RT-11/RSTS/E

FORTRAN IV User's Guide

Order No. DEC-11-LRRUA-A-D

RT-11/RSTS/E

FORTRAN IV User's Guide

Order No. DEC-11-LRRUA-A-D

digital equipment corporation · maynard, massachusetts

First Printing, December 1975

The information in this document is subject to change without notice and should not be construed as a commitment by Digital Equipment Corporation. Digital Equipment Corporation assumes no responsibility for any errors that may appear in this document.

The software described in this document is furnished under a license and may be used or copied only in accordance with the terms of such license.

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

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

The following are trademarks of Digital Equipment Corporation:

DIGITAL	DECsystem-10	MASSBUS
DEC	DECtape	OMNIBUS
PDP	DIBOL	OS/8
DECUS	EDUSYSTEM	PHA
UNIBUS	FLIP CHIP	RSTS
COMPUTER LABS	FOCAL	RSX
COMTEX	INDAC	TYPESET-8
DDT	LAB-8	TYPESET-10
DECCOMM		TYPESET-11

CONTENTS

PAGE

FIGURES

TABLES

PREFACE

This document provides information necessary to compile, link, execute, and debug a FORTRAN program under the RT-11 and RSTS/E operating systems. Chapter one describes the operating procedures. Chapter two provides information about the Object Time System (OTS). This system is a collection of routines, selectively linked to the user's program, which perform certain arithmetic, input/output, and system dependent service operations. It also detects and reports run-time error conditions. Chapter three describes system dependent information not included in the PDP-11 FORTRAN Language Reference Manual. Chapter four contains programming suggestions for increasing the effectiveness and efficiency of RT-11/RSTS/E FORTRAN IV. Chapter five describes the Concise Command Language option. The Appendices provide reference information about internal data representations, system subroutines, error diagnostics, and compatibility of FORTRAN IV with other PDP-11 FORTRAN processors.

Intended Audience

This manual should be used only after some knowledge of the FORTRAN language, as implemented on the PDP-11, has been acquired. The associated document which can be used for this purpose is the PDP-11 FORTRAN Language Reference Manual. The user should also be familiar with the operating system as described in either the RT-11 System Reference Manual or the RSTS/E System User's Guide. The RSTS/E Documentation Directory and the RT-11 Documentation Directory contain additional information on the respective documentation sets.

DOCUMENTATION CONVENTIONS

All monitor and system program command lines are terminated by pressing the RETURN key. Since this is a non-printing character, at certain places in the text the notation <CR> represents the RETURN key.

In examples of keyboard dialogue, monitor output and program output are underlined; user input is not.

In format descriptions, uppercase characters represent information that must be entered exactly as shown; lowercase characters represent variable information that must be supplied by the user.

Some special keyboard characters require that the CTRL (control) key

be pressed simultaneously with a second character. These characters are denoted by ↑ (up arrow); e.g., ↑Z (CTRL Z).

Ellipsis marks (...) indicate the omission of one or more words within a passage and show that the passage continues in the same vein.

CHAPTER 1

OPERATING PROCEDURES

1.1 USING THE FORTRAN IV SYSTEM

Figure 1-1 outlines the steps required to prepare a FORTRAN IV source program for execution under the RT-11 or RSTS/E Executive: (1) compilation, (2) linking, and (3) execution.

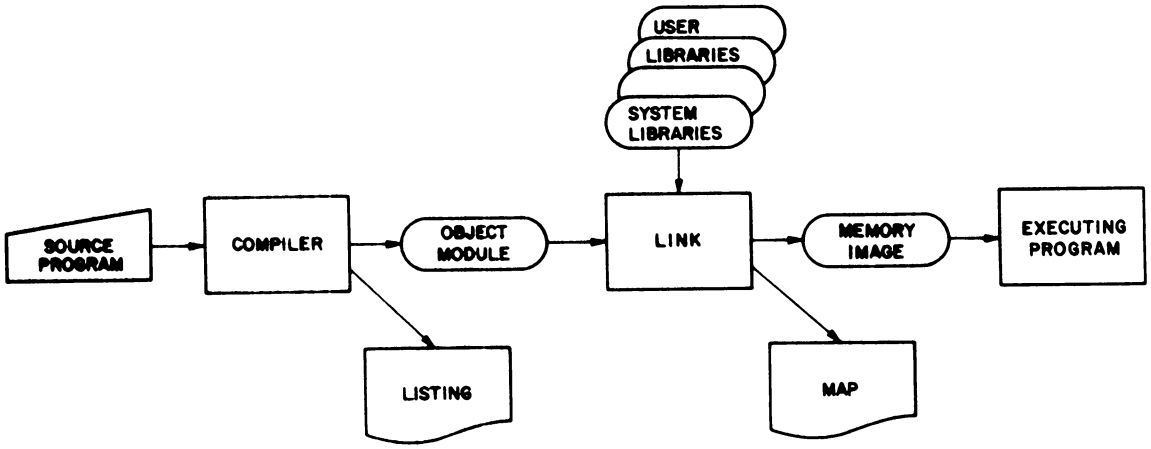


Figure 1-1 Steps in Compiling and Executing a FORTRAN IV Program

Step 1 in Figure 1-1 is initiated by running the FORTRAN IV Compiler, FORTRAN, accompanied by a command string that describes the input and output files, and switch options, if desired, to be used by the compiler. The compiler generates an object file which must be linked by the linker prior to execution.

Step 2 is initiated by running the linker, LINK, accompanied by a similar command string. The linker combines all program units and the necessary routines from the FORTRAN Library, and generates a memory image file.

Step 3 is initiated by the monitor R or RUN command for RT-11 and RUN $EXEC for RSTS/E.

1.1.1 Filename Specifications

The RT-11/RSTS/E FORTRAN IV Compiler and Linker accept a command string of the form:

```
output = input /sw
```

where

output	is the output filename specification(s).
input	is the input filename specification(s).
/sw	is one or more optional switches used to request certain functions from the FORTRAN IV Compiler and Linker. Switch options are tabulated in Section 1.2.1.

The user should note that imbedded blanks are not permitted in command string specifications.

Each filename specification has the form:

(RT-11)		(RSTS/E)
dev:filename.ext	or	dev:[p,pn]filename.ext<prot>

where

dev:	is an optional 2- to 3-character name specifying a legal device code as shown in Table 1-1. If the device code is omitted, the default storage is used.
filename	is any 1- to 6-character alphanumeric filename.
.ext	is any 0- to 3-character alphanumeric extension. If one is not specified, the FORTRAN IV Compiler supplies, by default, certain extensions as shown in Table 1-2.
[p,pn]	is a RSTS/E project (p), programmer number (pn) which is used to identify the owner of a file.
<prot>	is a RSTS/E protection code restricting access to a file. The degree of restriction is determined by a code or combination of codes as shown in Table 1-3. Protection codes have effect only on output files.

Table 1-1
Device Specifications

Device	RT-11	RSTS/E
Card reader	CR:	CR:
TA11 cassette (n=0 or 1)	CTn:	
Default storage	DK:	SY:
RP02 or RP03 disk (n=0 to 7)	DPn:	DPn:
RP04 disk (n=0 to 7)		DBn:
RS03/4 disk (n=0 to 7)	DSn:	
DECtape (n=0 to 7)	DTn:	DTn:
RX01 floppy disk (n=0 or 1)	DXn:	DXn:
Line printer (n=0 to 7)	LP:	LPn:
TU16 magtape (n=0 to 7)	MMn:	MMn:
TU10 or TS03 magtape (n=0 to 7)	MTn:	MTn:
High speed paper tape punch	PP:	PP:
High speed paper tape reader	PR:	PR:
RF11 fixed-head disk drive	RF:	DF0:
RK11 disk cartridge drive (n=0 to 7)	RKn:	DKn:
System device	SY:	SY:
Specified unit from which the system was bootstrapped.	SYn:	SY:
Current user terminal	TT:	KB: TT: or TI:

For more information on device specifications refer to the RT-11 System Reference Manual and the RSTS/E System User's Guide.

Table 1-2
Filename Extensions

File	Assumed Extension on Input File	Default Extension on Output File
Object file	-	.OBJ
Listing file	-	.LST
Source file	.FOR	-
Load Map file	-	.MAP
Save Image file	-	.SAV
Absolute Binary file	-	.LDA (/L)
Relocatable Image file	-	.REL (/R) (RT-11 only)

Table 1-3
Protection Codes (RSTS/E only)

Code	Meaning
1	read protect against owner
2	write protect against owner
4	read protect against owner's project number
8	write protect against owner's project number
16	read protect against all others who do not have owner's project number
32	write protect against all others who do not have owner's project number
64	compiled, run-only files
128	privileged program

These codes can be used singularly, or combined, to provide greater degrees of protection. For example, a protection code of <60> is a combination of codes <32>, <16>, <8>, and <4>. This combines the restrictions offered by each code into one code; that is, only the owner is allowed to read and write the file. Code <60> is the protection given to files by default under the RSTS/E system. This default can be changed by typing the ASSIGN command followed by the new value in angle brackets. The protection code <2> should not be specified on output files. For more information on protection codes, refer to Section 9.1 of the BASIC PLUS Language Manual.

1.1.2 Locating a File

The FORTRAN IV Compiler uses the filename specification to locate a file. It begins by searching the specified device for the filename with the specified extension. If the device is not specified, it searches default storage. If an extension is not specified, the extension .FOR is assumed.

Under the RSTS/E operating system, if the [project,programmer] number is specified, then the file being sought must exist under that account number. If the [project,programmer] number is not specified, the compiler searches the current user's directory. If the file is not in the user's directory, a search is made of the system library [1,2]. After the file is located, the protection code identifies the privileges granted the user. If a protection code is not specified on output, the default code of <60> is used.

If the file cannot be located or is protected against the user, the following message is printed:

```
?FIL NOT FND?
```

A similar form of this message appears if a filename specification given to a utility program (e.g., MACRO, LINK, etc.) references a file which cannot be found.

1.2 RUNNING THE FORTRAN IV COMPILER

To execute the FORTRAN IV Compiler the command:

(RT-11)
```
.R FORTRAN
*
```

(RSTS/E)
```
RUN $FORTRAN
*
```

is given. The FORTRAN IV Compiler then prints an asterisk (*) to indicate that it is ready to accept a command string.

The FORTRAN IV Compiler can produce two output files: an object file and a listing file. Up to six FORTRAN IV source language files are permitted as input files. If multiple input files are given, they are considered to be logically concatenated. However, source lines are not to be broken over file boundaries.

An input file can contain more than one program unit if that file resides on a random-access device. The object code for all program units is sent to the single object file, and is handled by the linker at link time.

A sample FORTRAN IV Compiler command sequence is shown below:

(RT-11)
```
.R FORTRAN
*OBJECT,LIST=FILE1
```

(RSTS/E)
```
RUN $FORTRAN
*OBJECT,LIST=FILE1
```

This command string directs the compiler to take the source file FILE1.FOR from the default device, and output the files LIST.LST and OBJECT.OBJ to the default device.

Either of the compiler output files can be eliminated by omitting its

file specification from the command string. For example:

(RT-11)	(RSTS/E)
.R FORTRAN	RUN $FORTRAN
*FILE1=FILE1	*FILE1=FILE1

produces FILE1.OBJ on the default device but no listing file, while

*,LP:=FILE1	*,LP:=FILE1

produces a listing on the line printer, but no object module output.

The CCL (Concise Command Language) option provides an alternative procedure for invoking system programs under RSTS/E. Chapter 5 contains more information on this option.

1.2.1 Compiler Switches

The FORTRAN IV Compiler command strings can contain switch options on the input and output file specifications. The switch options can use either octal or decimal values. Any switch of the form /S:n causes n to be interpreted as an octal value; whereas, /S!n causes n to be interpreted as a decimal value. The switches are as follows (they are initialized to the specified default values for each command string):

Switch	Description
/A	Add the compilation statistics to the list file (see Section 1.2.2).
/D	Compile lines with a letter D in column one. These lines are treated as comment lines by default (see Section 1.5).
/E	Accept a full 80 columns of FORTRAN source input per line. Columns 73 through 80 are treated as a sequence field (comments) by default.
/H	Print a list of compiler switches on the listing device specified. If no listing device is specified, the output is directed to the user's terminal.
/L:n	Specify the listing options. The argument n is coded as follows:

/L:0 or /L	list diagnostics only
/L:1 or /L:SRC	list source program only
/L:2 or /L:MAP	list storage map only
/L:4 or /L:COD	list generated code only

Any combination of the above list options can be specified by summing the argument values for the desired list options. For example:

/L:7 (or /L:ALL)

requests a source listing, a storage map listing, and a generated code listing. If this switch is omitted, the

Switch	Description
	default list option is 3 (source and storage map). See Section 1.2.2.
/N:m	Enable specification of the maximum number of logical units that can be concurrently open at execution time; m is an octal constant between 1 and 16 for RSTS/E and 1 and 17 for RT-11. The default is set to 6 if the switch is not specified. This switch functions only when one of the input files contains the main program unit.
/O	Include options-in-effect in list file. This list specifies the state of each compiler option, i.e., on or off. (See Section 1.2.2).
/P	Disable the global optimizer. Using this switch may reduce program storage requirements, but will slightly increase execution time.
/R:m	Enable specification of the maximum formatted record size allowed at execution time; m is an octal constant between 4 and 7777. The default is 136 (decimal) bytes if this switch is not specified. This switch functions only when one of the input files contains the main program unit.
/S	Suppress ISNs (Internal Sequence Numbers, for source line number accounting). This option reduces storage requirements for generated code and slightly decreases execution time but disables line number information during Traceback (see Section 2.6).
/T	Allocate two words for default length integer variables. Normally, single storage words are the default allocation for integer variables not given an explicit length specification (i.e., INTEGER*2 or INTEGER*4). When two words are allocated, only the low address word is used to store the value.
/U	Disable USR (User Service Routines) swapping at runtime. By default the USR is always swapped. This switch will function only when one of the input files contains the main program unit. (This switch has no meaning under RSTS/E but is included for compatibility.)
/V	Disable all vectoring of arrays (see Section 2.5).
/W	Enable compiler warning diagnostics (see Section C.1.3).

1.2.2 Listing Formats

There are five optional sections that can be included in the compilation listing. By default the source program and the storage map are included in the compilation listing. The list of options-in-effect, the generated code, and the compiler statistics can

also be included. Any combination of these sections can be requested by using switches in the compiler command string (see Section 1.2.1). A description of each section is given below. Figure 1-2 provides a sample of the information included in each section.

```
FORTRAN IV      V01C-01     THU 13-NOV-75 16:14:06

*EX2/L:ALL/O/A=EX2

OPTIONS IN EFFECT:

                  SOURCE
                  MAP
                  CODE
                NOLEAPYEAR
                  OPTIM
                  LRECL=0136
                  STAT
                  ISNS
                NOCOL80
                  USRSWAP
                NODIAG
                NOINTEGER*4
                  NLCHN=06
                NODEBUG
                  VECTOR
                NOWARN

FORTRAN IV      V01C-01     THU 13-NOV-75 16:14:06                PAGE 001

0001          INTEGER INT
0002          REAL REAL
0003          COMPLEX IMAG
0004          DOUBLE PRECISION DBLE
0005          DATA INT/100/
0006          REAL = INT/2 + 5.
0007          DBLE = REAL/2. + 3.14159682516D0
0008          IMAG = CMPLX( REAL, 3.21 )
0009          WRITE (5,10) IMAG
0010    10    FORMAT(1X,2F8.5)
0011          STOP
0012          END


FORTRAN IV      STORAGE MAP

NAME     OFFSET   ATTRIBUTES

INT      000006   INTEGER*2 VARIABLE
REAL     000034   REAL*4    VARIABLE
IMAG     000040   COMPLEX*8 VARIABLE
DBLE     000050   REAL*8    VARIABLE
CMPLX    000000   COMPLEX*8 PROCEDURE
```

Figure 1-2 A Sample Compilation Listing

```
FORTRAN IV      GENERATED CODE


ISN #0006
000060  LSN$     #000006
000064  MOI$MS   000006
000070  DII$IS   #000002
000074  CFI$
000076  ADF$IS   #040640
000102  MOF$SM   000034

ISN #0007
000106  ISN$
000110  MOF$MS   000034
000114  DIF$IS   #040400
000120  CDF$
000122  ADD$MS   000020
000126  MOD$SM   000050

ISN #0008
000132  ISN$
000134  REL$     000030
000140  REL$     000034
000144  CAL$     #000002 CMPLX+#000000
000152  MOD$RM   000040

ISN #0009
000156  ISN$
000160  REL$     000016
000164  REL$     000010
000170  IFW$
000172  REL$     000040
000176  TVC$
000200  EOL$

ISN #0011
000202  LSN$     #000013
000206  STP$


***** COMPILATION STATISTICS *****
*                                 *
*                                 *
*-------- COMPILER  TABLES --------*
* SYMBOLS:           00081 WORDS *
* PROGRAM:           00039 WORDS *
* FREE CORE:         21285 WORDS *
*                                 *
*--------- OBJECT CODE ----------*
* IMPURE SECTION:    00024 WORDS *
* PURE SECTION:      00044 WORDS *
* TOTAL:             00068 WORDS *
*                                 *
***********************************
```

Figure 1-2 (Cont.) A Sample Compilation Listing

1.2.2.1 Options Listing - The options-in-effect list can be used as a quick reference to the status of each possible compiler option. Options in the list preceded by 'NO' are not in effect; those not preceded by 'NO' are in effect. The maximum number of logical units that can be concurrently open (NLCHN) and the maximum record length (LRECL) are given as the default values or the values specified by the /N and /R switches respectively. Also included are the day of the week, date, and time of compilation and a copy of the compiler command string for identification purposes.

1.2.2.2 Source Listing - The source program is listed in this section just as it appeared in the input file. Internal sequence numbers are added by the compiler for easier reference. Note that internal sequence numbers are not always incremented. For example, the statement following a logical IF will have an internal sequence number two greater than that of the IF. The IF statement has internally been assigned two sequence numbers: one for the comparison and one for the associated statement.

1.2.2.3 Storage Map Listing - This section includes a list of all symbolic names referenced in the program unit. An offset from the base of the program unit (subject to relocation at link time) is given for all local symbols. There is also a description of the symbolic name including usage, data type and, in the case of COMMON blocks and array names, the defined size in storage units.

NOTE

Blank COMMON is described as COMMON BLOCK / / in the storage map, but is located on a LINK map as a CSECT named .$$$$.

1.2.2.4 Generated Code Listing - This section of the list file contains a symbolic representation of the object code generated by the compiler (see Section 2.2) including a location offset from the base of the program unit, the symbolic Object Time System (OTS) routine name, and routine arguments. The code generated for each statement is labeled with the same internal sequence number as that specified in the source program listing, providing easier cross reference.

1.2.2.5 Compilation Statistics - This section includes a report on memory usage during the compilation process and the storage requirements for the object code generated by the compiler. However, this report does not accurately reflect the amount of storage required for the executable program. The object code storage requirements printed on the list must be increased by the size of any COMMON blocks referenced, plus, any routines required from the FORTRAN library.

1.2.3 Compiler Memory Requirements

The amount of memory available for compilation and the options available for obtaining additional space differ between RT-11 and RSTS/E. Under RT-11, device handlers and the symbol table require a portion of memory during compilation; if the remaining memory is insufficient after minimizing the number of different physical devices and variable names specified, the program unit can be segmented into program units small enough to compile in the available amount of memory.

Under RSTS/E, additional space can be acquired by switching from a non-privileged to a privileged account, increasing the system swap maximum, or segmenting the program. These options and the amount of memory available are discussed in more detail in the following subsections.

1.2.3.1 Compiler Memory Requirements Under RT-11 - During the compilation process, the RT-11 Resident Monitor (RMON), the compiler root segment, one overlay region, the stack, and the required device handlers (other than the handler for the system device which is included in the RMON) must be core resident. The remaining memory is used for the symbol table and the internal representation of the program. In a machine with 8K of memory, this allows the compilation of a program unit as large as several hundred statements in length. However, if the compiler runs out of memory during the compilation process (an error message is typed on the terminal; see Section C.1), the program unit must be divided into two or more program units, each of which must be small enough to compile in the available memory.

Since device handlers and the symbol table must be resident during the compilation process, minimizing the number of different physical devices specified in the command string and reducing the number of variable names increases the amount of memory available for object code generation.

1.2.3.2 Compiler Memory Requirements Under RSTS/E - When the RSTS/E FORTRAN IV Compiler is invoked, it acquires as much free memory (up to 28K words) as may be permitted to the current user. The private core maximum for the account under which the user is running determines the amount of free storage allocated. The amount of storage available will never exceed the SWAP MAX currently set by the system manager. In addition, the system manager may restrict the dynamic memory requested by FORTRAN IV for every user.

Hence, if the compiler runs out of free space during a compilation (this is reported by a "FATAL ERROR T" message typed on the user's terminal), several actions can be attempted to accommodate the compilation:

1. If the compilation was attempted by a non-privileged user whose private core maximum is smaller than the system swap maximum, the compilation may proceed, if done under a privileged account; this may allow the compiler to acquire a larger memory area. Alternatively, the user can contact the system manager to have the private core maximum increased as required.

2. If a compilation should terminate with insufficient space under a privileged account, two courses of action are available:

   a. The user can consult with the system manager to determine if the FORTRAN compilation size limit can be increased to accommodate large compilations, or

   b. the user can segment the program unit which will not compile into two or more smaller program units, and/or reduce the number of variables, arrays, and constants in the program unit (to save compiler symbol table space).

1.3 LINKING PROCEDURES

The Linker, LINK, combines one or more user-written program units together with selected routines from any user libraries (see Section 1.3.1) and the default FORTRAN IV System Library, FORLIB. LINK also provides an overlay capability (see Section 1.3.2).

LINK generates a single runnable core image file and an optional load map from one or more object files created by the MACRO assembler or the RT-11/RSTS/E FORTRAN IV Compiler.

The LINK command has the form:

(RT-11)	(RSTS/E)
.R LINK	RUN $LINK
*command string	*command string

The command string has the following format:

```
dev:binout,dev:mapout=dev:obj1,dev:obj2,.../s1/s2/s3
```

where:

dev:	is a random access device for the save image output file (binout), and any appropriate device in all other instances. If dev: is not specified, the default device is assumed. If the output is to be LDA format (i.e., the /L switch was used), the output file need not be on a random-access device.
binout	is the name to be assigned to the linker's save image, or LDA format output file. This file is optional; if not specified, no binary output is produced. (Save image is the assumed output format unless the /L switch is used.)
mapout	is the optional load map file.
obj1,...	are files of one or more object modules to be input to the linker (these may be library files).
/s1/s2/s3	are optional switches as explained in Table 1-4.

Table 1-4
Linker Switches

Switch Name	Command Line	Meaning
/A	1st	Alphabetizes the entries in the load map.
/B:n	1st	Bottom address of program is indicated as n. The bottom address determines the amount of stack (SP) space available to the program being linked. The default bottom is 1000(octal), which provides approximately 80 words of stack. This can be increased by specifying the /B switch with an argument greater than 1000(octal).
/C	any	Continues input specification on another command line. Used also with /O.
/F	1st	Instructs the linker to use the default FORTRAN library, FORLIB.OBJ. Note that FORLIB does not have to be specified in the command line.
/I	1st	Includes in the program image, (see Section 1.3.3), the library object modules that define the specified global symbols.
/L	1st	Produces an output file in LDA format.
/M or /M:n	1st	Specifies the stack address at the terminal keyboard or via n.
/O:n	any but the 1st	Indicates that the program has an overlay structure: n specifies the overlay region to which the module is assigned.
/R	1st	Produces an output file in relocatable image format for execution as a foreground job under RT-11.
/S	1st	Allows the maximum amount of space in memory to be available for the linker's symbol table. (This switch should only be used when a particular link stream causes a symbol table overflow.)
/T or /T:n	1st	Specifies the transfer address at terminal keyboard or via n.

An example of the LINK command format as used with FORTRAN IV is given below.

(RT-11)	(RSTS/E)
.R LINK	RUN $LINK
*LOAD,MAP=MAIN,SUB1,SUB2/F	*LOAD,MAP=MAIN,SUB1,SUB2/F

This command string requests LINK to combine the object module MAIN.OBJ with the object modules SUB1.OBJ and SUB2.OBJ into the single core image file LOAD.SAV. A load map file MAP.MAP is also produced. All files are on the default device.

The switch, /F, specifies that the default FORTRAN Library on the system device, SY:FORLIB.OBJ, is to be searched for any routines that are not found in the other object modules. These include any library functions, system subroutines, or object time system routines. Note that the switch alone, without the explicit file specification, causes the default FORTRAN Library to be searched. This switch should be included if any of the object modules specified in the command string were created by the FORTRAN Compiler. This switch can be omitted, however, if the FORTRAN Library file specification, SY:FORLIB, is explicitly included in the command string as illustrated in the following example.

(RT-11)	(RSTS/E)
.R LINK	RUN $LINK
*LOAD,MAP=MAIN,SY:FORLIB	*LOAD,MAP=MAIN,SY:$FORLIB

The optional load map file specification, if included, requests the linker to output a list of module names, common blocks and global symbols together with their absolute memory address assignments.

For a more detailed description of LINK refer to the Linker chapter of the RT-11 System Reference Manual or the RSTS/E FORTRAN IV Utilities Manual. For an alternative procedure for invoking LINK under RSTS/E, read Chapter 5.

1.3.1 Library Usage

The FORTRAN IV programmer may want to create a library of commonly used assembly language and FORTRAN functions and subroutines. The system program LIBR provides a library creation and modification capability. The Librarian chapter of the RT-11 System Reference Manual or the RSTS/E FORTRAN IV Utilities Manual describes the LIBR program in detail.

A library file can be included in the LINK command string simply by adding the file specification to the input file list. LINK recognizes the file as a library file and links only those routines that are required. The LINK command string;

```
*LOAD=MAIN,LIB1/F
```

requests LINK to combine MAIN.OBJ with any required functions or subroutines contained in LIB1.OBJ. The default FORTRAN system library, FORLIB, is then searched for any other required routines. The entire memory image is output to the file LOAD.SAV.

If the /F switch is used, all user created libraries are searched before the default FORTRAN system library, FORLIB.

Under certain circumstances library directories are made memory resident to speed up library searches. This reduces the amount of memory available for the symbol table. Directories are made resident if 12K or more memory is available, there is room for the particular directory, and the /S linker switch is not included in the command string. If the linker fails because it ran out of symbol table space linking object files, one of which was a library file, and the above conditions were also in effect, another attempt can be made including the /S linker switch. This slows down the linking process, but allows the maximum possible symbol table space.

In the interest of maintaining the integrity of the DEC-distributed FORTRAN Library, the creation of a user library is preferred to the modification of, or addition to, the FORTRAN Library (FORLIB).

1.3.2 Overlay Usage

The overlay feature of the linker allows segmentation of the memory image so that the entire program need not be core resident at one time. This allows the execution of a program that otherwise would not fit in the available memory.

An overlay structure consists of a root segment and one or more overlay regions. The root segment contains the FORTRAN IV main program and blank COMMON. The root segment can also contain some subroutines and function subprograms. An overlay region is an area of memory allocated for two or more overlay segments, only one of which can be core resident at one time. An overlay segment consists of one or more subroutines or function subprograms.

At runtime, if a call is made to a routine that is contained in an overlay segment, the overlay handler checks to see if the segment is resident in its overlay region. If the segment is in memory, control is passed to the routine. If the segment is not resident, the overlay handler reads the overlay segment from the memory image file on the system device (or another device of the same type as the system device) into the specified overlay region. This destroys the previous overlay segment in that overlay region. Control is then passed to the routine.

When dividing a FORTRAN IV program into a root segment and overlay regions, and subsequently dividing each overlay region into overlay segments, routine placement should be carefully considered. The user should always remember that it is illegal to call a routine located in a different overlay segment in the same overlay region, or an overlay region with a lower numeric value (as specified by the linker overlay switch, /O:n), than the calling routine. The user should divide each overlay region into overlay segments that never need to be resident simultaneously.

The FORTRAN IV main program unit must be placed in the root segment.

In an overlay environment, subroutine calls and function subprogram references must refer only to one of the following:

1. A FORTRAN library routine (e.g. ASSIGN, DCOS).

2. A FORTRAN or assembly language routine contained in the root segment.

3. A FORTRAN or assembly language routine contained in the same overlay segment as the calling routine.

4. A FORTRAN or assembly language routine contained in a segment whose region number is greater than that of the calling routine.

In an overlay environment, COMMON blocks must be placed so that they are resident when referenced. Blank COMMON is always resident since it is always placed in the root segment. All named COMMON must be placed either in the root segment, or into the segment whose region number is the lowest of all segments that reference the COMMON block. A named COMMON block can not be referenced by two segments in the same region unless the COMMON block appears in a segment of the lower region number. The linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN IV main program or a subprogram that is located in the root segment. Otherwise, the linker places a COMMON block in the first segment encountered in the linker command string that references that COMMON block.

All COMMON blocks that are data initialized (by use of DATA statements) must be so initialized in the segment in which they are placed.

The entire overlay initialization process is handled by LINK. The command format outlined below, (and further explained in the Linker Chapter of the RT-11 System Reference Manual or the RSTS/E FORTRAN IV Utilities Manual) is used to describe the overlay structure to the linker. LINK combines the runtime overlay handler with the user program, making the overlay process completely transparent to the user's program.

The size of the overlay region is automatically computed to be large enough to contain the largest overlay segment in that overlay region.

The root segment and all overlay segments are contained in the memory image file generated by LINK.

Two switches are used to specify the overlay structure to LINK. The overlay switch is of the form:

```
/O:n
```

where n is an octal number specifying the overlay region number. The command continuation switch is of the form:

```
/C
```

This switch allows the user to continue long command strings on the next line of input.

The first line of the LINK overlay structure command string should contain, as the input list, all object modules that are to be included in the root segment. This line should be terminated with the /C and /F switches. The /O:n switch can not appear in the first line of the command string. If all modules which are to be placed in the root segment cannot be specified on the first command line, additional modules can be specified on subsequent command lines, each ending with

a /C. The entire root segment must be specified before any overlays.

All subsequent lines of the command string should be terminated with an /O:n switch specifying an overlay region and/or a /C switch. The presence of only a /C switch specifies that this is a continuation of the previous line and therefore, a continuation of the specification of that overlay segment. The object modules on each line, or set of continued lines, constitute an overlay segment and share the specified overlay region with all other segments in the same numeric value overlay region. All but the last line of the command string should contain the /C switch.

For example, given the following overlay structure description:

1. A main program and the object module SUB1 are to occupy the root segment.

2. The object module SUB2 is to share an overlay region with the object module SUB3 (never co-resident).

3. The object modules SUB4 and SUB5 are to share a second overlay region with the object modules SUB6 and SUB7.

the following command string could be used:

(RT-11)

```
.R LINK
*LOAD=MAIN,SUB1/F/C
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7
```

(RSTS/E)

```
RUN $LINK
*LOAD=MAIN,SUB1/F/C
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7
```

1.3.3 Stand-Alone FORTRAN

FORTRAN programs can be developed under the RT-11 or RSTS/E FORTRAN IV systems and output in an absolute binary format for execution on a satellite machine with minimum peripherals. The satellite machine can have a minimum of 4K words of memory and requires only a paper-tape reader for program loading.

When operating in the stand-alone environment, the only input/output device supported by FORTRAN-level input/output is the terminal. Other devices or equipment interfaces can be supported by appropriate user-written assembly language subroutines.

To generate a stand-alone program, the source program units should be compiled as usual. At link time, special options are specified to generate a stand-alone program. The /L switch must be included in the LINK command string to cause an absolute binary format (LDA) output file to be generated. The /I switch must also be given to allow two special modules to be requested from the FORTRAN IV Library. These two modules are:

$SIMRT	FORTRAN IV system simulator.
$nK	module which specifies the memory size of the target machine where n is 4,8,12,16,20,24, or 28.

Care must be taken to use the proper FORTRAN IV Library, as the library used must reflect the hardware arithmetic options available on the satellite machine. Hence, the system default library (SY:FORLIB.OBJ), which is used when the /F switch is specified to LINK, may not be appropriate. Consult with the system manager (RSTS/E users) or the Getting Started with RT-11 FORTRAN document (RT-11 users) for information on the various libraries.

The following command sequence generates a file LOAD.LDA, which can be punched on paper tape and loaded, using the Absolute Loader, on any PDP-11 with 8K or more of memory:

(RT-11)

```
.R LINK
*LOAD,LP:=MAIN,SUBS/F/L/I

LIBRARY SEARCH:
$SIMRT <CR>
$8K <CR>
<CR>

*↑C
.
```

(RSTS/E)

```
RUN $LINK
*LOAD,LP:=MAIN,SUBS/F/L/I

LIBRARY SEARCH:
$SIMRT <CR>
$8K <CR>
<CR>

*↑C
READY
```

1.4 EXECUTION PROCEDURES

1.4.1 Execution Under RT-11

To start execution of the memory image file generated by LINK, use the monitor R or RUN commands. The command:

```
.R FILESPEC
```

or

```
.RUN DEV:FILESPEC
```

causes the file on the system device, FILESPEC.SAV, which is the filename specification as described in Section 1.1.1, to be loaded into memory and executed.

The following example shows how to take three FORTRAN source files containing a main program and several subroutines through the procedures necessary to compile, link, and execute that program:

```
.R FORTRAN
*MAIN,LP:=MAIN,SUB
*SUB1,LP:=SUB1
*↑C
.R LINK
*MAIN,LP:=MAIN,SUB1/F
*↑C
.R MAIN
```

1.4.2 Execution under RSTS/E

To start execution of the memory image file generated by LINK, the following command sequence is used:

```
RUN $EXEC
*filespec/CORE:n
```

where

filespec	is the filename specification described in Section 1.1.1.
/CORE:n	is an optional switch specifying the maximum memory requirements for the executing program; n represents the number of K words of memory to allocate for the program.

If /CORE:n is not specified, a default value is used based on the high limit of the program as established by LINK. The high limit is the last entry in the link map. The default value is determined from Table 1-5. The default value allows sufficient memory for the default size record buffer (136 bytes; controlled by the /R switch to the compiler) and two file buffers. This allows a maximum of two non-terminal files to be open concurrently (terminal input/output requires no file buffer space).

The program to be run must have the compiled program (64 decimal) bit set in its protection code to be runnable by EXEC. This will happen automatically if the default extension, .SAV, is used for the LINK utiility. This will not be the case if another LINK extension is specified.

Table 1-5
Default Memory Allocation Values

IF HIGH LIMIT IS IN THE RANGE:, (octal)	THEN THE DEFAULT VALUE FOR N IS: (decimal)
000000 - 004000	3
004000 - 010000	4
010000 - 014000	5
014000 - 020000	6
020000 - 024000	7
024000 - 030000	8
030000 - 034000	9
034000 - 040000	10
040000 - 044000	11
044000 - 050000	12
050000 - 054000	13
054000 - 060000	14
060000 - 064000	15
064000 - 070000	16
070000 - 074000	17
074000 - 100000	18
100000 - 104000	19
104000 - 110000	20
110000 - 114000	21
114000 - 120000	22
120000 - 124000	23
124000 - 130000	24
130000 - 134000	25
134000 - 140000	26
140000 - 144000	27
144000 - 150000	28

If more than two files are to be open concurrently, or if a record buffer larger than the default is needed, the user can explicitly specify the /CORE:n switch. The value specified in the /CORE:n switch is determined by:

1. Establishing the default value.
   To establish the default value, ascertain the high limit of the program (this is the last entry in the link map), then search Table 1-5 for the appropriate default value.

2. Adding any additional space required for concurrently open files.
   If more than two files are to be open concurrently, additional space is added as illustrated in Table 1-6.

Table 1-6
Additional Run-Time Buffer Space

# OF FILES CONCURRENTLY OPEN (NOT INCLUDING TERMINAL)	VALUE TO ADD TO DEFAULT # FOR /CORE:n
0-2	0
3-6	1
7-10	2
11-14	3
15	4

3. Adding any additional space required for non-standard record buffers.
   If a record buffer larger than the default (136 bytes) has been allocated with the /R switch to the compiler, additional space is added as illustrated in Table 1-7.

Table 1-7
Additional Run-Time Record Buffer Storage

VALUE SPECIFIED FOR /R!m AT COMPILATION TIME	VALUE TO ADD TO DEFAULT # FOR /CORE;n
/R!4 - /R!1024	0
/R!1025 - /R!2048	1
/R!2049 - /R!3072	2
/R!3072 - /R!4095	3

The value computed for n in this manner is a close approximation to the minimum value required. The actual minimum may be less than the value computed, (by 1 or 2K).

The following sequence of commands illustrates the compilation, linking, and execution of a RSTS/E program (MAIN) contained in three source files:

```
RUN $FORTRAN
*MAIN,LP:=MAIN,SUB
*SUB1,LP:=SUB1
*^C

READY

RUN $LINK
*MAIN,LP:=MAIN,SUB1/F
*^C

READY

RUN $EXEC
*MAIN
```

Chapter 5 describes the CCL (Concise Command Language) option which

provides an alternative procedure for invoking FORTRAN IV system programs under RSTS/E.

1.5 DEBUGGING A FORTRAN IV PROGRAM

The debugging program, ODT, usually cannot be effectively used with a FORTRAN IV program due to the nature of the object code generated by the FORTRAN IV Compiler (see Section 2.2).

However, in addition to the FORTRAN OTS error diagnostics which include the Traceback feature (see Section 2.6), there is another debugging tool available to the FORTRAN programmer. The letter D in column one of a FORTRAN IV statement allows that statement to be conditionally compiled. These statements are considered comment lines by the FORTRAN IV Compiler unless the /D switch is used in the compiler command string. In this case, the lines are compiled as regular FORTRAN statements. Liberal use of PAUSE statements (see Section 3.5), and selective variable print out can provide the user with a method of monitoring program execution. This feature allows the inclusion of debugging aids that can be compiled in the early program development stages and later treated as regular comment lines.

CHAPTER 2

FORTRAN IV OPERATING ENVIRONMENT

2.1 FORTRAN IV OBJECT TIME SYSTEM

The FORTRAN IV Object Time System (OTS) is composed of the following:

1. math routines, including the FORTRAN IV library functions and other arithmetic routines (e.g., floating-point routines),
2. miscellaneous utility routines (ASSIGN, DATE, SETERR, etc.),
3. routines which handle various types of FORTRAN I/O,
4. error handling routines which process arithmetic errors, I/O errors, and system errors, and
5. miscellaneous routines required by the compiled code.

The FORTRAN library is designed as a collection of many small modules so that unnecessary routines can be omitted during linking. For example, if the user program performs only sequential-access, formatted I/O, none of the direct-access I/O routines is included in the executable program.

2.2 OBJECT CODE

Typical FORTRAN IV operations often require common sequences of PDP-11 machine instructions. For example, at the end of any DO-loop, the index variable must be incremented, compared with the limit value, and a conditional branch taken. Other standard sequences can be generated to locate an element of a multi-dimensional array, initialize an input/output operation, or simulate a floating-point operation not supported by the hardware configuration.

These common sequences of PDP-11 instructions are contained in a library known as the Object Time System. The FORTRAN compiler selects a certain combination of these instruction sequences to implement a FORTRAN program. During program execution, these sequences are threaded together and effect the desired result.

The compiler refers to a library instruction sequence by generating a word containing the address of the first instruction in the sequence, followed by information upon which the instructions are to operate. In the case of the end-of-DO-loop sequence, the information required is the location of the index variable, the limit value, and the address of the beginning of the loop. At runtime, register R4 is used to thread together the various references to library instruction sequences; the last instruction executed by each instruction sequence is JMP @(R4)+, which transfers control to the next library instruction sequence.

The mnemonics (global names) used for the library routine names follow a logically consistent format. The mnemonics are four to six characters in length. The first two characters specify an operation. The third character specifies the mode of the operation, i.e., integer, floating, double precision, complex, or logical. The fourth character is always a dollar sign ($). The fifth and sixth characters, if present, specify a source and destination for the operation, respectively. The source element for the operation can be a memory location, the hardware stack, the hardware registers, or an in-line argument which can be called through R4. The destination element for an operation can be a memory location, the hardware stack, or a location specified as an in-line argument which can be called through R4.

The library routines perform arithmetic operations, compare values, test values, calculate subscripts, convert from one mode type to another, and transfer program control. There are special routines to handle Internal Statement Numbers (ISNs), enabling the FORTRAN IV Traceback feature, a routine to handle subprogram control transfer, and a routine to push the address of variables on the hardware stack. There are also several routines to handle special FORTRAN runtime operations such as PAUSE, STOP, I/O initialization, and I/O data transfers.

For example, the following FORTRAN program:

```
FORTRAN IV      V01C-01     THU 13-NOV-75 16:25:38                PAGE 001

      C
      C     PROGRAM TO DEMONSTRATE THE CODE GENERATED BY THE
      C     FORTRAN IV COMPILER.
      C
0001        DIMENSION RARRAY(10,10)              !ALLOCATE A REAL*4 ARRAY
0002        DATA N/4/, I/2/                      !INITIALIZE N, I
0003        I = (3*2 - 5) + I                    !ADD ONE TO I
0004        J = (I+100)*(N**2)                   !COMPUTE AN EXPRESSION
0005        A = 2.0                              !ASSIGN A VALUE TO A
0006        RARRAY(2,1) = RARRAY(1,1) + A        !SUM OF TWO REAL VALUES
0007        END
```

generates object code that can be symbolically represented as follows (the storage map is included for reference):

```
FORTRAN IV      STORAGE MAP

NAME    OFFSET  ATTRIBUTES

RARRAY  000006  REAL*4    ARRAY (10,10) VECTORED
N       000626  INTEGER*2 VARIABLE
I       000630  INTEGER*2 VARIABLE
J       000656  INTEGER*2 VARIABLE
A       000660  REAL*4    VARIABLE
```

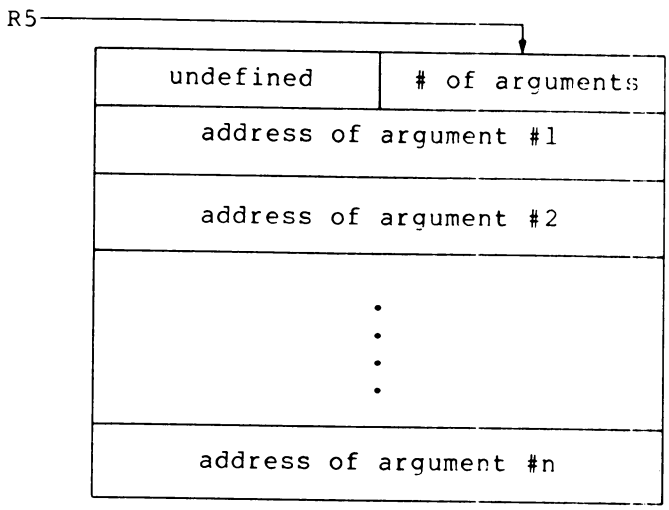
```
FORTRAN IV      GENERATED CODE


ISN #0003
000664  ICI$M   000630                  ;INCREMENT THE INTEGER WHOSE ADDRESS
                                        ;IS 000630 (I)


ISN #0004
000670  MOI$MS  000630                  ;MOVE THE VALUE OF INTEGER I ONTO STACK
000674  ADI$IS  #000144                 ;ADD 100 TO VALUE ON TOP OF STACK
000700  MOI$MS  000626                  ;MOVE THE VALUE OF INTEGER N ONTO STACK
000704  MUI$MS  000626                  ;AND SQUARE IT (MULTIPLY BY ITSELF)
000710  MUI$SS                          ;MULTIPLY (I+100) BY (N**2)
000712  MOI$SM  000656                  ;STORE VALUE ON TOP OF STACK INTO J

ISN #0005
000716  MOF$IM  #040400 000660          ;MOVE AN IMMEDIATE FLOATING CONSTANT
                                        ;(2.0) TO A

ISN #0006
000724  MOF$MM  000660  000012          ;MOVE RARRAY(1,1) TO RARRAY(2,1)
000732  ADF$MM  000006  000012          ;AND ADD A TO RARRAY(2,1)

ISN #0007
000740  RET$                           ;RETURN TO OPERATING SYSTEM
                                        ;(EXIT FROM PROGRAM)
```

2.3 SUBPROGRAM LINKAGE

All instances of subprogram linkage are performed in the same manner, including linkage of user-written FORTRAN IV subprograms, and assembly language subprograms. Control is passed to the subprogram via the following instruction:

```
        JSR     PC,routine
```

Register 5 (R5) contains the address of an argument list having the following format:



The value -1 is stored in the argument list as the address of any null arguments. Null arguments in CALL statements appear as successive commas, e.g., CALL SUB (A,,B)

Control is returned to the calling program via the instruction:

```
RTS     PC
```

As an example of argument transmission, an assembly language subroutine may be written to sum as many integer arguments as it finds in each parameter list, and transmit the result back to the FORTRAN IV program as the value of a final, additional argument. The FORTRAN CALL statements which invoke this routine would be of the form:

```
CALL IADD(num1,num2,...,numn,isum)
```

where num1 through numn represent a variable number of integer quantities to be summed, and isum represents the variable or array element into which the sum is to be placed.

Given the following MACRO-11 subprogram:

```
            .TITLE    ADDER
            .GLOBL    IADD
IADD:       MOV       (R5)+,R0        ;GET # OF ARGUMENTS
            CLR       R1              ;PREPARE WORKING REG.
            DECB      R0              ;FIND # OF TERMS TO ADD
1$:         ADD       @(R5)+,R1       ;ADD NEXT TERM
            DECB      R0              ;DECREMENT COUNTER
            BNE       1$              ;LOOP IF NOT DONE
            MOV       R1,@(R5)+       ;RETURN RESULT
            RTS       PC              ;RETURN CONTROL
```

the sequence of FORTRAN IV calls:

```
CALL IADD(1,5,7,I)
CALL IADD(15,30,10,20,5,J)
```

would cause the variable I to be given the value 13, and the variable J to be assigned the value 80.

2.4 SUBPROGRAM REGISTER USAGE

A subprogram that is called by a FORTRAN IV program need not preserve any registers. However, the contents of the hardware stack must be kept such that each 'push' onto the stack is matched by a 'pop' from the stack prior to exiting from the routine.

User-written assembly language programs that call FORTRAN IV subprograms must preserve any pertinent registers before calling the FORTRAN IV routine and restore the registers, if necessary, upon return.

Function subprograms return a single result in the hardware registers. The register assignments for returning the different variable types are listed below:

Integer and Logical functions - result in R0

Real functions - high order result in R0, low order result in R1

Double Precision functions - result in R0-R3, lowest order result in R3

Complex functions - high order real result in R0, low order real result in R1, high order imaginary result in R2, low order imaginary result in R3

In addition, assembly language subprograms, which use the FP11 Floating Point unit, may be required to save and restore the FPU status. FORTRAN IV assumes that the FPU status is set by default to:

- Short Floating mode (SETF)
- Short Integer mode (SETI)
- Floating Truncate mode

If the assembly language routine will modify these defaults, it must preserve the FPU status on entry by executing the following instruction:

```
STFPS   -(SP)
```

and restore the status (prior to returning to the calling program) by executing the instruction:

```
LDFPS   (SP)+
```

2.5 VECTORED ARRAYS

Array vectoring is a process which decreases the time necessary to reference elements of a multi-dimensional array by using additional memory to store the array.

Multi-dimensional arrays, which are actually stored sequentially in memory, require certain address calculations to determine the location of individual elements of the array. Typically, a mapping function is used to perform this calculation. For example, to locate the element LIST(1,2,3) in an array dimensioned LIST(4,5,6), a function equivalent

to the following can be used. This function identifies a location as an offset from the origin of the array storage.

```
(s1-1) + d1 * (s2 - 1) + d1 * d2 * (s3 - 1) =
(  0 ) + 4  * (   1  ) +  4 *  5 * (   2  ) = 44
```

where si = subscript i
di = dimension i

Since such a mapping function requires multiplication operation(s), and since some PDP-11 hardware configurations do not have the MUL instruction, the compiler may 'vector' some arrays and thereby reduce execution time at the expense of memory storage.

Since array vectors map only the declared dimensions of the array, the user must ensure that references to arrays are within their declared bounds. A reference outside the declared bounds of a vectored array causes unpredictable results (e.g. a program interrupt). The user should pay particular attention to arrays passed to subprograms where the dimensions declared in the subprogram differ from those specified in the calling program. In such cases, two sets of vectors are created: one for the calling program and one for the subprogram. The subprogram vectors map only that portion of the array declared by the subprogram. (The PDP-11 FORTRAN Language Reference Manual contains more information on dimensions.)

If an array is vectored, a particular element in the array can be located by a simplified mapping function, without the need for multiplication. Instead, a table lookup is performed to determine the location of a particular element. For example, a vectored, two-dimensional array B(5,5) automatically has associated with it a one-dimensional vector that would contain relative pointers to each column of array B. The location of the element B(m,n), relative to the beginning of the array, could then be computed as:

```
Vector(n) + m
```

using only addition operations. Figure 2-1 graphically depicts the array vectoring process.

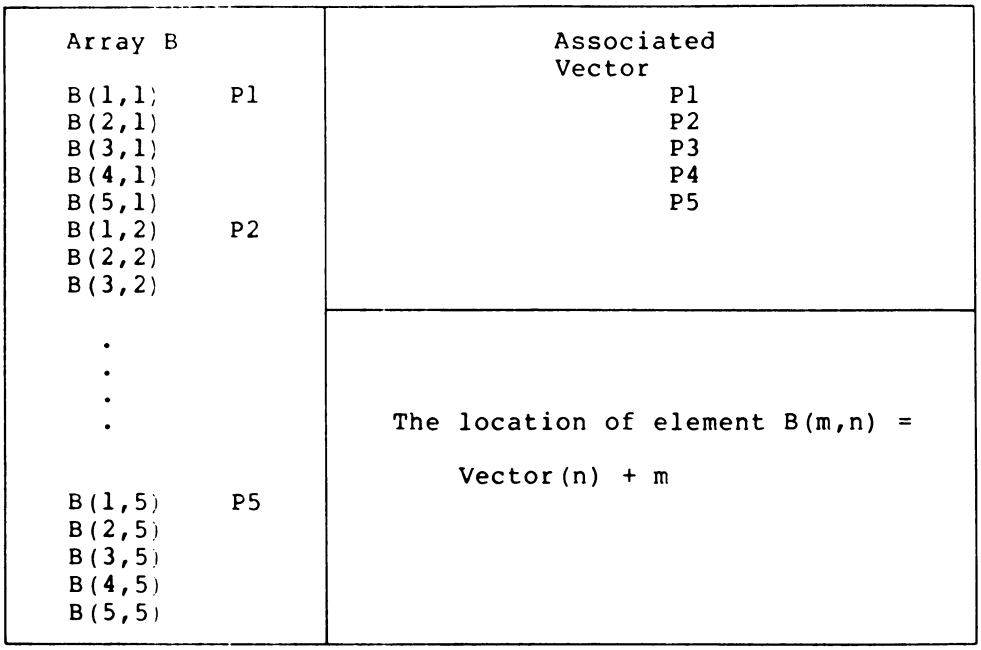


Figure 2-1 Array Vectoring

The compiler decides whether to vector a multi-dimensional array based on a ratio of the amount of space required to vector the array to the total storage space required by the array. If this ratio is greater than 25%, the array is not vectored and a standard mapping function is used instead. Arrays with adjustable dimensions are never vectored. Vectored arrays are noted as such in the storage map listing.

The compiler /V switch can be used to suppress all array vectoring.

The amount of memory required to vector an array can be computed as the sum of all array dimensions except the first. For example, the array X(50,10,30) requires 10+30=40 words of vector table. Note that the array V(5,100) requires 100 words of vector storage, whereas the array Y(100,5) requires only 5 words of vector storage. It is therefore advantageous to place an array's largest dimension first if it is to be vectored.

Wherever possible, vector tables are shared among several different arrays. The compiler arranges sharable vectors under the following conditions:

1. Arrays are in the same program unit.

2. For the ith dimension vector to be shared by the arrays, dimensions to the left of the ith dimension must be equivalent in each array.

For example, given the statement DIMENSION A(10,10),B(10,20), A and B share a 20 word vector for the second dimension that contains the values 0, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160, 170, 180, 190, of which the array A uses only the first ten elements.

2.6 TRACEBACK FEATURE

RT-11/RSTS/E FORTRAN IV fatal runtime errors include a traceback feature. This feature locates the actual program unit and line number of a runtime error. Immediately following the error message, the error handler lists the line number and program unit name in which the error occurred. If the program unit is a SUBROUTINE or FUNCTION subprogram, the error handler traces back to the calling program unit and displays the name of that program unit and the line number where the call occurred. This process continues until the calling sequence has been traced back to a specific line number in the main program. This allows an exact determination of the location of an error even if the error occurs in a deeply nested subroutine.

```
0001      A=0.0
0002      CALL SUB1(A)
0003      CALL EXIT
0004      END

0001      SUBROUTINE SUB1(B)
0002      CALL SUB2(B)
0003      RETURN
0004      END

0001      SUBROUTINE SUB2(C)
0002      CALL SUB3(C)
0003      RETURN
0004      END

0001      SUBROUTINE SUB3(D)
0002      E=1.0
0003      F=E/D
0004      RETURN
0005      END


          Trace back of Fatal Error:

                    ?ERR 12 FLOATING ZERO DIVIDE

                    IN   ROUTINE "SUB3  "   LINE 3
                    FROM ROUTINE "SUB2  "   LINE ?
                    FROM ROUTINE "SUB1  "   LINE 2
                    FROM ROUTINE ".MAIN."   LINE 2
```

Figure 2-2 The Traceback Feature

Note in Figure 2-2 that the line number in the traceback of routine 'SUB2' is simply a question mark (?). This is because the module was compiled with the /S switch in effect (see Section 1.2.1).

2.7 RUNTIME MEMORY ORGANIZATION (RT-11 only)

Figure 2-3 describes the runtime memory organization in both a USR (User Service Routines) swapping system and a USR resident system. If user-written interrupt handling routines are linked with a FORTRAN IV program, care should be taken to avoid USR swapping over the interrupt routines and any associated data. The USR is swapped in above the interrupt vectors at location 1000 (octal), and extends about 2K words from that point (see Figure 2-3). Interrupt routines must therefore be loaded above the area used for USR swapping if the USR will be actively swapped. The USR is actively swapped unless explicitly disabled by the compiler /U switch.

Some of the runtime memory segments are a fixed size. These include the resident monitor, the OTS work area, the stack and interrupt vector areas, and, in the USR resident system, the User Service Routine area.

Other runtime memory segments may vary in length. The user program, of course, is not always of the same length. The device handlers, channel tables, and I/O buffers are allocated space dynamically. Only those handlers that are needed for currently active devices are core resident, and I/O buffers are allocated and deallocated as required.

However, there is a maximum total amount of space that can be allocated to the varying length memory segments. In an 8K swapping system, approximately 5K words are available for these varying length segments. In an 8K resident system, approximately 3K words are available.

If a large FORTRAN IV program cannot be run in the amount of memory available, reducing the size of one of the varying length runtime memory segments can allow successful program execution. Use of overlay capabilities (see Section 1.3.2) can help reduce the amount of core needed for the user program. Minimizing the number of different physical devices used for I/O by the program can reduce the number of handlers that must be core resident. And finally, if the program closes a file by use of the CALL CLOSE routine (see Section B.4) as soon as it has finished I/O to that file, the buffer space allocated for that file can be deallocated and reused if a new file is subsequently opened.

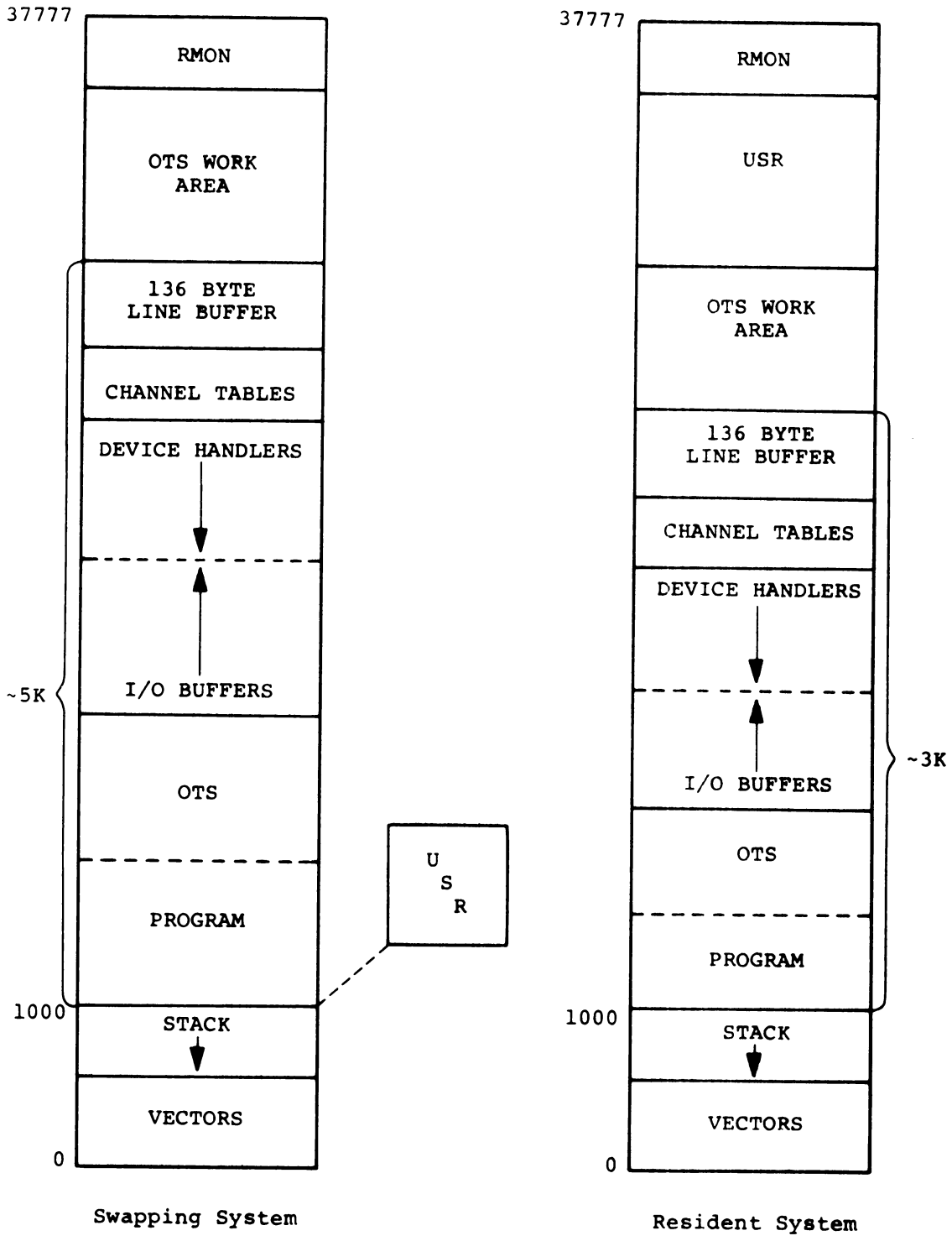


Figure 2-3 RT-11 8K System Runtime Memory Organization

CHAPTER 3

FORTRAN IV SPECIFIC CHARACTERISTICS

This chapter deals with information specific to RT-11 and RSTS/E FORTRAN IV that is omitted from, or relaxes restrictions included in, the PDP-11 FORTRAN Language Reference Manual.

It should be noted that deviations from FORTRAN syntax requirements outlined in the PDP-11 FORTRAN Language Reference Manual, even if acceptable in RT-11 and RSTS/E FORTRAN IV, decrease the portability of the program, and can prohibit successful execution on another system.

3.1 SOURCE LINES

A valid RT-11/RSTS/E FORTRAN IV source line consists of the following:

1. An optional, one to five character, numeric statement label, followed by

2. sufficient blanks to position the next character at column 7 (if not a continuation line) or column 6 (for continuations; a continuation signal character will be typed)

   or

   a tab character followed by any non-alphabetic character to signal continuation,

   or

   a tab character, if the line is not a continuation,

3. a valid FORTRAN statement, or the continuation of a statement, and

4. an optional comment field delimited on the left by an exclamation point (!).

Totally blank records (a source line of only a carriage return-line feed combination) are ignored on input. If a line is not totally blank, it must contain some portion of a FORTRAN statement.

3.2 VARIABLE NAMES

RT-11/RSTS/E FORTRAN IV allows variable names to exceed six characters. However, only the first six characters are significant and should be unique among all variable names in the program unit. A warning diagnostic is given for each variable name which exceeds six characters in length. The diagnostic is enabled if /W is included in the compiler command string.

3.3 INITIALIZATION OF COMMON VARIABLES

RT-11/RSTS/E FORTRAN IV allows any variables in COMMON, including blank COMMON, to be initialized in any program unit by use of the DATA statement.

3.4 CONTINUATION LINES

A line is assumed to be a continuation line if the first character, following a tab on an input line to the compiler, is non-alphabetic.

RT-11/RSTS/E FORTRAN IV does not place any limits on the number of continuation lines that a statement may contain.

3.5 STOP AND PAUSE STATEMENTS

Execution of a program may be temporarily suspended by the use of the PAUSE statement. When this occurs, the word PAUSE and the contents of the text string (if any) are typed on the user's terminal. To continue program execution, type a carriage return.

For example, use of the FORTRAN statement:

```
PAUSE 'MOUNT A NEW TAPE'
```

causes the following line to be printed at the user's terminal:

```
PAUSE -- MOUNT A NEW TAPE
```

Execution of the STOP statement closes all files and returns control to the operating system. To terminate a program execution without printing the STOP statement, a CALL EXIT routine should be used, (see Section B.7).

The STOP statement causes the following output to be printed:

```
STOP -- text
```

where text is the optional text string from the source statement.

3.6 DEVICE/FILE DEFAULT ASSIGNMENTS

Listed in Table 3-1 are the device and filename default assignments. The default device assignments can be changed prior to execution by use of the monitor ASSIGN command. For example, the monitor command:

(RT-11)	(RSTS/E)
.ASSIGN LP:7	ASSIGN LP:7

connects logical unit 7 to a physical device, the line printer. The device and/or filename assignments can be changed at execution time by use of the ASSIGN system subroutine (see Section B.2). Valid logical unit numbers other than those listed below (10-99) are assigned to the system default device. The default filename conventions hold for logical units not listed below, e.g., unit number 49 will have a default filename of FTN49.DAT. For more information, the user can refer to the RT-11 System Reference Manual, Section 2.7.2.4, or the RSTS/E System User's Guide, Section 2.6.3.

Table 3-1
FORTRAN Logical Device Assignments

Logical Unit Number	Default Device	Default Filename
1	System disk, SY:	FTN1.DAT
2	Default Device	FTN2.DAT
3	Default Device	FTN3.DAT
4	Default Device	FTN4.DAT
5	Terminal, TT:(Input)	FTN5.DAT
6	Line printer, LP:	FTN6.DAT
7	Terminal, TT:(Output)	FTN7.DAT
8	High-speed paper tape reader, PR:	FTN8.DAT
9	High-speed paper tape punch, PP:	FTN9.DAT

Although any combination of valid logical unit numbers can be used, there is an imposed maximum number of units that can be simultaneously active. By default, six logical units can be concurrently active. The number can be changed by use of the /N switch in the compiler command string while compiling the main program unit (see Section 1.2.1).

A formatted READ statement of the form:

```
READ f,list
```

is equivalent to:

```
READ(1,f)list
```

For all purposes these two forms function identically. For example, assigning logical unit number 1 to the terminal, in both cases, causes input to come from the terminal.

The ACCEPT, TYPE, and PRINT statements also have similar functional

analogies. Assigning devices to logical units 5, 7, and 6 affects respectively the ACCEPT, TYPE, and PRINT statements.

3.7 STATEMENT ORDERING RESTRICTIONS

RT-11/RSTS/E FORTRAN IV does not impose as strict statement ordering requirements as those outlined in the PDP-11 FORTRAN Language Reference Manual. There are only three statement ordering requirements that must be met:

1. In a subprogram, the first non-comment line must be a function, subroutine or block data statement.

2. The last line in a program unit must be an END statement.

3. Statement functions must be defined before they are referenced.

However, if the statement ordering requirements as outlined in the PDP-11 FORTRAN Language Reference Manual are not followed and if the /W compiler switch (enable warning diagnostics) is included in the compiler command string, a warning diagnostic is included with the source listing.

3.8 MAXIMUM RECORD LENGTHS

The line buffer allocated to temporarily store I/O records is by default 136 bytes. This restricts all I/O records in formatted I/O statements and ENCODE and DECODE statements to a maximum of 136 characters. The size of this buffer, and consequently the maximum record length, can be changed by including the /R switch in the compiler command string while compiling the main program unit. The maximum size of the line buffer is 4095 bytes (7777 octal).

3.9 DIRECT-ACCESS I/O

RT-11/RSTS/E FORTRAN IV allows creation and modification of direct-access files.

3.9.1 DEFINE FILE Statement

The first parenthesized argument in a DEFINE FILE statement specifies the length, in records, of the direct-access file being initialized. However, if the statement is part of a file creation procedure, this value may not be readily available. RT-11/RSTS/E FORTRAN IV allows some extra flexibility in this situation. Under RT-11, a file length specification of zero records causes a large contiguous file to be allocated initially and the unused portion to be automatically de-allocated when the file is closed. The "END=" construction is particularly useful in this situation for determining the actual length of the file.

Under RSTS/E a file length specification of zero records causes the

file to be extended dynamically as required by the highest record number referenced during program execution, if the record size is an exact multiple or divisor of 256.

3.9.2 Creating Direct-Access Files

The first I/O operation performed on a direct-access file during file creation must be a WRITE operation. A READ or FIND operation under such circumstances produces a fatal error condition.

3.10 INPUT/OUTPUT FORMATS

RT-11/RSTS/E FORTRAN IV allows formatted input and output for transferring of ASCII files. Unformatted and direct-access input and output are available for transferring binary records.

Some run-time errors can be intercepted and control transferred to a pre-determined program label by use of the ERR= parameter. This parameter can be specified in the READ, WRITE, ENCODE or DECODE statements. Note that a count n error will become fatal on the nth occurrence of the error.

The following errors can be intercepted:

ERROR NUMBER	ERROR TYPE	MESSAGE
5	Count 3	Input conversion error
23	Fatal	Hardware I/O error
45	Fatal	Incompatible variable and format
46	Fatal	Infinite format loop

3.10.1 Formatted I/O

The formatted input/output routines read or write variable-length, formatted ASCII records. A record consists of a number of ASCII characters, transmitted under control of a format specification, followed by a record separator character(s).

On input, the parity bit of each input character is removed, (set to zero); only the 7-bit ASCII character is transferred. On output, the parity bit is written as a zero.

On output to a printing device (KB:, TT:, or LP:), the record separator appended to each record consists of a carriage return character. The carriage return can be suppressed by use of the "$" format separator character in the FORMAT statement (see Chapter 5 of the PDP-11 FORTRAN Language Reference Manual). The first character of each record is deleted from the record and is interpreted as a carriage control character.

The control characters inserted are:

FIRST CHARACTER OF RECORD	CONTROL CHARACTERS OUTPUT
'+'	none
'b' (space)	one LF (line feed)
'0' (zero)	two LF's
'1' (one)	one FF (form feed)
any other character	one LF

The selected control characters replace the first character of the record.

On output to a non-printing device (i.e., disk file), each record is preceeded by a line feed character, and followed by a carriage return character. An additional line feed character is inserted at the end of the file when it is closed (if the file was being created). No translation of carriage control information occurs, (i.e., the first character of the record is written, as given, to the file). If translation is desired, the 'CC' argument should be specified in an ASSIGN or OPEN system subroutine call (see Sections B.2 and B.3).

The maximum length of a formatted record (including any leading or trailing carriage control information) is determined by the record buffer length, which is set by the /R switch to the compiler (see Section 1.2.1).

3.10.2 Unformatted I/O

The unformatted input/output routines read or write variable-length, binary records with additional control and positioning information. The control information is added to allow file positioning through the auxiliary I/O statements (BACKSPACE, etc.). The internal structure of these files is unique to FORTRAN. Therefore, if a file is to be read or written by a program composed in another language, formatted I/O should be used to convey ASCII data and direct-access I/O should be used for binary data.

3.10.3 Direct-Access I/O

The direct-access input/output routines read or write fixed-length, binary records. The logical record structure for a direct-access file is determined by the DEFINE FILE statement. The records contain only the specified data; no control information or record separators are used.

The direct-access record structure is independent of the physical block size of the I/O device. However, more efficient operation results if the record size is an exact divisor or multiple of 256 words.

3.11 MIXED MODE COMPARISONS

When comparing a single precision number to a double precision number, the double precision number may appear to be greater than the single precision number in magnitude even if they should be equal. For example:

```
DOUBLE PRECISION D

A=55.1

D=55.1D0

IF(A.LT.D)STOP
```

In the example above A compares less than D. This is due to the fact that 55.1 is a repeating binary fraction. Before the comparison, the 24 bit fractional (mantissa) part of A is extended with 32 zero bits. These low order 32 bits are now less than the low order 32 bits of D, and D therefore compares greater than A.

CHAPTER 4

INCREASING FORTRAN IV PROGRAMMING EFFICIENCY

4.1 FACTORS AFFECTING PROGRAM EFFICIENCY

The purpose of this chapter is to provide information helpful to the programmer who is interested in minimizing program execution time or storage space requirements.

The relative efficiency of an RT-11/RSTS/E FORTRAN IV object program depends on several factors. These factors fall into two classes:

1. The way in which the programmer codes the source program, and

2. the way in which the compiler treats the source program.

These two factors can be interrelated. The effectiveness of compiler optimizations can, in some cases, be increased by certain programming techniques in the source program. Also, awareness on the part of the programmer of which FORTRAN constructs are handled most efficiently by the compiler can be utilized when coding the source program.

Section 4.2 deals with the situations in which the compiler generates the most efficient code. Section 4.3, Programming Techniques, contains hints on improving programming efficiency.

Each topic discussed in the following section is flagged with one of the following remarks:

(space)	indicates that the primary function of the discussion is to minimize program memory requirements.
(time)	indicates that the primary concern is minimization of execution time.

A particular topic can have both designations, indicating a savings in both space and time.

4.2 INCREASING COMPILATION EFFECTIVENESS

The following 12 programming suggestions will increase compilation effectiveness.

1. Using the Optimizer effectively (space,time)

   Avoiding certain programming constructs allows the optimizer

greater freedom to discover common subexpressions in source programs. Specifically, the following situations should be avoided:

- Unnecessary statement labels (labels which are never referenced).

- Usage of equivalenced and COMMON variables, and SUBROUTINE and FUNCTION dummy arguments.

- Usage of arithmetic IF statements which could be logical IF's. For example:

```
      IF (A - 7.0) 100,200,100

200   CONTINUE
```

  can be written as an equivalent logical IF:

```
      IF (A .NE.  7.0) GOTO 100
```

  and allows the label "200" to be removed from the program.

2. Passing arguments to subprograms (space, time)

   To minimize overhead in FUNCTION and SUBROUTINE calls, parameters should be passed in COMMON blocks rather than standard argument lists. Variables in COMMON are handled as efficiently as local variables.

   Minimizing the number of elements in the argument list (by placing others in COMMON) reduces the time required to execute the transfer of control to the called routine.

3. Statement functions (time)

   Arithmetic and logical statement functions are implemented as internal FUNCTION subprograms. Hence, all suggestions concerning argument lists apply to statement functions also.

4. Minimizing array vector table storage (space)

   The RT-11/RSTS/E FORTRAN IV array vectoring feature is designed to decrease the time required to compute the address associated with an element of a multi-dimensional array by precomputing certain of the multiplication operations involved. The values precomputed are stored in a table called the "vector" for the particular array dimension. It is desirable to minimize the space allocated to these vectors.

   The following steps can be taken by the programmer to reduce the space required for array vectors:

   - Specify the largest dimensions first in the statement which allocates the array. This minimizes the number of vector table entries. as the first dimension is never vectored. For example,

     INTEGER A(350,10) requires 10 words to vector

INTEGER A(10,350) requires 350 words to vector

The compiler computes a space tradeoff factor which relates the number of words required for vector storage to the number of words required to store the array. If this tradeoff is favorable (i.e., the vector table is small compared to the array), the array is vectored. Therefore, the proper ordering of dimensions not only saves table space for all vectored arrays, but can also cause other arrays to be made eligible for vectoring.

- Try to keep similar arrays dimensioned in the same order. This will cause certain arrays to share vector tables. For example:

  INTEGER A(9,4,5), B(9,4,7), C(9,8)

  all share the same two vectors, one for the second array dimension and one for the third. The vector for the second dimension will have eight elements (@ 1 word each) because C has the largest second dimension, 8. Similarly, the vector for the third dimension has seven elements.

  In the general case, two arrays share a vector table for dimension i if each dimension less than i in each array is identical to the same dimension for the other array. In the example given above, arrays A, B, and C share the vector for the second dimension because each array has a first dimension equal to 9.

- Vectoring can be disabled completely by specifying the /V switch in the command string to the compiler. This causes no vector tables to be generated, but the resulting program executes slower than with vectoring. This tradeoff can be made if array usage is not heavy in speed-critical sections of the program, or if space is the primary goal.

5. Multi-Dimensioned array usage (time)

When using multi-dimensional arrays, the number of specified variable subscripts affects the time required to make the array reference. Therefore, the following steps can be taken to optimize array references:

- Use arrays with as few dimensions as possible.

- Use constant subscripts whenever possible. Constant subscripts are computed during compilation and require no extra operations at execution time.

- Make totally constant array references wherever appropriate. These references receive the highest level of optimization. For example,

  ```
  I = 1
  A(I) = 0.0
  ```

  is not as efficient as

```
I = 1
A(1) = 0.0
```

The former case requires a runtime subscript operation; in the latter, the compiler can calculate the address of the first element of array A at compilation time.

6. Formatted input/output (space,time)

RT-11/RSTS/E FORTRAN IV precompiles and compacts FORMAT statements that are presented in the source program. This affects the space required to store the format at runtime, and the speed of the input/output operations which make use of the format.

For this reason, object-time formats (i.e., those formats that are specified in arrays rather than as FORMAT statements) are considerably less efficient.

7. Data type selection (space,time)

Due to the addressing modes of the PDP-11 processors and various optimization considerations internal to FORTRAN IV, more efficient code can be generated for certain data types than for others. Specifically:

- Use the INTEGER data type wherever possible. RT-11/RSTS/E FORTRAN IV performs extensive optimizations on this data type.

- Use REAL*4 rather than DOUBLE PRECISION (REAL*8) wherever possible. Single-precision operations are significantly faster than double-precision, and storage space is saved.

- Avoid unnecessary mode-mixing. For example:

  ```
  A = 0.0
  ```

  is preferable to

  ```
  A = 0
  ```

- Use two REAL*4 variables rather than a COMPLEX*8 if usage of COMPLEX variables in the program is not heavy. REAL*4 operations receive more optimization than COMPLEX operations.

8. Testing "flag" variables (space)

Wherever possible, comparisons with zero should be used. Comparing any data type to a zero value is a special case which requires less executable code. An example of such a case is the following:

```
IF (I .LT.  1) GOTO 100
```

requires more code than,

```
IF (I .LE.  0) GOTO 100
```

9. *2, **2 Operations (time)

Explicitly specifying *2 when doubling the value of an expression, or **2 when squaring the value of an expression can lead to more efficient code. For example:

```
A = (B + ARRAY(C))**2
```

is preferable to

```
A = (B + ARRAY(C)) * (B + ARRAY(C))
```

despite the fact that (B + ARRAY(C)) is computed only once in either case. Note that this applies only to expression values; I**2 is as efficient as I*I.

10. Assigned GOTO usage (time)

The implementation of the assigned GOTO operation in RT-11/RSTS/E FORTRAN IV is extremely efficient in space and time.

For instance, internal subroutines can be constructed by assigning a "return label" to an integer variable and executing a GOTO to the first statement in the routine. The routine returns control to the appropriate point by doing an assigned GOTO through the previously mentioned integer variable. However, using this feature may make the program difficult to debug.

The following usage of an external subroutine with no arguments can be recoded to use assigned GOTO:

```
          .
          .
          .
I=5               SUBROUTINE SUBR
J=23                    .
                        .
                        .
CALL SUBR         N=I*SQRT (FLOAT(J))
                        .
I=18                    .
                        .
J=409             RETURN
CALL SUBR
          .
          .
          .
```

When assigned GOTO is used, the statements of the subroutine are inserted into the body of the program and the CALL statements are replaced by GOTO statements, as follows:

```
        .
        .
        .
        I=5
        J=23
        ASSIGN 100 TO IRET          !SET RETURN ADDRESS
        GOTO  1000                  !AND JUMP TO SUBR CODE
100     I = 18
        J = 409
        ASSIGN 200 TO IRET          !SET  NEW  RETURN  ADDRESS
        GOTO 1000
200     .
        .
1000    .                           !CODE FROM SUBROUTINE SUBR
        .
        .
        .
        N=I*SQRT (FLOAT(J))
        .
        .
        .
        GOTO IRET                   !RETURN TO CALLER
```

11. Compilation switches (space)

To minimize the space required for program execution, the following switches should be supplied to the compiler:

/S to suppress line number traceback
/V to suppress all array vectoring

In addition, the /T (two-word integer default) switch should not be specified unless required. The specification of the /P (disable optimizer) switch can reduce storage requirements, but this is not predictable from a simple consideration of the source program. Therefore, both settings of the /P switch should be tried to determine which is optimal.

The /U switch should not be specified if it is not required (i.e., no user-written interrupt or completion routines exist

in the linked program).

Also, to conserve space at execution time, minimal values for the /N and /R compiler switches should be specified. The /N switch value should be the number of logical units that can be concurrently active. The /R switch should be set to the maximum formatted record length plus two (for the carriage return-line feed combination which can accompany a record).

12. Compilation switches (time)

When optimizing an object program for execution time, the following compiler switches should be specified:

/S to suppress line number traceback
/U to prevent the USR (RT-11 file service routines) from swapping at runtime (RT-11 only; ignored under RSTS/E)

In addition, since global optimization and array vectoring speed program execution, the following switches should not be specified:

/P will disable global optimizer
/V will disable array vectoring

4.3 PROGRAMMING TECHNIQUES

In the following eight examples, a comparison is made between different programming methods. These comparisons show more efficient programming techniques available to the user. While both methods are correct for the particular operation, the technique on the right has been found more efficient than the technique on the left.

1. Make use of the increment parameter in DO loops:

INEFFICIENT

```
      DIMENSION A(20)
      DO 100 I=1,10
      A(2*I)=B
100   CONTINUE
```

EFFICIENT

```
      DIMENSION A(20)
      DO 100 I=2,20,2
      A(I)=B
100   CONTINUE
```

In the inefficient example, an additional calculation is performed (i.e., 2*I) each time through the loop. These calculations are avoided in the efficient example by having the count incremented by two.

2. Avoid placing calculations within loops whenever possible:

INEFFICIENT

```
      DO 10 I=1,20
      DO 20 J=1,50
20    A(J)=A(J)+I*B*C
10    CONTINUE
```

EFFICIENT

```
      TEMP1=B*C
      DO 10 I=1,20
      TEMP2=I*TEMP1
      DO 20 J=1,50
20    A(J)=A(J)+TEMP2
10    CONTINUE
```

The calculation (B*C) within the loop of the inefficient example is evaluated 1000 times. Calculations are handled more economically when done outside the loop. In the efficient example, 980 "FLOATS" and 1979 floating multiplies were saved by performing the (B*C and I) calculations outside the loop.

3. Proper nesting of DO loops can increase speed by minimizing the loop initialization.

INEFFICIENT

```
   DIMENSION A(100,10)
   DO 60 I=1,100
   DO 60 J=1,10
60 A(I,J)=B
```

EFFICIENT

```
   DIMENSION A(100,10)
   DO 60 J=1,10
   DO 60 I=1,100
60 A(I,J)=B
```

In the first example, the inner DO loop is initialized 100 times, while in the efficient example it is only initialized 10 times.

4. The most efficient way to zero a large array, or to set each element to some value, is to equivalence it to a single-dimension array. This technique is even useful for copying large, multi-dimensional arrays.

INEFFICIENT

```
   INTEGER A(20,100)
   DO 20 I=1,100
   DO 20 J=1,20
20 A(J,I)=0
```

EFFICIENT

```
   INTEGER A(20,100),ATEMP(2000)
   EQUIVALENCE (A,ATEMP)
   DO 20 I=1,2000
20 ATEMP(I)=0
```

MORE EFFICIENT

```
   INTEGER A(20,100)
   REAL*8 ATEMP(500)
   EQUIVALENCE (A,ATEMP)
    DO 20 I=1,500
20 ATEMP(I)=0.0D0
```

In the efficient example, an EQUIVALENCE is used to place two arrays at the same point in memory, thus avoiding a subscript calculation each time through the loop. The more efficient example makes use of the 8 bytes in REAL*8 and, by equivalencing, places 4 integers in the array and zeroes them in one operation, thus quartering the number of iterations.

5. Avoid implied data conversions by using constants of the appropriate type.

INEFFICIENT

```
   DOUBLE PRECISION D
   REAL A
 4 A=A+1
   IF (A.EQ.0)GOTO 4
   D=3.14159
```

EFFICIENT

```
   DOUBLE PRECISION D
   REAL A
 4 A=A+1.
   IF (A.EQ.0.0)GOTO 4
   D=3.14159D0
```

Since a library routine must be called to perform a data conversion, use of the correct constant improves the size and

execution speed of the program.

6. Do as much calculation in INTEGER mode as possible.

INEFFICIENT	EFFICIENT
A=B+I+J	A=B+(I+J)

Also, do as much calculation in REAL mode when the dominant mode of an expression is DOUBLE PRECISION or COMPLEX. Calculation is most efficient in integer mode, less efficient in REAL mode, and least efficient in DOUBLE PRECISION or COMPLEX. Remember, in the absence of parentheses, evaluation generally proceeds from left to right.

7. Use COMMON to pass arguments and return results of subprograms whenever possible.

INEFFICIENT	EFFICIENT
	COMMON/SUBRA/A,B,C,D,E COMMON/FUNCTA/Y,Z . . .
CALL SUBR(A,B,C,D,E)	CALL SUBR
X=FUNCT(Y,Z)	X=FUNCT()
CALL SUBR(A,B,C,D,E)	CALL SUBR
. . .	. . .
END	END
SUBROUTINE SUBR(A,B,C,D,E)	SUBROUTINE SUBR
	COMMON/SUBRA/A,B,C,D,E
. . .	. . .
END	END
FUNCTION FUNCT(Y,Z)	FUNCTION FUNCT
	COMMON/FUNCTA/Y,Z
. . .	. . .
END	END

COMMON is handled more efficiently than formal argument lists. Generally, it is possible to use COMMON for argument passage if a subprogram is referenced from only one place, or if it is always referenced with the same actual arguments.

NOTE

In PDP-11 FORTRAN IV, function subprograms are not required to have arguments.

8. Avoid division within programs wherever possible.

INEFFICIENT	EFFICIENT
A=B/2.	A= B*.5

Multiplication is faster than division and thus saves execution time.

CHAPTER 5

CONCISE COMMAND LANGUAGE OPTION

5.1 INTRODUCTION TO THE RSTS/E FORTRAN IV CCL OPTION

The Concise Command Language (CCL) commands provide an alternative method for invoking RSTS/E system programs. CCL commands allow a user to run a system program by specifying a single command for the program to execute. The user types the CCL command and the program command on one line and enters it to the system. The system loads the program into the user's job area and writes the program command to the core common area. This operation destroys the current contents of the user's job area. The program runs, reads the command from the core common area, and executes. If an error is encountered, the program prints a related message and terminates. CCL options are available for the following system programs: FORTRAN, LINK, MACRO, and EXEC.

The CCL option must have been installed at the time of RSTS/E System Generation. RSTS/E users should contact the system manager for the availability of this option.

5.2 COMMAND INTERFACE

The CCL command to invoke the FORTRAN IV Compiler has the form:

```
FOR command line
```

where

command line — has the form: output = input/sw
The output and input filename specifications are described in Section 1.1.1; the compiler switches are described in Section 1.2.1.

The command to invoke the linker, LINK, has the form:

```
LINK command line
```

where

command line — has the form: output = input/sw
The output and input filename specifications and switch options are described in Section 1.3.

The command to invoke MACRO has the form:

MACRO command line

where

command line — has the form: output = input/sw
The output and input filename specifications and switch options are described in Section 2.7 of the RSTS/E FORTRAN IV Utilities Manual and Section 5.7 of the RT-11 System Reference Manual.

The command to invoke EXEC has the form:

EXEC command line

where

command line — has the form: filename specification/switch
The filename specification and switch option are described in Section 1.4.2.

5.2.1 CCL Command Restrictions

Several switch options included in the LINK utility are not acceptable to the LINK CCL command line. These switch option restrictions do not apply to the "RUN $LINK" invocation of the linker utility but only to one line of input to the LINK CCL command. The restricted switches are the following:

- /C — continue input specification on multiple lines
- /I — include requested library modules
- /M — specify stack address as global symbol (/M:n form is acceptable)
- /O — indicate overlay structure
- /T — specify transfer address as global symbol (/T:n form is acceptable)

5.2.2 CCL Command Comparison

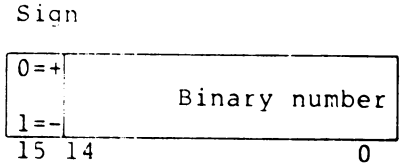
The following example illustrates the two methods available to the user for creating a source and assembly program, as well as linking and execution.

RSTS/E Command String	CCL
RUN $FORTRAN *MAIN=MAIN,SUBR/S *^Z	FOR MAIN=MAIN,SUBR/S
READY	READY
RUN $MACRO *MACSUB=MACSUB ERRORS DETECTED:0 FREE CORE: 1024 WORDS *^Z	MACRO MACSUB=MACSUB ERRORS DETECTED:0 FREE CORE: 1024 WORDS
READY	READY
RUN $LINK *PROG=MAIN,MACSUB/F *^Z	LINK PROG=MAIN,MACSUB/F
READY	READY
RUN $EXEC *PROG	EXEC PROG

APPENDIX A

FORTRAN DATA REPRESENTATION

A.1 INTEGER FORMAT



Integers are stored in a two's complement representation. If the /T compiler switch (see Section 1.2.1) is used, an integer is assigned two words, although only the high order word (i.e., the word having the lower address) is significant. By default, integers will be assigned to a single storage word. Explicit length integer specifications (INTEGER*2 and INTEGER*4) will always take precedence over the setting of the /T switch. Integer constants must lie in the range -32767 to +32767. For example:

```
+22 = 000026(octal)
 -7 = 177771(octal)
```

A.2 FLOATING-POINT FORMATS

The exponent for both 2-word and 4-word floating-point formats is stored in excess 128 (200(octal)) notation. Binary exponents from -128 to +127 are represented by the binary equivalents of 0 through 255 (0 through 377(octal)). Fractions are represented in sign-magnitude notation with the binary radix point to the left. Numbers are assumed to be normalized and, therefore, the most significant bit is not stored because of redundancy (this is called hidden bit normalization). This bit is assumed to be a 1 unless the exponent is 0 (corresponding to 2-128) in which case it is assumed to be 0. The value 0 is represented by two or four words of zeros. For example, +1.0 would be represented by:

```
40200
0
```

in the 2-word format, or:

```
40200
0
0
0
```

in the 4-word format. -5 would be:

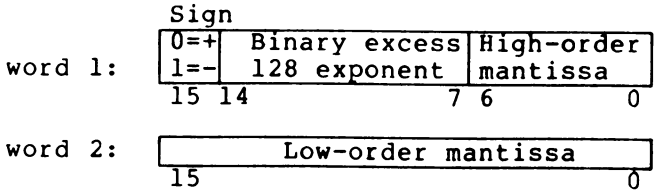
```
140640
0
```

in the 2-word format, or:

```
140640
0
0
0
```

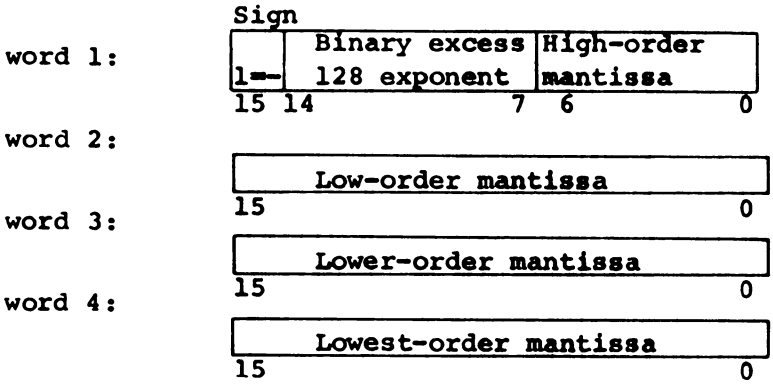
in the 4-word format.

A.2.1 REAL Format (2-Word Floating Point)



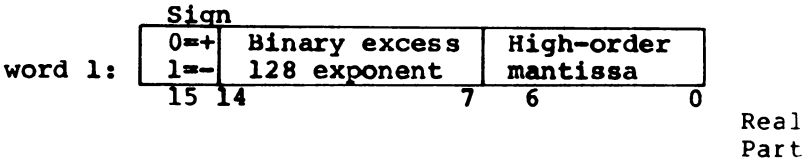
Since the high-order bit of the mantissa is always 1, it is discarded, giving an effective precision of 24 bits (or approximately 7 digits of accuracy). The magnitude range lies between approximately .29 X 10^{-38} and .17 X 10^{39}.

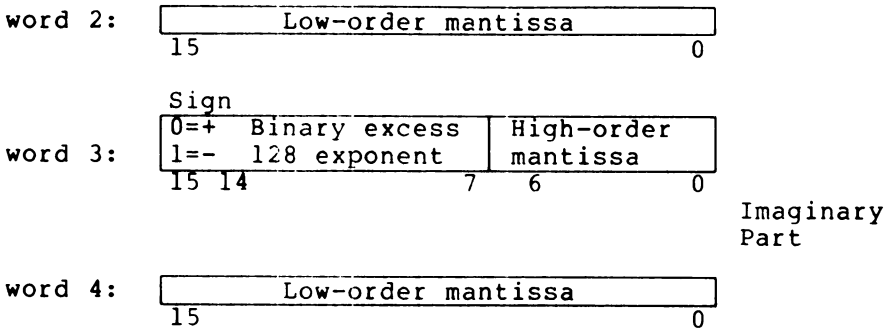
A.2.2 DOUBLE PRECISION Format (4-Word Floating Point)



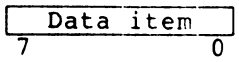
The effective precision is 56 bits (or approximately 17 decimal digits of accuracy). The magnitude range lies between .29 X 10^{-38} and .17 X 10^{39}.

A.2.3 COMPLEX Format



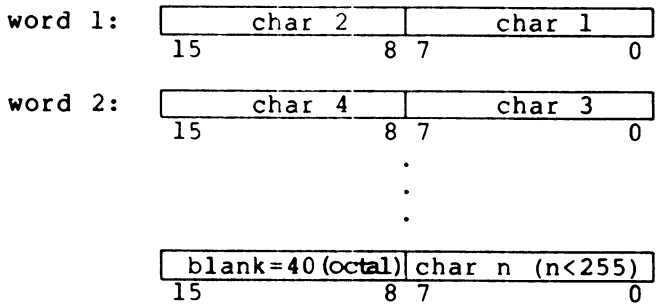


A.3 LOGICAL*1 FORMAT



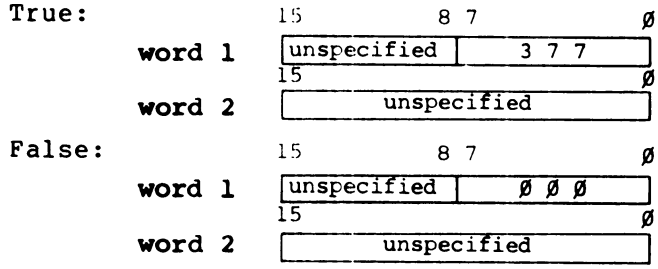
Any non-zero value is considered to have a logical value of .TRUE. The range of numbers from +127 to -128 can be represented in LOGICAL*1 format. LOGICAL*1 array elements are stored in adjacent bytes.

A.4 HOLLERITH FORMAT



Hollerith constants are stored internally, one character per byte. Hollerith values are padded on the right with blanks to fill the associated data item, if necessary. Hollerith constants can only be used in DATA, FORMAT, and CALL statements. Only the quoted form of Hollerith constants can be used in STOP and PAUSE statements.

A.5 LOGICAL FORMAT



Logical (LOGICAL*4) data items are treated as LOGICAL*1 values for use with arithmetic and logical operators. Any non-zero value in the low order byte is considered to have a logical value of true in logical expressions.

A.6 RADIX-50 FORMAT

Radix-50 character set

Character ASCII	Octal Equivalent	Radix-50 Equivalent
space	40	0
A-Z	101-132	1-32
$	44	33
.	56	34
unused		35
0-9	60-71	36-47

The following table provides a convenient means of translating between the ASCII character set and its Radix-50 equivalents. For example, given the ASCII string X2B, the Radix-50 equivalent is (arithmetic is performed in octal):

```
X  =113000
 2 =002400
  B=000002
X2B=115402
```

Table A-1
ASCII/Radix-50 Equivalents

Single Char. or First Char.	Second Character	Third Character
space 000000	space 000000	space 000000
A 003100	A 000050	A 000001
B 006200	B 000120	B 000002
C 011300	C 000170	C 000003
D 014400	D 000240	D 000004
E 017500	E 000310	E 000005
F 022600	F 000360	F 000006
G 025700	G 000430	G 000007
H 031000	H 000500	H 000010
I 034100	I 000550	I 000011
J 037200	J 000620	J 000012
K 042300	K 000670	K 000013
L 045400	L 000740	L 000014
M 050500	M 001010	M 000015
N 053600	N 001060	N 000016
O 056700	O 001130	O 000017
P 062000	P 001200	P 000020
Q 065100	Q 001250	Q 000021
R 070200	R 001320	R 000022
S 073300	S 001370	S 000023
T 076400	T 001440	T 000024
U 101500	U 001510	U 000025
V 104600	V 001560	V 000026
W 107700	W 001630	W 000027
x 113000	x 001700	x 000030
y 116100	y 001750	y 000031
z 121200	z 002020	z 000032
$ 124300	$ 002070	$ 000033
. 127400	. 002140	. 000034
unused 132500	unused 002210	unused 000035
0 135600	0 002260	0 000036
1 140700	1 002330	1 000037
2 144000	2 002400	2 000040
3 147100	3 002450	3 000041
4 152200	4 002520	4 000042
5 155300	5 002570	5 000043
6 160400	6 002640	6 000044
7 163500	7 002710	7 000045
8 166600	8 002760	8 000046
9 171700	9 003030	9 000047

APPENDIX B

LIBRARY SUBROUTINES

B.1 LIBRARY SUBROUTINE SUMMARY

In addition to the functions intrinsic to the FORTRAN IV system, there are subroutines in the FORTRAN library which the user can call in the same manner as a user-written subroutine. These subroutines are:

ASSIGN	allows specification at run-time of the filename or device and filename to be associated with a FORTRAN IV logical unit number.
OPEN	causes the specified file to be opened and associated with a particular FORTRAN IV logical unit (RSTS/E only).
CLOSE	closes the specified logical unit after writing any active buffers to the file.
DATE	returns a 9-byte string containing the ASCII representation of the current date.
IDATE	returns three integer values representing the current month, day, and year.
EXIT	terminates the execution of a program and returns control to the executive.
USEREX	allows specification of a routine to be invoked as part of program termination.
RANDU, RAN	returns a pseudo random-real number with a uniform distribution between 0 and 1.
SETERR	allows the user to set a count specifying the number of times to ignore a certain error condition.

B.2 ASSIGN

The ASSIGN subroutine allows the association of device and/or filename information with a logical unit number. The ASSIGN call, if present, must be executed before the logical unit is opened for I/O operations (by READ or WRITE) for sequential-access files, or before the associated DEFINE FILE statement for random-access files. The assignment remains in effect until the end of the program or until the file is closed by CALL CLOSE, and a new CALL ASSIGN performed. The call to ASSIGN has the general form:

CALL ASSIGN(n, name, icnt, mode, control, numbuf)

CALL ASSIGN requires only the first argument, all others are optional, and if omitted, are replaced by the default values as noted in the argument descriptions. However, if any argument is to be included, all arguments that precede it must also be included.

NOTE

Under RSTS/E, any project-programmer number or protection code information supplied to CALL ASSIGN is ignored. If the ability to supply such information is desired, CALL OPEN should be used.

A description of the arguments to the ASSIGN routine follows:

n — logical unit number expressed as an integer constant or variable.

name — Hollerith or literal string containing any standard RT-11 or RSTS/E filename specification. If the device is not specified, then the device remains unchanged from the default assignments. If a filename is not specified, the default names, as described in Section 3.6, are used. There are three switches which can be included in the file specification. The switches are:

- /N — specifies no carriage control translation. This switch, if present, will override the value of the 'control' argument.
- /C — specifies carriage control translation. This switch, if present, will override the value of the 'control' argument.
- /B:n — specifies the number of buffers, n, to use for I/O operations. The single argument, n, should be of value 1 or 2. This switch, if present, will override the value of the 'numbuf' argument. Multi-buffering is not supported under RSTS/E.

If 'name' is simply a device specification, the device is opened in a non-file structured manner, and the device is treated in a non-file structured manner. Indiscriminate use of this feature on directory devices such as disk or DECtape can be dangerous (i.e., damage the directory structure).

icnt — specifies the number of characters in the string 'name'. If 'icnt' is zero, the string 'name' is processed until the first blank or null character is encountered. If 'icnt' is negative, program execution is temporarily suspended. A prompt character (*) is sent to the terminal, and a filename specification, with the same form as 'name' above, terminated by a carriage return, is accepted from the keyboard.

mode — specifies the method of opening the file on this unit. This argument can be one of the following:

'RDO' — the file is read only. A fatal error occurs if a FORTRAN WRITE is attempted on this unit. If the specified file does not exist, runtime error 28 (OPEN FAILED FOR FILE) is reported.

'NEW' — a new file of the specified name is created; this file does not become permanent until the associated logical unit is closed via the CALL CLOSE routine or program termination. If execution is aborted by typing CTRL C, the file is not preserved.

'OLD' — the file already exists. If the specified file does not exist, runtime error 28 (OPEN FAILED FOR FILE) is reported.

'SCR' — the file is only to be used temporarily and is deleted when it is closed.

If this argument is omitted, the default is determined by the first I/O operation performed on that unit. If a WRITE operation is the first I/O operation performed on that unit, 'NEW' is assumed. If a READ operation is first, 'OLD' is assumed.

control — specifies whether carriage control translation is to occur. This argument can be one of the following:

'NC' — all characters are output exactly as specified. The record is preceeded by a line feed character and followed by a carriage return character.

'CC' — the character in column one of all output records is treated as a carriage control character. (See the PDP-11 FORTRAN Language Reference Manual.)

If not specifically changed by the CALL ASSIGN subroutines, the terminal and line printer assume by default 'CC', and all other devices assume 'NC'.

numbuf — specifies the number of internal buffers to be used for the I/O operation. A value of 1 is appropriate under normal circumstances. If this argument is omitted, one internal buffer is used. Multi-buffering is not supported under RSTS/E.

B.3 OPEN (RSTS/E only)

The subroutine OPEN is an extension of the ASSIGN routine for RSTS/E. OPEN allows a specified file to be opened and associated with a particular FORTRAN logical unit. In the OPEN call, all arguments (except 'n' and 'name') are optional and will default if not specified.

A description of the arguments to the OPEN subroutine follows:

CALL OPEN (n, name, icnt, disp, control, numbuf, iotype, p, pn, prot, mode, cluster)

where

n — is the integer specification of the logical unit to be associated with the file.

name — is a variable, array, or quoted string, whose contents specify the name (and possibly the project-programmer number and protection code) of the file to be opened.

icnt — is an integer value which controls the interpretation of the 'name' argument. If icnt is positive, it specifies the number of characters to be taken from the 'name' argument as the filename string. If 'icnt' is zero, 'name' is scanned until the first blank or null character is encountered. If a negative value is given for 'icnt', program execution is temporarily suspended, a prompt character (*) is sent to the terminal, and a filename specification, terminated by a carriage return, is accepted from the keyboard.

disp — is a string specification of the disposition of the file on this unit. This argument can be one of the following:

'RDO' the file is read only. A fatal error occurs if a FORTRAN WRITE is attempted on this unit. The file is assumed to have the 'OLD' attribute.

'NEW' the file is to be created. If a file of the same name already exists in the directory, it will be superseded by the new file when it is closed.

'OLD' the file is assumed to exist. If the file is not found in the specified directory, or is protected against access by the user, a fatal error results.

'SCR' the file is only to be used temporarily and will be deleted when it is closed.

If this argument is not given, the default is set to 'NEW'.

control — is a string argument which specifies whether carriage control translation is to occur. This argument can be one of the following:

'NC' all characters are output exactly as specified.
'CC' the character in column one of all output records (formatted) is treated as a carriage control character (see the PDP-11 FORTRAN Language Reference Manual.)

If not specifically changed by the OPEN routine, the user's terminal and the line printer assume by default 'CC', and all other devices assume 'NC'.

numbuf — retained for argument list compatibility with the RT-11 ASSIGN system subroutine; has no function under RSTS/E and should be omitted.

iotype — is a string argument which specifies the type of input/output operations to be performed on a unit. This argument can be one of the following:

'FOR' the unit is to be opened for formatted input/output.
'UNF' the unit is to be opened for unformatted input/output.
'RAN' the unit is to be opened for random-access input/output.

If this argument is not specified, it defaults to 'FOR'.

p — is an integer value giving the default project code to be used (in conjunction with the "pn" argument) if no project-programmer number specification is found in 'name'.

pn — is an integer value giving the default programmer code to be assumed if no project-programmer specification appears in 'name'.

prot — is an integer value specifying the protection code to be assigned by default if no protection code indication occurs in 'name'. This argument only takes effect on output files.

mode — is an integer specification of the RSTS/E mode to be used on opening the file (refer to the RSTS/E Programming Manual for device-specific mode information).

cluster — is an integer specification of the cluster size to be assigned to the file to be opened. This argument only takes effect on output files (i.e., files with the 'NEW' or 'SCR' attribute).

B.4 CLOSE

The CLOSE routine is used to explicitly close any file open on the specified logical unit. If the file was open for output, any partially filled buffers are written to the file before closing it. After the execution of CALL CLOSE, any buffers associated with the logical unit are freed for reuse and all information supplied in any previous CALL ASSIGN for the logical unit is deleted. The logical unit is thus free to be associated with another file.

An implicit CLOSE operation is performed on all open logical units when a program terminates (due to a fatal error condition, or the execution of STOP or CALL EXIT).

The format of the call is:

```
CALL CLOSE (ilun)
```

where ilun is an integer constant, variable, array element, or expression specifying the logical unit to be closed.

B.5 DATE

The DATE subroutine can be used in a FORTRAN program to obtain the current date as set within the system. The DATE subroutine is called as follows:

```
CALL DATE(array)
```

where array is a predefined array able to contain a 9-byte string. The array specification in the call can be expressed as the array name alone:

```
CALL DATE(a)
```

in which the first three elements of the real array a are used to hold the date string, or:

```
CALL DATE(a(i))
```

which causes the 9-byte string to begin at the i(th) element of the array a.

The date is returned as a 9-byte (9-character) string in the form:

```
dd-mmm-yy
```

where:

```
dd is the 2-digit date
mmm is the 3-letter month specification
yy is the last two digits of the year
```

For example:

```
25-DEC-76
```

B.6 IDATE

IDATE returns three integer values representing the current month, day, and year. The call has the form:

```
CALL IDATE(i,j,k)
```

If the current date were March 19, 1976 the values of the integer variables upon return would be:

```
i = 3
j = 19
k = 76
```

B.7 EXIT

A call to the EXIT subroutine, in the form:

```
CALL EXIT
```

is equivalent to the STOP statement. It causes program termination, closes all files, and returns to the monitor.

B.8 USEREX

USEREX is a subroutine which allows specification of a routine to which control is passed as part of program termination. This allows disabling of interrupts enabled in non-FORTRAN routines. If these interrupts are not disabled prior to program exit the integrity of the operating system cannot be assured. The form of the subroutine call is:

```
CALL USEREX (name)
```

where 'name' is the routine to which control is passed and should appear in an EXTERNAL statement somewhere in the program unit. Control is transferred with a JMP instruction after all procedures required for FORTRAN IV program termination have been completed. The transfer of control takes place instead of the normal return to the monitor. Thus, if the user desires to have control passed back to the monitor, the routine specified by USEREX must perform the proper exit procedures.

B.9 RANDU,RAN

The random number generator can be called as a subroutine, RANDU, or as an intrinsic function, RAN. The subroutine call is performed as follows:

```
CALL RANDU (i(1) ,i(2) ,x)
```

where i(1) and i(2) are previously defined integer variables and x is the real variable name, in which a random number between 0 and 1 is returned. i(1) and i(2) should be initially set to 0. i(1) and i(2) are updated to a new generator base during each call. Resetting i(1) and i(2) to 0 repeats the random number sequence. The values of i(1) and i(2) have a special form; only 0 or saved values of i(1) and i(2) should be stored in these variables.

Use of the random number subroutines is similar to the use of the RAN function where:

```
x=RAN(i(1) ,i(2))
```

is the functional form of the random number generator.

B.10 SETERR

SETERR allows the user to specify the disposition of certain OTS detected error conditions. Only OTS error diagnostics 1 - 16 should be changed from their default error classification (see Section C.2). If errors 0 or 20 - 63 are changed from the default classification of FATAL, execution continues but in an undetermined state. The form of the call is:

```
CALL SETERR (number,ncount)
```

where 'number' is an integer variable or expression specifying the OTS error number (see Section C.2), and 'ncount' is an integer variable or expression with one of the following values:

Value	Meaning
0	Ignore all occurrences after logging them on the user terminal.
1	First occurrence of the error will be fatal.
2-127	The nth occurrence of the error will be fatal; the first n-1 occurrences will be logged on the user terminal.
255	Ignore all occurrences of the error.

APPENDIX C

FORTRAN IV ERROR DIAGNOSTICS

C.1 COMPILER ERROR DIAGNOSTICS

The FORTRAN IV Compiler, while reading and processing the FORTRAN source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, missing or illegal statement parameters.

The error diagnostics are generally clear in specifying the exact nature of the error. In most cases, a check of the general form of the statement in question as described in the PDP-11 FORTRAN Language Reference Manual will help determine the location of the error.

Some of the most common causes of syntax errors, however, are typing mistakes. A typing mistake can sometimes cause the compiler to give very misleading error diagnostics. The user should note, and take care to avoid, the following common typing mistakes:

1. Missing commas or parentheses in a complicated expression or FORMAT statement.

2. Misspelling of particular instances of variable names. If the compiler does not detect this error (it usually cannot), execution may also be affected.

3. An inadvertent line continuation signal on the line following the statement in error.

4. If the user terminal does not clearly differentiate between 0 (zero) and the letter O, what appear to be identical spellings of variable names may not appear so to the compiler, and what appears to be a constant expression may not appear so to the compiler.

If any error or warning conditions are detected in a compilation, the following message is printed on the initiating terminal:

[name] ERRORS: n, WARNINGS: m

where:

[name] is the 6-character name of the program unit being compiled; .MAIN. indicates the main program or the program unit name as specified in the SUBROUTINE FUNCTION or PROGRAM statement.

n represents the number of error-class messages (i.e.,

those messages that cause the statement in question to be deleted).

m — represents the number of warning-class messages (i.e., those messages indicating conditions that can be ignored or corrected, such as missing END statements or questionable programming practices). Note that some warning conditions will only be detected if the /W switch is specified, (see Section C.1.3.).

The next three sections describe the initial phase and secondary phase error diagnostics and the fatal FORTRAN IV Compiler error diagnostics.

The following is an example of a FORTRAN IV program with diagnostics issued by the compiler.

```
FORTRAN IV      V01C-01     THU 13-NOV-75 16:24:16                PAGE 001

0001          DOUBLE PRECISION DBLE, DBLE2
0002          DATA INT/100/, DBLE/500/
0003          DBLE2 = INT/2 + 5. + DBLE
0004          WRITE,(6,10) DBLE,DBLE2
0005   10     FORMAT(1X,2F12.6)
0006   10     STOP
***** M
0007          INTEGER INT
0008          END


FORTRAN IV      DIAGNOSTICS

IN LINE 0002 MSG #036   MODES OF VARIABLE 'DBLE' AND DATA ITEM DIFFER
IN LINE 0004 MSG #050   SYNTAX ERROR
[ WARNING ]  MSG #095   NON-STANDARD STATEMENT ORDERING


FORTRAN IV      STORAGE MAP

NAME     OFFSET  ATTRIBUTES

DBLE     000010  REAL*8     VARIABLE
DBLE2    000026  REAL*8     VARIABLE
INT      000006  INTEGER*2  VARIABLE
```

C.1.1 Errors Reported By The Initial Phase Of The Compiler

Some of the easily recognizable FORTRAN syntax errors are detected by the initial phase of the compiler. Errors that cause the statement in question to be aborted are tabulated in the ERROR count, whereas those that are correctable by the compiler are counted as WARNINGS.

The error diagnostics are printed after the source statement to which they apply (the L error diagnostic is an exception). The general form of the diagnostic is as follows:

```
***** c
```

Where c is a code letter whose meaning is described below:

INITIAL PHASE ERROR DIAGNOSTICS

Code Letter	Description
B	Columns 1-5 of a continuation line are not blank. Columns 1-5 of a continuation line must be blank except for a possible 'D' in column 1; the columns are ignored (WARNING).
C	Illegal continuation. Comments cannot be continued and the first line of any program unit cannot be a continuation line. If a line consists only of a carriage return-line feed combination, then it is considered to be a blank line. If it has a label field then it must have a statement field. The line is ignored (WARNING).
E	Missing END statement. An END statement is supplied by the compiler if end-of-file is encountered (WARNING).
H	Hollerith string or quoted literal string is longer than 255 characters or longer than the remainder of the statement; the statement is aborted.
I	Non-FORTRAN character used. The line contains a character that is not in the FORTRAN character set and is not used in a Hollerith string or comment line. The character is ignored (WARNING).
K	Illegal statement label definition. Illegal (non-numeric) character in statement label; the illegal statement label is ignored (WARNING).
L	Line too long to print. There are more than 80 characters (including spaces and tabs) in a line. Note: this diagnostic is issued preceding the line containing too many characters. The line is truncated to 80 characters (WARNING).
M	Multiply defined label. The label is ignored (WARNING).

Code Letter	Description
P	Statement contains unbalanced parentheses. The statement is aborted.
S	Syntax error. Multiple equal signs, etc. The statement is not of the general FORTRAN statement form; the statement is aborted.
U	Statement could not be identified as a legal FORTRAN statement. The statement is aborted.

C.1.2 Errors Reported By Secondary Phases Of The Compiler

Those compiler error diagnostics not reported by the initial phase of the compiler appear immediately after the source listing and immediately before the storage map. Since the diagnostics appear after the entire source program has been listed, they must designate the statement to which they apply by using the internal sequence numbers assigned by the compiler.

The general form of the diagnostic is:

```
IN LINE nnnn MSG#m text
```

Where nnnn is the internal sequence number of the statement in question, m specifies an internal error number, and text is a short description of the error.

Below, listed alphabetically, are the error diagnostics. Included with each diagnostic is a brief explanation. Refer to the PDP-11 FORTRAN Language Reference Manual for information to help correct the error.

The notation **** signifies that a particular variable name or statement label appears at that place in the text.

SECONDARY PHASE ERROR DIAGNOSTICS

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****
: An adjustable array was not a dummy argument in a subprogram or the adjustable dimensions were not integer dummy arguments in the subprogram. A dimension of one is used. Correct the source program.

ARRAY EXCEEDS MAXIMUM SIZE or
ARRAY **** EXCEEDS MAXIMUM SIZE
: The storage required for a single array or for all arrays in total is more than is physically addressable (>32K words). This particular error may reference either the actual statement containing the array in question, or the first statement in the program unit. Correct the statement in error or reduce the space necessary for array storage.

ARRAY **** HAS TOO MANY DIMENSIONS
: An array has more than seven dimensions. Correct the program. The legal range for dimensions is one to seven.

ATTEMPT TO EXTEND COMMON BACKWARDS
: While attempting to EQUIVALENCE arrays in COMMON, an attempt was made to extend COMMON past the recognized beginning of COMMON storage. Correct the program logic.

COMMON BLOCK EXCEEDS MAXIMUM SIZE
: An attempt was made to allocate more space to COMMON than is physically addressable (>32k words). Correct the statement in error.

CONSTANT IN FORMAT STATEMENT NOT IN RANGE
: An integer constant in a FORMAT statement was not in the proper range. Check that all integer constants are within the legal range (1 to 255).

DANGLING OPERATOR
: An operator (+,-,*,/, etc.) is missing an operand. Example: I=J+. Correct the statement in error.

DEFECTIVE DOTTED KEYWORD
: A dotted relational operator was not recognized or there is a possible misuse of decimal point. Check the format for relational operators; correct the statement in error.

DO TERMINATOR **** PRECEDES DO STATEMENT
: The statement specified as the terminator of a DO loop did not appear after the DO statement. Correct the program logic.

EXPECTING LEFT PARENTHESIS AFTER ****
: An array name or function name reference is not followed by a left parenthesis. Correct the statement so that the left parenthesis is included.

EXTRA CHARACTERS AT END OF STATEMENT
: All the necessary information for a syntactically correct FORTRAN statement has been found on this line, but more information exists. Check that a comma is not missing from the line or that an unintentional continuation signal does not appear on the next line.

FLOATING CONSTANT NOT IN RANGE
: A floating constant in an expression is too close to zero to be represented in the internal format. Use zero if possible.

ILLEGAL ADJACENT OPERATOR
: Two operators (*,/, logical operators, etc.) are illegally placed next to each other. Example: I/*J. Correct the statement in error.

ILLEGAL DO TERMINATOR ORDERING AT LABEL ****
: DO loops are nested improperly. Verify that the range of each DO loop lies completely within the range of the next outer loop.

ILLEGAL DO TERMINATOR STATEMENT ****
A DO statement terminator was not valid. Verify that the DO statement terminator is not a GOTO, arithmetic IF, RETURN, another DO statement, or logical IF containing one of these statements.

ILLEGAL ELEMENT IN I/O LIST
An item, expression, or implied DO specifier in an I/O list is of illegal syntax. Correct the I/O list.

ILLEGAL STATEMENT IN BLOCK DATA
An illegal statement was found in a BLOCK DATA subprogram. Verify that a FORMAT or executable statement does not occur in a BLOCK DATA subprogram.

ILLEGAL STATEMENT ON LOGICAL IF
The statement contained in a logical IF was not valid. Verify that the statement is not another logical IF or DO statement.

ILLEGAL TYPE FOR OPERATOR
An illegal variable type has been used with an exponentiation or logical operator. Check that the variable type is valid for the operation in question.

ILLEGAL USAGE OF OR MISSING LEFT PARENTHESIS
A left parenthesis was required but not found, or a variable reference or constant is illegally followed by a left parenthesis. Correct the format of the statement in error.

INTEGER OVERFLOW
An integer constant or expression value is outside the range -32767 to +32767. Correct the value of the integer constant or expression so that it falls within the legal range (-32767 to +32767).

INVALID COMPLEX CONSTANT
A complex constant has been improperly formed. Correct the statement in error.

INVALID DIMENSIONS FOR ARRAY ****
An attempt was made, while dimensioning an array, to explicitly specify zero as one of the dimensions. Verify that zero is not used as a dimension.

INVALID EQUIVALENCE
An illegal EQUIVALENCE, or EQUIVALENCE that is contradictory to a previous EQUIVALENCE, was encountered. Correct the program logic.

INVALID FORMAT SPECIFIER
A format specifier was illegally used. Correct the statement so that the format specifier is the label of a FORMAT statement or an array name.

INVALID IMPLICIT RANGE SPECIFIER
An illegal implicit range specifier, (i.e., non-alphabetic specifier, or specifier range in reverse alphabetic order), was encountered. Verify that the implicit range specifier indicates alphabetic characters in alphabetic order.

INVALID LOGICAL UNIT
A logical unit reference was incorrect. Correct the logical unit reference so that it is an integer variable or constant in the range 1 to 99.

INVALID OCTAL CONSTANT
An octal constant is too large or contains a digit other than 0-7. Correct the constant so that it contains only legal digits that fall within the octal range 0 to 177777.

INVALID OPTIONAL LENGTH SPECIFIER
A data type declaration optional length specifier is illegal. For example, REAL*4 and REAL*8 are legal, but REAL*6 is not. Correct the statement so that it contains only a valid data type declaration length.

INVALID RADIX-50 CONSTANT
An illegal character was detected in a RADIX-50 constant. Verify that only characters from the RADIX-50 character set are used in a RADIX-50 constant.

INVALID RECORD FORMAT
The third parenthetical argument in a DEFINE FILE statement was not the single character U. Correct the DEFINE FILE statement.

INVALID STATEMENT LABEL REFERENCE
Reference has been made to a statement number that is of illegal construction. For example, GOTO 999999 is illegal since the statement number is too long. Check that the statement number consists of one to five decimal digits placed in the first five columns of a statement's initial line and that it does not contain only zeroes.

INVALID SUBROUTINE OR FUNCTION NAME
A name used in a CALL statement or function reference is not valid. For example, use of an array name in a CALL statement routine name reference is illegal. Verify that the name specified in the statement is spelled correctly.

INVALID TARGET FOR ASSIGNMENT
The left side of an arithmetic assignment statement is not a variable name or array element reference. Correct the statement in error.

INVALID TYPE SPECIFIER
An unrecognizable data type was used. Verify that the data type indicated is valid.

INVALID USAGE OF SUBROUTINE OR FUNCTION NAME
A function name appeared in a DIMENSION, COMMON, DATA, or EQUIVALENCE or data type declaration statement. Correct the statement in error.

INVALID VARIABLE NAME
A variable name contains an illegal character, is missing, or does not begin with an alphabetic character. Correct the statement in error.

LABEL ON DECLARATIVE STATEMENT
A label was found on a declarative statement. Correct the program so that declarative statements do not have labels.

MISSING ASSIGNMENT OPERATOR
The first operator seen in an arithmetic assignment statement was not an equal sign (=). For example, I+J=K. Correct the arithmetic assignment statement in error.

MISSING COMMA
The comma delimiter was expected but not found. Correct the format of the statement in error.

MISSING DELIMITER IN EXPRESSION
Two operands have been placed next to each other in an expression with no operator between them. Correct the statement in error.

MISSING LABEL
A statement label was expected but not found. For example, ASSIGN J TO I is illegal; a valid statement label reference should precede 'TO' but does not. Verify that the reference preceding 'TO' is a valid statement label of an executable statement in the same program unit as the ASSIGN statement.

MISSING RIGHT PARENTHESIS
A right parenthesis was expected but not found. For example, READ(5,100,) is illegal; the first non-blank character after the format reference should be a right parenthesis but is not. Correct the format of the statement in error.

MISSING QUOTATION MARK
In a FIND statement, the logical unit number and record number were not separated by a single quotation mark. Correct the statement in error.

MISSING VARIABLE
A variable was expected, but not found. For example, ASSIGN 100 TO 1 is illegal; a variable name should follow the 'TO' but does not. Verify that the reference following 'TO' is a valid integer variable name.

MISSING VARIABLE OR CONSTANT
An operand (variable or constant) was expected but a delimiter (comma, parenthesis, etc.) was found. For example, WRITE() is illegal; a unit number should follow the open parenthesis, but a delimiter (close parenthesis) is encountered instead. Correct the format of the statement in error.

MODES OF VARIABLE **** AND DATA ITEM DIFFER
The data type of each variable and its associated data list item must agree in a DATA statement. Correct the format of the items in the DATA statement.

MULTIPLE DECLARATION FOR VARIABLE ****
A variable appeared in more than one data type declaration statement or dimensioning statement. Subsequent declarations are ignored. Correct the program logic.

NUMBER IN FORMAT STATEMENT NOT IN RANGE
An integer constant in a FORMAT statement is greater than 255 or is zero. Correct the statement so that the integer constant falls within the legal range (one to 255).

PARENTHESES NESTED TOO DEEPLY
Group repeats in a FORMAT statement have been nested too deeply. Limit group repeats to eight levels of nesting.

P-SCALE FACTOR NOT IN RANGE -127 TO +127
P-scale factors were not in the range -127 to +127. Correct the statement in error.

REFERENCE TO INCORRECT TYPE OF LABEL ****
A statement label reference that should be a label on a FORMAT statement is not such a label, or a statement label reference that should be a label on an executable statement is not such a label. Correct the program logic.

REFERENCE TO UNDEFINED STATEMENT LABEL
A reference has been made to a statement label that has not been defined anywhere in the program unit. Correct the program logic.

STATEMENT MUST BE UNLABELED
A DATA, SUBROUTINE, FUNCTION, BLOCK DATA, arithmetic statement function definition, or declarative statement was labeled. Correct the statement in error.

STATEMENT TOO COMPLEX
An arithmetic statement function has more than ten dummy arguments or the statement is too long to compile. Verify that the number of dummy arguments in an arithmetic statement does not exceed ten; break long statements into two or more smaller statements.

SUBROUTINE OR FUNCTION STATEMENT MUST BE FIRST
A SUBROUTINE, FUNCTION or BLOCK DATA statement is not the first statement in a program unit. Ensure that, if present, these statements appear first in a program unit.

SUBSCRIPT OF ARRAY **** NOT IN RANGE
Array subscripts that are constants or constant expressions are found to be outside the bounds of the array's dimensions. The operation in question is aborted. Correct the program.

SYNTAX ERROR

The general form of the statement was not formatted correctly. Check the general format of the statement in error and correct the program.

SYNTAX ERROR IN INTEGER OR FLOATING CONSTANT

An integer or floating constant has been incorrectly formed. For example, 1.23.4 is an illegal floating constant because it contains two decimal points. Correct the format of the integer or floating constant in question.

UNLABELED FORMAT STATEMENT

A FORMAT statement was not labeled. Correct the FORMAT statement in error by assigning it the proper label.

USAGE OF VARIABLE **** INVALID

An attempt was made to EXTERNAL a common variable, an array variable, or a dummy argument. Or an attempt was made to place in COMMON a dummy argument or external name. Correct the program logic.

VALUE OF CONSTANT NOT IN RANGE

An integer constant in the designated source program line exceeds the maximum unsigned value (65535). This error is also printed if an invalid dimension is specified for an array or if the exponent of a floating point constant is too large. Correct the statement in error.

VARIABLE **** INVALID IN ADJUSTABLE DIMENSION

A variable used as an adjustable dimension was not an integer dummy argument in the subprogram unit. Correct the program.

WRONG NUMBER OF SUBSCRIPTS FOR ARRAY ****

An array reference does not have the same number of subscripts as specified when the array was dimensioned. Correct the statement in error.

C.1.3 Warning Diagnostics

Warning diagnostics report conditions which are not true error conditions, but which can be potentially dangerous at execution time, or can present compatibility problems with FORTRAN compilers running on other DEC operating systems. The warning diagnostics are normally suppressed, but can be enabled by use of the /W compiler switch. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics (see Section C.1.2) except that the line number reference is replaced with '[WARNING]'. A listing of the warning diagnostics follows:

ADJUSTABLE DIMENSIONS ILLEGAL FOR ARRAY ****

Adjustable arrays must be parameter arrays in a subprogram, and the adjustable dimensions must be integer dummy arguments in the subprogram. Any variation from this rule will cause a dimension of 1 to be used and this warning message to be issued.

NON-STANDARD STATEMENT ORDERING
: Although the FORTRAN IV Compiler has less-restrictive statement ordering requirements than those outlined in Chapter 7 of the PDP-11 FORTRAN Language Reference Manual, non-adherence to the stricter requirements may cause error conditions on other FORTRAN compilers. See Section 3.7 of this document.

VARIABLE **** IS NOT WORD ALIGNED
: Placing a non-LOGICAL*1 variable or array after a LOGICAL*1 variable or array in COMMON or equivalencing non-LOGICAL*1 variables or arrays to LOGICAL*1 variables or arrays can cause this condition. An attempt to reference the variable at runtime will cause an error condition.

VARIABLE **** NAME EXCEEDS SIX CHARACTERS
: A variable name of more than six characters was specified. The first six characters were used as the true variable name. Other FORTRAN compilers may treat this as an error condition. See Section 3.2 of this document.

C.1.4 Fatal Compiler Error Diagnostics

Listed below are the fatal compiler error diagnostics. These diagnostics, which are sent directly to the initiating terminal, report hardware error conditions, conditions which may require rewriting of the source program, and compiler errors which may require attention from your Software Support Representative. The form of the diagnostic is:

```
FATAL ERROR n
```

where n is an error code having one of the following values:

Code	Meaning
C	Constant subscript overflow. Too many constant subscripts have been employed in a statement. Simplify the statement
O	Unrecoverable error occurred while the compiler was writing the object file (.OBJ). Possibly, insufficient output file space. Rectify hardware problem, or make more space available for output by deleting unnecessary files.
P	Optimizer push down overflow - statement too complex, or too many common subexpressions occurred in one basic block of a program. Simplify complex statements.
R	Unrecoverable hardware error occurred while the compiler was reading source file. Rectify hardware problem.
S	Subexpression stack overflow - statement too complex. An attempt was made to compile a statement which would overflow the runtime stack at execution time. Simplify complex statements.
T	Core Overflow. Break up program into subprograms or compile on a larger machine.
W	Unrecoverable error occurred while the compiler was writing listing file. Possibly, listing file space is not large enough. Rectify hardware problem, or make more space available for listing file by deleting unnecessary files.
Y	Code generation stack overflow - statement too complex. Simplify complex statements.
Z	Compiler error - Report this error to the local Software Performance Report (SPR) Center listed in the DIGITAL SOFTWARE NEWS. Use an SPR form to report the error and include a program listing (made with the /L:ALL option, see Section 1.2.1) and a machine readable source program, if possible.

C.2 OBJECT TIME SYSTEM ERROR DIAGNOSTICS

The Object Time System detects certain I/O, arithmetic, and system failure error conditions and reports them on the user terminal. These error diagnostics are printed in either a long or short form.

The short form of the message appears as:

```
?ERR nn
```

where nn is a decimal error identification number.

The long form of the message appears as:

```
?ERR nn text
```

where nn is a decimal error identification number and text is a short error description.

The default message form is long. The short message error module can be linked to the program by using the /I linker switch. The module named $SHORT should be included from the FORTRAN library.

There are four classes of OTS error conditions. Each error condition is assigned to one of these classes. An error condition classification for the error codes 1-16 can be changed by using the system subroutine SETERR. (See Section B.10). Error codes 0 and 20-63 should not be changed from their FATAL classification or indeterminable results will occur. The classifications are:

IGNORE	the error is detected but no error message is sent to the terminal. Execution continues.
WARNING	the error message is sent to the terminal and execution continues.
FATAL	the error message is sent to the terminal and execution is terminated.
COUNT:n	the error message is sent to the terminal and execution continues until the nth occurrence of the error, at which time the error will be treated as FATAL.

If a program is terminated by a fatal error condition, active files are not closed. Under RT-11, when control is returned to the Monitor, a CLOSE command can be given to close all active files, although some of the output to these active files may have been lost.

The OTS error diagnostics are listed below along with the error type and a brief explanation, where necessary.

Error number	Error type	Message
0	FATAL	NON-FORTRAN ERROR CALL This message indicates an error condition (not internal to the FORTRAN run-time system) that may have been caused by one of four situations: (RT-11 only) 1. A foreground job using SYSLIB completion routines was not allocated enough space (using the FRUN /N switch) for the initial call to a completion routine. Check Appendix G (Section G.1) and Appendix O (Section O.1.4) of the RT-11 System Reference Manual for the formula used to allocate more space.

Error number	Error type	Message
		(RT-11 only) 2. There was not sufficient memory for the background job. Make more memory available by unloading unnecessary handlers, deleting unwanted files, compressing the device. (RT-11 only) 3. Under the single-job monitor, a SYSLIB completion routine interrupted another completion routine. Use the F/B Monitor to allow more than one active completion routine. 4. An assembly language module linked with a FORTRAN program issued a TRAP instruction with an error code that was not recognized by the FORTRAN error handler. Check the program logic.
1	FATAL	INTEGER OVERFLOW During an integer multiplication, division, or exponentiation operation, the value of the result exceeded 32767 in magnitude. Correct the program logic.
2	FATAL	INTEGER ZERO DIVIDE During an integer mode arithmetic operation, an attempt was made to divide by zero. Correct the program logic.
3	FATAL	COMPILER DETECTED ERROR An attempt was made to execute a FORTRAN statement in which the compiler had previously detected errors. Consult the program listing generated by the compiler (if one was requested) and correct the program for the errors detected at compile time.
4	WARNING	COMPUTED GOTO OUT OF RANGE The value of the integer variable or expression in a computed GOTO statement was less than 1 or greater than the number of statement label references in the list. Control is passed to the next executable statement. Examine the source program and correct the program logic.
5	COUNT:3	INPUT CONVERSION ERROR During a formatted input operation, an illegal character was detected in an input field. A value of zero is returned. Examine the input data and correct the invalid record.

Error number	Error type	Message
6	IGNORE	OUTPUT CONVERSION ERROR During a formatted output operation, the value of a particular number could not be output in the specified field length without loss of significant digits. The field is filled with asterisks ('*'). Correct the FORMAT statement to allow a greater field length.
10	COUNT:3	FLOATING OVERFLOW During an arithmetic operation, the absolute value of a floating-point expression exceeded the largest representable real number. A value of zero is returned. Correct the program logic.
11	IGNORE	FLOATING UNDERFLOW During an arithmetic operation, the absolute value of a floating-point expression became less than the smallest representable real number. The real number is replaced with a value of zero. Correct the program logic.
12	FATAL	FLOATING ZERO DIVIDE During a REAL mode arithmetic operation an attempt was made to divide by zero. The result of the operation is set to zero. Correct the program logic.
13	COUNT:3	SQRT OF NEGATIVE NUMBER An attempt was made to take the square root of a negative number. The result is replaced by zero. Correct the program logic.
14	FATAL	UNDEFINED EXPONENTIATION OPERATION An attempt was made to perform an illegal exponentiation operation. (For example -3.**.5 is illegal because the result would be an imaginary number.) The result of the operation is set to zero. Correct the program logic.
15	FATAL	LOG OF NEGATIVE NUMBER An attempt was made to take the logarithm of a negative number or zero. The result of the operation is set to zero. Correct the program logic.

Error number	Error type	Message
16	FATAL	WRONG NUMBER OF ARGUMENTS One of the FORTRAN Library functions, or one of the system subroutines which checks for such an occurrence, was called with an improper number of arguments. Check the format of the particular library function or system subroutine call, and correct the call.

The following error diagnostics should not be changed from the FATAL classification by use of the system subroutine SETERR:

20	FATAL	INVALID LOGICAL UNIT NUMBER An illegal logical unit number was specified in an I/O statement. A logical unit number must be an integer within the range 1 to 99. Correct the statement in error.
21	FATAL	OUT OF AVAILABLE LOGICAL UNITS An attempt was made to have too many logical units simultaneously open for I/O. The maximum number of active logical units is six by default. To increase the maximum, recompile the main program using the /N switch to specify a larger number of available channels.
22	FATAL	INPUT RECORD TOO LONG During an input operation, a record was encountered that was longer than the maximum record length. The default maximum record length is 136 (decimal) bytes. To increase the maximum, recompile the main program using the /R switch to specify a larger run-time record buffer (the legal range is 4 to 4095).
23	FATAL	HARDWARE I/O ERROR A hardware error was detected during an I/O operation. Check the volume for an off-line or write-locked condition, and retry the operation. Try another unit or drive if possible, or use another device.
24	FATAL	ATTEMPT TO READ/WRITE PAST END OF FILE During a sequential read operation, an attempt was made to read beyond the last record of the file. During a random-access read, this message indicates that an attempt was made to reference a record number that was not within the bounds of the file.

Error number	Error type	Message
		Use the "END=" parameter to detect this condition, or correct the program logic so that no request is made for a record outside the bounds of the file. During a WRITE operation, this message indicates that the space available for the file is insufficient. (RT-11 action) Try to make more file space available by deleting unnecessary files and compressing the device, or by using another device. (RSTS/E action) This condition is equivalent to the "NO ROOM FOR USER ON DEVICE" system error. Make more space available by deleting files from the current account, or use another device.
25	FATAL	ATTEMPT TO READ AFTER WRITE An attempt was made to read after writing on a sequential file located on a file-structured device. A write operation must be followed by a REWIND or BACKSPACE before a read operation can be performed. Correct the program logic.
26	FATAL	RECURSIVE I/O NOT ALLOWED An expression in the I/O list of a WRITE statement caused initiation of another READ or WRITE operation. (This can happen if a FUNCTION that performs I/O is referenced within an expression in an I/O list.) Correct the program logic.
27	FATAL	ATTEMPT TO USE DEVICE NOT IN SYSTEM An attempt was made to access a device that was not legal for the system in use. Use the system ASSIGN command to create the required logical device name, or change the statement in error.
28	FATAL	OPEN FAILED FOR FILE The file specified was not found, or there was no room on the device. Verify that the file exists as specified. Delete unnecessary files from the device, or use another device.
29	FATAL	NO ROOM FOR DEVICE HANDLER (RT-11 only) There was not enough free memory left to accommodate a specific device handler. Move the file to the system device or to a device whose handler is resident. Make more memory available by unloading unnecessary handlers, unloading the foreground job, using the single-job Monitor, or SET USR SWAP, if possible.

Error number	Error type	Message
30	FATAL	NO ROOM FOR BUFFERS There was not enough free memory left to set up required I/O buffers. (RT-11 only) Reduce the number of logical units that are open simultaneously at the time of the error. If using double buffering or if another file is currently open, use single buffering. Make more memory available by unloading unnecessary handlers, unloading the foreground job, using the single-job Monitor, or SET USR SWAP, if possible. (RSTS/E only) Increase the space available for buffering by specifying the appropriate value for the /CORE switch to EXEC, or reduce the number of logical units that are open simultaneously at the time of the error.
31	FATAL	NO AVAILABLE I/O CHANNEL More than the maximum number of channels available to the FORTRAN IV run-time system (15 for RT-11; 14 for RSTS/E, exclusive of the terminal) were requested to be simultaneously opened for I/O. Close any logical units previously opened that need not be open at this time.
32	FATAL	FMTD-UNFMTD-RANDOM I/O TO SAME FILE An attempt was made to perform any combination of formatted, unformatted, or random access I/O to the same file. Correct the program logic.
33	FATAL	ATTEMPT TO READ PAST END OF RECORD An attempt was made to read a larger record than actually existed in a file. Check the construction of the data file; correct the program logic.
34	FATAL	UNFMTD I/O TO TTY OR LPT An attempt was made to perform an unformatted write operation on the terminal or line printer. Assign the logical unit in question to the appropriate device using the ASSIGN system command, the ASSIGN or OPEN FORTRAN library routine, or (RT-11 only) the IASIGN SYSLIB routine.
35	FATAL	ATTEMPT TO OUTPUT TO READ ONLY FILE An attempt was made to write on a file designated as read-only. Check the CALL ASSIGN or OPEN system subroutine or (RT-11 only) IASIGN SYSLIB function to ensure that the correct arguments were used. Check for a possible programming error.

Error number	Error type	Message
36	FATAL	BAD FILE SPECIFICATION STRING The Hollerith or literal string specifying the device/filename in the CALL ASSIGN or CALL OPEN system subroutine could not be interpreted. Check the format of the CALL ASSIGN or CALL OPEN statement.
37	FATAL	RANDOM ACCESS READ/WRITE BEFORE DEFINE FILE A random-access read or write operation was attempted before a DEFINE FILE was performed. Correct the program so that the DEFINE FILE operation is executed before any random-access read or write operation.
38	FATAL	RANDOM I/O NOT ALLOWED ON TTY OR LPT Random-access I/O was illegally attempted on the terminal or line printer. Assign the logical unit in question to the appropriate device using the ASSIGN keyboard monitor command, the ASSIGN or OPEN FORTRAN library routine, or (RT-11 only) the IASIGN SYSLIB routine.
39	FATAL	RECORD LARGER THAN RECORD SIZE IN DEFINE FILE A record was encountered that was larger than that specified in the DEFINE FILE statement for a random-access file. Shorten the I/O list or redefine the file specifying larger records.
40	FATAL	REQUEST FOR A BLOCK LARGER THAN 65535 An attempt was made to reference an absolute disk block address greater than 65535. Correct the program logic.
41	FATAL	DEFINE FILE ATTEMPTED ON AN OPEN UNIT A file was open on a unit and another DEFINE FILE was attempted on that unit. Close the open file using CALL CLOSE before attempting another DEFINE FILE.
42	FATAL	MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT The OTS ran out of free memory while scanning an array format generated at run-time. (RT-11 only) Use a FORMAT statement specification at compile-time rather than object-time formatting, or make more memory available by unloading unnecessary handlers, unloading the foreground job, using the single-job Monitor, or SET USR SWAP, if possible.

Error number	Error type	Message
		(RSTS/E only) Use a FORMAT statement specification at compile-time rather than object-time formatting, or allocate more space by using the /CORE switch to EXEC.
43	FATAL	SYNTAX ERROR IN OBJECT TIME FORMAT A syntax error was encountered while the OTS was scanning an array format generated at run-time. Correct the programming error.
44	FATAL	2ND RECORD REQUEST IN ENCODE/DECODE An attempt was made to use ENCODE and DECODE on more than one record. Correct the FORMAT statement associated with the ENCODE or DECODE so that it specifies only one record.
45	FATAL	INCOMPATIBLE VARIABLE AND FORMAT TYPES An attempt was made to output a real variable with an integer field descriptor or an integer variable with a real field descriptor. Correct the FORMAT statement associated with the READ or WRITE, ENCODE or DECODE.
46	FATAL	INFINITE FORMAT LOOP The format associated with an I/O statement, which includes an I/O list, had no field descriptors to use in transferring those variables. Correct the FORMAT statement in error.
47	FATAL	ATTEMPT TO STORE OUTSIDE PARTITION (RT-11 only) In an attempt to store data into a subscripted variable, the address calculated for the array element in question did not lie within the section of memory allocated to the job. The subscript in question was out-of-bounds. (This message is issued only when bounds checking modules have been installed in FORLIB.) Correct the program logic.
48	FATAL	UNIT ALREADY OPEN An attempt was made to perform an illegal operation on an open file.
49	FATAL	ENDFILE ON RANDOM IFLE An ENDFILE statement contains a unit number of a file which is open as a random-access file.

Error number	Error type	Message
59	WARNING	USR NOT LOCKED (RT-11 only) This message is issued when the FORTRAN program is started. If the program was running in the foreground, the /U switch was used during compilation, and the USR was swapping (i.e., a SET USR NOSWAP command has not been done). Re-examine the intent of the /U switch at compile time and either compile without /U or issue a SET USR NOSWAP command.
60	FATAL	STACK OVERFLOWED The hardware stack overflowed. More stack space may be required for subprogram calls and opening of file. Proper traceback is impaired. This message occurs in the background only. Allocate additional space by using the /B switch at link-time. Check for a programming error.
61	FATAL	ILLEGAL MEMORY REFERENCE Some type of bus error occurred, most probably an illegal memory address reference. If an assembly language routine was called, check for a coding error in the routine. Otherwise, insure that the correct FORTRAN library was called.
62	FATAL	FORTRAN START FAIL The program was loaded into memory but there was not enough free memory remaining for the OTS to initialize work space and buffers. (RT-11 only) If running a background job, make more memory available by unloading unnecessary handlers, using the single-job Monitor. If running a foreground job, specify a larger value using the FRUN /N switch. Refer to the formulas in Appendix G (Section G.1) and Appendix O (Section O.1.4) of the RT-11 System Reference Manual. (RSTS/E only) Allocate more space by using the /CORE switch to EXEC.
63	FATAL	ILLEGAL INSTRUCTION The program attempted to execute an illegal instruction (e.g., floating-point arithmetic instruction on a machine with no floating-point hardware). If an assembly language routine was called, check for a coding error in the routine. Otherwise, ensure that the correct FORTRAN library was called.

APPENDIX D

COMPATIBILITY WITH OTHER PDP-11 LANGUAGE PROCESSORS

FORTRAN IV is a new implementation of FORTRAN for the PDP-11, available under RT-11, RSTS/E, RSX-11M, RSX-11D and IAS. It has been designed to be upward compatible with FORTRAN IV PLUS (F4P) and compatible with the earlier FORTRAN (FTN) on DOS-11 and RSX-11D V4A. However, some differences exist as a result of:

1. correcting deficiencies in FTN FORTRAN.

2. language specification decisions necessary to promote the goal of an upward compatible line of FORTRANS, and

3. providing significant extensions to FTN FORTRAN in a manner consistent with the FORTRAN language, the ANSI FORTRAN Standard and existing FORTRANs.

D.1 FORTRAN IV COMPATIBILITY WITH FTN V08.04

This section summarizes differences which may affect conversion from FTN to FORTRAN IV.

Items marked with an asterisk (*) denote differences from FTN which are common to both FORTRAN IV PLUS (F4P) and FORTRAN IV.

D.1.1 Language Differences

*1. FTN permits transfer of control to FORMAT statements, which execute as CONTINUE statements. FORTRAN IV does not, and issues compile-time error messages.

*2. FTN V07.14 provided an uncounted form for Radix-50 constants used in DATA statements. V08.04 added the counted form and promised de-support of the uncounted form. Only the counted form is provided in FORTRAN IV.

3. In FORTRAN IV, the syntax of the IMPLICIT statement is different from that required by FTN.

4. FTN provides expressions in FORMAT statements (called variable format expressions); FORTRAN IV does not.

5. FTN permits Hollerith constants containing one or two characters to be used in expressions (as integer type).

FORTRAN IV permits Hollerith constants in DATA, CALL, FORMAT, STOP, and PAUSE statements only. Data initialized variables may be used equivalently elsewhere.

D.1.2 Implementation Differences

*1. FTN allows both formatted and unformatted records in the same file by means of the MXDFUF subroutine call. This feature is not supported in FORTRAN IV.

*2. The FTN service subroutines PDUMP and SETPDU are not provided in FORTRAN IV. Similar but more flexible debugging output may be obtained by using WRITE statements in debug lines.

*3. The TRCLIB library for statement level execution tracing is not available in FORTRAN IV.

*4. Under FTN, Q format returns the number of characters in the input record (i.e., the record length) independent of the current scan position. Under FORTRAN IV, Q format returns the number of characters remaining in the input record following the scan position. The record length may be obtained by using the Q format first in the format specification.

*5. FTN performs the following actions when opening a unit for direct access I/O: first it checks if a file already exists; if so, it uses it, if not, it creates a new file for use.

In FORTRAN IV, the type of open depends on whether a READ or WRITE statement is causing the open operation. If it is a READ, then the file must already exist or an error results. If it is a WRITE, then a new file is created.

*6. The implementation of error handling in FORTRAN IV is significantly different from that in FTN. In particular, the use of error classes for controlling error handling is replaced by control over each individual error condition.

*7. FTN does not compile format specifications for use at run-time, whereas FORTRAN IV does. One difference results:

> Format specifications stored in arrays are recompiled at run-time each time they are used. If an H format is used in a READ statement to read data into the format itself, that data does not get copied back into the original array. Hence, it will not be available on a subsequent use of that array as a format specification. This is in accord with the ANSI FORTRAN language specification.

This consideration does not apply to format specifications defined in FORMAT statements.

*8. FTN implements the ENDFILE statement as a close operation. FORTRAN IV implements the ENDFILE statement by writing an end of file record in a file.

9. FORTRAN IV checks that the labels used in an assigned GOTO are valid labels in the program unit, but it does not check

at run-time whether an assigned label is in the list in the GOTO statement. FTN checks at run-time.

10. The FTN /-ON compiler option switch causes two-word allocation (instead of one-word allocation) of all INTEGER and LOGICAL variables. The similar FORTRAN IV /T compiler option switch affects allocation of unspecified size INTEGER variables only; unspecified LOGICAL variables are always allocated two words in FORTRAN IV.

11. The FTN and F4P implementations of adjustable dummy arrays copy dummy argument dimension variables into an internal array descriptor upon entry to a subprogram. A subsequent assignment to a dummy argument dimension variable within that subprogram does not affect the array subscript calculation.

    FORTRAN IV does not use array descriptor blocks but rather references dummy argument dimension variables directly during subscript calculations. Hence, an assignment to a dummy argument dimension variable will affect array subscript calculations.

D.2 DIFFERENCES BETWEEN FORTRAN IV-PLUS AND FORTRAN IV

This section summarizes differences that may affect conversion from FORTRAN IV to FORTRAN IV PLUS.

D.2.1 Language Differences

1. F4P transforms FORTRAN defined function name references into a special kind of internal calling form, while FORTRAN IV does not. If a user supplies a routine to replace the F4P FORTRAN defined routine (for example, writes a SIN routine) it will generally be necessary to include EXTERNAL statements (with the user name prefixed by an asterisk, '*') to cause that routine to be referenced. This is not necessary in FORTRAN IV.

D.2.2 Implementation Differences

1. FORTRAN IV logical tests treat any non-zero bit pattern in the low-order byte of a LOGICAL variable as .TRUE., and an all-zero bit pattern as .FALSE.

   F4P tests only the highest-order bit of the value and treats a one as .TRUE. and a zero as .FALSE.

2. FORTRAN IV, like FTN, does not enforce the restrictions stated in the PDP-11 Language Reference Manual concerning modifying the control variable and/or the parameters of a DO loop within the body of the loop.

3. FORTRAN IV, like FTN, defines named COMMON blocks in terms of named .CSECTs.

4. In FORTRAN IV, INTEGER*4 causes 32-bit allocation (4 bytes), but only 16 bits are used for computation. In F4P, INTEGER*4 causes both 32-bit allocation and 32-bit computation.

5. FORTRAN IV checks that the labels used in an assigned GOTO are valid labels in the program unit, but it does not check at run-time whether an assigned label is in the list in the GOTO statement. F4P does check at run-time.

6. FORTRAN IV permits an unlimited number of continuation lines. In F4P, up to 5 continuation lines are permitted by default, and up to 99 may be obtained by means of the compiler /CO switch.

7. For unformatted input/output operations, both sequential and direct-access, FORTRAN IV reads/writes four bytes of data if the variable is allocated four bytes of storage. F4P does also. However, since INTEGER*4 values in FORTRAN IV generally have an undefined high-order part, they generally may not be read as INTEGER*4 values by F4P. Similarly, since FORTRAN IV and F4P logical tests are different (see item 1 above), care must be taken when interchanging logical values.

8. For random-access input/output operations, FORTRAN IV takes the END= exit on an end-file condition and the ERR= exit only for hardware I/O errors. F4P ignores any END= specification and takes the ERR= exit for both.

D.3 RSTS/E FORTRAN IV FILE COMPATIBILITY

RSTS/E FORTRAN IV can read and write files compatible with the other language processors available on the RSTS/E operating system (BASIC-PLUS and COBOL). Certain file formats, however, are not supported under FORTRAN IV. Also, care must be taken when creating files with FORTRAN if they are to be processed by a program written in another language.

D.3.1 Sequential Stream ASCII Files

All three language processors available on RSTS/E share the concept of sequential stream files composed of ASCII data. In FORTRAN, this type of file is accessed with formatted READ and WRITE statements. BASIC-PLUS processes stream files through the INPUT, INPUT LINE, and PRINT statements. COBOL sequential stream files, which do not contain any binary data, also fall into this category.

A stream ASCII file consists of variable-length data records terminated by a carriage control sequence, usually CR - LF. No extraneous formatting information (such as byte counts or checksums) is included in the file.

BASIC-PLUS programs create this type of file by default. The COBOL language processor will create compatible files if no binary data (e.g., COMPUTATIONAL variable type) is written. COBOL data of type DISPLAY is usually acceptable in a stream ASCII file. FORTRAN will accept files structured in this fashion in formatted READ statements.

To create compatible files with the FORTRAN formatted WRITE statement, care must be taken to assure proper positioning of carriage control information. By default, FORTRAN records begin with the vertical forms control characters required for the record to be written, followed by the data record itself, and terminated by a carriage return character. For example, the statements:

```
        TYPE 100
100     FORMAT(' HELLO')
```

will create the following record:

```
        <LF>HELLO<CR>
```

where <LF> represents the line feed character and <CR> denotes the carriage return character. It is desirable to create records terminated by the carriage return-line feed sequence for compatibility. The previous example may be rewritten to achieve this:

```
        TYPE 200
200     FORMAT('+HELLO'/)
```

The '+' character suppresses the initial line feed generated by default, and the '/' record terminator causes the line feed to be generated at the end of the record, as desired.

Note that the above technique applies only to files which have been created with the 'CC' attribute in CALL ASSIGN or CALL OPEN. By default, files output to non-printing devices have carriage control translation suppressed. Such files consist of an initial line feed character, followed by records in the standard stream ASCII format (i.e., terminated by CR-LF sequences).

D.3.2 Virtual Array Files

The BASIC-PLUS language provides the capability to create files which are accessed from the program as arrays of data elements. All 'records' in the file are of a fixed length, usually one integer or floating-point value. String virtual arrays are also provided.

FORTRAN programs can read and write virtual array files in a very straightforward fashion. For example, consider the integer virtual array referenced by the following BASIC-PLUS dimension statement:

```
        DIM #chan,I%(9999%)
```

To read this array file from FORTRAN, the programmer would first describe the file format with the DEFINE FILE statement as follows:

```
        DEFINE FILE unit (10000,1,U,ivar)
```

Note that this statement establishes the size of each record as one word (the second argument inside parentheses), and indicates that the number of such records in the file is 10000. The file may now be accessed by the statement:

```
        READ (unit'index)ivalue
```

where index represents the subscript to be used (note that FORTRAN

subscripts begin at one, whereas BASIC-PLUS uses zero-origin indexing). The value of the selected element will be read into the integer variable specified by ivalue. To handle virtual arrays of floating point values, the same format for the DEFINE FILE statement is used, replacing the size of the record in words with the value appropriate to the math package used by the BASIC-PLUS program in question. (If the two-word math package was used to create the file, the value 2 should be specified as the second argument inside parentheses; 4 is appropriate to the four-word math package.)

To handle two-dimensional virtual arrays, the DEFINE FILE statement must be coded to account for the total number of elements in the array. For example, the following BASIC-PLUS dimension statement allocates a two-dimensional integer array:

```
DIM #chan,I%(m,n)
```

where m and n specify the array dimensions (remember that indexing starts at zero in BASIC). The equivalent DEFINE FILE statement is:

```
DEFINE FILE unit ((m+1)*(n+1),1,U,ivar)
```

The expression (m+1)*(n+1) computes the number of array elements specified by the previous DIM statement. To access this array as done in the following BASIC-PLUS line:

```
I1% = I%(J%,K%)
```

the FORTRAN programmer must compute the vector index as:

```
READ (unit'(J*(n+1) + K + 1))I1
```

To access a string virtual array, the DEFINE FILE statement should specify the maximum string size divided by two (as element sizes are specified in words). For example, the BASIC-PLUS statement:

```
DIM #chan,A$(100%) = 128%
```

would be equivalent to the DEFINE FILE statement:

```
DEFINE FILE unit (101,64, U, ivar)
```

The strings may be stored in the FORTRAN program in LOGICAL*1 arrays of the appropriate length. The following FORTRAN code will read one element of the virtual string array specified in the above example:

```
LOGICAL*1 STRING(128)
READ (unit'index) STRING
```

Strings stored in virtual array files are padded on the right with null characters (000 octal) to the specified record length.

For more information on string array refer to the BASIC-PLUS Language Reference Manual, Chapter five.

D.3.3 BASIC-PLUS Record I/O Files

BASIC-PLUS record I/O files may be accessed from FORTRAN programs using the direct-access I/O facility. The DEFINE FILE statement should specify the number of words in each logical record of the file as the size. FORTRAN direct-access I/O demands that all records of the file have the same length. If this is true of the record I/O file in question, the FORTRAN system will do all record blocking/deblocking for the user. If the file in question has records of several different sizes, the programmer should read the file on a block-by-block basis.

To read the file on a block-by-block basis, the DEFINE FILE statement specifies 256 as the record size. Each record is read into a 256-word array area (usually an INTEGER*2 array of 256 elements). The programmer must then find the records desired in each block by indexing through the buffer array.

When accessing values stored using the CVT%$ and CVTF$ functions, or when creating files which will be read by BASIC-PLUS programs which will use the CVT$% and CVT$F functions to retrieve values, special care must be taken. The CVT%$ and CVTF$ functions store the binary values byte-reversed from the normal orientation expected by FORTRAN (and the representation in virtual array files). For example, the function call CVT%$(13%) returns the value 000,015 as octal byte values, (with 000 as the low-order byte and 015 as the high-order byte). Hence, the FORTRAN programmer must reverse the bytes of each word of a value written by a BASIC-PLUS program which uses these functions. To perform the required byte-reversal operation on a floating-point value which has been read into the variable A, the following code can be used:

```
REAL*4 A
LOGICAL*1 TEMP(4),T
EQUIVALENCE (A,TEMP)
T = TEMP(1)
TEMP(1) = TEMP(2)
TEMP(2) = T
T = TEMP(3)
TEMP(3) = TEMP(4)
TEMP(4) = T
```

When writing files which are to be read by BASIC-PLUS, the byte-reversal must also be performed.

D.3.4 COBOL Files

As noted above, sequential stream ASCII files produced by COBOL can be processed by FORTRAN IV using formatted I/O. These files must not, however, contain any binary data, as a special format is used to represent COBOL records containing such values. Accessing files with the COBOL relative organization is not supported from FORTRAN IV due to the complex internal structure of such files.

D.3.5 IAM Files

The IAM (Indexed Access Method) file organization available under BASIC-PLUS is not currently supported in the FORTRAN IV environment.

INDEX

Page numbers are underlined, in some cases, to point the user to the major or definitive occurrence of an indexed item.

RT-11/RSTS/E FORTRAN IV
User's Guide
DEC-11-LRRUA-A-D

READER'S COMMENTS

NOTE: This form is for document comments only. Problems with software should be reported on a Software Problem Report (SPR) form.

Did you find errors in this manual? If so, specify by page.

_____

_____

_____

_____

_____

_____

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

_____

_____

_____

_____

_____

_____

Is there sufficient documentation on associated system programs required for use of the software described in this manual? If not, what material is missing and where should it be placed?

_____

_____

_____

_____

_____

_____

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

- ☐ Assembly language programmer
- ☐ Higher-level language programmer
- ☐ Occasional programmer (experienced)
- ☐ User with little programming experience
- ☐ Student programmer
- ☐ Non-programmer interested in computer concepts and capabilities

Name _____ Date _____

Organization _____

Street _____

City _____ State _____ Zip Code or Country _____

If you require a written reply, please check here. ☐

Please fold on dotted line.

---------- Fold Here ----------

---------- Do Not Tear - Fold Here and Staple ----------

FIRST CLASS
PERMIT NO. 33
MAYNARD, MASS.

BUSINESS REPLY MAIL
NO POSTAGE STAMP NECESSARY IF MAILED IN THE UNITED STATES

Postage will be paid by:

digital

Software Communications
P. O. Box F
Maynard, Massachusetts 01754

DIGITAL EQUIPMENT CORPORATION
MAYNARD, MASSACHUSETTS 01754