pdp11

RT-11 FORTRAN IV User's Guide

Order Number: AA-JQ68A-TC

digital

RT-11 FORTRAN IV User's Guide

Order Number: AA-JQ68A-TC

This document describes the operating procedures for the FORTRAN IV compiler and Object Time System (OTS) library under the RT-11 V5.4 operating system. In conjunction with the *PDP-11 FORTRAN IV Language Reference Manual*, this document provides the information required to write and run a FORTRAN IV program under RT-11 V5.4.

Operating System and Version:	RT-11 V5.4
Software Version:	FORTRAN IV, Version 2.8

digital equipment corporation maynard, massachusetts

January, 1987

DEC	EduSystem	RSTS
DEC/CMS	FMS-11	RSX
DEC/MMS	FMS-11/RSX	RT-11
DECnet	IAS	UNIBUS
DECsystem-10	MASSBUS	VAX
DECsystem-20	MicroPDP-11	VAXcluster
DECUS	Micro/RSX	VMS
DECtape	PDP	VT
DECwriter	PDT	digital
DIBOL	Professional	
DIGITAL	RMS	

This document was prepared using an in-house documentation production system. All page composition and make-up was performed by $\TeX$, the typesetting system developed by Donald E. Knuth at Stanford University. $\TeX$ is a trademark of the American Mathematical Society.

Contents

Preface

This release of RT-11 FORTRAN IV does not provide any new functionality. Rather, it fixes existing bugs.

Associated Documents

The following documents are relevant to RT-11 FORTRAN IV language programming:

- *PDP-11 FORTRAN IV Language Reference Manual*
- *RT-11 System User's Guide*

Conventions Used in This Document

All monitor and system program command lines are terminated by pressing the RETURN key. Since this is a nonprinting character, the notation <RET> represents the RETURN key in command lines.

In examples, user responses shown in uppercase characters indicate that you should type the characters exactly as shown.

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 an up arrow (for example, ^Z) or CTRL/Z.

Ellipsis marks (... or vertical dots) indicate the omission of one or more words within a passage or lines of code within a code example, and show that the passage or code example continues in the same vein.

Structure of This Document

This manual is organized as follows:

- Chapter 1, "Operating Procedures," contains the information needed to compile, link, and execute FORTRAN IV language programs. It includes a new section about virtual jobs and covers additional linker options.
- Chapter 2, "FORTRAN IV Operating Environment," describes how to use the facilities of the PDP-11 FORTRAN IV Object Time System (OTS) library. It includes information about virtual and vectored arrays, program sections, and runtime error detection and memory organization.

- Chapter 3, "FORTRAN IV Specific Characteristics," discusses FORTRAN IV access methods and I/O, including information on logical device assignments and record structure.
- Chapter 4, "Increasing FORTRAN IV Programming Efficiency," covers programming considerations relevant to typical FORTRAN IV language applications.
- Appendix A, "FORTRAN IV Data Representation," summarizes internal data representation.
- Appendix B, "Library Subroutines," describes user-accessible FORTRAN IV Library subroutines.
- Appendix C, "FORTRAN IV Error Diagnostics," describes compiler and OTS library diagnostics messages.
- Appendix D, "Compatibility with FORTRAN-77," covers FORTRAN language and implementation differences.
- Appendix Awe, "The FORTRAN IV System Simulator ($SIMRT)."

Intended Audience

Ideally, this manual should be used only after you acquire some knowledge of the FORTRAN IV language as implemented on the PDP-11 computer system. For this purpose, you can use the *PDP-11 FORTRAN IV Language Reference Manual.* You should also be familiar with the RT-11 operating system as described in the *RT-11 System User's Guide.* The *RT-11 Documentation Directory* contains additional information on the respective documentation sets.

1 OPERATING PROCEDURES

1.1 USING THE FORTRAN IV SYSTEM

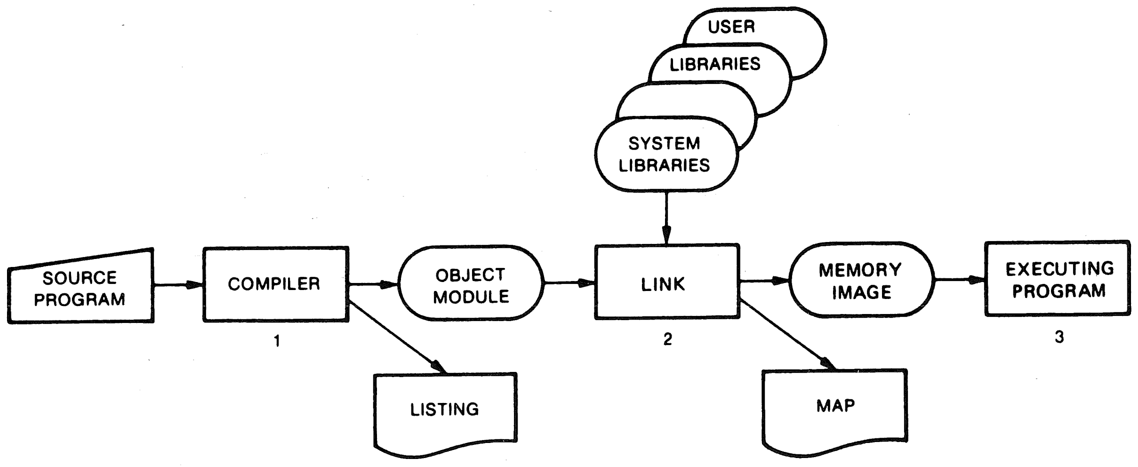
Figure 1-1 outlines the steps required to prepare a FORTRAN IV language source program for execution under the RT-11 executive:

1 Compilation

2 Linking

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, with a command string that describes the input and output files, and desired options to be used by the compiler. The compiler generates an object file that must be linked by the linker prior to execution.

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

Step 3 is initiated by the monitor RUN command.

1.1.1 Compiler-Generated Code

The FORTRAN IV compiler translates the symbolic (FORTRAN) program into an object program in binary form, which results in machine instructions. If you find that procedures and instructions in a high-level language like FORTRAN IV sometimes restrict your freedom to handle data as you would like, you can insert an assembly language routine. This process is discussed in more detail in Section 2.3.

The FORTRAN IV compiler produces two types of object code for a program:

- inline (PDP-11 machine language)
- threaded (linked OTS library references)

See Section 2.2 for more information about the object code produced. The default value assumed for the /CODE (or /I) option (see Section 1.1.1.1) is determined at installation time to satisfy the configuration in which the compiler is installed.

1.1.1.1 Code Options

Inline code is selected for RT-11 through the keyboard monitor command options /CODE:EAE, /CODE:EIS, or /CODE:FIS.

Be careful to select the option suitable to the available hardware configuration. Some configurations will not support execution of inline code. Table 1-1 shows valid options for certain configurations.

Table 1-1 Code Generation Options

	Arithmetic Hardware Options			
Target System	**KE11-A,B**	**KE11-E**	**KE11-F, KEV11**	**Valid Code Options /CD:**
LSI-11, 11/03	-	-	NO	THR
	-	-	YES	EIS, FIS, or THR
11/04, 11/05, 11/10,	NO	-	-	THR
11/15, 11/20	YES	-	-	EAE or THR
11/35, 11/40	NO	NO	NO	THR
	YES	NO	NO	EAE or THR
	NO	YES	NO	EIS or THR
	NO	YES	YES	EIS, FIS, or THR
11/23, 11/23+, 11/24, 11/34, 11/44, 11/45, 11/50, 11/53, 11/55, 11/60, 11/73, 11/83, 11/84	-	-	-	EIS or THR

1.1.1.2 Code Selection and Error Messages

When the compiler produces inline code (/CODE: [/I:] followed by EAE, EIS, or FIS) the object program executes at greater speed and generally uses less physical memory. Inline code achieves this optimization, in part, by omitting instructions to detect or report certain error conditions. However, you can generate code for error checking by including the /CODE:THR (/I:THR) option in the compiler command line. Table 1-2 demonstrates the diagnostic benefits of the threaded code option.

Table 1-2 Threaded Code Error Messages

	Result	
Error	**Inline Code**	**Threaded Code**
1. The result of an integer multiply operation cannot be expressed as a one-word integer.	No diagnostic message is produced. Execution continues and the result of the operation is truncated to 16 bits.	A fatal error occurs and the diagnostic message: "?Err 1 Integer Overflow" is produced.
2. A divide by zero occurs during integer arithmetic.	No diagnostic message is produced and the result of the operation is undefined.	A fatal error occurs and the diagnostic message "?Err 2 Integer zero divide" is produced.
3. The value of the arithmetic expression of a computed GOTO is less than one or greater than the number of labels in the list.	No diagnostic message is produced. Execution continues at the next executable statement.	The warning diagnostic "?Err 4 Computed GOTO out of range" is produced. Execution resumes at the next executable statement.

1.1.2 File Name Specifications

The FORTRAN and LINK commands, respectively, pass file name specifications to the FORTRAN IV compiler and linker. The designator .TYP (type) is used for RT-11.

See Table 1-3 for device specifications, and Section 1.2.2 for compiler options.

Each file name specification (filespec) has the form:

RT-11—dev:filnam.typ

where:

dev: = An optional two- or three-character name specifying a legal device code as shown in Table 1-3 for a logical device name. If the device code is omitted, the default storage (DK:) is used for RT-11.

filnam = Any one to six character alphanumeric file name.

.typ (RT-11) = Any zero to three character alphanumeric extension. If one is not specified, the FORTRAN IV compiler supplies, by default, certain extensions as shown in Table 1-4.

Table 1-3 Device Specifications for RT-11

Device	RT-11
Card reader	CR:
TA11 cassette (n = 0 or 1)	CTn:
Default storage	DK:
RK05 disk (n = 0 to 7)	-
RK06 or RK07 disk (n = 0 to 7)	DMn:
RP02 or RP03 disk	DPn:(n=0 to 1)
RL01 or RP02 disk	DLn:(n=0 to 4)
RM02 or RM03 disk (n = 0 to 1)	-
RP04, RP05, or RP06 disk (n = 0 to 7)	-
RS03 or RS04 disk (n = 0 to 7)	DSn:
TC11 DECtape	DTn:(n=0 to 7)
TU58 DECtape II	DDn:(n=0 to 4)
PDT-11/130 DECtape II (n = 0 to 1)	PDn:
RX01 floppy disk	DXn:(n=0 to 3)
PDT-11/150 floppy disk	PDn:
Serial line printer	LS:
RX02 floppy disk	DYn:(n=0 to 3)
Line printer	LP:
TU16, TE16, TU45, or TU77 magtape (n = 0 to 7)	MMn:
TU10, TE10, or TS03 magtape (n = 0 to 7)	MTn:
TS11 magtape (n = 0 to 3)	MSn:
High-speed paper tape punch	PC:
High-speed paper tape reader	PC:
RF11 fixed-head disk drive	RF:
System device	SY:
Specified unit from which the system was bootstrapped	SYn:
Current user terminal	TT:
Auxiliary terminal	-

For more information on device specifications, refer to the *RT-11 System User's Guide*.

Table 1-4 File Name Types (Extensions)

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

Table 1-5 File Protection Codes

Code	Meaning
1	Read protection against owner
2	Write protection against owner
4	Read protection against owner's project group
8	Write protection against owner's project group
16	Read protection against all others who do not have owner's project number
32	Write protection against all others who do not have owner's project number
64	Executable program: can be run only
	Individual codes added to the compiled protection <64> have meanings different from those of the data file protection codes above. These compiled codes follow:
1	Execute protection against owner
2	Read and write protection against owner
4	Execute protection against owner's project group
8	Read and write protection against owner's project group
16	Execute protection against all others who do not have owner's project number
32	Read and write protection against all others who do not have owner's project number
128	Program with temporary privileges (normally occurs only when file's protection includes <64>)

1.1.3 Locating a File

The FORTRAN IV compiler locates a file by searching the specified device for the file name with the specified file type (RT-11) when the device is not specified and assumes .FOR when a file type is not specified.

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

```
?FORTRAN-F-FILE NOT FOUND
```

A similar form of this message appears if a file name specification given to a utility program (for example, MACRO, LINK) references a file that cannot be found or is protected against the user.

1.2 RUNNING THE FORTRAN IV COMPILER

The FORTRAN IV compiler accepts a command string in the form:

output = input/option (/sw)

where:

output = The output file name specification(s)

input = The input file name specification(s)

/option (RT-11) = One or more options used to request certain functions from the FORTRAN IV compiler. Options are explained in Section 1.2.2. Options can be appended to any file specification in the command string.

Note that embedded blanks are not permitted in command string specifications.

1.2.1 Under RT-11 (with Keyboard Monitor Commands)

To compile a FORTRAN IV program, give either of these commands:

```
FORTRAN[/option...] filespec[/option...] [...filespec[/option...]]
```

or

```
FORTRAN[/option...]
FILES? filespec[/option...][...filespec[/option...]]
```

where:

filespec = A source file to be compiled

When more than one file is listed for a single compilation, separate the files by plus (+) signs. FORTRAN IV will create an output file with the same name as the first input file and give it a .OBJ file type. However, if you separate the input files by commas, FORTRAN IV will produce an .OBJ file for each input file listed.

For example:

FORTRAN/LIST/SHOW:ALL/NOLINENUMBERS TEST1+TEST2

compiles TEST1.FOR and TEST2.FOR together and produces TEST1.OBJ. A listing of the compilation complete with source, storage map, and code listings will be sent to the line printer device, LP:. Generation of internal sequence numbers (ISNs) in the object program will be disabled.

1.2.2 Compiler Options (Switches)

The FORTRAN IV compiler command strings use specified options (switches) of either octal or decimal values on the input and output file specifications. Any option in the form /S:n causes n to be interpreted as an octal value (as long as n contains only the digit 0-7); any option in the form /S:n. causes n to be interpreted as a decimal value.

RT-11 and RSTS/E FORTRAN IV compiler options (switches) are described in Table 1-6.

Table 1-6 Compiler Options (Switches)

RT-11 Switch	RSTS/E Switch[1]	Explanation[2]
/ALLOCATE:n	-	Used after the /OBJECT or /LIST option to guarantee space for a maximum file size of n blocks
/CODE:xxx	/I:xxx	Selects type of object code to be generated. Defaults to value selected at installation. The proper values are: EAE (selects EAE hardware) EIS (selects EIS hardware) FIS (selects EIS & FIS hardware) THR (selects threaded code)
/DIAGNOSE	/B	Enables expanded listings of compiler internal diagnostic information
/EXTEND	/E	Allows source line input from columns 73-80
/HEADER	/O	Prints an "Options-in-Effect" section prefacing the listing
/I4	/T	Defaults to two-word integers (I*4). (Normally defaults to one-word integers—I*2.)
/LINENUMBERS	-	Indicates internal sequence numbers are to be included in the executable program for routine diagnostics
/LIST[:filespec]	-	Generates a listing. A file name can be optionally specified.
/NOLINENUMBERS	/S	Suppresses generation of internal sequence numbers
/NOOBJECT	-	Does not generate object files
	/Q[1]	Inhibits printing names of program units (from program, FUNCTION, SUBROUTINE, and BLOCK DATA statements) as each program unit is compiled. Note that .MAIN. refers to the main program and .DATA. refers to an unnamed BLOCK DATA.
/NOSWAP	/U	Disables USR swapping at run time
/NOVECTORS	/V	Suppresses array vectoring of multidimensional arrays
/OBJECT[:filespec]	-	Produces an object file (default). The destination for the object file can be optionally specified.
/ONDEBUG	/D	Compile lines with a "D" in column one for debugging purposes
	/Z[1]	Causes pure code and pure data sections to take R (read-only) attribute
/RECORD:n	/R:n	Specifies the maximum record length (in bytes) on run time I/O ($4<n<4095$)

[1]RT-11 users can use the RSTS/E switch when the compiler is invoked with the RUN command.

[2]Defaults are determined at installation time.

Table 1-6 (Cont.) Compiler Options (Switches)

RT-11 Switch	RSTS/E Switch[1]	Explanation[2]
/SHOW[:n]	/L[:n]	Specifies the listing options. The argument n is encoded as follows: 0 or null—list diagnostics only 1 or SRC—list source program and diagnostics only 2 or MAP—list storage map and diagnostics only 4 or COD—list generated code and diagnostics only Any combination of the above list options can be specified by summing the numeric argument values for the desired list options. For example: 7 or ALL requests a source listing, a storage map, and a generated code listing. If this option is omitted, the default option is /SHOW:3, (/L:3) source and storage map.
/STATISTICS	/A	Prints compilation statistics
/SWAP	-	Allows the USR to swap over the FORTRAN IV program (default)
/UNITS:N	/N:n	Allows a maximum of n simultaneously open I/O channels at run time (l<n<15)
/VECTORS	-	Uses tables to access multidimensional arrays (default)
/WARNINGS	/W	Enables compiler warning diagnostics; used in conjunction with /SHOW (/L) option
	/X:xxx[1]	Indicates cross-compilation for the target environment specified. Compiler diagnostic messages will be generated as if compilation had occurred under the foreign environment. Values are: RT (selects RT-11) RST (selects RSTS/E) RSX (selects RSX-11)

[1]RT-11 users can use the RSTS/E switch when the compiler is invoked with the RUN command.

[2]Defaults are determined at installation time.

1.2.3 Listing Formats

You can direct the compiler to furnish any combination of five optional sections in the compilation listing. Use the options (switches) described in Section 1.2.2 to call for the list of options in effect, the generated code, and the compiler statistics. The source program and the storage map are included by default. The following sample compilation listing describes each section and gives an example of the information included.

```
FORTRAN IV      V02.8      Fri 19-Sept-86 00:40:58

,EX2=EX2/L:ALL/I:THR/O/A

OPTIONS IN EFFECT:
```

```
          SOURCE
          MAP
          CODE
          LEAPYEAR
        NOREADONLY
          LRECL=0136
          STAT
          ISNS
        NOCOL80
          USRSWAP
        NODIAGNOSE
        NOINTEGER*4
          NLCHN=06
        NODEBUG
          VECTOR
        NOWARN
          CODE:THR
          LOG
```

```
FORTRAN IV      V02.8      Thu 19-Sept-86 00:40:58

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. +3.1415926535D0
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 for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000030 (   12. words)

Name   Type  Offset     Name   Type   Offset     Name   Type   Offset
DBLE   R*8   000020     IMAG   C*8    000010     INT    I*2    000002
REAL   R*4   000004

Subroutines, Functions, Statement and Processor-Defined Functions:

Name   Type  Name    Type    Name    Type    Name    Type    Name  Type
CMPLX   C*8


FORTRAN IV      Generated Code for Program Unit .MAIN.

Statement #0006
000006  LSN$       #000006
000012  MOI$MS     $DATA+#000002
000016  DII$IS     #000002
000022  CFI$
000024  ADF$IS     #040640
000030  MOF$SM     $DATA+#000004

Statement #0007
000034  ISN$
000036  MOF$MS     $DATA+#000004
000042  DIF$IS     #040400
000046  CDF$
000050  ADD$MS     $DATAP+#000020
000054  MOD$SM     $DATA+#000020

Statement #0008
000060  ISN$
000062  REL$       $DATAP+#000030
000066  REL$       $DATA+#000004
000072  CAL$       #000002 CMPLX+#000000
000100  MOD$RM     $DATA+#000010
```

```
Statement #0009
000104  ISN$
000106  REL$      $DATAP+#000016
000112  REL$      $DATAP+#000010
000116  IFW$
000120  REL$      $DATA+#000010
000124  TVC$
000126  EOL$

Statement #0011
000130  LSN$      #000013
000134  STP$


Compilation Statistics:

Symbol table size: 00091 words
Internal form size: 00039 words
Free dynamic memory: 19983 words

Compilation time: 00:00:01
```

FORTRAN IV listings of generated code list the first instruction starting at location 6 of the procedure. This section explains the two instructions generated that are not listed.

The first two lines of code are not generated in the FORTRAN IV listings for either subroutines or the main segment.

The two lines generated for the main program unit are:

```
JSR     R4,$$OTI            (Inline code)
.WORD   NAMPTR
```

or

```
JSR     R4,$OTI             (Threaded code)
.WORD   NAMPTR
```

The location that NAMPTR contains is the address of the two-word segment name in RAD50, which is the name of the main program. If no main-program name is specified by means of a PROGRAM statement, the default main-program name, .MAIN., is used.

The two lines generated in a subprogram unit are:

```
JST     R4,$OTIS
.WORD   NAMPTR
```

The location that NAMPTR contains is the address of the two-word segment name in RAD50, which is the name of the routine.

1.2.3.1 Options Listing

Use the options-in-effect list as a quick reference to the status of each possible compiler option. Those preceded by "NO" are not in effect. The maximum number of logical units that can be open concurrently (NLCHN) and the maximum record length (LRECL) are given as the default values or the values specified by the /UNITS:n (/N:n) and /RECORD:n (/R:n) options respectively. The day of the week, date, and time of compilation and a copy of the compiler command string for identification purposes are also furnished.

1.2.3.2 Source Listing

This section lists the source program as it appeared in the input file. The compiler adds internal sequence numbers for easier reference. Note that internal sequence numbers are not always incremented by 1. For example, the statement following a logical IF has an internal sequence number two greater than that of the IF, because the compiler assigns one for the comparison and one for the associated statement.

1.2.3.3 Storage Map Listing

This section lists all symbolic names referenced by the program unit. Local variables are allocated in the $DATA psect. The addresses of parameter variables and arrays are placed in the $DATA psect at subroutine invocation and are denoted by "@" preceding the offset or section name. The listing includes the symbolic name, data type, usage, psect, and offset. In the case of COMMON blocks, virtual arrays, and array names, the listing includes the defined size in bytes (octal) and words (decimal) as well as the dimensions.

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

1.2.3.4 Generated Code Listing

This section contains:

- A symbolic representation of the object code generated by the compiler (see Section 2.2).
- A location offset into psect $CODE.
- The symbolic Object Time System (OTS) library routine name.
- Routine arguments of threaded code plus the equivalent assembler code for the inline code. When the /DIAGNOSE (/B) compiler option is used, the right-hand column of the inline generated code listing shows those registers available for use following an operation. The code generated for each statement provides easy cross-reference by showing the same internal sequence number (ISN) as was specified in the source program listing.

1.2.3.5 Compilation Statistics

This section provides a report on memory usage during the compilation process and elapsed wall-clock time for the compilation.

1.2.4 Compiler Memory Requirements

Under the RT-11 operating system, device handlers and the symbol table require a portion of memory during compilation. If you require more memory after minimizing the number of different physical devices and variable names specified, you can segment the program into program units small enough to compile in the available space.

1.2.4.1 Compiler Memory Requirements Under RT-11

During compilation, the following must reside in main memory: 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). Note that FORTRAN IV will dynamically load device handlers required for compilation. The remaining memory provides for the symbol table and the internal representation of the program. In a machine with 8K words of memory, this allows the compilation of a program unit as large as several hundred statements. However, if the compiler runs out of memory during compilation, an error message is delivered to the user's terminal; see Section C.1. The program must be divided into two or more program units, each small enough to compile in the available memory.

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

1.3 LINKING PROCEDURES

When SYSLIB is created under RT-11 to include FORLIB, the /LINKLIBRARY:FORLIB (/F) option is redundant, because FORLIB is part of SYSLIB and is not called separately.

1.3.1 Linking Under RT-11

The RT-11 linker, LINK, combines one or more user-written program units with selected routines from any user libraries and the default FORTRAN IV OTS library to form the default system subroutine library, SYSLIB. LINK generates a single runnable memory image file and an optional load map from the one or more object files created by the MACRO assembler or the FORTRAN IV compiler.

The default types for the executable file are .SAV for a background or mapped environment program, and .REL for a foreground program. The default output device is DK:.

The default name of the .SAV or .REL file is that of the first concatenated input object file specified. When FORLIB resides in SYSLIB, the required elements of the FORTRAN IV library will be linked automatically since any undefined global references are correlated and resolved through SYSLIB.

The LINK command adheres to the syntax:

```
LINK[/option...] filespec[/option...][,...filespec[/option...]]
```

or

```
LINK[/option...]
FILE?  filespec[/option...][,...filespec[/option...]]
```

where "filespec" represents the file to be linked and "options" are those described in Table 1-7.

Table 1-7 Linker Options Available Under RT-11

Option	Explanation
/ALLOCATE:n	Guarantees space for a maximum file of n blocks
/ALPHABETIZE	Lists program's global symbols alphabetically in the load map
/BITMAP	Creates a memory usage bitmap (default setting)
/BOTTOM:n	Specifies a bottom address for a background program
/BOUNDARY:value	Starts a specific program section on a particular address boundary. Argument value must be a power of 2. Prompts you: Boundary section? Enter name of section, then <RET>
/DEBUG[:filespec]	Links ODT to the linked program
/EXECUTE[:filespec]	Designates the executable file
/EXTEND:n	Extends a program section to octal value n. Prompt: Extend section?
/FILL:n	Initializes unused locations in the load module to n (an octal value)
/FOREGROUND[:stacksize]	Generates a .REL file for a foreground link
/INCLUDE	Allows subsequent entry at the keyboard of global symbols to be taken from any library and included in the linking process. When the /INCLUDE option is typed, the linker prints: Library search? Reply with the list of global symbols to be included in the load module. Press the carriage return key <RET> to enter each symbol in the list.
/LDA	Produces executable file in LDA format for use with the Absolute Loader
/LIBRARY	Same as /LINKLIBRARY (included for compatibility with other systems)
/LINKLIBRARY:filespec	This option is ignored unless a file specification is typed. The file specification is included as an object module library in the linking operation.
/MAP[:filespec]	Produces a link map on the listing device LP: or in the file specified
/NOEXECUTE	Does not create a .SAV file
/PROMPT	Causes the LINKer and LIBRarian to prompt for CSI formatted commands. The LINKer/LIBRarian treat the command strings as continuation lines until a // is seen. /PROMPT is equivalent to // mode of continuation. Use this option to specify overlays, for example: .LINK/PROMPT ROOT *OVR1/O:1 *OVR2/O:1 *OVR3/O:2 *OVR4/O:2// This creates two overlay regions with two segments each.
/ROUND:n	Rounds up a section so that the root is a whole number multiple of n (a power of 2). Prompt: Round section:
/RUN	For background jobs only, executes the resulting .SAV file
/SLOWLY	Allows largest memory area for symbol table
/STACK[:n]	Modifies the stack address (default is loc. 42). Give an octal value (:nnnnnn) or else system prompts for a global symbol: Stack symbol?

Table 1-7 (Cont.) Linker Options Available Under RT-11

Option	Explanation
/SYMBOLTABLE[:filespec]	Creates a file containing symbol definitions for all global symbols. Enter the symbol table file specification as the third output specification in the LINK command.
/TOP:value	Specifies the highest address to be used by the relocatable code. The argument value represents an unsigned, even octal number.
/TRANSFER[:n]	Prompts for a global symbol to be used as the starting address of the program. The user can specify a starting address (represented by n).
/WIDE	Sets the number of columns for the width of the link map to 6. The default width is normally 3 for an 80 column wide listing.
/XM	Enables special .SETTOP features in the XM monitor. This option allows a virtual job to map a scratch region in extended memory with the .SETTOP programmed request. See the RT-11 Programmer's Reference Manual for further information on these special .SETTOP features.

Examples of linker options under RT-11 are:

1. LINK A,B,C	Links A.OBJ, B.OBJ, and C.OBJ on DK: and creates A.SAV on DK:
2. LINK/MAP A	Links A.OBJ and creates A.SAV on DK: and a map on LP:
3. LINK/MAP:RK1:/EXE:RK0: A,B,C	Links A.OBJ, B.OBJ, and C.OBJ. The map A.MAP goes to RK1: and the executable file A.SAV goes to RK0:.
4. LINK/MAP/EXE.F00 B,C,D,E,LIB/LIB	Links B.OBJ, C.OBJ, D.OBJ, E.OBJ, and the library LIB.OBJ to create FOO.SAV on DK: and a map on LP:.

1.4 LIBRARY USAGE

You can create a library of commonly used assembly language and FORTRAN IV functions and subroutines through the system program LIBR, which provides for library creation and modification. The librarian chapter of the *RT-11 System User's Guide* describes the LIBR program in detail.

Include a library file 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 the required routines. 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 IV system library, FORLIB.OBJ, is then searched for any other required routines. /F is not specified if FORLIB has been incorporated into SYSLIB. At this point, any unresolved GLOBALS are resolved through SYSLIB. The entire memory image is output to the file LOAD.SAV.

If the /F option or switch is used, all user-created libraries are searched before the default FORTRAN IV system library FORLIB.OBJ. Consult the linker chapter of the *RT-11 System User's Guide* for a detailed description of multilibrary global resolution.

If the linker fails for lack of symbol table space, use the /S linker option in your next attempt. This could slow the linking process, but it allows the maximum possible symbol table space.

To maintain the integrity of the DEC-distributed FORTRAN IV library, create a user library rather than modifying or adding to the FORTRAN IV library (FORLIB) or to the system library (SYSLIB).

1.4.1 Overlay Usage

Use the overlay feature of the linker to segment the memory image so the entire program is not memory-resident at one time. This allows the execution of a program too large for 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, COMMON, subroutines, function subprograms, and any .PSECT that has the GBL attribute and is referenced from more than one segment. An overlay region is an area of memory allocated for two or more overlay segments, only one of which can be resident at one time. An overlay segment consists of one or more subroutines or function subprograms.

When a call is made at run time to a routine in an overlay segment, the overlay handler verifies that the segment is resident in its overlay region. If the segment is in memory, control passes to the routine. If the segment is not resident, the overlay handler reads the overlay segment from the memory image file into the specified overlay region. This destroys the previous overlay segment in that overlay region. Control then passes to the routine.

Give careful consideration to placing routines when you divide a FORTRAN IV program into a root segment and overlay regions, and subsequently divide each overlay region into overlay segments. 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 /O:n) than the calling routine. 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:

- A FORTRAN IV library routine (for example, ASSIGN or DCOS)
- A FORTRAN IV or assembly language routine contained in the root segment
- A FORTRAN IV or assembly language routine contained in the same overlay segment as the calling routine
- A FORTRAN IV or assembly language routine contained in a segment whose region number is different from that of the calling routine

In an overlay environment, you must place the COMMON blocks so they are resident when you reference them. Blank COMMON is always resident because it is always placed in the root segment. You must place all named COMMON blocks either in the root segment or in the segment whose region number is lowest of all the segments that reference the COMMON block. A named COMMON block cannot be referenced by two different segments in the same region unless the COMMON block appears in a segment of a different region number. The linker automatically places a COMMON block into the root segment if it is referenced by the FORTRAN IV main program or by a subprogram 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 User's Guide*) 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 options are used to specify the overlay structure to LINK. The overlay option is in the form:

```
/O:n
```

where:

n = An octal number specifying the overlay region number

The command continuation option has two forms:

```
/C
//
```

/C at the end of each continuation line allows the user to continue long command strings on the next line of input. // is used at the end of the first line and again at the end of the last line of input.

The first line of the LINK overlay structure command string should contain, as the input list, all object modules to be included in the root segment. This line should be terminated with the /C option. The /O:n option cannot appear in the first line of the command string. If all modules 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 /C. The entire root segment must be specified before any overlays.

Any subsequent lines of the command string should be terminated with the /O:n option specifying an overlay region and/or the /C option. The presence of only the /C option specifies 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 continuation 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 option.

For example, assume FORLIB has been built into SYSLIB and 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 coresident).

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

You could use the following command string:

```
.LINK/PROMPT/EXE:LOAD MAIN+SUB1
*SUB2/O:1/C
*SUB3/O:1/C
*SUB4/O:2/C
*SUB5/C
*SUB6/O:2/C
*SUB7//
```

1.4.2 Extended Memory Overlays for RT-11

You can use LINK to create an overlay structure that uses extended memory for privileged or virtual FORTRAN IV jobs. You will need an XM monitor and a hardware configuration that includes a Memory Management Unit to run a program having overlays in extended memory, but you can link such a program on any RT-11 system.

The extended-memory overlay structure is different from the low-memory overlay structure in that extended-memory overlays can reside concurrently in extended memory. This difference allows for speedier execution because, once a program is read in, it requires fewer I/O transfers with the auxiliary mass-storage volume. In fact, if all program data is resident and the program is loaded, the program might run without an auxiliary mass-storage volume.

Note that you must observe with extended-memory overlays the same restrictions that apply to low-memory overlays, especially those pertaining to return paths.

The following command string illustrates the use of extended-memory overlays to create a privileged FORTRAN IV job instead of the low-memory overlays used in the Section 1.4.1 example.

```
.LINK/PROMPT/EXE:LOAD MAIN+SUB1
*SUB2/V:1/C
*SUB3/V:1/C
*SUB4/V:2/C
*SUB5/C
*SUB6/V:2/C
*SUB7//
```

Refer to the *RT-11 System User's Guide* for more information on low-memory or extended-memory overlays.

1.4.3 Standalone FORTRAN IV

You can develop FORTRAN IV programs under the RT-11 FORTRAN IV system and receive output in an absolute binary format for execution on a satellite machine with minimum peripherals. The satellite machine needs only a minimum of 4K words of memory and only a paper tape reader or a serial line unit for program loading.

You can also use the standalone FORTRAN IV capability to construct Read-Only Memory (ROM) applications programs. See Section 2.5.4 for more details on ROM environments.

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

To generate a standalone program, compile the source program units as usual. At link time, you must specify special options to generate the standalone program. You must include the /L option in the LINK command string to cause an absolute binary format (LDA) output file to be generated. You must also include the /I option to allow a special module to be requested from the FORTRAN IV library. This module is:

```
$SIMRT      FORTRAN IV system simulator
```

Since the library used must reflect the hardware arithmetic options available on the satellite machine, you must be careful to use the proper FORTRAN IV library. Hence, the system default library SY:FORLIB.OBJ, which is used when the /F option is specified to LINK, might not be appropriate. RT-11 users should consult the *FORTRAN IV Installation Guide* for information on the various libraries.

The following RT-11 command sequence generates a file, LOAD.LDA, which can be punched on paper tape and loaded with the Absolute Loader on any PDP-11 computer.

```
.LINK/LDA/INCLUDE/EXE:LOAD MAIN,SUBS
Library search?   $SIMRT
Library search?   <RET>
```

See Appendix E for further information on standalone FORTRAN IV capabilities.

1.4.4 RT-11 FORTRAN IV Programs Run as Virtual Jobs

You can develop FORTRAN IV programs, run as virtual jobs, that can access a full 32K words of address space. (Virtual jobs cannot access the I/O page.) When a FORTRAN IV job becomes a virtual job, the FORTRAN IV OTS library initialization code uses the special features of the .SETTOP programmed request to allocate a full 32K words of address space. The initialization code then places the OTS library work area ion extended memory at the high limit returned by the .SETTOP request. This allocation method differs from that of privileged FORTRAN IV jobs. Even if they use extended memory overlays, privileged jobs allocate all the free space in low memory for the OTS library work area.

To make a FORTRAN IV program a virtual job, you compile the source program units as usual but then at link time specify special options.

You specify the /INCLUDE (/I) option to allow a special module to be requested from the FORTRAN IV library. This module is $QBLK, which is the FORTRAN IV Data Area definition for Queue Elements. Next, you specify the /XM option or the /V option (specified on the first line of input to the link) to allow the special .SETTOP features of the XM monitor to be enabled.

The following command sequence generates a file, VIRFOR.SAV, which can be run as a virtual job.

```
.LINK/EXE:VIRFOR/INCLUDE/XM  MAIN+SUBS
Library search?  $QBLK
Library search?  <RET>
```

For more information on virtual and privileged jobs, see the *RT-11 System User's Guide*, the *RT-11 Software Support Manual*, and the *RT-11 Programmer's Reference Manual*.

Note: **The module $QBLK sets the virtual bit in the Job Status Word through an Asect; therefore, effect all other changes to the Job Status Word at runtime with IPEEK and IPOKE system functions.**

1.4.5 Unformatted Byte I/O

An optional module, UIOBYT.OBJ, is included in the distribution kit. If this module is placed in the FORTRAN IV OTS library, each byte element in an unformatted I/O statement is transferred as a byte rather than as a word. You can use UIOBYT.OBJ to save disk space when you do unformatted byte I/O. Select one of the following procedures, as appropriate for your system.

To add this module to the FORTRAN IV OTS library (FORLIB), type:

```
.R LIBR
*FORLIB[-1]=FORLIB,UIOBYT/U/G
Global?   $ERRS
Global?   $ERRTB
Global?   <RET>
* ^C
.
```

If FORLIB is incorporated in SYSLIB, type:

```
.R LIBR
*SYSLIB[-1]=FORLIB,UIOBYT/U/G
Global?   $ERRS
Global?   $ERRTB
Global?   <RET>
* ^C
```

1.4.6 Smaller Execution-Time Programs

When FORTRAN IV programs are linking, the default error message module included from the FORTRAN IV library contains the ASCII text corresponding to each possible object time error. However, the FORTRAN IV library also incorporates an alternate error message module that contains no text; when this module is running, when an object time error occurs only the error number is printed. The resulting core savings in using the shorter form is approximately 850 words; the corresponding on-disk savings in the size of each .SAV or .REL file is three to four blocks. (The core savings can be particularly significant if you are generating programs to be run in the foreground.) The shorter error message module can be included when linking a FORTRAN IV program as follows:

RT-11	Filespecs
.LINK/INCLUDE	MAIN
Library search?	$SHORT
Library search?	<RET>

1.5 EXECUTION PROCEDURES

Section 1.5.1 describes program execution under the RT-11 operating system.

1.5.1 Execution under RT-11

Use the monitor RUN command to start execution of the memory image file generated by LINK. The command:

```
.RUN dev:filespec
```

causes the file on the device (DEV:) to be loaded into memory and executed. Filespec.sav is the file name specification as described in Section 1.1.1.

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

```
.FORTRAN/LIST  MAIN+SUB,SUBT
.LINK/MAP   MAIN,SUBT
.RUN MAIN
```

This searches SYSLIB when any undefined references are present. (The FORTRAN IV OTS library is assumed to have been incorporated into SYSLIB upon system installation.)

1.6 DEBUGGING A FORTRAN IV PROGRAM

The "debug line" capability of FORTRAN IV language programs is effective in debugging because it allows a FORTRAN IV statement to be conditionally compiled. Here are some suggestions for using that capability.

Try to locate the statement in error by typing out intermediate values and results. Place a "D" in column one of each source line you have added for debugging purposes. These lines will not be compiled unless you specify the debug option, keyboard monitor command option /ONDEBUG for RT-11 in the compiler command string. The program can be recompiled without the /D option after the problem has been corrected. All the debugging statements will be treated as comments.

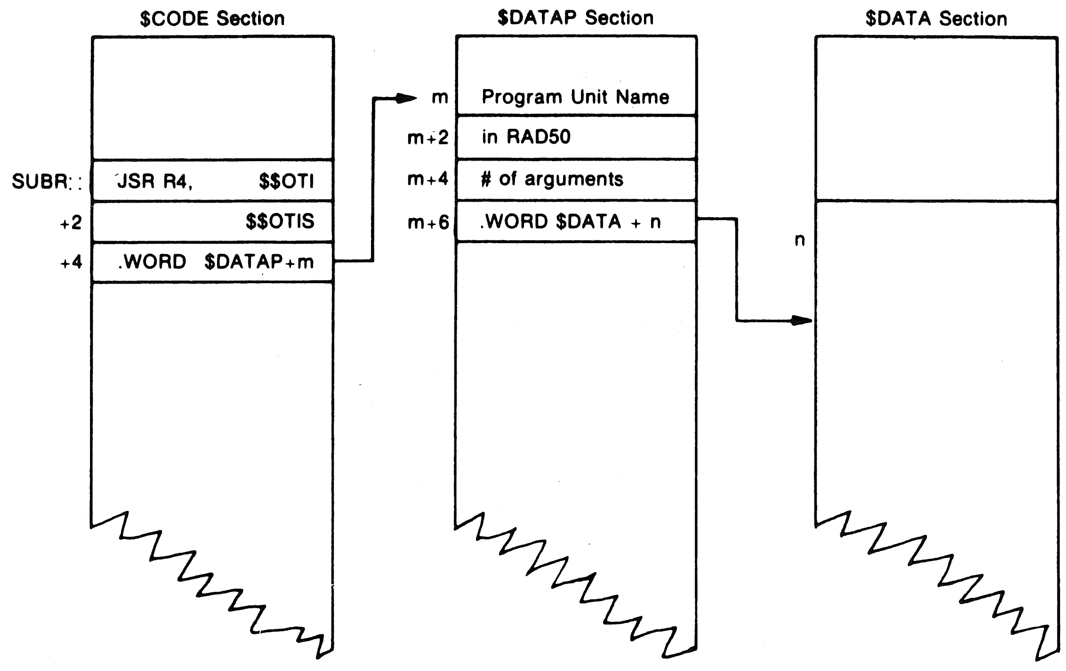
Use the operating system's ODT debugging aid for inline code.

ODT debugging requires the generated code listing for the program (see Section 1.2.2, /L, or /SHOW listing control option). The resulting numbers printed in the left margin of the listing are the octal offsets of the listed machine instructions within the local PSECT $CODE.

Note that only one base address is listed for $CODE in the LINK map when multiple program units are present. To find the base of $CODE for a specific subprogram unit, locate the address of the entry point to the unit.

The variables and data items referenced symbolically in the generated code listing are located in the PSECT $DATA at the offsets indicated by the storage map section of the compiler. All local $DATA sections are formed into a single allocation on the link map when multiple program units are linked to form a single executable program. To find the base address section for a particular program unit, examine the word at offset 4 in the unit's $CODE section. This value will be the address of the base of the unit's pure data section ($DATAP). Examining the word at offset 6 from the $DATAP section will give the base address of the associated $DATA impure section. See Figure 1-2.

Figure 1-2 Finding the Base Address Section



Note: Only one of the following will be found in any program or subprogram unit.

```
MAIN:: JSR  R4,  $OTI   - threaded main program
SUBR:: JSR  R4,  $OTIS  - threaded SUBR or FUNCT
MAIN:: JSR  R4,  $$OTI  - inline main program
SUBR:: JSR  R4,  $$OTIS - inline SUBR or FUNCT
```

2 FORTRAN IV OPERATING ENVIRONMENT

2.1 FORTRAN IV OBJECT TIME SYSTEM

The PDP-11 FORTRAN IV object time system (OTS) library provides the user with a variety of common sequences of PDP-11 machine instructions invoked by compiled FORTRAN IV programs. The OTS library consists of many small functional modules from which the compiler selects only those required to implement the FORTRAN IV program. The required sequences are integrated with the compiler-generated code during linkage to form the executable program. For example, if the user program performs only sequential access, formatted I/O, none of the direct access I/O routines is included.

The FORTRAN IV OTS library is comprised of the following:

- Mathematics routines, including the FORTRAN IV library functions and other arithmetic routines (for example, floating-point routines)
- Miscellaneous utility routines (for example, RANDU, DATE, SETERR)
- Routines that handle various types of FORTRAN IV I/O
- Error-handling routines that process arithmetic errors, I/O errors, and system errors
- Miscellaneous routines required by the compiled code

2.1.1 OTS Library File

Although it is possible to integrate the OTS library into the RT-11 system library file SYSLIB.OBJ, it is recommended that the OTS library be stored as a separate library file, usually called FORLIB.OBJ. When you are linking FORTRAN IV programs, use of the /F linker option will automatically refer the linker to this file.

2.2 OBJECT CODE

The FORTRAN IV compiler translates programs written in the symbolic PDP-11 FORTRAN IV language into "object code" (machine language). The resulting object modules, combined with required modules from the FORTRAN IV OTS library, form executable programs of PDP-11 machine instructions.

The compiler produces two distinctly different types of object programs by generating either threaded code or inline code. When you select inline code (through the options /CODE: [/I:] EAE, EIS, or FIS), the compiler produces the one-to-one PDP-11 machine instructions required for the specific arithmetic hardware in the system configuration. Symbolic FORTRAN IV library routines are referenced to perform only those functions that cannot be achieved in short sequences of machine instructions. A program compiled through an inline code option produces an object program that conforms specifically to the type of hardware selected at compilation time.

When you select threaded code (through the /CODE:THR [/I:THR] option), the object program produced uses a symbolic library routine to perform each operation required for program execution; the executable program consists of a "threaded" list of the addresses of library routines and appropriate operand addresses. This type of code generation produces an object module that operates independently of hardware arithmetic configuration. It can be combined with any of the FORTRAN IV OTS libraries to produce a valid executable program for each type of arithmetic hardware without any need for recompilation.

Consider the following when you decide whether to use inline or threaded code.

1 When the program does not contain REAL*4, REAL*8, or COMPLEX*8 arithmetic operations:
   - Inline code always executes faster than threaded
   - The differences in size between inline and threaded programs are slight

2 When the program contains large amounts of REAL*4, REAL*8, and COMPLEX*8 arithmetic (scientific computation):
   - Threaded code is much smaller than inline code
   - Execution speed is nearly the same for both.

Note: Although the above relationships are generally true, they do vary from program to program. Therefore, DIGITAL recommends that you compile and test production programs with both inline and threaded code. This procedure will allow you to determine the best type of code in terms of size and speed for your application.

Table 2-1 gives a comparison of threaded and inline code.

Table 2-1 Threaded/Inline Code Comparison for Statement I=J*K+REAL

	Inline Code		
Threaded Code	**For FIS**	**For EIS**	**For EAE**
MOI$MS J	MOV J,R1	MOV J,R1	MOV #$EAE,R5[1]
MUI$MS K	MUL K,R1	MUL K,R1	MOV J,(R5)+ MOV K,@R5
	MOV R1,-(SP)	MOV R1,-(SP)	MOV -(R5),-(SP)
CFI$	JSR PC,$CVTIF	JSR PC,$CVTIF	JSR PC,$CVTIF
ADF$MS REAL[2]	MOV REAL+2,-(SP) MOV REAL,-(SP) FADD SP	MOV REAL+2,-(SP) MOV REAL,-(SP) JSR PC,$ADDF	MOV REAL+2,-(SP) MOV REAL,-(SP) JSR PC,$ADDF
CIF$	JSR PC,$CVTFI	JSR PC,$CVTFI	JSR PC,$CVTFI
MOI$SM I	MOV (SP)+,I	MOV (SP)+,I	MOV (SP)+,I

[1]$EAE represents the address of the KE11-A (or -B) accumulator register (AC).

[2]Note that the threaded code sequence for this floating-point addition requires only two words of memory plus the size of the ADF$MS routine, whereas the inline code uses five words (FIS), four words (EIS), or six words (EAE). This demonstrates the savings in storage in using threaded code for floating-point operations.

2.2.1 Processor-Defined Functions

The compiler generates inline code for the following processor-defined functions (PDFs):

Function	Definition
IABS(I)	Integer absolute value
IDIM(I,J)	Integer positive difference
ISIGN(I,J)	Integer transfer of sign
MOD(I,J)	Integer remainder
MIN0(I,J)	Integer minimum of integer list
MAX0(I,J)	Integer maximum from integer list
IFIX(A)	Real to integer conversion
FLOAT(I)	Integer to real conversion
REAL(C)	Complex to real conversion, obtain real part
DBLE(A)	Real to double conversion
SNGL(A)	Double to real conversion

Since the code for a PDF is generated by the compiler, no global reference to the function name is produced. A problem could arise when the function call is to be interpreted as a call to a user-written routine. To force the compiler to treat the apparent PDF call as a reference to a user routine, specify the routine as external to the program with an EXTERNAL statement.

For example, when the statement I = IABS(J) is compiling, code equivalent to the following is produced:

```
        MOV    J,I
        BPL    1$
        NEG    I
 1$:    ...
```

If the statement EXTERNAL IABS is included, code equivalent to the following will be produced:

```
        .GLOBAL  IABS
        MOV      #J,-(SP)
        MOV      #1,-(SP)
        MOV      SP,R5
        JSR      PC,IABS
        CMP      (SP)+,(SP)+
        MOV      R0,I
```

2.2.2 Virtual Array Options (RT-11 Only)

The VIRTUAL statement declares arrays that are assigned space outside the program's directly addressable memory and are manipulated through the virtual array facility of FORTRAN IV. The VIRTUAL statement allows arrays to be stored in large data areas that are accessed at high speed.

For more detailed information on virtual arrays, see the *PDP-11 FORTRAN IV Language Reference Manual.*

Virtual arrays are limited only by the number of elements, not by the total storage available. Under the RT-11 operating system, all memory above the initial 28K words is available for virtual array storage. Except for the size of physical memory, there is no limit to the number or to the total size of all virtual arrays a program can access. The maximum number of elements in a virtual array is 32767. Thus, the largest LOGICAL*1 virtual array is 16K words, or 32767 bytes. The largest REAL*8 virtual array is 128K words, or 262136 bytes. This is a total of 32767 elements, each of which occupies 8 bytes. The limit is 32767 elements because FORTRAN IV requires array subscripts to be positive integers.

Virtual array support for RT-11 uses the Program Logical Address Space (PLAS) extensions when it operates under the XM monitor. The FORTRAN IV OTS library directly manipulates the KT-11 mapping registers to provide virtual array support when operating under RT-11 single job and FB monitors.

All virtual arrays declared in a program unit are allocated in a .VSECT, a type of program section. The compiler concurrently generates a .VSECT additive type of relocation as it references the virtual array in the object program.

The .VSECT program section is unnamed and has the "concatenate" attribute. This allows the accumulating of all virtual storage requirements for a linked job or task as the concatenation of the .VSECTS.

The syntax of the VIRTUAL statement is identical to that of the DIMENSION statement and involves only substituting the keyword VIRTUAL for the keyword DIMENSION. However, there is a significant semantic difference between the two because of the limitations imposed on the DIMENSION statement. Local arrays declared by the DIMENSION statement are limited by the maximum memory available to the program. Section 2.2.6 demonstrates how to convert an existing program to use the

VIRTUAL feature. The three virtual array options available in building the RT-11 OTS library are:

NOVIR.OBJ
VIRP.OBJ
VIRNP.OBJ

2.2.3 NOVIR.OBJ

This module specifies no virtual array support and is intended for the user who prefers to optimize the size of FORLIB rather than use virtual array support. When NOVIR.OBJ is included in the library, all references to the virtual array routines (made by the compiler when virtual arrays are available) produce ERROR 64, Virtual Array Initialization Failure.

2.2.4 VIRP.OBJ

This module is for PLAS support and requires both the XM monitor and EIS (or FIS or FPU) hardware for program execution. Failure to adhere to these requirements will result in runtime ERROR 64, Virtual Array Initialization Failure. VIRP uses the 4K words of virtual memory addresses starting at 160000 octal (normally the PDP-11 I/O page) for a window. For this reason, programs using VIRP support cannot reference the I/O page.

The program initialization code uses the PLAS .ALLOC directive to allocate a region of the required size from the extended memory pool. This region is a contiguous section of physical memory large enough to include all virtual arrays declared in the executable program.

Note: **If FORTRAN IV is unable to allocate a region of the proper size, a FATAL FORTRAN IV error message results.**

A window of 4K words initially maps the first 4K words of the virtual array region. When a virtual array element lies outside the window, the PLAS .REMAP directive causes a window-turn operation to allow access.

2.2.5 VIRNP.OBJ

This module provides virtual array support for the single job (SJ) and the foreground/background (FB) monitors. VIRNP supports full 11/70 22-bit addressing capability. The largest amount of virtual memory available is 2020K words (2048K minus 28K). VIRNP supports the KT11 Memory Management Unit directly, mapping the job and RT-11 to kernel space and the virtual array to user space. The module turns on the KT11 immediately before a virtual array fetch/store, and off immediately after, thus keeping KT11 mapping overhead to a minimum.

When VIRNP support is used under the FB monitor, the foreground and the background jobs cannot use virtual arrays concurrently, because both will reference the same region of extended memory for array storage. The XM monitor and PLAS virtual support must be used when two concurrent jobs require access to virtual arrays.

2.2.6 Converting a Program to Use Virtual Arrays

First, be sure to observe the usage restrictions for virtual arrays covered in the *PDP-11 FORTRAN IV Language Reference Manual.* To convert the existing program, declare the arrays by using the VIRTUAL instead of the DIMENSION statement. The program does not require additional access coding.

The following example illustrates a general, minimum effort program conversion.

1. Identify the non-virtual arrays to be converted to virtual arrays.
2. Locate the DIMENSION and the type declaration statements in which these arrays are declared. Replace DIMENSION statements with equivalent VIRTUAL statements. Replace array-declarative type declaration statements with VIRTUAL statements to define the array dimensions, and remove the dimensioning information from the type declaration statements.
3. Compile the program. Observe all compilation errors; these will occur where the syntax restrictions outlined in the *PDP-11 FORTRAN IV Language Reference Manual* have been violated. In some cases, you might have to reformulate the data structures to use virtual arrays effectively.
4. Check the code to ensure that virtual array parameters are passed correctly to subprograms.

   - If the argument list of a subprogram call includes an unsubscripted virtual array name, the argument list of the SUBROUTINE or FUNCTION statement must have an unsubscripted virtual array name in its corresponding dummy argument. This establishes access to the virtual array for the subprogram. The declaration of the virtual array in the subprogram must be dimensionally compatible with the virtual declaration in the calling program. All changes to the virtual array that occurred during subprogram execution are retained when control returns to the calling program.

     When you pass entire arrays as subprogram parameters, be certain that the matching arguments are defined as both virtual or both non-virtual. Mismatches of array types are not detectable at either compilation or execution time, and the results are undefined.

   - If the argument list of a subprogram reference includes a reference to a virtual array element, the matching formal parameter in the SUBROUTINE or FUNCTION statement must be a non-virtual variable. Value assignments to the formal parameter occurring within the subprogram do not alter the stored value of the virtual array element, the calling program. To alter the value of that element, the calling program must include a separate assignment statement that references the virtual array element directly.

The following example demonstrates the process of changing non-virtual arrays to virtual arrays.

```
      DIMENSION A(1000,20)
      INTEGER*2 B(1000)
      DATA D/1000*0/
      CALL ABC(A,B,1000,20)
      WRITE(2,*)(A(I,1),I=1,1000
      END

      SUBROUTINE ABC(X,Y,N,M)
      DIMENSION X(N,M)
      INTEGER*2 Y(N)
      DO 10, I=1,N
10    X(I,1)=Y(I)
      RETURN
      END
```

This program contains arrays A and B.

Array A is declared in a DIMENSION statement and is of the default data type. Thus, substituting the keyword VIRTUAL for the keyword DIMENSION is sufficient for its conversion.

Note, however, that array B and its dimensions are declared in a type declaration statement (in the second line of the program).

To convert B into a virtual array, its declarator must be moved to a VIRTUAL statement; also, the variable B must remain in the type declaration statement, but without a dimension specification.

A and B are both passed to subroutine ABC as arrays, rather than array elements. Thus, the associated subroutine parameters must also be converted to virtual arrays.

The following compiled listing illustrates the program after the first phase of the conversion.

```
FORTRAN IV      V02.8      Thu 25-Sep-86  00:41:38

0001        VIRTUAL A(1000,20), B(1000)
0002        INTEGER*2 B
0003        DATA B/1000*0/
0004        CALL ABC(A,B,1000,20)
0005        WRITE(2,*)(A(I,1),I=1,1000)
0006        END

FORTRAN IV      Diagnostics for Program Unit .MAIN.

In line 0003, Error:  Usage of variable "B" invalid


FORTRAN IV      Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000006  (   3. words)

Name   Type  Offset     Name   Size  Offset      Name   Size  Offset
I      I*2   000000

VIRTUAL Arrays, Total Size = 00240200 (  41024. words)

Name     Type    Offset    -------Size------- Dimensions
A      R*4 Vec  00000000  00234200 (  40000.) (1000,20)
B      I*2      00234200  00003720 (   1000.) (1000)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name   Type   Name   Type   Name   Type   Name   Type   Name   Type
ABC     R*4

FORTRAN IV      V02.8      Thu 25-Sep-86 00:41:40          Page 001
```

```
0001          SUBROUTINE ABC(X,Y,M,N)
0002          VIRTUAL Y(N), X(N,M)
0003          INTEGER*2 Y
0004          DO 10, I=1,N
0005   10     X(I,1)=Y(I)
0006          RETURN
0007          END

FORTRAN IV        Storage Map for Program Unit ABC

Local Variables, .PSECT $DATA, Size = 000012 (      5.words)

Name    Type  Offset      Name    Type  Offset       Name    Type  Offset
I       I*2   000010      M       I*2 @ 000004       N       I*2 @ 000006

VIRTUAL Arrays, Total Size = 00000000  (       0. words)

Name      Type      Offset    -------Size-------- Dimensions
X       R*4    @     000000     ****    (  ****  ) (N,M)
Y       I*2    @     000002     ****    (  ****  ) (N)
```

Note that the main program compilation causes an error message. Data statements must not refer to virtual arrays. The user substitutes a DO loop to achieve the same result.

The following listing shows the program after the conversion is completed.

```
FORTRAN IV        V02.8        Thu 25-Sep-86 00:41:25           Page 001

0001          VIRTUAL A(1000,20), B(1000)
0002          INTEGER*2 B
0003          DO 5, I=1,1000
0004   5      B(I)=0
0005          CALL ABC(A,B,1000,20)
0006          WRITE(2,*)(A(I,1),1000)
0007          END

FORTRAN IV        Storage Map for Program Unit .MAIN.

Local Variables, .PSECT $DATA, Size = 000006 (    3. words)

Name    Type  Offset      Name   Type  Offset       Name   Type  Offset
I       I*2   000000

VIRTUAL Arrays, Total Size = 00240200 (  41024. words)

Name      Type     Offset   -------Size-------- Dimensions
A       R*4   Vec 00000000 00234200 (  40000.) (1000,20)
B       I*2       00234200 00003720 (   1000.) (1000)

Subroutines, Functions, Statement and Processor-Defined Functions:

Name    Type    Name    Type    Name    Type    Name    Type    Name    Type
ABC      R*4

FORTRAN IV       V02.8       Thu 25-Sep-86 00:41:27             Page 001

0001          SUBROUTINE ABC(X,Y,M,N)
0002          VIRTUAL Y(N), X(N,M)
0003          INTEGER*2 Y
0004          DO 10, I=1,N
0005   10     X(I,1)=Y(I)
0006          RETURN
0007          END

FORTRAN IV        Storage Map for Program Unit ABC

Local Variables, .PSECT $DATA, Size = 000012  (      5.words)

Name    Type  Offset      Name    Type  Offset       Name    Type  Offset
I       I*2   000010      M       I*2 @ 000004       N       I*2 @ 000006

VIRTUAL Arrays, Total Size = 00000000 (        0. words)
```

```
Name      Type    Offset     -------Size-------- Dimensions
X       R*4    @  000000     ****   (  **** )  (N,M)
Y       I*2    @  000002     ****   (  **** )  (N)

PDS> COPY DKO:BADVRT.LST/RT TI:
```

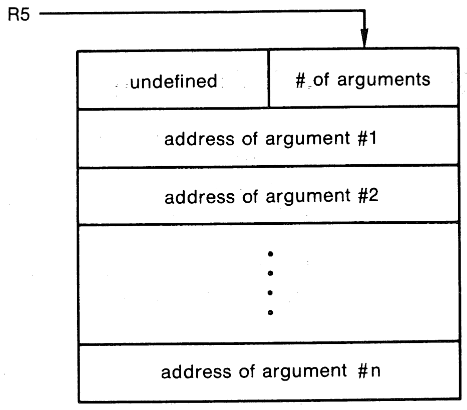
2.3 SUBPROGRAM LINKAGE

Subprogram linkage operates identically for all subprograms, including those written by the user in FORTRAN IV and in assembly language.

The following instruction is used to pass control to the subprogram:

```
JSR     PC, routine name
```

Register 5 (R5), prior to calling the subprogram, is set to the address of an argument list having the following format:



ZK-1197-82

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, for example, CALL SUB (A,,B).

Note: **Be certain that the called subprogram does not modify the argument list passed to it by the calling program.**

The instruction

```
RTS     PC
```

causes control to return to the calling program.

The following is an example of argument transmission: an assembly language subroutine is written to sum all integer arguments it finds in each parameter list, and to return the result to the FORTRAN IV program as the value of a final, additional argument. The FORTRAN IV CALL statements that invoke this routine take the form:

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

where num1 through numn represent a variable number of integer quantities to be summed, and isum represents the variable or array element in 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.

2.3.1 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 so 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 in Table 2-2.

Table 2-2 Return Value Convention for Function Subprograms

Type	Result in
INTEGER*2 LOGICAL*1	R0
INTEGER*4 LOGICAL*4	R0—low-order result R1—high-order result
REAL	R0—high-order result R1—low-order result
DOUBLE PRECISION	R0—highest-order result R1- R2— R3—lowest-order result
COMPLEX	R0—high-order real result R1—low-order real result R2—high-order imaginary result R3—low-order imaginary result

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

- Short floating mode (SETF)

- Short integer mode (SETI)
- Floating truncate mode

Should the assembly language routine 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)+
```

2.4 VECTORED ARRAYS

Array vectoring decreases the time necessary to reference elements of a multidimensional array by using additional memory to store the array.

Since multidimensional arrays are stored sequentially in memory, certain address calculations determine the location of individual elements. Typically, a mapping function performs this calculation. For example, to locate the element LIST(1,2,3) in an array dimensioned LIST(4,5,6), use a function equivalent to the following equation. 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

Such a mapping function requires multiplication operation(s), and some PDP-11 hardware configurations do not have the MUL instruction. The compiler can reduce execution time at the expense of memory storage by "vectoring" some arrays.

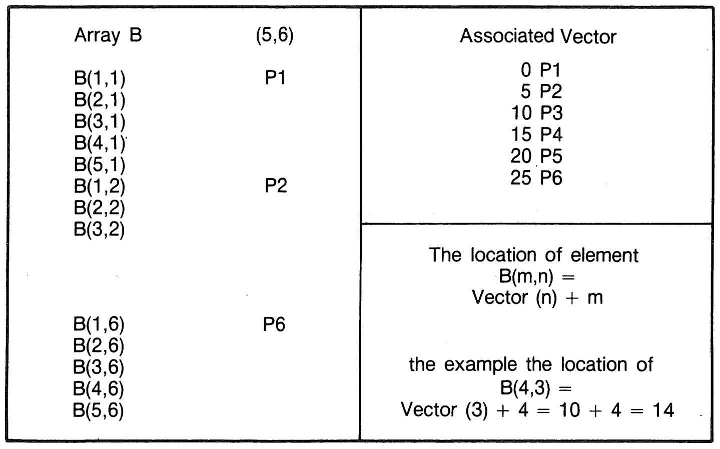
Since array vectors map only the declared dimensions of the array, you must ensure that references to arrays are within their declared bounds. A reference outside the declared bounds of a vectored array causes unpredictable results (for example a program interrupt). You must give particular attention 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 IV Language Reference Manual* contains more information on dimensions.

A specific element in a vectored array can be located by a simplified mapping function, without the need for multiplication. Instead, a table lookup determines the location. For example, a vectored two-dimensional array B(5,6) 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 depicts the array vectoring process.

Figure 2-1 Array Vectoring



The compiler bases the decision to vector a multidimensional array on the computed ratio of the space required to vector the array to the total storage space it requires. The array is not vectored if this ratio is greater than 25 percent. 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 option /NOVECTORS (/V) suppresses all array vectoring.

The amount of memory required to vector an array is 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 good programming practice to place an array's largest dimension first when it will be vectored.

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

- Arrays are in the same program unit.
- 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, or which the array A uses only the first 10 elements.

2.5 PROGRAM SECTIONS

Program sections (PSECTS) contain code and data and are identified by unique names as segments of the object program. The attributes associated with each PSECT direct the Link utility to combine several separately compiled FORTRAN IV program units, assembly language modules, and library routines into an executable program. The following attributes are associated with these sections:

Concatenate	(CON)	or	Overlay	(OVR)
Data	(D)	or	Instruction	(I)
Global	(GBL)	or	Local	(LCL)
Relocatable	(REL)	or	Absolute	(ABS)
Read/Write	(RW)	or	Read-Only	(RO)

2.5.1 Compiled Code PSECT Usage

The compiler organizes compiled output into three program sections that have the names, attributes, and contents shown in Table 2-3.

Table 2-3 Compiler Organization of Program Sections

Section Name	Attributes	Contents
$CODE	RW,[1] I, LCL, REL, CON	All executable code, including threaded and inline, for a program unit
$DATAP	RW,[1] D, LCL, REL, CON	Pure data (for example, constants, FORMATs, array vectors) that cannot change during program execution
$DATA	RW, D, LCL, REL, CON	Impure data, variables, temporary storage, and arrays used in the FORTRAN IV program

[1]The RO/RW attribute for sections $CODE and $DATAP is controlled by the compiler /Z command option. See Section 1.2.2.

When named PSECTS with the CON attribute are concatenated by the LINK utility, all PSECTS with the same name are allocated together beginning at the same address. The length of the resulting PSECT is the sum of the individual sections so defined.

2.5.2 Common Block PSECT Usage

FORTRAN IV COMMON storage is placed in named PSECTs. The PSECT name is identical to the COMMON block name specified in the FORTRAN IV program. PSECTs used for COMMON storage are given the attributes RW, D, GBL, REL, and OVR.

For example, the statement:

```
COMMON /X/ A,B,C
```

produces the equivalent of the following MACRO-11 code:

```
        .PSECT  X, RW, D, GBL, REL, OVR
A:      .BLKW   2
B:      .BLKW   2
C:      .BLKW   2
```

FORTRAN IV blank COMMON uses the section name .$$$$.; thus the statement:

COMMON C,B /X/ A

produces the equivalent of:

```
        .PSECT  .$$$$., RW, D, GBL, REL, OVR
C:      .BLKW   2
B:      .BLKW   2
        .PSECT  X, RW, D, GBL, REL, OVR
A:      .BLKW   2
```

When named PSECTs with the OVR attribute are combined by the LINK utility, all PSECTs with the same name are allocated together beginning at the same address. The resulting PSECT has a length that is the maximum of the individual sections so combined.

2.5.3 OTS Library PSECT Usage

Modules included in the OTS library and referenced by the compiled program are segmented into five program sections, as shown in Table 2-4.

Table 2-4 Organization of OTS Library Modules

Section Name	Attributes	Contents
OTS$I	RW, I, LCL, REL, CON	All pure code and data for the module
OTS$P	RW, D, GBL, REL, OVR	Pure tables of addresses of other OTS library modules
OTS$D	RW, D, LCL, REL, CON	Pure data referenced by the module
OTS$S	RW, D, LCL, REL, CON	Scratch storage referenced by the module
OTS$O	RW, I, LCL, REL, CON	OTS library routines sensitive to USR swapping

2.5.4 Ordering of PSECTs in Executable Programs

The order in which program sections are allocated in the executable program is controlled by the order in which they are first presented to the LINK utility. >x>(Executable programs>PSECT order)

Applications that are sensitive to this ordering typically separate sections containing read-only information (such as executable code and pure data) from impure sections containing variables.

The main program unit of a FORTRAN IV program (normally the first object program in sequence presented to LINK) declares the following PSECT ordering:

Section Name	Attributes
OTS$I	RW, I, LCL, REL, CON
OTS$P	RW, D, GBL, REL, OVR
SYS$I	RW, I, LCL, REL, CON
USER$I	RW, I, LCL, REL, CON
$CODE	RW, I, LCL, REL, CON
OTS$O	RW, I, LCL, REL, CON
SYS$O	RW, I, LCL, REL, CON
$DATAP	RW, D, LCL, REL, CON
OTS$D	RW, D, LCL, REL, CON
OTS$S	RW, D, LCL, REL, CON
SYS$S	RW, D, LCL, REL, CON
$DATA	RW, D, LCL, REL, CON
USER$D	RW, D, LCL, REL, CON
.$$$$.	RW, D, GBL, REL, OVR
Other COMMON Blocks	RW, D, GBL, REL, OVR

In the RT-11 environment, the User Service Routine (USR) can swap over pure code, but must not be loaded over constants or impure data that can be passed as arguments to it.

The above ordering collects all pure sections before impure data in memory and USR can safely swap over sections $CODE, OTS$I, OTSP, SYSI, and USER$I.

Assembly language routines used in applications sensitive to PSECT ordering should use the same program sections as output by the compiler for this purpose, that is, place pure code and read-only data in section USER$I and all impure storage in section USER$D. This will ensure that the assembly language routines will participate in the separation of code and data.

Note that the ordering of PSECTs in an overlay program will follow the guidelines herein for each overlay segment (for example, the root segment will contain pure sections followed by impure sections, and each overlay segment will have a similar separation of pure and impure internal to its structure).

In overlay environments, PSECTs with the GBL attribute will be allocated in the root segment if they are referenced by more than one overlay segment in the same region.

To build an application based on read-only memory (ROM), include sections $CODE, OTS$I, OTSP, SYSI, USER$I, $DATAP, OTS$O, and OTS$D in the read-only memory. Use the "round section" capability of the LINK utility to increase the size of the section OTS$D, to round the base address of section OTS$S up to the first available read-write memory location following the ROM.

ROM-based applications created in this manner must obey these programming restrictions:

- Variables or arrays cannot be initialized with a DATA statement. Use assignment statements instead.
- Formatted READ statements cannot transfer ASCII data into Hollerith fields in the FORMAT statement itself. Instead, place the data in a variable or array.

The PSECT ordering specified by the compiler might not be suitable for some applications. The user can define ordering by creating a MACRO-11 source file that contains only PSECT declarations. The order in which the declarations appear in this file will be the order in which the sections will be allocated in the executable program. Be sure that the attributes specified in this file agree with those assigned by the compiler.

The MACRO-11 source file containing the desired ordering must be assembled and used as the first input file in the LINK command.

The above technique allows the use of data-initialized COMMON blocks in ROM applications (if the program does not attempt to modify the values in these COMMON blocks, which will be placed in read-only memory). The new ordering defined by the MACRO-11 source file should then be:

```
.PSECT   OTS$I        RW, I, LCL, REL, CON
.PSECT   OTS$P        RW, D, GBL, REL, OVR
.PSECT   SYS$I        RW, I, LCL, REL, CON
.PSECT   USER$I       RW, I, LCL, REL, CON
.PSECT   $CODE        RW, I, LCL, REL, CON
.PSECT   OTS$O        RW, I, LCL, REL, CON
.PSECT   SYS$O        RW, I, LCL, REL, CON
.PSECT   $DATAP       RW, D, LCL, REL, CON
.PSECT   com1         RW, D, GBL, REL, OVR
.PSECT   com2         RW, D, GBL, REL, OVR
  .        .            .
  .        .            .
  .        .            .
  .        .            .
.PSECT   comn         RW, D, GBL, REL, OVR
.PSECT   OTS$D        RW, D, LCL, REL, CON
.PSECT   OTS$S        RW, D, LCL, REL, CON
.PSECT   SYS$S        RW, D, LCL, REL, CON
.PSECT   $DATA        RW, D, LCL, REL, CON
.PSECT   USER$D       RW, D, LCL, REL, CON
.PSECT   .$$$$.       RW, D, GBL, REL, OVR
```

where com$_1$, com$_2$, ... com$_n$ represent the names of the read-only COMMON blocks that will contain pure data.

For further information on ROM applications, see the *PROM/RT-11 User's Guide*.

2.6 TRACEBACK FEATURE

The traceback feature included in RT-11 FORTRAN IV fatal runtime error messages 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 (see the example following). 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      A=1.0
0003      F=A/D
0004      RETURN
0005      END

          Traceback 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
```

Note in the example above that the line number in the traceback of routine 'SUB2' is simply a question mark. This is because the module was compiled with the /NOLINENUMBERS option (/S) in effect (see Section 1.2.2).

2.7 RUNTIME MEMORY ORGANIZATION FOR RT-11

The runtime memory organization in both a user service routine (USR) swapping system and a USR resident system operates as shown in Figure 2-2. When you link user-written interrupt-handling routines with a FORTRAN IV program, avoid USR swapping over the interrupt routines and any associated data. The USR is swapped in above the interrupt vectors at location $$OTSI and extends about 2K words from that point (see Figure 2-2). Interrupt routines must therefore be loaded above the area used for USR swapping when the USR will be actively swapped. Unless explicitly disabled by the compiler option /NOSWAP (/U), the USR is actively swapped.

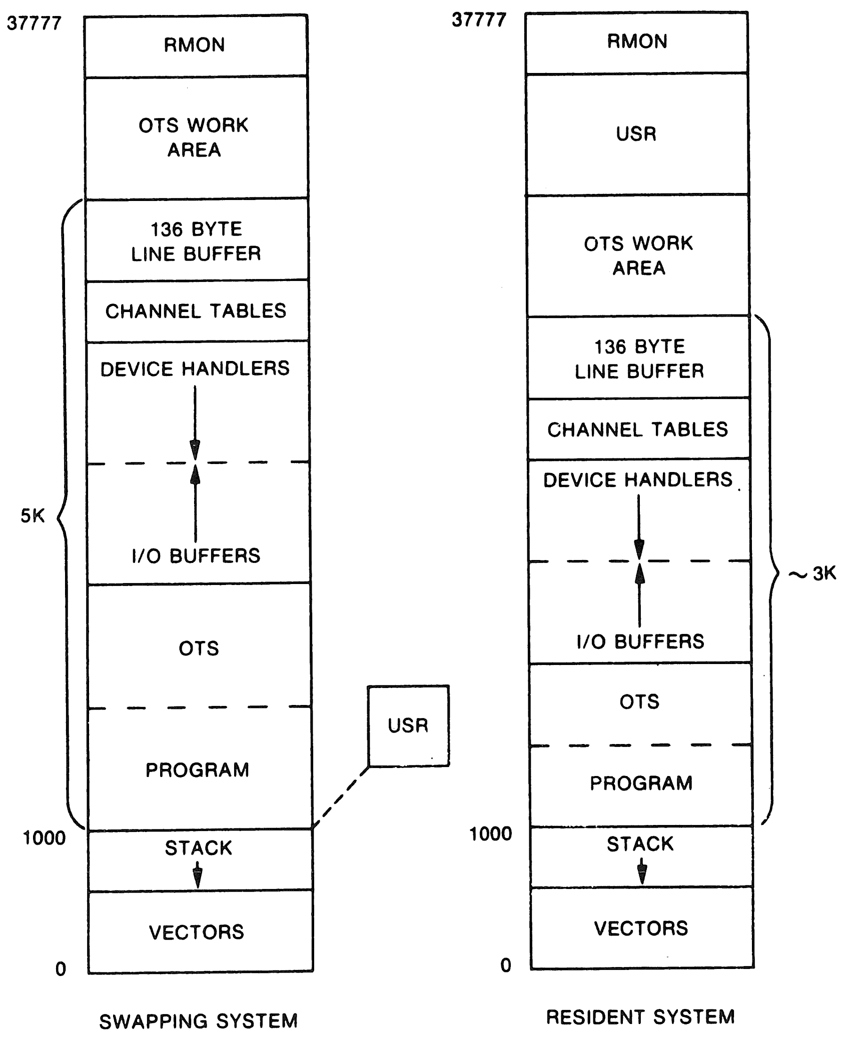
Some runtime memory segments are fixed in size. These include the resident monitor, the OTS library work area, the stack and interrupt vector areas, and, in the USR resident system, the USR area.

Other runtime memory segments vary in length in accordance with the length of the user program. The device handlers, channel tables, and I/O buffers are allocated space dynamically. Only those handlers needed for currently active devices are resident. I/O buffers are allocated and deallocated as required.

There is a limit, however, to the amount of space that can be allocated to the varying-length memory segments. In an 8K swapping system, approximately 5K words are available; in an 8K resident system, the total is approximately 3K words.

If a large FORTRAN IV program cannot be run in the amount of memory available, reduce the size of a runtime memory segment to allow successful program execution. Use overlay capabilities (see Section 1.4.1) to reduce the amount of memory needed for the user program. Minimize the number of different physical devices used for I/O to reduce the number of handlers that must be resident. When the program finishes I/O to a file, close it by using the CALL CLOSE routine (see Section B.4) or the CLOSE statement, thereby reallocating the buffer space to any new file to be opened.

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



3 FORTRAN IV SPECIFIC CHARACTERISTICS

This chapter applies specifically to the FORTRAN IV language as it operates under the RT-11 system. The material presented here both relaxes the restrictions imposed by the *PDP-11 FORTRAN IV Language Reference Manual* and provides additional material essential to RT-11 users but not covered in the manual.

Note that these deviations from the FORTRAN IV syntax requirements of the manual apply only to RT-11. Your resulting program cannot be transported freely to another operating system without careful planning for that system's peculiarities and language requirements. FORTRAN IV relaxes the strict statement ordering specifications of the manual and makes only the following three statement ordering requirements:

1. The first noncomment line in a subprogram 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.

If you do not follow the statement ordering requirements of the manual, you can have a warning diagnostic included with the source listing by using the /WARNINGS option (/W).

3.1 OPEN/CLOSE STATEMENT RESTRICTIONS

Although the OPEN and CLOSE statements have an optional "ERR=label" error exit, only the following OPEN/CLOSE error conditions will cause control to be passed to "label":

- Syntax error in filespec for NAME=
- File not found NAME=
- OPEN (UNIT=n, ...) when a file is already open on unit n
- CLOSE (UNIT=n, ...) when no file is open on unit n

See Section 3.10 for READ, WRITE, ENCODE, and DECODE errors.

3.1.1 Keyword Constraints

Valid keywords in the FORTRAN IV OPEN/CLOSE statements are the same for RT-11 as those described in the *PDP-11 FORTRAN IV Language Reference Manual*. Since FORTRAN IV does not support some keywords and options under RT-11, however, certain constraints must be recognized in constructing OPEN/CLOSE statements. These constraints are given in Table 3-1.

Table 3-1 Keyword Constraints under RT-11

Keyword	RT-11 Constraint
ACCESS='APPEND'	Not supported—results in error at compile time and at run time
BUFFERCOUNT=bc	Specifies the number of buffers (one or two) to be used when outputting data on the logical unit—if not specified, one buffer is allocated
BLOCKSIZE=bs	Not supported—results in error at compile time and run time
CARRIAGE CONTROL	Formatted files must have the attribute 'FORTRAN' or 'LIST'—unformatted or random access files must have the attribute 'NONE'
DISPOSE='PRINT'	Not supported—results in error at compile time
EXTENDSIZE=es	Not supported—results in error at compile time and at run time
FORM='FORMATTED'	Not supported on direct access files
INITIALSIZE=is	Default initial allocation is the larger of two areas: the second largest empty area or one half of the largest area. INITIALSIZE controls the entire allocation for the file since extension is not supported.
MAXREC=mr	The size of the initial allocation made for a direct access file is the larger of two parameters: INITIALSIZE or the product RECORDSIZE and MAXREC
NOSPANBLOCKS	Not supported—error
RECORDSIZE	Must be specified for direct access files
SHARED	Not supported—results in error at compile time and at run time

3.2 SOURCE LINES

A valid 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 nonalphabetic character to signal continuation,

or

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

3 A valid FORTRAN IV 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 a FORTRAN IV statement.

3.3 VARIABLE NAMES

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 occurs for each variable name that exceeds six characters in length. Warning diagnostics will appear in a compilation listing if the /WARNINGS option (/W) is included in the compiler command string.

3.4 INITIALIZATION OF COMMON VARIABLES

FORTRAN IV allows any variables in COMMON, including blank COMMON, to be initialized in any program unit by use of the DATA statement.

3.5 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 nonalphabetic.

RT-11 FORTRAN IV does not place any limits on the number of continuation lines per statement.

3.6 PAUSE AND STOP STATEMENTS

The PAUSE statement types the word PAUSE and the contents of the test string (if any) on the user's terminal. This causes temporary suspension of program execution. To resume program execution, type a carriage return.

For example, use of the FORTRAN IV statement:

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

causes the following line to print 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 program execution without printing the STOP message, use CALL EXIT (see Section B.7).

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

```
STOP—text
```

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

3.7 DEVICE/FILE DEFAULT ASSIGNMENTS

The device and file name default assignments are listed in Table 3-2. You can change the default device assignments prior to execution by using the OPEN statement or the monitor ASSIGN command.

The RT-11 monitor command:

```
.ASSIGN LP: 7
```

connects logical unit 7 to a physical device, the line printer. You can change the device and file name assignments at execution time by using the ASSIGN system subroutine or the FORTRAN IV OPEN statement. Valid logical unit numbers other than those listed below (10-99) are assigned to the system default device on RT-11. The default filename conventions hold for logical units not listed below. For example, logical unit number 49 will have a default file name of FTN49.DAT. Refer to the *RT-11 System User's Guide* for more information.

Table 3-2 FORTRAN IV Logical Device Assignments

Logical Unit Number	Default Device	Default File Name
1	System disk or public structure 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, PC:	FTN8.DAT
9	High-speed paper tape punch, PC:	FTN9.DAT

Although any combination of valid logical unit numbers can be used, there is an imposed maximum number of simultaneously active units. By default, six logical units can be concurrently active. The number can be changed by use of the /UNITS option (/N) in the compiler command string while compiling the main program unit (see Section 1.2.2).

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. In both cases, for example, assigning logical unit number 1 to the terminal 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 the ACCEPT, TYPE, and PRINT statements respectively.

3.8 MAXIMUM RECORD LENGTH

The line buffer allocated to store I/O records temporarily is by default 136 bytes. This restricts all I/O records in formatted I/O statements to a maximum of 136 characters. The size of this buffer, and consequently the maximum record length, can be changed by including the /RECORD option (/R) 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

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 might not be readily available. FORTRAN IV allows some extra flexibility in this situation. Under the RT-11 operating system, a file length specification of zero records causes a large contiguous file to be allocated initially and the unused portion to be automatically deallocated when the file is closed. The "END= " construction is particularly useful in this situation for determining the actual length of the file.

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

FORTRAN IV allows formatted input and output for transferring ASCII records, and unformatted and direct access input and output for transferring binary records. Note, however, that FORTRAN IV does not support ACCEPT or SEND I/O to other KB:s.

Some runtime errors can be intercepted and control transferred to a predetermined 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 4	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 seven-bit ASCII character is transferred. Also, null characters (bytes of zero) are not transmitted on input.

On output to any device, the record separator appended to each record consists of a carriage return character. For all records, regardless of their carriage control character, the carriage return can be suppressed by use of the "$" format separator character in the FORMAT statement (see Chapter 8 of the *PDP-11 FORTRAN IV Language Reference Manual*. The first character of each record is deleted from the record and is interpreted as a carriage control character.

3.10.2 Unformatted I/O

FORTRAN IV variable-length, binary records are output to unformatted, sequential access files, whose structure permits not only retrieval of data, but also file positioning operations such as BACKSPACE and REWIND.

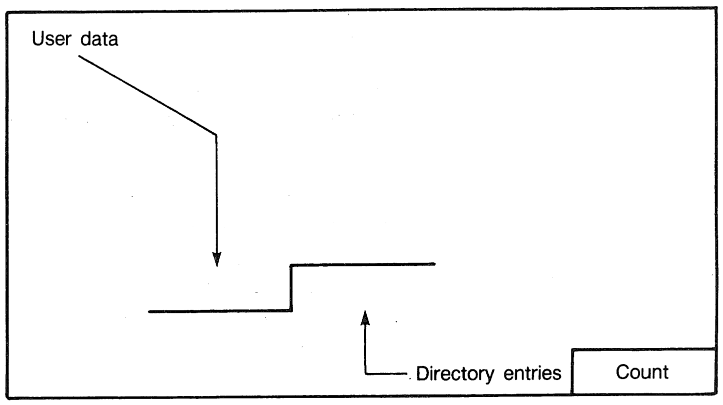
Unformatted, sequential access files are stored in blocks of 512 bytes, each organized as follows:

- User data extends forward from the beginning of the block
- Zero or more directory entries extend backward from the count word
- The count word occupies the last word of the block

This structure is illustrated in Figure 3-1.

Each time an unformatted record is written, new directory information for that record is stored in the block in which the record ends: first, a new directory entry is created to point to the end of the record; then the count word is corrected to reflect the current number of directory entries in the block.

Figure 3-1 Block Structure in Unformatted, Sequential Access Files



3.10.2.1 User Data

Unformatted, variable-length records can be constructed in units of either words or bytes. To perform byte-oriented unformatted I/O, link object module UIOBYT.OBJ with your program, either explicitly in the link command, or implicitly by first integrating the module into the OTS library according to the instructions given in the *RT-11 FORTRAN IV Installation Guide*.

3.10.2.2 Directory Entries

For each record in a file, one directory entry, pointing to the end of the record, is stored in the block in which the record ends. Each directory entry occupies one word. Its high-order bit, when set, indicates that the record to which it refers is an end-of-file record, produced by the ENDFILE statement.

3.10.2.3 Count Word

The last word of each block is reserved for the count word, whose low byte contains the count of directory entries in the block. Bits 15, 14, and 13 of the word are flags with the following interpretations:

Bit	Meaning when set
15	End of file, i.e., block is last in the file
14	Lowest directory entry is not stored
13	Omitted directory entry applies to an ENDFILE record

3.10.2.4 Omission of Lowest Directory Entry

With byte-oriented I/O only, it can happen that after storage of a record, just one byte remains unused in the block in which the record ends. Since directory entries occupy one word, storage of the directory entry for that record is impossible.

When this occurs, the directory entry for the record is omitted, and all information concerning the record is stored instead in the count word. First, bit 14 of the count word is set to flag the omission. Then, if the record is an end-of-file record, bit 13 of the count word is set as well. Finally, the count of directory entries is updated to include the missing entry.

For file-processing purposes, the end of such a record is computed as one more than the address at which the directory entry should have been located.

A subsequent record, if any, would begin not in the single free byte, but at the beginning of the next block.

3.10.2.5 Example of a File

For example, assume that the last three words (in octal) of block 0 of an unformatted file are:

```
100040, 40, 2
```

and the last four words (in octal) of block 1 of the same unformatted file are:

```
106, 56, 46, 100003
```

This is interpreted as follows:

Block 0 contains records 1,2, and the start of record 3. Record 1 extends from block 0, byte 0 to (but not including) block 0, byte 40. Record 2 is an ENDFILE record because bit #15 is on in the end-of-record pointer; ENDFILE records have no length. Record 3 spans blocks beginning at block 0, byte 40 and extends to (but does not include) block 1, byte 46. Block 1 also contains record 4 (byte 46 to 56) and record 5 (byte 56 to 106). Since the high-order bit of word 255 of block 1 is set, this file contains only 5 records.

3.10.2.6 Limitations

Since unformatted, sequential access files have a structure that is unique to FORTRAN IV, they are not easily read by programs written in another language. To construct a file for processing by a programming language other than FORTRAN IV, use formatted I/O for ASCII data and direct access I/O for binary data.

3.10.3 Direct-Access I/O

The direct access I/O routines read or write fixed-length, binary records. The logical record structure for a direct-access file is determined by the DEFINE FILE statement or the /RECORDSIZE option in the OPEN 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 you compare a single precision number to a double precision number, the double precision number might appear to be not equal to the single precision number in magnitude even though 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 because 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. With some other values (for example, 5.51), the single precision value will compare greater than the double precision value because of the conversion-rounding conventions from double precision to single precision value.

3.12 FORTRAN IV BUFFERED I/O

FORTRAN IV output to sequential files other than the console terminal is sent to a 512-character buffer, which is written to the output device only when the buffer is filled or the file is closed; this process is most noticeable in a program that generates line printer output.

If you want to force the current buffer to output to the line printer without closing and reopening the file, include a 'REWIND' statement at each of those points.

4 INCREASING FORTRAN IV PROGRAMMING EFFICIENCY

4.1 FACTORS AFFECTING PROGRAM EFFICIENCY

This chapter is directed to the programmer who is interested in minimizing program execution time or storage space requirements.

The relative efficiency of a FORTRAN IV object program derives from several factors, which can be categorized in two ways:

1 The way the programmer codes the source program

2 The way the compiler treats the source program

These two factors are interrelated. Compiler optimizations can be increased by certain programming techniques in the source program. The programmer should code the source program to use the FORTRAN IV constructs that the compiler handles most efficiently.

Section 4.2 discusses situations in which the compiler generates the most efficient code. Section 4.3, "Programming Techniques," includes hints on improving programming efficiency.

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

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

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

4.2 INCREASING COMPILATION EFFECTIVENESS

The following eleven programming suggestions will increase compilation effectiveness.

1 Effective use of the optimizer (space, time)

Avoid use of certain programming constructs to allow the optimizer greater freedom to discover common subexpressions in source programs. Specifically, avoid the use of equivalenced and COMMON variables, and SUBROUTINE and FUNCTION dummy artuments.

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 blocks are handled as efficiently as local variables.

Minimizing the number of elements in the argument list (by placing others in COMMON blocks) 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 as well.

4 Minimizing array vector table storage (space)

The FORTRAN IV array-vectoring feature is designed to decrease the time required to compute the address associated with an element of a multidimensional 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.

You can take the following steps to reduce the space required for array vectors:

- Specify the largest dimensions first in the statement that 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 that relates the number of words required for vector storage to the number of words required to store the array. If this tradeoff is favorable (for example, 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 (at one word each) because C has the largest second dimension, eight. Similarly, the vector for the third dimension has seven elements.

  In general, 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 nine.

- You can disable vectoring completely by specifying the /NOVECTORS (/V) option in the command string to the compiler. This causes no vector tables to be generated, but the resulting program executes more slowly 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 Multidimensioned array usage (time)

When you use multidimensional 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 I/O (space, time)

FORTRAN IV precompiles and compacts FORMAT statements that are presented in the source program. This affects the space required to store the format at run time and the speed of the I/O operations that make use of the format.

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

7 Data type selection (space, time)

Because of 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. FORTRAN IV performs extensive optimization on this data type.
- Use REAL*4 rather than double precision (REAL*8) wherever possible. Single precision operations are significantly faster than double precision operations, 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 that requires less executable code. For example:

```
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 Compilation options (space)

To minimize the space required for program execution, you should supply the following options to the compiler:

/NOLINENUMBERS (/S)—Suppresses line number traceback
/VECTORS (/V)—Suppresses all array vectoring

In addition, the /I4 (/T) (two-word integer default) option should not be specified unless it is required.

The /NOSWAP (/U) option should not be specified unless it is required; that is, no user-written interrupt or completion routines exist in the linked program.

Specify minimal values for the /UNITS (/N) and /RECORDS (/R) compiler options to conserve space at execution time. The /UNITS (/N) value should be the number of logical units that can be concurrently active. Set the /RECORDS (/R) option to the maximum formatted record length plus two (for the carriage return/line feed combination that can accompany a record).

11 Compilation options (time)

Specify the following compiler options to optimize an object program for execution time.

/NOLINENUMBERS (/S)—Suppresses line number traceback
/NOSWAP (/U)—Prevents the USR RT-11 file service routines from swapping at run time

Do not specify the following option, because global array vectoring speeds program execution.

/NOVECTORS (/V)—Disables array vectoring

4.3 PROGRAMMING TECHNIQUES

The following examples compare 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 left has been found more efficient than the technique on the right.

1 Make use of the increment parameter in DO loops.

```
          EFFICIENT                               INEFFICIENT

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

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

2 Avoid placing calculations within loops whenever possible.

```
          EFFICIENT                               INEFFICIENT

          TEMP1=B*C                               DO 10 I=1,20
          DO 10 I=1,20                            DO 20 J=1,50
          TEMP2=I*TEMP1                       20  A(J)=A(J)+I*B*C
          DO 20 J=1,50                        10  CONTINUE
      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 performed outside the loop. In the efficient example, 980 "floats" and 1979 floating multiplies are 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.

```
          EFFICIENT                               INEFFICIENT

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

In the inefficient 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 multidimensional arrays.

```
          EFFICIENT                               INEFFICIENT

          INTEGER A(20,100)                       INTEGER A(20,100)
          REAL*8 ATEMP(500)                       DO 20 I=1,100
          EQUIVALENCE (A,TEMP)                    DO 20 J=1,20
          DO 20 I=1,500                       20  A(J,I)=0
      20  ATEMP(I)=0.0D0
```

The efficient example makes use of the eight bytes in REAL*8 and, by equivalencing, places four integers in the array and zeros them in one operation, thus quartering the number of iterations.

5 Do as much calculation in INTEGER mode as possible.

```
EFFICIENT                    INEFFICIENT

A=B+(I+J)                      A=B+I+J
```

Also, do as much calculation as possible 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.

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

```
EFFICIENT                    INEFFICIENT

COMMON/SUBRA/A,B,C,D,E
COMMON/FUNCTA/Y,Z
       .
       .
       .
CALL SUBR                    CALL SUBR(A,B,C,D,E)
X=FUNCT()                    X=FUNCT(Y,Z)
CALL SUBR                    CALL SUBR(A,B,C,D,E)
       .                              .
       .                              .
       .                              .
END                          END
SUBROUTINE SUBR              SUBROUTINE SUBR(A,B,C,D,E)
COMMON/SUBRA/A,B,C,D,E                .
       .                              .
       .                              .
       .                     END
END
FUNCTION FUNCT               FUNCTION FUNCT(Y,Z)
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, but they must have empty parentheses for the compiler to recognize them as functions; for example, IGETC().**

7 Avoid division within programs wherever possible.

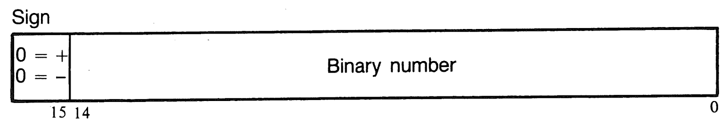
```
EFFICIENT                    INEFFICIENT

  A=B/2                        A= B*.5
```

Multiplication is faster than division and thus saves execution time.

A FORTRAN IV DATA REPRESENTATION

A.1 INTEGER FORMAT



Integers are stored in a two's complement representation. If the /I4 (/T) compiler option (see Section 1.2.1) is used, an integer is assigned two words, although only the low-order word (for example, 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 /I4 (/T) option. Integer constants must be in the range -32767 to +32767. For example:

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

A.2 FLOATING-POINT FORMATS

The exponent for both two-word and four-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 the bit 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 two-word format, or:

40200
0
0
0

in the four-word format.

-5 would be:

140640
0

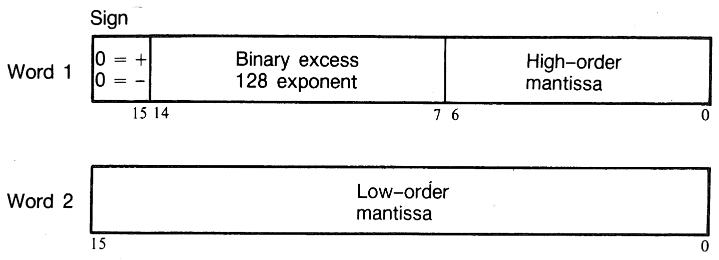
in the two-word format, or:

140640

0
0
0

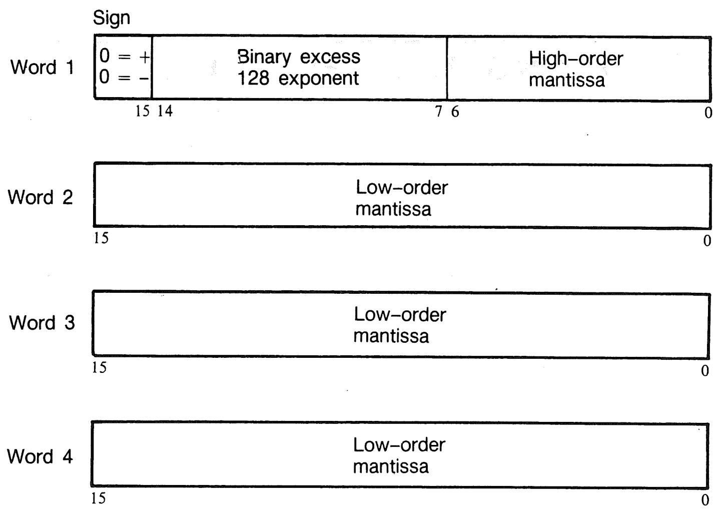
in the four-word format.

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



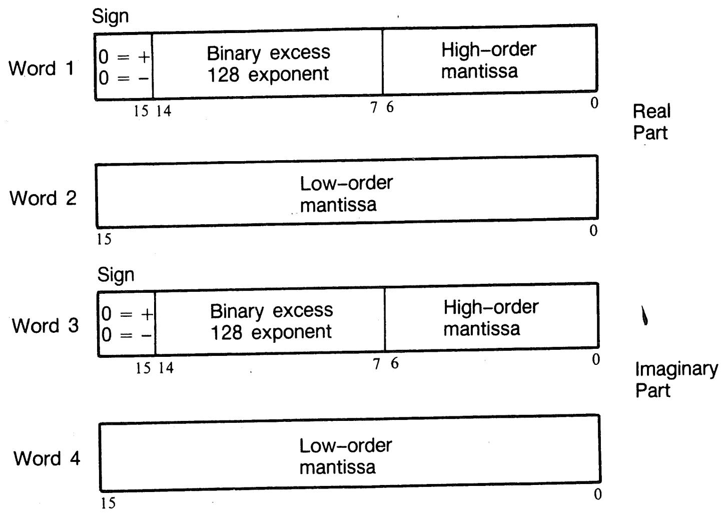
Since the high-order bit of the mantissa is always 1, it is discarded, giving an effective precision of 24 bits (or approximately 7 digits of accuracy). The magnitude range lies between approximately .29 X 10-38 and .17 X 1039.

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

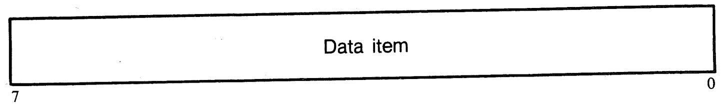


The effective precision is 56 bits (or approximately 17 decimal digits of accuracy). The magnitude range lies between .29 X 10-38 and .17 X 1039.

A.2.3 COMPLEX Format

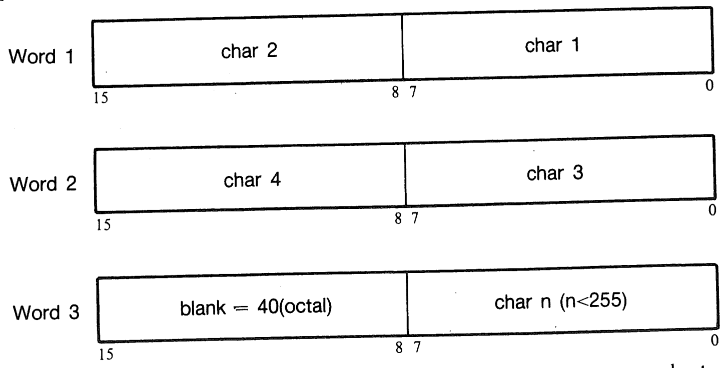


A.3 LOGICAL*1 FORMAT



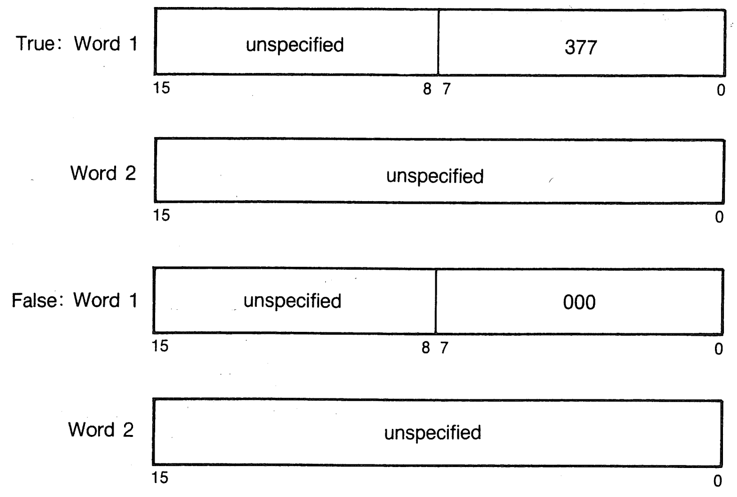
Any non-zero value is considered to have a logical value of .TRUE. The range of numbers from +127 to -128 can be represented in LOGICAL*1 format. LOGICAL*1 array elements are stored in adjacent bytes.

A.4 HOLLERITH FORMAT



Hollerith constants are stored internally, one character per byte. If necessary, Hollerith values are padded on the right with blanks to fill the associated data item.

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

Table A-1 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. The following example demonstrates conversion of the ASCII string X2B into Radix-50 representation. (Numeric values are octal.)

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

Table A-2 ASCII/Radix-50 Equivalents

Single or First Character	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

Table A-2 (Cont.) ASCII/Radix-50 Equivalents

Single or First Character	Second Character	Third Character
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

B LIBRARY SUBROUTINES

B.1 LIBRARY SUBROUTINE SUMMARY

In addition to the functions intrinsic to the FORTRAN IV system, you can call the following subroutines the same way you call a user-written subroutine.

ASSIGN	Allows specification at run time of the file name or device and file name to be associated with a FORTRAN IV logical unit number
CLOSE	Closes the specified logical unit after writing any active buffers to the file
DATE	Returns a nine-byte string containing the ASCII representation of the current date
ERRSNS	Allows the user program to obtain information about the most recent error that has occurred during program execution
ERRTST	Allows the user program to monitor the types of errors detected during program execution
EXIT	Terminates the excecution of a program and returns control to the executive
GETSTR	Reads a character string from a specified FORTRAN IV logical unit
IASIGN	Sets information in the FORTRAN IV logical unit table
ICDFN	Defines additional I/O channels
IDATE	Returns three integer values representing the current month, day, and year
IFETCH	Loads a device handler into memory
IFREEC	Returns the specified RT-11 channel to the available pool of channels for the FORTRAN IV I/O system
IGETC	Allocates an RT-11 channel and informs the FORTRAN IV I/O system of its use
IGETSP	Returns the address and size (in words) of free space obtained from the FORTRAN IV system
ILUN	Returns the RT-11 channel number with which a FORTRAN IV logical unit is associated
INTSET	Establishes a specified FORTRAN IV subroutine as an interrupt service routine with a specified priority

IQSET	Expands the size of the RT-11 monitor queue from the free space managed by the FORTRAN IV system
PUTSTR	Writes a variable-length character string on a specified FORTRAN IV logical unit
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
USEREX	Allows specification of a routine to be invoked as part of program termination

B.2 ASSIGN

The CALL ASSIGN statement should not be used in conjunction with the CALL OPEN statement. In proper context, the ASSIGN subroutine allows the association of device and file name 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 or the CLOSE statement, 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.

A description of the arguments of the ASSIGN routine follows.

Table B-1 ASSIGN Routine Arguments

Argument	Description
n	Logical unit number expressed as an integer constant or variable
name	Hollerith or literal string containing any standard file specification. If the device is not specified, then the device remains unchanged from the default assignments. If a file name is not specified, the default names, as described in Section 3.7, are used. The three options that can be included in the file specification are: /N specifies no carriage control translation. This option overrides the value of the 'control' argument. /C specifies carriage control translation. This option overrides 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 option overrides the value of the 'numbuf' argument. 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 (for example, 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 file name 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 IV 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, the CLOSE statement or program termination. If execution is aborted by typing $\boxed{\text{CTRL/C}}$, 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.

Table B-1 (Cont.) ASSIGN Routine Arguments

Argument	Description
	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 preceded by a linefeed 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 IV Language Reference Manual.) If not specifically managed 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.

B.3 CLOSE

The user has the option of the CALL CLOSE routine and the CLOSE statement. The CALL CLOSE routine is a subset of the CLOSE statement (see the *PDP-11 FORTRAN IV Language Reference Manual*). CLOSE explicitly closes 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 it is closed. After the execution of CALL CLOSE, any buffers 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 = An integer constant, variable, array element, or expression specifying the logical unit to be closed.

B.4 DATE

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

```
CALL DATE (array)
```

where:

array = A predefined array able to contain a nine-byte (nine-character) string.

The array specification in the call can be expressed as the array name alone:

```
CALL DATA (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 nine-byte string to begin at element i of the array a.

The date is returned as a nine-byte string in the following form:

```
dd-mmm-yy
```

where:

dd = Two-digit day date (with leading zero if necessary)
mmm = Three-letter month specification (uppercase)
yy = Last two digits of the year

For example:

```
25-DEC-86
```

The nine-byte array is set to blanks if the system date has not been set.

B.5 ERRSNS

ERRSNS allows specification of from zero to two arguments. When ERRSNS is called with zero arguments, previous error data is cleared, thus allowing testing for errors in certain program sections. The general call is:

```
CALL ERRSNS (ires,iunit)
```

where:

ires = An INTEGER*2 variable or array element into which the numeric code for the most recent error will be stored. A zero will be stored if there is no error.

iunit = An INTEGER*2 variable or array element into which the logical unit number of I/O errors will be stored, if the most recent error was I/O related.

B.6 ERRTST

ERRTST allows the user program to monitor the types of errors detected during program execution. The general form of the call is:

```
CALL ERRTST (ierr,ires)
```

where:

ierr = An INTEGER*2 quantity specifying the error number for which the test is to be done

ires = An INTEGER*2 variable or array element that is to receive the status of the error

ires=1 if an error has occurred
ires=2 if an error has not occurred

A call to ERRTST will reset the flag governing the specified error condition.

B.7 EXIT

A call to the EXIT subroutine, in the form:

```
CALL EXIT
```

is equivalent to the STOP statement except that no STOP message appears on the user's terminal. Use of the EXIT statement causes program termination, the closing of all files, and return to the monitor.

B.8 GETSTR

The GETSTR subroutine reads a formatted ASCII record from a specified FORTRAN IV logical unit into a specified array. The data is truncated (trailing blanks removed) and a null byte is inserted at the end to form a character string.

GETSTR can be used in main program routines or in completion routines, but it cannot be used in both at the same time. If GETSTR is used in a completion routine, it cannot be the first I/O operation on the specified logical unit.

The call has the form:

```
CALL GETSTR (lun,out,len,err)
```

where:

lun = The integer FORTRAN IV logical unit number of a formatted sequential file from which the string is to be read

out = The array to receive the string; this array must be at least one element longer than len

len = The integer number representing the maximum length of the string that is allowed to be input err = The LOGICAL*1 error flag that is set to .TRUE. if an error occurred. If an error did not occur, the flag is .FALSE.

Error conditions are indicated by err. If err is .TRUE., the values returned are as follows:

err = -1 End-of-file for a read operation
err = -2 Hard error for a read operation
err = -3 More than len bytes were contained in a record

The following example reads a string of up to 80 characters from logical unit 5 into the array STRING.

```
        LOGICAL*1 STRING(81) ,ERR


        CALL GETSTR(5,STRING,80,ERR)
```

B.9 IASIGN

The IASIGN function sets information in the FORTRAN IV logical unit table (overriding the defaults) for use when the FORTRAN IV OTS library opens the logical unit. This function can be used with ICSI (see the *RT-11 Programmer's Reference Manual*) to allow a FORTRAN IV program to accept a standard CSI input specification. IASIGN must be called before the unit is opened; that is, before any READ, WRITE, PRINT, TYPE, ACCEPT, or OPEN statements are executed that reference the logical unit.

The call has the form:

i = IASIGN (lun,idev[,ifiltyp[,isize[,itype]]])

where:

lun = An INTEGER*2 variable, constant, or expression specifying the FORTRAN IV logical unit for which information is being specified

idev = A one-word Radix-50 device name; this can be the first word of an ICSI input or output file specification

ifiltyp = A three-word Radix-50 file name and file type; this can be words 2 through 4 of an ICSI input or output file specification

isize = The length (in blocks) to allocate for an output file; this can be word 5 of an ICSI output specification. If 0, the larger of either one-half the largest empty segment or the entire second largest empty segment is allocated. If the value specified for length is -1, the entire largest empty segment is allocated

itype = An integer value determining the optional attributes to be assigned to the file. This value is obtained by adding the values that correspond to the desired operations:

1 Use double buffering for output

2 Open the file as a temporary file

3 Force a LOOKUP on an existing file during the first I/O operation. (Otherwise, the first FORTRAN IV I/O operation determines how the file is opened. Normally if the first I/O operation is a write, an IENTER would be performed on the specified logical unit. A read always causes a LOOKUP.

4 Expand carriage control information (see Notes below)

5 Do not expand carriage control information

6 File is read only

Notes:

Expanded carriage control information applies only to formatted output files and means that the first character of each record is used as a carriage control character when processing a write operation to the given logical unit. The first character is removed from the record and converted to the appropriate ASCII characters to simulate the requested carriage control.

If carriage control information is not expanded, the first character of each record is unmodified and the FORTRAN IV OTS library outputs a line feed, followed by the record, followed by a carriage return.

If carriage control is unspecified, the FORTRAN IV OTS library sends expanded carriage control information to the terminal and line printer and sends unexpanded carriage control information to all other devices and files. See the *PDP-11 FORTRAN IV Language Reference Manual* for further carriage control information.

Errors:

i = 0 Normal return
i < > 0 The specified logical unit is already in use, or there is no space for another logical unit association

The example below shows:

1 Creation of an output file on logical unit 3, using the first output file given to the RT-11 Command String Interpreter (CSI)

2 Setting up of the output file for double buffering

3 Creation of an input file on logical unit 4, based on the first input file specification given to the RT-11 CSI

4 Setting up the input file for read-only access

```
      INTEGER*2 SPEC(39)
      REAL*4 EXT(2)
      DATA EXT/GRDATDAT,GRDATDAT/     !DEFAULT FILE TYPE IS DAT


10    IF(ICSI(SPEC,EXT,,,0).NE.0) GOTO 10
C
C     DO NOT ACCEPT ANY SWITCHES
C
      CALL IASIGN(3,SPEC(1),SPEC(2),SPEC(5),1)
      CALL IASIGN(4,SPEC(16),SPEC(17),0,32)
```

B.10 ICDFN

The ICDFN function increases the number of I/O channels. Note that ICDFN defines new channels; any channels defined with an earlier ICDFN function are not used. Thus, an ICDFN for 20(decimal) channels (while the 16(decimal) original channels are defined) causes only 20 I/O channels to exist; the space for the original 16 is unused. The space for the new channel area is allocated out of the free space managed by the FORTRAN IV system.

The call has the form:

i = ICDFN (num[,area])

where:

num = The integer number of channels to be allocated. The number of channels must be greater than 16 and can be a maximum of 256. The program can use all new channels greater than 16 without a call to IGETC; the FORTRAN IV system I/O uses only the first 16 channels. This argument must be positioned so the USR cannot swap over it

area = The space allocated from within the calling program. Under FB and SJ monitors; be sure the space is outside the USR swapping area. If this argument is not specified, the space for the channels is allocated in the FORTRAN IV OTS library work area.

When you use the ICDFN call, follow these guidelines:

1 ICDFN cannot be issued from a completion or interrupt routine.

2 It is recommended that the ICDFN function beused at the beginning of the main program before any I/O operations are initiated.

3 If ICDFN is executed more than once, a completely new set of channels is created each time ICDFN is called.

4 ICDFN requires that extra memory space be allocated to foreground programs (see the *RT-11 Programmer's Reference Manual*).

5 Any channels that were open prior to the ICDFN are copied over to the new set of channel status tables.

This function can have the following results:

i = 0 Normal return
i = 1 An attempt was made to allocate fewer channels than already exist
i = 2 Not enough free space available for the channel area

For example:

```
IF(ICDFN(24).EQ.2) STOP 'NOT ENOUGH MEMORY'
```

B.11 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, 1986 the values of the integer variables upon return would.be:

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

If the system date has not been set, i is returned as zero.

B.12 IFETCH

The IFETCH function loads a device handler into memory from a system device, making the device available for I/O operations. The handler is loaded into the free area managed by the FORTRAN IV system. Once the handler is loaded, it cannot be released and the memory in which it resides cannot be reclaimed. IFETCH requires the USR and cannot be issued from a completion or interrupt routine. IFETCH issued from a foreground job will fail unless the handler is already in memory.

The function has the form:

```
i = IFETCH (devnam)
```

where:

devnam = The one-word Radix-50 name of the device for which the handler is desired. This argument can be the first word of an ICSI input or output file specification. This argument must be positioned so the USR cannot swap over it.

For further information on loading device handlers into memory, see the .FETCH programmed request in the *RT-11 Programmer's Reference Manual.*

Errors:

i = 0 Normal return
i = 1 Device name specified does not exist
i = 2 Not enough room to load the handler
i = 3 No handler for the specified device exists on the system device

The example requests that the DX handler be loaded into memory; execution stops if the handler cannot be loaded.

```
REAL*4 IDNAM
DATA IDNAM/3RDX/
.
.
.
IF (IFETCH(IDNAM).NE.O) STOP 'FATAL ERROR FETCHING HANDLER'
```

B.13 IFREEC

The IFREEC function returns a specified RT-11 channel to the available pool of channels. Before IFREEC is called, the specified channel must be closed or deactivated with a CLOSEC or ICLOSE or a purge (see the *RT-11 Programmer's Reference Manual*). IFREEC cannot be called from a completion or interrupt routine. IFREEC calls must be issued only for channels that have been successfully allocated by IGNETC calls; otherwise, the results are unpredictable.

The IFREEC function has the form:

i = IFREEC (chan)

where:

chan = The integer number of the channel to be freed

Errors:

i = 0 Normal return
i = 1 Specified channel not currently allocated

See the example in Section B.15 for the IGETC call.

B.14 IGETC

The IGETC function allocates an RT-11 channel, in the range 0 to 15 (decimal), to be used by other SYSLIB routines and marks it in use so that the FORTRAN IV I/O system will not access it. IGETC cannot be issued from a completion or interrupt routine.

The IGETC function has the form:

i = IGETC0

Function result:

i = n Channel n has been allocated

Error:

i = -1 No channels are available

The following example illustrates the IGETC function:

```
ICHAN=IGETC( )                  !ALLOCATE CHANNEL
IF(ICHAN.LT.0) STOP 'CANNOT ALLOCATE CHANNEL'


CALL IFREEC(ICHAN)              !FREE IT WHEN THROUGH


END
```

B.15 IGETSP

The IGETSP subroutine obtains free space from the FORTRAN IV system and returns the address and size (in number of words) of the allocated space. When this space is obtained, it is allocated for the duration of the program.

The IGETSP subroutine has the form:

i = IGETSP (min,max,iaddr)

where:

min = Minimum space to be obtained without an error indicating the desired amount of space is not available

max = Maximum space to be obtained

iaddr = Integer specifying the address of the start of the free space (buffer). Note that iaddr does not directly denote the storage area as a standard FORTRAN IV variable would. Rather, it denotes a word that contains the address of the storage space. It is most useful with IPEEK and IPOKE, or with assembly language subroutines.

Note: **Exercise extreme caution to avoid using all the free space allocated by the FORTRAN IV system. If the FORTRAN IV system runs out of dynamic free space, fatal errors (Error 29, 30, 42, etc.) occur. See the RT-11 System Message Manual.**

Function results:

i = n The actual size allocated whose value is min .LE. n .LE. max. The size (min, max, n) is specified in words.

i = -1 Not enough free space is available to meet the minimum requirements; no allocation was taken from the FORTRAN IV system free space.

An example of the IGETSP function follows.

```
N=IGETSP(256,256,IBUFF)                    !GET 256 WORD BUFFER
IF(N.LT.0) STOP 'CANNOT GET BUFFER SPACE!' !NO SPACE AVAILABLE
```

B.16 ILUN

The ILUN function returns the RT-11 channel number with which a FORTRAN IV logical unit is associated.

The ILUN function has the form:

i = ILUN (lun)

where:

lun = An integer expression whose value is a FORTRAN IV logical unit number in the range 1-99

Function results:

i = +n—RT-11 channel number n is associated with lun

Errors:

i = -1—Logical unit is not open
i = -2—Logical unit is opened to console terminal

For example:

```
    PRINT 99
99  FORMAT('PRINT DEFAULTS TO LOGICAL UNIT 6, WHICH FURTHER DEFAULTS TO LP:')
    ICHAN=ILUN(6)                !WHICH RT-11 CHANNEL IS RECEIVING I/O?
```

B.17 INTSET

The INTSET function establishes a FORTRAN IV subroutine as an interrupt service routine, assigns it a priority, and attaches it to a vector. INTSET requires that extra memory be allocated to foreground programs that use it (see the *RT-11 Programmer's Reference Manual*).

The INTSET function has the form:

i = INTSET (vect,pri,id,crtn)

where:

vect = The integer specifying the address of the interrupt vector to which the subroutine is to be attached

pri = The integer specifying the actual priority level (4-7) at which the device interrupts

id = The identification integer to be passed as the single argument to the FORTRAN IV routine when an interrupt occurs. This allows a single crtn to be associated with several INTSET calls

crtn = A FORTRAN IV subroutine to be established as the interrupt routine. This name should be specified in an EXTERNAL statement in the FORTRAN IV program that calls INTSET. The subroutine has one argument:

SUBROUTINE crtn(id)
INTEGER id

When the routine is entered, the value of the integer argument is the value specified for id in the appropriate INTSET call.

Follow these guidelines when using the INTSET function.

1 The id argument can be used to distinguish between interrupts from different vectors if the routine to be activated services multiple devices.

2 When you use INTSET with the FB or XM monitor, you must use the SYSLIB call DEVICE in almost all cases to prevent interrupts from interrupting beyond program termination.

3 If the interrupt routine (crtn) has control for a period of time longer than the time in which two more interrupts using the same vector occur, interrupt overrun is considered to have occurred. The error message:

?SYSLIB-F-Interrupt overrun

is printed and the job is aborted. Jobs requiring very fast interrupt response are not viable with FORTRAN IV, since FORTRAN IV overhead lowers RT-11's interrupt response rate.

4 The interrupt routine (crtn) is actually run as a completion routine by the RT-11 .SYNCH macro. The pri argument is used for the RT-11 .INTEN macro.

5 A .PROTECT request is issued for the vector, but no attempt is made to report an error if the vector is already protected; the vector is taken over unconditionally. (See the .PROTECT programmed request in the *RT-11 Programmer's Reference Manual.*)

6 The FORTRAN IV interrupt service subroutine (crtn) cannot call the USR.

7 INTSET cannot be called from a completion or interrupt routine.

8 Interrupt enable should not be set on the associated device until the INTSET call has been successfully executed.

Errors:

i = 0 Normal return
i = 1 Invalid vector specification
i = 2 Reserved for future use
i = 3 No space is available for the linkage setup

An example of the INTSET function follows.

```
EXTERNAL CLKSUB                      !SUBR TO HANDLE KW11-P CLOCK
 .
 .
 .
I=INTSET("104,6,0,CLKSUB)            !ATTACH ROUTINE
IF (I.NE.0) GOTO 100                 !BRANCH IF ERROR
 .
 .
 .
END
SUBROUTINE CLKSUB(ID)
 .
 .
 .
END
```

B.18 IQSET

You use the IQSET function to make the RT-11 I/O queue larger - that is, to add available elements to the queue. These elements are allocated out of the free space managed by the FORTRAN IV system. IQSET cannot be called from a completion or interrupt routine.

The IQSET function has the form:

i = IQSET (qlent[,area])

where:

qleng = The integer number of elements to be added to the queue. This argument must be positioned so the USR does not swap over it.

area = The space allocated from within the calling program. Under FB and SJ monitors, make sure the space is outside the USR swapping area. If this argument is not specified, the space for the elements is allocated in the FORTRAN IV OTS library work area.

All RT-11 I/O transfers are done through a centralized queue management system. If I/O traffic is very heavy and not enough queue elements are available, the program issuing the I/O requests is suspended until a queue element becomes available. In an FB or XM system, the other job can run while the first program waits for the element. When IQSET is used in a program to be run in the foreground, the FRUN command must be modified to allocate space for the queue elements (see the *RT-11 Programmer's Reference Manual*).

It is good programming practice to follow the rule that each program should contain one more queue element than the total number of I/O and timer requests that will be active simultaneously. Timing functions such as ITWAIT and MRKT also cause elements to be used and must be considered when you allocate queue elements for a program. Note that if synchronous I/O is done (for example, IREADW/IWRITW) and no timing functions are done, no additional queue elements need be allocated. Note also that FORTRAN IV allocates four queue elements by default.

The following subroutines require queue elements:

IRCVD/IRCVDC/IRCVDF/IRCVDW	ITIMER
IREAD/IREADC/IREADF/IREADW	ITWAIT
ISCHED	IUNTIL
ISDAT/ISDATC/ISDATF/ISDATW	IWRITE/IWRITC/IWRITF/IWRITW
ISLEEP	MRKT
ISPFN/ISPFNC/ISPFNF/ISPFNW	MWAIT

For further information on adding elements to the queue, see the .QSET programmed request.

Errors:

i = 0 Normal return
i = 1 Not enough free space is available for the number of queue elements to be added; no allocation was made.

The following example illustrates the IQSET function:

```
IF9IQSET(5).NE.0) STOP 'NOT ENOUGH FREE SPACE FOR QUEUE ELEMENTS'
```

B.19 PUTSTR

The PUTSTR subroutine writes a variable-length character string to a specified FORTRAN IV logical unit. PUTSTR can be used in main program routines or in completion routines but not in both programs at the same time. If PUTSTR is used in a completion routine, it must not be the first I/O operation on the specified logical unit.

The PUTSTR function has the form:

CALL PUTSTR (lun,in,char,err)

where:

lun = The integer specification of the FORTRAN IV logical unit number to which the string is to be written in = The array containing the string to be written

char = An ASCII character appended to the beginning of the string before it is output. If 0, no extra character is output. This character is used primarily for carriage control purposes.

err = A LOGICAL*1 variable that is .TRUE. for an error condition and .FALSE. for a no-error condition

Errors:

err = -1 End-of-file for write operation
err = -2 Hardware error for write operation

The following example illustrates the PUTSTR function.

```
LOGICAL*1 STRING(81),ERR
.
.
.
!OUTPUT STRING WITH DOUBLE SPACING
CALL PUTSTR(7,STRING,'0',ERR)
```

B.20 RANDU,RAN

The pseudo 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)

i(1) and i(2) = Previously defined integer variables
x = The real variable name, in which a random number between 0 and 1 is returned

i(1) and i(2) should be set initially to zero. i(1) and i(2) are updated to a new generator base during each call. Resetting i(1) and i(2) to zero 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.

The random number generator can also be called as a function, as follows:

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

B.21 SETERR

SETERR allows the user to specify the disposition of certain OTS library-detected error conditions. Only OTS library error diagnostics 1-16 should be changed from their default error classification (see Section C.2). If errors 0 or 20-69 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 = An integer variable or expression specifying the OTS library error number (see Section C.2) ncount = A LOGICAL*1 variable expression with one of the following values:

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

128-255—Ignore all occurrences of the error

B.22 USEREX

The USEREX subroutine allows specification of a routine written in MACRO-11 to which control is passed as part of program termination. This allows disabling of interrupts enabled in non-FORTRAN IV 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 = 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.

C FORTRAN IV ERROR DIAGNOSTICS

C.1 COMPILER ERROR DIAGNOSTICS

The FORTRAN IV compiler, while reading and processing the FORTRAN IV source program, can detect syntax errors (or errors in general form) such as unmatched parentheses, illegal characters, unrecognizable key words, and 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 IV 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. You should take care to avoid the following common typing mistakes:

- Missing commas or parentheses in a complicated expression or FORMAT statement
- Misspelling of particular instances of variable names. If the compiler does not detect this error (it usually cannot), execution might also be affected.
- An inadvertent line continuation signal on the line following the statement in error
- If the user terminal does not clearly differentiate between zero (0) and the letter O, what appear to be identical spellings of variable names might not appear so to the compiler, and what appears to be a constant expression might 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:

```
?FORTRAN-I-[name] Errors:n,Warnings:m
```

where:

[name] = The six-character name of the program unit being compiled. .MAIN. is the default name of the main program if no PROGRAM statement is used. The default name for BLOCK program units is .DATA.

n = The number of error-class messages (for example, those messages that cause the statement in question to be deleted)

m = The number of warning-class messages (for example, 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 following four sections describe the initial phase and secondary phase error diagnostics and the fatal FORTRAN IV compiler error diagnostics.

The following example demonstrates a FORTRAN IV program with diagnostics issued by the compiler.

```
FORTRAN IV       V02.8       Thu 20-Nov-86  00:41:51           Page 001

0001            DOUBLE PRECISION DBLE DBLE2
0002            DATA INT/100/ DBLE2/500./
0003            DBLE2 = INT/2 + 5. + DBLE
0004            WRITE,(6,10) DBLE,DBLE
0005     10     FORMAT(1X,2F12.8)
0006     10     STOP
*****    M
0007            INTEGER INT
0008            END


FORTRAN IV        Diagnostics for Program Unit  MAIN

In line 0004,  Error:     Syntax error
In line 0008, Warning:    Variable "DBLEDB" name exceeds 6 characters
In line 0009, Warning:    Non-standard statement ordering


FORTRAN IV        Storage Map for Program Unit  MAIN

Local Variables,  PSECT  $DATA, Size = 000024 (    10. words)

Name   Type  Offset       Name    Type  Offset       Name    Type  Offset
DBLE   R*4   000020       DBLEDB  R*8   000010       DBLE2   R*4   000004
INT    I*2   000002
```

C.1.1 Errors Reported by the Initial Phase of the Compiler

Some of the easily recognizable FORTRAN IV 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 = A code letter whose meaning is described below

Table C-1 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 ignored.
I	Non-FORTRAN IV character used. The line contains a character that is not in the FORTRAN IV 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).
P	Statement contains unbalanced parentheses. The statement is ignored.
S	Syntax error such as multiple equal signs, etc. The statement is not of the general FORTRAN IV statement form; the statement is ignored.
U	Statement could not be identified as a legal FORTRAN IV statement. The statement is ignored.

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:

```
                    Error:
IN LINE nnnn,                 text
                    Warning:
```

where:

nnnn = The internal sequence number of the statement in question
text = A short description of the error

The error diagnostics are listed alphabetically below. Included with each diagnostic is a brief explanation. Refer to the *PDP-11 FORTRAN IV 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.

Table C-2 Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
ACCESS='DIRECT' REQUIRES FORM='UNFORMATTED'	FORM='FORMATTED' has been specified for a direct access file. FORTRAN IV supports only unformatted direct access I/O. Correct the program logic.
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 can 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 1 to 7.
**** ATTEMPTS TO EXTEND COMMON BLOCK 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 a decimal point. Check the format for relational operators; correct the statement in error.
DEFINE FILE MODE MUST BE 'U'	The third argument inside parentheses in a DEFINE FILE statement is not 'U' (unformatted). Correct the mode specification.
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.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
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.
EXPECTING LEFT PARENTHESIS AFTER SUBPROGRAM NAME	A SUBROUTINE or FUNCTION name occurs without an argument list specification. Check for typographical error or the use of the same name for a local variable and a subprogram.
EXTRA CHARACTERS AT END OF STATEMENT	All the necessary information for a syntactically correct FORTRAN IV 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 CHARACTERS IN EXPRESSION	An illegal character has been found in an expression. Check for a typographical error in the statement.
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 ENCODE/DECODE FORMAT SPECIFIER	The format specification (second argument inside parentheses) in an ENCODE/DECODE statement is not a FORMAT statement label or array name. Correct the FORMAT specification.
ILLEGAL ENCODE/DECODE LENGTH EXPRESSION	The length specification (first argument inside parentheses) in an ENCODE or DECODE statement is not an integer expression. Correct the length expression.
ILLEGAL ENCODE/DECODE TARGET	The third argument inside parentheses in an ENCODE or DECODE statement is not the name of an array, array element, or variable. Correct the target specification.
ILLEGAL INITIAL VALUE EXPRESSION IN DO STATEMENT	A valid integer expression does not follow the equals sign in a DO statement. Correct the initial value expression.
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.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
ILLEGAL SUBSCRIPTS OR SUBPROGRAM ARGUMENT	An illegal element occurred within a subscript list or argument list to a subprogram. Correct the erroneous 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 END= OR ERR= KEYWORD	The END= or ERR= specification in an input/output statement is incorrectly formatted. Check for a typographical error in the statement.
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 (for instance, nonalphabetic 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 PROGRAM 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 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.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
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 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 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 COMMA IN OPEN OR CLOSE KEYWORD LIST	Two options in an OPEN/CLOSE keyword list are not separated by a comma. Check for a typographical error in the statement.
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 EXPRESSION	A required expression (for example, the limit expression in a DO statement) was omitted. Correct the syntax of the statement.
MISSING LABEL	A statement label was expected but not found. For example, ASSIGN J TO I is illegal; a valid statement label reference should but does not precede 'TO.' 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 LABEL LIST AFTER COMMA	In an assigned GOTO statement, the integer variable was followed by a comma but no list was found. Check for a typographical error in the statement.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
MISSING LEFT PARENTHESIS AFTER OPEN OR CLOSE	An OPEN or CLOSE statement does not have a left parenthesis preceding the keyword list. Check for a typographical error in the statement.
MISSING OPERATOR AFTER EXPRESSION	An expression was not terminated by a comma, right parenthesis, or other operator. Check for a typographical error in the statement.
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 RIGHT PARENTHESIS	A right parenthesis was expected but not found. For example, READ(5,100.) is illegal; the first nonblank character after the format reference should be a right parenthesis but is not. Correct the format of the statement in error.
MISSING 'TO' IN ASSIGN STATEMENT	The keyword 'TO' does not follow the label specification in an ASSIGN statement. Check for a typographical error in the statement.
MISSING VALUE FOR KEYWORD IN OPEN OR CLOSE STATEMENT	A keyword requiring a value was specified without a value. Correct the syntax of the statement.
MISSING VARIABLE	A variable was expected, but not found. For example, ASSIGN 100 TO 1 is illegal; a variable name should but does not follow the 'TO.' 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.
MODE OF EXPRESSION MUST BE INTEGER	An integer variable or expression is required. For example, in the initial, terminal, and incremental parameters of a DO statement.
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.
MULTIPLE DECLARATION OF OPEN OR CLOSE KEYWORD	A keyword has been specified more than once in a single OPEN or CLOSE statement. Remove the incorrect or duplicate reference to the keyword.
OPEN OR CLOSE KEYWORD VALUE MUST BE QUOTED STRING	A keyword that requires a quoted-string value was given an expression value. Correct the syntax of the statement.
OPEN OR CLOSE STATEMENT REQUIRES UNIT= SPECIFIER	No UNIT= specification is present in the OPEN or CLOSE statement in question to select the desired logical unit. Add the UNIT= specification to the statement.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
PARENTHESES NESTED TOO DEEPLY	Group repeats in a FORMAT statement have been nested too deeply. Limit group repeats to eight levels of nesting.
PROGRAM OR BLOCK DATA STATEMENT MUST BE FIRST	A program name or block data name statement was used, but was not the first statement found. Correct the program so that the statement is first.
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 undefined. 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.
SYNTAX ERROR IN LABEL LIST	The list of labels for an assigned or computed GOTO statement is improperly formatted or contains a reference to an entity that is not the label on an executable statement. Correct the format of the list.
TARGET MUST BE ARRAY	An array element was referenced in an ENCODE or DECODE statement without having been previously dimensioned. Correct the program logic.
UNARY OPERATOR HAS TOO MANY OPERANDS	Two operands have been specified for an operator (such as .NOT.) that accepts only one operand. Check for a typographical error in the statement.

Table C-2 (Cont.) Secondary Phase Error Diagnostics

Diagnostic	Explanation/Fix
UNLABELED FORMAT STATEMENT	A FORMAT statement was not labeled. Correct the FORMAT statement in error by assigning it the proper label.
UNRECOGNIZED KEYWORD IN OPEN OR CLOSE STATEMENT	The OPEN or CLOSE statement in question contains a keyword that is not recognized by the compiler. Check for a misspelling or typographical error in the keywords used in the statement.
UNRECOGNIZED VALUE FOR OPEN OR CLOSE KEYWORD	A keyword that requires a quoted-string value was specified with an unrecognized value string. For example, DISPOSE = 'SURE'. Specify a valid value for the keyword.
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 OPERANDS FOR BINARY OPERATOR	An operator that requires two operands was specified with only one operand. Example: I=_*J. Check for a typographical error in the statement.
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 that are not true error conditions but can be potentially dangerous at execution time, or present compatibility problems with FORTRAN IV compilers running on other DIGITAL operating systems. The warning diagnostics are normally suppressed, but can be enabled by use of the /WARNINGS (/W) compiler option. The form and placement of the warning diagnostics are the same as those for the secondary phase error diagnostics (see Section C.1.2). A listing of the warning diagnostics follows.

Table C-3 Warning Diagnostics

Diagnostic	Explanation
LOOP ENTRY AT LABEL ****	A transfer of control occurs from outside the containing DO loop to the label indicated. This might indicate a programming error (if the loop does not have extended range).
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 IV Language Reference Manual, nonadherence to the stricter requirements might cause error conditions on other FORTRAN IV compilers. See Chapter 3 of this document.
POSSIBLE MODIFICATION OF ****	The indicated variable, used as a control parameter of a DO loop, might be modified within the body of that loop.
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 run time 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 IV compilers might treat this as an error condition. See Section 3.2 of this document.

C.1.4 Fatal Compiler Error Diagnostics

The fatal compiler error diagnostics are listed below. These diagnostics, sent directly to the initiating terminal, report hardware error conditions, conditions that might require rewriting of the source program, and compiler errors that might require attention from your Software Support representative.

Table C-4 Fatal Compiler Error Diagnostics

Diagnostic	Explanation/Fix
?FORTRAN-F-CODE GENERATION STACK OVERFLOW	A statement is too complex to process. Simplify complex statements.
?FORTRAN-F-COMPILER FATAL ERROR, ANALYSIS FOLLOWS	Some type of unexpected error was encountered by FORTRAN IV. Please send the console listing with a copy of your program (on some machine-readable medium) with an SPR report.
?FORTRAN-F-CONSTANT SUBSCRIPT STACK OVERFLOW	Too many constant subscripts have been employed in a statement. Simplify the statement.
?FORTRAN-F-DEVICE FULL	There is insufficient room available on an output device specified to create the object or listing files required. Make more space available on the device by deleting unnecessary files and/or using the SQUEEZE command, or by redirecting the object on listing files to another device.
?FORTRAN-F-DYNAMIC MEMORY OVERFLOW	The program unit currently being compiled cannot be processed in the available memory space. Break the program unit in question into smaller subprograms, or recompile on a larger machine.
?FORTRAN-F-ERROR READING SOURCE FILE	An unrecoverable error occurred while the compiler was attempting to read a source program input file. Correct the hardware problem and recompile.
?FORTRAN-F-ERROR WRITING LISTING FILE	An unrecoverable error occurred while the compiler was attempting to write the listing output file. Make sure that the output device is write-enabled, and that sufficient free space exists on the device for the output file. Recompile the program.
?FORTRAN-F-ERROR WRITING OBJECT FILE	An unrecoverable error occurred while the compiler was attempting to write the object program output file. Make sure the output device is write-enabled, and that sufficient free space exists on the device for the output file. Recompile the program.
?FORTRAN-F-FILE NOT FOUND	An input file specified in the command string was not found. Correct the command string to refer to an existing file.
?FORTRAN-F-HELP FILE NOT FOUND	The FORTRAN IV help file, SY:FORTRA.HLP, was not present on the system device when the help switch was given to the compiler. No help information is available. Replace the file from the FORTRAN IV distribution medium if help information is required.
?FORTRAN-F-ILLEGAL VALUE FOR /x SWITCH	An illegal value has been specified for a compiler command string switch. Refer to Section 1.2.2 for compiler option information.

Table C-4 (Cont.) Fatal Compiler Error Diagnostics

Diagnostic	Explanation/Fix
?FORTRAN-F-ILLEGAL COMMAND	The command string presented to the compiler was illegal in format. Correct the command string.
?FORTRAN-F-ILLEGAL DEVICE	A device specification in a compiler command string was illegal. Correct the command string.
?FORTRAN-F-OPTIMIZER STACK OVERFLOW	A statement is too complex to process, or too many common subexpressions occurred in one basic block of the source program. Simplify complex statements.
?FORTRAN-F-SUBEXPRESSION STACK OVERFLOW	An attempt was made to compile a statement that could overflow the runtime stack at execution time. Simplify complex statements.
?FORTRAN-F-UNKNOWN SWITCH-/x	An illegal switch has been specified in the compiler command string. Refer to Section 1.2.2 for compiler option information.

C.2 OBJECT TIME SYSTEM LIBRARY ERROR DIAGNOSTICS

The Object Time System (OTS) library 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 = A decimal error identification number

The long form of the message appears as:

?ERR nn text

where:

nn = A decimal error identification number
text = A short error description

For RT-11, the default message form is long. The short message error module can be linked to the program by using the /I linker switch. The global named $SHORT should be included from the FORTRAN IV library.

There are four classes of OTS library 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 20-69 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. Execution continues.
FATAL	The error message is sent to the terminal. Execution terminates.
COUNT:n	The error message is sent to the terminal. 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 cannot be 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 might have been lost.

The OTS library error diagnostics are listed in Table C-5, with the error type and a brief explanation where necessary.

Note: **Referring to Table C-5, any error diagnostics classified as FATAL with an error number of 20 or greater should not be changed from FATAL by use of the system subroutine SETERR.**

Table C-5 OTS Library Error Diagnostics

Error Number	Error Type	Message
0	FATAL	NON-FORTRAN IV ERROR CALL This message indicates an error condition (not internal to the FORTRAN IV runtime system) that might have been caused by one of four situations: 1 For RT-11 only, a foreground job using SYSLIB completion routines has not allocated enough space (using the FRUN /N option) for the initial call to a completion routine. Check the <u>RT-11 Advanced Programmer's Guide</u> "System Subroutine Library" for the formula used to allocate more space. 2 For RT-11 only, there was not sufficient memory for the background job. Make more memory available by unloading unnecessary handlers, deleting unwanted files, compressing the device. 3 For RT-11 only, under the single-job monitor, a SYSLIB completion routine interrupted another completion routine. Use the FB monitor to allow more than one active completion routine. 4 An assembly language module linked with a FORTRAN IV program issued a TRAP instruction with an error code that was not recognized by the FORTRAN IV 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.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
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 GENERATED ERROR An attempt was made to execute a FORTRAN IV 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 one 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:4	INPUT CONVERSION ERROR During a formatted or list-directed 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.
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:4	FLOATING OVERFLOW During an arithmetic operation, the absolute value of a floating-point expression exceeded the largest representable real number. An indeterminate value 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:4	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.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
15	FATAL	LOG OF ZERO OR 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.
16	FATAL	WRONG NUMBER OF ARGUMENTS One of the FORTRAN IV library functions, or one of the system subroutines that 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 them all.
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 /UNITS (/N) option 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 /RECORDS (/R) option to specify a larger runtime 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 not within the bounds of the file. Use the "END=" parameter to detect this condition, or correct the program logic so 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. For RT-11, try to make more file space available by deleting unnecessary files and compressing the device, or by using 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.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
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.)
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, there was no room on the device, or there was an attempt to create a file that already exists as a protected file. 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.
30	FATAL	NO ROOM FOR BUFFERS There was not enough free memory left to set up required I/O buffers. For RT-11 only, reduce the number of logical units open simultaneously at the time of the error. If you are 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.
31	FATAL	NO AVAILABLE I/O CHANNEL More than the maximum number of channels available to the FORTRAN IV runtime system exclusive of the terminal were requested to be opened simultaneously for I/O (15 for RT-11). 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 TT OR LP 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 OPEN statement, the ASSIGN or OPEN FORTRAN IV library routine, or, for RT-11, only the IASIGN SYSLIB routine.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
35	FATAL	ATTEMPT TO OUTPUT TO READ ONLY FILE An attempt was made to write on a file designated as read only. Check the OPEN statement, or for RT-11 only the IASIGN SYSLIB function to ensure that the correct arguments were used. Check for a possible programming error.
36	FATAL	BAD FILE SPECIFICATION STRING The Hollerith or literal string specifying the device/file name OPEN statement, in the CALL ASSIGN or CALL OPEN system subroutine, could not be interpreted. Check the format of the OPEN statement, 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 the DEFINE FILE operation is executed before any random-access read or write operation.
38	FATAL	RANDOM I/O NOT ALLOWED ON TT OR LP 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, OPEN statement, the ASSIGN or OPEN FORTRAN IV library routine, or for 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 the CLOSE statement before attempting another DEFINE FILE.
42	FATAL	MEMORY OVERFLOW COMPILING OBJECT TIME FORMAT The OTS library ran out of free memory while scanning an array format generated at run time. For 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 if possible, using the single-job monitor, or SET USR SWAP if possible.
43	FATAL	SYNTAX ERROR IN OBJECT TIME FORMAT A syntax error was encountered while the OTS library was scanning an array format generated at run time. Correct the programming error.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
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 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 (for 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 and threaded code is selected at compile time.) 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 FILE An ENDFILE statement contains a unit number of a file that is open as a random access file.
50	FATAL	KEYWORD VALUE ERROR IN OPEN STATEMENT A numeric value specified for a keyword in the OPEN statement is not within the accepted range. Check the expression in the OPEN statement, and modify to yield a valid value.
51	FATAL	INCONSISTENT OPEN/CLOSE STATEMENT SPECIFICATIONS The specifications in an OPEN or subsequent CLOSE statement have indicated one or more of the following: • A 'NEW' or 'SCRATCH' file that is 'READONLY' • 'APPEND' to a 'NEW,' 'SCRATCH,' or 'READONLY' file • 'SAVE' or 'PRINT' of a 'SCRATCH' file • 'DELETE' or 'PRINT' of a 'READONLY' file Correct the OPEN or CLOSE statement to remove the conflict.
52	WARNING	ATTEMPT TO DELETE A PROTECTED FILE (for RT-11 only) An attempt was made through a DISP= 'DELETE' option in an OPEN or CLOSE statement to delete a protected file. Either unprotect the file or correct the conflict in the OPEN and/or CLOSE statement.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
53	WARNING	LIST DIRECTED I/O SYNTAX ERROR The repeat count of the input record has the wrong type or value. The repeat count must be a positive non-zero integer.
59	WARNING	USR NOT LOCKED (for RT-11 only) This message is issued when the FORTRAN IV program is started. If the program was running in the foreground, the /NOSWAP option was used during compilation, and the USR was swapping (for example, a SET USR NOSWAP command has not been done). Reexamine the intent of the /NOSWAP option at compile time and either compile without /NOSWAP or issue a SET USR NOSWAP command.
60	FATAL	STACK OVERFLOWED The hardware stack overflowed. More stack space might be required for subprogram calls and opening a file. Proper traceback is impaired. This message occurs in the background only. Allocate additional space by using the /BOTTOM (/B) option 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, ensure that the correct FORTRAN IV library was called.
62	FATAL	FORTRAN IV START FAIL The program was loaded into memory but there was not enough free memory remaining for the OTS library to initialize work space and buffers. For 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 /BUFFER option. Refer to the RT-11 Advanced Programmers Guide "System Subroutine Library" for the correct formula.
63	FATAL	ILLEGAL INSTRUCTION The program attempted to execute an illegal instruction (for example, 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 IV library was called.

Table C-5 (Cont.) OTS Library Error Diagnostics

Error Number	Error Type	Message
64	FATAL	VIRTUAL ARRAY INITIALIZATION FAILURE • The total storage requirements for virtual arrays in the program exceeds the currently available memory on the system. • Another job is currently using virtual array support under the FB or SJ monitor. • PLAS virtual array support is attempted under the FB or SJ monitor. • Any non-PLAS virtual array support is attempted under XM monitor. • PLAS support is attempted without EIS hardware. • Virtual array support is attempted without OTS library virtual array support. If another job is currently executing (under the XM monitor with PLAS virtual array support), make sure the total virtual storage demands for both programs are satisfied by the available memory on the machine. Reduce virtual storage requirements by decreasing the size of virtual arrays declared in the program.
65	FATAL	VIRTUAL ARRAY MAPPING ERROR An attempt has been made to reference outside the bounds of the extended memory region allocated to virtual arrays in this program, probably caused by a subscript out of bounds. Verify that the subscripts of the virtual arrays referenced in the statement indicated are within the declared bounds.
66	FATAL	UNSUPPORTED OPEN/CLOSE KEYWORD OR OPTION A keyword or keyword value in the OPEN or CLOSE statement indicated is invalid under this particular operating system. Refer to Section 3.1.1 for system-dependent restrictions.
67	WARNING	UNSUPPORTED OPEN/CLOSE KEYWORD OR OPTION A keyword or keyword value in the OPEN or CLOSE statement indicated is not meaningful in the current operating system environment. The presence of the keyword or option is ignored.
68	FATAL	DIRECT ACCESS RECORD SIZE ERROR The size of a direct access record exceeds 32767 double words. Correct the program logic.

D COMPATIBILITY WITH FORTRAN-77

FORTRAN IV is an implementation of FORTRAN for the PDP-11 computer system, available under the RT-11, RSTS/E, RSX-11M, RSX-11M-PLUS, IAS, and VAX-11 RSX operating systems.

D.1 DIFFERENCES BETWEEN FORTRAN-77 AND FORTRAN IV

This section summarizes differences that can affect conversion from FORTRAN IV to FORTRAN-77 (F77).

D.1.1 Language Differences

F77 transforms FORTRAN-defined function name references into a special kind of internal calling form, while FORTRAN IV does not. If you supply a routine of your own to replace the F77 FORTRAN-defined routine (for example, you write your own SIN routine), you will generally have 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.1.2 Implementation Differences

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

  F77 tests only the highest-order bit of the value and treats a one as .TRUE. and a zero as .FALSE.

2 In FORTRAN IV, INTEGER*4 causes 32-bit allocation (4 bytes) but only 16 bits are used for computation. In F77, INTEGER*4 causes both 32-bit allocation and 32-bit computation.

3 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. F77 does check at run time.

4 FORTRAN IV permits an unlimited number of continuation lines. In F77, up to five continuation lines are permitted by default, and up to 99 can be obtained by means of the compiler /CO switch.

5 For unformatted I/O operations, both sequential and direct access, FORTRAN IV and F77 both read/write four bytes of data if the variable is allocated four bytes of storage. However, since INTEGER*4 values in FORTRAN IV generally have an undefined high-order part, they generally cannot be read as INTEGER*4 values by F77. Similarly, since FORTRAN IV and F77 logical tests are different (see item 1 above), care must be taken when interchanging logical values.

6 For random access I/O operations, FORTRAN IV takes the END= exit on an end-file condition and the ERR= exit only for hardware I/O errors. F77 ignores any END= specification and takes the ERR= exit for both.

7 In performing formatted I/O under A format, FORTRAN IV clears the high-order bit of each transferred character and discards null characters (bytes or zeros); FORTRAN IV on RSX and FORTRAN-77 do neither.

E THE FORTRAN IV SYSTEM SIMULATOR ($SIMRT)

E.1 $SIMRT CAPABILITIES AND RELATIONSHIP TO RT-11

$SIMRT supports the following RT-11 monitor calls:

EMT 240		.WAIT (NOP)
EMT 340		.TTYIN, .TTINR
EMT 341		.TTYOUT, .TTOUR
EMT 342		.DSTATUS (only for TT:)
EMT 346		.LOCK (NOP)
EMT 347		.UNLOCK (NOP)
EMT 350		.EXIT
EMT 351		.PRINT
EMT 353		.QSET (NOP)
EMT 354		.RCTRLO
EMT 374	Subcode 8.	.DATE (returns 0)
EMT 375	Subcode 0,	.WAIT (NOP)
	Subcode 3,	.TRPSET
	Subcode 16,	.GTJB (no channel table adrs)
	Subcode 24,	.SFPA
	Subcode 27,	.CNTXSW (NOP)

Any others create an error condition and a printed message.

E.2 $SIMRT TERMINAL HANDLING

$SIMRT terminal I/O is fully interrupt-driven and ring-buffered.

The following features are supported:

- CTRL/S, CTRL/Q (XON, XOFF)—"SET TT PAGE" mode
- CTRL/O—Cancel terminal output
- CTRL/U—Cancel input line
- Rubout—Delete previous character (echoes backslash)
- CTRL/C—Abort execution
- Conversion of <TAB>s to spaces on output
- Output of altmode (escape) as "$"
- Output of CTRL characters as "^char"

- Support of lowercase input if done with .ASECT, for example:

```
.ASECT
. = 44 ;for JSW
.WORD   040000
```

- Support of RT-11-style fill character and count with .ASECT, for example:

```
.ASECT
. = 56 ;for fill char and count
.BYTE  015,004    ;four nulls after carriage return
```

The TT$SPC mode (single-character mode) is not supported.

E.3 SYSLIB CALLS UNDER $SIMRT

Table E-1 Validity of SYSLIB Calls under $SIMRT

Routine Name	Valid?	Routine Name	Valid?
CHAIN	No	IWRITE,W,C,F	No
CLOSEC	No	JADD	Yes
CONCAT	Yes	JCMP	Yes
CVTTIM	Maybe[1]	JDIV	Yes
DEVICE	No	JAFIX	Yes
GETSTR	Yes	JDFIX	Yes
GTIM	No	IDJFLT,DJFLT	Yes
GTJB	Yes[2]	IAJFLT,AJFLT	Yes
GTLIN	No	JICVT	Yes
IADDR	Yes	JJCVT	Yes
IASIGN	Yes[2]	JMOV	Yes
ICDFN	No	JMUL	Yes
ICHCPY	No	JSUB	Yes
ICMKT	No	JTIME	Maybe[1]
ICSI	No	LEN	Yes
ICSTAT	No	LOCK	Yes[2]
IDELET	No	LOOKUP	No
IDSTAT	Yes[2]	MRKT	No
IENTER	No	MTATCH	No
IFETCH	No	MTDTCH	No
IFREEC	Yes[2]	MTGET	No
IGETC	Yes[2]	MTIN	No
IGETSP	Yes	MTOUT	No
IJCVT	Yes	MTPRNT	No

[1]Will operate if the following user routine is added in .PSECT USER$I: .GLOBL $GVAL $GVAL: CLR RO (if 60-cycle clock) MOV 0,RO (if 50-cycle clock) RTS PC

[2]Operates correctly, but is not useful in the $SIMRT environment.

Table E-1 (Cont.) Validity of SYSLIB Calls under $SIMRT

Routine Name	Valid?	Routine Name	Valid?
ILUN	Yes[2]	MTRCTO	No
INDEX	Yes	MTSET	No
INSERT	Yes	MWAIT	No
INTSET	No	PRINT	Yes
IPEEK	Yes	PURGE	No
IPEEKB	Yes	PUTSTR	Yes
IPOKE	Yes	R50ASC	Yes
IPOKEB	Yes	RAD50	Yes
IQSET	Yes[2]	RCHAIN	Yes[2]
IRAD50	Yes	RCTRLO	Yes
IRCVD,W,C,F	No	REPEAT	Yes
IREAD,W,C,F	No	RESUME	No
IRENAM	No	SCCA	No
IREOPN	No	SCOMP	Yes
ISAVES	No	SCOPY	Yes
ISCHED	No	SECNDS	No
ISDAT,W,C,F	No	SETCMD	No
ISLEEP	No	STRPAD	Yes
ISPFN,W,C,F	No	SUBSTR	Yes
ISPY	No	SUSPND	No
ITIMER	No	TIMASC	Maybe[1]
ITLOCK	No	TIME	No
ITTINR	Yes	TRANSL	Yes
ITTOUR	Yes	TRIM	Yes
ITWAIT	No	UNLOCK	Yes[2]
IUNTIL	No	VERIFY	Yes
IWAIT	Yes[2]		

[1]Will operate if the following user routine is added in .PSECT USER$I: .GLOBL $GVAL $GVAL: CLR RO (if 60-cycle clock) MOV 0,RO (if 50-cycle clock) RTS PC

[2]Operates correctly, but is not useful in the $SIMRT environment.

E.4 MODIFYING SIMRT

To modify the standalone FORTRAN IV support module, SIMRT, obtain a copy of SIMRT.MAC and FRT.MAC from your binary distribution medium according to the procedures described in the *RT-11 Installation Guide/Release Notes*. If you wish to modify SIMRT.MAC, for example, to support new devices, be certain that MACRO.SAV and SYSMAC.SYL are on your system volume. The command procedures that follow will not work without them. Next, assemble SIMRT.MAC by typing the following command:

```
.MACRO/LIST:SIMRT/OBJECT:UNI FRT+SIMRT
```

This command produces a listing file named SIMRT.LST and an object file named UNI.OBJ.

Next, place the new standalone support module in the FORTRAN IV OTS library using the appropriate command procedure:

If your FORTRAN IV OTS library is in SYSLIB, then proceed as follows:

```
.R LIBR
*SYSLIB[-1]=SYSLIB,UNI/U/G
Global?  $ERRS
Global?  $ERRTB
Global?  $OVRH
Global?  $OVTAB
Global?  <RET>
* CTRL/C
```

If your FORTRAN IV OTS library is in FORLIB, however, then proceed as follows:

```
.R LIBR
*FORLIB[-1]=FORLIB,UNI/U/G
Global?   $ERRS
Global?   $ERRTB
Global?   <RET>
* CTRL/C
```

The new version of standalone FORTRAN IV support is now part of your FORTRAN OTS library.

Note: **You must assume responsibility for changes such as those shown above that you make to SIMRT.MAC or to FRT.MAC. Such changes are not supported by DIGITAL.**

Index

digital

digital equipment corporation

Printed in U.S.A.