invoking the compiler using the MCR command line interface.

Table 2-2: FORTRAN Command Qualifiers and Switches

Qualifier	Switch	FORTRAN IV	FORTRAN-77	VAX FORTRAN
	/A[2]	Yes	No	No
/ANALYSIS_DATA		No	No	Yes
/CHECK	/CK	No	Yes	Yes
/CODE	/CD[1], /I[2]	Yes	No	No
/CONTINUATIONS	/CO	No	Yes	Yes
/CROSS_REFERENCE		No	No	Yes
/DEBUG	/DB	No	Yes	Yes[3]
	/DI[1], /B[2]	Yes	No	No
/D_LINES		No	No	Yes

[1]For FORTRAN IV, this switch is available only on RSX systems.

[2]For FORTRAN IV, this switch is available only on RSTS/E systems.

[3]This qualifier exhibits some minor differences, depending on which FORTRAN language implementation you are using. Refer to the appropriate qualifier section for more information.

Table 2–2 (Cont.): FORTRAN Command Qualifiers and Switches

Qualifier	Switch	FORTRAN IV	FORTRAN-77	VAX FORTRAN
/DLINES	/DE[1], /D[2]	Yes	Yes	No[4]
/DML		No	No	Yes
/EXTEND	/EX[1], /E[2]	Yes	Yes	No[4]
/EXTEND_SOURCE		No	No	Yes
/FOR		Yes	No	No
/F77	/F77	No	Yes	Yes
/G_FLOATING		No	No	Yes
/IDENTIFICATION	/ID[1]	Yes	Yes	No[6]
/I4	/I4[1], T[2]	Yes	Yes	Yes
	/LA	No	Yes	No
/LIBRARY		No	No	Yes
/LINE_NUMBERS	/SN[1], /S[2,7]	Yes	No	No
/LIST	/LI[1], /L[2]	Yes	Yes	Yes[3]
	/LO[1], /Q[2,7]	Yes	No	No
/MACHINE_CODE	/LI[1], /L[2]	Yes	Yes	Yes[3]
/MAP	/LI[1], /L[2]	Yes	Yes	No[4]
	/N[2]	Yes	No	No
	/O[2]	Yes	No	No
/OBJECT		No	Yes	Yes
/OPTIMIZE	/OP	No	Yes	Yes
	/R[2]	Yes	No	No
/SHAREABLE	/RO[1], /Z[2]	Yes[5]	Yes[5]	No
/SHOW		No	No	Yes[4]
/SOURCE	/LI[1], /L[2]	Yes	Yes	No
	/SP[1]	Yes	No	No
/STANDARD	/ST	No	Yes	Yes
/TRACEBACK	/TR	No	Yes	No

[1]For FORTRAN IV, this switch is available only on RSX systems.

[2]For FORTRAN IV, this switch is available only on RSTS/E systems.

[3]This qualifier exhibits some minor differences, depending on which FORTRAN language implementation you are using. Refer to the appropriate qualifier section for more information.

[4]The effects of this qualifier can be implemented in VAX FORTRAN using another qualifier. Refer to the section describing this qualifier for more information.

[5]This qualifier is available only on the RSX–11M–PLUS operating system.

[6]This qualifier or switch is not provided on certain systems because the specified behavior is performed by default.

[7]This qualifier is not synonymous with the other associated qualifiers or switches. Instead, it inhibits the behavior specified by the associated qualifiers.

Table 2–2 (Cont.): FORTRAN Command Qualifiers and Switches

Qualifier	Switch	FORTRAN IV	FORTRAN-77	VAX FORTRAN
	/U[2]	Yes	No	No
/VECTORS	/VA[1], /V[2,7]	Yes	No	No
/WARNINGS	/WR[1], /W[2]	Yes	Yes	Yes[3]
/WORK_FILES	/WF	No	Yes	No
	/X[2]	Yes	No	No

[1]For FORTRAN IV, this switch is available only on RSX systems.

[2]For FORTRAN IV, this switch is available only on RSTS/E systems.

[3]This qualifier exhibits some minor differences, depending on which FORTRAN language implementation you are using. Refer to the appropriate qualifier section for more information.

[7]This qualifier is not synonymous with the other associated qualifiers or switches. Instead, it inhibits the behavior specified by the associated qualifiers.

The following sections discuss each qualifier or switch, which system or systems it applies to, and any inconsistencies in the various interpretations. Typically, each section refers to the DCL command qualifier but also discusses any corresponding switches. Switches that do not correspond to a particular DCL qualifier are discussed in separate sections.

2.2.2.1 /A Switch

The /A switch is valid only for FORTRAN IV on RSTS/E systems. It causes the compiler to print compilation statistics. The default behavior for the /A switch is determined when FORTRAN IV is installed on your RSTS/E system.

2.2.2.2 /ANALYSIS_DATA Qualifier

The /ANALYSIS_DATA qualifier is valid only for VAX FORTRAN. This qualifier produces a file that contains analysis data about the source code being compiled. The default is /NOANALYSIS_DATA. The source-code analysis files created when you specify this qualifier can be used with such products as the VAX Source Code Analyzer.

2.2.2.3 /CHECK Qualifier

The /CHECK qualifier is valid for FORTRAN–77 and VAX FORTRAN, but it is interpreted differently depending on which FORTRAN compiler you are using. The corresponding switch is /CK.

For FORTRAN–77, the /CHECK qualifier or /CK switch tells the compiler to generate code to check that all your array references are within the array address bounds. The compiler does not check individual subscripts against the dimension specifications.

The VAX FORTRAN /CHECK qualifier has the format /CHECK=*arg*, where *arg* can have any one of the following values:

- BOUNDS

The BOUNDS argument provides similar capabilities to the FORTRAN-77 /CHECK qualifier. If you specify /CHECK=BOUNDS, the VAX FORTRAN compiler checks array and substring references to ensure that they are within the address boundaries specified in the array or character variable declaration.

For array bounds, only the address reference is checked. The system only checks to determine whether or not you are in the same array; it does not check each individual dimension. Array bounds checking is not performed for arrays that are dummy arguments in which the last dimension bound is specified as an asterisk (*), or both upper and lower dimensions are 1.

- OVERFLOW

  The OVERFLOW argument specifies that the VAX FORTRAN compiler checks all BYTE, INTEGER*2, and INTEGER*4 calculations for arithmetic overflow. Real and complex calculations are always checked for overflow and are not affected by /NOCHECK. Integer exponentiation is performed by a routine in the mathematics library; this routine always checks for overflow, even if /CHECK=NOOVERFLOW is specified.

- UNDERFLOW

  The UNDERFLOW argument tells the VAX FORTRAN compiler to check all real and complex calculations for floating underflow.

- ALL

  The ALL argument specifies that all OVERFLOW, UNDERFLOW, and BOUNDS checks be performed by the VAX FORTRAN compiler.

- NONE

  The NONE argument specifies that no checks be performed.

The default for FORTRAN-77 is /NOCHECK or /NOCK, meaning that no checks are performed. In VAX FORTRAN, the default is /CHECK=(NOBOUNDS, OVERFLOW, NOUNDERFLOW).

2.2.2.4 /CODE:*arg* Qualifier

The /CODE:*arg* qualifier is valid only for FORTRAN IV. On RSX systems, the /CD:*arg* switch provides the same capability as the DCL /CODE qualifier. On RSTS/E systems, use the /I:*arg* switch.

These qualifiers and switches allow you to indicate that the code to be compiled includes some special characteristics. This characteristic is indicated by *arg*, which can take any one of the following values:

- EAE

  The EAE argument specifies that the code uses the Extended Arithmetic Element.

- EIS

  The EIS argument specifies that the code uses the Extended Instruction Set.

- FIS

  The FIS argument specifies that the code uses the Floating Instruction Set, which also includes the Extended Instruction Set.

- THREADED (DCL) or THR (MCR)

  The THREADED or THR argument specifies that the code is threaded.

The default DCL qualifier is /CODE:THREADED. The default switch on RSX systems is /CD:THR. On RSTS/E systems, the /I:*arg* switch defaults to the value specified during installation.

2.2.2.5 /CONTINUATIONS Qualifier

The /CONTINUATIONS qualifier is valid only for FORTRAN–77 and VAX FORTRAN, but it has a different format depending on which FORTRAN compiler you are using. The corresponding switch is /CO. This qualifier specifies the number of continuation lines allowed in a source program statement; it was initially designed for use with punch card systems. The /CONTINUATIONS qualifier has the following form:

FORTRAN–77 qualifier:	/CONTINUATIONS:*n*
FORTRAN–77 switch:	/CO:*n*
VAX FORTRAN:	{ /CONTINUATIONS:*n* } { /CONTINUATIONS = *n* }

In all cases, *n* can accept values from 0 through 99. The default is 19.

2.2.2.6 /CROSS_REFERENCE Qualifier

The /CROSS_REFERENCE qualifier applies only to VAX FORTRAN. This qualifier specifies that the storage map section of a listing file should include information about using symbolic names, including the line numbers of the lines in which the symbols are defined and referenced. The default is /NOCROSS_REFERENCE.

2.2.2.7 /DEBUG Qualifier

The /DEBUG qualifier applies only to FORTRAN–77 and VAX FORTRAN, but it has a different format depending on which FORTRAN compiler you are using. The corresponding switch is /DB. For FORTRAN–77, /DEBUG or /DB specifies that the FORTRAN–77 compiler should provide information for use by the PDP–11 Symbolic Debugger.

For VAX FORTRAN, the /DEBUG qualifier tells the compiler to provide information to the VAX Symbolic Debugger and the run-time error traceback mechanism. The VAX FORTRAN /DEBUG qualifier has the form /DEBUG=*arg*, where *arg* can have any one of the following values:

- SYMBOLS

  The SYMBOLS argument specifies that the VAX FORTRAN compiler provide the debugger with local symbol definitions for user-defined variables, arrays (including dimension information), structures, and labels of executable statements.

- TRACEBACK

  The TRACEBACK argument specifies that the VAX FORTRAN compiler provide an address correlation table so that the debugger and the run-time error traceback mechanism can translate virtual addresses into source program routine names and compiler-generated line numbers.

- ALL

  The ALL argument specifies that the VAX FORTRAN compiler provide both local symbol definitions and an address correlation table. If you do not specify an argument to the /DEBUG qualifier, /DEBUG = ALL is the default.

- NONE

  The NONE argument specifies that the VAX FORTRAN compiler not provide any debugging information. This is the same as /NODEBUG.

The VAX FORTRAN default is /DEBUG = (NOSYMBOLS, TRACEBACK). On FORTRAN-77, the default is /NODEBUG or /NODB.

NOTE

If you use the /DEBUG qualifier, it is strongly recommended that you also use the /NOOPTIMIZE qualifier. Optimizations performed by the FORTRAN compilers can cause unexpected behavior when using the appropriate symbolic debuggers.

2.2.2.8 /DI and /B Switches

The /DI and /B switches are valid only for FORTRAN IV. Use /DI for FORTRAN IV on RSX systems, and use /B for FORTRAN IV on RSTS/E systems. Both of these switches enable expanded listings of compiler internal diagnostic information. The default on RSX systems is /NODI. For FORTRAN IV on RSTS/E systems, the default is determined when FORTRAN IV is installed.

2.2.2.9 /D_LINES Qualifier

The /D_LINES qualifier is valid only for VAX FORTRAN, but the /DLINES qualifier provides the same capability for FORTRAN-77. This qualifier specifies that lines with a D in column 1 are compiled and not treated as comment lines. The default is /NOD_LINES, which indicates that lines with a D in column 1 are treated as comments.

2.2.2.10 /DLINES Qualifier

The /DLINES qualifier is valid only for FORTRAN-77, but you can achieve the same behavior for FORTRAN IV and FORTRAN-77 by using switches. (To achieve the same behavior using VAX FORTRAN, use the /D_LINES qualifier.) For FORTRAN IV on RSX systems and FORTRAN-77, the switch corresponding to the /DLINES qualifier is /DE. For FORTRAN IV on RSTS/E systems, the corresponding switch is /D.

All of these command qualifiers and switches specify that lines with a D in column 1 be compiled and not treated as comment lines. The default is /NODLINES, /NODE, or /NOD, all of which indicate that lines with a D in column 1 are treated as comments.

2.2.2.11 /DML Qualifier

The /DML qualifier is valid only on VAX FORTRAN. It specifies that the FORTRAN Data Manipulation Language (DML) preprocessor is to be invoked before the compiler. The preprocessor produces an intermediate file of FORTRAN source code in which FORTRAN DML commands are expanded into FORTRAN statements. The compiler is then automatically invoked to compile this intermediate file. You can use the VAX FORTRAN /SHOW=PREPROCESSOR qualifier along with the /DML qualifier to cause the preprocessor-generated source code to be included in the listing file. For more information on the /SHOW qualifier, see Section 2.2.2.32. The default is /NODML.

2.2.2.12 /EXTEND Qualifier

The /EXTEND qualifier applies only to the FORTRAN–77 compiler. However, the /EXTEND_SOURCE qualifier provides the same capability for VAX FORTRAN, and the /EX and /E switches provide the same capability for FORTRAN IV and FORTRAN–77. For FORTRAN IV on RSX systems and FORTRAN–77, the corresponding switch is /EX. For FORTRAN IV on RSTS/E systems, the corresponding switch is /E.

All of these command qualifiers and switches let you specify that the range of your source text be extended from columns 1 through 72 to columns 1 through 132. If a line is longer than 132 characters, the FORTRAN IV and FORTRAN–77 compilers signal a fatal read error and your compilation is immediately terminated. The default is /NOEXTEND, /NOEX, or /NOE.

2.2.2.13 /EXTEND_SOURCE Qualifier

The /EXTEND_SOURCE qualifier applies only to the VAX FORTRAN compiler, but it performs the same function as the FORTRAN–77 /EXTEND qualifier and the /EX and /E switches. The /EXTEND_SOURCE qualifier lets you specify that the range of your VAX FORTRAN source text be extended from columns 1 through 72 to columns 1 through 132. If a line is longer than 132 characters, the VAX FORTRAN compiler signals a fatal read error and your compilation is immediately terminated. The default is /NOEXTEND_SOURCE.

2.2.2.14 /FOR Qualifier

Use the /FOR qualifier at the PDP–11 DCL command line to invoke the FORTRAN IV compiler. On RSX systems, the default file type for a FORTRAN IV program is FTN. On RSTS/E systems, the default file type is FOR.

2.2.2.15 /F77 Qualifier

The /F77 qualifier is valid only for FORTRAN–77 and VAX FORTRAN. The corresponding switch is /F77. This qualifier specifies that the compiler use the FORTRAN–77 interpretation rules for those statements with a meaning that is incompatible with FORTRAN IV-PLUS. If you specify the /NOF77 qualifier, the compiler selects FORTRAN IV-PLUS interpretations in cases of incompatibility. The default is /F77.

2.2.2.16 /G_FLOATING Qualifier

The /G_FLOATING qualifier is valid only for the VAX FORTRAN compiler. It lets you control how the VAX FORTRAN compiler implements the following data types:

- REAL*8
- COMPLEX*16
- DOUBLE PRECISION
- DOUBLE COMPLEX

The default for this qualifier is /NOG_FLOATING, which causes the compiler to implement double-precision quantities using the VAX D_floating data type. If you specify /G_FLOATING, the compiler implements such quantities using the VAX G_floating data type.

2.2.2.17 /IDENTIFICATION Qualifier

The /IDENTIFICATION qualifier is valid only for FORTRAN–77, but the corresponding switch /ID is also valid for FORTRAN IV on RSX systems. This qualifier tells the compiler to print its identification and version number on the user's terminal. The default is /NOIDENTIFICATION or /NOID.

2.2.2.18 /I4 Qualifier

The /I4 qualifier is valid on FORTRAN IV, FORTRAN–77, and VAX FORTRAN. There are also corresponding switches: use /I4 for FORTRAN–77 and for FORTRAN IV on RSX systems; use /T for FORTRAN IV on RSTS/E systems.

All of these qualifiers and switches specify that the default allocation for integer variables be 2 words (4 bytes). This includes INTEGER and LOGICAL data types.

For FORTRAN IV and FORTRAN–77, the default is /NOI4 or /NOT, which means that integer variables are interpreted as INTEGER*2 and LOGICAL*2. VAX FORTRAN defaults to /I4, meaning that integer variables are interpreted as INTEGER*4 and LOGICAL*4.

2.2.2.19 /LA Switch

The /LA switch is valid only for FORTRAN–77 in MCR interactive mode. This switch causes the current switch settings to be retained (latched) for subsequent compilations in MCR interactive mode. The default is /NOLA.

2.2.2.20 /LIBRARY Qualifier

The /LIBRARY qualifier is valid only for VAX FORTRAN. This qualifier specifies that a file is a text library file. You can specify one or more text library files in a list of files that are concatenated by using plus signs. However, at least one of the files in the list must be a nonlibrary file. The default file type for a VMS library file is TLB.

2.2.2.21 /LINE_NUMBERS Qualifier

The /LINE_NUMBERS qualifier is valid only on FORTRAN IV. The corresponding switch is /SN for FORTRAN IV on RSX systems. The /LINE_NUMBERS qualifier and /SN switch tell the FORTRAN IV compiler to include Internal Sequence Numbers in your object code.

FORTRAN IV on RSTS/E systems does not supply a switch to specify this default behavior. The /S switch is actually the equivalent of /NOLINE_NUMBERS or /NOSH, which causes the FORTRAN IV compiler on RSTS/E systems to suppress the generation of Internal Sequence Numbers.

Internal Sequence Numbers may take up space in the object code and may reduce program execution speed, but they are useful in determining which line caused a run-time error when you debug your program. /LINE_NUMBERS is the default DCL qualifier for FORTRAN IV. /SN is the default switch for FORTRAN IV on RSX systems. The default on RSTS/E systems is to generate Internal Sequence Numbers; you cannot specify this behavior using a switch.

2.2.2.22 /LIST Qualifier

The /LIST qualifier is valid for FORTRAN IV, FORTRAN–77, and VAX FORTRAN, but there are some minor differences between the PRO and PDP–11 FORTRAN interpretation and the VAX FORTRAN interpretation of this qualifier. The /LIST qualifier generally tells any one of the FORTRAN compilers to generate a listing file.

There is no corresponding switch for either FORTRAN IV or FORTRAN–77, because the effects of the /LIST qualifier can be accomplished by specifying a file specification in the list-file field of the appropriate command line. By including a listing file specification, you indicate that the compiler should generate a listing file. See Section 2.2.1 for more information on the appropriate command lines for FORTRAN IV and FORTRAN–77.

The /LI and /L switches let you indicate what information to include in the listing file. For FORTRAN IV on RSX systems and FORTRAN–77, the switch is /LI:*n*. For FORTRAN IV on RSTS/E systems, the switch is /L:*n*. For either switch, *n* can take the following values:

0 or null:	List diagnostics only
1 or SRC:	List source program and diagnostics only
2 or MAP:	List storage map and diagnostics only
4 or COD:	List generated code and diagnostics only

You can specify any combination of the above list of options by adding together the numeric argument values to compute the desired list options. For example, specifying 7 or ALL requests a source listing, a storage map, and a generated code listing. The default for both switches (/LI:*n* and /L:*n*) is $n = 3$; that is, /LI:3 and /L:3. If you specify a listing file name in the command line for FORTRAN IV or FORTRAN–77 and you do not specify either the /LI or /L switch, by default the listing file will include a listing of the source program and storage map.

NOTE

For the PRO and PDP–11 FORTRAN compilers, the default file type of a listing file is LST. For VAX FORTRAN, the default file type of a listing file is LIS.

When using the DCL /LIST qualifier, you can specify it as either a command qualifier or as a file specification qualifier on PRO and PDP–11 systems. That is, for the PRO and PDP–11 FORTRAN compilers, /LIST is a positional qualifier. The command qualifier and file specification qualifier have the following form:

Command qualifier:	FORTRAN/LIST file-name.FTN
File specification qualifier:	FORTRAN file-name.FTN/LIST

If you specify /LIST as a command qualifier, the PRO and PDP–11 FORTRAN compilers print the listing file and place a copy in your UFD. If you specify /LIST as a file specification qualifier, the PDP–11 FORTRAN compiler places the file in your directory but does not automatically print the file.

When using the VAX DCL /LIST qualifier, the position of the qualifier in the command line does not have any effect on the printing of the file. If you want to print the file, you must do so explicitly. For VAX FORTRAN, the default is /NOLIST for an interactive compilation and /LIST for a compilation performed in batch mode.

2.2.2.23 /LO and /Q Switches

The /LO and /Q switches are valid only for FORTRAN IV. The /LO switch tells the FORTRAN IV compiler on RSX systems to print the names of program units (from PROGRAM, FUNCTION, SUBROUTINE, and BLOCK DATA statements) on your terminal as they are compiled. The /Q switch inhibits the printing of this information for the FORTRAN IV compiler on RSTS/E systems; that is, the /Q switch is actually the equivalent of /NOLO. The default on both systems is to print this information.

2.2.2.24 /MACHINE_CODE Qualifier

The /MACHINE_CODE qualifier is valid for FORTRAN IV, FORTRAN-77, and VAX FORTRAN, but there are some minor differences. FORTRAN IV and FORTRAN-77 both provide corresponding switches. For FORTRAN IV on RSX systems and FORTRAN-77, specify a file specification for the listing file field and use the /LI:3 switch. For FORTRAN IV on RSTS/E systems, specify a listing file and use the /L:3 switch. (For more information on the /LI and /L switches, see Section 2.2.2.22.)

You can use any one of the following methods to specify that the PDP-11 FORTRAN compiler listing include binary machine code and diagnostics:

- Specify the DCL /MACHINE_CODE qualifier.
- Specify the /LI:3 switch to the listing file specification in the command line for either FORTRAN-77 or FORTRAN IV on RSX systems.
- Specify the /L:3 switch to the listing file specification in the command line for FORTRAN IV on RSTS/E systems.

The DCL /MACHINE_CODE qualifier is a positional qualifier for PRO and PDP-11 FORTRAN, which is similar to the /LIST positional qualifier. For more information on positional qualifiers, see Section 2.2.2.22.

The /MACHINE_CODE qualifier for the VAX FORTRAN compiler does not imply /LIST. If you do not request a listing file using /LIST, the VAX /MACHINE_CODE qualifier is ignored. This qualifier tells the VAX FORTRAN compiler to include a symbolic representation of the object code generated by the compiler. The format of this symbolic representation is very similar to a VAX MACRO assembly listing. Do not try to compile this listing since it includes items that are not supported by VAX MACRO. The default is /NOMACHINE_CODE.

2.2.2.25 /MAP Qualifier

The /MAP qualifier is only available through the PRO and PDP-11 FORTRAN compilers. The /SHOW=MAP qualifier provides the same capability for VAX FORTRAN. Section 2.2.2.32 describes the /SHOW qualifier. Using a different command line, you can achieve the identical behavior by including a file specification for the listing-file field of the appropriate command line and specifying either /LI:2 for FORTRAN IV on RSX systems and FORTRAN-77, or /L:2 for FORTRAN IV on RSTS/E systems.

You can use any one of the following methods to specify that the compiler listing include a storage map and diagnostics:

- Specify the DCL /MAP qualifier.
- Specify the /LI:2 switch to the listing file specification in the command line for either FORTRAN IV on RSX systems or FORTRAN-77.

- Specify the /L:2 switch to the listing file specification in the command line for FORTRAN IV on RSTS/E systems.

Like the DCL /MACHINE_CODE qualifier, /MAP implies the DCL /LIST qualifier on the PRO and PDP-11 FORTRAN compilers, and is a positional qualifier. For more information, see Section 2.2.2.22.

2.2.2.26 /N Switch

The /N switch is valid only for FORTRAN IV on RSTS/E systems. It specifies the maximum number of simultaneously open I/O channels allowed at run time. The format of this switch is /N:*n*, where $1 < n < 15$. The default behavior is specified when FORTRAN IV is installed on your RSTS/E system.

2.2.2.27 /O Switch

The /O switch is valid only for FORTRAN IV on RSTS/E systems. It specifies that the compiler print an "Options-In-Effect" section before the listing. The default behavior is specified when FORTRAN IV is installed on your RSTS/E system.

2.2.2.28 /OBJECT Qualifier

The /OBJECT qualifier is valid only for VAX FORTRAN, but you can request an object file using the appropriate FORTRAN-77 or FORTRAN IV command line. You use the /OBJECT qualifier, or the appropriate field in the command line, to specify a name for the object file generated by the compiler. The VAX FORTRAN compiler specifies the /OBJECT qualifier by default so that an object file is generated. If you want to suppress the object file, specify the /NOOBJECT qualifier.

An object file is not automatically generated by the PRO and PDP-11 compilers. To request that the FORTRAN IV or FORTRAN-77 compiler generate an object file, you must include a file specification in the object-file field of the appropriate command line. For example:

```
FOR> objectfile.OBJ = inputfile.FTN
F77> objectfile.OBJ = inputfile.FTN
```

In this example, the object file to be generated is named OBJECTFILE.OBJ.

2.2.2.29 /OPTIMIZE Qualifier

The /OPTIMIZE qualifier is valid only for FORTRAN-77 and VAX FORTRAN. The corresponding switch is /OP. The /OPTIMIZE qualifier and /OP switch let you specify that the compiler is to produce optimized code; /OPTIMIZE or /OP is the default. If you invoke the VAX or PDP-11 Symbolic Debugger, you can specify /NOOPTIMIZE or /NOOP to ensure that the debugger has sufficient information to locate errors in the source program.

2.2.2.30 /R Switch

The /R switch is valid only for FORTRAN IV on RSTS/E systems. This qualifier specifies the maximum record length (in bytes) on run-time I/O. This switch has the format /R:*n*, where *n* has the range $4 < n < 4095$. The default behavior is specified when FORTRAN IV is installed on your RSTS/E system.

2.2.2.31 /SHAREABLE Qualifier

The /SHAREABLE qualifier is available to the PRO and PDP–11 FORTRAN compilers. The corresponding switch for FORTRAN–77 and FORTRAN IV on RSX systems is /RO. For FORTRAN IV on RSTS/E systems, the corresponding switch is /Z.

These command qualifiers and switches tell the compiler to generate pure code and pure data sections as read-only. This might be useful if you want to take advantage of code sharing in multiuser tasks on RSX–11M–PLUS systems. These qualifiers specify whether to place the data in the $IDATA or $PDATA program section, so they can be used on RSTS/E systems as well. The default is /NOSHAREABLE, /NORO, or /NOZ.

2.2.2.32 /SHOW Qualifier

The /SHOW qualifier is valid only for VAX FORTRAN. The /SHOW qualifier controls whether or not optionally listed source lines (that is, text module source lines and preprocessor-generated source lines) and a symbol map appear in the source listing.

You must specify the VAX FORTRAN /LIST qualifier in order for the /SHOW qualifier to take effect. The /SHOW qualifier has the format /SHOW=*arg*, where *arg* can have any one of the following values:

- DICTIONARY

  The DICTIONARY argument specifies that FORTRAN source representations of any CDD records referenced by DICTIONARY statements be included in the listing file.

- INCLUDE

  The INCLUDE argument specifies that the source lines from any files specified by INCLUDE statements be included in the source listing.

- MAP

  The MAP argument specifies that the symbol map be included in the listing file. If the /CROSS_REFERENCE qualifier is specified, MAP is ignored. This argument provides the same feature for VAX FORTRAN as the /MAP qualifier does for the PRO and PDP–11 FORTRAN compilers.

- PREPROCESSOR

  The PREPROCESSOR argument specifies that preprocessor-generated source lines be included in the listing file. The negative form, NOPREPROCESSOR, specifies that the source lines be excluded from the source listing.

- SINGLE

  The SINGLE argument specifies that the symbolic names of parameter constants be included in cross-reference listings, even if they are not referenced outside the PARAMETER statements in which they are declared. The negative form, NOSINGLE, specifies that names of parameter constants be suppressed if they are only declared and not referenced elsewhere. This is useful for cross-reference listings of small programs that specify INCLUDE declarations but use only a small number of the parameter constant names that have been declared.

- ALL

  The ALL argument specifies that all optionally listed source lines be included in the listing file.

- NONE

  The NONE argument specifies that no optionally listed source lines be included in the listing file.

The default arguments for the /SHOW qualifier are NODICTIONARY, NOINCLUDE, MAP, NOPREPROCESSOR, and SINGLE. Specifying the /SHOW qualifier without any arguments is equivalent to /SHOW=ALL. /NOSHOW without any arguments is equivalent to specifying /SHOW=NONE.

2.2.2.33 /SOURCE Qualifier

The /SOURCE qualifier is valid only on the PRO and PDP-11 FORTRAN compilers. Using a different command line, you can achieve the same behavior by including a file specification for the listing-file field of the appropriate command line and specifying either /LI:2 for FORTRAN IV on RSX systems and FORTRAN-77, or /L:2 for FORTRAN IV on RSTS/E systems.

You can use any one of the following methods to specify that the compiler listing include source code:

- Specify the DCL /MAP qualifier.
- Specify the /LI:2 switch to the listing file specification in the command line for either FORTRAN-77 or FORTRAN IV on RSX systems.
- Specify the /L:2 switch to the listing file specification in the command line for FORTRAN IV on RSTS/E systems.

Both the DCL qualifier and the corresponding switches are very similar to the /MAP qualifier and its corresponding switches. For more information, see Section 2.2.2.25.

2.2.2.34 /SP Switch

The /SP switch is valid only for FORTRAN-77 and FORTRAN IV on RSX systems. This switch tells the compiler to automatically spool the listing file. The default is /SP.

2.2.2.35 /STANDARD Qualifier

The /STANDARD qualifier is valid only for FORTRAN-77 and VAX FORTRAN because the purpose of this qualifier is to tell the compiler to flag any extensions to the ANSI X3.9-1978 standard. The corresponding switch is /ST. These qualifiers have the following formats:

Qualifier:	/STANDARD=*arg*
Switch:	/ST:*arg*

In either case, *arg* can have one of the following values:

- SEMANTIC (VAX FORTRAN only)

  The SEMANTIC argument specifies that an informational message be issued for syntax extensions to the current ANSI standard — both for extensions that occur in individual statements and for extensions that occur as a result of usage across statements in a program unit. For example, an assignment statement that specifies a record field is not flagged as an extension if SYNTAX checking is in effect. It is flagged as an extension if SEMANTIC checking was in effect.
- SOURCE (FORTRAN-77) or SOURCE_FORM (VAX FORTRAN)

  Either the SOURCE (FORTRAN-77) or SOURCE_FORM (VAX FORTRAN) argument specifies that an informational message be issued for statements using tab formatting or containing lowercase characters.

- SYNTAX

  The SYNTAX argument specifies that an informational message be issued for syntax extensions to the current ANSI standard.

- ALL

  The ALL argument specifies that informational messages be issued for both the SYNTAX and SOURCE form extensions to the current ANSI standard.

- NONE

  The NONE argument specifies that no informational messages be issued for extensions to the current ANSI standard. The /STANDARD=NONE qualifier is invalid on RSTS/E systems, so use /NOSTANDARD.

The default is /NOSTANDARD or /NOST, which is equivalent to /STANDARD=NONE. If you specify the /NOWARNINGS qualifier, the /STANDARD qualifier is ignored. Specifying /STANDARD without any arguments is equivalent to specifying /STANDARD=(SYNTAX, NOSOURCE) for FORTRAN-77 or /STANDARD=(SYNTAX, NOSOURCE_FORM) for VAX FORTRAN. Specifying the /ST switch without any arguments is equivalent to specifying /ST:SYNTAX.

2.2.2.36 /TRACEBACK:*arg* Qualifier

The /TRACEBACK:*arg* qualifier is valid only for FORTRAN-77. The corresponding switch is /TR:*arg*. The /TRACEBACK qualifier and /TR switch control the amount of extra code to be included in the compiled output for use by the OTS during error traceback. This code is used to produce diagnostic information and to identify faulty statements.

The /TRACEBACK qualifier and /TR switch accept any one of the following values for *arg*:

- BLOCKS

  The BLOCKS argument specifies that traceback information be compiled for all subroutine and function entries and initial statements in sequences called blocks.

- LINES

  The LINES argument is equivalent to the ALL argument.

- NAMES

  The NAMES argument specifies that traceback information be compiled only for subroutine and function entries.

- ALL

  The ALL argument specifies that error traceback information be compiled for all source statements and function and subroutine entries.

- NONE

  The NONE argument specifies that no traceback information be produced.

The default is /TRACEBACK:BLOCKS or /TR:BLOCKS. If you do not specify an argument to /TRACEBACK or /TR, the default is TRACEBACK:ALL or /TR:ALL.

2.2.2.37 /U Switch

The /U switch is valid only for FORTRAN IV on RSTS/E systems. It disables USR swapping at run time. The default behavior is specified when FORTRAN IV is installed on your RSTS/E system.

2.2.2.38 /VECTORS Qualifier

The /VECTORS qualifier is valid only for the FORTRAN IV compiler running on an RSX system. The corresponding switch is /VA for FORTRAN IV on RSX systems. These command qualifiers tell the compiler to "vectorize" arrays. This increases the size of the object program but decreases the overall execution time for your program. /VECTORS or /VA is the default behavior.

FORTRAN IV on RSTS/E systems does not supply a switch to specify this default behavior. The /V switch is the equivalent of /NOVECTORS or /NOVA, which causes the FORTRAN IV compiler on RSTS/E systems to suppress the vectorization of arrays.

2.2.2.39 /WARNINGS Qualifier

The /WARNINGS qualifier is valid for FORTRAN IV, FORTRAN-77, and VAX FORTRAN, but there are some differences between the qualifiers for the various compilers. The corresponding switch is /WR for FORTRAN IV on RSX systems and FORTRAN-77. For FORTRAN IV on RSTS/E systems, the corresponding switch is /W.

For the PRO and PDP-11 FORTRAN compilers, the /WARNINGS qualifier tells the compiler to issue warning diagnostics. /WARNINGS is the default.

For VAX FORTRAN, the /WARNINGS qualifier specifies that the compiler generate informational (I) and warning (W) diagnostic messages in response to informational and warning-level errors. The VAX FORTRAN /WARNINGS qualifier has the format /WARNINGS=*arg*, where *arg* can be any one of the following:

- DECLARATIONS

  The DECLARATIONS argument causes the compiler to print warnings for any undeclared data items used in the program. DECLARATIONS acts as an external VAX FORTRAN IMPLICIT NONE declaration. The default is NODECLARATIONS.

- GENERAL

  The GENERAL argument causes the compiler to generate informational and warning diagnostic messages. An informational message indicates that a correct VAX FORTRAN statement may have unexpected results, or that it contains a nonstandard syntax or source form. A warning message indicates that the compiler has detected acceptable but nonstandard syntax or has performed some corrective action. In either case, a warning message indicates that unexpected results may occur. To suppress I and W diagnostic messages, specify the negative form of this qualifier (NOGENERAL). The default is GENERAL.

- ULTRIX (VAX FORTRAN only)

  The ULTRIX argument causes the compiler to issue diagnostics for language features not supported by VAX FORTRAN on ULTRIX systems. Using this option, you can develop VAX FORTRAN programs on a VMS system and transport those programs to an ULTRIX system. The default is NOULTRIX.

- VAXELN (VAX FORTRAN only)

  The VAXELN argument causes the compiler to issue diagnostic messages for language features not supported by VAX FORTRAN on a VAXELN system. The default is NOVAXELN.

- ALL

  The ALL argument causes the compiler to print all informational and warning messages, including warning messages for any undeclared data items.
- NONE

  The NONE argument suppresses all informational and warning messages.

For VAX FORTRAN, the default is WARNINGS = (NODECLARATIONS, GENERAL, NOULTRIX, NOVAXELN).

2.2.2.40 /WORK_FILES:*n* Qualifier

The /WORK_FILES:*n* qualifier is valid only for FORTRAN-77. The corresponding switch is /WF:*n*. The /WORK_FILES:*n* qualifier and /WF:*n* switch specify the number of temporary on-disk files to be used during compilation. You can assign *n* a value between 1 and 3; the default is 2. If you increase *n*, this increases the maximum possible size of your program, but decreases the overall time of compilation.

2.2.2.41 /X Switch

The /X switch is valid only for FORTRAN IV on RSTS/E systems. It indicates cross-compilation for the target environment. This switch has the format /X:*arg*, where *arg* can have one of the following values:

RT:	Selects RT-11
RST:	Selects RSTS/E
RSX:	Selects RSX-11

The default behavior is determined when FORTRAN IV is installed on your RSTS/E system.

2.2.3 Compiler Control Statements

The FORTRAN-77 and VAX FORTRAN compilers provide the following compiler control statements (FORTRAN IV does not support any of these statements):

- INCLUDE *'file-spec'* or
  INCLUDE *'[file-spec](module-name)'*

  The INCLUDE statement is valid for both FORTRAN-77 and VAX FORTRAN. It specifies that the contents of a file or a text library module be incorporated into the FORTRAN compilation directly following the INCLUDE statement. The *file-spec* argument is a character constant that specifies the file to be included. The *module-name* argument is the name of a text module located in a text library.
- DICTIONARY *'cdd-path'*

  The DICTIONARY statement is valid only for VAX FORTRAN. This statement incorporates VAX Common Data Dictionary (CDD) data definitions into the current FORTRAN source file during compilation. The *cdd-path* argument is the full or relative pathname of a CDD object.

- OPTIONS *qualifier*

  The OPTIONS statement is valid only for VAX FORTRAN. You can use this statement to override or confirm the FORTRAN command qualifiers in effect for a particular program unit. The OPTIONS statement can take any one of the following arguments:

  - /CHECK = *arg*
  - /EXTEND_SOURCE
  - /F77
  - /G_FLOATING
  - /I4

  All of these arguments to the OPTIONS statement have corresponding FORTRAN command qualifiers.

2.2.4 Syntax and Format

Because VAX FORTRAN is an extension of PRO and PDP–11 FORTRAN, most of the language syntax and format is identical. Coexistence between PRO, PDP–11, and VAX systems, or migration from one to another, is made easier by the many similarities between the various FORTRAN compilers. However, because one is an extension of the other, there are some differences.

Differences between the various FORTRAN implementations involve the following areas:

- Logical tests
- Floating-point results
- Character and Hollerith constants
- Logical unit numbers
- Assigned GO TO label list
- Effect of the DISPOSE = 'PRINT' specification

FORTRAN–77 and VAX FORTRAN are both based on the most recent American National Standard FORTRAN–77 (X3.9-1978). FORTRAN IV is based on the previous standard, X3.9-1966. Incompatibilities between the two standards exist in the following areas:

- The minimum iteration count of DO loops
- The EXTERNAL statement
- The defaults for the OPEN statement's BLANK and STATUS keywords
- The X format edit descriptor
- The effect of attempting to open a connected unit

These differences are described in the following sections.

2.2.5 Statements

Despite the similarity of the various FORTRAN compilers, some differences do exist between the statements available in each language, or the way these statements are handled by the compiler. The following sections describe these differences.

2.2.5.1 Assigned GO TO Label Lists

The labels that you specify in an assigned GO TO label list are checked by the various FORTRAN compilers to ensure their validity in the program unit that you are compiling. However, both VAX FORTRAN and FORTRAN IV do not perform a check at run time to ensure that a label that is actually assigned is in the list. FORTRAN-77 does perform this check. For example:

```
        ASSIGN 400 TO MEGO
        GO TO MEGO, (150, 200, 350)
500     TYPE *,'Statement not found in Assigned GO TO'
        .
        .
        .
400     I = 1
```

In this example, the ASSIGN statement assigns a value of 400 to MEGO. This variable is then used to indicate a statement label for the assigned GO TO statement; however, the label 400 is not included in the label list provided in the assigned GO TO statement. If a statement labeled 400 does exist somewhere in the program unit, the inconsistency between the various compilers is exhibited as shown in the following table:

Language	Action
FORTRAN IV	Executes the ASSIGN and GO TO statements and continues execution at statement 400.
FORTRAN-77	Executes the ASSIGN statement and checks to see if label 400 is in the label list provided in the GO TO statement. Because it is not, execution continues at the next statement, in this case, statement 500.
VAX FORTRAN	Executes the ASSIGN and GO TO statements and continues execution at statement 400.

2.2.5.2 DO Loop Minimum Iteration Count

In FORTRAN-77 and VAX FORTRAN, the body of a DO loop is not executed if the end condition of the loop is satisfied when the DO statement is executed. In FORTRAN IV, however, the body of a DO loop is always executed at least once. For example:

```
        I = 1
        DO 10 J = 1,I
           .
           .
           .
10      CONTINUE
```

FORTRAN IV performs a single iteration of this loop, even though the loop condition (J = I) is satisfied. If you compile this loop using FORTRAN-77 or VAX FORTRAN, the loop is not executed because the condition is already satisfied. If you specify the /NOF77 qualifier when compiling the FORTRAN-77 or VAX FORTRAN program, the execution of this loop matches the FORTRAN IV behavior.

2.2.5.3 EXTERNAL Statement

If you specify a function using FORTRAN IV with the same name as a FORTRAN intrinsic function or library function in an EXTERNAL statement, that function name is assumed to refer to the named intrinsic or library function and not a user-defined function. If you want to use the FORTRAN IV EXTERNAL statement to specify a user-defined function, you must precede the function name with an asterisk.

In the following example, the first EXTERNAL statement, if compiled under FORTRAN IV, indicates the FORTRAN intrinsic SIN function. However, since SIN is preceded by an asterisk in the second EXTERNAL statement, FORTRAN IV assumes that this refers to a user-defined function named SIN.

```
EXTERNAL SIN
EXTERNAL *SIN
```

If you specify a function using FORTRAN-77 or VAX FORTRAN with the same name as a FORTRAN intrinsic function or library function in an EXTERNAL statement, that function name is always assumed to refer to a user-defined function. If you precede a function name with an asterisk using FORTRAN-77 or VAX FORTRAN, the asterisk has no meaning and is marked as an error. However, if you specify the /NOF77 switch when compiling your FORTRAN-77 or VAX FORTRAN program, the function name in the EXTERNAL statement is interpreted as it would be under FORTRAN IV, and the asterisk is valid.

To use the name of an intrinsic function as an argument to a subprogram in FORTRAN-77 or VAX FORTRAN, you must use the INTRINSIC statement. This statement provides the same capabilities for FORTRAN-77 and VAX FORTRAN that the EXTERNAL statement (without an asterisk) provides for FORTRAN IV.

2.2.5.4 Blank Common Program Section (.$$$$.)

Under FORTRAN-77, the blank common program section (.$$$$.) has the SAV attribute; it does not have this attribute under FORTRAN IV. The SAV attribute on a program section lets you pull that program section into the root segment of an overlay.

In FORTRAN-77, the /F77 compiler switch lets you control the default assignment of the SAV attribute; under /F77, the blank common program section is assigned the SAV attribute by default.

2.2.5.5 X Format Edit Descriptor

The *n*X edit descriptor lets you specify that the transmission of the next character to or from a record should occur at the position *n* characters to the right of the current position. In a FORTRAN-77 or VAX FORTRAN output statement, the character positions that are skipped are not modified, and the length of the output record is not affected. However, in a FORTRAN IV statement, the X edit descriptor writes blanks and may extend the output record.

In the following example, the FORMAT statement specifies that transmission of the character string 'ABCDEF' begin a single space to the right of the current (first) position. The T4 indicates that the current position be set to the fourth character position. At this location, FORTRAN-77 and VAX FORTRAN skip two positions. FORTRAN IV writes two blanks and then the string '12345' followed by three blanks. The different interpretations of the X edit descriptor are illustrated in the output of the following statements:

```
      WRITE (1,10)
10    FORMAT(1X, 'ABCDEF', T4, 2X, '12345', 3X)
```

These statements produce the following results:

```
FORTRAN-77 and VAX FORTRAN          FORTRAN IV
    #ABCD12345                      #AB##12345###
```

The /NOF77 switch does not affect the interpretation of the X edit descriptor. If you want to simulate the FORTRAN IV method of interpreting the X edit descriptor, change *n*X to *n*(' ').

2.2.6 Subroutines

FORTRAN-77 supplies a number of utility subroutines. These subroutines are described in the *PDP-11 FORTRAN-77 User's Guide*.

Six of these subroutines are supplied as a standard part of VAX FORTRAN. Table 2–3 describes these FORTRAN subroutines and the operations they perform.

Table 2–3: FORTRAN Subroutines

Subroutine	Operation Performed
DATE	Returns a 9-byte string containing the ASCII representation of the current date.
ERRSNS	Returns information about the most recently detected error condition. If the error was an I/O error, the additional arguments receive the following information: • The primary file system error code: FCS-11 F.ERR value or RMS–11 STS value • The secondary file system error code: FCS-11 F.ERR+1 value or RMS–11 STV value • The logical unit number
EXIT	Terminates the execution of a program and returns control to the operating system. If you do not specify the exit status parameter, the terminator status is success.
IDATE	Returns three integer values representing the current month, day, and year.
SECNDS	Provides the system time of day, or elapsed time, as a floating-point value in seconds. The value returned by SECNDS is accurate to the resolution of the system clock: 0.0166... seconds for a 60-cycle (60 Hertz) clock, and 0.02 seconds for a 50-cycle (50 Hertz) clock.
TIME	Returns an 8-byte string containing the ASCII representation of the current time in hours, minutes, and seconds.

VAX FORTRAN supports the following subroutines and functions for the purposes of maintaining compatibility with PDP–11 FORTRAN. These subroutines and functions are:

ASSIGN	CLOSE
ERRSET	ERRTST
FDBSET	IRAD50
RAD50	RAN
RANDU	R50ASC
USEREX	

Most of these subroutines have been superseded by features included in VAX FORTRAN, so if you are migrating to VAX FORTRAN you may want to use the new VAX FORTRAN capabilities instead of these subroutines. However, if you are planning coexistence between PRO, VAX, and PDP–11 systems, continue to use these subroutines to ensure transportability of your programs.

2.2.6.1 ASSIGN Subroutine

The ASSIGN subroutine lets you supply device or file name information to a logical unit. The assignment you make remains in effect until your program terminates, or until you close the logical unit using a CLOSE statement or the CLOSE subroutine. Since the ASSIGN subroutine assigns device or file information to a logical unit, you must call this subroutine before issuing the first I/O statement for that logical unit.

If you use the ASSIGN subroutine for a particular logical unit, you can also use the FDBSET subroutine and the DEFINE FILE statement. However, you cannot use the OPEN or INQUIRE statements for that same logical unit. There are two other ways that you can assign a device or a file name to a logical unit number: specify the FILE keyword in an OPEN statement, or use the ASSIGN system command.

A call to the ASSIGN subroutine has the following format:

```
CALL ASSIGN (n [,name] [,icnt])
```

The arguments for this subroutine have the following definitions:

n
is an integer value specifying the logical unit number. If you only specify the logical unit number in your CALL statement, you nullify any previous associations pertaining to that unit, and the file/device association takes on the default value. That is, by only passing the **n** parameter, you reset any previous values of **name** and **icnt**.

name
is a variable, array, array element, or character constant containing any standard file specification.

icnt
is an INTEGER*2 value that specifies the number of characters contained in the string name. If you omit the **icnt** argument or specify it as zero, any file specification is read until the first ASCII null character is encountered. If you specify the **icnt** argument, you must specify the **name** argument.

2.2.6.2 CLOSE Subroutine

The CLOSE subroutine lets you close the file currently open on a particular logical unit. A call to the CLOSE subroutine has the following format:

```
CALL CLOSE(n)
```

The definition of the argument **n** is as follows:

n
is an integer value specifying the logical unit.

After you close the file, the logical unit again assumes the default file name specification.

2.2.6.3 ERRSET Subroutine

The ERRSET subroutine lets you determine what action to take in response to a particular error detected by the Run-Time Library. This error action is independent of other errors, so if you specify certain behavior for a particular error, that behavior is not followed if some other error occurs.

The VAX condition-handling facility provides a more general method for defining actions to be taken when errors are detected. If you are planning to migrate your applications to VAX systems, refer to the *VMS Run-Time Library Routines Reference Manual* for more information on condition handlers.

A call to the ERRSET subroutine has the following format:

```
CALL ERRSET (number [,contin ,count ,type ,log ,maxlim])
```

In this format, the arguments have the following definitions:

number
is an integer value specifying the error number. You cannot specify a null argument for **number**, but null arguments are legal for all the other parameters to ERRSET.

contin
is a logical value that determines whether or not to continue executing your program after an error is detected.

.TRUE.	Continue after error is detected.
.FALSE.	Exit after error is detected.

count
is a logical value that determines whether or not the error is counted against your maximum error limit.

.TRUE.	Count the error against the maximum error limit.
.FALSE.	Do not count the error against the maximum error limit.

type
is a logical value that determines where control is passed after an error is detected.

.TRUE.	Pass control to an ERR transfer label, if one is specified.
.FALSE.	Return control to the routine that detected the error for default error recovery.

log
is a logical value that determines whether or not to produce an error message for the detected error.

.TRUE.	Produce an error message for this error.
.FALSE.	Do not produce an error message for this error.

maxlim
is a positive INTEGER*2 value that specifies the maximum error limit. The default is set to 15 at the time of program initialization.

If you specify a null argument for any other parameter, it has no effect on the current state of that argument. A null argument does not reset the previous value of the parameter.

NOTE

An external reference to ERRSET or ERRTST causes a special PDP–11 FORTRAN compatibility error handler to be established before the main program is called. This special error handler transforms the executing environment to resemble that of PDP–11 FORTRAN.

2.2.6.4 ERRTST Subroutine

The ERRTST subroutine lets you check for a specific error. You can use ERRTST to determine what error has occurred and take appropriate actions in response. A call to the ERRTST subroutine has the following format:

```
CALL ERRTST (i ,j)
```

The parameters for this subroutine have the following definitions:

i
is an integer value specifying the error number.

j
is a variable used to receive the return value of the error check.

j = 1:	Error **i** has occurred.
j = 2:	Error **i** has not occurred.

2.2.6.5 FDBSET Subroutine

You can use the FDBSET subroutine to specify special I/O options. It is provided primarily for compatibility with older FORTRAN implementations; similar and more extensive capabilities are available through the OPEN statement.

Using the FDBSET subroutine for a particular logical unit allows you to use the ASSIGN subroutine and the DEFINE FILE statement. However, you cannot use the OPEN or INQUIRE statements for the same logical unit. For VAX FORTRAN, the recommended method to specify I/O options is to use the OPEN statement. If you are planning to migrate to VAX systems, use the OPEN statement to assign a logical unit and specify any special options. If you do use the FDBSET subroutine, it must only be called before issuing the first I/O statement for the specified logical unit.

A call to the FDBSET subroutine has the following format:

```
CALL FDBSET (unit [,acc ,share ,numbuf ,initsz ,extend])
```

The parameters in this subroutine call have the following definitions:

unit
is an integer value specifying the logical unit. This is the only required parameter; all others are optional.

acc
is a character constant specifying the access mode to be used. The possible access modes are:

'READONLY':	Establish read-only access.
'NEW':	Create a new file.
'OLD':	Access an existing file.
'APPEND':	Extend an existing sequential file.
'UNKNOWN':	Try accessing an existing file first. If no such file exists, create a new one.

share
is a character constant 'SHARE' indicating that shared access is allowed.

numbuf
is an INTEGER*2 value specifying the number of buffers to be used for multibuffered I/O.

initsz
is an INTEGER*2 value specifying the number of blocks initially allocated for a new file.

extend
is an INTEGER*2 value specifying the number of blocks by which to extend a file.

2.2.6.6 IRAD50 Subroutine

You can use the IRAD50 subroutine to convert Hollerith data to Radix–50 form. You may call IRAD50 as a function subprogram if the return value is desired (see the first format below), or as a subroutine if the return value is not desired (see the second format below). The IRAD50 subroutine has the following formats:

```
n = IRAD50 (icnt ,input ,output)
CALL IRAD50 (icnt ,input ,output)
```

In the first format, *n* is an INTEGER*2 value containing the number of characters actually converted. In either format, the parameters have the following definitions:

icnt
is an INTEGER*2 value specifying the maximum number of characters to be converted.

input
is a Hollerith string to be converted to Radix–50. Note that the scanning of input characters terminates on the first non-Radix–50 character in the input string.

output
is a numeric variable or array element in which the Radix–50 results are stored. Three Hollerith characters are packed into each output word. The number of output words is computed as $(icnt + 2)/3$. For example, if a value of 4 is specified for **icnt**, two output words will result, even if an input string of only one character is converted.

2.2.6.7 RANDU Subroutine

The RANDU subroutine computes a pseudo-random number by implementing a general random number generator of the multiplicative congruential type. The random number computed by this random number generator is a single-precision value uniformly distributed in the following range:

$$0.0\ .LE.\ VALUE\ .AND.\ VALUE\ .LT.\ 1.0$$

A call to the RANDU subroutine has the following format:

```
CALL RANDU (i1 ,i2 ,x)
```

In this call format, the parameters have the following definitions:

i1
is an INTEGER*2 variable or array element containing the seed for computing the random number.

i2
is an INTEGER*2 variable or array element containing the seed for computing the random number.

x
is a real variable or array element in which the computed random number is stored.

The values of **i1** and **i2** are updated during the computation to contain the updated seed values.

The algorithm for computing the random number value is:

$$X(n+1) = 69069 * X(n)\ +\ 1\ (MOD\ 2^{32})$$

For compatibility, another multiplicative congruential random number generator is provided as the file LB:[1,1]F4PRAN.OBJ. The algorithm used by this random number generator is:

If $i1 = 0,\ i2 = 0,$

set the generator base

$X(n+1)\ =\ 2^{16}\ +\ 3$

Otherwise

$X(n+1)\ =\ (2^{16}\ +\ 3) * X(n)\ mod\ 2^{31}$

Store generator base **X(n = 1)** in **i1**, **i2**.

The result of this algorithm is **X(n+1)** scaled to a real value **Y(n = 1)** for $0.0 \leq Y(N = 1)\ <\ 1.$

2.2.6.8 R50ASC Subroutine

The R50ASC subprogram lets you convert Radix–50 values to Hollerith strings. A call to R50ASC has the following format:

```
CALL R50ASC (icnt ,input ,output)
```

In this format, the parameters have the following definitions:

icnt
is an INTEGER*2 value specifying the number of ASCII characters to be produced.

input
is a numeric variable or array element containing the Radix–50 data. The number of words of input is determined by $(icnt + 2)/3$.

output
is a numeric variable or array element where the Hollerith characters are to be stored.

If the undefined Radix–50 code is detected, or if the Radix–50 word exceeds 174777 (octal), question marks are placed in the output location.

2.2.6.9 USEREX Subroutine

The USEREX subroutine lets you specify a routine that will be called as part of your program termination process. This allows you to perform clean-up operations, for example, in non-FORTRAN routines.

On VAX systems, you can establish a termination handler directly by calling the system service routine SYS$DCLEXH. For more information, see the *VMS System Services Reference Manual.*

A call to the USEREX subroutine has the following format:

```
CALL USEREX (name)
```

In this format, **name** has the following definition:

name
specifies the routine to be called. The name of this routine must appear in an EXTERNAL statement in the program unit.

NOTE

Do not try to perform FORTRAN I/O operations as part of an exit handler. The FORTRAN I/O system provides its own exit-handling functions, such as file closing, so user-specified I/O operations may not work as expected. All OTS error handling is also disabled.

2.2.7 Functions

VAX FORTRAN also supports certain functions for compatibility with PDP–11 FORTRAN. The following sections discuss these functions. Some of these functions may have been superseded by features included in VAX FORTRAN, so if you are planning to migrate to VAX systems, consider using these new capabilities. To ensure the transportability of your programs, use these functions.

2.2.7.1 RAD50 Function

The RAD50 function subprogram provides a simplified way to encode six Hollerith characters as two words of Radix–50 data. The entry point for this function has the following format:

```
RAD50 (name)
```

In this function entry point, the **name** parameter has the following definition:

name
is a numeric variable name or array element corresponding to a Hollerith string.

NOTE

The RAD50 function is equivalent to calling the IRAD50 subroutine with **icnt** specified as 6 and **output** specified as RAD50. The following FORTRAN statements accomplish the same task:

```
FUNCTION RAD50 (A)

CALL IRAD50 (6,A,RAD50)
RETURN
END
```

You may use the RAD50 function as an argument to an RSX–11 system directive subroutine. For example:

```
REAL*8 A
DATA A/'TASK A'/
CALL REQUES (RAD50(A),...)
```

2.2.7.2 RAN Function

The RAN function subprogram returns a pseudo-random number as the function value. The entry point for the RAN function is:

```
RAN (j)
   or
RAN (i1 ,i2)
```

In this entry point, the parameters have the following definitions:

j
is an INTEGER*4 variable or array element containing the seed for computing the random number.

i1
is an INTEGER*2 variable or array element containing the seed for computing the random number.

i2
is an INTEGER*2 variable or array element containing the seed for computing the random number.

The values of **i1** and **i2** are updated during the computation of the random number to contain the updated seed values.

NOTE

The algorithm for computing the random number value is identical to the first algorithm specified in the RANDU subroutine (see Section 2.2.6.7). The RAN function is equivalent to calling the RANDU subroutine with RAN specified as the variable to receive the random number. These calls to the RAN function and the RANDU subroutine have the following formats:

```
TYPE *,RAN(i1, i2)

CALL RANDU(i1, i2, ran)
TYPE *,ran
```

This RAN function is distinguished from the single-argument VAX FORTRAN RAN function by the number of arguments. The single-argument form uses a statistically better algorithm and is recommended when compatibility with PDP–11 FORTRAN is not an issue.

2.2.8 Data Definitions

There are some differences in data definition between VAX FORTRAN, FORTRAN IV, and FORTRAN–77. The following sections describe these differences.

2.2.8.1 Floating-Point Results

If you call a math library routine, you may receive different results depending on which FORTRAN implementation you are using. The VAX implementations of the math library routines take advantage of the VAX instruction set, which is not available to the PDP–11 FORTRAN compilers. These VAX functions typically produce results with an accuracy equal to or greater than the corresponding PDP–11 functions, but in some cases there may be differences.

In addition, VAX FORTRAN does not immediately convert floating-point constants without exponents to REAL*4, as is the case in PRO and PDP–11 FORTRAN. This feature provides greater accuracy when such constants are used in double-precision expressions.

2.2.8.2 Character and Hollerith Constants

VAX FORTRAN and FORTRAN–77 support both Hollerith and character constants, with the following respective notations:

```
nHa..a
'a..a'
```

In FORTRAN IV, both of these notations are used for Hollerith constants. (Note that Hollerith constants have no data type; Hollerith constants assume a data type consistent with their use.)

In most cases, the conflicting use of the 'a..a' notation is not a problem; VAX FORTRAN and FORTRAN–77 can determine from the context of your program if you intend to use a Hollerith constant or a character constant. There is one case in which this is not true. In an actual argument list for a CALL or function reference, where the subprogram has a dummy argument, a constant in the 'a..a' notation is always passed as a character constant, never as a Hollerith constant.

Consider the following code segment. If subroutine F expects a Hollerith constant to be passed as argument *S*, execution of this code fragment is not correct since the actual and dummy arguments must agree in data type.

```
SUBROUTINE F(S)
    .
    .
    .
CALL F('ABCD')
```

To avoid this problem, use the actual Hollerith constant format for specifying the argument as follows:

```
SUBROUTINE F(S)
    .
    .
    .
CALL F(4HABCD)
```

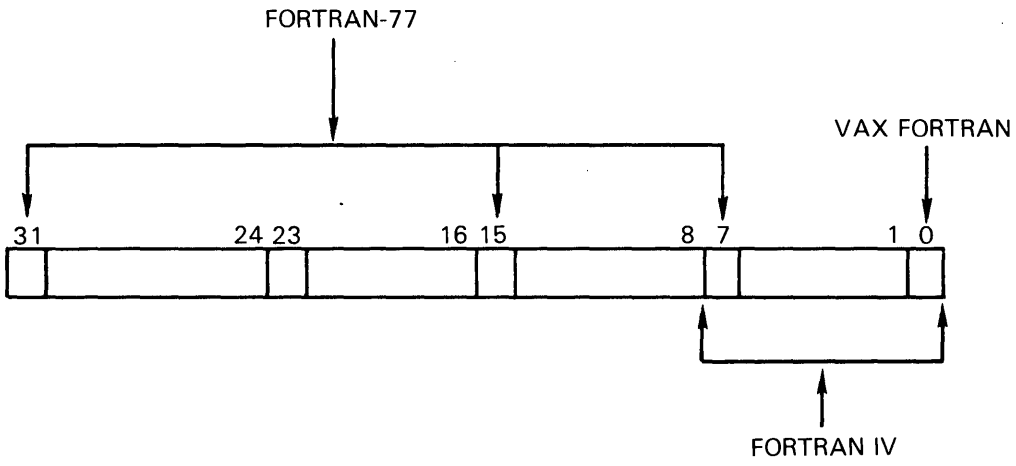
2.2.9 Expressions

The logical constants .TRUE. and .FALSE. are defined, respectively, as all ones and all zeros by both VAX FORTRAN and PDP–11 FORTRAN. However, when testing these constants, each compiler determines the value of the constant by checking different parts of the constant itself. The following list details the methods used by the various FORTRAN compilers to test .TRUE. and .FALSE. constants:

- VAX FORTRAN tests the low-order bit (bit 0) of a logical value. This is the system-wide VAX convention for testing logical values. VAX FORTRAN always evaluates any odd number as TRUE, because it tests bit 0.
- FORTRAN–77 tests the sign bit of a logical value: bit 7 for LOGICAL*1, bit 15 for LOGICAL*2, and bit 31 for LOGICAL*4. FORTRAN–77 always evaluates any negative number as TRUE, because it tests the sign bit.
- FORTRAN IV tests the low-order byte of a logical value; all zeros is a .FALSE. value, and any nonzero bit pattern is a .TRUE. value.

Figure 2–2 illustrates the data tested by each of the FORTRAN compilers.

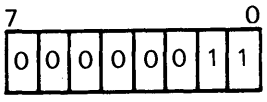
Figure 2–2: Logical Test Comparison



In most cases, this difference does not have any effect on the logical test comparison results. It is significant only for nonstandard FORTRAN programs that perform arithmetic operations on logical values and then make logical tests on the result. For example:

```
LOGICAL*1 BA
BA = 3
IF (BA) GO TO 10
```

The LOGICAL*1 variable *BA* has the following format:



The various compilers interpret this LOGICAL*1 value as follows:

- FORTRAN IV produces a value of .TRUE., since it interprets any nonzero bit pattern in the entire byte as a .TRUE. value.
- FORTRAN–77 produces a value of .FALSE., since it checks the sign bit (in this case, bit 7) to determine a .TRUE. or .FALSE. value.
- VAX FORTRAN produces a value of .TRUE., since it tests bit 0 to determine a .TRUE. or .FALSE. value.

2.2.10 Character Sets

The character sets for the various FORTRAN compilers have minor incompatibilities. Table 2–4 lists the special characters that are only valid under VAX FORTRAN.

Table 2–4: Character Set Incompatibilities

Symbol	Character
_	Underscore
%	Percent sign
&	Ampersand

By avoiding these special characters, you should not encounter any character set incompatibilities when transporting your programs.

2.3 System Dependencies

Using a system-specific feature can improve program execution speed, but does so at the expense of transportability. If you must use the system-dependent features of FORTRAN language elements that are incompatible across systems, try to isolate them in subprograms. This minimizes the changes you must make to convert the programs because it keeps the program logic from being tied to these system-specific features.

The following sections discuss differences in system dependencies.

2.3.1 I/O Differences and File Transfer

If you do not specify a logical unit number in an I/O statement, a default unit number is used. Table 2–5 shows the differences in the defaults used by VAX FORTRAN and those used by FORTRAN IV and FORTRAN-77.

Table 2–5: Default Logical Unit Numbers

I/O Statement	FORTRAN IV/FORTRAN-77 Unit	VAX FORTRAN Unit
READ	1	-4
PRINT	6	-1
TYPE	5	-2
ACCEPT	5	-3

The FORTRAN IV and FORTRAN-77 compilers use the normal logical unit numbers as the defaults; VAX FORTRAN uses unit numbers that are unavailable to users. This feature prevents conflicts between I/O statements that use the default logical unit numbers and those that use explicit logical unit numbers. This should have no visible effect on program execution.

If you try to OPEN a logical unit already connected to a file, FORTRAN IV generates an error. In VAX FORTRAN and FORTRAN-77, the behavior is as follows:

- If the file specification specified (or the default) matches that of the currently opened file, the new value (if any) of the BLANK keyword is used and the new open request is ignored.
- If the file specifications do not match, the currently open file is closed and the new file is opened.

In either case, neither VAX FORTRAN nor FORTRAN-77 generates an error, and neither language supports any method to achieve behavior that is compatible with FORTRAN IV.

2.3.2 Optimization of I/O with Operating System-Specific Features

If you want to optimize the I/O operations performed by your programs, you can use system services such as $QIO that are available on your system. However, these system service calls are not totally transportable. For more information, refer to the appropriate operating system compatibility guide.

2.3.3 File Naming Conventions

There is a difference between the file naming conventions of FORTRAN IV, FORTRAN-77, and VAX FORTRAN. Specifically, FORTRAN IV and FORTRAN-77 source code files have a default file type of FTN. VAX FORTRAN source code files have a default file type of FOR.

There are also some differences between file naming conventions on PDP-11, PRO, and VAX systems. The most obvious difference is in the length of allowable file names. On PDP-11 and PRO systems, a file name can consist of from 1 to 9 alphanumeric characters. On VAX systems, a file name can consist of from 1 to 39 characters. Also, PDP-11 and PRO listing files have a default file type of LST; VAX listing files have a default file type of LIS. VAX systems also provide a default file type for text library files (TLB); these files are not supported on PDP-11 and PRO systems. For more information on default file types, refer to the compatibility guide for the appropriate operating system.

2.3.4 Transportable File Specification Format

There is no way to include completely transportable file specifications in a FORTRAN program. However, there are two techniques that can make it easier to support FORTRAN programs that include file specifications. These techniques are:

1. Using logical names
2. Assigning file specifications to string variables

The following sections describe these techniques.

2.3.4.1 Using Logical Names

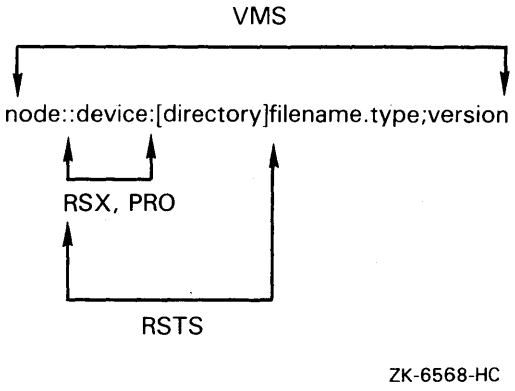
Logical names let you assign a mnemonic name to a file specification. Depending on the system, you can use a logical name to represent different fields of a file specification. Table 2–6 defines what fields of a file specification can be assigned to a logical name for each system.

Table 2–6: Logical Name Capabilities for File Specifications

System	Logical Name Capability
RSTS	Device and account
RSX	Device
P/OS	Device
VMS	Full file specification, including node, device, and directory

Figure 2–3 illustrates the fields of a file specification that can be assigned logical names.

Figure 2–3: File Specification Logical Name Comparison



Regardless of the system, you have to assign logical names at the monitor command level, not within a FORTRAN program.

The main advantage of using logical names is that you can redefine at least the device portion of a file specification at the monitor command level, regardless of the system.

2.3.4.2 File Specifications in String Variables

Placing file specifications in string variables allows you to avoid some of the common problems of hard-coding device specifications, which makes your code more transportable.

For example, if you assign a file specification to a string variable and then specify that string variable in all statements referencing that particular file, change the assignment of the string variable to transport your program from one system to another.

2.3.5 File Support

The types of files supported by FORTRAN IV, FORTRAN-77, and VAX FORTRAN differ according to which file services are supported by that compiler. Table 2-7 shows the different types of files supported by the various FORTRAN compilers.

Table 2-7: Supported Files for DIGITAL FORTRAN Compilers

Compiler	Supported File Structures
FORTRAN IV	Direct access or sequential, with fixed- or variable-length records
FORTRAN-77	Direct access, sequential, relative, or indexed sequential, with fixed- or variable-length records
VAX FORTRAN	Direct access, sequential, relative sequential or direct, or indexed sequential or keyed, with fixed- or variable-length records

For more information on file services, see the appropriate operating system compatibility manual.

2.3.6 Keywords

There are minor differences in some of the values available for the OPEN statement keywords. The following sections describe these differences.

2.3.6.1 OPEN Statement BLANK Keyword Default

In FORTRAN-77 and VAX FORTRAN, the BLANK keyword to the OPEN statement lets you control the interpretation of blanks in numeric input fields.

- When BLANK specifies 'NULL', all blanks in a numeric input field are ignored (except if the field is comprised of all blanks, in which case it is treated as zero).
- When BLANK specifies 'ZERO', all blanks other than leading blanks are treated as zeros.

For both FORTRAN-77 and VAX FORTRAN, the default is BLANK = 'NULL'.

The FORTRAN IV OPEN statement does not support a BLANK keyword. FORTRAN IV interprets blanks in a numeric input field as zeros, which is identical to specifying BLANK = 'ZERO' in the OPEN statement. If you use the /NOF77 qualifier to compile a FORTRAN-77 or VAX FORTRAN program, or you do not explicitly open a logical unit with an OPEN statement, both FORTRAN-77 and VAX FORTRAN default to BLANK = 'ZERO'.

The following example illustrates the differences between these interpretations:

```
Program:

       OPEN (UNIT=1, STATUS='OLD') READ(1,10)I, J
 10    FORMAT (2I5) END

Data Record:

#1#2####12
```

These statements under FORTRAN IV (or under FORTRAN-77 or VAX FORTRAN with the /NOF77 qualifier) have the following results:

```
I = 1020
J = 12
```

Under FORTRAN-77 or VAX FORTRAN, these same statements have the following results:

```
I = 12
J = 12
```

You can also use the BZ edit descriptor in the FORMAT statement to indicate that blanks be interpreted as zeros. However, this edit descriptor only applies to that particular FORMAT statement; it does not specify a default interpretation of blanks for an entire file.

2.3.6.2 OPEN Statement STATUS Keyword Default

In FORTRAN-77 and VAX FORTRAN, you can use the STATUS keyword to the OPEN statement to specify the initial status of a file ('OLD', 'NEW', 'SCRATCH', or 'UNKNOWN'). In FORTRAN IV, use the TYPE keyword instead of STATUS. (The TYPE keyword is also supported on VAX FORTRAN and FORTRAN-77 for compatibility with FORTRAN IV.)

By default, both FORTRAN-77 and VAX FORTRAN assume a STATUS of 'UNKNOWN'. An OPEN statement compiled under either of these compilers first searches to see if a file of that name already exists. If so, that file is opened; if not, a new one is created. FORTRAN IV assumes a TYPE of 'NEW' so that, by default, an OPEN statement compiled by FORTRAN IV creates a new file with the name specified instead of first searching for any existing file of that name.

If you specify the /NOF77 qualifier to either VAX FORTRAN or FORTRAN-77, the default status of the file is 'NEW' as in FORTRAN IV.

2.3.6.3 OPEN Statement INITIALSIZE Keyword

For VAX FORTRAN, the space requested by the INITIALSIZE keyword is allocated contiguously, if possible, on what is called a best-try basis. If you specify an INITIALSIZE value and there is sufficient contiguous space available, the space allocated is contiguous. If there is not enough contiguous space, the space allocated is noncontiguous.

For FORTRAN-77, allocation of contiguous or noncontiguous space depends on the sign of the value specified for the INITIALSIZE and EXTENDSIZE keywords. To be compatible with PDP-11 FORTRAN, VAX FORTRAN uses the absolute value of the user-supplied value.

2.3.6.4 DISPOSE = 'PRINT' Specification

On RSTS/E, RSX-11M, RSX-11M-PLUS, and VMS systems, a file printed under DISPOSE = 'PRINT' is always saved. To print and then delete a file, specify DISPOSE = 'PRINT/DELETE'.

2.3.7 Record Management Services

FORTRAN-77 provides the following methods for controlling file and record access:

- File Control Services (FCS)
- Record Management Services (RMS)

These file-handling facilities differ in the types of files they support, but the calls to mutually supported routines are identical. For example, indexed or relative files are not supported under FCS, but are supported by RMS.

If you do not use any file structures that are not supported by FCS, you can still request that FORTRAN-77 use RMS as your file-handling facility by specifying this option at link time. VAX FORTRAN supports only RMS; FORTRAN IV supports only FCS.

2.3.8 Block I/O

If you are using VAX FORTRAN or FORTRAN-77 with RMS, you can perform block I/O. Block I/O is an intermediate step between RMS record operations and the direct use of the $QIO system service. Block I/O operations let you directly read or write the blocks of a file. For more information on block I/O, refer to the appropriate operating system compatibility manual.

2.3.9 Run-Time Libraries

Run-time libraries are provided for all PRO, PDP-11, and VAX systems. Although these library routines may have some internal differences, they are designed to make these differences transparent to you. Therefore, if you have a program that calls run-time libraries, that program should be transportable between PRO, PDP-11, and VAX systems. For more information on run-time libraries, see the appropriate operating system compatibility manual.

2.4 Program Segmentation

It is a good idea to modularize your code, even if you are not explicitly concerned with transportability. Modularizing your code means that you take a complex task, divide it into modules, and code each module as a separate procedure. This has several advantages over coding a complex program as a single module. For example:

- You can use any modular procedure in any of your programs.
- You can add a modular procedure to a library at any time.
- You do not have to rewrite common algorithms every time you need them in a program.
- You can divide a complex program into simpler procedures, which reduces development time and complexity and increases reliability.
- You can modify or replace a procedure without having to modify each program that calls it.
- You can significantly reduce the time and effort involved in debugging an application.

Each module should contain a single procedure or a group of related procedures.

2.5 Procedure Calling

After you modularize your code, you must pull in all the modules required by your main program. These modules can be coded and invoked either as function or subroutine subprograms.

2.5.1 Calling Function Subprograms

A function subprogram is a program unit consisting of a FUNCTION statement followed by a series of statements that define the computing procedure of that function. To transfer control to a function subprogram, you use what is called a function reference. A function reference means using the name of the function as a variable in an assignment statement. A RETURN or END statement returns control from the function subprogram to the calling program.

A function subprogram returns a single value to the calling program unit by assigning that value of the function's name.

The following example illustrates the definition of a FORTRAN function subprogram, and the statement used to transfer control to that function:

```
      FUNCTION ROOT(A)
      X = 1.0
2     EX = EXP(X)
      EMINX = 1./EX
      ROOT = ((EX + EMINX) * .5 + COS(X) -A)/((EX - EMINX) * .5 - SIN(X))
      IF (ABS(X-ROOT) .LT. 1E-6) RETURN
      X = ROOT
      GO TO 2
      END
```

This function subprogram uses the Newton-Raphson iteration method to obtain the root of the following function:

$$F(X) = cosh(X) + cos(X) - A = 0$$

The value of A is passed as an argument. This calculation is repeated until the difference between the values is less than 1.0E-6. To invoke this function, use the following statement:

```
FUNCROOT = ROOT(A)
```

2.5.2 Calling Subroutine Subprograms

A subroutine subprogram is a program unit consisting of a SUBROUTINE statement followed by a series of statements that define the computing procedure. To transfer control to a subroutine subprogram, you must use the FORTRAN CALL statement. You must also include a RETURN or END statement to return control from the subroutine subprogram to the calling program.

The following example illustrates a simple subroutine subprogram:

```
SUBROUTINE ADD(SUM,A,B)
SUM = A + B
RETURN
END
```

To invoke this subroutine subprogram, include the following statement in your calling program:

```
FIRST = 5.0
SECOND = 9.5
TOTAL = 0.0
CALL ADD(TOTAL, FIRST, SECOND)
```

2.6 Errors and Error Handling

Differences in run-time support between VAX FORTRAN and PRO and PDP–11 FORTRAN are reflected in run-time error numbers and in run-time error reporting. The following sections discuss these differences.

2.6.1 Run-Time Library Error Numbers

If you have a program that uses the ERRSNS subroutine, that program may need to be modified because certain PRO and PDP–11 FORTRAN run-time errors were either deleted from, or redefined in, the VAX Run-Time Library. Table 2–8 lists the error numbers that are affected.

Table 2–8: Incompatible Error Numbers

Error Number	Incompatibility
2 through 14	Deleted; these error numbers reported fatal PDP–11 hardware conditions.
37 (INCONSISTENT RECORD LENGTH)	Redefined; continuation is not allowed.
65 (FORMAT TOO BIG FOR "FMTBUF")	Deleted; this error cannot occur because space is acquired dynamically for run-time formats.
72, 73, 82, 83, 84	Redefined; floating-point arithmetic errors and math library errors return -0.0 (a hardware reserved operand) rather than +0.0.
75 (FPP FLOATING TO INTEGER CONVERSION OVERFLOW)	Deleted; error number 70 is now reported. Error number 70 indicates ARITHMETIC TRAP, INTEGER OVERFLOW.
86 (INVALID ERROR NUMBER)	Deleted; error number 48 is now reported. Error number 48 indicates INVALID ARGUMENT TO FORTRAN RUN-TIME LIBRARY.
91 (COMPUTED GO TO OUT OF RANGE)	Deleted; no error is generated by the VAX hardware when this condition occurs. Program execution continues in line.
92 (ASSIGNED LABEL NOT IN LIST)	Deleted; VAX FORTRAN does not perform this check at run time.
94 (ARRAY REFERENCE OUTSIDE ARRAY)	Deleted; error number 77 is now reported. Error number 77 indicates TRAP, SUBSCRIPT OUT OF RANGE.
95 through 101	Deleted; these error numbers reported PDP–11 FORTRAN errors that cannot occur in VAX FORTRAN.

2.6.2 Error Handling and Reporting

VAX FORTRAN differs from FORTRAN IV and FORTRAN-77 in the way that it treats error continuation, I/O errors, and OPEN or CLOSE statement errors. The following sections discuss these differences.

2.6.2.1 Continuing After Errors

In PDP-11 FORTRAN, program execution after errors, such as floating-point overflows, normally continues until 15 such errors occur. At that point, execution is terminated. VAX FORTRAN, however, sets a limit of one such error; program execution normally terminates when the first error occurs. To change this behavior, you can take one of the following steps:

- Include a condition handler in your program to change the severity level of the error. Severity levels of Warning and Error permit continuation.
- Include the ERRSET subroutine. ERRSET alters the Run-Time Library's default error processing to match the behavior of FORTRAN-77.

2.6.2.2 I/O Errors with IOSTAT or ERR Specified

If an IOSTAT or ERR specification is included in the I/O statement, VAX FORTRAN neither generates an error message nor increments the image error count when an I/O error occurs. Under these circumstances, FORTRAN IV and FORTRAN-77 both report the error and increment the task error count.

2.6.2.3 OPEN or CLOSE Statement Errors

Unlike PRO and PDP-11 FORTRAN, VAX FORTRAN reports only the first error encountered in an OPEN or CLOSE statement. PRO and PDP-11 FORTRAN report all errors detected in processing these statements.

Index

W

X

digital